



Norbert
Eder

Windows Presentation Foundation 4

Einführung und Praxis



inklusive CD-ROM

Grundlagen der Windows Presentation Foundation

Die Windows Presentation Foundation (zukünftig der Einfachheit halber »WPF« genannt) existiert nun schon einige Jahre, ist aber bei einem Großteil der Entwickler noch immer nicht angekommen. Dies betrifft nicht nur die Projektumsetzungshäufigkeit, bei der WPF zum Zuge kommt, sondern auch grundlegendes Wissen über die WPF. Viele Projekte werden nach wie vor mittels Windows Forms umgesetzt, da diesbezüglich Wissen vorhanden und die WPF ein nicht einschätzbarer Faktor ist. Dieses Kapitel möchte sich daher der WPF mitsamt ihrer Prinzipien und grundlegenden Eigenschaften widmen, um die Technologie aus einem »schwarzen Loch« ins Licht zu hieven. Im ersten Teil werden allgemeine Grundlagen rund um diese Technologie vermittelt. Dies beinhaltet nicht nur Vorteile, Nachteile und Nutzen, sondern auch Werkzeuge, die zum Einsatz kommen bzw. kommen können. Der zweite Teil dieses Kapitels beschäftigt sich mit einer Einführung in die unterschiedlichen Anwendungs- und Projekttypen und gibt Ihnen einen Überblick über den Aufbau und die Komponenten von WPF-Anwendungen.

1.1 Einführung

Die WPF stellt eine erhebliche Verbesserung zur bis dato bekannten Entwicklung von Benutzerschnittstellen dar. Bisher wurde (zumindest im .NET-Umfeld) je nach Anforderung auf Windows-Forms- oder ASP.NET-Anwendungen gesetzt. Andere Möglichkeiten haben sich nicht aufgetan. Die Nachteile der Windows-Forms-Anwendungen liegen jedoch förmlich auf der Hand:

- Eine Trennung zwischen Darstellung und Logik ist schwierig und wird kaum ernsthaft durchgesetzt. Die Hauptschwierigkeit liegt darin, dass die Oberfläche vom Entwickler selbst gestaltet werden muss und entsprechend oft jede Menge Logik innerhalb der Formulare zu finden ist, da eine Abgrenzung der Aufgaben nicht stattfindet und daher auch nicht umgesetzt wird. Eine echte Abgrenzung von Darstellung und Logik ist in der Praxis zudem kaum durchführbar.
- Effekte von modernen Oberflächen sind nur mit erhöhtem Aufwand möglich. Windows Forms basiert auf GDI+ und unterliegt daher gewissen Einschränkungen. GDI+ ist eine C++-Schnittstelle, welche die Kommunikation mit der

Hardware für uns unternimmt und entsprechende Funktionalitäten zur Verfügung stellt. Jedoch werden nur allgemeine Fähigkeiten der Hardware unterstützt. Da die Windows Presentation Foundation mit der DirectX-Schnittstelle kommuniziert, können weitere Features in Anspruch genommen werden, die so via GDI+ nicht zur Verfügung stehen (beispielsweise Hardwarebeschleunigung). Daher lassen sich Oberflächen flexibler gestalten und selbst 3D-Effekte sind mit relativ wenig Aufwand möglich (ebenso lässt sich Hardwarebeschleunigung verwenden). Auch fehlt den Windows Forms die notwendige Skalierbarkeit der User Interfaces, um bei unterschiedlichen Auflösungen Ergebnisse ohne Darstellungsverlust anbieten zu können. Und nicht nur die unterschiedlichen Auflösungen kommen hier ins Spiel: Als Beispiel denke man an lokalisierte Anwendungen (also Anwendungen, die an die Sprachumgebung des Benutzers angepasst sind). Hier kommt es sehr oft vor, dass Texte unterschiedlich lang sind usw. Dies kann uns von der WPF abgenommen werden, da sich die Steuerelemente entsprechend in ihrer Größe anpassen.

- Bedingt durch das Web sind es viele Benutzer gewohnt, sich ihre Oberflächen selbst anzupassen bzw. aus vordefinierten Designs zu wählen. Dies ist mittels Windows Forms zwar machbar, jedoch ist der Aufwand entsprechend hoch. Das bedeutet, dass oftmals eigene Implementierungen von Steuerelementen entwickelt werden müssen, die Skin-fähig sind. Standardelemente bieten diese Möglichkeit vielfach nicht an und können daher nur bedingt für derartige Zwecke verwendet werden. Die WPF bietet von Haus aus freie Hand hinsichtlich des Aussehens von Steuerelementen, wodurch hier hauptsächlich grafisches Talent denn Programmierkönnen gefragt ist.

An diesen Punkten (weitere ließen sich jedenfalls leicht finden) ist durchaus erkennbar, worauf die WPF hauptsächlich abzielt. Die Oberflächen von Anwendungen haben sich in den letzten Jahren geändert. Heute reicht es nicht mehr aus, Funktionalitäten zur Verfügung zu stellen. Es ist wichtig, dass Anwendungen intuitiv sind, Wartezeiten verkürzen (durch Animationen etc.) und grundsätzlich hübsch anzusehen sind. Als Stichwort sei an dieser Stelle die »Usability« zu nennen – ein nicht unwesentlicher Bereich im heutigen Oberflächendesign. Sie ist vor allem bei Anwendungen wichtig, mit denen tagesin, tagaus gearbeitet wird. Hierzu würden zwar die Möglichkeiten, die uns Windows Forms bietet, ausreichen, aber wie sieht es mit einer Lösung aus, die uns Zeit erspart und noch dazu wesentlich flexibler agiert?

Genau an dieser Stelle setzt die WPF unter anderem an und bietet sowohl Entwicklern als auch Designern unterschiedliche Vorteile:

- *eXtensible Application Markup Language*, oder kurz XAML. Dabei handelt es sich um ein erweitertes XML-Format, um grafische Oberflächen deklarativ zu entwickeln. Die deklarative Programmierung beschäftigt sich nicht mit dem »Wie« (imperative Programmierung), sondern mit dem »Was«. Daher wird

über XAML beschrieben, was dargestellt werden soll. Es ist also möglich, mithilfe von XAML Steuerelemente zu platzieren, anzuordnen, das Aussehen zu beschreiben und sogar mit Transformationen zu versehen. Natürlich sind auch weitere Möglichkeiten gegeben (mehr dazu in Kapitel 2).

