

Richard Kaiser

www.rkaiser.de

**WPF und MVVM
mit Visual Studio
2015/2017**

Inhalt

1 Die Entwicklungsumgebung 1

1.1	Elementare WPF-Konzepte: Ein erstes kleines Programm.....	1
1.1.1	Ein WPF-Projekt anlegen.....	1
1.1.2	Einfache Steuerelemente für die Benutzeroberfläche.....	3
1.1.3	XAML-Grundkonzepte.....	5
1.1.4	Grafik-Grundkonzepte von WPF.....	6
1.1.5	Das Eigenschaftfenster und der XAML-Editor.....	6
1.1.6	Benannte Steuerelemente.....	10
1.1.7	Aktionen beim Anklicken eines Buttons.....	11
1.2	Der Quelltexteditor.....	12
1.2.1	Tastenkombinationen.....	12
1.2.2	Intellisense.....	13
1.2.3	Die Formatierung des Quelltexts.....	14
1.2.4	Definitionen einsehen.....	14
1.2.5	Symbole suchen.....	15
1.2.6	Namen umbenennen.....	15
1.2.7	Zeichenfolgen suchen und ersetzen.....	16
1.3	Elemente von Visual Studio anordnen.....	17
1.4	Einige Tipps zur Arbeit mit Projekten.....	19
1.5	Die Online-Dokumentation (Microsoft-Dokumentation).....	20
1.5.1	Hilfe mit <i>F1</i> in Visual Studio.....	20
1.5.2	Die Microsoft-Dokumentation zu WPF im Internet.....	21
1.5.3	Der Aufbau der MSDN-Dokumentation Seiten.....	24
1.6	Projektmappen und der Projektmappen-Explorer Θ.....	25
1.7	XAML-Browser-Anwendungen (XBAP).....	27

2 Die üblichen Steuerelemente (Common Controls).....29

2.1	Die Klasse <i>MainWindow</i> und der Zugriff auf Steuerelemente.....	29
2.2	Labels, Datentypen und Compiler-Fehlermeldungen.....	31
2.3	Methoden und das Steuerelement <i>TextBox</i>	39
2.3.1	Methoden.....	39
2.3.2	Mehrzeilige <i>TextBox</i> en.....	41
2.4	Vererbung und Komposition bei <i>ListBox</i> und <i>ComboBox</i>	43
2.4.1	Komposition und der Zugriff auf Eigenschaften und Feldvariablen.....	43
2.4.2	Vererbung und die Elemente von Basisklassen.....	44
2.4.3	<i>TextBlock</i> , <i>TextBox</i> und <i>Label</i>	45
2.4.4	Die <i>Content</i> -Eigenschaft der <i>ContentControl</i> -Klassen.....	46
2.5	Buttons und Ereignisse.....	47
2.5.1	Parameter der Ereignisbehandlungsmethoden.....	48
2.5.2	Ereignisse beim Start und Ende eines Programms.....	49
2.5.3	Der Fokus und die Tabulatorreihenfolge.....	50
2.5.4	Verschachtelte Steuerelemente: Buttons mit Bildern.....	52
2.5.5	Die Weiterleitung von Ereignissen bei verschachtelten Steuerelementen.....	54
2.5.6	Die Basisklasse <i>ButtonBase</i>	57
2.5.7	Steuerelemente, die nicht in der Toolbox sind: <i>RepeatButton</i>	57
2.5.8	TouchScreen-Ereignisse.....	58

2.6	CheckBoxen, RadioButtons und einfache <i>if</i> -Anweisungen	58
2.7	Container-Steuerelemente: GroupBox und TabControl	60
2.8	Hauptmenüs und Kontextmenüs	61
2.8.1	Hauptmenüs und der Menüdesigner	62
2.8.2	Kontextmenüs	68
2.9	Standarddialoge	68
2.10	Weitere Fenster und eigene Dialoge	72
2.10.1	Modale Dialoge	72
2.10.2	Einfache Dialoge mit <i>MessageBox.Show</i>	73
3	XAML	75
3.1	Die XML-Syntax für XAML	75
3.1.1	XML-Elemente, XML-Attribute und die erzeugten C#-Objekte	75
3.1.2	Namensbereiche: Ein erster Einblick	77
3.1.3	Kommentare	77
3.1.4	XML-Steuerzeichen	78
3.2	Die Code-Behind Klasse	78
3.3	Stammelemente	80
3.3.1	Window-basierte WPF-Anwendungen	81
3.3.2	Page-basierte WPF-Anwendungen	81
3.4	Namensbereiche	82
3.4.1	Die Elemente der Namensbereichs <i>./presentation</i>	82
3.4.2	Die Elemente des Namensbereichs <i>./xaml</i>	83
3.4.3	CLR-Namensbereiche	85
3.4.4	mc:Ignorable	86
3.5	Objekteigenschaften in XAML setzen	87
3.5.1	Die Attribut-Syntax	87
3.5.2	Die Property-Element Syntax	87
3.5.3	Default-Eigenschaften und die Content-Property	89
3.5.4	Die Attached Property Syntax	90
3.5.5	Eventhandler	92
3.6	Typkonversionen in XAML	93
3.6.1	Typkonverter	93
3.6.2	Markup-Extensions	94
3.7	Baumstrukturen	96
3.8	Das XAML Serialisierungsformat	98
4	Commands	101
4.1	Das Interface <i>ICommand</i>	101
4.2	Vordefinierte Commands	102
4.3	Ein Command für mehrere Steuerelemente	105
4.4	Steuerelemente mit eingebauten Commands	105
5	Layout-Container	107
5.1	Gemeinsamkeiten der Layout-Container	108
5.2	Einfache Layout-Container	109
5.2.1	StackPanel	109
5.2.2	WrapPanel	110
5.2.3	UniformGrid	111
5.3	Grid	112
5.3.1	Ein Grid ohne Zeilen- und Spaltendefinitionen	112
5.3.2	Ein Grid mit Zeilen und Spalten	113
5.3.3	Zeilen und Spalten zusammenfassen	114
5.3.4	Relative Zeilen- und Spaltengrößen	115
5.3.5	Variable Zeilen- und Spaltengrößen mit einem GridSplitter	115
5.4	DockPanel	117
5.4.1	Ein DockPanel mit einer ToolBar, einer Statusbar und einem Menu	117
5.4.2	Ein DockPanel mit einem zweispaltigen Grid wie im Windows Explorer	118

5.4.3	Ein DockPanel mit einem TabControl	119
5.4.4	Eine DialogBox	120
5.5	Canvas und 2D-Grafiken.....	121

6 Dependency Properties125

6.1	Vorteile von Dependency Properties gegenüber gewöhnlichen Eigenschaften	125
6.1.1	Speicherersparnis.....	125
6.1.2	Dynamische Eigenschaften	126
6.2	Die Definition und Registrierung einer Dependency Property	126
6.3	Metadaten von Dependency Properties	128

7 Ressourcen129

7.1	File Ressourcen.....	129
7.2	Objekt Ressourcen	130
7.2.1	Ressourcen Collections	131
7.2.2	Ressourcen-Hierarchien	132
7.2.3	Der Zugriff auf System-Ressourcen.....	132
7.2.4	Der Zugriff auf Ressourcen im Code	132
7.3	Ressourcen-Dictionaries.....	133
7.3.1	Die Definition und Verwendung von Ressource Dictionaries	133
7.3.2	Ressourcen in mehreren Assemblies verwenden	134

8 Styles135

8.1	Eigenschaften in Styles setzen und verwenden.....	135
8.2	Styles für bestimmte Klassen und Style-Hierarchien.....	136
8.3	EventSetter, Trigger und EventTrigger	137

9 Datenbindung.....139

9.1	Datenquellen.....	140
9.1.1	Bindungsausdrücke mit <i>ElementName</i> und <i>Path</i>	140
9.1.2	Bindung an Eigenschaften des aktuellen Window-Objekts	141
9.1.3	Bindung an Ressourcen-Daten und <i>static</i> Elemente.....	142
9.1.4	Bindung an Objekte mit <i>RelativeSource</i>	143
9.2	Der DataContext	144
9.2.1	Ein einfaches MVVM-Beispiel	145
9.2.2	Verschiedene Möglichkeiten zur Definition des DataContext.....	146
9.3	Bindungsfehler	148
9.4	Change Notification (<i>INotifyPropertyChanged</i>).....	151
9.5	Weitere Eigenschaften der Klasse Binding	152
9.5.1	Bindungsrichtungen	152
9.5.2	Die <i>StringFormat</i> Eigenschaft zur Formatierung von vordefinierten Datentypen	152
9.5.3	ValueConverter (<i>IValueConverter</i> implementieren) - BoolToVisibleConverter	153
9.6	Binden an Collections von Objekten.....	155
9.6.1	Daten aus einer Collection mit einer ItemSource darstellen.....	156
9.6.2	Vorlagen zur Formatierung von Daten (Data Templates)	159
9.6.3	Den Inhalt der Collection verändern mit <i>ObservableCollection</i>	160
9.6.4	An einen LINQ-Ausdruck binden.....	160
9.7	Command-Bindung.....	161
9.7.1	<i>ICommand</i> -Eigenschaften und Datenbindung	161
9.7.2	Die Hilfsklasse <i>RelayCommand</i>	163

10 MVVM165

10.1	Die Bestandteile von MVVM.....	165
10.1.1	Das Model	166

10.1.2	Der View	166
10.1.3	Das ViewModel	166
10.1.4	Abhängigkeiten	167
10.2	Ein minimales Beispiel	168

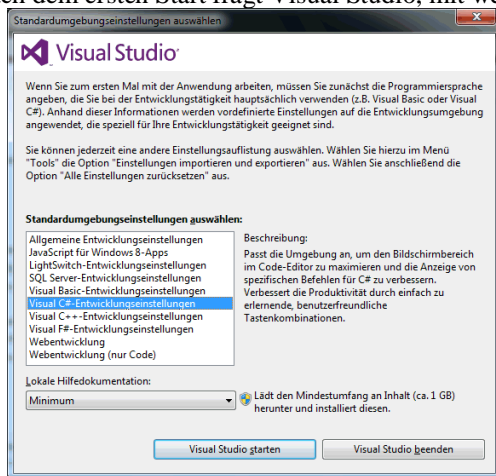
1 Die Entwicklungsumgebung

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:



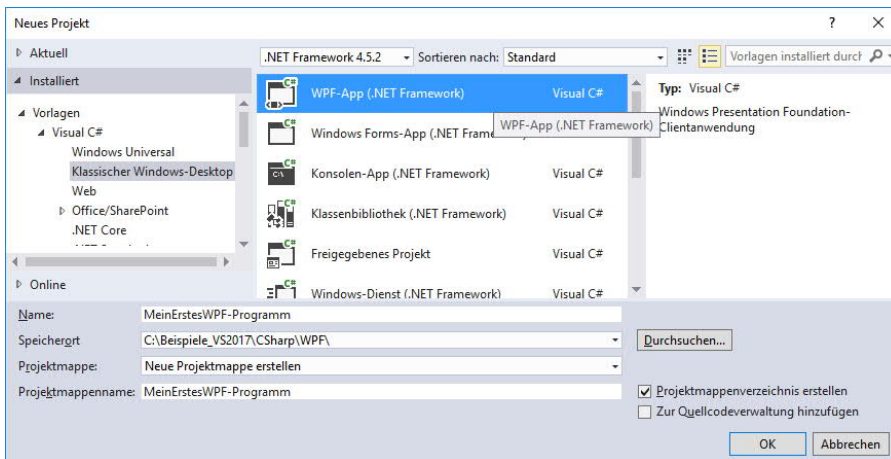
Wenn Sie mit C# arbeiten, wählen Sie hier **Visual C#**. Sie können diese Einstellungen jederzeit mit *Extras/Einstellungen importieren und exportieren* ändern.

1.1 Elementare WPF-Konzepte: Ein erstes kleines Programm

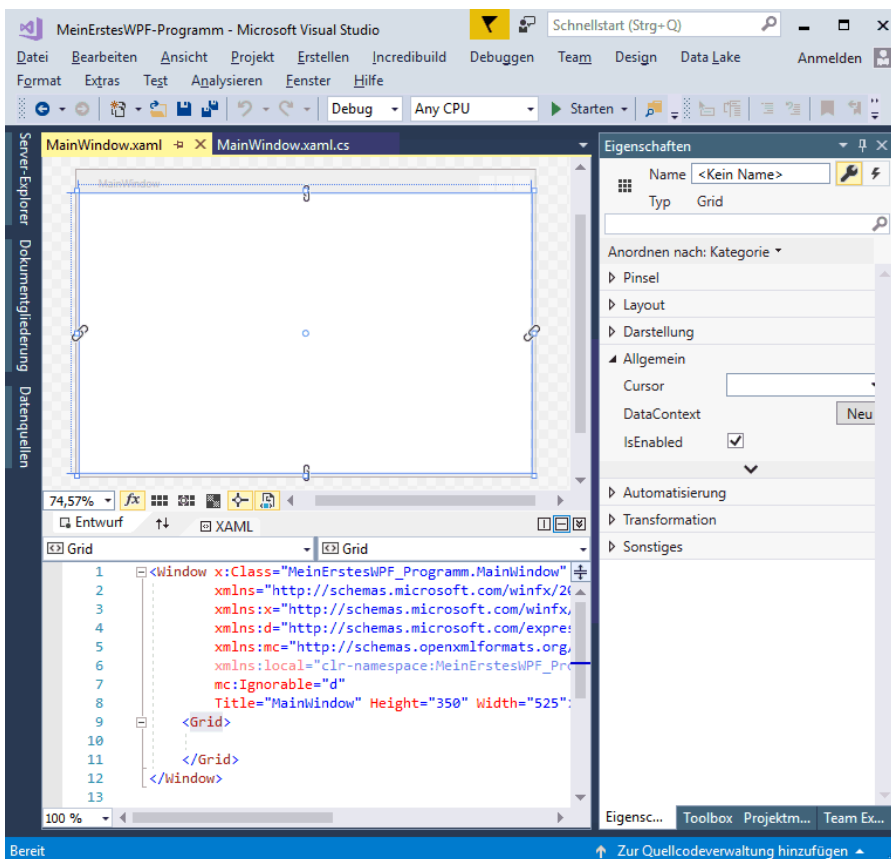
Im Folgenden sollen am Beispiel eines einfachen WPF-Projekts die elementaren Grundkonzepte von WPF-Anwendungen gezeigt werden.

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:



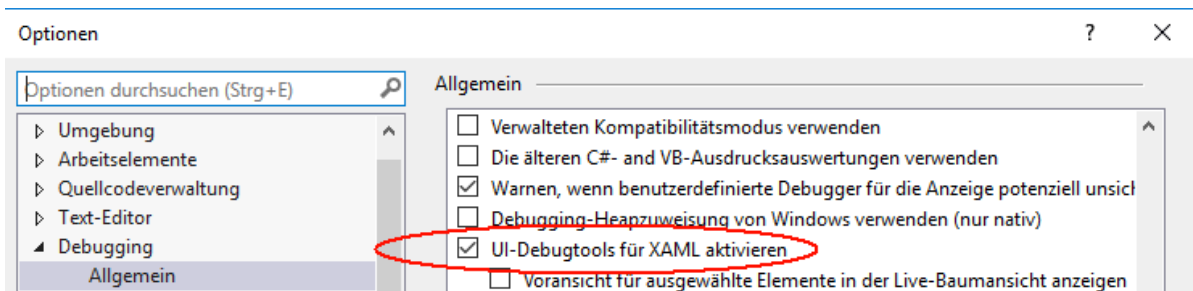
Stören Sie sich hier nicht an den **XAML-Debug Tools**



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:

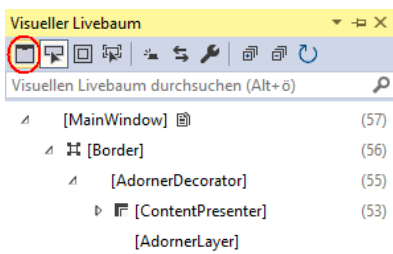


Mit *Extras/Optionen/Debugging/Allgemein/UI-Debugtools for XAML* abgeschaltet werden:



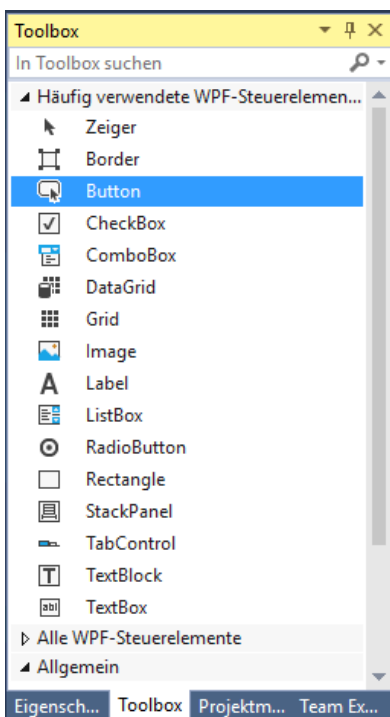
Da die damit verbundenen Debug-Möglichkeiten aber oft hilfreich sind (siehe Abschnitt 2.5.4), schalten Sie es besser doch nicht ab.

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

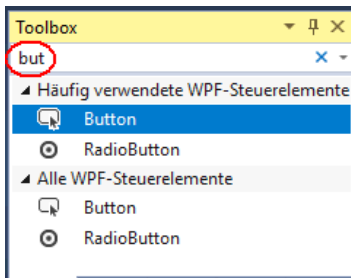


1.1.2 Einfache Steuerelemente für die Benutzeroberfläche

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.

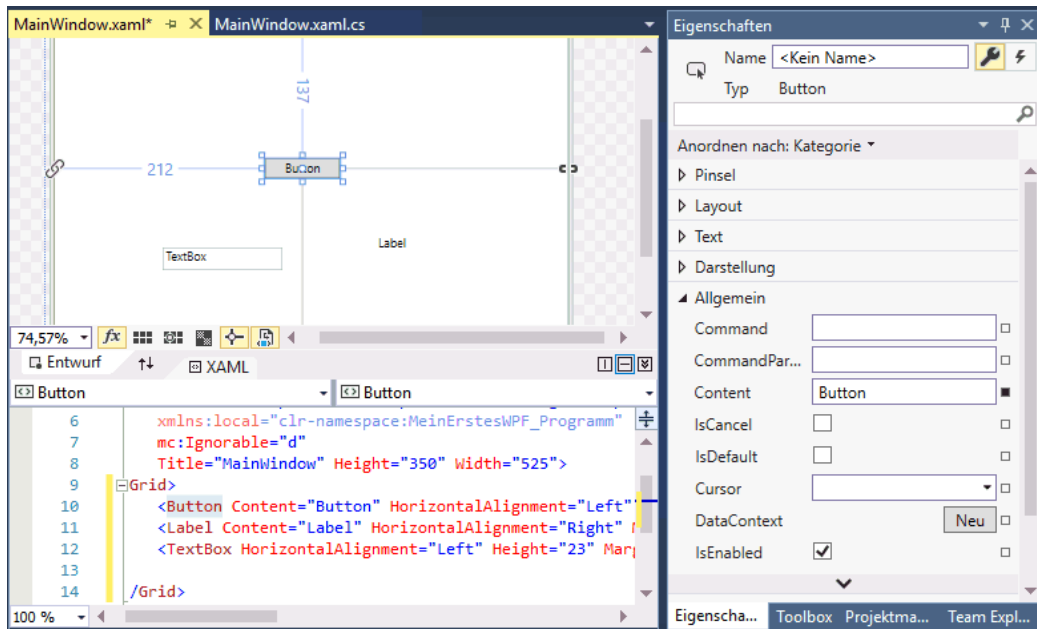


In der obersten Zeile kann man einen Suchbegriff eingeben. Dann werden nur noch die Steuerelemente angezeigt, die zu diesem Namen passen:



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.

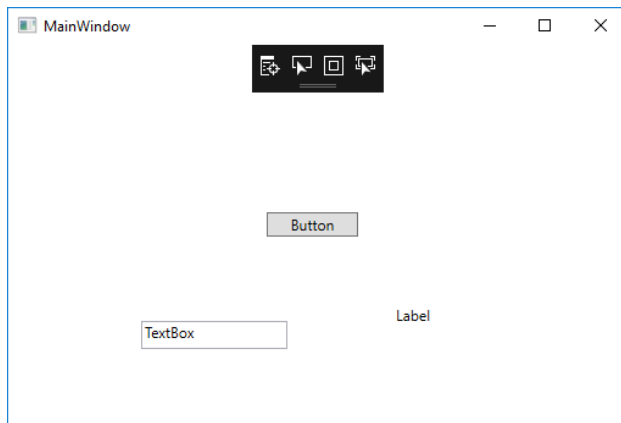
Beispiel: Nachdem man ein Label (Zeile neun in *Häufig verwendete WPF-Steuerelemente*, mit dem großen A), eine TextBox (letzte Zeile, Aufschrift *ab*) und einen Button (dritte Zeile) auf das MainWindow gesetzt hat, sieht Visual Studio etwa folgendermaßen aus:



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

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.

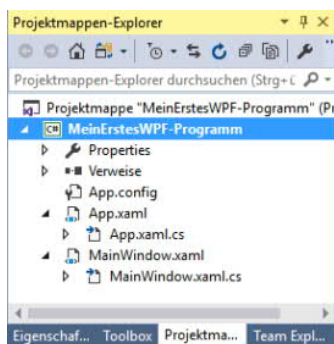
Vergessen Sie nicht, Ihr **Programm** zu **beenden**, bevor Sie es weiterbearbeiten. Sie können den Compiler nicht erneut starten, solange das Programm noch läuft.

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.

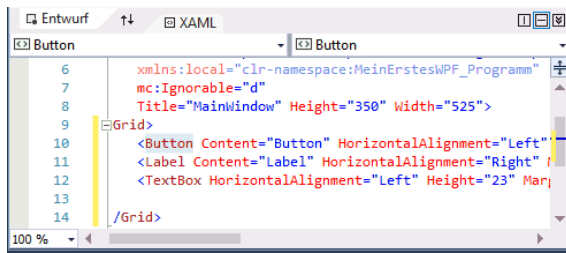
1.1.3 XAML-Grundkonzepte

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 („EXtensible Application Markup Language“, 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 letzten 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 Eigenschaftensfenster 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 Eigenschaftensfenster und der XAML-Editor

Das Aussehen und Verhalten eines Steuerelements ergibt sich aus seinen Eigenschaften und Ereignissen. Diese können im Eigenschaftensfenster (*Ansicht/Eigenschaftensfenster*, 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: