Richard Kaiser

www.rkaiser.de

WPF und MVVM

mit Visual Studio 2015/2017

Inhalt

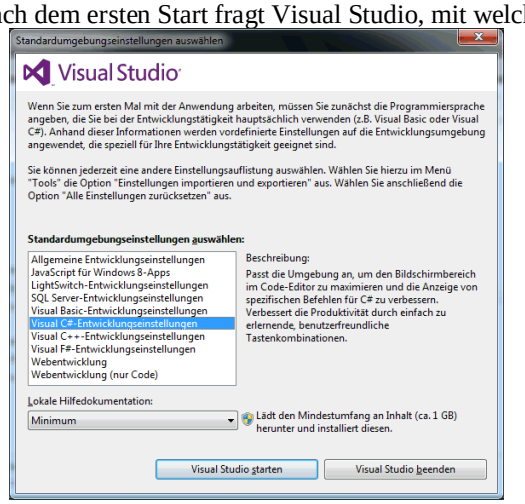
1 Die Entwicklungsumgebung

Visual Studio besteht aus verschiedenen Werkzeugen (Tools), die einen Programmierer bei der Entwicklung von Software unterstützen. Eine solche Zusammenstellung von Werkzeugen zur Softwareentwicklung bezeichnet man auch als Programmier- oder **Entwicklungsumgebung**.

Einfache Entwicklungsumgebungen bestehen nur aus einem Editor und einem Compiler. Für eine effiziente Entwicklung von komplexeren Anwendungen (dazu gehören viele Windows-Anwendungen) sind aber oft weitere Werkzeuge notwendig. Wenn diese wie in Visual Studio in einem einzigen Programm integriert sind, spricht man auch von einer **integrierten Entwicklungsumgebung** (engl.: „integrated development environment“, **IDE)**.

In diesem Kapitel wird zunächst an einfachen Beispielen gezeigt, wie man mit Visual Studio 2015/2017 Windows-Programme mit einer grafischen Benutzeroberfläche entwickeln kann. Anschließend (ab Abschnitt 1.2) werden dann die wichtigsten Werkzeuge von Visual Studio ausführlicher vorgestellt. Für viele einfache Anwendungen (wie z.B. die Übungsaufgaben) reichen die Abschnitte bis 1.5. Die folgenden Abschnitte sind nur für anspruchsvollere oder spezielle Anwendungen notwendig. Sie sind deshalb mit dem Zeichen Θ gekennzeichnet und können übergangen werden. Weitere Elemente der Entwicklungsumgebung werden später beschrieben, wenn sie dann auch eingesetzt werden können.

Nach dem ersten Start fragt Visual Studio, mit welcher Programmiersprache bevorzugt gearbeitet wird:



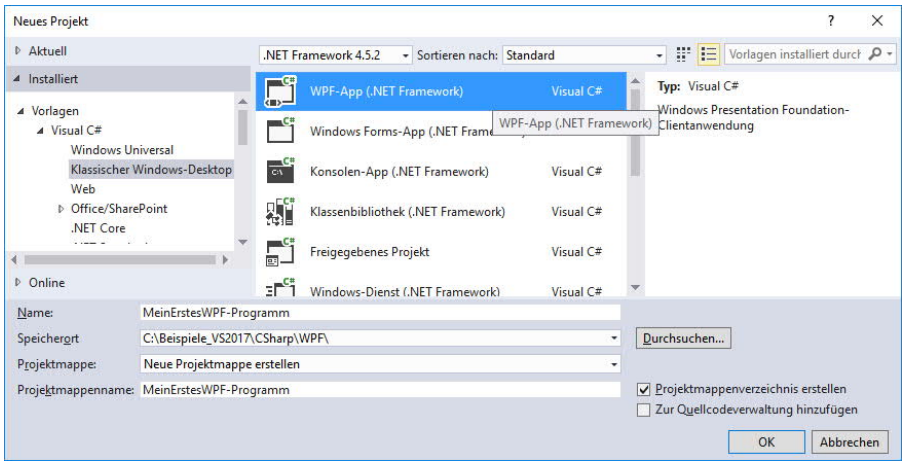
Wenn Sie mit C# arbeiten, wählen Sie hier **Visual C#**. Sie können diese Einstellungen jederzeit mit *Extras|Einstellungen importieren und exportieren* ändern.