- *Trennung zwischen Design und Logik.* Durch die Einführung von XAML wurde es erheblich erleichtert, das Design von der Logik zu trennen. So ist es möglich, dem Designer die Gestaltung der Oberfläche zu überlassen, während der Entwickler für die Implementierung der Funktionalität sorgt. Damit können die Aufgaben wesentlich einfacher verteilt und Zuständigkeiten besser zugewiesen werden. So kann sich jeder Teilhabende auf seinen Part konzentrieren. Wird dieses Konzept sauber verfolgt, sind sowohl das User Interface als auch die tatsächliche Implementierung austauschbar. Es findet also tatsächlich eine Trennung zwischen Design und Logik in einer Form statt, die bisher nicht so einfach umzusetzen war.
- *Verbesserte 2D- und 3D-Unterstützung.* Die WPF stellt sehr viele Schnittstellen zur Verfügung, die es erheblich erleichtern, Benutzeroberflächen zu gestalten. Dies schränkt sich nicht auf den 2D-Bereich ein, sondern auch 3D-Oberflächen werden unterstützt. Durch die Anbindung an Direct3D ist man nicht auf die Funktionalität beschränkt, die GDI+ zur Verfügung stellt, sondern kann quasi aus dem Vollen schöpfen (Zusatzfunktionen der Hardware, die über DirectX zur Verfügung gestellt werden). 3D-Effekte, sich drehende Körper, Transformationen, verlustfreies Zoomen und vieles mehr ist möglich, ohne monatelange Implementierungsdauer. Trotz der vereinfachten Darstellung sei jedoch darauf verwiesen, dass kleine Erfolge sehr schnell erzielt werden, dennoch ist einiges an Know-how notwendig, um auch komplexere Aufgabenstellungen in annehmbarer Zeit umsetzen zu können. Der Teufel liegt ja bekanntlich im Detail und dies trifft auch auf die WPF zu.
- *Browseranwendungen.* Es lassen sich nicht nur anspruchsvolle Windows-Anwendungen entwickeln. Die WPF kann auch für die Entwicklung von Browseranwendungen herangezogen werden – dies sowohl unter dem Deckmantel der WPF selbst als auch durch Silverlight. Dafür steht die WPF zwar nicht im vollen Funktionsumfang zur Verfügung, dennoch können damit anspruchsvolle Internetanwendungen (beispielsweise für das Intranet) erstellt werden. Die Herangehensweise unterscheidet sich nicht zur Entwicklung unter Windows, es muss dem Entwickler jedoch bewusst sein, dass es Unterschiede im angebotenen Funktionsumfang gibt.

Wie Sie sehen, sind dies größtenteils Vorteile, die das Entwickeln von Oberflächen betrifft. Es sei jedoch darauf verwiesen, dass sich auch unter der Haube einiges geändert hat, wodurch sich nicht nur die Entwicklung von User Interfaces vereinfachen lässt. Auf die genauen Einzelheiten werden wir in den nachfolgen-

den Kapiteln noch näher zu sprechen kommen. Es sei jedoch bereits jetzt vorweggenommen, dass sich die Vorteile nicht nur auf Grafisches beschränken.

1.1.1 Nutzen der WPF

Eine Aufzählung der Vorteile ist natürlich eine schöne Sache, jedoch erschließt sich dadurch meist nicht auf den ersten Blick, welchen Nutzen etwas tatsächlich bringt und ob sich ein Umstieg wirklich lohnt. Daher widmet sich dieser Abschnitt dem effektiven Nutzen, den die WPF sowohl dem Designer als auch dem Entwickler (und im Endeffekt dem Kunden/Anwender) bietet.

Wenn Sie mit der Entwicklung unter der WPF beginnen, werden Ihnen recht schnell zwei Dinge auffallen:

- Vieles kann um einiges leichter erreicht werden als bisher. Seien es Animationen, 3D-Effekte oder die Ablage des Aussehens in eigenen Dateien (und die damit verbundene einfache Möglichkeit, das Aussehen der gesamten Anwendung innerhalb von Minuten zu ändern). Sehr schnell werden Sie auch erfahren, dass ab sofort Dinge, die Ihnen früher vermutlich zu kompliziert erschienen, nun leicht von der Hand gehen.
- Haben Sie bereits unter Windows Forms Anwendungen implementiert, dann werden Sie Ihre WPF-Anwendungen wahrscheinlich mit den gleichen Gedanken und derselben Herangehensweise starten. Hier werden Sie erfahren, dass sich einiges Bekanntes in Unbekanntes verwandelt hat. So finden sich an vielen Stellen nicht mehr die gewohnten Eigenschaften. Oft muss daher mühsam herausgefunden werden, wie die ehemals einfache Aufgabe jetzt realisiert werden kann. Lassen Sie sich dadurch jedoch nicht entmutigen. Sind die neuen Konzepte erst einmal bekannt, ist auch dieser Weg einfach zu beschreiten. Wichtig beim Umstieg von Windows Forms auf die WPF ist, dass Sie bisher gefestigtes Wissen loslassen und sich frei von alten Gewohnheiten auf diese Technologie stürzen.

Eines dieser neuen Konzepte wird durch die vorhandenen Steuerelemente abgebildet. Bis dato hatte man Standardelemente zur Verfügung, die auf ein Formular gezogen wurden und somit das Aussehen einer Anwendung bestimmten. Für besondere Fälle mussten eigene Steuerelemente implementiert werden, vor allem wenn eigene Funktionalität oder ein bestimmtes Aussehen notwendig waren. Soweit hat sich auch in der WPF nichts verändert. Der Unterschied liegt jedoch – wie immer – im Detail. Unter Windows Forms hatten alle Elemente eine bestimmte Aufgabe und waren entsprechend dieser Aufgabe in ihrer Funktion sehr eingeschränkt. Sollten mehrere Steuerelemente miteinander kombiniert werden, musste meist eine Ableitung her und schon entstand ein neues Benutzerelement. Hier lässt uns die WPF wesentlich mehr Freiraum.

Nehmen wir als Beispiel das `ListBox`-Element aus den Windows Forms. In den meisten Fällen wird dieses Element verwendet, um Daten aufzulisten und dem Benutzer eine Mehrfachauswahl zu bieten. Großartige Gestaltungsmöglichkeiten werden dem Entwickler jedoch nicht geboten. Sollen Einträge beispielsweise in Form einer Visitenkarte dargestellt werden (eventuell zusammen mit einem Bild) würde dies unter Windows Forms bereits einen beträchtlichen Zeit- und Implementierungsaufwand mit sich bringen. Mithilfe der WPF kann über eine Vorlage (mehr dazu in den Kapiteln 6 und 8) definiert werden, wie die Inhalte auszusehen haben und darüber hinaus können beliebige Steuerelemente innerhalb der einzelnen Zeilen (»Items«) platziert werden. So ist es mittels der WPF auch sehr einfach zu bewerkstelligen, weitere Steuerelemente einzubinden, um einzelne Einträge auf unterschiedliche Art und Weise bearbeitbar zu machen. Ein Beispiel in Verbindung mit einem `ComboBox`-Element ist in Abbildung 1.1 zu sehen.

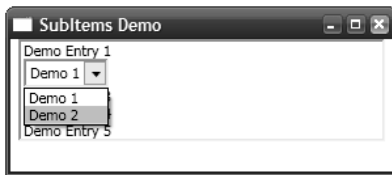


Abb. 1.1: `ComboBox` in einer `ListBox` anzeigen

Dabei handelt es sich um ein Feature, welches einen unheimlichen Vorteil bietet: Es wird erstmals möglich, Controls fast beliebig zu verschachteln, ohne sofort ein neues Steuerelement erstellen zu müssen.

Aber nicht nur gestaltungstechnisch, sondern auch technologisch ändert sich einiges für den Softwareentwickler. Bis dato wurde das Zeichnen der Oberfläche durch die GDI-Funktionen von Windows übernommen. Unter der WPF erfolgt die Darstellung unter Zuhilfenahme von `Direct3D`. Bisher bekannte Einschränkungen sind daher nicht mehr gültig und Anwendungen profitieren zudem von der dadurch möglichen Hardwareunterstützung. Ein weiterer Unterschied liegt in der vektorbasierten Ausgabe. Dies bedeutet, dass die Anwendung bei jeglicher Auflösung immer korrekt (und verlustfrei) dargestellt wird, unabhängig der vom Entwickler bzw. dem Anwender gewählten Konfiguration. Interessanter Nebeneffekt: Verlustfreies Zoomen wird dadurch realisierbar.

1.1.2 Interaktion

Ein weiterer Nutzen, der jedoch gesondert dargestellt werden soll, ist die Möglichkeit der Interaktion mit Bestehendem. So können WPF-Anwendungen nicht nur mit Win32-Anwendungen interagieren, sondern auch:

- WPF-Teile in herkömmliche Windows-Forms-Anwendungen
- und Windows-Forms-Steuerelemente in WPF-Anwendungen

integriert werden.

Dies mag auf den ersten Blick zwar ein eher unbedeutendes Feature sein, jedoch tun sich in der Realität einige Möglichkeiten auf.

Beispielsweise existiert in der WPF-Welt kein Steuerelement, das es mit dem unter Windows Forms bekannten `PropertyGrid` aufnehmen kann. Anstatt diese Funktionalität nachzubilden, ist es mit sehr wenig Aufwand möglich, das bekannte Element in eine WPF-Anwendung zu integrieren. Auf die gleiche Art und Weise kann ein WPF-Steuerelement in eine Windows-Forms-Anwendung übernommen werden. Sehr interessant, wenn es um Transformationen bzw. Animationen geht.

Schlussendlich funktioniert dies auch im Zusammenspiel mit Office, Visual Studio und anderen Produkten.

Die unterschiedlichen Möglichkeiten der Interaktion bieten den Vorteil, dass eine Anwendung nicht komplett unter der WPF entwickelt werden muss. Es können lediglich die Teile ausgegliedert werden, die sich mithilfe der WPF wesentlich einfacher umsetzen lassen. Das Ergebnis wird anschließend in die eigentliche Anwendung (dies kann beispielsweise eine Windows-Forms-Anwendung sein) integriert und steht sofort zur Verfügung. So ist es möglich, für aufwendige Bereiche Zeit (= Aufwand) zu sparen und dennoch größtenteils in einer bekannten Umgebung zu arbeiten.

1.1.3 Werkzeuge

Die beste Technologie bringt (meist) wenig, wenn es keine Unterstützung durch unterschiedliche Hilfsmittel gibt. Da die Windows Presentation Foundation nun doch schon einige Zeit verfügbar ist, kann auf entsprechende Werkzeuge zurückgegriffen werden. Dieser Abschnitt soll einige Möglichkeiten aufzeigen, die Sie in die Lage versetzen, mit der WPF bzw. XAML zu arbeiten. Werkzeuge, die auf spezielle Bedürfnisse zugeschnitten und für bestimmte Aufgabenstellungen vorgesehen sind, werden in den jeweiligen Kapiteln näher aufgeführt. Hier soll lediglich ein kleiner Überblick gegeben werden, der Ihnen als Stütze bei der Suche Ihres Favoriten dienen soll.

Visual Studio 2010/Cider

Das wohl am meisten genannte Werkzeug (und auch für Entwickler am sinnvollsten) ist Visual Studio 2010 zusammen mit dem integrierten XAML-Editor/-Designer Cider. Da der Editor tatsächlich voll integriert ist, verspricht dies eine optimale Zusammenarbeit. Die Notwendigkeit, verschiedene Anwendungen zu nutzen, ist somit nicht gegeben. Fast. Zwar bietet Cider viele Möglichkeiten, den-

noch wird der Entwickler beim Thema Performance etwas enttäuscht. Gerade beim erstmaligen Laden der XAML-Visualisierung vergeht viel Zeit, bis die Oberfläche gerendert wird.

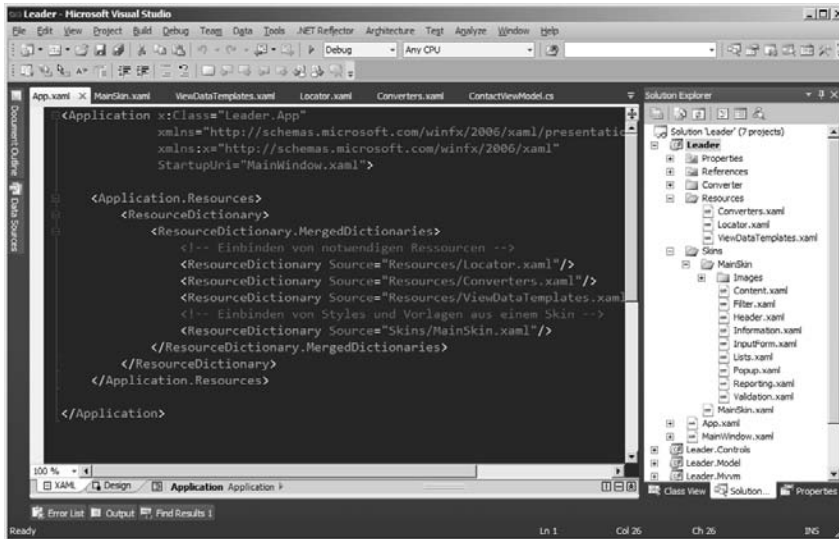


Abb. 1.2: Visual Studio 2010

Hier nun die wichtigsten Eigenschaften und Möglichkeiten von Cider:

- **Rapid Application Development:** Wie bereits von frühen Visual Basic IDEs und natürlich auch Windows Forms bekannt, können Steuerelemente bequem mittels Maus auf ein WPF-Fenster oder Steuerelement gezogen und platziert werden. Der dahinter liegende XAML-Code wird automatisch generiert und muss somit nur mehr angepasst werden.
- **Split View:** Wenn es um die Oberfläche und den XAML-Code geht, stehen dem Entwickler drei unterschiedliche Modi zur Verfügung. In der DESIGN VIEW wird lediglich die fertige Oberfläche angezeigt bzw. besteht die Möglichkeit, Steuerelemente beliebig zu setzen bzw. zu verändern. In der XAML VIEW ist nur der XAML-Code sichtbar, kann erstellt und/oder verändert werden. Schließlich steht noch die SPLIT VIEW zur Verfügung, die beide Ansichten vereint und untereinander darstellt. Veränderungen in einer Ansicht sind automatisch in der zweiten Ansicht sichtbar.
- **IntelliSense:** Wie im normalen Codefenster steht auch für den XAML-Editor IntelliSense zur Verfügung. Schnelles Setzen von Attributen und Werten (beispielsweise Farben, Pfade) wird so möglich und spart jede Menge Zeit und Nerven.

- **Fehlererkennung:** Tippfehler passieren, auch andere Fehler gehören zum Alltag des Entwicklers. Der WPF-Designer macht darauf aufmerksam und zeigt vorhandene Fehler wie gewohnt in der Fehlerliste an. In manchen Fällen gilt es jedoch, einfallsreich zu sein, da die eine oder andere Fehlermeldung dann doch nicht unbedingt das darstellt, was sie eigentlich besagen sollte.

Für den WPF-Entwickler bzw. für kleinere Anwendungen sollte der WPF Designer Cider ausreichend sein. Sind Designer und Entwickler nicht dieselbe Person, ist es jedoch anzuraten, dem Designer kein Visual Studio bereitzustellen, sondern stattdessen Expression Blend zu wählen.

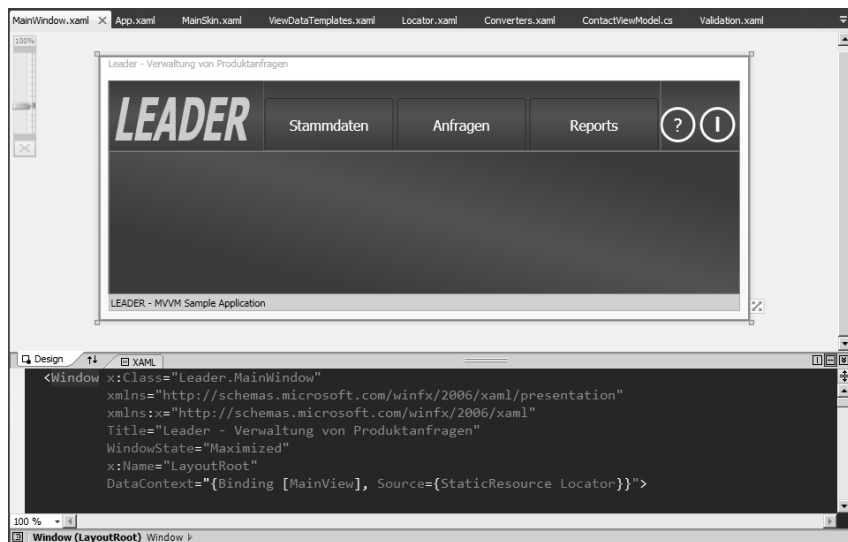


Abb. 1.3: Visual Studio WPF Designer (Cider)

Visual Studio bietet natürlich noch weitere Unterstützung. So wurde auch das Eigenschaftfenster im Vergleich zur Vorgängerversion um zahlreiche Optionen erweitert. Sehr hilfreich ist hierbei die Möglichkeit, Datenbindungen darüber zu setzen, wie in Abbildung 1.4 zu sehen. Mehr zum Thema Datenbindung erfahren Sie in Kapitel 6.

Ebenfalls äußerst hilfreich ist die *Document Outline*. Hiermit ist es möglich, sich innerhalb kürzester Zeit einen Überblick über den Aufbau einer XAML-Datei zu verschaffen. Zusätzlich wird zu jedem Eintrag die visuelle Repräsentation des selektierten Eintrags (inklusive seiner Untereinträge (»Kinder«)) dargestellt (Abbildung 1.4).

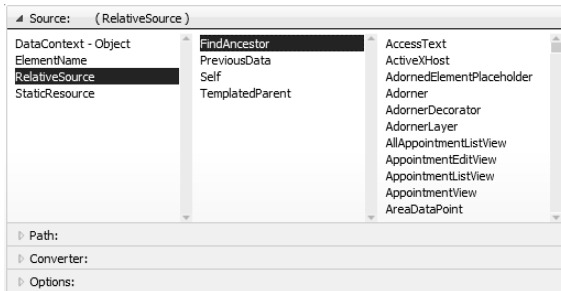


Abb. 1.4: Datenbindungen über Visual Studio 2010

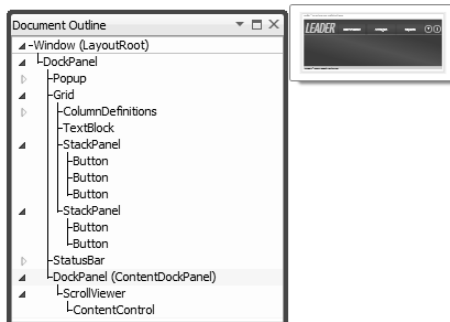


Abb. 1.5: Aufbau einer XAML-Datei

Für den Entwickler ist Visual Studio 2010 sicherlich die beste Wahl. Für den Designer bietet sich das günstigere Microsoft Expression Blend an.

Microsoft Expression Blend

- Microsoft Expression Blend entstammt dem Microsoft Expression Studio. Diese Suite besteht aus insgesamt vier Tools:
- Expression Blend
- Expression Design
- Expression Encoder Pro
- Expression Web + SuperPreview

Die vorgenannten Anwendungen wenden sich an professionelle Webdesigner, -entwickler und Grafiker, um anspruchsvolle Webanwendungen bzw. allgemein Oberflächen erstellen zu können.

