KAPITEL 3

Hinzufügen und Verwalten von Inhalt

Die Windows Presentation Foundation (WPF) bietet für das Erstellen und Anzeigen von Grafiken, Bildern und Mediendateien bessere Unterstützung als je zuvor. In diesem Kapitel lernen Sie, wie Sie mithilfe von WPF Grafiken erstellen und anzeigen, die grafische Oberfläche verändern und Audio- und Videoinhalt hinzufügen.

In diesem Kapitel abgedeckte Prüfungsziele:

- Ändern der Benutzeroberfläche während der Laufzeit
- Hinzufügen von Multimediainhalt zu einer Anwendung in WPF
- Erstellen und Anzeigen von Grafik

Lektionen in diesem Kapitel:

Bevor Sie beginnen

Um die Lektionen in diesem Kapitel durcharbeiten zu können, müssen Sie folgende Vorbereitungen getroffen haben:

- Ihnen steht ein Computer zur Verfügung, der die im Abschnitt »Einführung« am Anfang dieses Buchs aufgelisteten Hardwarevoraussetzungen erfüllt.
- Auf Ihrem Computer ist Microsoft Visual Studio 2010 Professional Edition installiert.
- Sie sind mit der Syntax von Visual Basic oder C# sowie dem Microsoft .NET Framework 4.0 vertraut.

Praxistipp

Matthew Stoecker

WPF macht es geradezu simpel, meine Anwendung durch Audio und Video aufzuwerten. Zusammen mit der verbesserten Unterstützung für Grafiken macht dieses Feature es einfacher als je zuvor, anspruchsvolle Anwendungen zu entwickeln.

Lektion 1: Verwalten der grafischen Oberfläche

WPF bietet für das Erstellen und Anzeigen von Grafik bessere Unterstützung als je zuvor. In dieser Lektion erfahren Sie, wie Sie diese verbesserte Unterstützung ausnutzen, indem Sie eigene Grafikeffekte entwickeln. Sie lernen, wie Sie mit unterschiedlichen Pinseln Effekte entwerfen, viele unterschiedliche Formen erstellen und sie transformieren und wie sie feststellen, wann die Maus sich innerhalb solcher Formen befindet.

Am Ende dieser Lektion werden Sie in der Lage sein, die folgenden Aufgaben auszuführen:

- Erstellen von Pinseln, die unterschiedliche Effekte bewirken, und Beschreiben, wodurch sich Pinsel voneinander unterscheiden
- Einen Pinsel auf verschiedene Eigenschaften anwenden, die sich auf die optische Darstellung von Steuerelementen auswirken
- Erstellen unterschiedlicher Linien und Formen
- Anwenden von Transformationen auf unterschiedliche Linien und Formen
- Ermitteln, wann sich die Maus über einer Form befindet

Veranschlagte Zeit für diese Lektion: 30 Minuten

Pinsel

Pinsel (brush) sind in WPF die wichtigsten Objekte, um die Benutzeroberfläche zu zeichnen. Jedes Steuerelement hat Eigenschaften, denen Sie ein `Brush`-Objekt zuweisen können. Sie steuern das Aussehen eines Steuerelements, indem Sie jeder Eigenschaft einen anderen Pinsel zuweisen. Diese Eigenschaften gibt es bei verschiedenen Steuerelementen, aber kein Steuerelement definiert alle Eigenschaften. Tabelle 3.1 beschreibt einige der Eigenschaften.

Tabelle 3.1 Elementeigenschaften, denen ein `Brush`-Object zugewiesen werden kann

Eigenschaft	Beschreibung
`Background`	Der Pinsel, der dieser Eigenschaft zugewiesen wird, zeichnet den Hintergrund des Steuerelements.
`BorderBrush`	Der Pinsel, der dieser Eigenschaft zugewiesen wird, zeichnet den Rand des Steuerelements.
`Fill`	Der Pinsel, der dieser Eigenschaft zugewiesen wird, zeichnet den Innenbereich einer Form.
`Foreground`	Der Pinsel, der dieser Eigenschaft zugewiesen wird, zeichnet den Vordergrund des Steuerelements inklusive Inhalt des Steuerelements, wenn der Inhalt eine Zeichenfolge ist.
`OpacityMask`	Dieser Eigenschaft können Sie ein `Brush`-Objekt zuweisen, aber es wird nur die Deckungskomponente ausgewertet. Sie wird durch den Deckungsgrad bestimmt, der dem Pinsel zugewiesen ist. Wenn Sie dieser Eigenschaft einen transparenten Pinsel zuweisen, erscheint also das gesamte Steuerelement transparent. Sie können für diese Eigenschaft exotische Pinsel verwenden, um ungewöhnliche Transparenzeffekte zu erzielen.
`Stroke`	Der Pinsel, der dieser Eigenschaft zugewiesen wird, zeichnet den Rand einer Form.

Pinsel sind Objekte, die unveränderbar (freezable) gemacht werden können. Sie können daher nur Änderungen an `Brush`-Objekten vornehmen, solange die Methode `Freeze` nicht aufgerufen wurde. Wurde die Methode `Brush.Freeze` einmal ausgeführt, wird der Pinsel schreibgeschützt, sodass keine weiteren Änderungen möglich sind.

SolidColorBrush

`SolidColorBrush` ist die einfachste `Brush`-Klasse. Wie der Name andeutet, zeichnet sie eine einzige, gleichförmige Farbe ohne jegliche Muster oder Farbverläufe. In der Klasse `Brushes` stehen mehrere einfarbige Pinsel zur Verfügung, deren Name die jeweilige Farbe angibt. Auf diese Pinsel greifen Sie folgendermaßen zu:

' Visual Basic

```
Dim aBrush As Brush
aBrush = Brushes.AliceBlue
```

// C#

```
Brush aBrush;
aBrush = Brushes.AliceBlue;
```

In XAML-Code (Extensible Application Markup Language) können Sie einer Eigenschaft einen benannten Pinsel zuweisen, indem Sie einfach seinen Namen angeben:

```
<Button Background="Tomato"></Button>
```

Sie können auch eine hexadezimale Notation verwenden, um Pinsel in XAML-Code zuzuweisen. Dabei geben Sie eine achtstellige Zahl an, die die Farbe festlegt. Das erste Ziffernpaar ist der Wert (von 00 bis FF) der Transparenz, das zweite Ziffernpaar gibt die Stärke des Rotkanals an, das dritte die Stärke des Grünkanals und das letzte die Stärke des Blaukanals. Der Hexadezimalzahl stellen Sie das Zeichen # voran. Im nächsten Beispiel wird dem Hintergrund einer Schaltfläche ein tiefrotes `SolidColorBrush`-Objekt zugewiesen:

```
<Button Background="#FFFF0000"></Button>
```

Sie können auch ein neues `SolidColorBrush`-Objekt anlegen, indem Sie den Wert jedes Kanals direkt angeben:

```
<Button>
  <Button.Background>
    <SolidColorBrush>
      <SolidColorBrush.Color>
        <Color A="255" R="0" G="0" B="255"/>
      </SolidColorBrush.Color>
    </SolidColorBrush>
  </Button.Background>
</Button>
```

Im Programmcode können Sie mit der Methode `Color.FromArgb` die einzelnen Kanäle definieren:

' Visual Basic

```
Dim aBrush As SolidColorBrush
aBrush = New SolidColorBrush(Color.FromArgb(255, 0, 255, 0))
```

// C#

```
SolidColorBrush aBrush;
aBrush = new SolidColorBrush(Color.FromArgb(255, 0, 255, 0));
```

LinearGradientBrush

Mit `LinearGradientBrush` erzeugen Sie einen Pinsel, der aus zwei oder mehr Farben einen Farbverlauf (gradient) zeichnet. Dabei sind zahlreiche Effekte möglich. Abbildung 3.1 zeigt, wie es aussieht, wenn ein `LinearGradientBrush`-Objekt den Hintergrund eines Fensters zeichnet.

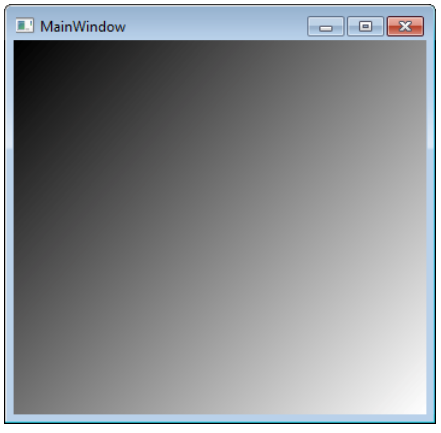

Abbildung 3.1 Der Fensterhintergrund wird mit einem `LinearGradientBrush` gezeichnet

Dieser Effekt wird mit dem folgenden Code erzeugt:

```
<Grid>
  <Grid.Background>
    <LinearGradientBrush>
      <GradientStop Color="Black" Offset="0"/>
      <GradientStop Color="White" Offset="1"/>
    </LinearGradientBrush>
  </Grid.Background>
</Grid>
```

Das `LinearGradientBrush`-Objekt legt über ein Koordinatensystem fest, wie sich der Farbverlauf zusammensetzt. Dieses Koordinatensystem basiert auf einem Rechteck, das den überzeichneten Bereich umschließt. Die obere linke Ecke des Rechtecks hat die Koordinate (0,0), die rechte untere Ecke die Koordinate (1,1). Die Koordinaten werden also relativ zur Größe des Zeichenbereichs angegeben, nicht in Pixeleinheiten.

Jedes `LinearGradientBrush`-Objekt enthält eine Auflistung mit `GradientStop`-Objekten, die jeweils zwei wichtige Eigenschaften haben: `Color` und `Offset`. Die Eigenschaft `Color` legt fest, welche Farbe eingemischt wird, und `Offset` ist eine Zahl, die den Punkt im Koordinatensystem angibt, an dem die angegebene Farbe rein ist und nicht mit anderen Farben gemischt wird. In Abbildung 3.1 ist die obere linke Ecke völlig schwarz und die rechte untere Ecke völlig weiß. Die beiden Farben werden entlang des Farbverlaufs gemischt.

