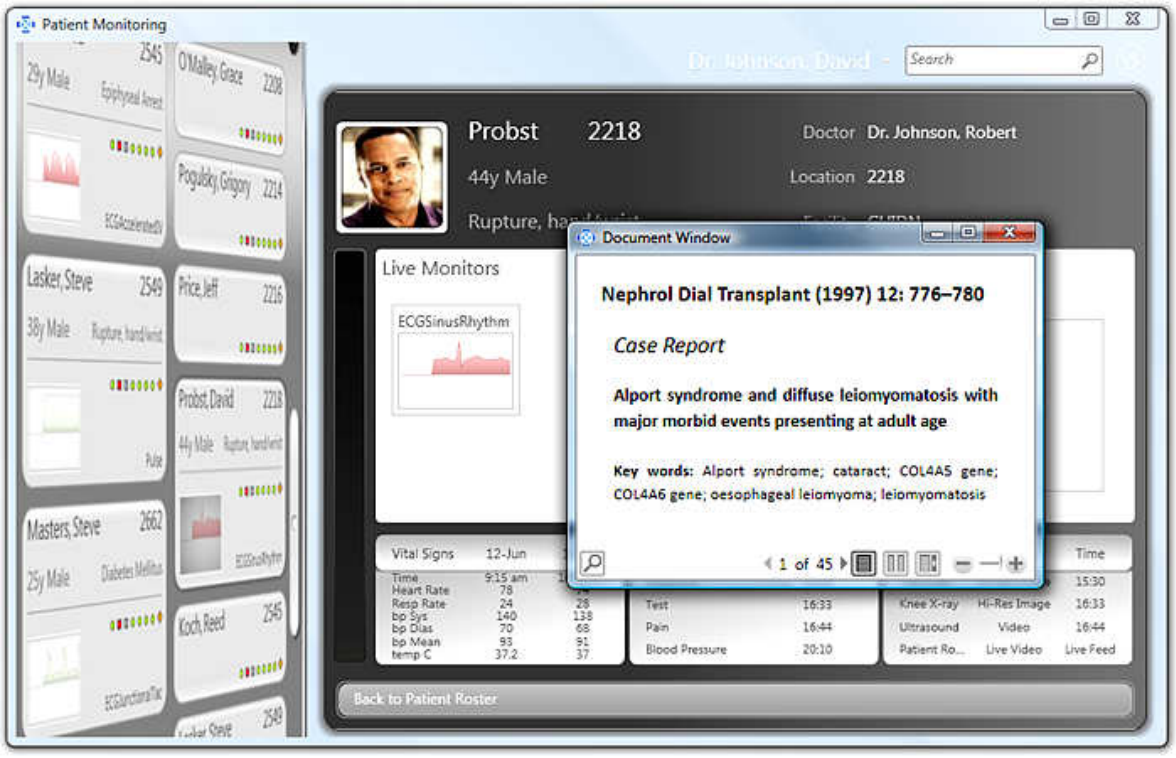
Einführung in WPF

.NET Framework 4

Windows Presentation Foundation (WPF) ist ein Präsentationssystem der nächsten Generation zum Erstellen von Windows-Clientanwendungen mit einem visuell herausragenden Benutzererlebnis. Mit WPF kann ein breites Spektrum an eigenständigen oder in einem Browser gehosteten Anwendungen erstellt werden. Ein Beispiel ist die Beispielanwendung Contoso Healthcare, die in der folgenden Abbildung dargestellt wird.



Der Kern von WPF ist ein auflösungsunabhängiges und vektorbasiertes Renderingmodul, das die Funktionen moderner Grafikhardware nutzt. Dieser Kern wird von WPF um einen umfassenden Satz von Anwendungsentwicklungsfunktionen erweitert. Hierzu zählen Extensible Application Markup Language (XAML), Steuerelemente, Datenbindung, Layout, 2-D- und 3-D-Grafik, Animationen, Stile, Vorlagen, Dokumente, Medien, Text und Typographie. Da WPF in Microsoft .NET Framework enthalten ist, können Sie Anwendungen erstellen, die andere Elemente der .NET Framework-Klassenbibliothek beinhalten.

Diese Übersicht ist für Einsteiger gedacht und beschreibt die wichtigsten Funktionen und Begriffe von WPF. Sie ist möglicherweise auch für erfahrene WPF-Entwickler geeignet, die einen Überblick über WPF erhalten möchten.

Hinweis

Informationen zu neuen und aktualisierten Funktionen von WPF in .NET Framework 4 finden Sie unter Neues in WPF Version 4.

Dieses Thema enthält folgende Abschnitte.

Programmieren mit WPF

WPF ist eine Teilmenge von .NET Framework-Typen, die sich zum größten Teil im System.Windows-Namespace befinden. Wenn Sie zuvor Anwendungen mit .NET Framework unter Verwendung verwalteter Technologien wie ASP.NET und Windows Forms erstellt haben, sollten Ihnen die Grundlagen der WPF-Programmierung vertraut sein. Es werden Klassen instanziiert, Eigenschaften festgelegt, Methoden aufgerufen, Ereignisse behandelt, und das alles mit Ihrer bevorzugten .NET Framework-Programmiersprache, wie z. B. C# oder Visual Basic.

Zur Unterstützung einiger fortgeschrittener WPF-Funktionen und zur Vereinfachung der Programmierung enthält WPF zusätzliche Programmierkonstrukte, die Eigenschaften und Ereignisse optimieren: *Abhängigkeitseigenschaften* und *Routingereignisse*. Weitere Informationen zu Abhängigkeitseigenschaften finden Sie unter Übersicht über Abhängigkeitseigenschaften. Weitere Informationen über Routingereignisse finden Sie unter Übersicht über Routingereignisse.

Markup und Code-Behind

WPF bietet zusätzliche Programmiererweiterungen zur Entwicklung von Clientanwendungen für Windows. Eine maßgebliche Verbesserung ist die Möglichkeit, eine Anwendung sowohl mit *Markup* als auch mit *Code-Behind* zu entwickeln, eine Erfahrung, mit der ASP.NET-Entwickler vertraut sein sollten. Im Allgemeinen wird mithilfe von Extensible Application Markup Language (XAML)-Markup die Darstellung einer Anwendung implementiert und mithilfe von verwalteten Programmiersprachen (Code-Behind) das entsprechende Verhalten. Diese Trennung von Darstellung und Verhalten bietet folgende Vorteile:

- Entwicklungs- und Wartungskosten werden reduziert, da darstellungsspezifisches Markup und verhaltensspezifischer Code nicht eng aneinander gekoppelt sind.
- Die Entwicklung ist effizienter, da Designer und Entwickler parallel die Darstellung einer Anwendung und das Verhalten der Anwendung implementieren können.
- Mithilfe mehrerer Entwurfstools kann XAML-Markup implementiert und freigegeben werden, um die Anforderungen der beteiligten Anwendungsentwickler zu erfüllen. Microsoft Expression Blend ist auf Designer zugeschnitten, während sich Visual Studio 2005 an Entwickler richtet.
- Die Globalisierung und Lokalisierung für WPF-Anwendungen wird stark vereinfacht (siehe Übersicht über WPF-Globalisierung und -Lokalisierung).

Nachfolgend erhalten Sie eine kurze Einführung in WPF-Markup und Code-Behind. Weitere Informationen über dieses Programmiermodell finden Sie unter Übersicht über XAML (WPF) und unter Code-Behind und XAML in WPF.

Markup

XAML ist eine auf XML basierende Markupsprache, mit der die Darstellung einer Anwendung deklarativ implementiert werden kann. Sie wird üblicherweise dazu verwendet, Fenster, Dialogfelder, Seiten und Benutzersteuerelemente zu erstellen und diese mit Steuerelementen, Formen und Grafiken zu füllen.

Im folgenden Beispiel wird mithilfe von XAML die Darstellung eines Fensters mit einer einzelnen Schaltfläche implementiert.

```
<Window
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    Title="Window with Button"
    Width="250" Height="100">

  <!-- Add button to window -->
  <Button Name="button">Click Me!</Button>

</Window>
```

Insbesondere werden dabei durch XAML ein Fenster und eine Schaltfläche mithilfe der Elemente **Window** bzw. **Button** definiert. Jedes Element wird mit Attributen konfiguriert, wie z. B. beim **Window**-Element, durch dessen **Title**-Attribut der Text in der Titelleiste des Fensters festgelegt wird. Zur Laufzeit werden die in Markup definierten Elemente und Attribute von WPF in Instanzen von WPF-Klassen konvertiert. Beispielsweise wird das **Window**-Element in eine Instanz der Window-Klasse konvertiert, deren Title-Eigenschaft dem Wert des **Title**-Attributs entspricht.

In der folgenden Abbildung wird die user interface (UI) dargestellt, die im vorherigen Beispiel mithilfe von XAML definiert wurde.

Weitere Informationen finden Sie unter Übersicht über XAML (WPF).

Da XAML auf XML basiert, wird die damit erstellte UI in einer Hierarchie geschachtelter Elemente erstellt. Diese wird als *Elementstruktur* bezeichnet. Mithilfe der Elementstruktur können UIs auf logische und intuitive Weise erstellt und verwaltet werden. Weitere Informationen finden Sie unter Strukturen in WPF.

Code-Behind

Das hauptsächliche Verhalten einer Anwendung liegt darin, die Funktionalität zu implementieren, mit der auf die Interaktionen des Benutzers reagiert wird. Dazu zählen das Behandeln von Ereignissen (z. B. Klicken auf Menüs, Symbolleisten oder Schaltflächen) sowie das Aufrufen von Geschäftslogik und Datenzugriffslogik in Reaktion darauf. In WPF wird dieses Verhalten im Allgemeinen in Code implementiert, der mit Markup verknüpft ist. Diese Art von Code wird als Code-Behind bezeichnet. Im folgenden Beispiel werden Code-Behind und aktualisiertes Markup aus dem vorherigen Beispiel dargestellt.

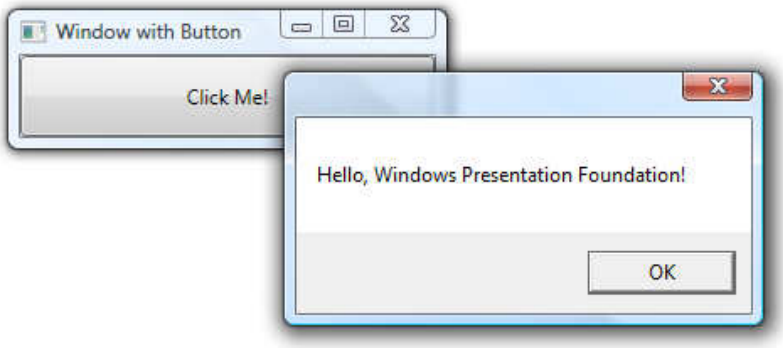
```
<Windowxmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"xmlns:x="http://schemas.microsoft.c
```

C#

```
using System.Windows; // Window, RoutedEventArgs, MessageBoxnamespace SDKSample
{
    publicpartialclass AWindow : Window
    {
        public AWindow()
        {
            // InitializeComponent call is required to merge the UI// that is defined in markup with this c
            InitializeComponent();
        }

        void button_Click(object sender, RoutedEventArgs e)
        {
            // Show message box when button is clicked
            MessageBox.Show("Hello, Windows Presentation Foundation!");
        }
    }
}
```

In diesem Beispiel wird durch Code-Behind eine Klasse implementiert, die von der Window-Klasse abgeleitet wird. Das **x:Class**-Attribut wird verwendet, um das Markup der Code-Behind-Klasse zuzuordnen. **InitializeComponent** wird vom Konstruktor der Code-Behind-Klasse aufgerufen, um die in Markup definierte Benutzeroberfläche mit der Code-Behind-Klasse zusammenzuführen. (**InitializeComponent** wird bei der Erstellung der Anwendung automatisch generiert, eine manuelle Implementierung ist daher nicht notwendig.) Mit der Kombination von **x:Class** und **InitializeComponent** wird sichergestellt, dass die Implementierung bei der Erstellung immer ordnungsgemäß initialisiert wird. Von der Code-Behind-Klasse wird außerdem ein Ereignishandler für das Click-Ereignis der Schaltfläche implementiert. Beim Klicken auf die Schaltfläche wird vom Ereignishandler durch Aufrufen der MessageBox.Show-Methode ein Meldungsfeld angezeigt.

In der folgenden Abbildung wird das Ergebnis nach dem Klicken auf die Schaltfläche dargestellt.



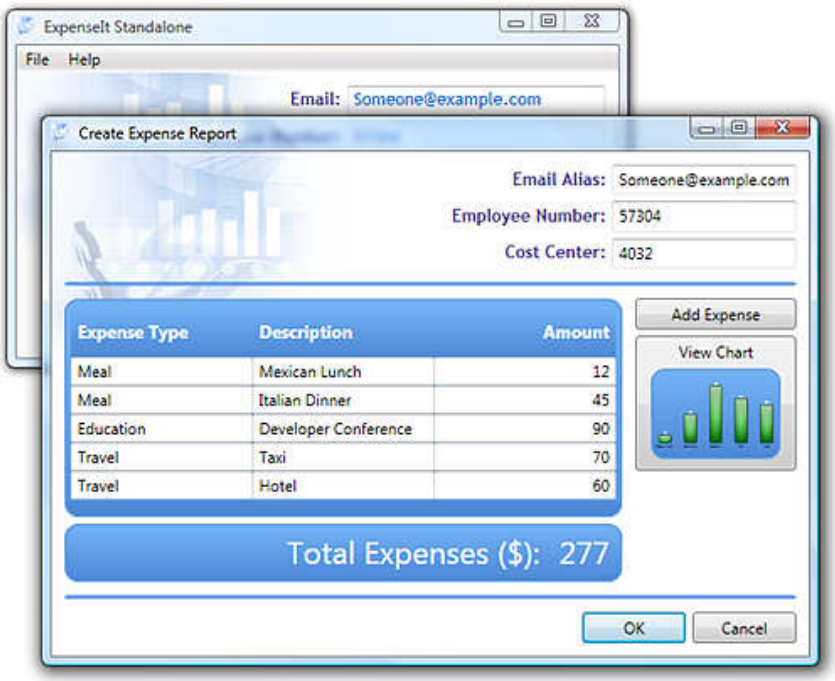
Weitere Informationen finden Sie unter Code-Behind und XAML in WPF.

Anwendungen

.NET Framework, System.Windows sowie Markup und Code-Behind bilden die Grundlage für die Anwendungsentwicklung mit WPF. Darüber hinaus verfügt WPF über umfassende Funktionen zum Erstellen von Benutzerumgebungen mit umfangreichen Inhalten. Um diese Inhalte zu verpacken und für Benutzer als Anwendungen bereitzustellen, bietet WPF Typen und Dienste, die zusammengefasst als *Anwendungsmodell* bezeichnet werden. Das Anwendungsmodell unterstützt die Entwicklung von eigenständigen und im Browser gehosteten Anwendungen.

Eigenständige Anwendungen

Für eigenständige Anwendungen können Sie die Window-Klasse verwenden, um Fenster und Dialogfelder zu erstellen, auf die über Menüleisten und Symbolleisten zugegriffen werden kann. Die folgende Abbildung zeigt eine eigenständige Anwendung mit einem Hauptfenster und einem Dialogfeld.

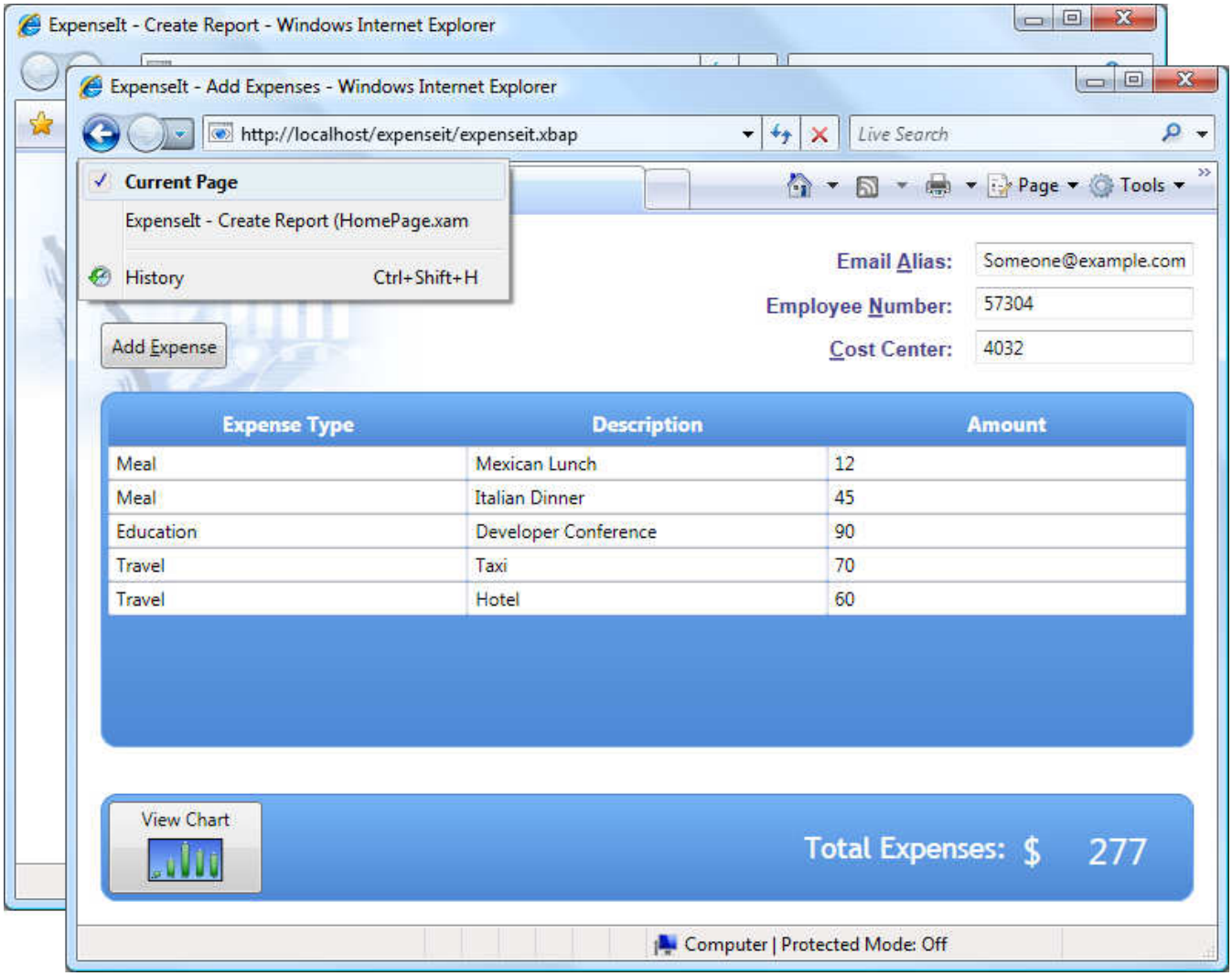


Darüber hinaus können Sie die folgenden WPF-Dialogfelder verwenden: MessageBox, OpenFileDialog, SaveFileDialog, und PrintDialog.

Weitere Informationen finden Sie unter Übersicht über WPF-Fenster.

Im Browser gehostete Anwendungen

Für im Browser gehostete Anwendungen (XAML browser applications (XBAPs)) können Sie Seiten (Page) und Seitenfunktionen (PageFunction<T>) erstellen, zwischen denen Sie mithilfe von Links (Hyperlink-Klassen) navigieren können. Die folgende Abbildung zeigt eine Seite in XBAP, die in Internet Explorer 7 gehostet wird.



WPF-Anwendungen können sowohl in Microsoft Internet Explorer 6 als auch in Internet Explorer 7 gehostet werden. WPF bietet die beiden folgenden Optionen für alternative Navigationshosts:

- Frame zum Hosten von Inseln navigierbarer Inhalte in Seiten oder Fenstern.
- NavigationWindow zum Hosten navigierbarer Inhalte in einem gesamten Fenster.

Weitere Informationen finden Sie unter Übersicht über die Navigation.

Die Application-Klasse

XBAPs und eigenständige Anwendungen sind häufig so komplex, dass zusätzliche anwendungsspezifische Dienste erforderlich sind, z. B. Start- und Lebensdauerverwaltung sowie freigegebene Eigenschaften und Ressourcen. Diese und andere Dienste sind in der Application-Klasse gekapselt, die wie im folgenden Beispiel gezeigt einfach mit XAML implementiert werden kann.

```
<Application
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    StartupUri="MainWindow.xaml" />
```

Dieser Markupcode ist die *Anwendungsdefinition* für eine eigenständige Anwendung. Durch ihn wird WPF angewiesen, ein Application-Objekt zu erstellen, durch das MainWindow beim Start der Anwendung automatisch geöffnet wird.