Mit Expression Blend erhält der Designer ein ausgefeiltes Werkzeug, um WPF-Oberflächen zu entwickeln. Hierbei werden ihm sämtliche grafischen Möglichkei-

ten angeboten. Wahlweise kann dazu ein Designer, aber auch der XAML-Editor verwendet werden. So ist es beispielsweise möglich, ohne eine Zeile Sourcecode innerhalb weniger Minuten eine kleine Animation zu erstellen.

Da Expression Blend auch mit .NET Solutions umgehen kann und zudem einwandfrei mit Visual Studio 2010 zusammenarbeitet, ist es das ideale Tool für den Designer, um die gewünschte Oberfläche zu erstellen. Die angelegten Dateien können mit dem Entwickler geteilt werden, welcher schlussendlich die Implementierung vornimmt. Die Trennung von Design und Logik ist somit sowohl auf Anwendungsebene als auch auf Basis der Zuständigkeiten möglich.

Die wichtigsten Features von Expression Blend sind:

- *SketchFlow*: Die Erstellung von schnellen Screen-Prototypen inklusive Ablaufsteuerung der einzelnen Screens. Mithilfe des Resultats können Oberflächen/Layouts sehr schnell (auch gemeinsam mit dem Kunden) erzeugt und besprochen werden. Das Resultat ist ein voll funktionstüchtiger WPF-Prototyp. Zusätzlich können in dem ausgeführten Prototyp Anmerkungen in grafischer Form sowie Kommentare hinterlassen werden, die in der Prototyp-Solution zusammengefasst und verarbeitet werden können. Auch die Generierung einer Dokumentation ist möglich.



Abb. 1.6: Microsoft Expression Blend

- *Import von Adobe Photoshop und Adobe Illustrator*: Oberflächen werden vielfach mit den beiden genannten Anwendungen designed. Blend erlaubt den Import

derartiger Dateien. Dabei kann auch auf einzelne Layer näher eingegangen werden.

- *IntelliSense*: Wie Visual Studio unterstützt auch Expression Blend IntelliSense.
- *Beispieldaten*: Einfache Beispieldaten können sehr schnell erstellt bzw. importiert werden. Dadurch ist es bereits frühzeitig möglich, Probleme der Benutzeroberfläche zu erkennen.

Kaxaml

- Kaxaml (siehe <http://www.kaxaml.com/>) ist ein kostenloser XAML-Editor, der geschriebenes XAML sofort interpretiert und darstellt. Dabei stehen kleine Helferlein zur Verfügung, die den Umgang wesentlich erleichtern:
- *Snippets*: Es werden viele vorgefertigte XAML-Teile angeboten. Diese können entweder direkt verwendet oder zu Lernzwecken genutzt werden. Zusätzlich besteht die Möglichkeit, eigene Snippets abzulegen und zu verwalten.

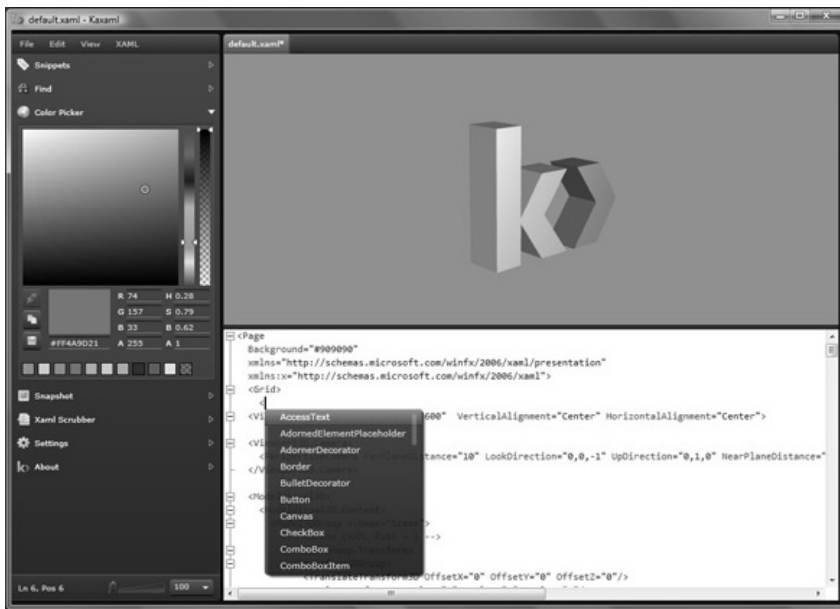


Abb. 1.7: Kaxaml

- *Snapshot*: Von der aktuellen Darstellung kann ein Snapshot erstellt werden, welcher wahlweise in die Zwischenablage kopiert wird oder ins Dateisystem geschrieben werden kann.

- **XAML Scrubber:** Wird lange an einer XAML-Datei gearbeitet und eventuell mit unterschiedlichen Editoren, kann es schon mal vorkommen, dass die Formatierung und damit die Übersichtlichkeit darunter leiden. Hierbei bietet der Scrubber Unterstützung: Einfach die gewünschten Einstellungen (Einrückung etc.) tätigen und schon wird das gesamte XAML formatiert. Zusätzlich zu dieser Funktionalität ist es auch möglich, Attribute nach ihrer Wichtigkeit zu sortieren.

Grundsätzlich ist dieses Tool kaum vergleichbar mit beispielsweise Expression Blend, dennoch ermöglicht es, ohne viel Aufwand kleinere Szenarien und/oder Demos zu entwickeln. Dafür ist es in der Regel nicht notwendig, große Umgebungen zu laden, die zudem einen hohen Ressourcenverbrauch haben.

XamlPad

- Bei XamlPad handelt es sich um ein Werkzeug aus dem Windows SDK. Dieses wird bei dessen Installation mit auf den Rechner gespielt und ist ab dann verfügbar. Wie schon bei Kaxaml handelt es sich auch hierbei um einen visuellen Editor für XAML. Die einzelnen Funktionen sind jedoch nicht sehr ausgeprägt und stellen vielmehr die minimalen Anforderungen eines XAML-Editors dar. Für kleinere Demos bzw. Beispiele ist XamlPad dennoch durchaus empfehlenswert.



Abb. 1.8: XamlPad

1.1.4 Systemvoraussetzungen

Last but not least gibt es Voraussetzungen, unter denen WPF-Anwendungen lauffähig sind. Diese werden wie folgt definiert:

- Windows 7
- Windows Vista
- Windows XP Service Pack 2
- Windows Server 2003 Service Pack 1

Erstmals wurde die Windows Presentation Foundation mit dem .NET Framework 3.0 eingeführt. Das bedeutet, dass es mindestens installiert sein muss, um WPF-Anwendungen ausführen zu können.