Der Farbverlauf, über den die Farben zusammengemischt werden, liegt auf einer Linie, die vom Start- zum Endpunkt verläuft. In der Standardeinstellung liegt der Startpunkt bei (0,0) und der Endpunkt bei (1,1). Somit entsteht ein diagonaler Farbverlauf, der von der linken oberen zur rechten unteren Ecke läuft. Mit den Eigenschaften `LinearGradientBrush.StartPoint`

und `LinearGradientBrush.EndPoint` können Sie andere Start- und Endpunkte für `LinearGradientBrush` festlegen. Der folgende Code erstellt zum Beispiel ein `LinearGradientPoint`-Objekt, das keinen diagonalen, sondern einen horizontalen Farbverlauf zeichnet (Abbildung 3.2 zeigt das Ergebnis):

```
<Grid>
  <Grid.Background>
    <LinearGradientBrush StartPoint="0,1" EndPoint="1,1">
      <GradientStop Color="Black" Offset="0"/>
      <GradientStop Color="White" Offset="1"/>
    </LinearGradientBrush>
  </Grid.Background>
</Grid>
```

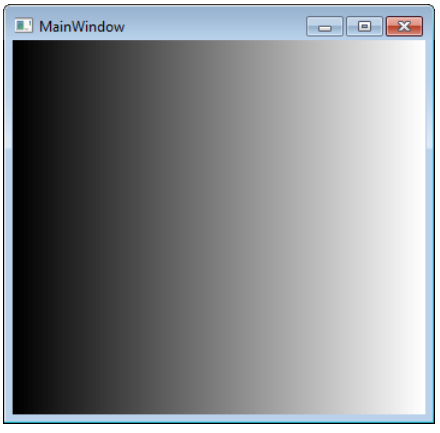

Abbildung 3.2 Ein horizontales `LinearGradientBrush`-Objekt

Sie können mehr als zwei `GradientStop`-Objekte in einem `LinearGradientBrush`-Objekt verwenden. Das nächste Beispiel demonstriert ein `LinearGradientBrush`-Objekt, das mehrere Farben über den Farbverlauf hinweg mischt. Abbildung 3.3 zeigt das Ergebnis dieses Codes:

```
<Grid>
  <Grid.Background>
    <LinearGradientBrush StartPoint="0,0" EndPoint="1,1">
      <GradientStop Color="Black" Offset="0"/>
      <GradientStop Color="Red" Offset=".25"/>
      <GradientStop Color="Blue" Offset=".5"/>
      <GradientStop Color="Green" Offset=".75"/>
      <GradientStop Color="White" Offset="1"/>
    </LinearGradientBrush>
  </Grid.Background>
</Grid>
```

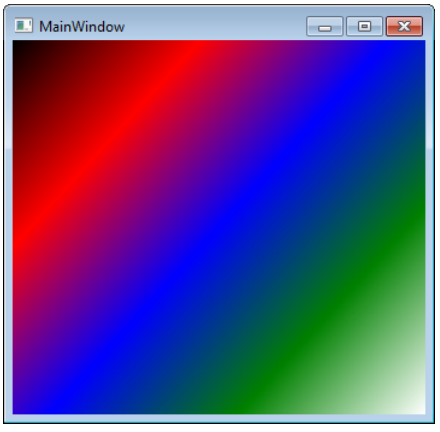

Abbildung 3.3 Ein mehrfarbiges `LinearGradientBrush`-Objekt

Die Linie, die den Farbverlauf definiert, muss nicht in einer Ecke des Koordinatensystems beginnen. Sie können in einem `LinearGradientBrush`-Objekt beispielsweise `StartPoint` die Koordinate (.3,.3) und `EndPoint` die Koordinate (.7,.7) zuweisen, oder irgendein anderes Koordinatenpaar mit Werten zwischen 0 und 1. Wenn die Linie, die den Farbverlauf definiert, nicht von einem Ende des Koordinatensystems zum anderen reicht, hängt es vom Wert der Eigenschaft `LinearGradientBrush.SpreadMethod` ab, wie der übrige Bereich gezeichnet wird. Tabelle 3.2 beschreibt die möglichen Werte und Auswirkungen dieser Eigenschaft.

Tabelle 3.2 Werte der Eigenschaft `LinearGradientBrush.SpreadMethod`

Wert	Beschreibung
`Pad`	Dies ist der Standardwert für die Eigenschaft `SpreadMethod`. Sie zeichnet den Bereich außerhalb des Farbverlaufs einfarbig.
`Reflect`	Wenn `SpreadMethod` den Wert `Reflect` hat, wird der Farbverlauf in der umgekehrten Richtung fortgesetzt, also wie ein Spiegelbild.
`Repeat`	Hat `SpreadMethod` den Wert `Repeat`, wird der Farbverlauf in derselben Richtung wiederholt.

RadialGradientBrush

Das `RadialGradientBrush`-Objekt ähnelt `LinearGradientBrush`. Es mischt eine Reihe von Farben entlang einem Farbverlauf und enthält eine Auflistung mit `GradientStop`-Objekten, die festlegen, wie der Farbverlauf zusammengestellt wird. Der Hauptunterschied ist, dass der Farbverlauf aus einem Punkt des Koordinatensystems in konzentrischen Kreisen nach außen verläuft. Der Mittelpunkt des Farbverlaufs wird durch die Eigenschaft `RadialGradientBrush.GradientOrigin` definiert (Standardeinstellung ist .5,.5). Die äußersten Kreise des Farbverlaufs werden durch die Eigenschaften `RadiusX` und `RadiusY` definiert. `RadiusX` gibt die Entfernung des äußersten Kreises auf der horizontalen Achse an, und `RadiusY` entsprechend den Abstand auf der vertikalen Achse. Die `GradientStop`-Objekte funktionieren genauso wie bei `LinearGradientBrush`, das heißt, sie legen die Punkte fest, an denen die Farben ineinander übergehen. Hier ein Codebeispiel, das ein `RadialGradientBrush`-Objekt definiert:

```
<Grid>
  <Grid.Background>
    <RadialGradientBrush Center=".5, .5" RadiusX=".5" RadiusY=".25">
      <GradientStop Color="Black" Offset="0"/>
      <GradientStop Color="White" Offset="1"/>
    </RadialGradientBrush>
  </Grid.Background>
</Grid>
```

Abbildung 3.4 zeigt, wie das Ergebnis aussieht.

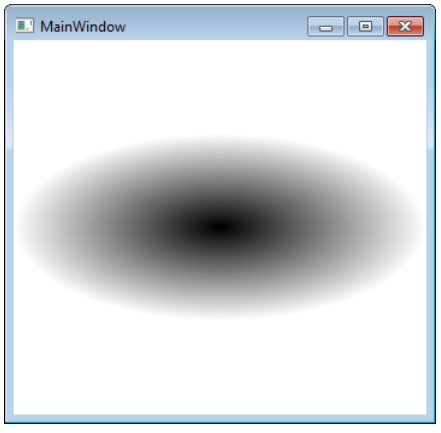

Abbildung 3.4 Ein `RadialGradientBrush`-Objekt

Schnelltest

- Wie wird der Farbverlauf von `RadialGradientBrush` oder `LinearGradientBrush` festgelegt?

Antwort zum Schnelltest

- Der Farbverlauf in diesen Pinseln wird durch eine Auflistung mit `GradientStop`-Objekte gesteuert. Jedes `GradientStop`-Objekt hat die Eigenschaften `Color` und `Offset`. Diese Eigenschaften geben an, welche Farbe eingemischt wird und an welchem Punkt im Farbverlauf die Farbe ihre höchste Sättigung erreicht.

ImageBrush

Mit dem `ImageBrush`-Objekt können Sie Objekte zeichnen, indem Sie ein Bild als Quelle für den Pinsel verwenden. Das Quellbild für `ImageBrush` tragen Sie in die Eigenschaft `ImageSource` ein:

```
<Grid.Background>
  <ImageBrush
    ImageSource="C:\Users\Public\Pictures\Sample Pictures\Forest.jpg">
  </ImageBrush>
</Grid.Background>
```

`ImageBrush` verwendet das angegebene Bild, um die sichtbaren Objekte zu zeichnen, denen der Pinsel zugewiesen ist. Ist das verwendete Bild kleiner als das sichtbare Objekt, dem der Pinsel zugewiesen ist, legt die Eigenschaft `Stretch` fest, wie der Pinsel das Objekt zeichnet. Tabelle 3.3 zeigt die möglichen Werte der Eigenschaft `Stretch`.

Tabelle 3.3 Werte der Eigenschaft `Stretch`

Wert	Beschreibung
`Fill`	Das Bild wird in beiden Richtungen so gedehnt, dass es den verfügbaren Platz ausfüllt. Das Seitenverhältnis des Bilds wird nicht bewahrt.
`None`	Das Bild wird in der Originalgröße gezeichnet. Alle Bereiche außerhalb des Originalbilds bleiben leer.
`Uniform`	Das Bild wird gedehnt, damit es den verfügbaren Platz ausfüllt, aber das Seitenverhältnis des Bilds bleibt erhalten. Alle Bereiche außerhalb des vergrößerten Bilds bleiben leer.
`UniformToFill`	Das Bild wird so vergrößert, dass es den verfügbaren Platz ausfüllt. Das Seitenverhältnis bleibt erhalten, aber der Inhalt wird bei Bedarf beschnitten, wenn der Container ein anderes Seitenverhältnis hat als das Bild.

Sie können einen Ausschnitt des Quellbilds auswählen, der zum Zeichnen verwendet wird. Dazu tragen Sie in die Eigenschaft `Viewbox` ein Rechteck ein (normalerweise relativ zum gedachten Rahmen, der das Bild einfasst). Dieser Bereich wird dann zum Zeichnen ausgeschnitten. Das folgende Beispiel demonstriert, wie Sie den oberen linken Quadranten eines Bilds ausschneiden und für das `ImageBrush`-Objekt verwenden:

```
<Grid.Background>
  <ImageBrush Viewbox="0,0,.5,.5"
    ImageSource="C:\Users\Public\Pictures\Sample Pictures\Forest.jpg">
  </ImageBrush>
</Grid.Background>
```

Wenn die Eigenschaft `Stretch` den Wert `None` hat, können Sie das Verhalten des `ImageBrush`-Pinsels zusätzlich mit der Eigenschaft `TileMode` ändern. Tabelle 3.4 listet die verfügbaren Werte für die Eigenschaft `TileMode` auf.

Mit der Eigenschaft `Viewport` legen Sie fest, welcher Ausschnitt des Originalbilds als Kachelquelle benutzt wird. Hier ein Beispiel:

```
<Grid.Background>
  <ImageBrush TileMode="FlipXY" Viewport="0,0,.5,.5"
    ImageSource="C:\Users\Public\Pictures\Sample Pictures\Forest.jpg">
  </ImageBrush>
</Grid.Background>
```

Tabelle 3.4 Werte für die Eigenschaft `TileMode`

Wert	Beschreibung
`FlipX`	Das Bild wird in vertikaler Richtung in der ursprünglichen Anordnung gekachelt. In horizontaler Richtung wechseln sich ursprüngliche und gespiegelte Anordnung ab.
`FlipXY`	Das Bild wird in horizontaler und vertikaler Richtung gekachelt, wobei sich in beiden Richtungen ursprüngliche und gespiegelte Anordnung abwechseln. Beachten Sie, dass Kacheln, die rechts unter einer Kachel in ursprünglicher Ausrichtung liegen, aussehen, als wären sie relativ zur ursprünglichen Ausrichtung um 180 Grad gedreht.
`FlipY`	Das Bild wird in horizontaler Richtung in der ursprünglichen Anordnung gekachelt. In vertikaler Richtung wechseln sich ursprüngliche und gespiegelte Anordnung ab.
`None`	Das Bild wird nicht in Form von Kacheln gezeichnet.
`Tile`	Das Bild wird in der ursprünglichen Ausrichtung gekachelt.

VisualBrush

`VisualBrush` ähnelt `ImageBrush`, aber statt ein Bild als Quelle zum Zeichnen zu verwenden, benutzt `VisualBrush` ein `Visual`-Element, zum Beispiel ein WPF-Steuerelement oder irgendein anderes Element, das von der Klasse `Visual` abgeleitet ist. Das folgende Beispiel zeigt ein `VisualBrush`-Objekt, das ein `Button`-Steuerelement als Quelle zum Zeichnen hernimmt:

```
<VisualBrush Visual="{Binding ElementName=Button1}">
</VisualBrush>
```

Sie können `VisualBrush` genauso wie `ImageBrush` verwenden und die Eigenschaften `Stretch` oder `TileMode` genauso festlegen.

Formen

Formen (shape) sind Grundelemente beim Zeichnen. Sie stehen für individuell gezeichnete Elemente wie Rechtecke, Ellipsen, Polygone und Linien. Formen werden zwar als einfache gezeichnete Elemente dargestellt, sie unterstützen aber dieselben Ereignisse der Benutzerinteraktion wie andere WPF-Elemente. Daher können Sie damit ganz einfach neue Steuerelemente erstellen.