Als wichtiger Aspekt sollte verstanden werden, dass Application eine allgemeine Plattform zur Unterstützung von eigenständigen oder im Browser gehostete Anwendungen bereitstellt. Beispielsweise könnte das vorangehende XAML von einer im Browser gehosteten Anwendung verwendet werden, um automatisch zu einer Seite zu navigieren, wenn ein XBAP gestartet wird (siehe folgendes Beispiel).

```
<Application
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    StartupUri="HomePage.xaml" />
```

Weitere Informationen finden Sie unter Übersicht über die Anwendungsverwaltung.

Sicherheit

Da XBAPs-Anwendungen in einem Browser gehostet werden, spielt Sicherheit eine wichtige Rolle. XBAPs-Anwendungen verwenden eine teilweise vertrauenswürdige Sandbox für die Sicherheit, um Beschränkungen zu erzwingen, die geringer oder gleichwertig mit den Beschränkungen sind, die für HTML-basierte Anwendungen gelten. Außerdem wurde jedes HTML-Feature, das in teilweiser Vertrauenswürdigkeit von XBAPs sicher ausgeführt werden kann, unter Verwendung eines umfassenden Sicherheitsprozesses getestet. Ausführliche Informationen dazu finden Sie unter WPF-Sicherheitsstrategie – Sicherheitsentwicklung.

Über XBAPs kann immer noch ein Großteil der WPF-Features sicher ausgeführt werden, wie unter WPF-Sicherheit mit teilweiser Vertrauenswürdigkeit beschrieben.

Steuerelemente

Dem Benutzer werden mithilfe des Anwendungsmodells konstruierte Steuerelemente bereitgestellt. In WPF ist "Steuerelement" ein Sammelbegriff, der sich auf eine Kategorie von WPF-Klassen bezieht, die in einem Fenster oder auf einer Seite gehostet werden, über eine user interface (UI) verfügen und ein bestimmtes Verhalten implementieren.

Weitere Informationen finden Sie unter Steuerelemente.

WPF-Steuerelemente nach Funktion

Die integrierten WPF-Steuerelemente sind hier aufgeführt.

- **Schaltflächen**: Button und RepeatButton.
- **Datenanzeige**: DataGrid, ListView und TreeView.
- **Datumsanzeige und -auswahl**: Calendar und DatePicker.
- **Dialogfelder**: OpenFileDialog, PrintDialog und SaveFileDialog.
- **Freihandeingaben**: InkCanvas und InkPresenter.
- **Dokumente**: DocumentViewer, FlowDocumentPageViewer, FlowDocumentReader, FlowDocumentScrollViewer und StickyNoteControl.
- **Eingabe**: TextBox, RichTextBox und PasswordBox.
- **Layout**: Border, BulletDecorator, Canvas, DockPanel, Expander, Grid, GridView, GridSplitter, GroupBox, Panel, ResizeGrip, Separator, ScrollBar, ScrollViewer, StackPanel, Thumb, Viewbox, VirtualizingStackPanel, Window und WrapPanel.
- **Medien**: Image, MediaElement und SoundPlayerAction.
- **Menüs**: ContextMenu, Menu und ToolBar.
- **Navigation**: Frame, Hyperlink, Page, NavigationWindow und TabControl.
- **Auswahl**: CheckBox, ComboBox, ListBox, RadioButton und Slider.
- **Benutzerinformationen**: AccessText, Label, Popup, ProgressBar, StatusBar, TextBlock und ToolTip.

Eingabe und Befehle

Steuerelemente reagieren hauptsächlich auf Benutzereingaben. Vom WPF-Eingabesystem werden sowohl direkte Ereignisse als auch Routingereignisse für Texteingabe, Fokusverwaltung und Mauspositionierung verwendet. Weitere Informationen finden Sie unter Übersicht über die Eingabe.

Anwendungen besitzen häufig komplexe Eingabeanforderungen. WPF bietet ein Befehlssystem, bei dem die Eingabeaktionen des Benutzers von dem Code, mit dem auf diese Aktionen reagiert wird, getrennt sind. Weitere Informationen finden Sie unter Befehlsübersicht.

Layout

Beim Erstellen einer UI werden die Steuerelemente durch Position und Größe zu einem Layout angeordnet. Eine der wichtigsten Anforderungen eines Layouts ist die Fähigkeit, sich an Änderungen von Fenstergröße und Anzeigeeinstellungen anzupassen. WPF bietet ein erstklassiges erweiterbares Layoutsystem, sodass kein zusätzlicher Code zum Anpassen des Layouts in diesen Fällen geschrieben werden muss.

Das Layoutsystem basiert auf relativer Positionierung, wodurch die Fähigkeit zur Anpassung an geänderte Fenster- und Anzeigebedingungen verbessert wird. Das Layoutsystem verwaltet außerdem die Aushandlung zwischen Steuerelementen zur Festlegung des Layouts. Die Aushandlung findet in zwei Schritten statt: Zuerst teilt ein Steuerelement dem übergeordneten Element mit, welche Position und Größe es benötigt. Anschließend teilt das übergeordnete Element dem Steuerelement mit, welcher Raum zur Verfügung steht.

Das Layoutsystem wird für die untergeordneten Steuerelemente mittels WPF-Basisklassen verfügbar gemacht. Für allgemeine Layouts wie Raster, Stapel und Andockfunktionen enthält WPF mehrere Layout-Steuerelemente:

- Canvas: Untergeordnete Steuerelemente stellen ihr eigenes Layout bereit.
- DockPanel: Untergeordnete Steuerelemente werden an den Rändern des Bereichs ausgerichtet.
- Grid: Untergeordnete Steuerelemente werden anhand von Zeilen und Spalten positioniert.
- StackPanel: Untergeordnete Steuerelemente werden entweder vertikal oder horizontal gestapelt.
- VirtualizingStackPanel: Untergeordnete Steuerelemente werden virtualisiert und auf einer einzelnen Linie angeordnet, die horizontal oder vertikal verläuft.
- WrapPanel: Untergeordnete Steuerelemente werden der Reihenfolge nach von links nach rechts angeordnet. Wenn sich in der jeweiligen Zeile mehr Steuerelemente befinden als der Raum zulässt, wird ein Zeilenumbruch durchgeführt.

Im folgenden Beispiel wird ein DockPanel verwendet, um ein Layout aus mehreren TextBox-Steuerelementen zu erstellen.

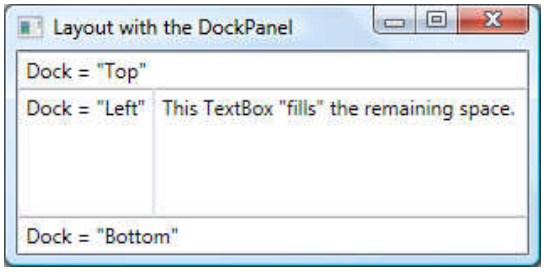
```
<Window
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    x:Class="SDKSample.LayoutWindow"
    Title="Layout with the DockPanel" Height="143" Width="319">

  <!--DockPanel to layout four text boxes-->
  <DockPanel>
    <TextBox DockPanel.Dock="Top">Dock = "Top"</TextBox>
    <TextBox DockPanel.Dock="Bottom">Dock = "Bottom"</TextBox>
    <TextBox DockPanel.Dock="Left">Dock = "Left"</TextBox>
    <TextBox Background="White">This TextBox "fills" the remaining space.</TextBox>
  </DockPanel>

</Window>
```

DockPanel ermöglicht den untergeordneten TextBox-Steuerelementen, Informationen über deren Anordnung bereitzustellen. Dazu wird von DockPanel eine Dock-Eigenschaft implementiert, die für untergeordnete Steuerelemente verfügbar gemacht wird, damit von diesen eine jeweilige Andockart festgelegt werden kann.

Hinweis

Eine Eigenschaft, die von einem übergeordneten Steuerelement zur Verwendung durch untergeordnete Steuerelemente implementiert wird, ist ein WPF-Konstrukt, das als *angefügte Eigenschaft* bezeichnet wird (siehe Übersicht über angefügte Eigenschaften).

Die folgende Abbildung zeigt das Ergebnis des XAML-Markups im vorangehenden Beispiel.



Weitere Informationen finden Sie unter Layoutsystem. Ein einführendes Beispiel finden Sie im Beispiel für einen WPF-Layoutkatalog.

Datenbindung

Die meisten Anwendungen werden erstellt, um Benutzern Mittel zum Anzeigen und Bearbeiten von Daten bereitzustellen. Bei WPF-Anwendungen wird die Arbeit zum Speichern und Zugreifen auf Daten von Technologien wie Microsoft SQL Server und ADO.NET übernommen. Nach dem Zugriff auf die Daten und dem Laden der Daten in die verwalteten Objekte einer Anwendung beginnt die aufwändige Arbeit für WPF-Anwendungen. Dies schließt im Wesentlichen zwei Dinge ein:

1. Kopieren der Daten aus den verwalteten Objekten in Steuerelemente, wo die Daten angezeigt und bearbeitet werden können.
2. Sicherstellen, dass mithilfe der Steuerelemente vorgenommene Änderungen an den Daten zurück in die verwalteten Objekte kopiert werden.

Um die Entwicklung von Anwendungen zu vereinfachen, bietet WPF ein Datenbindungsmodul zur automatischen Durchführung dieser Schritte. Die Kerneinheit des Datenbindungsmoduls ist die Binding-Klasse, deren Aufgabe es ist, ein Steuerelement (Bindungsziel) an ein Datenobjekt (Bindungsquelle) zu binden. Diese Beziehung wird in der folgenden Abbildung verdeutlicht.

Im folgenden Beispiel wird gezeigt, wie eine TextBox an eine Instanz eines benutzerdefinierten `Person`-Objekts gebunden wird. Die `Person`-Implementierung wird im folgenden Code dargestellt.

C#

```
namespace SDKSample
{
    class Person
    {
        string name = "No Name";

        public string Name
        {
            get { return name; }
            set { name = value; }
        }
    }
}
```

Mit dem folgenden Markup wird TextBox an eine Instanz eines benutzerdefinierten `Person`-Objekts gebunden.

```
<Windowxmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"xmlns:x="http://schemas.microsoft.c

...
```

```
<!-- Bind the TextBox to the data source (TextBox.Text to Person.Name) --><TextBoxName="personNameTextBox"T

...

</Window>
```

C#

```
using System.Windows; // Windownamespace SDKSample
{
    publicpartialclass DataBindingWindow : Window
    {
        public DataBindingWindow()
        {
            InitializeComponent();

            // Create Person data source
            Person person = new Person();

            // Make data source available for bindingthis.DataContext = person;
        }
    }
}
```

In diesem Beispiel wird die `Person`-Klasse in Code-Behind instanziiert und als Datenkontext für `DataBindingWindow` festgelegt. Im Markup ist die Text-Eigenschaft von TextBox an die `Person.Name` -Eigenschaft gebunden (mithilfe der "`{Binding ... }`" XAML -Syntax). Dieser XAML-Code weist WPF an, das TextBox-Steuerelement an das `Person`-Objekt zu binden, das in der DataContext-Eigenschaft des Fensters gespeichert ist.