1.1.5 Weiterführende Informationen

Wenn Sie in Teilbereichen weiterführende Informationen benötigen, dann seien Ihnen folgende Quellen mit auf den Weg gegeben:

- Microsoft Windows SDK – Unterstützt Sie mit sehr vielen Beispielen, Dokumentationen und Tools.
- Microsoft Developer Network – Das MSDN hält eine komplette Onlinedokumentation inklusive Input der Community für Sie bereit.
- Microsoft WindowsClient unter <http://windowsclient.net/wpf> – Viele Beispiele, Dokumentationen, Videos und Hands On Labs.

Zusätzlich zu diesen Möglichkeiten beinhaltet auch Expression Blend einige Beispiele sowohl für die WPF als auch für Silverlight.

1.2 Grundlagen der WPF

Bevor es in den nächsten Kapiteln an die ersten Beispiele und an eine konkrete Anwendung geht, sind einige grundlegende Informationen notwendig. So werden die unterschiedlichen Anwendungs- und Projekttypen erklärt, es wird aufgezeigt, wie die WPF mit Namespaces umgeht und eine kurze Einführung in das Thema der Ressourcen gegeben, wobei eine tiefere Erläuterung in einem eigenständigen Kapitel (siehe Kapitel 7) abgedeckt wird. Zusätzlich werden einfache (aber deshalb nicht unwichtige) Standardaufgaben gezeigt, die Sie vor Ihrer ersten Anwendung unbedingt kennen sollten.

1.2.1 Anwendungs- und Projekttypen

Um ein wenig Ordnung in ein verwirrendes Thema zu bringen, gilt es, zwischen zwei Typen zu unterscheiden (die sich jedoch teilweise überschneiden):

- Anwendungstypen
- Projekttypen

Unter den »Anwendungstypen« werden die unterschiedlichen Varianten zusammengefasst, die beschreiben, wie eine WPF-Anwendung ausgeführt wird. Hier gibt es insgesamt drei unterschiedliche Typen:

- *Windows-Anwendungen*: Diese nutzen meist das grafische Potenzial der WPF und sind als normale Windows Clients gedacht.
- *Browseranwendungen (XAML)*: Diese Anwendungen werden via Click Once installiert und im Browser ausgeführt.
- *Loose XAML*: Anwendungen, die keinen Code beinhalten, sondern lediglich aus XAML bestehen.

Als »Projekttypen« sind an dieser Stelle die Typen bezeichnet, die Sie als Projekt im Visual Studio kreieren können:

- WPF Application
- WPF Browser Application
- WPF Custom Control Library
- WPF User Control Library

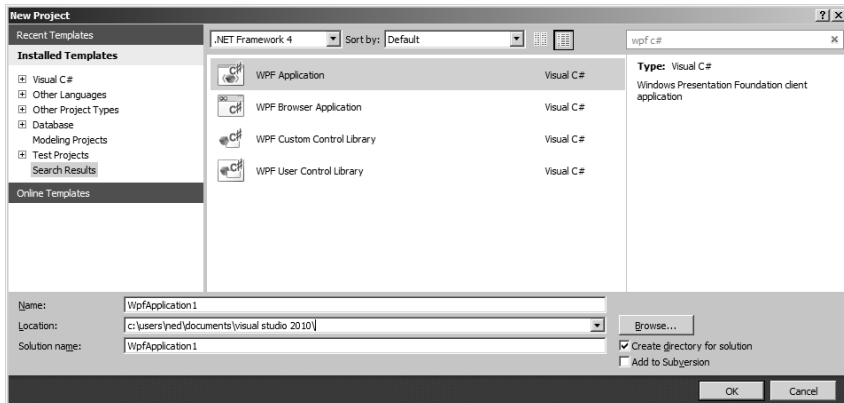


Abb. 1.9: WPF-Projekttypen

In Tabelle 1.1 finden Sie eine Kurzübersicht der einzelnen Projekttypen. Anschließend folgt eine ausführliche Beschreibung der einzelnen Anwendungstypen und welche Entscheidungen bei der Auswahl zum Tragen kommen.

Projekttyp	Beschreibung
WPF Application	Damit erstellen Sie eine typische Windows-Anwendung, welche auf der WPF basiert.
WPF Browser Application	Anwendungen, die im Browser ausgeführt werden und zudem mit Internetrechten ausgestattet sind.
WPF Custom Control Library	Bibliotheken für Steuerelemente, die von anderen
WPF User Control Library	Anwendungen benutzt werden können.

Tabelle 1.1: Übersicht der Projekttypen in Visual Studio

Bevor ein Projekt gestartet wird, sind sehr viele Überlegungen anzustellen und darauf basierend Entscheidungen zu treffen. So muss festgelegt werden, welche Technologie für die Umsetzung verwendet wird, auf welche Oberfläche (Windows, Web, Konsole) gesetzt wird, wie diese auszusehen hat, welche Design Patterns eingesetzt werden und viele weitere Fragen müssen beantwortet werden. Dies trifft auch auf eine WPF-Anwendung zu. Um im Bereich der WPF die richtigen Entscheidungen treffen zu können, müssen die Anforderungen genau bekannt sein. Erst dadurch lässt sich entscheiden, unter welchen Rahmenbedingungen die Anwendung implementiert wird und wie der Aufbau auszusehen hat. Hierzu gibt es drei unterschiedliche Wege:

- **XAML + Code:** Die Anwendung besteht sowohl aus XAML-Markups als auch aus Code, der für die Logik eingesetzt wird. Der Code wird vorwiegend mittels C# und VB.NET entwickelt. Dies stellt die am häufigsten verwendete Methode dar.
- **Code:** Auch ohne die Verwendung von XAML lassen sich WPF-Anwendungen entwickeln. Hierzu wird auch die Oberfläche über den Code generiert.
- **XAML:** Es ist nicht nur möglich, eine WPF-Anwendung komplett per Code zu entwickeln. Ebenso kann hierfür lediglich XAML zum Zuge kommen.

Wie bereits angesprochen, müssen sämtliche Vorgaben bekannt sein, um sich für eine der obigen Varianten entscheiden zu können. Wie unschwer zu erkennen ist, besitzen alle Varianten ihre Vor- und Nachteile. Finden wir gemeinsam heraus, welche das sind.