Alle Formen in WPF sind von der abstrakten Klasse `Shape` abgeleitet, die über mehrere Stufen selbst wiederum von der Klasse `Visual` abgeleitet ist. Diese Klasse stellt mehrere Eigenschaften für alle WPF-Formen zur Verfügung. Tabelle 3.5 beschreibt einige dieser Eigenschaften.

Tabelle 3.5 Eigenschaften der Klasse `Shape`

Eigenschaft	Beschreibung
`Fill`	Das `Brush`-Objekt, das den Innenbereich der Form zeichnet.
`Stroke`	Das `Brush`-Objekt, das die Ränder der Form zeichnet.
`StrokeThickness`	Legt die Dicke des Rands in geräteunabhängigen Einheiten fest.
`Stretch`	Legt fest, wie eine Form den Platz füllt, den sie belegt.

Wie andere Steuerelemente hat die Klasse `Shape` die Eigenschaften `Width`, `Height`, `Margin` und andere Eigenschaften, die die Größe und den Abstand des sichtbaren Elements steuern. Sind `Width` und `Height` keine expliziten Werte zugewiesen, legt die Eigenschaft `Stretch` fest, wie

eine Form gedehnt wird, um den verfügbaren Platz auszufüllen. Tabelle 3.6 beschreibt die möglichen Werte für die Eigenschaft `Stretch`.

Tabelle 3.6 Werte für die Eigenschaft `Stretch`

Wert	Beschreibung
`None`	Die Form wird in voller Größe gezeichnet.
`Uniform`	Die Form wird so vergrößert, dass sie den verfügbaren Platz ausfüllt, aber das Seitenverhältnis der Form bleibt erhalten.
`Fill`	Die Form wird in beiden Richtungen so vergrößert, dass sie den verfügbaren Platz ausfüllt. Das Seitenverhältnis der Form bleibt dabei nicht erhalten.
`UniformToFill`	Die Form wird so vergrößert, dass sie den verfügbaren Platz ausfüllt. Das Seitenverhältnis der Form bleibt erhalten, sie kann aber abgeschnitten werden.

Die Klassen *Rectangle* und *Ellipse*

Die Klasse `Rectangle` steht für die Darstellung eines Rechtecks und die Klasse `Ellipse` für eine Ellipse. Beide Formen sind sehr simpel. Die optische Darstellung wird in erster Linie durch die Eigenschaften `Height` und `Width` definiert. Die folgenden Beispiele zeigen ein Rechteck und eine Ellipse, die jeweils 100 Einheiten hoch und 200 Einheiten breit sind:

```
<Rectangle Height="100" Width="200" Fill="Blue"/>
<Ellipse Height="100" Width="200" Fill="Blue"/>
```

Ist den Eigenschaften `Height` und `Width` kein expliziter Wert zugewiesen, können Sie mit der Eigenschaft `Stretch` festlegen, wie die jeweilige Form den verfügbaren Platz ausfüllt.

Ein Rechteck mit abgerundeten Ecken erstellen Sie, indem Sie den Eigenschaften `RadiusX` und `RadiusY` geeignete Werte zuweisen. Diese Eigenschaften geben den X- und Y- Radius einer Ellipse an, die benutzt wird, um die Ecken des Rechtecks abzurunden. Der folgende XAML-Code zeichnet ein Rechteck mit abgerundeten Ecken:

```
<Rectangle RadiusX="20" RadiusY="10" Fill="blue" Height="50" Width="100"/>
```

Die Klassen *Line* und *Polyline*

Die Klasse `Line` steht für eine Linie, genauer gesagt eine Gerade, die zwei Koordinatenpunkte verbindet. Das zum Zeichnen der `Line`-Objekte verwendete Koordinatensystem hat seinen Ursprung in der oberen linken Ecke des Objekts, in dem die Gerade liegt. Ist die Gerade beispielsweise in eine `Grid`-Zelle eingebettet, liegt der Punkt (0,0) in der oberen linken Ecke dieser Zelle. Sie erstellen ein `Line`-Objekt, indem Sie Werte in die Eigenschaft `X1`, `Y1`, `X2` und `Y2` eintragen:

```
<Line Stroke="Red" X1="0" Y1="50" X2="100" Y2="440"/>
```

Die Klasse `Polyline` bildet eine etwas komplexere Form. Sie ist im Wesentlichen eine Abfolge miteinander verbundener Punkte. Die Klasse `Polyline` enthält die Auflistung `Points`, in der die Punkte der Form definiert sind. Die Form beginnt beim ersten Punkt der Auflistung und verbindet dann die weiteren Punkte durch gerade Linien, bis sie beim letzten Punkt endet. Das folgende Beispiel zeigt ein `Polyline`-Objekt, das eine Sägezahnform bildet:

```
<Polyline Stroke="Green"
  Points="300, 300 400, 400 400, 300 500, 400 500, 300"/>
```

Die Kommas zwischen den X- und Y-Koordinaten der einzelnen Punkte sind nicht nötig aber erlaubt und machen den Code besser lesbar.

Die Klasse *Polygon*

Die Klasse `Polygon` ähnelt `Polyline`. Wie die Klasse `Polyline` hat sie die Auflistung `Points`, in der die Punkte der Form definiert sind. Diese Punkte werden miteinander verbunden. Der wichtigste Unterschied besteht darin, dass die Klasse `Polygon` zusätzlich den ersten mit dem letzten Punkt verbindet und den Innenbereich der Form mit dem `Brush`-Objekt ausfüllt, das in der Eigenschaft `Fill` definiert ist. Das folgende Beispiel zeigt ein `Polygon`-Objekt, das dem `Polyline`-Objekt aus dem letzten Beispiel ähnelt:

```
<Polygon Fill="Green"
  Points="300, 300 400, 400 400, 300 500, 400 500, 300"/>
```

Wenn sich Linien in einem `Polyline`- oder `Polygon`-Objekt schneiden, erzeugen sie geschlossene Bereiche, die mit dem `Fill`-Pinsel der Form ausgemalt werden können. Die Eigenschaft `FillRule` steuert, wie umschlossene Bereiche gefüllt werden. Tabelle 3.7 listet die möglichen Werte für die Eigenschaft `FillRule` auf.

Tabelle 3.7 Werte für die Eigenschaft `FillRule`

Wert	Beschreibung
`EvenOdd`	WPF zählt, wie viele Linien geschnitten werden müssen, um zum umschlossenen Bereich zu gelangen. Kann ein umschlossener Bereich erreicht werden, indem eine ungerade Zahl von Linien geschnitten werden, wird er gefüllt. Kann er dagegen erreicht werden, indem eine gerade Zahl von Linien geschnitten werden, wird er nicht gefüllt.
`Nonzero`	WPF zählt, wie viele Linien geschnitten werden müssen, um zum umschlossenen Bereich zu gelangen, und wertet aus, in welcher Richtung sie gezeichnet wurden. Ist die Zahl der geschnittenen Linien, die in der einen Richtung gezeichnet wurden, genauso groß wie die Zahl der geschnittenen Linien, die in der anderen Richtung verlaufen, wird der Bereich nicht gefüllt. Andernfalls wird der Bereich gefüllt.

Path

`Path` ist die komplexeste Form in WPF. Ein `Path`-Objekt beschreibt eine komplexe Form, die aus mehreren `Geometry`-Objekten bestehen kann. Ein `Geometry`-Objekt können Sie sich als Plan für eine Form vorstellen. Es enthält alle Daten über das Aussehen einer Form, darunter Koordinaten und Größe. Aber das `Geometry`-Objekt implementiert keine der Ereignisbehandlungs- oder Steuerelementfunktionen, die ein `Shape`-Objekt zur Verfügung stellt.

Tabelle 3.8 beschreibt, welche Typen von `Geometry`-Objekten Sie verwenden können, um ein `Path`-Objekt zu erstellen.

Am einfachsten erstellen Sie ein `Path`-Objekt, indem Sie ein einzelnes `Geometry`-Objekt in die Eigenschaft `Path.Data` eintragen:

```
<Path Fill="Aqua">
  <Path.Data>
    <EllipseGeometry RadiusX="40" RadiusY="50"/>
  </Path.Data>
</Path>
```

Tabelle 3.8 `Geometry`-Objekte, aus denen ein `Path`-Object aufgebaut werden kann

Klasse	Beschreibung
`CombinedGeometry`	Fasst zwei `Geometry`-Objekte zu einem einzigen Objekt zusammen, sodass Sie unterschiedliche Effekte anwenden können, indem Sie der Eigenschaft `GeometryCombineMode` einen Wert zuweisen.
`EllipseGeometry`	Enthält Daten, die die Form einer Ellipse festlegen.
`GeometryGroup`	Steht für eine Gruppe von `Geometry`-Objekten, die zum `Path`-Objekt hinzugefügt werden.
`LineGeometry`	Steht für eine Gerade.
`PathGeometry`	Steht für eine komplexe Form oder Figur, die aus Linien, Bögen und Kurven zusammengesetzt ist.
`RectangleGeometry`	Steht für ein Rechteck.
`StreamGeometry`	Ein simpleres Gegenstück zu `PathGeometry`. `StreamGeometry` unterscheidet sich von `PathGeometry` dadurch, dass es schreibgeschützt ist, nachdem es erstellt wurde.

Mithilfe der Klasse `GeometryGroup` können Sie Pfade zusammenstellen, die mehrere Formen umfassen. Das folgende Beispiel kombiniert ein Rechteck und eine Ellipse:

```
<Path Fill="Aqua" Margin="100">
  <Path.Data>
    <GeometryGroup FillRule="Nonzero">
      <EllipseGeometry RadiusX="40" RadiusY="50"/>
      <RectangleGeometry Rect="0,0,10,100"/>
    </GeometryGroup>
  </Path.Data>
</Path>
```

Wenn ein `GeometryGroup`-Objekt überlappende `Geometry`-Objekte enthält, legt die Eigenschaft `FillRule` fest, wie die überlappenden Bereiche gefüllt werden. Das funktioniert genauso wie bei `Polygon` und `Polyline` (siehe Tabelle 3.7 weiter oben in dieser Lektion).

Mit der Klasse `CombinedGeometry` können Sie `Geometry`-Objekte erstellen, die zwei `Geometry`-Objekte zusammenfassen. Die Art der Kombination wird von der Eigenschaft `GeometryCombineMode` festgelegt. Tabelle 3.9 beschreibt die erlaubten Werte für die Eigenschaft `GeometryCombineMode`.

Tabelle 3.9 Werte der Eigenschaft `GeometryCombineMode`

Wert	Beschreibung
`Exclude`	Das kombinierte `Geometry`-Objekt deckt den Bereich ab, der entsteht, wenn das zweite `Geometry`-Objekt vom ersten abgezogen wird.
`Intersect`	Das kombinierte `Geometry`-Objekt bildet die Schnittmenge der beiden `Geometry`-Objekte.
`Xor`	Das kombinierte `Geometry`-Objekt überdeckt den Bereich, der nur in jeweils einem der beiden `Geometry`-Objekte enthalten ist. Dies ist das Gegenteil des Werts `Intersect`.
`Union`	Das kombinierte `Geometry`-Objekt bildet die Vereinigungsmenge der beiden `Geometry`-Objekte.