Das Datenbindungsmodul von WPF bietet zusätzliche Unterstützung wie Validierung, Sortierung, Filterung und Gruppierung. Außerdem werden von der Datenbindung Datenvorlagen zum Erstellen einer benutzerdefinierten UI für gebundene Daten unterstützt, wenn die von den WPF-Standardsteuerelementen angezeigte UI nicht ausreichend ist.

Weitere Informationen finden Sie unter Übersicht über Datenbindung. Ein einführendes Beispiel finden Sie unter Demo für die Datenbindung.

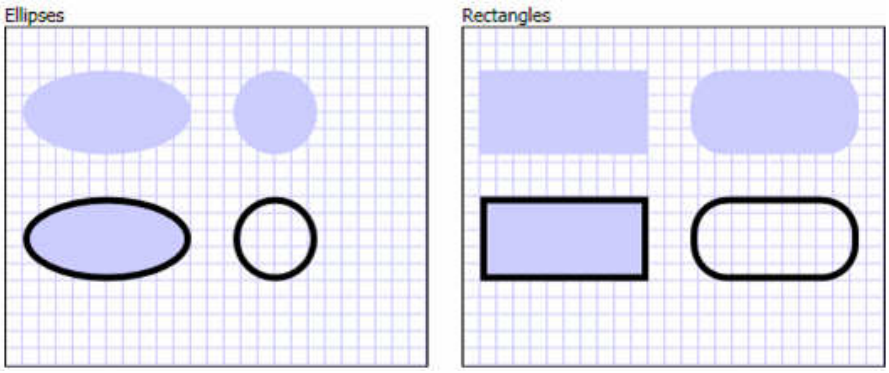
Grafik

Mit WPF wird ein umfangreicher, skalierbarer und flexibler Satz von Grafikfeatures mit den folgenden Vorteilen eingeführt:

- **Auflösungsunabhängige und geräteunabhängige Grafiken**. Die grundlegende Maßeinheit im WPF-Grafiksystem ist das geräteunabhängige Pixel, was 1/96 Zoll entspricht, unabhängig von der jeweiligen Bildschirmauflösung. Sie bildet die Grundlage für ein von Auflösung und Geräten unabhängiges Rendering. Jedes geräteunabhängige Pixel wird automatisch skaliert, um mit der DPI-Einstellung (Dots Per Inch) der Anzeige des jeweiligen Systems übereinzustimmen.
- **Höhere Genauigkeit**. Das WPF-Koordinatensystem wird mit Gleitkommazahlen in doppelter Genauigkeit gemessen. Transformationen und Durchlässigkeitswerte werden ebenfalls mit doppelter Genauigkeit ausgedrückt. WPF unterstützt auch eine breite Farbskala (scRGB) und verfügt über eine integrierte Unterstützung für die Verwaltung von Eingaben aus unterschiedlichen Farbräumen.
- **Erweiterte Unterstützung für Grafiken und Animationen**. WPF vereinfacht die Grafikprogrammierung durch die automatische Verwaltung von Animationsszenen. Szenenverarbeitung, Renderingschleifen und bilineare Interpolation können problemlos angewendet werden. Darüber hinaus unterstützt WPF eine Trefferüberprüfung und bietet vollständige Alpha-Compositing-Unterstützung.
- **Hardwarebeschleunigung**. Das Grafiksystem von WPF schöpft die Grafikhardware aus, um die CPU-Last zu verringern.

2D-Formen

WPF enthält eine Bibliothek allgemeiner vektorbasierter 2-D-Formen wie Rechtecke und Ellipsen, die in der folgenden Abbildung dargestellt sind.



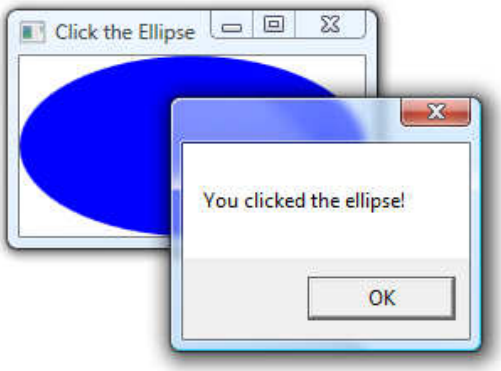
Eine interessante Eigenschaft von Formen ist, dass diese nicht nur der Anzeige dienen, sondern mit ihnen auch viele Features implementiert werden können, die Sie von Steuerelementen erwarten, wie z. B. Tastatur- und Mauseingaben. Im folgenden Beispiel wird die Behandlung des MouseUp-Ereignisses einer Ellipse gezeigt.

```
<Windowxmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"xmlns:x="http://schemas.microsoft.c
```

C#

```
using System.Windows; // Window, MessageBoxusing System.Windows.Input; // MouseButtonEventHandlernamespace
{
    publicpartialclass EllipseEventHandlingWindow : Window
    {
        public EllipseEventHandlingWindow()
        {
            InitializeComponent();
        }

        void clickableEllipse_MouseUp(object sender, MouseButtonEventArgs e)
        {
            // Display a message
            MessageBox.Show("You clicked the ellipse!");
        }
    }
}
```

In der folgenden Abbildung wird das Ergebnis des vorangehenden Codes dargestellt.



Weitere Informationen finden Sie unter Übersicht über Formen und die grundlegenden Funktionen zum Zeichnen in WPF. Ein einführendes Beispiel finden Sie unter Beispiel für Formelemente.

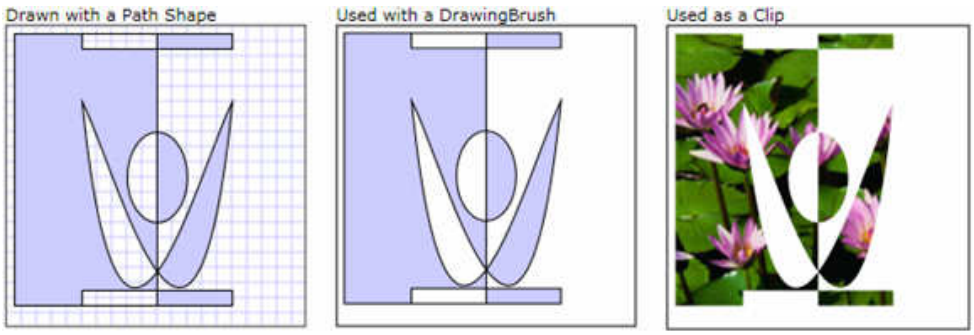
2D-Geometrien

Die von WPF bereitgestellten 2-D-Formen decken den Standardsatz von grundlegenden Formen ab. Möglicherweise müssen jedoch benutzerdefinierte Formen erstellt werden, um die Darstellung einer benutzerdefinierten UI zu optimieren. Für diesen Zweck stellt WPF Geometrien bereit. In der folgenden Abbildung wird die Verwendung von Geometrien zur Erstellung einer benutzerdefinierten Form veranschaulicht. Diese kann direkt gezeichnet, als Pinsel oder zum Ausschneiden anderer Formen und Steuerelemente verwendet werden.

Mithilfe von Path-Objekten können geschlossene oder offene Formen, Mehrfachformen und sogar gekrümmte Formen gezeichnet werden.

Geometry-Objekte können zum Ausschneiden, zur Trefferüberprüfung sowie zum Rendern von 2D-Grafikdaten

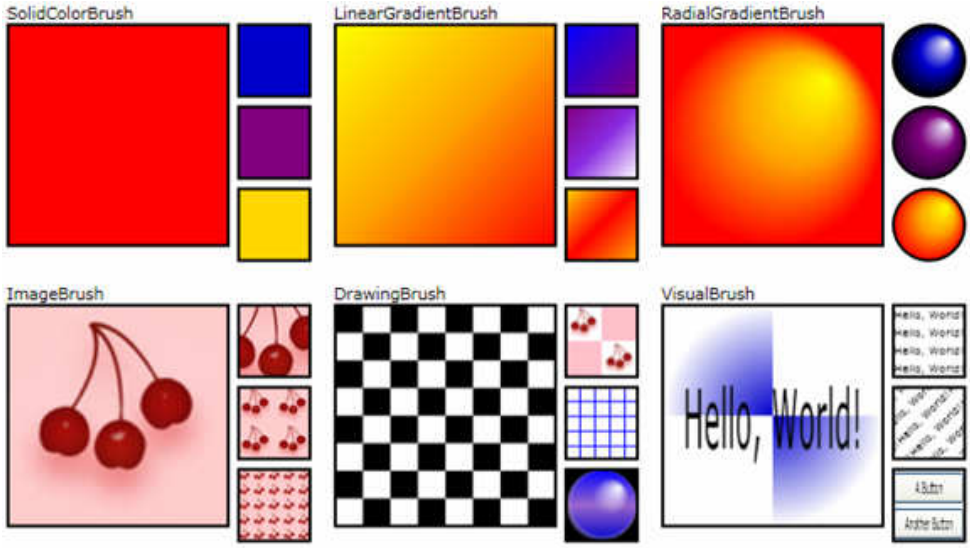
verwendet werden.



Weitere Informationen finden Sie unter Übersicht über die Geometrie. Ein einführendes Beispiel finden Sie unter Beispiele zu Geometrie.

2D-Effekte

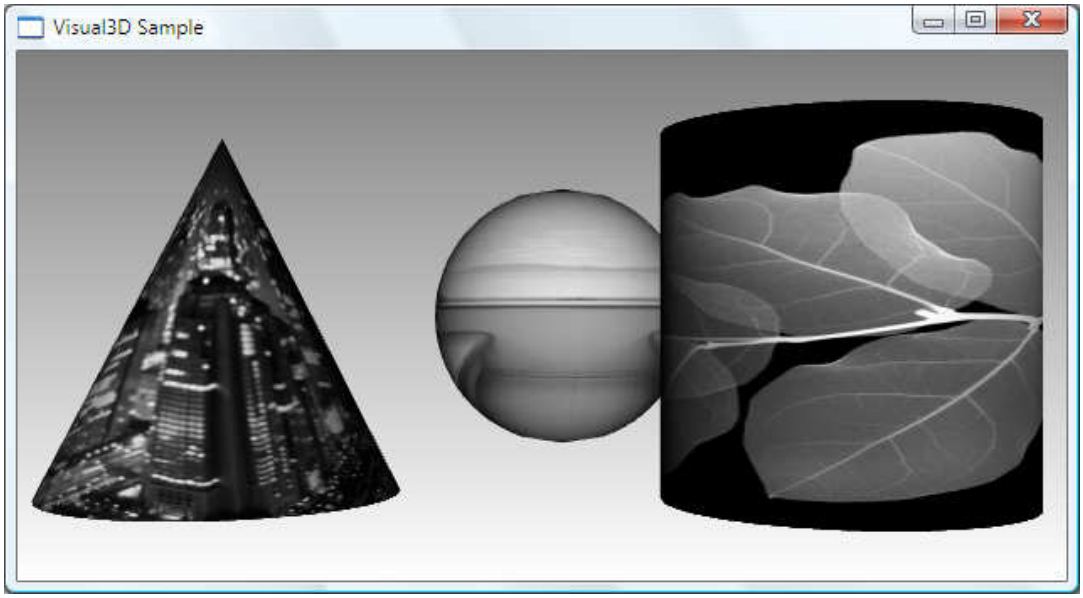
Eine Untermenge der 2-D-Funktionen von WPF enthält visuelle Effekte wie Farbverläufe, Bitmaps, Zeichnungen, Zeichnen mit Videos, Drehung, Skalierung und Neigung. Diese werden mithilfe von Pinseln erzielt. In der folgenden Abbildung werden einige Beispiele gezeigt.