1.1 Elementare WPF-Konzepte: Ein erstes kleines Programm

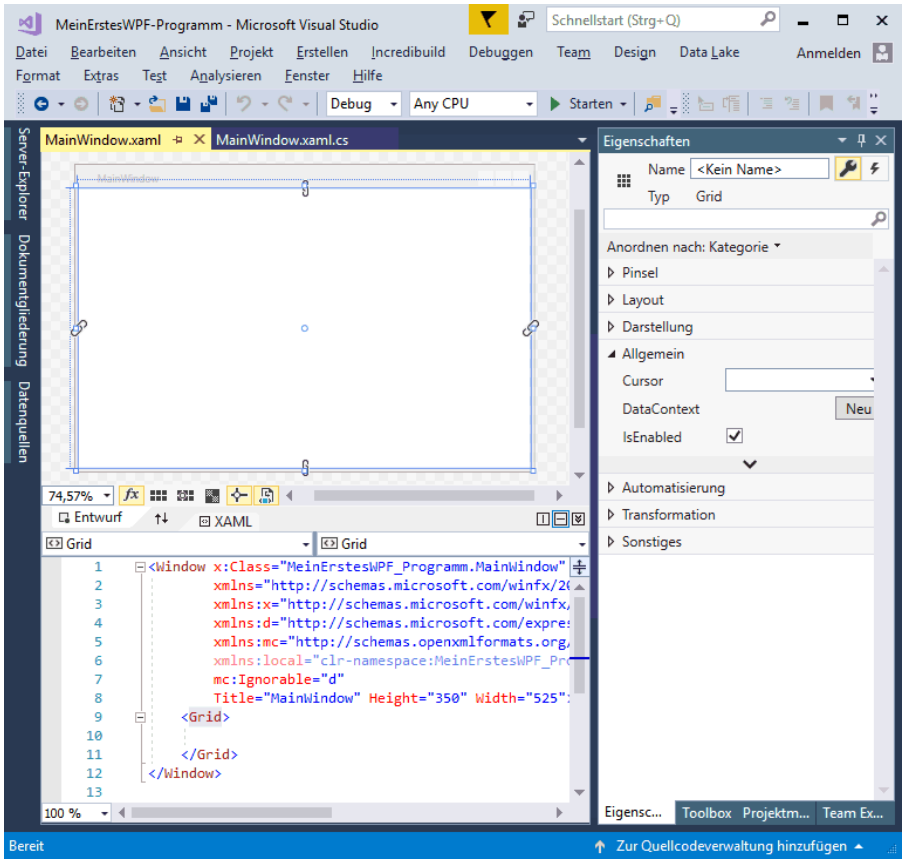
Im Folgenden sollen am Beispiel eines einfachen WPF-Projekts die elementaren Grundkonzepte von WPF-Anwendungen gezeigt werden.

1.1.1 Ein WPF-Projekt anlegen

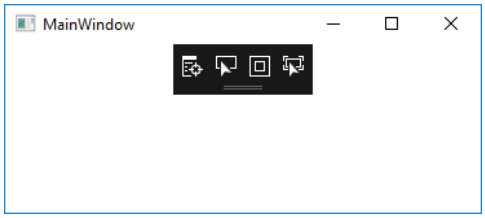
In Visual Studio findet man unter ***Datei|Neu|Projekt|Installiert|Vorlagen|Visual C#|Klassischer Windows Desktop*** für verschiedene Arten von Anwendungen. Eine Windows-Anwendung mit einer graphischen Benutzeroberfläche erhält man mit der Vorlage ***WPF-App***. Ein solches Projekt legt man dann an, indem man nach *Name* einen Namen und nach *Ort* ein Verzeichnis für das Projekt eingibt und dann den OK-Button anklickt:



Anschließend werden einige Elemente der IDE angezeigt. Die Anordnung dieser Elemente hängt von verschiedenen Faktoren ab und kann von der nächsten Abbildung abweichen:



