PROGRAMMER'S CHOICE

Empfohlen von:
dotnetpro

Frank Eller

Frank Eller

Visual C# 2008

Grundlagen, Programmiertechniken, Datenbanken

- Windows-Programmierung mit WPF und Windows Forms
- O/R-Mapping mit LINQ to SQL

ADDISON-WESLEY

90-Tage-Testversion
Visual Studio 2008 Professional

DVD CD

3 Das Visual Studio 2008

Das Visual Studio 2008 ist die optimale Entwicklungsumgebung für die Programmierung mit .NET bzw. mit C#. Die Möglichkeiten der Entwicklungsumgebung unterscheiden sich je nach verwendeter Version und sind derart umfangreich, dass sie in einem einzigen Kapitel keinesfalls vollständig beschrieben werden können. Daher geht es hier nur darum, die wichtigsten Elemente und ihre Verwendung hervorzuheben.

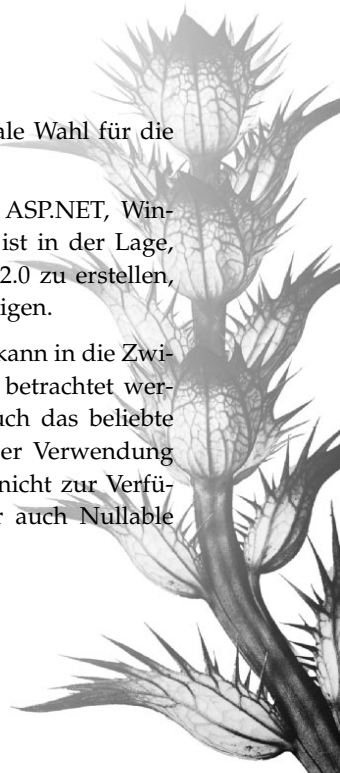
3.1 Einführung

Die Voraussetzungen an aktuelle Applikationen steigen, ebenso die Möglichkeiten und auch die Fehlerquellen. Die Zeiten, in denen ein einfacher Editor als Entwicklungsumgebung herhalten konnte, sind vorbei. Eine moderne IDE (*Integrated Development Environment* = Entwicklungsumgebung) hilft nicht nur bei der Codeeingabe, sondern stellt dem Entwickler eine Vielzahl von Tools und Möglichkeiten zur Verfügung, die ihm das Leben stark erleichtern.

3.1.1 Übersicht

Das Visual Studio ist aus zahlreichen Gründen die optimale Wahl für die Anwendungsentwicklung unter .NET:

- Das Visual Studio 2008 enthält visuelle Designer für ASP.NET, Windows.Forms-Applikation und WPF-Applikationen. Es ist in der Lage, Applikationen für alle .NET-Framework-Versionen ab 2.0 zu erstellen, sodass Sie nur noch eine Entwicklungsumgebung benötigen.
- Exceptions werden detailliert angezeigt, der Stacktrace kann in die Zwischenablage kopiert und dann in einem Editor genau betrachtet werden. Der Debugger unterstützt in vielen Bereichen auch das beliebte Edit&Continue, wobei dieses Feature allerdings bei der Verwendung einiger Sprachfeatures aus Implementierungsgründen nicht zur Verfügung steht. Das gilt beispielsweise für Generics oder auch Nullable Types, bei denen es sich ebenfalls um Generics handelt.



- Die IntelliSense-Hilfe ist bereits nach der Eingabe eines einzigen Buchstabens verfügbar. Es werden reservierte Wörter sowie Bezeichner von Klassen oder Datentypen angeboten; in der Regel springt IntelliSense direkt zum gewünschten (oder entsprechend dem Kontext sinnvollsten) Element.
- *SmartTags* helfen bei gängigen Operationen im Editor. Unter anderem können geänderte Variablennamen direkt für die gesamte Methode oder auch für eine komplette Klasse übernommen werden. Das hilfreichste Feature ist allerdings die Möglichkeit, über SmartTags benötigte Namespaces einzubinden. Ist der Name einer Klasse bekannt, aber nicht der Namespace, kann dieser über eine SmartTag-Anweisung eingefügt werden.
- Bekannte Datentypen werden im Editor farblich hervorgehoben. Damit können Sie jederzeit erkennen, ob alle benötigten Namespaces eingebunden sind, bzw. Sie sehen sofort, ob Sie sich vertippt haben. Für unterschiedliche Datentypen können Sie in den Optionen auch unterschiedliche Farben einstellen, falls Sie das möchten.
- Das Andocken der diversen Tool-Fenster des Visual Studio wird über sogenannte *Guides* (Ablageflächen) erleichtert.
- Integrierte Tools, beispielsweise *Code Snippets* zum Einfügen kleiner Codebestandteile oder auch Refactoring-Tools, erleichtern die Arbeit beim Programmieren. Integriert ist in den größeren Versionen des Visual Studio auch der *FxCop*, unverzichtbar wenn es darum geht, Code zu erzeugen, der uneingeschränkt aus anderen Programmiersprachen heraus verwendbar sein soll. Sie finden das Tool in den Projekteigenschaften unter der Bezeichnung *Code Analyse*. In der Professional- oder der Standard-Version ist das Tool nicht integriert, wird aber mitgeliefert. Sie finden die Installationsdatei auf Ihrer Festplatte unter *C:\Programme\Microsoft SDKs\Windows\v6.0A\FXCop*

3.1.2 Systemvoraussetzungen und Versionen

Systemvoraussetzungen

Microsoft gibt typischerweise recht konservative Systemvoraussetzungen für die Installation seiner Produkte vor, die nicht ganz der Realität entsprechen. Die Mindestvoraussetzungen für die Installation einer Visual Studio-Version sind laut Microsoft wie folgt:

- *Betriebssystem*: Windows XP (Home/Pro) mit Service Pack 2, Windows 2003 Server oder Windows Vista. Die Entwicklung funktioniert nicht mehr unter Windows 2000.
- *Hauptspeicher*: Mindestens 128 MB, empfohlen mindestens 256 MB
- *Festplatte*: Mindestens 2,5 GB freier Festplattenspeicher auf dem Installationslaufwerk, mindestens 1,2 GB freier Speicher auf dem Systemlaufwerk (also dem Laufwerk, auf dem Windows installiert ist). Das gilt für eine Installation inklusive MSDN. Da MSDN auch die Hilfefunktion beinhaltet, dürfte dies das Standardvorgehen sein.
- *Bildschirmauflösung*: Mindestens 800×600, empfohlen 1024×768
- Ein CD-/DVD-Laufwerk sowie (natürlich) eine Maus werden benötigt.

Diese Vorgaben sind sehr optimistisch. Zwar läuft das Visual Studio mit einer derart eingerichteten Umgebung, allerdings weder performant, noch ist ein sinnvolles Arbeiten möglich. Vor allem die Bildschirmauflösung ist offensichtlich scherzhaft gemeint, denn mit 800 × 600 Bildpunkten zu arbeiten, ist enorm frustrierend bis unmöglich.

Ein heute aktueller Computer liefert üblicherweise ausreichend Performance für ein flüssiges Arbeiten. Die nachfolgenden Vorgaben sind als empfehlenswert zu betrachten; liegen Ihre Systemdaten irgendwo zwischen dem absoluten Minimum (wie von Microsoft vorgegeben) und den empfehlenswerten Daten, so sind Sie ausreichend ausgerüstet:

- *Betriebssystem*: Windows XP (Home/Pro) mit Service Pack 2, Windows 2003 Server oder Windows Vista
- *Prozessor*: Ein P4 mit mindestens 1,5 GHz oder ein entsprechender AMD-Prozessor. Hier gilt: Je schneller, desto besser.
- *Hauptspeicher*: Mindestens 512 MB, besser 1024 MB oder gar 2048 MB
- *Festplatte*: Mindestens 4 GB freier Festplattenspeicher auf dem Installationslaufwerk, mindestens 2 GB freier Speicher auf dem Systemlaufwerk. Da Programme bei einer Standardinstallation üblicherweise auch auf dem Windows-Systemlaufwerk installiert sind, sollten dort also ca. 6 GB frei sein. Beachten Sie, dass zusätzlich auch noch Platz für die programmierten Applikationen vorhanden sein muss.
- *Bildschirmauflösung*: Mindestens 1280 × 1024 – je mehr, desto besser. Ab 1600 × 1200 Bildpunkten macht das Arbeiten Spaß, unterhalb von 1280 × 1024 wird es schnell frustrierend. 1024 × 768 gehen gerade noch so.

Visual Studio-Versionen

Die Entwicklungsumgebung ist in zahlreichen Versionen verfügbar. Am unteren Ende der Skala, als preisgünstigste weil kostenlose Alternative, stehen die Express-Versionen. Sie sind auf eine einzige Sprache bzw. eine einzige Technologie beschränkt. So gibt es Editionen für C#, Visual Basic und C++, die ausschließlich auf die Entwicklung von Windows-Applikationen ausgelegt sind.

Für den Webentwickler existiert ebenfalls eine Express-Version, die *Web Developer Edition*. Hier ist die Entwicklung auf ASP.NET beschränkt, dafür kann aber mit allen Programmiersprachen gearbeitet werden.

Den Einstieg in die Visual Studio-Linie bildet die Standard-Edition von Visual Studio .NET, gefolgt von der Professional-Version. Beide bieten bereits den Vorteil, mit einer beliebigen Programmiersprache arbeiten zu können. Was in der Hauptsache bei der Standard-Edition fehlt, sind die Crystal Reports-Steuerelemente für Auswertungen sowie die Möglichkeit, Setup-Projekte zu erstellen. Lediglich *ClickOnce* (siehe auch Abschnitt 29.5 ab Seite 1072) wird hier unterstützt. Ab der Professional-Version ist auch der SQL Server 2005 in der Developer-Edition enthalten.

Das Nonplusultra bilden die Team-Editionen des Visual Studio, zusammengefasst unter dem Oberbegriff *Team System*. Microsoft bietet hier für jeden das richtige Produkt an. Die Visual Studio-Editionen richten sich an den Softwarearchitekten, den Programmierer (Developer) sowie den Tester. Dementsprechend sind in den Versionen unterschiedliche Tools enthalten.

Die Basis bildet bei allen diesen Produkten die Professional-Edition von Visual Studio. Zusätzliche Tools ergeben sich lediglich für die entsprechende Rolle des Entwicklers. So wird der Softwarearchitekt Modeling-Tools erhalten, die an UML erinnern, aber mehr in die .NET-Richtung zielen; Der Softwaretester erhält die Möglichkeit, Testabläufe zu erstellen und durchzuführen. Diese Versionen sind die teuersten und umfangreichsten.

> > > HINWEIS

Das vorliegende Buch wurde mit der Professional-Edition des Visual Studio geschrieben, daher stammen auch alle Screenshots von dieser Version. Die Beispiele des Buches laufen aber auch mit jeder anderen Version des Visual Studio, insbesondere auch mit der Express-Edition von Visual C#. Eine Übersicht über die Features der verschiedenen Versionen finden Sie im Internet unter http://msdn2.microsoft.com/de-de/vs2008/products/bb980920(en-us).aspx (Professional und Standard) sowie http://msdn2.microsoft.com/de-de/vsts2008/products/bb991841(en-us).aspx (Team Editions).

3.2 Wichtige Fenster der Entwicklungsumgebung

Die Abbildungen in diesem Abschnitt stammen aus der Professional-Edition des Visual Studio. Die hier beschriebenen wichtigsten Fenster sind jedoch in allen Editionen enthalten und – was noch wichtiger ist – auch am gleichen Platz zu finden. Die Bedienung gestaltet sich in jeder Version des Visual Studio gleich.

Abbildung 3.1 zeigt das Visual Studio in der Standardeinstellung mit einem geladenen Windows.Forms-Projekt. Die wichtigsten Fenster sind sofort sichtbar. Links angedockt befindet sich die Toolbox, die alle verfügbaren Steuerelemente enthält. Auf der rechten Seite sehen Sie den Projektmappen-Explorer, der eine Übersicht über die im Projekt (oder in mehreren Projekten) enthaltenen Dateien liefert. Darunter finden Sie das Eigenschaftenfenster, eigentlich ein kombiniertes Eigenschaften-/Ereignisfenster, das Ihnen Zugriff auf die Eigenschaften eines Steuerelements bietet. Mithilfe dieser Eigenschaften ändern Sie unter anderem das Aussehen und Verhalten der Steuerelemente. Im unteren Bereich finden Sie die Fehlerliste, in der Sie nach einem erfolglosen Kompiliervorgang sämtliche aufgetretenen Compiler-Fehler finden. Diese Fenster finden Sie sowohl im WPF-Designer als auch im Windows.Forms-Designer. Allerdings macht Letzterer den deutlich unausgereifteren Eindruck; mehr darüber in Abschnitt 3.2.4 ab Seite 79.

Der Hauptarbeitsbereich (der mittlere Bereich des Visual Studio) ändert sein Aussehen je nach Erfordernis. Für das Design einer Form (im Buch auch häufig als Formular bezeichnet) wird hier der visuelle Designer eingeblendet. In ihm können Sie Steuerelemente auf der Form platzieren und ausrichten. Wird eine Textdatei (d. h. eine Datei mit Quellcode) geöffnet, befindet sich hier der Texteditor des Visual Studio, je nach Programmiersprache mit unterschiedlichen Möglichkeiten. Für andere Dateien, z. B. Ressourcen-Dateien, existieren noch weitere Ansichten. Im Falle einer WPF-Applikation wird entsprechend der WPF-Designer geöffnet.

Jedes geöffnete Dokument wird als Registerkarte am oberen Rand des Arbeitsbereichs abgelegt, wodurch ein schneller Zugriff möglich ist. Ebenso ist es natürlich auch möglich, über den Projektmappen-Explorer auf die Datei zuzugreifen. Ist die Datei bereits geöffnet, wird sie in den Vordergrund gebracht, ansonsten geöffnet.

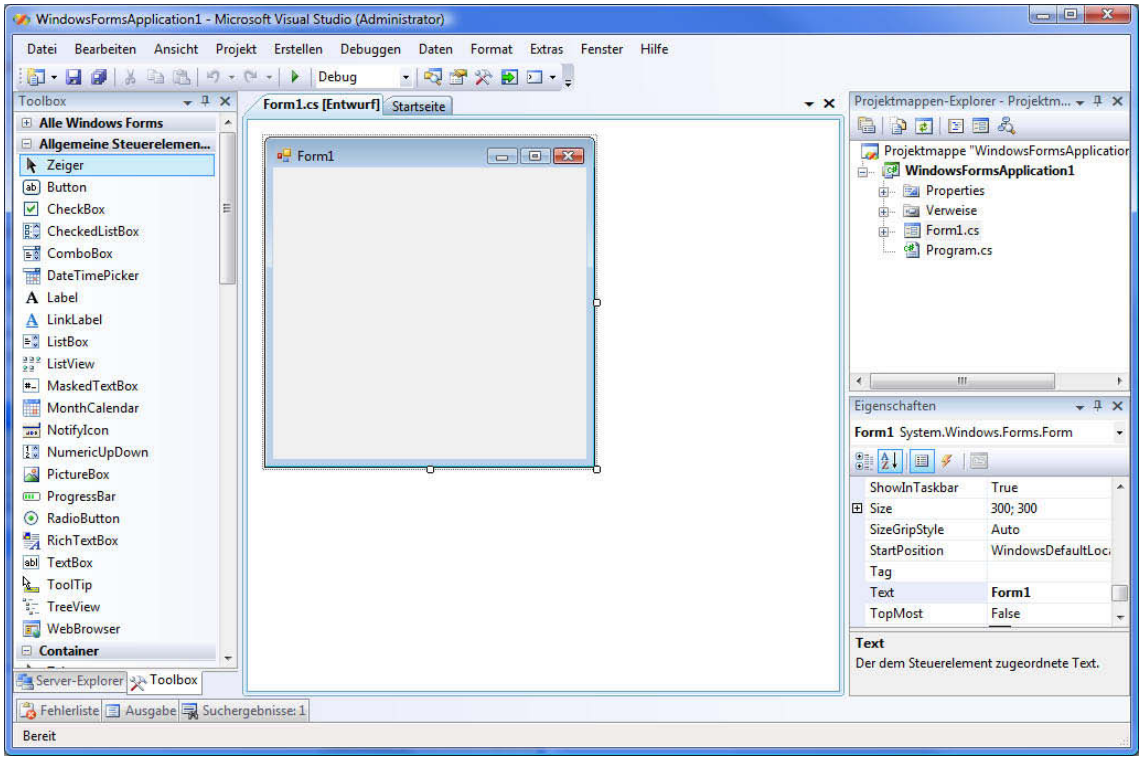


Abbildung 3.1: Das Visual Studio, Professional-Edition in der Standardeinstellung

Sie haben die Möglichkeit, diese Anzeige auf eine MDI-Anzeige (*Multiple Document Interface*) umzustellen. Die entsprechende Einstellmöglichkeit finden Sie in den Programmoptionen (EXTRAS | OPTIONEN) unter UMGEBUNG | ALLGEMEIN. Die Auswahl eines gewünschten Dokuments ist dann allerdings nur noch über den Projektmappen-Explorer bzw. das Menü FENSTER möglich; die Registerkarten sind da weit komfortabler.

Alle angedockten Fenster können, um die Arbeitsfläche (etwa bei einer niedrigen Bildschirmauflösung) zu vergrößern, auch ausgeblendet werden. Hierzu finden Sie rechts oben einen kleine Pin in jedem Fenster. Ist ein Fenster ausgeblendet, erscheint es als Schaltfläche an der Seite; wenn Sie die Maus darüber ziehen, wird das Fenster eingeblendet, verkleinert dann aber nicht den Arbeitsbereich, sondern überdeckt ihn.

Vor allem beim Gestalten der Anwendungsoberfläche ist es weit angenehmer, sämtliche benötigten Fenster ständig geöffnet zu haben. Aus diesem Grund auch die vom Autor als fast schon als Minimum angesehene Auflösung von 1028 × 1024 Bildpunkten. Als das Optimum hat sich die Arbeit mit zwei Bildschirmen herausgestellt; in diesem Fall können Sie die gesamte Arbeitsfläche nutzen und dennoch alle Toolfenster auf dem zweiten Bildschirm platzieren.

3.2.1 Der Projektmappen-Explorer

Im Projektmappen-Explorer verwalten Sie Ihre Projekte. Die Aufteilung in Projektmappe und Einzelprojekt ist wichtig, weil unter .NET jede Assembly auch ein Projekt ist. Innerhalb einer Gesamtanwendung wird also auch jede DLL (Projekttyp *Klassenbibliothek*) als Einzelprojekt angesehen. Die Projektmappe reflektiert dabei die Gesamtanwendung, die Projekte die jeweiligen DLLs sowie die ausführbare *.exe*-Datei.

Ordner = = Namespace

Doch der Projektmappen-Explorer ist mehr als eine einfache Anzeige der enthaltenen Dateien. Er zeigt außerdem die Ordnerstruktur innerhalb eines Projekts (d. h. im Projektverzeichnis) an, wobei ein Ordner automatisch einem Namespace entspricht. Namespaces sind eine (virtuelle) Unterteilungsmöglichkeit für Klassen innerhalb von Projekten. Sie haben dadurch die Möglichkeit, Ihre Klassen in Kategorien zu unterteilen, die sinnvoll die Verwendung der enthaltenen Klassen darstellen. Auch das .NET Framework ist so angelegt. Die Klassen für den Dateizugriff befinden sich beispielsweise im Namespace `System.IO` (wobei IO für *Input/Output* steht).

Virtuell ist diese Unterteilungsmöglichkeit deshalb, weil es sich bei Namespaces nicht um »physikalisch« vorhandene Klassen handelt. Es existiert keine Klasse namens `Namespace`, aus der heraus alle darin enthaltenen Datentypen ermittelt werden könnten. Vielmehr ist jedem Datentyp bekannt, in welchem Namespace er sich befindet.

Zwar reflektiert die Ordnerstruktur im Projektmappen-Explorer die vorgeschlagene Unterteilung, der Namespace, in dem sich eine Klasse befindet, wird aber in der Datei festgelegt, in der die Klasse programmiert ist. Dieser ist somit beliebig änderbar. Der Name des Ordners wird vom Visual Studio hier nur standardmäßig vorgegeben.

Der »Haupt-Namespace« eines Projekts entspricht dem Projektnamen. Alle weiteren Namespaces sind diesem Projektnamen untergeordnet (bzw. sollten es sein). Sie arbeiten am angenehmsten, wenn Sie sich den Automatismus des Visual Studio zunutze machen und die Ordnerstruktur innerhalb des Projektmappen-Explorers auch als Namespace-Struktur des Projekts annehmen.

> > > HINWEIS

Die Ordner, die in einem Projekt enthalten sein sollen, müssen entweder über das Kontextmenü des Projekts im Projektmappen-Explorer angelegt werden oder später hinzugefügt. Ein Ordner, den Sie im Dateisystem anlegen, ist nicht automatisch auch Bestandteil des Projekts. Das gleiche gilt für Dateien. Neue Klassen beispielsweise, die sich in der Regel auch immer in einer eigenen Datei befinden, sollten ebenfalls über den Projektmappen-Explorer hinzugefügt werden.

Aktives Projekt

Es liegt in der Natur der Sache, dass immer nur ein Projekt das Startprojekt sein kann (also dasjenige Projekt, das ausgeführt wird, wenn sie aus dem DEBUGGEN-Menü entweder STARTEN/ F5 oder STARTEN OHNE DEBUGGEN/ Strg + F5 auswählen). Das aktive Projekt ist immer fett dargestellt. Sie können es ändern, indem Sie das Kontextmenü eines nicht aktiven Projekts aufrufen und dort den Menüpunkt ALS STARTPROJEKT FESTLEGEN auswählen.

Durch einen Doppelklick auf eines der enthaltenen Elemente wird dieses in der Entwicklungsumgebung geöffnet oder, falls es schon geöffnet ist, in den Vordergrund gebracht. Die Datei, an der Sie gerade arbeiten, ist im Projektmappen-Explorer automatisch markiert.

Neue Elemente

Eine Applikation kann viele unterschiedliche Elemente beinhalten, zum Beispiel Formulare oder Klassen. Der Projektmappen-Explorer ist auch hierfür verantwortlich. Über einen Rechtsklick auf den Projektnamen oder einen im Projekt angelegten Ordner können Sie neue Elemente anlegen. Diese werden dann entweder innerhalb des Ordners (und damit innerhalb des korrespondierenden Namespaces) oder aber im Hauptbereich der Applikation (also als Bestandteil des Hauptnamespaces) angelegt.

Verweise auf DLLs

Um die Klassen innerhalb einer DLL verwenden zu können, müssen Sie diese referenzieren bzw. den Verweisen des Projekts hinzufügen. Jedes Projekt besitzt dazu einen Ordner namens *Verweise*. Die wichtigsten DLLs (je nach Projekttyp) sind bereits eingefügt, aber falls Ihr Projekt komplexer wird, ist es sehr wahrscheinlich, dass Sie weitere DLLs benötigen werden. Über das Kontextmenü des *Verweise*-Ordners, Menüpunkt VERWEIS HINZUFÜGEN, können Sie über einen Dialog weitere DLLs hinzufügen.

Dabei stehen sowohl sämtliche DLLs des .NET Frameworks zur Auswahl als auch COM-Komponenten bzw. die DLLs, die Bestandteil Ihrer Projektmappe sind. Der Dialog stellt auch eine Registerkarte DURCHSUCHEN zur Verfügung, über die Sie auch DLLs hinzufügen können, die nicht in einer der Listen auftauchen (z. B. DLLs von Drittanbietern). Außerdem merkt sich das Visual Studio, welche DLLs Sie zuletzt hinzugefügt haben, und bietet diese ebenfalls unter einer eigenen Registerkarte an. Den Dialog sehen Sie in Abbildung 3.2.

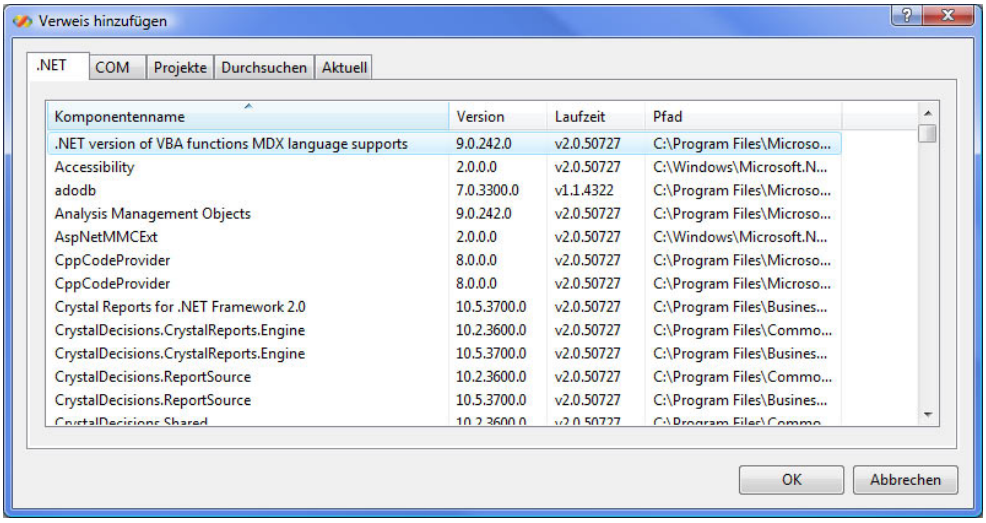


Abbildung 3.2: Der Dialog zum Hinzufügen von Verweisen

Über das gleiche Kontextmenü ist es auch möglich, einen sogenannten *Webverweis* hinzuzufügen. Dabei handelt es sich um Web Services. Sie können also mithilfe des Visual Studio sehr einfach Web Services konsumieren und erstellen.

> > > HINWEIS

Ein kurzer Hinweis zum Aufbau des .NET Frameworks: Die Klassen von .NET sind in DLLs organisiert. Die absolute Basisfunktionalität befindet sich in der Datei mscorlib.dll, die immer automatisch eingebunden ist – ohne sie wäre überhaupt nichts möglich. Die übrigen DLLs, die Verwendung finden können, tragen per Konvention den Namen des enthaltenen (Haupt-)Namespaces. Die Klassen für die Windows.Forms-Programmierung, die im Namespace `System.Windows.Forms` *angesiedelt sind, befinden sich demnach in der DLL System.Windows.Forms.dll.*

» » » VERWEIS

Detaillierte Informationen über das Arbeiten mit dem Projektmappen-Explorer erhalten Sie über die Hilfe, indem Sie nach dem Stichwort »Projektmappen-Explorer« suchen. Der direkte Link lautet:

ms-help://MS.VSCC.v90/MS.MSDNQTR.v90.de/dv_vbcnexpress/html/5a7acfe2-b342-4d82-acf7-275ff8b1f51d.htm

3.2.2 Die Toolbox

Der Aufbau einer Windows.Forms-Anwendung erfolgt über Steuerelemente, die sowohl visuell sein können (beispielsweise eine `TextBox` oder ein `Label`) oder nicht visuell (wie zum Beispiel die Dialoge – diese werden erst zur Laufzeit angezeigt, sind aber im Formular nicht sichtbar).

Zugriff auf alle diese Steuerelemente erhalten Sie über die *Toolbox*. Sie ist in Kategorien angeordnet und arbeitet kontextabhängig. Wenn Sie also ein Dokument im Texteditor geöffnet haben, finden Sie darin keine Steuerelemente; ist der visuelle Designer geöffnet, zeigt die Toolbox die verfügbaren Steuerelemente an.

Die Toolbox besitzt ein Kontextmenü, mit dem Sie eigene Kategorien oder auch weitere Steuerelemente, beispielsweise von Drittanbietern oder auch solche, die Sie selbst erstellt haben, hinzufügen können. Hierzu wählen Sie aus dem Kontextmenü den Eintrag ELEMENTE AUSWÄHLEN.

Sollte Ihnen die Anordnung in Kategorien nicht zusagen, ist das auch kein Problem. Die Toolbox enthält auch eine Kategorie, in der alle Steuerelemente für Windows.Forms enthalten sind. Dennoch ist die Einteilung in Kategorien vorteilhaft. Sie finden einfach schneller das Steuerelement bzw. diejenige Komponente, die Sie suchen.

» » » VERWEIS

Weitere Informationen über die Toolbox finden Sie in der integrierten Hilfe, indem Sie nach »Toolbox« suchen. Der direkte Link lautet:

ms-help://MS.VSCC.v90/MS.MSDNQTR.v90.de/dv_vsctrlcomp/html/b754dad3-1f32-464f-8b9f-065e17e0bc22.htm

3.2.3 Das Eigenschafts-/Ereignisfenster

Die Einstellungen der Komponenten und Steuerelemente auf dem Formular erfolgen über Eigenschaften. Über diese steuern Sie sowohl das Aussehen als auch in vielen Fällen das Standardverhalten eines Steuerelements. Das Eigenschaftsfenster ist daher dasjenige Fenster, das am häufigsten zum Einsatz kommt. Die wichtigsten Einstellungen ein Steuerelement betreffend können Sie ab dieser Version des Visual Studio auch direkt im Designer über das Aufgabenmenü vornehmen, das bei vielen Steuerelementen erscheint.

Auch die Einträge im Eigenschaftsfenster sind in Kategorien angeordnet. Wesentlich schneller zu erreichen sind die Eigenschaften hier aber über die alphabetische Anordnung, da der Name (oder wenigstens der ungefähre Name) einer Eigenschaft in der Regel bekannt ist. Sie können die Anordnung über den zweiten Button von links in der Toolbar des Eigenschaftsfensters umstellen. Der erste Button steht für die kategorisierte Darstellung.

Ereignisse

Applikationen unter Windows sind nicht an einen festen Ablauf gebunden. Stattdessen arbeiten sie mit Ereignissen, reagieren also auf die Aktionen des Benutzers. Auch die einzelnen Steuerelemente besitzen verschiedene Ereignisse. – und nicht nur diese. Sie selbst können Klassen schreiben, die Ereignisse verwenden bzw. auf Ereignisse anderer Klassen reagieren.

Das Eigenschaftsfenster ist eigentlich ein kombiniertes Fenster, denn es erlaubt auch den Zugriff auf die Ereignisse, die ein Steuerelement auslösen kann. Hierfür finden Sie in der Toolbar des Eigenschaftsfensters einen entsprechenden Button (der vierte von links, mit dem Blitz).

Um nun auf ein Ereignis zu reagieren, gibt es mehrere Möglichkeiten.

- Schreiben Sie in das Feld neben dem Ereignisnamen einfach den Namen der gewünschten Methode, die aufgerufen werden soll, wenn das Ereignis eintritt. Falls die Methode noch nicht existiert, wird sie vom Visual Studio automatisch angelegt.
- Klicken Sie doppelt auf den Ereignisnamen oder auf das Auswahlfeld daneben. Das Visual Studio erzeugt daraufhin einen Namen für die Ereignismethode und fügt diese in den Programmcode ein.
- Falls bereits mehrere Ereignisbehandlungsroutinen existieren, können Sie über das Auswahlfeld neben dem Ereignisnamen auch eine dieser Methoden auswählen und sie dem Ereignis zuordnen.

3.2.4 Der WPF-Designer

Wenn Sie ein WPF-Projekt starten, finden Sie die gleichen Fenster wie bei Windows.Forms, allerdings macht der Designer insgesamt einen deutlich weniger ausgereiften Eindruck. Unter WPF wird mit XAML-Code gearbeitet. Der Designer ist in der Standardansicht daher zweigeteilt, wie es Abbildung 3.3 zeigt.

Sie können die Art der Ansicht wählen. Sollten Sie eher der Typ sein, der visuell arbeiten möchte, können Sie die Designansicht als Hauptansicht wählen. Wenn Sie mit XAML arbeiten wollen, können Sie in den Optionen auch diese Ansicht als Standard einstellen. Am unteren Rand des Designers finden Sie Tabs, mit denen Sie die Ansicht umstellen.

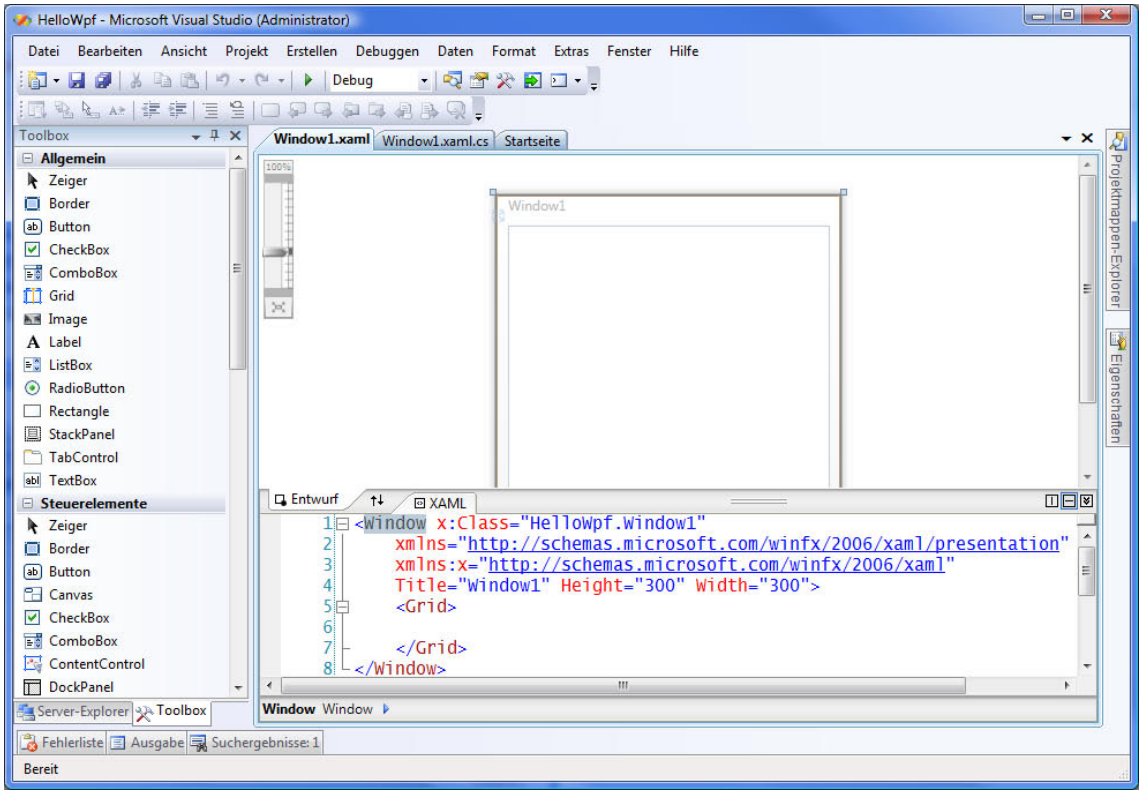


Abbildung 3.3: Der WPF-Designer in der Standardansicht

Das Fenster (die Basisklasse heißt `Window`, nicht mehr `Form`) das Sie designen, wird im Unterschied zum Windows.Forms-Designer mittig angezeigt und kann über einen Regler an der linken Seite vergrößert oder verkleinert angezeigt werden. Dabei handelt es sich um eine Zoom-Funktion, d. h. die Abmessungen des Fensterobjekts werden nicht verändert.

Der Projektmappen-Explorer

Der Projektmappen-Explorer ändert sich nicht, was seine Funktionsweise oder den Aufbau angeht. Er beinhaltet die gleiche Funktionalität wie unter Windows.Forms.

Die Toolbox

Ebenfalls in Kategorien aufgeteilt, erscheint die WPF-Toolbox. Leider waren die Entwickler bei Microsoft hier nicht so sorgfältig wie bei der Windows.Forms-Toolbox. Es gibt zwei Kategorien: eine für häufig verwendete Steuerelemente sowie eine für alle Steuerelemente. Leider hat man es hier versäumt, eine saubere Aufteilung vorzunehmen, was dem Entwickler das Leben stark erleichtert hätte.

Was auffällt, ist übrigens, dass einige Komponenten, die unter Windows.Forms noch vorhanden sind, hier fehlen. Beispielsweise die Dialoge. Diese sind natürlich im .NET Framework vorhanden, allerdings werden sie nicht mehr in der Toolbox angezeigt, weil der WPF-Designer keinen Bereich mehr für Komponenten besitzt, die keine visuelle Repräsentation besitzen.

> > > HINWEIS

Nach dem ersten Start des Visual Studio fällt auf, dass sich Steuerelemente aus der Toolbox manchmal nicht auf ein Fenster ziehen lassen, auch wenn das eigentlich möglich sein sollte. Nach einem Neustart hat dann alles funktioniert. Sollte das auch bei Ihnen der Fall sein, einfach noch mal neu starten.

Das Eigenschaftenfenster

Das Eigenschaftsfenster, eigentlich eines der wichtigsten Fenster bei der Entwicklung von Windows-Applikationen, macht unter WPF nicht den besten Eindruck. Nett ist sicherlich die Tatsache, dass in einer kleinen Box oben links innerhalb des Eigenschaftenfensters eine kleine Vorschau des gerade markierten Elements angezeigt wird – und zwar immer im aktuellen Design. Da die gleiche Ansicht auch der Designer zeigt, ist das aber eher ein Gimmick.

Negativer fällt da das Fehlen einer Schaltfläche zum Umschalten auf Ereignisse auf. Diese können schlichtweg nicht angezeigt werden, was eigentlich eine Frechheit ist. In Expression Blend, dem von Microsoft bevorzugten Werkzeug für das Design von WPF-Anwendungen und –Steuerelementen, geht das nämlich schon lange. Es wäre schön gewesen, wenn die Entwickler bei Microsoft ein so wichtiges Detail berücksichtigt hätten.

> > > HINWEIS

Da sich Ereignisse nicht im visuellen Designer festlegen lassen, müssen diese in der Regel direkt im XAML-Code festgelegt werden. Standardereignisse der Steuerelemente können durch einen Doppelklick auf dieselben eingefügt werden.

Der Name der gewünschten Ereignisbehandlungsroutine wird normalerweise automatisch entsprechend dem von Microsoft seit Jahren verwendeten Schema generiert (die IntelliSense-Hilfe zeigt einen entsprechenden Eintrag zum Erstellen eines neuen Eventhandlers). Aber auch eigene Bezeichnungen können angegeben werden. Beim Umschalten in den Code fällt dann allerdings auf, dass die Ereignisbehandlungsroutine nicht generiert wurde.

Abhilfe schafft hier ein Rechtsklick auf den Namen des Eventhandlers in XAML. Das Kontextmenü enthält einen Menüpunkt, mit dem Sie zum Ereignis-Handler navigieren können. Wenn Sie diesen aufrufen, wird die entsprechende Methode erstellt, und Sie springen direkt dorthin. Nach wie vor umständlich, aber wenigstens geht es.

Schön wäre es auch, wenn die Eigenschaften wie unter Windows.Forms alphabetisch angeordnet werden könnten. Diese Möglichkeit ist ebenfalls nicht enthalten, dafür besitzt das Eigenschaftenfenster ein Suchfeld, wobei immer diejenigen Eigenschaften angezeigt werden, die die darin eingegebenen Buchstaben im Namen tragen. Alles in allem ist dieses Eigenschaftenfenster wenig ausgereift und mit ein Grund, warum der Autor fast nicht im Design-Modus programmiert, sondern die benötigten Elemente direkt in XAML schreibt. Abbildung 3.4 zeigt das Eigenschaftenfenster des WPF-Designers.

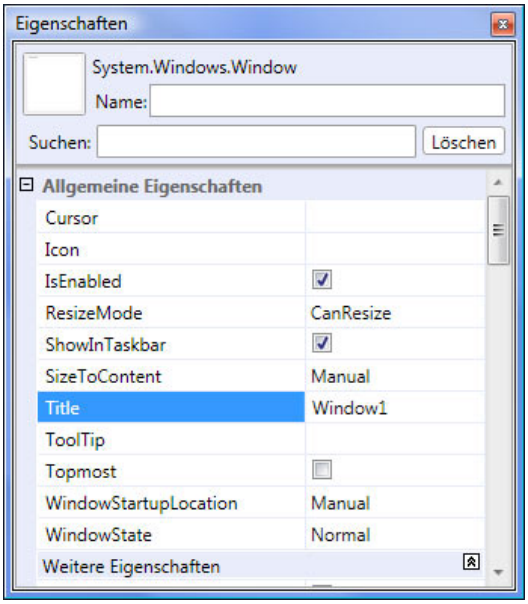


Abbildung 3.4: Das Eigenschaftenfenster des WPF-Designers

3.2.5 Die Projekteigenschaften

Alle Einstellungen für ein Projekt lassen sich komfortabel im mittleren Arbeitsbereich der Entwicklungsumgebung einstellen. Markieren Sie hierzu das Projekt (nicht die Projektmappe, sondern das Projekt, dessen Einstellungen Sie ändern wollen) und wählen Sie aus dem Menü PROJEKT den Menüpunkt EIGENSCHAFTEN. Alternativ können Sie auch den entsprechenden Button im Projektmappen-Explorer anklicken (oben ganz links). Abbildung 3.5 zeigt die neu gestalteten Projekteigenschaften.

Die Projekteigenschaften sind in mehrere Tabs aufgeteilt.

- ANWENDUNG ist der Bereich, in dem Sie grundlegende Eigenschaften bezüglich des aktuellen Projekts einstellen können. Dazu gehören beispielsweise der Name der generierten Assembly, der Standardnamespace, das Zielframework oder die Art der Applikation. Unter ASSEMBLYINFORMATIONEN finden sie weitere Einstellmöglichkeiten. diese können Sie auch manuell vornehmen - im Ordner *Properties* im Projektmappen-Explorer finden Sie die Datei *AssemblyInfo.cs*, die mehrere applikationsweite Einstellungen enthält.
- ERSTELLEN ist der Bereich, in dem Sie steuern können, wie sich der Compiler verhält. Dort haben Sie die Möglichkeit, unverwalteten (unsicheren) Code zuzulassen oder zu verweigern sowie die Zielplattform zuzuweisen oder die Optimierung des Codes ein- bzw. auszuschalten.

  Ebenfalls enthalten ist eine Einstellung für die Warnstufe. Je höher Sie diese einstellen, desto genauer die Ausgabe in der Fehlerliste. Falls gewünscht können Sie Warnungen des Compilers auch als Fehler behandeln. Sie können den Ausgabepfad angeben sowie eine XML-Dokumentationsdatei generieren lassen (aus den im Code eingetragenen Dokumentationskommentaren). Eine richtige Hilfedatei ist das allerdings nicht, dazu müssen Sie das erzeugte XML erst noch in eine lesbare Form konvertieren.

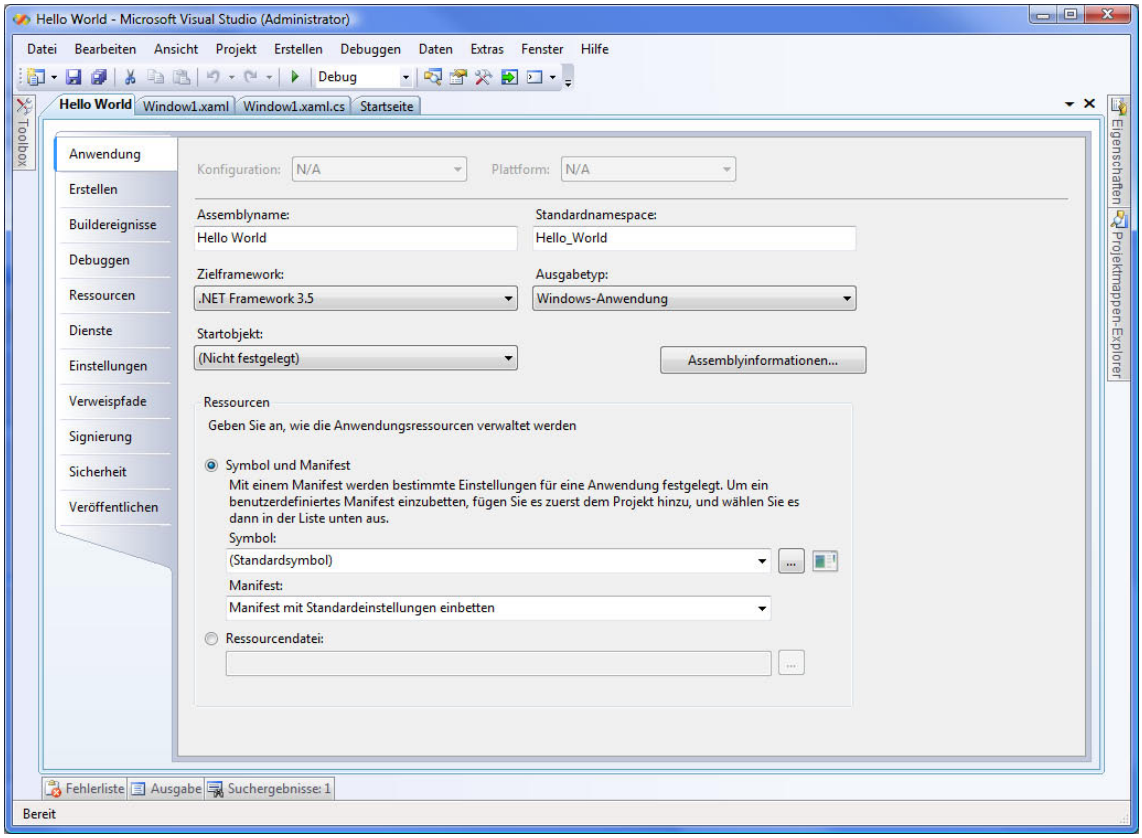


Abbildung 3.5: Die Projekteigenschaften im Visual Studio 2008

- BUILDEREIGNISSE sind bei der normalen Anwendungsentwicklung nicht so wichtig. Hier können Sie im Stile der guten alten Batch-Dateien Befehle eingeben, die entweder unmittelbar vor der Kompilierung oder danach ausgeführt werden. Falls gewünscht werden diese Ereignisse immer ausgeführt oder auch nur bei erfolgreichem Kompiliervorgang.
- Im Bereich DEBUGGEN können Sie einige Einstellungen vornehmen, die Ihnen das Testen oder die Fehlersuche erleichtern. Beispielsweise können Sie ein externes Programm starten, Kommandozeilenparameter übergeben oder auch das Arbeitsverzeichnis festlegen.
- Der Bereich RESSOURCEN ermöglicht es Ihnen, Ressourcen anzulegen. Sie können alle Arten von Ressourcen unterbringen, z. B. Grafiken, Texte, Mediendateien und vieles andere mehr. Der Zugriff innerhalb des Programms erfolgt über die Klasse Resources, die vom Visual Studio verwaltet und mit jeder neuen Ressource entsprechend erweitert wird.
- Der Bereich DIENSTE ist nue im Visual Studio 2008. Hiermit können Sie die Authentifizierungsmöglichkeiten von ASP.NET unter Windows nutzen. Der enorme Vorteil dieser Variante ist, dass praktisch alles bereits eingebaut ist und funktioniert, sozusagen eine Built-In-Benutzerverwaltung.

- Der Bereich EINSTELLUNGEN bezeichnet Programmeinstellungen. Derartige Einstellungen können Sie zur Laufzeit vornehmen und sie benutzerspezifisch speichern (das System macht das automatisch, Sie müssen lediglich das Speichern anstoßen). Somit haben Sie hier die Möglichkeit, jedem Benutzer seine eigenen Einstellungen zu geben.
- Der Bereich VERWEISPFADE ermöglicht Ihnen, anzugeben, in welchen Verzeichnissen das Visual Studio nach den DLLs suchen soll, die in Ihrem Projekt eingebunden sind.
- Der Bereich SIGNIERUNG ist wichtig, wenn Sie einer DLL einen starken Namen geben wollen. Darin können Sie eine Schlüsseldatei erzeugen lassen und die Signierung des Projekts veranlassen. Signierte Dateien haben einen starken Namen und sind somit vom .NET-System eindeutig identifizierbar.
- Die Bereiche SICHERHEIT und VERÖFFENTLICHEN haben mit ClickOnce zu tun, einer mit dem Visual Studio 2005 eingeführten Art der Verteilung/Installation von Anwendungen.

3.3 Der Editor

Der Texteditor des Visual Studio unterstützt Sie durch zahlreiche kleine Hilfestellungen, die oft im Verborgenen liegen. Er unterstützt Syntaxhervorhebung, lässt sich beliebig anpassen, liefert Informationen über die letzten Änderungen innerhalb der aktuellen Visual Studio-Session und beinhaltet eine Bookmark-Verwaltung, mit deren Hilfe Sie auch in komplexen und umfangreichen Anwendungen schnell zu einem bestimmten Punkt innerhalb Ihres Quelltextes springen können.

3.3.1 Anpassung des Editors

Vor allem in C-basierten Sprachen gibt es immer wieder das Problem, dass jeder Programmierer seinen Code anders formatiert. Wenn Sie einen Codeabschnitt nehmen und ihn fünf verschiedenen Programmierern geben, mit der Vorgabe, ihn zu formatieren, so werden Sie ziemlich sicher mindestens drei verschiedene Ergebnisse erhalten.

Um diesem Manko aus dem Weg zu gehen, können Sie im Visual Studio einstellen, wie der Code formatiert werden soll. Unter anderem können sie festlegen, ob die öffnende geschweifte Klammer in der gleichen Zeile stehen soll wie ein Methodenkopf, ob es zwischen leeren Klammern beim Methodenaufruf ein Leerzeichen geben soll, ob Deklarationen in einer Zeile stehen sollen oder getrennt, und so fort.

Diese große Menge an Einstellungen nehmen Sie im Optionsdialog des Visual Studio vor (Menüpunkt EXTRAS | OPTIONEN). Wählen Sie dort den Eintrag TEXT-EDITOR | C# | FORMATIERUNG. Abbildung 3.6 zeigt den Dialog.

Ganz gleich, welche Datei Sie öffnen, Sie können sicher sein, dass der Inhalt (falls es sich um eine C#-Datei handelt) genau so formatiert und angezeigt wird, wie Sie es vorgeben. Formatierungen sind möglich im Bezug auf Einzüge, Zeilenwechsel, Leerzeichen und Umbrüche. Zum Neuformatieren eines Dokuments verwenden Sie die Tastenkombination Strg+K, Strg+D.

Das Einstellen der Textfarbe und Schriftart ist ebenfalls möglich. Während die Formatierungseinstellungen sich allerdings auf eine Sprache beschränken, sind die Einstellungen für die Schriftart global für alle Sprachen und damit auch unter UMGEBUNG | SCHRIFTARTEN UND FARBEN zu finden.

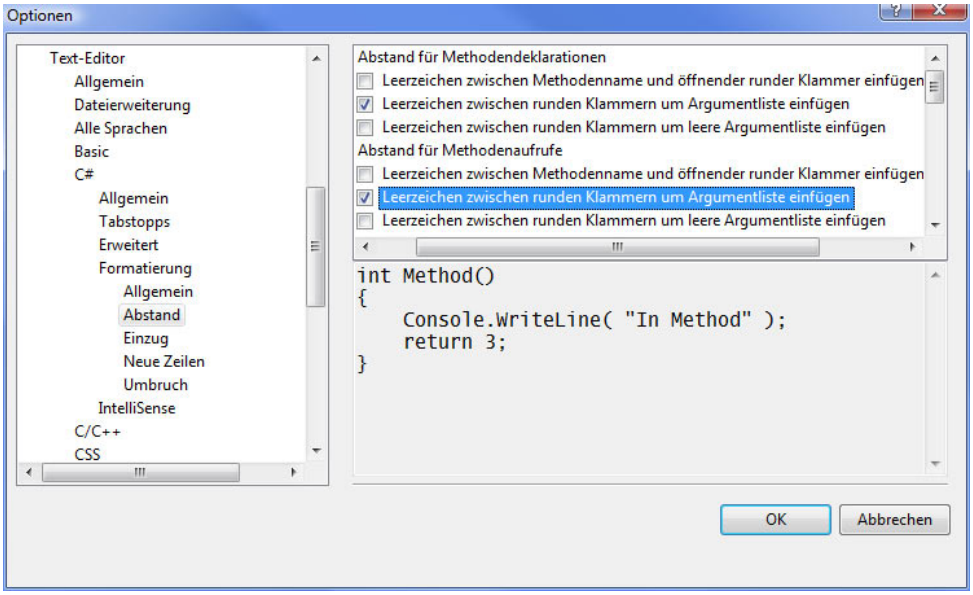


Abbildung 3.6: Der Dialog für die Formatierungsoptionen

> > > HINWEIS

Leider formatieren Entwickler nicht nur unterschiedlich, sie haben auch unterschiedliche Vorgehensweisen und benennen ihre Variablen ebenso unterschiedlich. Wenn es darum geht, im Team zu arbeiten, sollte der erste Schritt immer die Festlegung sogenannter Code-Conventions sein, die die Benennung von Programmelementen, die Formatierung und allgemeingültige Vorgehensweisen bei der Programmierung für das gesamte Team festlegen.

3.3.2 IntelliSense

Die IntelliSense-Hilfe schlägt bereits nach dem ersten eingegebenen Buchstaben passende Wörter vor. Dabei bezieht sie auch bekannte Klassennamen und reservierte Wörter mit ein.

3.3.3 SmartTags

Der Visual Studio-Editor unterstützt *SmartTags*, die Sie auch aus den Office-Produkten kennen. So ermöglicht Ihnen Visual Studio direkt nach dem Umbenennen eines Bezeichners, diese Änderung für das gesamte Projekt zu übernehmen (es wird ein rotes SmartTag am Ende des Bezeichners angezeigt). Auch wenn Ihnen der Name eines Datentyps zwar bekannt ist, nicht aber der Namespace, in dem er sich befindet, hilft ein SmartTag und gibt Ihnen die Möglichkeit, entweder den Namen voll zu qualifizieren oder den benötigten Namespace einzubinden. Abbildung 3.7 zeigt diese Möglichkeit. Voraussetzung dafür ist allerdings, dass die DLL, in der sich der Datentyp befindet, referenziert ist.

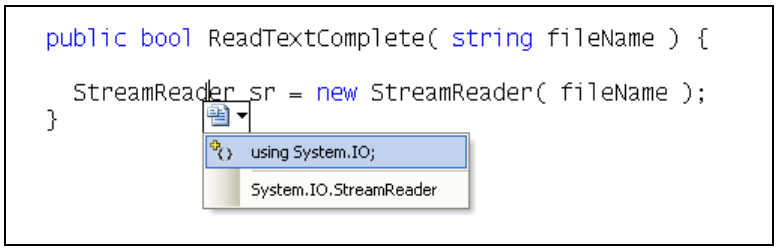


Abbildung 3.7: SmartTags in Visual Studio

Eine weitere Hilfe finden Sie bei Interfaces oder abstrakten Basisklassen. Dort sind Sie gezwungen, vorgegebene Methoden zu implementieren. Über ein SmartTag fügt das Visual Studio alle benötigten Methodenrümpfe ein.

3.3.4 Änderungen innerhalb einer Sitzung

Komfortabel ist die neue Möglichkeit, die gemachten Änderungen zu verfolgen. Diese Änderungsverfolgung bezieht sich grundsätzlich auf eine Sitzung mit dem Visual Studio, d. h., wenn Sie das Visual Studio schließen, werden Sie später nicht mehr feststellen können, welche Änderungen Sie vorgenommen haben. Innerhalb einer Sitzung ist es allerdings recht angenehm zu sehen, wo Änderungen durchgeführt wurden bzw. welche Änderungen bereits gespeichert wurden.

Wenn Sie Quellcode ändern, wird an der linken Seite des Editors ein farbiger Balken angezeigt, der während der gesamten Sitzung bestehen bleibt. Bei »frischen« Änderungen, die noch nicht abgespeichert sind, ist dieser Balken gelb. Haben Sie gespeichert, wird er grün dargestellt. Damit können Sie auch unter den gemachten Änderungen unterscheiden. Leider können Sie nicht nach dem Auftreten solcher Änderungsbalken suchen.

3.3.5 Refactoring

Refactoring, also das Verbessern des Quellcodes, ist ein ständiger Vorgang. Jeder ernsthafte Programmierer ist immerzu dabei, Refactorings durchzuführen. Was dieser Begriff bedeutet, ist schnell erklärt.

Beim Programmieren geht es meist zuerst einmal darum, dass etwas funktioniert. Häufig tritt danach die unbefriedigende Situation auf, dass der Quelltext nicht »sauber« ist, beispielsweise weil der gleiche Code mehrfach verwendet wurde (Copy&Paste), weil Methoden zu lang wurden (und damit unüberschaubar), oder weil man feststellt, dass eine gewisse Funktionalität doch besser in einer eigenen Klasse Platz gefunden hätte.

Was auf diese Erkenntnis folgt, ist in der Regel der entsprechende Umbau. Mehrfach verwendeter Code wird in eine eigene Methode ausgelagert, die dann aufgerufen wird. Unüberschaubare Methoden werden zerteilt und somit überschaubar (und wartbar). Für eine umfangreiche Funktionalität wird eine eigene Klasse erzeugt usw.

Dieses Vorgehen nennt man *Refactoring*. Das Umbauen des Quelltextes, sodass er danach leichter lesbar und damit auch leichter wartbar ist. Das Visual Studio hilft bei diesen Vorgehensweisen, indem es zahlreiche Refactorings zur Verfügung stellt. Unter anderem sind darunter das Kapseln eines Feldes in einer Eigenschaft, die Extraktion einer Methode, die Extraktion von Interfaces, das Umbenennen von Klassen (wobei auch die Referenzen projektweit geändert werden) oder auch das Vertauschen von Parametern einer Methode. All das ist sowohl in den Visual Studio-Editionen als auch in der Visual C# Express-Edition enthalten.

» » » VERWEIS

Das Standardwerk zum Thema Refactoring ist das gleichnamige Buch von Martin Fowler. Auf Deutsch ist es bei Addison-Wesley (ISBN: 3-8273-2278-2) erschienen. Obwohl für absolute Einsteiger noch etwas schwierig, ist es doch schon für den fortgeschrittenen Programmierer zu empfehlen. Nach der Lektüre dieses Buches werden Sie definitiv besseren Code schreiben.

3.4 Tools und Hilfsmittel

Nicht alles, was das Visual Studio an Tools bietet, kann hier angesprochen werden. Zwei der Tools sollen im Vordergrund stehen, da sie enorm nützlich sind: Der Klassendesigner (Klassendiagramm) und die *Object Test Bench*, in der deutschen Version als »Objekttestcenter« übersetzt. Beide Tools sind nicht in den Express-Versionen, aber ab der Standard-Version des Visual Studio verfügbar.

3.4.1 Das Klassendiagramm

Die meisten Programmierer schreiben den Code für ihre Klassen selbst. Andere hätten gerne eine Möglichkeit, die Klasse erst zu designen (d. h. vorzugeben, welche Methoden, Eigenschaften und Felder enthalten sind) und erst später die Funktionalität hinzuzufügen. Die klassische Vorgehensweise ist dabei UML. Allerdings bedarf es teurer Tools, um die mittels UML designten Klassen auch in Quellcode umzusetzen.

Das Visual Studio beinhaltet ein Tool namens *Klassendiagramm*, mit dem genau dies eingeschränkt möglich ist. Ähnlich wie UML können hier Klassen und Interfaces zusammengestellt werden. Das Visual Studio kümmert sich darum, entsprechende Dateien und die Funktionsrümpfe anzulegen. Die Anzeige des Klassendesigners und der Code in der entsprechenden Datei bleiben dabei absolut synchron.

Auch aus bestehenden Applikationen heraus können Sie ein Klassendiagramm erzeugen. Wählen sie einfach aus dem Kontextmenü des Projekts den Eintrag KLASSENDIAGRAMM ANZEIGEN, und es wird automatisch erstellt. Danach können Sie Ihre Klassen »gestalten«, statt sie zu programmieren. Die Funktionalität müssen Sie allerdings selbst hinzufügen.

Abbildung 3.8 zeigt das Klassendiagramm des Visual Studio. Dort wird auch die Kontextsensitivität der Toolbox deutlich. In dieser Ansicht (ebenfalls nur eine Ansicht des Hauptarbeitsbereichs) zeigt sie die möglichen Elemente, die hinzugefügt werden können. Dazu gehören unter anderem Klassen, aber auch Strukturen (`struct`), Aufzählungen (`enum`), Interfaces und abstrakte Klassen.

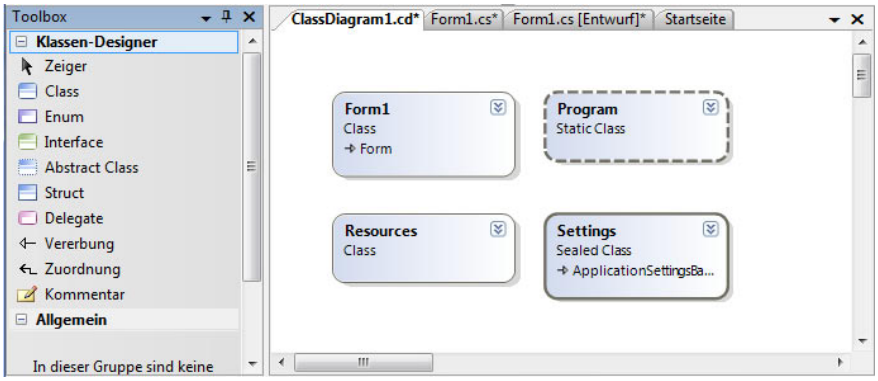


Abbildung 3.8: Der Klassendesigner des Visual Studio. Die Toolbox zeigt die verfügbaren Elemente.

3.4.2 Das Objekttestcenter

Software ist erst fertig, wenn sie getestet ist. Je umfangreicher die Software ist, desto umfangreicher müssen auch die Tests sein. Mittlerweile wurde dies (glücklicherweise) erkannt, und Tools für das Testen existieren in großer Anzahl. Auch das Visual Studio kommt in einer Version für Tester daher.

Das Objekttestcenter ermöglicht zwar keine umfangreichen Unittests, wie es die speziell dafür vorgesehenen Tools machen, allerdings können Sie damit kontrollieren, ob sich ein Objekt so verhält, wie Sie das vorgesehen haben, und ob die darin enthaltenen Methoden die gewünschten Ergebnisse liefern. Zugriff darauf erhalten Sie aus dem Klassendesigner oder aus der Klassenansicht (ANSICHT | KLASSENANSICHT, Strg + ⇧ + C). Über das Kontextmenü einer Klasse können Sie diese instanziieren (INSTANZ ERSTELLEN), was in der Anzeige des Objekttestcenters und eines erzeugten Objekts resultiert. Mit diesem Objekt können Sie arbeiten, d. h., sämtliche Methoden des Objekts aufrufen oder auch Werte zuweisen. Abbildung 3.9 zeigt das Objekttestcenter mit einem erzeugten Objekt, von dem eine Methode aufgerufen wird.

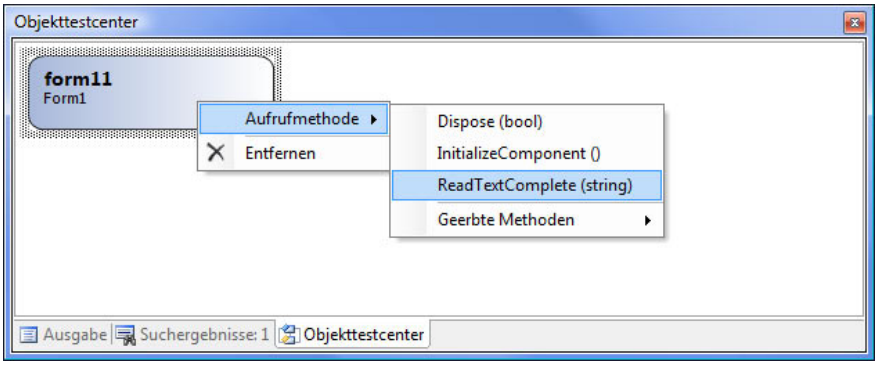


Abbildung 3.9: Das Objekttestcenter des Visual Studio

Da hier eine Instanz einer Klasse erzeugt wird, funktioniert das Ganze nur dann, wenn diese Klasse keine Programmierfehler enthält. Ist sie kompilierbar, erweist sich das Objekttestcenter als große Hilfe beim Testen von Methoden und Eigenschaften. Selbst für eine eventuelle Parameterübergabe ist gesorgt, wie in Abbildung 3.10 zu sehen ist. Die Methode erwartet in diesem Fall einen Parameter vom Typ `String`, also eine Zeichenkette. Der Parameter muss daher in Anführungszeichen gesetzt werden.

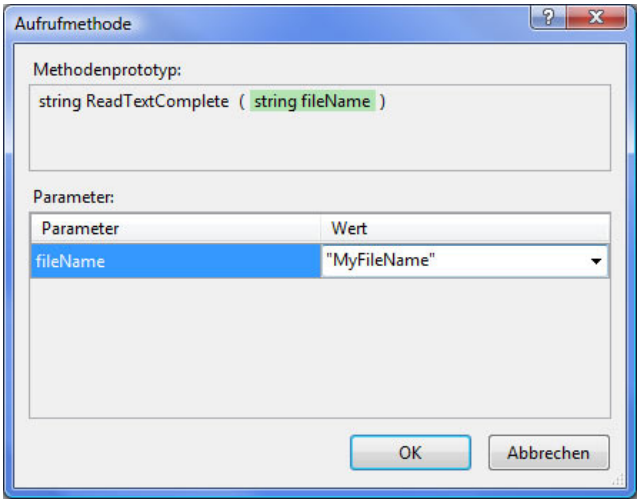


Abbildung 3.10: Übergabe eines Parameters an das Objekttestcenter

3.4.3 Code Snippets (Codeausschnitte)

Code Snippets sind kleine Codefragmente, die immer wieder benötigt werden. Das Visual Studio beinhaltet den sogenannten *Codeausschnitte-Manager* (im Original *Code Snippets Manager*). In ihm können Sie beliebig viele Codeausschnitte unterbringen, die innerhalb des Editors dann über ein Tastenkürzel oder über das Kontextmenü eingefügt werden können – zumindest in der Theorie. Denn es gibt noch keinen Editor für eigene Codeausschnitte. Codeausschnitte werden in XML-Dateien mit der Endung *.snippet* abgespeichert und können in den Codeausschnitte-Manager importiert werden.

> > > HINWEIS

Eine Anleitung zum Erstellen von Codeausschnitten finden Sie in der MSDN auf der Seite http://msdn2.microsoft.com/en-us/library/ms165393(VS.80).aspx. Leider ist es dafür erforderlich, umfangreiche Mengen an XML-Code zu schreiben. Ein passendes Tool hierfür finden Sie unter der folgenden Adresse: http://www.codeplex.com/snippy/

3.5 Fazit

In einem kurzen Abschnitt innerhalb eines Buches können nicht alle Vorzüge einer Entwicklungsumgebung dargestellt werden. Schon gar nicht, wenn es sich um eine so umfangreiche Software wie das Visual Studio 2008 handelt. Sie dürften allerdings bereits erkannt haben, dass diese Entwicklungsumgebung wirklich zahlreiche nützliche Tools enthält, die Ihnen die Arbeit stark erleichtern. Sofern Sie damit arbeiten – gleich mit welcher Version –, werden Sie sicherlich feststellen, dass derzeit kein vergleichbares Tool existiert. Das Visual Studio ist definitiv die beste Wahl für die Entwicklung unter Microsoft .NET.