Weitere Informationen finden Sie unter Übersicht über WPF-Pinsel. Ein einführendes Beispiel finden Sie unter Beispiel für Pinsel.

3D-Rendering

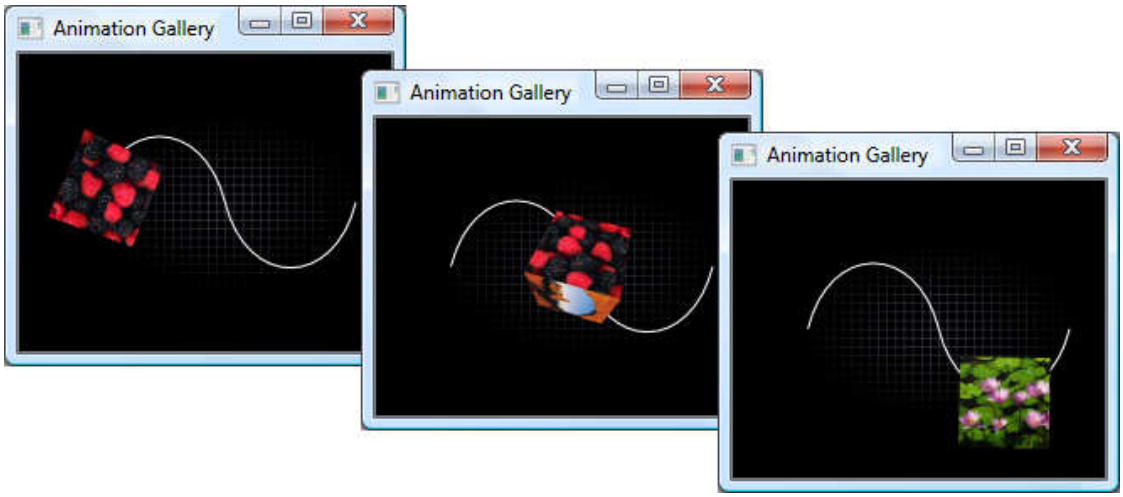
WPF enthält auch 3-D-Renderingfunktionen, die in 2-D-Grafik integriert sind, um noch ansprechendere und interessantere UIs erstellen zu können. Als Beispiel werden in der folgenden Abbildung 2-D-Bilder dargestellt, die auf 3-D-Formen gerendert wurden.



Weitere Informationen finden Sie unter Übersicht über 3D-Grafiken. Ein einführendes Beispiel finden Sie unter Beispiel zu 3D-Festkörpern.

Animation

Durch die Unterstützung von WPF für Animationen können Sie Steuerelemente wachsen, bewegen, drehen sowie ein- und ausblenden lassen, um z. B. interessante Seitenübergänge zu erzeugen. Die meisten WPF-Klassen und sogar benutzerdefinierte Klassen können animiert werden. In der folgenden Abbildung wird eine einfache Animation in Aktion gezeigt.



Weitere Informationen finden Sie unter Übersicht über Animationen. Ein einführendes Beispiel finden Sie unter Beispielsammlung zu Animationen.

Medien

Eine Möglichkeit, Inhalte interessant zu vermitteln, ist die Verwendung audiovisueller Medien. WPF bietet spezielle Unterstützung für Bilder, Video und Audio.

Bilder

Die meisten Anwendungen enthalten Bilder, und WPF bietet mehrere Möglichkeiten ihrer Verwendung. Die folgende Abbildung zeigt eine UI mit einem Listenfeld, das Miniaturbilder enthält. Bei Auswahl einer Miniaturansicht wird das Bild in voller Größe angezeigt.

Weitere Informationen finden Sie unter Übersicht über die Bildverarbeitung.

Video und Audio

Mit dem MediaElement-Steuerelement kann sowohl Video als auch Audio wiedergegeben werden. Es ist flexibel genug, um als Grundlage für einen benutzerdefinierten Media Player verwendet zu werden. Mit dem folgenden XAML-Markup wird ein Media Player implementiert.

```
<MediaElement
  Name="myMediaElement"
  Source="media/wpf.wmv"
  LoadedBehavior="Manual"
  Width="350" Height="250" />
```

Das Fenster in der folgenden Abbildung zeigt das MediaElement-Steuerelement in Aktion.

Weitere Informationen finden Sie unter Grafiken und Multimedia.

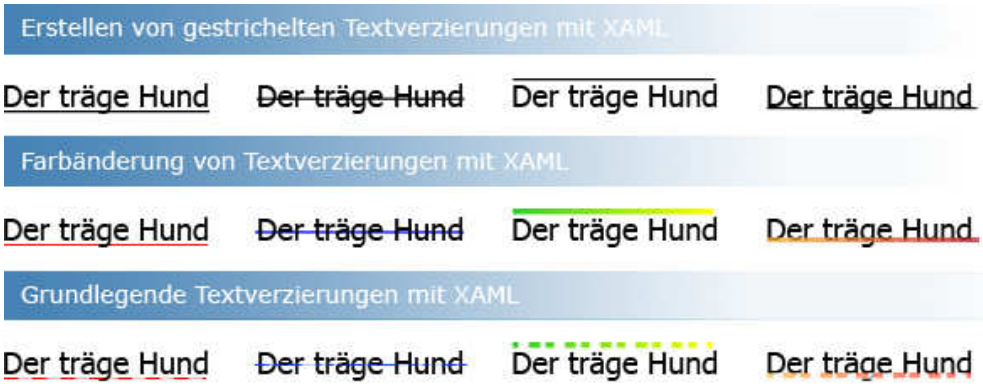
Text und Typografie

Um ein hochwertiges Textrendering zu ermöglichen, bietet WPF die folgenden Features:

- OpenType-Schriftartunterstützung

- ClearType-Optimierungen
- Hohe Leistungsfähigkeit durch Nutzung von Hardwarebeschleunigung
- Einbinden von Text in Medien, Grafiken und Animationen
- Internationale Schriftartunterstützung und Fallbackmechanismen

Zur Demonstration der Texteinbindung in Grafiken wird in der folgenden Abbildung die Anwendung von Textdekorationen veranschaulicht.



Weitere Informationen finden Sie unter Typografie in WPF.

Dokumente

Von WPF werden drei Arten von Dokumenten nativ unterstützt: Flussdokumente, korrigierte Dokumente und XML Paper Specification (XPS)-Dokumente. Darüber hinaus verfügt WPF auch über Dienste zum Erstellen, Anzeigen, Verwalten, Kommentieren, Packen und Drucken von Dokumenten.

Flussdokumente

Bei Flussdokumenten werden Anzeige und Lesbarkeit optimiert, indem Inhalte bei Änderungen der Fenstergröße oder Anzeigeeinstellungen dynamisch angepasst und neu aufgebaut werden. Das folgende XAML-Markup enthält die Definition eines FlowDocument.

```
<FlowDocument xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation">

  <Paragraph FontSize="18" FontWeight="Bold">Flow Document</Paragraph>

  <Paragraph>
    Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy
    nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi
    enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis
    nisl ut aliquip ex ea commodo consequat. Duis autem vel eum iriure.
  </Paragraph>


...


</FlowDocument>
```

Im folgenden Beispiel wird gezeigt, wie ein Flussdokument zum Anzeigen, Durchsuchen und Drucken in einen FlowDocumentReader geladen wird.

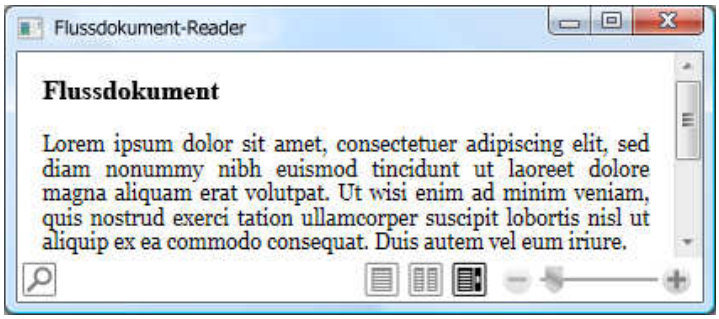
```
<Windowxmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"xmlns:x="http://schemas.microsoft.c
```

C#

```
using System.Windows; // Windowusing System.Windows.Documents; // FlowDocumentusing System.IO; // FileStrea
{
    publicpartialclass FlowDocumentReaderWindow : System.Windows.Window
```

```
    {
        public FlowDocumentReaderWindow()
        {
            InitializeComponent();

            // Open the file that contains the FlowDocumentusing (FileStream xamlFile = new FileStream("AFl
                FileMode.Open, FileAccess.Read))
            {
                // Parse the file with the XamlReader.Load method
                FlowDocument content = XamlReader.Load(xamlFile) as FlowDocument;

                // Set the Document property to the parsed FlowDocument objectthis.flowDocumentReader.Docum
            }
        }
    }
}
```

Im folgenden Beispiel wird das Ergebnis dargestellt.



Weitere Informationen finden Sie unter Übersicht über Flussdokumente.

Einheitlich dargestellte Dokumente

Einheitlich dargestellte Dokumente sind für Anwendungen geeignet, bei denen eine präzise WYSIWYG-Präsentation ("What you see is what you get") erforderlich ist, besonders im Hinblick auf das Ausdrucken. Zu den typischen Verwendungsformen von einheitlich dargestellten Dokumenten zählen Desktopveröffentlichung, Wortverarbeitung und Formularlayout, wo die Beibehaltung der ursprünglichen Seitendarstellung wichtig ist.

In einheitlich dargestellten Dokumenten wird die genaue Anordnung der Inhalte unabhängig vom jeweiligen Gerät aufrechterhalten. Beispielsweise wird ein einheitlich dargestelltes Dokument, das mit 96 DPI (Dots Per Inch) auf einem Bildschirm angezeigt wird, auf dieselbe Weise dargestellt, wenn es auf einem Laserdrucker mit 600 DPI oder auf einem Fotodrucker mit 4800 DPI gedruckt wird. Das Layout bleibt in allen Fällen gleich, auch wenn die Qualität des Dokuments je nach Eigenschaften des jeweiligen Geräts variiert.

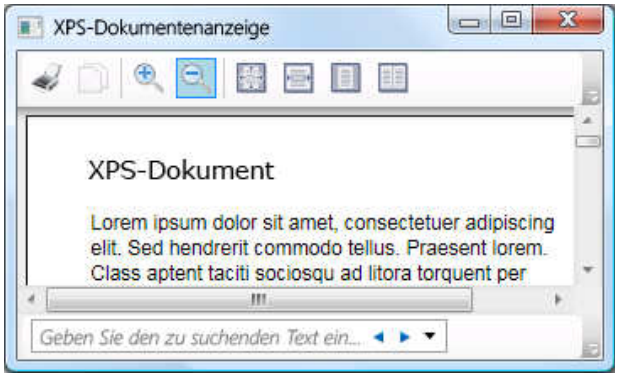
Weitere Informationen finden Sie unter Dokumente in WPF.

XPS-Dokumente

XML Paper Specification (XPS)-Dokumente basieren auf korrigierten Dokumenten von WPF. XPS-Dokumente werden mit einem XML-basierten Schema beschrieben, bei dem es sich im Grunde um eine mehrseitige Darstellung eines elektronischen Papiers handelt. XPS ist ein offenes, plattformübergreifendes Dokumentformat zur Vereinfachung der Erstellung, der Freigabe, des Drucks und der Archivierung von mehrseitigen Dokumenten. Die XPS-Technologie bietet u. a. folgende wichtige Features:

- Packen von XPS-Dokumenten als ZipPackage-Dateien, die den Open Packaging Conventions (OPC) entsprechen
- Das Hosten in eigenständigen und browserbasierten Anwendungen
- Das manuelle Erstellen und Bearbeiten von XPS-Dokumenten in WPF-Anwendungen
- Hochwertiges Rendering durch Erzielen der maximalen Qualität des Ausgabegeräts
- Windows Vista-Druckwarteschlangen
- Direktes Routing von Dokumenten zu XPS-kompatiblen Druckern
- UI-Einbindung in DocumentViewer

Die folgende Abbildung zeigt ein XPS-Dokument, das von einem DocumentViewer angezeigt wird.



Mithilfe von DocumentViewer können Benutzer XPS-Dokumente auch durchsuchen, drucken und deren Ansicht ändern.

Weitere Informationen finden Sie unter Dokumente in WPF.

Anmerkungen

Anmerkungen sind Notizen oder Kommentare, die Dokumenten zur Information oder Kennzeichnung hinzugefügt werden. Im Gegensatz zum Schreiben von Notizen auf gedruckte Dokumente ist das "Schreiben" von Notizen auf elektronische Dokumente oft eingeschränkt oder nicht möglich. Dagegen ist in WPF ein System für Anmerkungen zur Unterstützung von Kurznotizen und Kennzeichnungen enthalten. Diese Anmerkungen können außerdem auf im DocumentViewer-Steuerelement gehostete Dokumente angewendet werden, wie in der folgenden Abbildung veranschaulicht wird.

Weitere Informationen finden Sie unter Übersicht über Anmerkungen.

Verpacken

Mit WPF System.IO.Packaging APIs können Daten, Inhalte und Ressourcen Ihrer Anwendungen in einzelnen, portablen, leicht verteilbaren und leicht zugreifbaren ZIP-Dokumenten organisiert werden. Digitale Signaturen können eingebunden werden, um in einem Paket enthaltene Elemente zu authentifizieren und sicherzustellen, dass diese Elemente nicht manipuliert oder bearbeitet wurden. Sie können Pakete mithilfe der Rechteverwaltung auch verschlüsseln, um den Zugriff auf geschützte Informationen einzuschränken.

Weitere Informationen finden Sie unter Dokumente in WPF.

Drucken

.NET Framework umfasst ein Subsystem für das Drucken, das von WPF durch die Unterstützung für eine verbesserte Drucksystemsteuerung ergänzt wird. Folgende Verbesserungen gibt es im Druckbereich:

- Echtzeitinstallation von Remote-Druckservern und Warteschlangen
- Dynamische Erkennung von Druckerfunktionen
- Dynamische Einstellung von Druckeroptionen
- Wiederholtes Routing und Festlegen der Priorität von Druckaufträgen

XPS-Dokumente verfügen außerdem über eine grundlegende Verbesserung bezüglich der Leistung. Der vorhandene Microsoft Windows Graphics Device Interface (GDI)-Druckpfad erfordert üblicherweise zwei Konvertierungen:

- Zuerst wird das Dokument in ein Druckprozessorformat wie Enhanced Metafile (EMF) konvertiert.

- Anschließend wird eine Konvertierung in die Seitenbeschreibungssprache des Druckers durchgeführt, wie z. B. PCL (Printer Control Language) oder PostScript.

Bei XPS-Dokumenten werden diese Konvertierungen umgangen, da es sich bei einer Komponente des XPS-Dateiformats um eine Druckprozessorsprache und um eine Seitenbeschreibungssprache handelt. Dadurch werden die Größe von Spooldateien und die Auslastung von Netzwerkdruckern verringert.

Weitere Informationen finden Sie unter Übersicht über das Drucken.

Anpassen von WPF-Anwendungen

Sie haben bis jetzt die Bausteine von WPF zur Entwicklung von Anwendungen kennen gelernt. Das Anwendungsmodell wird zum Hosten und Bereitstellen von Anwendungsinhalten verwendet, die hauptsächlich aus Steuerelementen bestehen. Das WPF-Layoutsystem wird verwendet, um die Anordnung von Steuerelementen in einer UI zu vereinfachen und sicherzustellen, dass die Anordnung bei Änderungen von Fenstergröße und Anzeigeeinstellungen erhalten bleibt. Da die meisten Anwendungen die Interaktion von Benutzern mit Daten ermöglichen, wird die Datenbindung verwendet, um den Arbeitsaufwand für das Einbinden der Daten in die UI zu reduzieren. Zur Optimierung der visuellen Darstellung Ihrer Anwendung verwenden Sie das umfangreiche Funktionsspektrum für Grafik, Animation und Medien von WPF. Wenn Ihre Anwendung mit Text und Dokumenten umgeht, können Sie die WPF-Funktionen für Text, Typografie, Dokumente, Anmerkungen, Verpackung und Druck verwenden.

Oft reichen die Grundlagen jedoch nicht aus, um ein wirklich herausragendes und visuell eindrucksvolles Benutzererlebnis zu kreieren. Die Standardsteuerelemente von WPF passen möglicherweise nicht zum gewünschten Erscheinungsbild Ihrer Anwendung. Daten können vielleicht nicht auf die bestmögliche Art angezeigt werden. Der Gesamteindruck Ihrer Anwendung passt eventuell nicht zum Standardaussehen und Verhalten der Windows-Designs. Präsentationstechnologien erfordern auf viele Arten Erweiterbarkeit, so auch beim visuellen Aspekt.

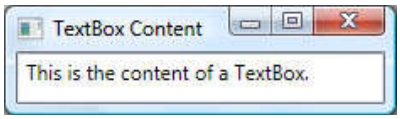
Aus diesem Grund bietet WPF eine Vielzahl von Mechanismen zum Erzeugen einzigartiger Benutzererlebnisse, wie z. B. ein umfangreiches Inhaltsmodell für Steuerelemente, Trigger, Steuerelement- und Datenvorlagen, Stile, UI-Ressourcen, Designs und Skins.

Inhaltsmodell

Die meisten WPF-Steuerelemente haben hauptsächlich die Aufgabe, Inhalte anzuzeigen. In WPF werden Typ und Anzahl der Elemente, aus denen sich der Inhalt eines Steuerelements zusammensetzt, als *Inhaltsmodell* des Steuerelements bezeichnet. Einige Steuerelemente können ein einzelnes Element und einen einzelnen Inhaltstyp enthalten. Beispielsweise ist der Inhalt eines TextBox-Steuerelements ein Zeichenfolgenwert, der der Text-Eigenschaft zugewiesen ist. Im folgenden Beispiel wird der Inhalt eines TextBox-Steuerelements festgelegt.

```
<Window
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    x:Class="SDKSample.TextBoxContentWindow"
    Title="TextBox Content">

...

<TextBox Text="This is the content of a TextBox." />

...

</Window>
```

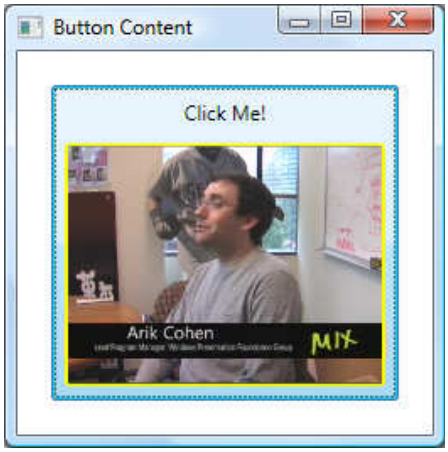
In der folgenden Abbildung wird das Ergebnis dargestellt.



Andere Steuerelemente können jedoch mehrere Elemente verschiedener Inhaltstypen enthalten. Der Inhalt einer Button, der durch die Content-Eigenschaft festgelegt ist, kann beispielsweise aus einer Vielzahl von Steuerelementen wie Layout-Steuerelementen, Text, Bildern und Formen bestehen. Das folgende Beispiel zeigt ein Button-Steuerelement mit Inhalten, zu denen DockPanel, Label, Border und MediaElement zählen.

```
<Window
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    x:Class="SDKSample.ButtonContentWindow"
    Title="Button Content">


...


<Button Margin="20">
  <!-- Button Content -->
  <DockPanel Width="200" Height="180">
    <Label DockPanel.Dock="Top" HorizontalAlignment="Center">Click Me!</Label>
    <Border Background="Black" BorderBrush="Yellow" BorderThickness="2"
      CornerRadius="2" Margin="5">
      <MediaElement Source="media/wpf.wmv" Stretch="Fill" />
    </Border>
  </DockPanel>
</Button>


...


</Window>
```

In der folgenden Abbildung wird der Inhalt dieser Schaltfläche dargestellt.



Weitere Informationen zu den Inhaltstypen, die von den verschiedenen Steuerelementen unterstützt werden, finden Sie unter WPF-Inhaltsmodell.

Trigger

Obwohl die Hauptaufgabe von XAML-Markup in der Implementierung der Darstellung einer Anwendung besteht, lassen sich mit XAML auch einige Aspekte des Verhaltens einer Anwendung implementieren. Ein Beispiel ist die Verwendung von Triggern, um die Darstellung einer Anwendung aufgrund von Benutzerinteraktionen zu ändern. Weitere Informationen finden Sie unter "Trigger" in Erstellen von Formaten und Vorlagen.

Steuerelementvorlagen

Die standardmäßigen UIs für WPF-Steuerelemente werden üblicherweise mithilfe anderer Steuerelemente und Formen erstellt. Beispielsweise besteht ein Button-Steuerelement aus den Steuerelementen ButtonChrome und ContentPresenter. Vom ButtonChrome-Steuerelement wird die Standarddarstellung der Schaltfläche bereitgestellt, während mit dem ContentPresenter-Steuerelement der Inhalt der Schaltfläche angezeigt wird, der durch die Content-Eigenschaft angegeben wird.

Nicht immer passt die Standarddarstellung eines Steuerelements zur Gesamtdarstellung einer Anwendung. In diesem Fall können Sie ControlTemplate verwenden, um die Darstellung der UI des Steuerelements anzupassen, ohne Inhalte und Verhalten zu ändern.

Im folgenden Beispiel wird gezeigt, wie die Darstellung eines Button-Steuerelements mithilfe von ControlTemplate

geändert wird.

```
<Windowxmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"xmlns:x="http://schemas.microsoft.c
```

C#

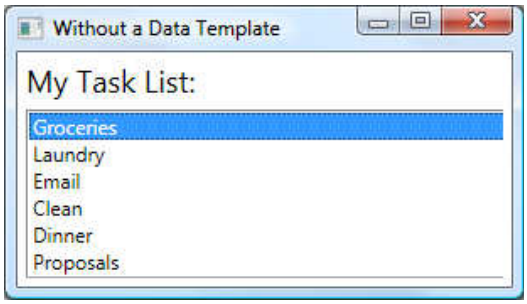
```
using System.Windows; // Window, RoutedEventArgs, MessageBoxnamespace SDKSample
{
    publicpartialclass ControlTemplateButtonWindow : Window
    {
        public ControlTemplateButtonWindow()
        {
            InitializeComponent();
        }

        void button_Click(object sender, RoutedEventArgs e)
        {
            // Show message box when button is clicked
            MessageBox.Show("Hello, Windows Presentation Foundation!");
        }
    }
}
```

In diesem Beispiel wurde UI der Standardschaltfläche durch ein Ellipse-Steuerelement mit dunkelblauem Rand ersetzt und mit einem RadialGradientBrush gefüllt. Vom ContentPresenter-Steuerelement wird der Inhalt des Button-Steuerelements ("Click Me!") angezeigt. Beim Klicken auf das Button-Steuerelement wird das Click-Ereignis als Teil des Standardverhaltens des Button-Steuerelements weiterhin ausgelöst. Das Ergebnis wird in der folgenden Abbildung dargestellt.

Weitere Informationen finden Sie unter ControlTemplate. Ein einführendes Beispiel finden Sie unter Beispiel zum Formatieren mit ControlTemplates.

Datenvorlagen

Während mit einer Steuerelementvorlage die Darstellung eines Steuerelements festgelegt wird, kann mit einer Datenvorlage die Darstellung des Inhalts eines Steuerelements festgelegt werden. Datenvorlagen werden häufig verwendet, um die Anzeige gebundener Daten zu verbessern. Die folgende Abbildung zeigt die Standarddarstellung für ein ListBox-Steuerelement, das an eine Auflistung von `Task`-Objekten gebunden ist, bei der jede Aufgabe über einen Namen, eine Beschreibung und eine Priorität verfügt.

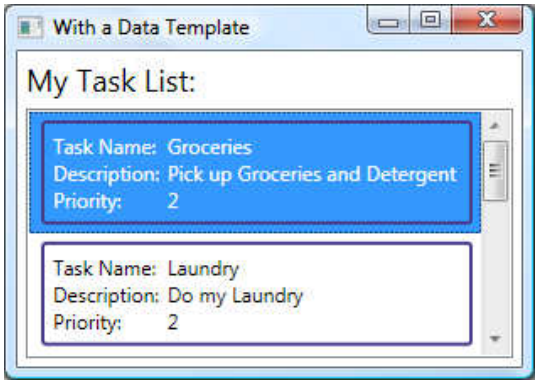


Die Standarddarstellung entspricht dem, was von einem ListBox-Steuerelement zu erwarten ist. Die Standarddarstellung der einzelnen Aufgaben enthält jedoch nur den Aufgabennamen. Um den Aufgabennamen, die Beschreibung und die Priorität anzuzeigen, muss die Standarddarstellung der gebundenen Listenelemente des ListBox-Steuerelements mithilfe eines DataTemplate geändert werden. Mit dem folgenden XAML-Code wird eine

solche DataTemplate-Vorlage definiert, die mithilfe des ItemTemplate-Attributs auf die einzelnen Aufgaben angewendet wird.

```
<Window
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  x:Class="SDKSample.DataTemplateWindow"
  Title="With a Data Template">


...


<Window.Resources>
  <!-- Data Template (applied to each bound task item in the task collection) -->
  <DataTemplate x:Key="myTaskTemplate">
    <Border Name="border" BorderBrush="DarkSlateBlue" BorderThickness="2"
      CornerRadius="2" Padding="5" Margin="5">
      <Grid>
        <Grid.RowDefinitions>
          <RowDefinition/>
          <RowDefinition/>
          <RowDefinition/>
        </Grid.RowDefinitions>
        <Grid.ColumnDefinitions>
          <ColumnDefinition Width="Auto" />
          <ColumnDefinition />
        </Grid.ColumnDefinitions>
        <TextBlock Grid.Row="0" Grid.Column="0" Padding="0,0,5,0" Text="Task Name:"/>
        <TextBlock Grid.Row="0" Grid.Column="1" Text="{Binding Path=TaskName}"/>
        <TextBlock Grid.Row="1" Grid.Column="0" Padding="0,0,5,0" Text="Description:"/>
        <TextBlock Grid.Row="1" Grid.Column="1" Text="{Binding Path=Description}"/>
        <TextBlock Grid.Row="2" Grid.Column="0" Padding="0,0,5,0" Text="Priority:"/>
        <TextBlock Grid.Row="2" Grid.Column="1" Text="{Binding Path=Priority}"/>
      </Grid>
    </Border>
  </DataTemplate>
</Window.Resources>


...


<!-- UI -->
<DockPanel>
  <!-- Title -->
  <Label DockPanel.Dock="Top" FontSize="18" Margin="5" Content="My Task List:"/>

  <!-- Data template is specified by the ItemTemplate attribute -->
  <ListBox
    ItemsSource="{Binding}"
    ItemTemplate="{StaticResource myTaskTemplate}"
    HorizontalContentAlignment="Stretch"
    IsSynchronizedWithCurrentItem="True"
    Margin="5,0,5,5" />

</DockPanel>


...


</Window>
```

Die folgende Abbildung zeigt das Ergebnis dieses Codes.



Beachten Sie, dass das Verhalten und die Gesamtdarstellung des ListBox-Steuerelements beibehalten wurden. Lediglich die Darstellung der vom Listenfeld angezeigten Inhalte wurde geändert.

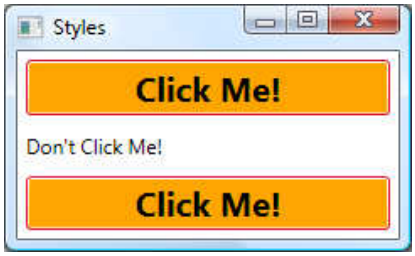
Weitere Informationen finden Sie unter Übersicht über Datenvorlagen. Ein einführendes Beispiel finden Sie unter Einführung in das Beispiel für Datenvorlagen.

Stile

Stile ermöglichen Entwicklern und Designern die Standardisierung auf ein bestimmtes Erscheinungsbild ihres Produkts. Von WPF wird ein solides Formatmodell bereitgestellt, dessen Grundlage das Style-Element bildet. Im folgenden Beispiel wird ein Stil erstellt, mit dem die Hintergrundfarbe für jedes Button-Steuerelement in einem Fenster auf `Orange` festgelegt wird.

```
<Window
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    x:Class="SDKSample.StyleWindow"
    Title="Styles">


...


<!-- Style that will be applied to all buttons -->
<Style TargetType="{x:Type Button}">
  <Setter Property="Background" Value="Orange" />
  <Setter Property="BorderBrush" Value="Crimson" />
  <Setter Property="FontSize" Value="20" />
  <Setter Property="FontWeight" Value="Bold" />
  <Setter Property="Margin" Value="5" />
</Style>


...


<!-- This button will have the style applied to it -->
<Button>Click Me!</Button>

<!-- This label will not have the style applied to it -->
<Label>Don't Click Me!</Label>

<!-- This button will have the style applied to it -->
<Button>Click Me!</Button>


...


</Window>
```

Da sich dieses Format auf alle Button-Steuerelemente bezieht, wird es automatisch auf alle Schaltflächen im Fenster angewendet, wie in der folgenden Abbildung veranschaulicht wird.



Weitere Informationen hierzu finden Sie unter Erstellen von Formaten und Vorlagen. Ein einführendes Beispiel finden Sie unter Einführung zum Beispiel zu Stilen und Vorlagen.

Ressourcen

Die Steuerelemente in einer Anwendung sollten die gleiche Darstellung haben. Dies kann sich beispielsweise auf Schriftarten, Hintergrundfarben, Steuerelementvorlagen, Datenvorlagen und Formate beziehen. Mithilfe der WPF-Unterstützung für user interface (UI)-Ressourcen können diese Ressourcen zur erneuten Verwendung an einem einzigen Speicherort gekapselt werden.

Im folgenden Beispiel wird eine allgemeine Hintergrundfarbe festgelegt, die von Button und Label gemeinsam verwendet wird.

```
<Window
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    x:Class="SDKSample.ResourcesWindow"
    Title="Resources Window">

  <!-- Define window-scoped background color resource -->
  <Window.Resources>
    <SolidColorBrush x:Key="defaultBackground" Color="Red" />
  </Window.Resources>


...


<!-- Button background is defined by window-scoped resource -->
<Button Background="{StaticResource defaultBackground}">One Button</Button>

<!-- Label background is defined by window-scoped resource -->
<Label Background="{StaticResource defaultBackground}">One Label</Label>


...


</Window>
```

In diesem Beispiel wird mithilfe des **Window.Resources**-Eigenschaftenelements eine Ressource für die Hintergrundfarbe implementiert. Diese Ressource ist für alle untergeordneten Elemente von Window verfügbar. Es gibt eine Vielzahl von Ressourcenbereichen, von denen einige nachfolgend in der Reihenfolge aufgeführt sind, in der sie aufgelöst werden:

1. Ein einzelnes Steuerelement (mithilfe der geerbten FrameworkElement.Resources-Eigenschaft).
2. Window oder Page (ebenfalls mithilfe der geerbten FrameworkElement.Resources-Eigenschaft).
3. Application (mithilfe der Application.Resources-Eigenschaft).

Durch die Vielzahl an Bereichen erhalten Sie Flexibilität in Bezug auf die Art, mit der Sie die Ressourcen definieren und freigeben.

Anstatt die Ressourcen direkt mit einem bestimmten Bereich zu verknüpfen, können Sie eine oder mehrere Ressourcen mithilfe eines separaten ResourceDictionary verpacken, auf das in anderen Teilen einer Anwendung verwiesen werden kann. Im folgenden Beispiel wird eine Standardhintergrundfarbe in einem Ressourcenwörterbuch definiert.

```
<ResourceDictionary
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">

  <!-- Define background color resource -->
  <SolidColorBrush x:Key="defaultBackground" Color="Red" />

  <!-- Define other resources -->


...


</ResourceDictionary>
```

Im folgenden Beispiel wir auf das Ressourcenwörterbuch verwiesen, das im vorherigen Beispiel definiert wurde, sodass es innerhalb einer Anwendung gemeinsam verwendet wird.

```
<Application
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    x:Class="SDKSample.App">

  <Application.Resources>
    <ResourceDictionary>
      <ResourceDictionary.MergedDictionaries>
        <ResourceDictionary Source="BackgroundColorResources.xaml"/>
      </ResourceDictionary.MergedDictionaries>
    </ResourceDictionary>
  </Application.Resources>


...


</Application>
```

Ressourcen und Ressourcenwörterbücher bilden die Grundlage der WPF-Unterstützung für Designs und Skins.

Weitere Informationen finden Sie unter Übersicht über Ressourcen.

Designs und Skins

Unter visuellem Aspekt wird mit einem Design die globale Darstellung von Windows und den darin ausgeführten Anwendungen definiert. Windows enthält mehrere Designs. Zum Beispiel bietet Microsoft Windows XP die Designs "Windows XP" und "Windows – klassisch", während Windows Vista die Designs "Windows Vista" und "Windows – klassisch" enthält. Mit der Darstellung, die durch ein Design definiert wird, wird die Standarddarstellung für eine WPF-Anwendung festgelegt. WPF ist jedoch nicht direkt in die Windows-Designs eingebunden. Da die Darstellung von WPF durch Vorlagen definiert wird, enthält WPF eine Vorlage für jedes der bekannten Windows-Designs, einschließlich Aero (Windows Vista), Classic (Microsoft Windows 2000), Luna (Microsoft Windows XP) und Royale (Microsoft Windows XP Media Center Edition 2005). Die visuelle Darstellung vieler Anwendungen basiert auf diesen Designs. Durch die Einheitlichkeit mit der Windows-Darstellung gewöhnen sich Benutzer leichter an neue Anwendungen.

Auf der anderen Seite basiert das Benutzererlebnis bei einigen Anwendungen nicht unbedingt auf den Standarddesigns. Beispielsweise basiert Microsoft Windows Media Player auf Audio- und Videodaten betrieben und gewinnt durch einen abweichenden Stil des Benutzererlebnisses. Bei solchen UIs-Anwendungen werden eher benutzerdefinierte, anwendungsspezifische Designs verwendet. Anwendungen mit solchen Skins bieten dem Benutzer oft die Möglichkeit, verschiedene Aspekte der Skin anzupassen. Microsoft Windows Media Player bietet voreingestellte Designs sowie einen Host für Skins von Drittanbietern.

Die Designs und Skins in WPF werden am einfachsten mit Ressourcenwörterbüchern definiert. Im folgenden Beispiel werden Beispieldefinitionen für Skins gezeigt.

```
<!-- Blue Skin --><ResourceDictionaryxmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"xmlns

...

</ResourceDictionary>

<!-- Yellow Skin --><ResourceDictionaryxmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"xml

...

</ResourceDictionary>
```

Weitere Informationen finden Sie in "Gemeinsam genutzte Ressourcen und Designs" unter Erstellen von Formaten und Vorlagen.

Benutzerdefinierte Steuerelemente

Obwohl WPF umfangreiche Unterstützung bei Anpassungen bietet, kann es passieren, dass die vorhandenen WPF-Steuerelemente nicht den Anforderungen der Anwendung oder der Benutzer genügen. Dies kann in folgenden Situationen der Fall sein:

- Die gewünschte UI kann nicht durch Anpassung des Aussehens und Verhaltens vorhandener WPF-Implementierungen erzeugt werden.
- Das gewünschte Verhalten wird von vorhandenen WPF-Implementierungen nicht unterstützt (oder ist umständlich zu erreichen).

An diesem Punkt können Sie jedoch eines der drei WPF-Modelle zum Erstellen eines neuen Steuerelements nutzen. Jedes Modell bezieht sich auf ein bestimmtes Szenario. Dabei muss das benutzerdefinierte Steuerelement von einer bestimmten WPF-Basisklasse abgeleitet werden. Die drei Modelle sind hier aufgeführt:

- **Benutzersteuerelementmodell**. Ein benutzerdefiniertes Steuerelement wird von UserControl abgeleitet und besteht aus einem oder mehreren anderen Steuerelementen.
- **Steuerelementmodell**. Ein benutzerdefiniertes Steuerelement wird von Control abgeleitet und dazu verwendet, Implementierungen zu erstellen, deren Verhalten mittels Vorlagen von ihrer Darstellung getrennt wird, so wie beim Großteil der WPF-Steuerelemente. Durch das Ableiten von Control erhalten Sie beim Erstellen einer benutzerdefinierten UI mehr Freiheit als mit Benutzersteuerelementen. Der Aufwand kann jedoch höher sein.
- **Frameworkelementmodell**. Ein benutzerdefiniertes Steuerelement wird von FrameworkElement abgeleitet, wenn seine Darstellung durch benutzerdefinierte Renderinglogik (nicht durch Vorlagen) definiert wird.

Im folgenden Beispiel wird ein benutzerdefiniertes numerisches "nach oben/nach unten"-Steuerelement gezeigt, das von UserControl abgeleitet wird.

```
<UserControlxmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"xmlns:x="http://schemas.micros
```

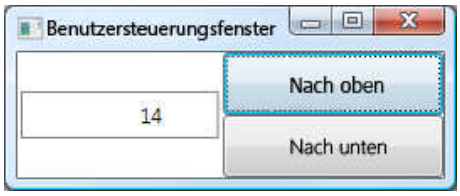
C#

```
using System; // EventArgsusing System.Windows; // DependencyObject, DependencyPropertyChangedEventArgs,// 
{
    publicpartialclass NumericUpDown : UserControl
    {
        // NumericUpDown user control implementation

...

    }
}
```

Im nächsten Beispiel wird der XAML-Code dargestellt, mit dem das Benutzersteuerelement in Window integriert wird.

```
<Window
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    x:Class="SDKSample.UserControlWindow"
    xmlns:local="clr-namespace:SDKSample"
    Title="User Control Window">


...


<!-- Numeric Up/Down user control -->
<local:NumericUpDown />


...


</Window>
```

Die folgende Abbildung zeigt das `NumericUpDown`-Steuerelement, das in einem Window gehostet wird.



Weitere Informationen zu benutzerdefinierten Steuerelementen finden Sie unter Übersicht über das Erstellen von Steuerelementen.

Bewährte Methoden für WPF

Wie bei allen Entwicklungsplattformen kann mit WPF das gewünschte Ergebnis auf verschiedene Arten erreicht werden. Um sichergehen zu können, dass Ihre WPF-Anwendungen das angestrebte Benutzererlebnis liefern und die Ansprüche der Zielgruppe allgemein erfüllt werden, gibt es empfohlene bewährte Methoden bezüglich Barrierefreiheit, Globalisierung, Lokalisierung und Leistung. Nachfolgend erhalten Sie weitere Informationen:

- Bewährte Methoden für Eingabehilfen
- Übersicht über WPF-Globalisierung und -Lokalisierung
- Optimieren der WPF-Anwendungsleistung
- Sicherheit (WPF)

Zusammenfassung

WPF ist eine umfangreiche Präsentationstechnologie zum Erstellen eines breiten Spektrums von visuell herausragenden Clientanwendungen. Diese Einführung gab einen Einblick in die wichtigsten Features von WPF.

Der nächste Schritt ist, WPF-Anwendungen zu erstellen!

Beim Erstellen dieser Anwendungen können Sie auf diese Einführung zurückgreifen, um sich einen Überblick über die wichtigsten Features zu verschaffen sowie Verweise auf ausführlichere Informationen zu diesen Features zu erhalten.

Empfohlene Übersichten und Beispiele

Die folgenden Übersichten und Beispiele wurden in dieser Einführung erwähnt.

Übersichten

Bewährte Methoden für Eingabehilfen	Übersicht über die Eingabe
Übersicht über die Anwendungsverwaltung	Layoutsystem

Befehlsübersicht	Übersicht über die Navigation
Steuerelemente	Übersicht über das Drucken
Übersicht über Datenbindung	Übersicht über Ressourcen
Übersicht über Abhängigkeitseigenschaften	Übersicht über Routingereignisse
Dokumente in WPF	Erstellen von Formaten und Vorlagen
	Typografie in WPF
	Sicherheit (WPF)
	Übersicht über WPF-Fenster
	WPF-Inhaltsmodell
	Übersicht über WPF-Globalisierung und -Lokalisierung
	Grafiken und Multimedia

Beispiele

Beispiel zu 3D-Festkörpern	Einführung in das Beispiel für Datenvorlagen
Beispielsammlung zu Animationen	Einführung zum Beispiel zu Stilen und Vorlagen
Beispiel für Pinsel	Beispiel für Formelemente
Demo für die Datenbindung	Beispiel zum Formatieren mit ControlTemplates
Beispiele zu Geometrie	Beispiel für einen WPF-Layoutkatalog

Siehe auch

Konzepte

Exemplarische Vorgehensweise: Erste Schritte mit WPF
WPF-Communityfeedback

Community-Inhalt