Das folgende Beispiel demonstriert ein `Path`-Objekt, das mithilfe eines `CombinedGeometry`-Objekts erstellt wird:

```
<Path Fill="Aqua" Margin="100">
  <Path.Data>
    <CombinedGeometry GeometryCombineMode="Exclude">
      <CombinedGeometry.Geometry1>
        <EllipseGeometry RadiusX="40" RadiusY="50"/>
      </CombinedGeometry.Geometry1>
      <CombinedGeometry.Geometry2>
        <RectangleGeometry Rect="0,0,10,100"/>
      </CombinedGeometry.Geometry2>
    </CombinedGeometry>
  </Path.Data>
</Path>
```

Sie können sehr komplexe Formen zusammenstellen, wenn Sie die Klasse `PathGeometry` einsetzen. Eine detaillierte Erforschung aller Fähigkeiten der Klasse `PathGeometry` würde den Rahmen dieser Lektion sprengen, aber Sie sollten sich in dieses Thema einarbeiten, wenn Sie komplizierte Grafikformen erzeugen wollen.

Transformationen

Transformationen (engl. transformation oder kurz transform) sind Objekte, mit denen Sie die Form eines Elements ändern, indem Sie das Koordinatensystem verändern, in dem es gezeichnet wird. Sie können Transformationen einsetzen, um ganz unterschiedliche Effekte auf Formen und Elemente anzuwenden, beispielsweise Drehung, Verschiebung, Scherung und kompliziertere Effekte.

Transformationsarten

Tabelle 3.10 zeigt, welche `Transform`-Objekte Sie zur Auswahl haben, um das Aussehen einer Form oder eines Elements zu verändern.

Sie wenden ein `Transform`-Objekt auf eine Form an, indem Sie es der Eigenschaft `RenderTransform` zuweisen. Das folgende Beispiel transformiert ein Rechteck in ein Parallelogramm:

```
<Rectangle Height="20" Width="50" Fill="blue">
  <Rectangle.RenderTransform>
    <SkewTransform AngleX="-30"></SkewTransform>
  </Rectangle.RenderTransform>
</Rectangle>
```

Tabelle 3.10 Unterschiedliche `Transform`-Klassen

Klasse	Beschreibung
`MatrixTransform`	Ändert das Koordinatensystem des Objekts, indem es eine affine 3×3-Transformationsmatrix auf das zugrunde liegende Koordinatensystem anwendet. Die Matrix wird durch die Eigenschaft `Matrix` definiert.
`RotateTransform`	Dreht das Koordinatensystem um den Punkt, der in den Eigenschaften `CenterX` und `CenterY` angegeben ist. Die Eigenschaft `Angle` legt den Winkel der Drehung fest. ▶

Klasse	Beschreibung
`ScaleTransform`	Transformiert das Koordinatensystem, indem es den Maßstab der Transformation vergrößert oder verkleinert. Die Eigenschaft `ScaleX` legt die horizontale Maßstabsänderung fest, die Eigenschaft `ScaleY` die vertikale Maßstabsänderung. Die Eigenschaften `CenterX` und `CenterY` enthalten den Mittelpunkt der Transformation.
`SkewTransform`	Schert das Koordinatensystem Ihres Objekts. Die Eigenschaften `AngleX` und `AngleY` legen den Winkel der Scherung in horizontaler und vertikaler Richtung fest, und die Eigenschaften `CenterX` und `CenterY` den Mittelpunkt der Transformation.
`TranslateTransform`	Verschiebt das Koordinatensystem Ihres Objekts um den horizontalen und vertikalen Betrag, der in den Eigenschaften `X` beziehungsweise `Y` angegeben ist.
`TransformGroup`	Fasst mehrere `Transform`-Objekte zusammen und wendet jedes auf das zugrunde liegende Koordinatensystem an.

Eine Transformation hat ihren Ausgangspunkt normalerweise in der oberen linken Ecke des transformierten Elements. Mithilfe der Eigenschaft `RenderTransformOrigin` können Sie einen anderen Ursprung für die Transformation einstellen. Das Koordinatensystem der Transformationen legt (0,0) als obere linke Ecke des Rechtecks fest, das das Element umfasst, und (1,1) als rechte untere Ecke. Das folgende Beispiel setzt den Ursprung der Transformation in die Mitte des Elements:

```
<Rectangle Height="20" Width="50" Fill="blue"
  RenderTransformOrigin=".5, .5">
  <Rectangle.RenderTransform>
    <SkewTransform AngleX="-30"></SkewTransform>
  </Rectangle.RenderTransform>
</Rectangle>
```

Transformieren von Elementen

Sie können Transformationen nicht nur auf Formen anwenden, sondern damit auch das Aussehen von Elementen verändern. Weil alle WPF-Elemente von der WPF gezeichnet werden, können Sie auf jedes WPF-Element ein `Transform`-Objekt anwenden. Ein `Transform`-Objekt beeinflusst nur das Aussehen eines Steuerelements. Die Funktion des Steuerelements bleibt dabei unverändert. Allerdings ist es möglich, dass massive Transformationen von Benutzeroberflächenelementen den Benutzer verwirren.

Sie wenden ein `Transform`-Objekt genauso auf ein Element an wie auf eine Form: Sie weisen es einfach der Eigenschaft `RenderTransform` zu. Das folgende Beispiel zeigt ein `Button`-Objekt, auf das ein `SkewTransform`-Objekt angewendet wird:

```
<Button Height="20" Width="50">
  <Button.RenderTransform>
    <SkewTransform AngleX="-30"></SkewTransform>
  </Button.RenderTransform>
</Button>
```

Spiegeln

Eine häufig genutzte optische Transformation ist das Spiegeln eines Elements, entweder horizontal an der Y-Achse oder vertikal an der X-Achse. Sie spiegeln ein Element mithilfe von `ScaleTransform`. Wollen Sie ein Element horizontal spiegeln, aber seine Größe und Form

beibehalten, weisen Sie der Eigenschaft ScaleX des ScaleTransform-Objekts den Wert –1 zu. Bei einer vertikalen Spiegelung weisen Sie der Eigenschaft ScaleY des ScaleTransform-Objekts den Wert –1 zu. Wenn Sie sowohl ScaleX als auch ScaleY auf –1 setzen, wird das Element an beiden Achsen gespiegelt. Das folgende Beispiel zeigt, wie eine Schaltfläche horizontal gespiegelt wird:

```
<Button Height="50" Width="150" VerticalAlignment="Bottom">
  <Button.RenderTransform>
    <ScaleTransform ScaleX="-1"></ScaleTransform>
  </Button.RenderTransform>Gespiegelte Schaltfläche
</Button>
```

Bei dieser Transformation wird Ihr Element an der gewählten Achse gespiegelt. Wollen Sie ein Element spiegeln, aber seine ursprüngliche Position beibehalten, können Sie der Eigenschaft RenderTransformOrigin des Elements den Wert .5,.5 zuweisen:

```
<Button RenderTransformOrigin=".5, .5" Height="50" Width="150"
  VerticalAlignment="Bottom">
  <Button.RenderTransform>
    <ScaleTransform ScaleX="-1"></ScaleTransform>
  </Button.RenderTransform>Gespiegelte Schaltfläche
</Button>
```

Prüfungstipp

Sie müssen die Unterschiede zwischen den geometrischen Transformationen kennen und wissen, wann Sie welche einsetzen. Besonders wichtig ist, dass Sie ein Element mit ScaleTransform spiegeln, nicht mit RotateTransform; dies wird häufig verwechselt.

Clipping

Im Abschnitt »Path« weiter oben in dieser Lektion haben Sie gesehen, wie Sie mit Geometry-Objekten komplexe Formen erzeugen. Sie können Geometry-Objekte auch einsetzen, um die Form von Elementen abzuschneiden (clip). Dazu verwenden Sie die Eigenschaft Clip, die von allen WPF-Elementen zur Verfügung gestellt wird. Wenn Sie der Eigenschaft Clip ein Geometry-Objekt zuweisen, schränken Sie die optische Darstellung des Steuerelements auf die Form ein, die vom Geometry-Objekt beschrieben wird. Dabei wird das Element nicht transformiert. Es werden lediglich die Teile außerhalb der Form abgeschnitten, die das Geometry-Objekt beschreibt. Abbildung 3.5 zeigt zwei Button-Elemente, die gleich groß sind. Bei der linken Schaltfläche wird die Eigenschaft Clip nicht verwendet, bei der rechten wurde dagegen der Eigenschaft Clip ein EllipseGeometry-Objekt zugewiesen.

Das folgende Beispiel zeigt, wie Sie Clipping auf ein Element anwenden:

```
<Button Height="100" Width="100" Name="Button1">
  <Button.Clip>
    <!--Da die Schaltfläche 100 x 100 Pixel umfasst, wird ein EllipseGeometry-Objekt
      mit dem Mittelpunkt 50,50 sowie RadiusX und RadiusY von 50 verwendet.
      Die Schaltfläche wird damit rund gezeichnet. -->
    <EllipseGeometry Center="50,50" RadiusX="50" RadiusY="50"/>
  </Button.Clip>Button
</Button>
```

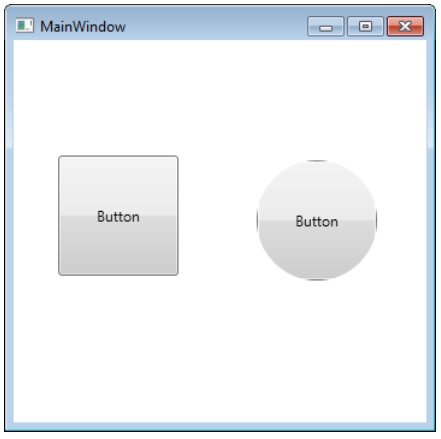

Abbildung 3.5 Zwei `Button`-Elemente, eines abgeschnitten, das andere nicht

Die Oberflächenhierarchie

Bei der WPF-Programmierung wird die Bedienoberfläche durch die Oberflächenhierarchie (visual tree) definiert, eine hierarchische Struktur der sichtbaren Elemente in Ihrer Anwendung. Alle WPF-Elemente, die automatisch dargestellt werden können, sind von der Klasse `Visual` abgeleitet. Dazu gehören Steuerelemente und Formen sowie spezialisierte Klassen, die das Aussehen der Steuerelemente verbessern. Nicht in diese Kategorie fallen dagegen Elemente, die keine Fähigkeit zur automatischen Darstellung besitzen. Ein solches Element, das nicht automatisch dargestellt werden kann, ist beispielsweise das Steuerelement `ListBoxItem`, das Inhalt speichert, aber selbst kein bestimmtes Aussehen hat.

Die Klasse `Visual` stellt die Grundlagen für die Darstellung sichtbarer Elemente auf der Benutzeroberfläche zur Verfügung. Sie implementiert Funktionen für Transformation, Clipping, Trefferprüfung und Berechnung des Begrenzungsrahmens sowie für das Speichern (Persistenz) der serialisierten Zeicheninformationen der Elemente. Aufgrund der Persistenz der Zeichendaten ist WPF in der Lage, die Grafikanzeige mithilfe des Speichermodus zu optimieren.

Das *Grafiksystem mit Speichermodus* (retained mode graphics) ist einer der wichtigsten Unterschiede zwischen der Darstellung der Benutzeroberflächen in WPF-Anwendungen und Windows Forms-Anwendungen. In Windows Forms ist es die Aufgabe der Anwendung, Regionen der Benutzeroberfläche für ungültig zu erklären und Befehle an das System zu senden, damit es diese Regionen neu zeichnet. Dies wird als *Grafiksystem mit unmittelbarem Modus* (immediate mode graphics) bezeichnet. Wenn die Anwendung stark ausgelastet ist, verzögert sich unter Umständen das Neuzeichnen oder die Benutzeroberfläche reagiert nicht mehr.