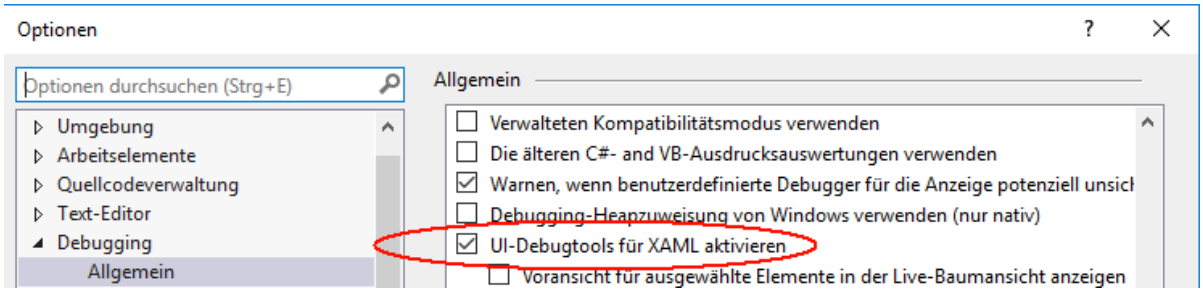
Das **Fenster** *MainWindow* ist der Ausgangspunkt für alle WPF-Anwendungen, die mit Visual Studio entwickelt werden. Es entspricht dem Fenster, das beim Start des Programms mit F5 angezeigt wird:



Stören Sie sich hier nicht an den **XAML-Debug Tools**

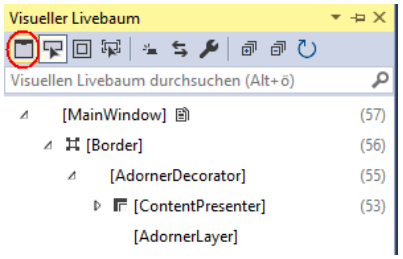
Sie werden nur angezeigt, wenn man das Programm im Debug-Modus startet, aber nicht im Release-Modus. Sie kann durch einen Klick auf die untere Zeile verkleinert werden:

Mit *Extras|Optionen|Debugging|Allgemein||UI-Debugtools for XAML* abgeschaltet werden:



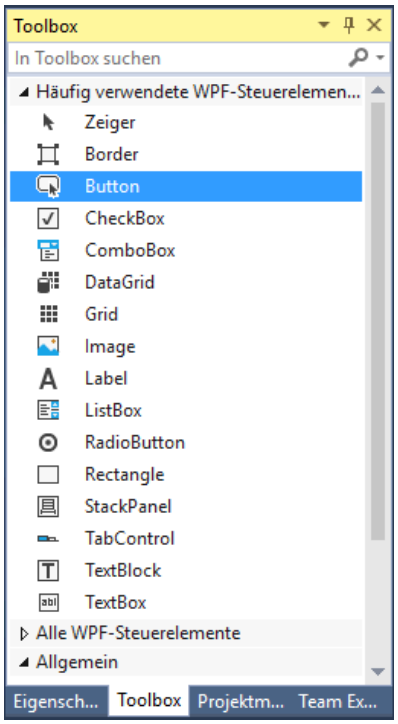
Da die damit verbundenen Debug-Möglichkeiten aber oft hilfreich sind (siehe Abschnitt 2.5.4), schalten Sie es besser doch nicht ab.

Es kann außerdem nach der Anzeige des Visuellen Livebaums durch ein Anklicken des linken Buttons ausgeschaltet werden:

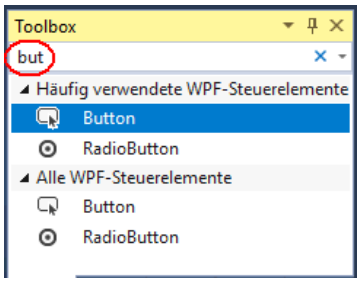


1.1.2 Einfache Steuerelemente für die Benutzeroberfläche

Das *MainWindow* kann mit den in der **Toolbox** (**Werkzeugkasten**) verfügbaren **Steuerelementen** (**Controls**) gestaltet werden. Die Toolbox wird nur angezeigt, wenn das MainWindow angezeigt wird. Sie enthält viele der unter Windows üblichen Steuerelemente.