XAML + Code-Anwendungen

Wie der Name schon sagt, handelt es sich dabei um Anwendungen, die sowohl XAML-Markup enthalten als auch Code (beispielsweise in C# geschrieben). Nach

der Erstellung eines Projekts dieses Typs sind die Dateien `App.xaml` und `App.xaml.cs` sowie die Dateien `MainWindow.xaml` und `MainWindow.xaml.cs` eingebunden. XAML-Dateien enthalten das Markup und beschreiben, wie dieser Teil auszusehen hat, während sich in den C#-Dateien die Logik verbirgt. Dadurch wird bereits ein Grundmaß an Trennung zwischen Präsentation und Logik vorgegeben. Dieses Vorgehen sollte von Ihnen unbedingt weitergeführt bzw. kann durchaus noch verschärft eingesetzt werden. Die Zusammengehörigkeit der XAML- und C#-Dateien wird auch durch den Solution Explorer von Visual Studio dargestellt.

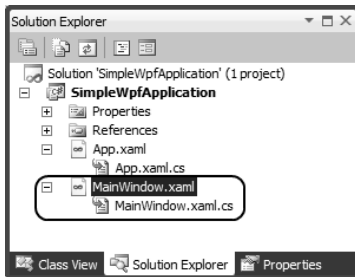


Abb. 1.10: Darstellung XAML- und Code-Behind-Dateien

Code-Anwendungen

Wie der Name schon sagt, wird kein XAML für die Darstellung verwendet. Dies kann natürlich mehrere Gründe haben. So kann es sich bei der Anwendung um ein Programm handeln, welches keine Ausgabe besitzen soll. Das wäre zwar grundsätzlich möglich, jedoch empfiehlt es sich, hierzu eine Konsolenanwendung zu erstellen und daher nicht mit der WPF zu arbeiten. Andere Gründe können allerdings darin liegen, dass entweder (aus welchem Grund auch immer) kein XAML verwendet werden darf/soll oder Sie als Entwickler XAML schlichtweg nicht erlernen möchten.

Grundsätzlich können Sie via Code dasselbe machen, wie das auch mittels XAML der Fall ist. In manchen Fällen sind sogar einige Dinge mehr möglich.

Um eine reine Code-Anwendung zu erstellen, erstellen Sie am besten eine normale WPF-Anwendung. Dadurch wird ein Projekt angelegt, das bereits alle notwendigen Referenzen enthält. In weiterer Folge löschen Sie die Dateien `App.xaml`, `App.xaml.cs` sowie `MainWindow.xaml` und `MainWindow.xaml.cs`. Nun erstellen Sie eine neue Klasse, die von `Window` erbt (dazu ist der Namespace `System.Windows` einzubinden). Beachten Sie, dass Sie natürlich einen Einstiegspunkt in Ihre Anwendung benötigen und daher eine `Main`-Methode erstellen müssen. Diese muss zudem mit dem Attribut `STAThread` versehen werden, um das Single-Thread-Modell zu aktivieren. Das ist für die Verwendung von UI-Kompo-

menten notwendig. Damit die benötigten WPF-Steuer-elemente eingebunden werden können, ist schließlich noch der Namespace `System.Windows.Controls` einzubinden. Nun kann darangegangen werden, die einzelnen Steuer-elemente per Code zu erstellen und zuzuweisen, damit diese schlussendlich auch angezeigt werden. Listing 1.1 zeigt ein vollständiges Beispiel für eine Anwendung ohne die Verwendung von XAML.

```
using System;
using System.Windows;
using System.Windows.Controls;

namespace OnlyCodeApp
{
    public class OnlyCodeApp : Window
    {
        public OnlyCodeApp()
        {
            this.Title = "Only Code";
            this.Width = 200;
            this.Height = 140;

            Button btnOpen = new Button();
            btnOpen.Height = 50;
            btnOpen.Content = "Click here!";
            btnOpen.Click += new RoutedEventHandler(btnOpen_Click);

            Button btnExit = new Button();
            btnExit.Height = 50;
            btnExit.Content = "Exit";
            btnExit.Click += new RoutedEventHandler(btnExit_Click);

            StackPanel stPanel = new StackPanel();
            stPanel.Orientation = Orientation.Vertical;
            stPanel.Children.Add(btnOpen);
            stPanel.Children.Add(btnExit);

            this.Content = stPanel;
        }

        private void btnOpen_Click(object sender, RoutedEventArgs e)
        {
            MessageBox.Show("You've clicked me");
        }

        private void btnExit_Click(object sender, RoutedEventArgs e)
```

```
{  
    Application.Current.Shutdown();  
}  
  
[STAThread]  
public static void Main()  
{  
    new Application().Run(new OnlyCodeApp());  
}  
}
```

Listing 1.1: Anwendung ohne Markup

Hinweis

Das hier gezeigte Beispiel finden Sie auf der beigelegten CD unter Kapitel101\Kap1_OnlyCodeApp.

Abbildung 1.11 zeigt das Ergebnis des Anwendungsbeispiels.



Abb. 1.11: Ergebnis der Code-Only-Anwendung

Hinweis

Anhand des gezeigten Beispiels sehen Sie, dass das Schreiben einer WPF-Anwendung ohne XAML grundsätzlich möglich ist. Dennoch empfiehlt es sich nicht, dies in der Realität zu tun. Das mag zwar bei kleineren Anwendungen durchaus sinnvoll sein, bei größeren Anwendungen kann so jedoch sehr schnell ein großer Teil der möglichen Flexibilität verloren gehen. In einigen Fällen wird es natürlich unvermeidlich sein, dass Steuerelemente dynamisch erzeugt werden.

XAML-Anwendungen

Reine XAML-Anwendungen bestehen, wie der Name schon sagt, nur aus XAML. Um eine derartige Anwendung zu erstellen, legen Sie einfach eine normale *WPF Application* an und löschen alle Code-Behind-Dateien, die standardmäßig mit

angelegt werden (`App.xaml.cs` und `MainWindow.xaml.cs`). Das Löschen dieser Dateien stellt kein Problem dar, da diese ohnehin keine Logik enthalten werden und somit innerhalb des Projekts überflüssig sind. Listing 1.2 zeigt das bereits bekannte Beispiel als reine XAML-Anwendung.

```
<Window x:Class="XamlOnlyApp.MainForm"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Title="XAML Only Application" Height="79" Width="300">
  <StackPanel Orientation="Vertical">
    <Button x:Name="ShowButton" Content="Click Me!"
      Click="ShowButton_Click"></Button>
    <x:Code>
      void ShowButton_Click(object sender, RoutedEventArgs e)
      {
        MessageBox.Show("You've clicked me");
      }
    </x:Code>
    <Button x:Name="ExitButton" Content="Exit"
      Click="ExitButton_Click"></Button>
    <x:Code>
      void ExitButton_Click(object sender, RoutedEventArgs e)
      {
        Application.Current.Shutdown();
      }
    </x:Code>
  </StackPanel>
</Window>
```

Listing 1.2: Reine Markup-Anwendung

Hinweis

Das gezeigte Beispiel finden Sie auf der beigelegten CD unter `Kapitel01\Kap1_XamlOnlyApp`.