Dagegen überträgt in WPF das Grafiksystem mit Speichermodus die serialisierten Zeichendaten an das System, sobald ein sichtbares Element erstellt wird. Anschließend übernimmt das System die Aufgabe, das Element neu zu zeichnen. Selbst wenn die Anwendung überlastet ist, kann das System daher auf Zeichenanforderungen reagieren und eine funktionierende Benutzeroberfläche präsentieren.

VisualTreeHelper

Damit Sie die Hierarchie der `Visual`-Elemente in einer WPF-Anwendung durchlaufen können, stellt WPF eine statische Hilfsklasse namens `VisualTreeHelper` zur Verfügung. Sie hat statische Methoden, mit denen Sie die Elemente in der Oberflächenhierarchie durchsuchen und darauf zugreifen können. Tabelle 3.11 beschreibt die Methoden von `VisualTreeHelper`.

Tabelle 3.11 Methoden der Klasse `VisualTreeHelper`

Methode	Beschreibung
`GetBitmapEffect`	Gibt den `BitmapEffect`-Wert für das angegebene `Visual`-Objekt zurück.
`GetBitmapEffectInput`	Gibt den `BitmapEffectInput`-Wert für das angegebene `Visual`-Objekt zurück.
`GetCacheMode`	Ruft die zwischengespeicherte Darstellung des angegebenen `Visual`-Objekts ab.
`GetChild`	Gibt das untergeordnete sichtbare Objekt aus dem angegebenen Auflistungsindex innerhalb des angegebenen übergeordneten Steuerelements zurück.
`GetChildrenCount`	Gibt an, wie viele untergeordnete Objekte das angegebene `Visual`-Objekt enthält.
`GetClip`	Gibt die Clipping-Region des angegebenen `Visual`-Objekts als `Geometry`-Wert zurück.
`GetContentBounds(Visual)`	Gibt das zwischengespeicherte Begrenzungsrechteck für das angegebene `Visual`-Objekt zurück.
`GetContentBounds(Visual3D)`	Gibt das zwischengespeicherte Begrenzungsrechteck für das angegebene `Visual3D`-Objekt zurück.
`GetDescendantBounds(Visual)`	Gibt die Vereinigungsmenge aller Inhalts-Begrenzungsrahmen des `Visual`-Objekts selbst sowie aller untergeordneten Objekte zurück.
`GetDescendantBounds(Visual3D)`	Gibt die Vereinigungsmenge aller Inhalts-Begrenzungsrahmen des `Visual3D`-Objekts selbst sowie aller untergeordneten Objekte zurück.
`GetDrawing`	Gibt den Zeicheninhalt des angegebenen `Visual`-Objekts zurück.
`GetEdgeMode`	Gibt den Randdarstellungsmodus des angegebenen `Visual`-Objekts als `EdgeMode`-Wert zurück.
`GetEffect`	Gibt den Bitmapeffekt für das angegebene `Visual`-Objekt zurück.
`GetOffset`	Gibt den Offset des `Visual`-Objekts zurück.
`GetOpacity`	Gibt die Transparenz des `Visual`-Objekts zurück.
`GetOpacityMask`	Gibt einen `Brush`-Wert zurück, der die Transparenzmaske des `Visual`-Objekts widerspiegelt.
`GetParent`	Gibt einen `DependencyObject`-Wert zurück, der das übergeordnete Objekt des `Visual`-Objekts angibt.
`GetTransform`	Gibt einen `Transform`-Wert für das `Visual`-Objekt zurück.
`GetXSnappingGuidelines`	Gibt eine Auflistung der vertikalen Ausrichtungslinien (X-Koordinate) zurück.
`GetYSnappingGuidelines`	Gibt eine Auflistung der horizontalen Ausrichtungslinien (Y-Koordinate) zurück.
`HitTest(Visual, Point)`	Gibt das oberste `Visual`-Objekt einer Trefferprüfung zurück, die durch einen `Point` angegeben wird. ▶

Methode	Beschreibung
`HitTest(Visual, HitTestFilterCallback, HitTestResultCallback, HitTestParameters)`	Führt eine Trefferprüfung für das angegebene `Visual`-Objekt durch, wobei vom Aufrufer definierte `HitTestFilterCallback`- und `HitTestResultCallback`-Methoden verwendet werden.
`HitTest(Visual3D, HitTestFilterCallback, HitTestResultCallback, HitTestParameters3D)`	Führt eine Trefferprüfung für das angegebene `Visual3D`-Objekt durch, wobei vom Aufrufer definierte `HitTestFilterCallback`- und `HitTestResultCallback`-Methoden verwendet werden.

Das folgende Beispiel demonstriert die Verwendung von `VisualTreeHelper`. Es geht mithilfe von `VisualTreeHelper` in einer Schleife alle Objekte der Oberflächenhierarchie durch:

' Visual Basic

```
Public Sub LoopVisuals(ByVal aVisual As Visual)
   For i As Integer = 0 To VisualTreeHelper.GetChildrenCount(aVisual) - 1
   ' Ruft das untergeordnete sichtbare Objekt mit dem angegebenen Index ab
      Dim bVisual As Visual = CType(VisualTreeHelper.GetChild(aVisual, i), Visual)
      ' Gewünschte Arbeiten mit dem Visual-Objekt ausführen.
      ' Diese Methode rekursiv aufrufen, um wiederum alle untergeordneten Visual-Objekte
      ' dieses untergeordneten Visual-Objekts zu durchlaufen.
      LoopVisuals(bVisual)
   Next i
End Sub
```

// C#

```
public void LoopVisuals(Visual aVisual)
{
   for (int i = 0; i < VisualTreeHelper.GetChildrenCount(aVisual); i++)
   {
      // Ruft das untergeordnete sichtbare Objekt mit dem angegebenen Index ab
      Visual bVisual = (Visual)VisualTreeHelper.GetChild(aVisual, i);
      // Gewünschte Arbeiten mit dem Visual-Objekt ausführen.
      // Diese Methode rekursiv aufrufen, um wiederum alle untergeordneten Visual-Objekte
      //  dieses untergeordneten Visual-Objekts zu durchlaufen.
        LoopVisuals(bVisual);
    }
}
```

Steuerelemente während der Laufzeit in der Bedienoberfläche hinzufügen und entfernen

Sie können sowohl in Windows Forms- als auch WPF-Anwendungen Steuerelemente während der Laufzeit hinzufügen und entfernen. In Windows Forms fügen Sie ein Steuerelement zu einem Formular oder einem anderen Containersteuerelement hinzu, indem Sie auf die Auflistung `Controls` dieses Steuerelements zugreifen. Das folgende Beispiel demonstriert, wie Sie während der Laufzeit eine Schaltfläche zum aktuellen Formular hinzufügen:

' Visual Basic

```
Dim aButton As New Button()
Me.Controls.Add(aButton)
```

// C#

```
Button aButton = new Button();
this.Controls.Add(aButton);
```

Auch wenn Sie ein Steuerelement während der Laufzeit entfernen wollen, benutzen Sie die Auflistung `Controls`. Sie können ein Steuerelement entweder direkt entfernen, über einen Verweis auf das Steuerelement, oder indem Sie seinen Index innerhalb der Auflistung angeben. Das folgende Beispiel demonstriert die verschiedenen Möglichkeiten:

' Visual Basic

```
' Entfernt das Steuerelement, das in der Variablen aButton angegeben ist
Me.Controls.Remove(aButton)
' Entfernt das Steuerelement mit dem Index 0 der Auflistung Controls
Me.Controls.RemoveAt(0)
```

// C#

```
// Entfernt das Steuerelement, das in der Variablen aButton angegeben ist
this.Controls.Remove(aButton);
// Entfernt das Steuerelement mit dem Index 0 der Auflistung Controls
this.Controls.RemoveAt(0);
```

In WPF-Anwendungen ist der Ablauf ähnlich, aber etwas komplizierter, weil Steuerelemente in beliebige Container-, Listen- oder Inhaltssteuerelemente eingebettet sein können. Bei Containersteuerelementen greifen Sie über die Eigenschaft `Children` auf die untergeordneten Steuerelemente zu. Bei Listensteuerelementen wie `ListBox` steht die Eigenschaft `Items` zur Verfügung, um auf Listenelemente zuzugreifen. Sowohl die Eigenschaft `Children` der Containersteuerelemente als auch die Eigenschaft `Items` der Listensteuerelemente stellen die Methoden `Add`, `Remove` und `RemoveAt` zur Verfügung, die ähnlich funktionieren wie ihre Gegenstücke in Windows Forms:

' Visual Basic

```
Dim aButton As New Button()
Dim bButton As New Button()
Grid1.Children.Add(aButton)
ListBox1.Items.Add(bButton)
' Entfernt die Steuerelemente, die in den Variablen aButton und bButton angegeben sind
Grid1.Children.Remove(aButton)
ListBox1.Items.Remove(bButton)
' Entfernt das Steuerelement mit dem Index 0 der Auflistung Controls
Grid1.Children.RemoveAt(0)
ListBox1.Items.RemoveAt(0)
```

// C#

```
Button aButton = new Button();
Button bButton = new Button();
Grid1.Children.Add(aButton);
ListBox1.Items.Add(bButton);
```

```
// Entfernt die Steuerelemente, die in den Variablen aButton und bButton angegeben sind
Grid1.Children.Remove(aButton);
ListBox1.Items.Remove(bButton);
// Entfernt das Steuerelement mit dem Index 0 der Auflistung Controls
Grid1.Children.RemoveAt(0);
ListBox1.Items.RemoveAt(0);
```

Bei Inhaltselementen brauchen Sie lediglich auf die Eigenschaft `Content` zuzugreifen, um ein Steuerelement hinzuzufügen oder zu entfernen:

' Visual Basic

```
' Erstellt ein Button-Steuerelement und fügt es zu einem anderen Button-Steuerelement
'  namens Button1 hinzu
Dim aButton As New Button()
Button1.Content = aButton
' Entfernt aButton aus Button1
Button1.Content = Nothing
```

// C#

```
// Erstellt ein Button-Steuerelement und fügt es zu einem anderen Button-Steuerelement
//  namens Button1 hinzu
Button aButton = new Button();
Button1.Content = aButton;
// Entfernt aButton aus Button1
Button1.Content = null;
```

Wird ein Steuerelement aus seiner übergeordneten Auflistung entfernt, wird es vom Garbage Collector beseitigt, sobald keine anderen Verweise mehr darauf vorhanden sind.

Übung Arbeiten mit Grafik

In dieser Übung erweitern Sie eine vorher erstellte Anwendung durch verschiedene Effekte. Das Ergebnis ist für eine Unternehmensumgebung wahrscheinlich nicht seriös genug, demonstriert aber einige der Techniken, die Sie in dieser Lektion kennengelernt haben. Sie legen in dieser Übung eine Datei im Stammverzeichnis von Laufwerk *C:* an, daher brauchen Sie den entsprechenden Dateisystemzugriff.

Übung Arbeiten mit Grafik

1. Öffnen Sie die Lösung aus dem Unterverzeichnis *Partial* zu Kapitel 3, Lektion 1.
2. Ändern Sie in der XAML-Ansicht den XAML-Code für `Button1` so, dass er einen neuen `SolidColorBrush`-Pinsel in die Eigenschaft `Background` einträgt (beachten Sie, dass sich der XAML-Code für die C#-Übungsdateien leicht unterscheidet):

```
<Button Height="23" Margin="30,76,0,0" Name="Button1"
  VerticalAlignment="Top" HorizontalAlignment="Left" Width="74">
  <Button.Background>
    <SolidColorBrush Color="Red"></SolidColorBrush>
   </Button.Background>Set Name
</Button>
```

3. Gehen Sie bei `Button2` genauso vor. Ändern Sie den Hintergrund auf einen `SolidColorBrush`-Pinsel in einer anderen Farbe.