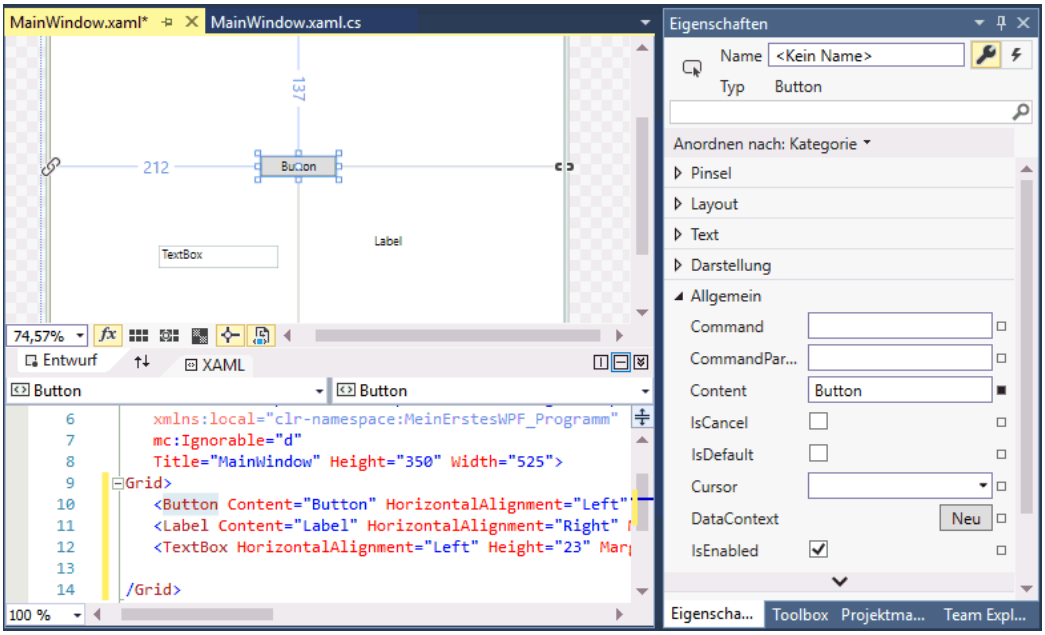


In der obersten Zeile kann man einen Suchbegriff eingeben. Dann werden nur noch die Steuererelemente angezeigt, die zu diesem Namen passen:



Um ein **Element** aus der Toolbox **auf das Fenster** zu **setzen**, zieht man es einfach auf das Fenster. Oder man klickt mit der Maus zuerst auf die Toolbox-Zeile (sie wird dann als markiert dargestellt) und dann auf die Stelle im Fenster, an die die linke obere Ecke kommen soll.

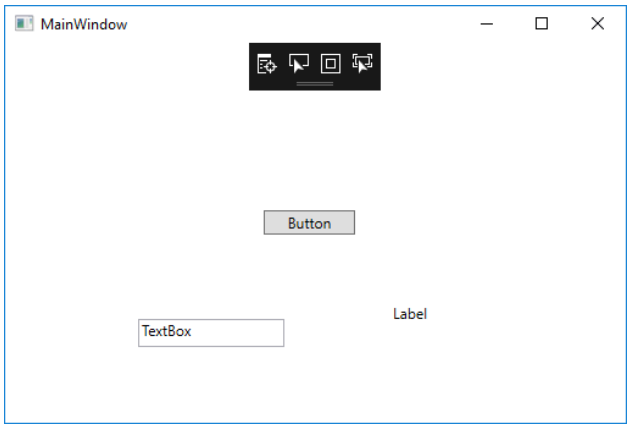
Beispiel: Nachdem man ein Label (Zeile neun in *Häufig verwendete WPF-Steuerelemente*, mit dem großen *A*), eine TextBox (letzte Zeile, Aufschrift *ab*) und einen Button (dritte Zeile) auf das MainWindow gesetzt hat, sieht Visual Studio etwa folgendermaßen aus:



Die Steuerelemente werden auch im XAML-Editor angezeigt, und das im MainWindow angeklickte Steuerelement im Eigenschaftenfenster.