Abb. 1.12: Das bekannte Beispiel als reine XAML-Anwendung

Hinweis

So ganz ohne Code kommt eine reine XAML-Anwendung dann doch nicht aus. Schließlich finden sich Buttons etc., die eine bestimmte Aktion durchführen sollen. Hierzu wird bei diesem Anwendungstyp das XAML-Tag `<x:Code>` verwendet. Damit ist es möglich, innerhalb von XAML-Markup, Code zu implementieren, der ausgeführt werden soll. Dies ist für kleine Demos absolut kein Problem, sollte so jedoch in einer Anwendung nie eingesetzt werden, da dadurch Visualisierung und Code vermischt werden. So ist es in weiterer Folge nicht nur schwieriger, die Oberfläche selbst zu verändern, sondern selbst die Suche nach einem etwaigen Fehler kann sich dadurch zu einem stundenlangen Martyrium hinziehen, mal davon abgesehen, dass dieser Code eher nicht als testbar anzusehen ist.

1.2.2 Assemblies und Namespaces

Die Typen der Windows Presentation Foundation finden sich über mehrere Assemblies und Namespaces verteilt. Die wichtigsten davon werden bei der Anlage eines Projekts automatisch von Visual Studio eingebunden (siehe Abbildung 1.14):

- PRESENTATIONCORE.DLL
- PRESENTATIONFRAMEWORK.DLL
- WINDOWSBASE.DLL

In Abbildung 1.13 finden Sie die ersten beiden genannten Assemblies in einer Architektur-Übersicht. Die hell hervorgehobenen Blöcke stellen die zu der WPF gehörenden Teile dar. Auffällig ist hierbei »Milcore« (MIL selbst steht für Media Integration Layer). Diese Bibliothek ist nicht verwaltet und ermöglicht die Integration mit DirectX. Sie wird für das Rendering verwendet. PRESENTATIONCORE und PRESENTATIONFRAMEWORK sind die Haupt-Assemblies, die für die WPF notwendig sind. Hierbei stellt PRESENTATIONCORE alle notwendigen Basisklassen für einen Presentation Layer zur Verfügung. PRESENTATIONFRAMEWORK enthält schlussendlich die WPF-spezifischen Implementierungen. Enthalten sind unter anderem sämtliche Steuerelemente.

Kopieren in
Ausgabe-Verzeichnis

Abb. 1.13: WPF-Architektur

Die WPF-spezifischen Namespaces finden sich unter `System.Windows` und den darunter liegenden Namespaces.

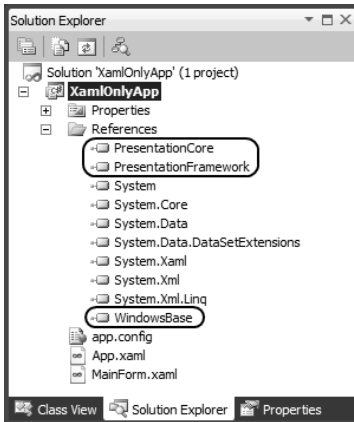


Abb. 1.14: Die Standardreferenzen

Hinweis

Im Laufe der Zeit werden Sie erfahren, dass einige Typen ein Äquivalent im Namespace `System.Windows.Forms` besitzen (beispielsweise der `OpenFileDialog`). Dies kann mitunter zu Problemen führen, wenn Sie irrtümlicherweise auch den Namespace `System.Windows.Forms` referenziert haben. Der Hintergrund ist einfach erklärt: Einerseits sollen die bereits bekannten Typen mit neuen Typen einfach assoziiert werden können. Auf der anderen Seite wurden sie in einen anderen Namespace gelegt, um eine deutliche Unterscheidung zu gewährleisten. Zudem ist es oft nicht gewünscht, den Windows-Forms-Namespace einzuschließen (zusätzliche Abhängigkeit usw.).

1.2.3 Ressourcen

Ressourcen wurden bereits an einigen Stellen angesprochen. Wer bereits Windows Forms entwickelt hat, weiß über das Thema Ressourcen grundsätzlich Bescheid. Bei der WPF werden sie zu den gleichen Zwecken verwendet, sind jedoch um einiges mächtiger.

Hinweis

Ressourcen sind ein hilfreiches Werkzeug, die einiges in der Programmierung erleichtern können. So ist es möglich, Informationen (Text, Grafiken etc.) über eigene Ressourcendateien zur Verfügung zu stellen. Diese können entweder direkt in ein Projekt eingebunden oder von externen Dateien geladen werden. Im letzteren Fall ist es sehr einfach möglich, die Ressourcen auszutauschen (Austausch aufgrund von Fehlern in der Übersetzung bei einer mehrsprachigen

Anwendung). Es ist ebenfalls möglich, Ressourcen in eine eigene Assembly zu packen. Diese kann zu einem späteren Zeitpunkt editiert bzw. ebenfalls ausgetauscht werden, sollte sich dies als notwendig erweisen.

Beispielsweise können nicht nur Zeichenfolgen ausgegliedert werden, sondern es besteht die Möglichkeit

- Steuerelemente auszugliedern
- Templates zu verwalten
- Styles zu definieren und zu ändern

Zudem gibt es unterschiedliche Arten von Ressourcen bzw. können diese auf unterschiedliche Art und Weise eingebunden werden:

- statisch
- dynamisch

Der Unterschied besteht darin, wann bzw. wie oft die Ressource ausgewertet wird. Bei einem statischen Zugriff wird die Ressource beim ersten Zugriff ausgewertet, bei allen weiteren nicht mehr. Beim dynamischen Zugriff auf eine Ressource entscheidet die WPF, wann die Ressource neu eingelesen werden muss. Daher können zur Laufzeit Änderungen erfolgen, die Auswirkung auf beispielsweise die Oberfläche haben.

Dies sollte als kurze Einführung in das Thema der Ressourcen genügen. Eine ausführliche Behandlung der Ressourcen finden Sie in Kapitel 7.

Hinweis

Alle in diesem Buch gezeigten Beispiele sind auf der beigelegten CD zu finden und wurden mit Visual Studio 2010 erstellt.

Dieses Kapitel hat eine erste Einführung in die WPF gegeben, die Nutzen-Frage geklärt, Werkzeuge und Systemvoraussetzungen vorgestellt sowie über die Anwendungstypen inklusive der wichtigsten Komponenten informiert. Das folgende Kapitel geht nun näher auf XAML ein.