4. Ändern Sie den XAML-Code für das `TextBox`-Objekt, indem Sie einen `ScaleTransform`-Abschnitt einfügen, der das Textfeld an der Y-Achse spiegelt, aber die Position beibehält:

```
<TextBox RenderTransformOrigin=".5,.5" Height="21" Margin="30,29,81,0"
  Name="textBox1" VerticalAlignment="Top">
  <TextBox.RenderTransform>
    <ScaleTransform ScaleX="-1"/>
   </TextBox.RenderTransform>
</TextBox>
```

5. Fügen Sie einen `SkewTransform`-Abschnitt in `Label1` ein, um die Darstellung des Steuerelements zu scheren:

```
<Label Margin="30,124,81,115" Name="label1">
  <Label.RenderTransform>
    <SkewTransform AngleX="30" AngleY="20"></SkewTransform>
   </Label.RenderTransform>
</Label>
```

6. Fügen Sie direkt vor der `<Grid>`-Deklaration den folgenden Code ein, um dem Fensterhintergrund ein `LinearGradientBrush`-Objekt zuzuweisen:

```
<Window.Background>
  <LinearGradientBrush>
    <GradientStop Color="Red" Offset="0"/>
    <GradientStop Color="Yellow" Offset=".5"/>
    <GradientStop Color="Lime" Offset="1"/>
   </LinearGradientBrush>
</Window.Background>
```

7. Drücken Sie F5, um Ihre Anwendung zu erstellen und auszuführen. Die Funktion der Anwendung bleibt gleich, aber die Benutzeroberfläche sieht völlig anders aus.

Zusammenfassung der Lektion

- `Brush` ist in WPF die wichtigste Klasse zum Zeichnen der Benutzeroberfläche. Alle `Brush`-Klassen sind von der abstrakten Klasse `Brush` abgeleitet. `SolidColorBrush`-Objekte zeichnen einfarbige Flächen, `LinearGradientBrush`- und `RadialGradientBrush`-Objekte Farbverläufe. Mit `ImageBrush`- und `VisualBrush`-Objekten können Sie Bilder oder andere sichtbare Elemente zeichnen.
- WPF bietet integrierte Unterstützung zum Erstellen von Zeichenformen. Formen unterstützen alle Benutzerinteraktionsereignisse, die bei Benutzeroberflächenelementen zur Verfügung stehen. Es gibt simple Formen wie `Rectangle` oder `Ellipse` und komplexe Formen wie `Polygon` oder `Path`.
- Mit Transformationen wenden Sie mathematische Transformationen auf Formen und Elemente an. Sie wenden ein `Transform`-Objekt auf eine Form oder ein Element an, indem Sie es der Eigenschaft `RenderTransform` des Objekts zuweisen, das Sie transformieren wollen.

- Indem Sie der Eigenschaft `Clip` eines Elements ein `Geometry`-Objekt zuweisen, schneiden Sie die Darstellung dieses Objekts ab. Ein Element, dessen Eigenschaft `Clip` Sie ein `Geometry`-Objekt zugewiesen haben, wird nur in dem Bereich gezeichnet, der die Schnittmenge des normalen Elements und des `Geometry`-Objekts bildet.
- Die Methode `VisualTreeHelper.HitTest` gibt das Objekt zurück, das am angegebenen Punkt angeklickt wurde. Der gelieferte Punkt ist relativ zur oberen linken Ecke des sichtbaren Elements angegeben.

Lernzielkontrolle

Mit den folgenden Fragen können Sie Ihr Wissen zu den Informationen aus Lektion 1, »Verwalten der grafischen Oberfläche«, überprüfen. Die Fragen finden Sie (in englischer Sprache) auch auf der Begleit-CD, Sie können sie also auch auf dem Computer im Rahmen eines Übungstests beantworten.

Hinweis Die Antworten

Die Antworten auf diese Fragen mit Erklärungen, warum die jeweiligen Auswahlmöglichkeiten richtig oder falsch sind, finden Sie im Abschnitt »Antworten« am Ende dieses Buchs.

1. Welcher der folgenden XAML-Codeausschnitte zeichnet ein `Button`-Steuerelement, das an der X-Achse gespiegelt ist, aber an der ursprünglichen Position angezeigt wird?

   A.

```
<Button Height="23" Margin="75,64,128,0" Name="Button1"
  VerticalAlignment="Top">
  <Button.RenderTransform>
    <ScaleTransform ScaleY="-1"/>
   </Button.RenderTransform>Button
</Button>
```

   B.

```
<Button Height="23" Margin="75,64,128,0" Name="Button1"
  VerticalAlignment="Top" RenderTransformOrigin=".5,.5">
  <Button.RenderTransform>
    <ScaleTransform ScaleY="-1"/>
   </Button.RenderTransform>Button
</Button>
```

   C.

```
<Button Height="23" Margin="75,64,128,0" Name="Button1"
  VerticalAlignment="Top">
  <Button.RenderTransform>
    <ScaleTransform ScaleX="-1"/>
   </Button.RenderTransform>Button
</Button>
```

D.

```
<Button Height="23" Margin="75,64,128,0" Name="Button1"
  VerticalAlignment="Top" RenderTransformOrigin=".5,.5">
  <Button.RenderTransform>
    <ScaleTransform ScaleX="-1"/>
   </Button.RenderTransform>Button
</Button>
```

2. Welches der folgenden XAML-Beispiele erstellt einen `LinearGradient`-Pinsel mit einem vertikalen Farbverlauf von Rot zu Gelb?

A.

```
<LinearGradientBrush>
  <GradientStop Color="Red" Offset="0"/>
   <GradientStop Color="Yellow" Offset="1"/>
</LinearGradientBrush>
```

B.

```
<LinearGradientBrush StartPoint="0,1" EndPoint="1,1">
  <GradientStop Color="Red" Offset="0"/>
   <GradientStop Color="Yellow" Offset="1"/>
</LinearGradientBrush>
```

C.

```
<LinearGradientBrush StartPoint="0,1" EndPoint="1,1">
  <GradientStop Color="Yellow" Offset="0"/>
   <GradientStop Color="Red" Offset="1"/>
</LinearGradientBrush>
```

D.

```
<LinearGradientBrush StartPoint="1,0" EndPoint="1,1">
  <GradientStop Color="Red" Offset="0"/>
   <GradientStop Color="Yellow" Offset="1"/>
</LinearGradientBrush>
```

Lektion 2: Hinzufügen von Multimediainhalt

WPF gibt Ihnen die Möglichkeit, Multimediainhalte in Ihre Anwendungen zu integrieren. Mit den Klassen `SoundPlayer`, `MediaPlayer` und `MediaElement` können Sie Audio und Video nahtlos in verschiedene Elemente Ihrer Anwendung einbetten. In dieser Lektion erfahren Sie, wie Sie Ihre Benutzeroberfläche mit diesen drei Klassen um Audio- und Videoinhalte erweitern.

Am Ende dieser Lektion werden Sie in der Lage sein, die folgenden Aufgaben auszuführen:

- Abspielen von unkomprimierten *.wav*-Dateien mit der Klasse `SoundPlayer`
- Beschreiben der Klasse `SoundPlayerAction` und Erklären, wie sie in einem Trigger benutzt wird
- Abspielen von Audio in einer WPF-Anwendung mithilfe der Klasse `MediaPlayer`
- Anzeigen von Video in einer WPF-Anwendung mithilfe der Klasse `MediaElement`
- Steuern der Medienwiedergabe mit `MediaPlayer` und `MediaElement`
- Behandeln medienspezifischer Ereignisse

Veranschlagte Zeit für diese Lektion: 20 Minuten

Verwenden von *SoundPlayer*

Die Klasse `SoundPlayer` wurde in .NET Framework 2.0 als verwaltete Klasse eingeführt, um das Abspielen von Audio in Microsoft Windows-Anwendungen zu ermöglichen. Sie ist relativ simpel und einfach zu benutzen, weist aber auch erhebliche Einschränkungen auf.

`SoundPlayer` kann nur unkomprimierte *.wav*-Dateien abspielen. Sie kann keine komprimierten *.wav*-Dateien oder Dateien in anderen Audioformaten lesen. Außerdem hat der Entwickler keine Kontrolle über Lautstärke, Balance, Geschwindigkeit oder andere Aspekte der Audiowiedergabe.

Trotz der Einschränkungen kann `SoundPlayer` nützlich sein, wenn Sie ohne großen Aufwand Audiofähigkeiten in Ihre Anwendungen integrieren wollen. Sie stellt Member zur Verfügung, mit denen Sie ganz einfach unkomprimierte *.wav*-Dateien laden und abspielen können. Tabelle 3.12 zeigt wichtige Member der Klasse `SoundPlayer`.

Tabelle 3.12 Wichtige Member der Klasse `SoundPlayer`

Member	Typ	Beschreibung
`IsLoadCompleted`	Eigenschaft	Ein boolescher Wert, der angibt, ob die *.wav*-Datei vollständig geladen ist.
`Load`	Methode	Lädt die *.wav*-Datei aus den Eigenschaften `Stream` oder `SoundLocation` synchron.
`LoadAsync`	Methode	Lädt die *.wav*-Datei aus den Eigenschaften `Stream` oder `SoundLocation` asynchron.
`LoadCompleted`	Ereignis	Wird ausgelöst, sobald das Laden der *.wav*-Datei abgeschlossen ist.
`LoadTimeout`	Eigenschaft	Gibt an, nach wie vielen Millisekunden die *.wav*-Datei geladen sein muss. ▶

Member	Typ	Beschreibung
`Play`	Methode	Spielt die angegebene *.wav*-Datei asynchron ab.
`PlayLooping`	Methode	Spielt die angegebene *.wav*-Datei asynchron ab. Sobald das Ende der Datei erreicht ist, wird die Datei erneut von Anfang an abgespielt. Das wird so lange wiederholt, bis die Wiedergabe beendet wird.
`PlaySync`	Methode	Spielt die angegebene *.wav*-Datei synchron ab.
`SoundLocation`	Eigenschaft	Der Speicherort der *.wav*-Datei.
`SoundLocationChanged`	Ereignis	Wird ausgelöst, wenn sich die Eigenschaft `SoundLocation` ändert.
`Stop`	Methode	Beendet die Wiedergabe.
`Stream`	Eigenschaft	Das `Stream`-Objekt, das die *.wav*-Datei enthält.
`StreamChanged`	Ereignis	Wird ausgelöst, wenn sich die Eigenschaft `Stream` ändert.

Sie spielen eine *.wav*-Datei mit der Klasse `SoundPlayer` folgendermaßen ab:

1. Deklarieren Sie eine neue Instanz der Klasse `SoundPlayer`:

   ' Visual Basic
   ```
   Dim aPlayer As System.Media.SoundPlayer = New System.Media.SoundPlayer()
   ```
   // C#
   ```
   System.Media.SoundPlayer aPlayer = new System.Media.SoundPlayer();
   ```

2. Geben Sie den Speicherort der *.wav*-Datei an, indem Sie entweder den Dateipfad in die Eigenschaft `SoundLocation` eintragen oder der Eigenschaft `Stream` ein `Stream`-Objekt zuweisen, das die *.wav*-Datei enthält:

   ' Visual Basic
   ```
   aPlayer.SoundLocation = "C:\myFile.wav"
   ' ODER
   aPlayer.Stream = myStream
   ```
   // C#
   ```
   aPlayer.SoundLocation = "C:\\myFile.wav";
   // ODER
   aPlayer.Stream = myStream;
   ```

3. Laden Sie die *.wav*-Datei. Das ist zwar optional, schont aber die Systemressourcen, wenn die Datei mehrmals abgespielt wird. Falls Sie die Datei nicht auf diese Weise laden, wird sie jedes Mal, wenn Sie die Datei abspielen, erneut eingelesen.

   ' Visual Basic
   ```
   aPlayer.Load()
   ' ODER
   aPlayer.LoadAsync()
   ```
   // C#
   ```
   aPlayer.Load();
   // ODER
   aPlayer.LoadAsync();
   ```

4. Spielen Sie die *.wav*-Datei mit der Methode `Play` oder `PlaySync` ab:

```
' Visual Basic
aPlayer.Play()
' ODER
aPlayer.PlaySync()
```

```
// C#
aPlayer.Play();
// ODER
aPlayer.PlaySync();
```

SoundPlayerAction

WPF stellt für die Nutzung im XAML-Code eine Klasse namens `SoundPlayerAction` zur Verfügung, die ein `SoundPlayer`-Objekt einbettet. Mit der Klasse `SoundPlayerAction` können Sie ein `SoundPlayer`-Objekt deklarativ anlegen und eine *.wav*-Datei als Reaktion auf einen Trigger abspielen.

Die einzige wichtige Eigenschaft der Klasse `SoundPlayerAction` ist `Source`. Sie weisen ihr den Pfad der *.wav*-Datei zu, die Sie abspielen wollen. Die im nächsten Beispiel hervorgehobene Codezeile spielt mithilfe der Klasse `SoundPlayerAction` eine Audiodatei ab, sobald die Maus über einer Schaltfläche steht:

```
<Button Height="23" Margin="93,57,110,0" Name="button1"
  VerticalAlignment="Top">
  <Button.Content>Button</Button.Content>
  <Button.Style>
    <Style>
      <Style.Triggers>
        <EventTrigger RoutedEvent="Button.MouseEnter">
          <SoundPlayerAction Source="C:\myFile.wav"/>
        </EventTrigger>
      </Style.Triggers>
    </Style>
  </Button.Style>
</Button>
```

MediaPlayer und *MediaElement*

Die Klassen `MediaPlayer` und `MediaElement` bieten umfassende Unterstützung für das Abspielen von Audio- und Videodateien in vielen unterschiedlichen Formaten. Beide Klassen greifen auf die Funktionen des Windows Media Player 10 zurück. Sie funktionieren daher zuverlässig in Anwendungen, die unter Windows Vista oder neuer laufen, weil dort standardmäßig mindestens Media Player 11 vorhanden ist. In Windows XP-Installationen funktionieren diese Klassen dagegen nur, wenn mindestens Windows Media Player 10 installiert ist.

Die Klassen `MediaPlayer` und `MediaElement` sind sich sehr ähnlich und haben viele Member gemeinsam. Der Hauptunterschied zwischen den beiden Klassen ist, dass `MediaPlayer` zwar sowohl Audio als auch Video lädt und abspielt, aber keine Bedienoberfläche hat und daher kein Video in der Benutzeroberfläche anzeigen kann. `MediaElement` ist dagegen ein voll ausgestattetes WPF-Element, mit dem Sie Video in Ihren Anwendungen anzeigen können. `MediaElement` bettet ein `MediaPlayer`-Objekt ein und stellt eine Bedienoberfläche zum Abspie-

len von Videodateien bereit. Ein weiterer Unterschied ist, dass Sie `MediaPlayer` nicht ohne Weiteres im XAML-Code verwenden können, während `MediaElement` für den Einsatz in XAML entworfen wurde.

Die Tabelle 3.13 zeigt wichtige Eigenschaften von `MediaPlayer` und `MediaElement`, und die Tabelle 3.14 wichtige Methoden der beiden Klassen.

Tabelle 3.13 Wichtige Eigenschaften von `MediaPlayer` und `MediaElement`

Eigenschaft	Beschreibung
`Balance`	Die Balance zwischen dem linken und rechten Lautsprecher. Der Wertebereich reicht von –1 (nur linker Lautsprecher) bis 1 (nur rechter Lautsprecher). Der Wert 0 bedeutet, dass beide Lautsprecher gleichlaut angesteuert werden.
`BufferingProgress`	Eine Zahl, die als Wert zwischen 0 und 1 angibt, welcher Anteil der Datei gepuffert wurde. Nur gültig, wenn die Datei in einem Streamingformat vorliegt, das Pufferung ermöglicht.
`DownloadProgress`	Eine Zahl, die als Wert zwischen 0 und 1 angibt, welcher Anteil der Datei heruntergeladen wurde.
`HasAudio`	Gibt an, ob die aktuelle Mediendatei Audio enthält.
`HasVideo`	Gibt an, ob die aktuelle Mediendatei Video enthält.
`LoadedBehavior`	Legt fest, ob die Datei nach dem Laden automatisch abgespielt wird. Hat die Eigenschaft den Wert `Play`, wird die Datei nach dem Laden automatisch abgespielt. Hat sie den Wert `Manual`, wird die Datei erst abgespielt, wenn die Methode `Play` aufgerufen wird. Diese Eigenschaft gibt es nur in `MediaElement`.
`NaturalDuration`	Enthält ein `Duration`-Objekt, das die Länge der aktuellen Mediendatei angibt, wenn sie in ihrer normalen Geschwindigkeit abgespielt wird.
`NaturalVideoHeight`	Gibt die Originalhöhe für das Videobild der aktuellen Mediendatei an.
`NaturalVideoWidth`	Gibt die Originalbreite für das Videobild der aktuellen Mediendatei an.
`Position`	Gibt ein `TimeSpan`-Objekt zurück, das die aktuelle Position innerhalb der Mediendatei angibt.
`Source`	Gibt die URI (Uniform Resource Identifier) zurück, aus der die Mediendatei geladen wurde. Bei `MediaPlayer` ist dies eine schreibgeschützte Eigenschaft, die beim Aufruf der Methode `Open` zugewiesen wird. In `MediaElement` können Sie die Eigenschaft `Source` dagegen deklarativ im XAML-Code festlegen.
`SpeedRatio`	Ein positiver `Double`-Wert, der angibt, mit welcher Geschwindigkeit die Datei abgespielt wird. Der Wert 1 steht für normale Geschwindigkeit. Werte größer oder kleiner als 1 sind der Faktor, mit dem die Wiedergabegeschwindigkeit multipliziert wird. Der Wert 2 bedeutet also doppelte Geschwindigkeit und 0,5 halbe Geschwindigkeit.

Tabelle 3.14 Wichtige Methoden von `MediaPlayer` und `MediaElement`

Methode	Beschreibung
`Open`	Öffnet die angegebene Mediendatei. Sie übergeben dieser Methode als Argument eine URI, die auf die Mediendatei verweist. Diese Methode steht nur in `MediaPlayer` zur Verfügung.
`Pause`	Unterbricht die Wiedergabe der aktuellen Datei, ändert aber nicht die Eigenschaft `Position`.
`Play`	Beginnt die Wiedergabe der Datei oder setzt die Wiedergabe der Datei fort, falls vorher die Methode `Pause` aufgerufen wurde.
`Stop`	Beendet die Wiedergabe und setzt die Eigenschaft `Position` auf den Anfang der Datei.

So spielen Sie Audio mit `MediaPlayer` ab:

1. Deklarieren Sie eine neue Instanz von `MediaPlayer`:

   ' Visual Basic
   ```
   Dim aPlayer As New System.Windows.Media.MediaPlayer()
   ```
   // C#
   ```
   System.Windows.Media.MediaPlayer aPlayer = new MediaPlayer();
   ```
2. Laden Sie die Mediendatei mit der Methode `Load`:

   ' Visual Basic
   ```
   aPlayer.Open(new Uri("crash.mp3"))
   ```
   // C#
   ```
   aPlayer.Open(new Uri("crash.mp3"));
   ```
3. Rufen Sie die Methode `Play` auf, um die Datei abzuspielen:

   ' Visual Basic
   ```
   aPlayer.Play()
   ```
   // C#
   ```
   aPlayer.Play();
   ```

So spielen Sie Audio oder Video automatisch mit `MediaElement` ab:

1. Deklarieren Sie im XAML-Code ein neues `MediaElement`-Objekt:
   ```
   <MediaElement Margin="52,107,66,35" Name="mediaElement1"/>
   ```
2. Tragen Sie in die Eigenschaft `Source` deklarativ die gewünschte Datei ein:
   ```
   <MediaElement Margin="52,107,66,35" Source="crash.mp3"
      Name="mediaElement1"/>
   ```
3. Die Datei wird sofort nach dem Laden abgespielt, was unmittelbar nach dem Laden des Fensters passiert. Der nächste Abschnitt beschreibt, wie Sie steuern, wann die Datei abgespielt wird.

So spielen Sie Audio oder Video mit `MediaElement` von Hand ab:

1. Deklarieren Sie im XAML-Code ein neues `MediaElement`-Objekt:
   ```
   <MediaElement Margin="52,107,66,35" Name="mediaElement1"/>
   ```
2. Tragen Sie in die Eigenschaft `Source` deklarativ die gewünschte Datei ein:
   ```
   <MediaElement Margin="52,107,66,35" Source="crash.mp3"
      Name="mediaElement1"/>
   ```

3. Setzen Sie die Eigenschaft `LoadedBehavior` auf `Manual`:

```
<MediaElement Margin="52,107,66,35" Source="crash.mp3"
   LoadedBehavior="Manual" Name="mediaElement1"/>
```

4. Rufen Sie im Programmcode die Methode `Play` auf, um die Datei abzuspielen:

```
' Visual Basic
mediaElement1.Play()
// C#
mediaElement1.Play();
```

Schnelltest

- Was ist der Unterschied zwischen `MediaPlayer` und `MediaElement`?

Antwort zum Schnelltest

- `MediaPlayer` und `MediaElement` bieten zwar denselben Funktionsumfang, aber `MediaElement` wird sichtbar dargestellt, während `MediaPlayer` eine Komponente ohne Bedienoberfläche ist. Daher sollten Sie `MediaElement` verwenden, um Video in Ihren Anwendungen anzuzeigen.

Behandeln medienspezifischer Ereignisse

`MediaPlayer` und `MediaElement` lösen drei medienspezifische Ereignisse aus, die in Tabelle 3.15 beschrieben werden.

Tabelle 3.15 Medienspezifische Ereignisse in `MediaPlayer` und `MediaElement`

Ereignis	Beschreibung
`MediaEnded`	Wird ausgelöst, wenn die Mediendatei fertig abgespielt wurde.
`MediaFailed`	Wird ausgelöst, wenn eine Datei nicht gefunden oder geladen werden kann.
`MediaOpened`	Wird ausgelöst, wenn das Laden der Mediendatei abgeschlossen ist.

Die Ereignisse `MediaEnded` und `MediaOpened` sind Standard-WPF-Routingereignisse, beide übergeben eine Instanz von `RoutedEventArgs` an den Ereignishandler. Sie können diese Ereignisse behandeln, um Code auszuführen, wenn die Datei geladen oder fertig abgespielt wurde. Das funktioniert genauso wie bei allen anderen Ereignissen.

Das Ereignis `MediaFailed` arbeitet etwas anders. Die Klassen `MediaElement` und `MediaPlayer` lösen keine Ausnahme aus, wenn sie die gewünschte Mediendatei nicht finden oder abspielen können, sondern das Ereignis `MediaFailed`. Daher kann die Ausführung der Anwendung fortgesetzt werden, selbst wenn ein Problem mit der Multimediadatei aufgetreten ist. Es wird zwar beim ursprünglichen Fehler eine Ausnahme ausgelöst, sie wird aber in eine Instanz von `ExceptionRoutedEventArgs` eingebettet, die an den Ereignishandler übergeben wird. Sie erhalten eine Instanz der Ausnahme, die ursprünglich ausgelöst wurde, aus der Eigenschaft `ErrorException` der `ExceptionRoutedEventArgs`-Instanz. Indem Sie die Eigenschaften der eingebetteten Ausnahme untersuchen, können Sie festlegen, wie die Anwendung reagiert. Das folgende Beispiel demonstriert eine simple Möglichkeit, das Ereignis `MediaFailed` zu behandeln. Es zeigt ein Meldungsfeld mit dem Text der eingebetteten Ausnahme an:

' Visual Basic

```
Private Sub mediaElement1_MediaFailed(ByVal sender As System.Object, _
  ByVal e As System.Windows.ExceptionRoutedEventArgs)
  MessageBox.Show(e.ErrorException.Message)
End Sub
```

// C#

```
private void mediaElement1_MediaFailed(object sender,
  ExceptionRoutedEventArgs e)
{
  MessageBox.Show(e.ErrorException.Message);
}
```

Übung Erstellen eines einfachen Mediaplayers

In dieser Übung erstellen Sie einen einfachen Mediaplayer, der die Fähigkeit bietet, Audio- und Videodateien zu laden und abzuspielen sowie die Wiedergabe zu starten, zu beenden und zu unterbrechen.

Übung Erstellen eines Mediaplayers

1. Öffnen Sie die Lösung aus dem Unterverzeichnis *Partial* zu Kapitel 3, Lektion 2. Diese Projektmappe enthält einen Verweis auf den Namespace `System.Windows.Forms`. Sie verwenden die Klasse `OpenFileDialog` aus diesem Namespace, um das Dateisystem zu durchsuchen.
2. Fügen Sie im XAML-Code ein `MediaElement` zur obersten Zeile des `Grid`-Steuerelements hinzu und setzen Sie die Eigenschaft `LoadedBehavior` auf den Wert `Manual`:

   ```
   <MediaElement LoadedBehavior="Manual" Margin="0" Name="MediaElement1"/>
   ```

3. Fügen Sie im XAML-Code vier Schaltflächen mit den Beschriftungen *Öffnen*, *Wiedergabe*, *Pause* und *Stopp* zur Symbolleiste hinzu und definieren Sie `Click`-Ereignishandler für jede Schaltfläche:

   ```
   <ToolBar Grid.Row="1" Margin="0" Name="ToolBar1">
     <Button Click="Button_Click">Öffnen</Button>
     <Button Click="Button_Click_1">Wiedergabe</Button>
     <Button Click="Button_Click_2">Pause</Button>
      <Button Click="Button_Click_3">Stopp</Button>
   </ToolBar>
   ```

4. Klicken Sie im Designer doppelt auf die Schaltfläche *Öffnen*, um den `Click`-Ereignishandler zu öffnen. Fügen Sie den folgenden Code ein:

   ' Visual Basic

   ```
   aDialog.ShowDialog()
   MediaElement1.Source = New Uri(aDialog.FileName)
   MediaElement1.Play()
   ```

   // C#

   ```
   aDialog.ShowDialog();
   MediaElement1.Source = new Uri(aDialog.FileName);
   MediaElement1.Play();
   ```

5. Klicken Sie im Designer doppelt auf die Schaltfläche *Wiedergabe*, um den `Click`-Ereignishandler zu öffnen. Fügen Sie den folgenden Code ein:

   ' Visual Basic
   ```
   MediaElement1.Play()
   ```
   // C#
   ```
   MediaElement1.Play();
   ```

6. Klicken Sie im Designer doppelt auf die Schaltfläche *Pause*, um den `Click`-Ereignishandler zu öffnen. Fügen Sie den folgenden Code ein:

   ' Visual Basic
   ```
   MediaElement1.Pause()
   ```
   // C#
   ```
   MediaElement1.Pause();
   ```

7. Klicken Sie im Designer doppelt auf die Schaltfläche *Stopp*, um den `Click`-Ereignishandler zu öffnen. Fügen Sie den folgenden Code ein:

   ' Visual Basic
   ```
   MediaElement1.Stop()
   ```
   // C#
   ```
   MediaElement1.Stop();
   ```

8. Fügen Sie im Designer eine Definition für das Ereignis `MediaElement1.MediaFailed` ein:

   ```
   <MediaElement MediaFailed="MediaElement1_MediaFailed"
      LoadedBehavior="Manual" Margin="0" Name="MediaElement1"/>
   ```

9. Fügen Sie im Code-Editor den folgenden Code in die Methode `MediaElement1_MediaFailed` ein:

   ' Visual Basic
   ```
   MessageBox.Show("Mediendatei konnte nicht geladen werden. " &
   e.ErrorException.Message)
   ```
   // C#
   ```
   MessageBox.Show("Mediendatei konnte nicht geladen werden. " +
   e.ErrorException.Message);
   ```

10. Drücken Sie F5, um Ihre Anwendung zu erstellen und auszuführen. Klicken Sie auf die Schaltfläche *Öffnen*, suchen Sie eine Mediendatei und testen Sie die Funktion aller Schaltflächen. Klicken Sie nun erneut auf *Öffnen* und wählen Sie eine Datei aus, die keine Mediendatei ist. Prüfen Sie, ob der Fehlerbehandlungscode wie erwartet funktioniert.

Zusammenfassung der Lektion

- Die Klasse `SoundPlayer` ermöglicht das unkomplizierte Abspielen von unkomprimierten *.wav*-Dateien. Sie kann aber keine komprimierten *.wav*-Dateien oder andere Audiodateien abspielen. Sie können ein `SoundPlayer`-Objekt deklarativ erstellen, indem Sie ein `SoundPlayerAction`-Objekt einfügen.
- `MediaPlayer` bietet umfassende Unterstützung zum Abspielen von Mediendateien. `MediaPlayer` beherrscht den Umgang mit vielen Audio- und Videodateitypen, hat aber keine Bedienoberfläche. `MediaElement` bettet ein `MediaPlayer`-Objekt ein und stellt eine Bedien-

oberfläche zum Anzeigen von Video bereit. Beide Elemente bieten die Möglichkeit, Lautstärke, Balance und andere Wiedergabefunktionen einzustellen.

- `MediaPlayer` und `MediaElement` lösen keine Ausnahmen aus, wenn ein Problem mit einer Mediendatei auftritt. Stattdessen müssen Sie das Ereignis `MediaFailed` behandeln, das eine eingebettete Ausnahme enthält, in der das Problem beschrieben ist. Sie können dann im Ereignishandler die gewünschte Aktion ausführen.

Lernzielkontrolle

Mit den folgenden Fragen können Sie Ihr Wissen zu den Informationen aus Lektion 2, »Hinzufügen von Multimediainhalt«, überprüfen. Die Fragen finden Sie (in englischer Sprache) auch auf der Begleit-CD, Sie können sie also auch auf dem Computer im Rahmen eines Übungstests beantworten.

Hinweis Die Antworten

Die Antworten auf diese Fragen mit Erklärungen, warum die jeweiligen Auswahlmöglichkeiten richtig oder falsch sind, finden Sie im Abschnitt »Antworten« am Ende dieses Buchs.

1. Wie behandeln Sie Fehler beim Laden einer Mediendatei in einem `MediaElement`-Objekt?
   A. Fangen Sie den Fehler in einem `Try...Catch`-Block ab und schreiben Sie Code, mit dem die Anwendung den Fehler korrigiert.
   B. Ignorieren Sie Fehler zu Mediendateien. Sie führen nicht zum Absturz der Anwendung.
   C. Behandeln Sie das Ereignis `MediaElement.MediaFailed`.
   D. Überprüfen Sie im Ereignis `MediaElement.MediaOpened`, ob der richtige Dateityp geöffnet wurde.
2. Welche Klasse ist die beste Wahl, wenn Sie in einer Anwendung unkomprimierte *.wav*-Dateien abspielen wollen?
   A. `SoundPlayer`
   B. `MediaPlayer`
   C. `MediaElement`
   D. Entweder A oder B, das hängt von den Anforderungen der Anwendung ab

Übung mit Fallbeispiel

In der folgenden Übung mit Fallbeispiel wenden Sie an, was Sie über das Hinzufügen und Verwalten von Inhalt gelernt haben. Die Lösungen zu den Fragen finden Sie im Abschnitt »Antworten« hinten in diesem Buch.

Übung mit Fallbeispiel: Das Unternehmen mit zweifelhaftem Geschmack

Sie entwickeln eine Anwendung für ein Unternehmen, in dem ein seltsames Geschmacksempfinden vorherrscht. Anscheinend hat das mit Konzerten zu tun, bei denen die Gründer Anfang der 1970er waren. Sie möchten, dass die Hauptanwendung rein schwarz-weiß gehalten ist. Alle Steuerelemente, in die Benutzer Daten eingeben, sollen dagegen in Batikmustern erstrahlen. Außerdem soll in der oberen linken Ecke der Anwendung permanent eine Mediendatei mit einem Konzertvideo laufen. Dieses Video muss die Form eines tanzenden Bären haben. Der Grafiker hat Ihnen kopfschüttelnd ein `Path`-Objekt in der Form eines tanzenden Bären entworfen, aber den Rest müssen Sie selbst erledigen.

Beantworten Sie Ihrem Manager folgende Fragen:

1. Wie können Sie die Eingabesteuerelemente in Batikmustern erstellen?
2. Wie schaffen Sie es bloß, ein Video in Form eines tanzenden Bären anzuzeigen?

Vorgeschlagene Übungen

Sie sollten die folgenden Aufgaben durcharbeiten, wenn Sie die in diesem Kapitel behandelten Prüfungsziele meistern wollen:

- **Übung 1** Verbessern Sie die Mediaplayer-Anwendung aus Lektion 2, damit der Benutzer Lautstärke, Balance und Wiedergabegeschwindigkeit einstellen kann.
- **Übung 2** Erforschen Sie die Klassen `VisualBrush` und `ImageBrush` und entwickeln Sie eine Anwendung, die mit diesen Pinseln den Hintergrund mit einer Vielzahl von Effekten zeichnet.
- **Übung 3** Erstellen Sie eine Mediaplayer-Anwendung, die auf Mediendateien zugreift, die als Binärressourcen eingebettet sind. (Hinweis: Sie müssen dabei mehrmals das Datei-E/A-System benutzen.)

Machen Sie einen Übungstest

Die Übungstests (in englischer Sprache) auf der Begleit-CD zu diesem Buch bieten zahlreiche Möglichkeiten. Zum Beispiel können Sie einen Test machen, der ausschließlich die Themen aus einem Prüfungslernziel behandelt, oder Sie können sich selbst mit dem gesamten Inhalt der Prüfung 70-511 testen. Sie können den Test so konfigurieren, dass er dem Ablauf einer echten Prüfung entspricht, oder Sie können einen Lernmodus verwenden, in dem Sie sich nach dem Beantworten einer Frage jeweils sofort die richtige Antwort und Erklärungen ansehen können.

Weitere Informationen Übungstests

Einzelheiten zu allen Optionen, die bei den Übungstests zur Verfügung stehen, finden Sie im Abschnitt »So benutzen Sie die Übungstests« in der Einführung am Anfang dieses Buchs.