Durch diese Spielereien haben Sie **schon ein richtiges Windows-Programm** erstellt – zwar kein besonders nützliches, aber immerhin. Sie können es folgendermaßen starten:

- mit *Debuggen|Debugging starten* von der Menüleiste, oder
- mit *F5* von einem beliebigen Fenster in Visual Studio oder
- durch den Aufruf der vom Compiler erzeugten Exe-Datei.



Dieses Programm hat schon viele Eigenschaften, die man von einem Windows-Programm erwartet: Man kann es mit der Maus verschieben, vergrößern, verkleinern und schließen.

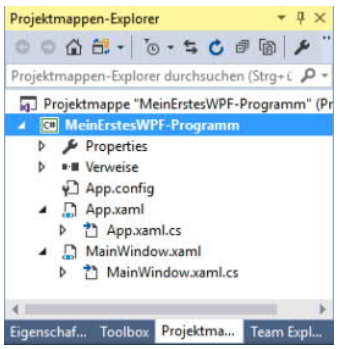
Vergessen Sie nicht, Ihr **Programm** zu **beenden**, bevor Sie es weiterbearbeiten. Sie können den Compiler nicht erneut starten, solange das Programm noch läuft.

Diese Art der Programmierung bezeichnet man als **visuelle Programmierung**. Während man bei der konventionellen Programmierung ein Programm ausschließlich durch das Schreiben von Anweisungen (Text) in einer Programmiersprache entwickelt, wird es bei der visuellen Programmierung ganz oder teilweise aus vorgefertigten grafischen Komponenten zusammengesetzt.

Mit Visual Studio kann die Benutzeroberfläche eines Programms visuell gestaltet werden. Damit sieht man bereits beim Entwurf des Programms, wie es später zur Laufzeit aussehen wird. Die Anweisungen, die als Reaktionen auf Benutzereingaben (Mausklicks usw.) erfolgen sollen, werden dagegen konventionell in einer Programmiersprache (z.B. C#**)** geschrieben.

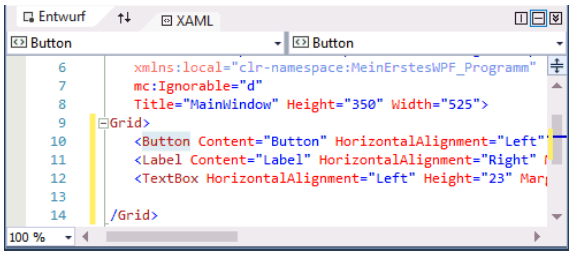
1.1.3 XAML-Grundkonzepte

Wie man im Projektmappen-Explorer (*Ansicht*|*Projektmappen-Explorer*) sieht, besteht ein Projekt für eine WPF-Anwendung aus XAML-Dateien mit der Namensendung xaml und C#-Dateien mit der Namensendung cs:



Die XAML-Datei zum *MainWindow* wird im **XAML-Editor** (Register „XAML“) angezeigt und enthält in dem XML-Dialekt XAML („E**X**tensible **A**pplication **M**arkup **L**anguage“, wird meist „xammel“ ausgesprochen) eine Beschreibung der grafischen Darstellung des Fensters.

Die Inhalte dieser Datei entsprechen weitgehend der *Form1.Designer.cs*-Datei einer Windows Forms-Anwendung. Der wesentliche Unterschied zwischen diesen beiden Dateien besteht in der Sprache, in der sie geschrieben sind: Die *Designer.cs*-Datei einer Forms-Anwendung ist in C# geschrieben, während die XAML-Datei einer WPF-Anwendung in XAML geschrieben ist.



Dieser „kleine“ Unterschied zwischen WPF und Forms-Anwendungen provoziert natürlich sofort die Frage „Was bringt das? Es ist doch egal, in welcher Sprache ein Fenster beschrieben wird“. Für Windows ist das sicher richtig. Aber Windows war auch nicht der Grund, von C# in einen XML-Dialekt zu wechseln. Vielmehr soll die Gestaltung von Fenstern nicht nur Programmierern ermöglicht werden, sondern auch Grafikern, die kein C# können. Dazu gibt es einige **freie XAML-Editoren** und von Microsoft **Expression Blend**.

Der „Umweg“ über die XAML-Dateien soll den folgenden Arbeitsablauf ermöglichen: Ein Programmierer gestaltet ein Fenster zunächst nur vorläufig, so dass es alle benötigten Elemente enthält und diese im Programm angesprochen werden können. Dann gibt er die XAML-Datei an einen **Grafiker**, der weder Visual Studio haben noch C# können muss. Nachdem der Grafiker die XAML-Datei optisch poliert hat, ersetzt der Programmierer seine XAML-Datei durch die des Grafikers.

Der Programmierer muss sich also im Wesentlichen nur noch um die Programmierlogik kümmern. Das sind meist Anweisungen, die als Reaktion auf ein sogenanntes Ereignis ausgeführt werden. Im einfachsten Fall löst das Anklicken eines Buttons ein solches Ereignis aus.

1.1.4 Grafik-Grundkonzepte von WPF

WPF steht für „Windows Presentation Foundation“ und soll die Grundlagen für anspruchsvolle Präsentationen unter Windows bieten.

Das wird einerseits durch die Trennung der Präsentation von der Programmierung erreicht, wie sie im letzen Abschnitt dargestellt wurde. Dazu kommen einige weitere Konzepte:

- Die grafischen Elemente von WPF (Fenster, Steuerelemente usw.) sind **Vektorgrafiken** und nicht Pixelgrafiken wie unter Windows Forms. Das bedeutet, dass sie stufenlos vergrößert oder verkleinert werden können, und dass dabei eine Linie immer als Linie gezeichnet wird. Bei der von Forms-Anwendungen verwendeten Pixelgrafik besteht eine Linie dagegen aus einzelnen Pixeln, die bei einer Vergrößerung oder Verkleinerung körnig aussehen können.
- Größenangaben erfolgen nicht wie bei Windows Forms-Anwendungen in Pixeln, sondern sind geräteunabhängig. Die Maßeinheiten sind 96 dpi (1/96 Zoll), Zoll oder Zentimeter. Damit ist die Größe eines Steuerelements auch nicht mehr von der Bildschirmauflösung abhängig, mit der die Anwendung dargestellt wird.
- Die meisten Steuerelemente können andere Steuerelemente enthalten. So kann z.B. ein Button Grafiken und andere Steuerelemente enthalten. Auch wenn das bei einem Button nicht immer sinnvoll ist: Bei anderen Steuerelementen kann das sehr hilfreich sein und ein sehr **flexibles Design** ermöglichen.
- Da die Steuerelemente sehr flexibel kombiniert werden können, wäre es ziemlich aufwendig, diese Kombinationen im Eigenschaftenfenster abzubilden. Mit Textanweisungen wie XAML ist das einfacher möglich.
- WPF unterstützt nicht nur 2D-Grafiken, sondern auch **3D-Grafiken**. Das ermöglicht Animationen auf Fenstern und anderen Steuerelementen. Obwohl WPF dabei die Hardware effizient nutzt, ist WPF auf gewöhnliche Anwendungen optimiert und nicht auf grafisch aufwendige Spiele. Für solche Spiele ist WPF normalerweise nicht geeignet.

1.1.5 Das Eigenschaftenfenster und der XAML-Editor

Das Aussehen und Verhalten eines Steuerelements ergibt sich aus seinen Eigenschaften und Ereignissen. Diese können im Eigenschaftenfenster (*Ansicht|Eigenschaftenfenster*, nicht mit *Ansicht|Eigenschaftenseiten* verwechseln) und im XAML-Editor gesetzt werden.

Die **Aufschrift** auf einem Steuerelement kann man im *MainWindow.xaml* über das Kontextmenü (rechte Maustaste) „Text bearbeiten“ ändern: