Frank Eller

Visual C# 2005

Grundlagen, Programmiertechniken, Datenbanken

An imprint of Pearson Education

München • Boston • San Francisco • Harlow, England
Don Mills, Ontario • Sydney • Mexico City
Madrid • Amsterdam

3 Das Visual Studio 2005

Bereits die Vorgängerversion des Visual Studio 2005 wurde vielerorts als die beste Entwicklungsumgebung gepriesen, die es für Geld zu kaufen gibt. Aber auch das Beste kann noch verbessert werden. In die 2005er Version der Microsoftschen Entwicklungsumgebung flossen zahlreiche neue und nützliche Ideen ein, die den Programmiereralltag erleichtern.

Sämtliche Möglichkeiten des Visual Studio zu beschreiben ist allerdings unmöglich – damit würde nicht nur der Rahmen dieses Kapitels sondern vermutlich des gesamten Buchs gesprengt. Es geht in diesem Kapitel lediglich darum, die wichtigsten Elemente der Entwicklungsumgebung hervorzuheben.

3.1 Einführung

Die Voraussetzungen an aktuelle Applikationen steigen, ebenso die Möglichkeiten aber 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.

Für .NET gibt es derzeit eigentlich nur zwei Entwicklungsumgebungen, nämlich das Visual Studio und *SharpDevelop* von Mike Krüger. Borland hatte sich zwar an einer IDE versucht, der C#Builder ist aber als eigenständige Version nicht mehr erhältlich sondern liegt nur noch Borlands Delphi 2005 bzw. auch der neuen Version 2006 bei.

SharpDevelop ist ein sehr interessantes OpenSource-Projekt, das daher auch im Quellcode vorliegt. Auch fortgeschrittene Programmierer können noch viel aus dem Quellcode lernen. Die IDE ist zwar noch nicht an .NET 2.0 angepasst und ein paar Sachen fehlen auch noch, sie wird allerdings ständig weiterentwickelt und ist vor allem im Bereich Open Source sehr beliebt. Für kleinere Applikationen ist sie sicherlich eine Alternative. Besonders zu erwähnen ist, dass der Quellcode mit SharpDevelop auch für Mono kompiliert werden kann. Unter Linux existiert ein Projekt namens *MonoDevelop*, mit dem eine reine Mono-IDE zur Verfügung gestellt werden soll. Diese basiert auf SharpDevelop.

VERWEIS

Mehr Informationen über *SharpDevelop* finden Sie im Internet auf der Sharpdevelop-Website unter *http://www.icsharpcode.net/OpenSource/SD/Default.aspx*.

3.1.1 Übersicht

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

- Der visuelle Designer wurde im Vergleich zum Vorgänger komplett überarbeitet und bietet nun eine verbesserte Ausrichtungsmöglichkeit durch *Snap Lines* sowie das Task-Menü für Komponenten, mit denen sich die gebräuchlichsten Einstellungen sofort im Designer vornehmen lassen.
- Der Debugger wurde verbessert, hier vor allem die Anzeige von Exceptions, die wesentlich detaillierter ist als im Vorgänger, sowie die Anzeige und mögliche Änderung von Werten direkt im Editor. Der Debugger unterstützt nun auch *Edit&Continue*, womit Anhalten, Ändern von Werten und das darauf folgende Fortsetzen eines Programms an der gleichen Stelle möglich ist.
- Die IntelliSense-Hilfe bietet nun bereits nach den ersten Buchstaben ihre Hilfe an und versucht, das gewünschte Wort zu erkennen. Reservierte Wörter wurden ebenso in die Liste aufgenommen wie selbst erstellte Klassen oder Objekte.
- *Smarttags* helfen bei gängigen Operationen im Editor. Unter anderem können geänderte Variablennamen direkt für die gesamte Methode oder auch eine gesamte 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 nun 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.
- Layoutanpassungen der Entwicklungsumgebung wurden verbessert. Das Andocken der diversen Tool-Fenster des Visual Studio, das in der Vorgängerversion immer ein Problem darstellte, wird nun über so genannte *Guides* (Ablageflächen) erleichtert.
- Integrierte Tools, beispielsweise *Code Snippets* zum Einfügen kleiner Codebestandteile oder auch Refactoring-Tools erleichtern die Arbeit beim Programmieren. Integriert wurde auch ein Tool namens *FxCop*, unverzichtbar wenn es darum geht, Code zu erzeugen, der uneingeschränkt aus anderen Programmiersprachen heraus verwendbar sein soll. In den größeren Versionen des Visual Studio finden Sie dieses Tool in den Projekteigenschaften unter der Bezeichnung *Code Analyse*. Leider nicht in der Professional-Version, weshalb Sie hier auf das Internet bzw. auf die entsprechende FxCop-Version angewiesen sind. Sie finden FxCop unter *http://www.gotdotnet.com/team/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 2000 (Client/Server) mit Service Pack 4, Windows XP (Home/Pro) mit Service Pack 2 oder Windows 2003 Server
- *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 die MSDN auch die Hilfefunktion beinhaltet, dürfte dies das Standardvorgehen sein
- *Bildschirmauflösung*: Mindestens 800x600, empfohlen 1024x768
- 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 800x600 Bildpunkten zu arbeiten ist *enorm* frustrierend bis unmöglich.

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

- *Betriebssystem*: Windows 2000 (Client/Server) mit Service Pack 4, Windows XP (Home/Pro) mit Service Pack 2 oder Windows 2003 Server
- *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 1280x1024, je mehr desto besser. Ab 1600x1200 Bildpunkten macht das Arbeiten Spaß, unterhalb von 1280x1024 wird es schnell frustrierend. 1024x768 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 oder C++, die ausschließlich auf die Entwicklung von Windows.Forms-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 des 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 26.5 ab Seite 926) wird hier unterstützt. Ab der Professional-Version ist auch der SQL Server 2005 in der Developer-Edition enthalten.

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

Die Basis bildet bei allen diesen Produkten die Professional-Edition des 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 Software-Tester 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 Buchs laufen aber auch mit jeder anderen Version des Visual Studio, insbesondere auch mit der Express-Edition von Visual C#.

3.2 Wichtige Fenster der Entwicklungsumgebung

Die Abbildungen auch in diesem Abschnitt stammen sämtlichst 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.

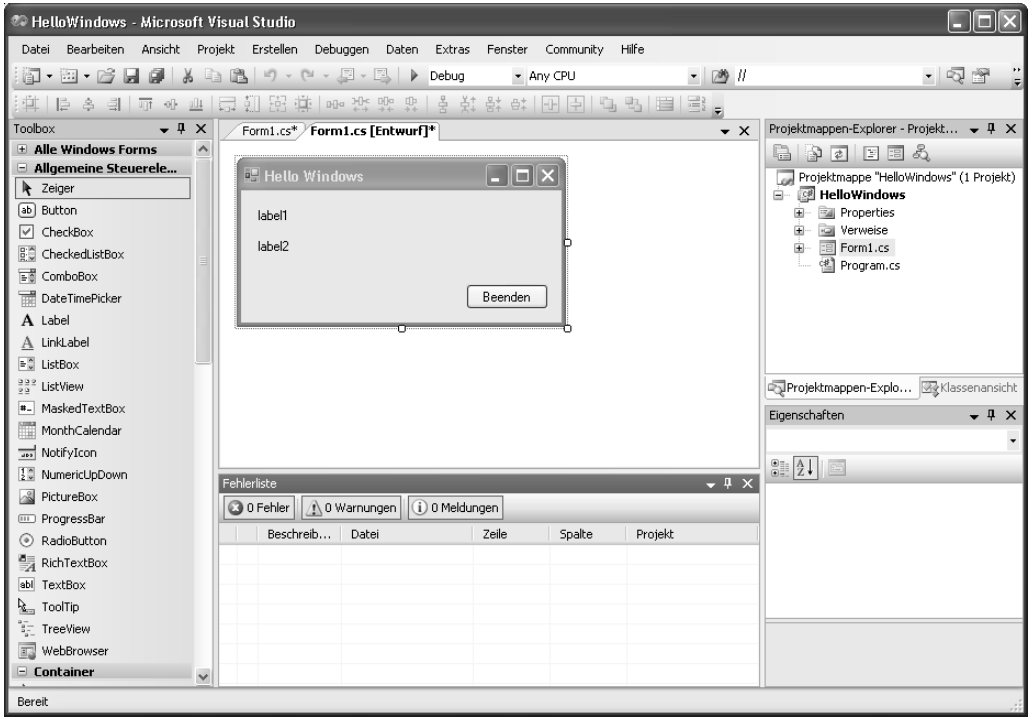

Abbildung 3.1: Das Visual Studio, Professional-Edition in der Standardeinstellung

Abbildung 3.1 zeigt das Visual Studio in der Standardeinstellung. 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 Eigenschaftsfenster, eigentlich ein kombiniertes Eigenschafts-/Ereignisfenster, das Ihnen Zugriff auf die Eigenschaften eines Steuerelements bietet. Mithilfe dieser Eigenschaften ändern Sie u.a. Aussehen und Verhalten der Steuerelemente. Im unteren Bereich finden Sie die Fehlerliste, in der Sie nach einem erfolglosen Kompiliervorgang sämtliche aufgetretenen Compilerfehler finden.

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 Studios, je nach Programmiersprache mit unterschiedlichen Möglichkeiten. Für andere Dateien, z.B. Ressourcendateien, existieren noch weitere Ansichten.

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.

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 Projektmappenexplorer bzw. das FENSTER-Menü 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 Minimum angesehene Auflösung von 1028x1024 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), 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 Studios 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 auch das Startprojekt sein kann (also das 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.

HINWEIS

Eine ärgerliche Tatsache des Visual Studio 2003 war es, dass die Änderung eines Dateinamens nicht sofort auch die Änderung des Namens der darin enthaltenen Klasse bedeutete. Das war vor allem bei Formularen frustrierend. Das Hauptformular der Anwendung hieß standardmäßig `Form1`, die Datei *Form1.cs*. Nun musste man sowohl die Klasse umbenennen als auch den Dateinamen und auch die Instanzierung der Klasse in `Main()`. Das Visual Studio 2005 bietet beim Ändern des Dateinamens nun sofort ein Refactoring an und ändert alle referenzierten Namen im Projekt.

Neue Elemente

Eine Applikation kann viele unterschiedliche Elemente beinhalten, z.B. Formulare oder Klassen. Der Projektmappen-Explorer ist auch hierfür verantwortlich. Über einen Rechtsklick auf entweder 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 Hauptnamespace) 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. 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). Weiterhin 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.

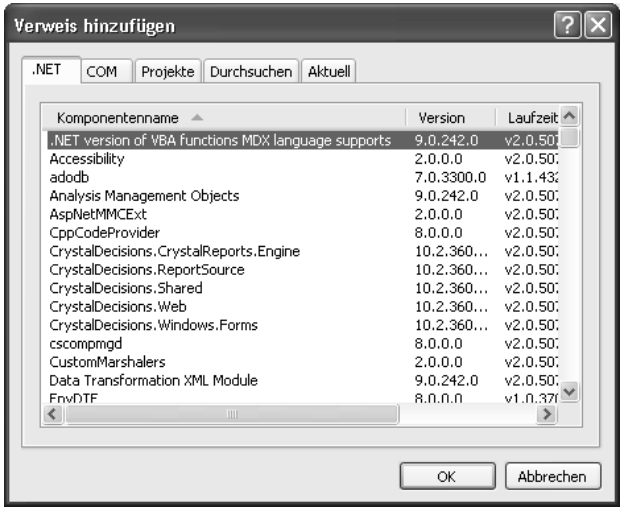

Abbildung 3.2: Der Dialog zum Hinzufügen von Verweisen

Über das gleiche Kontextmenü ist es auch möglich einen so genannten *Webverweis* hinzuzufügen. Dabei handelt es sich um Web Services – Sie können also mithilfe des Visual Studios 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 einfach nach »Projektmappen-Explorer« suchen. Der direkte Link ist:

ms-help://MS.VSCC.v80/MS.MSDN.v80/MS.VisualStudio.v80.de/dv_vsprojopt/html/ca0ad8e7-eda8-40d4-a76e-2a6864b16e00.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 z.B. 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.

Sie können zusätzlich zu den in der Toolbox enthaltenen Kategorien eigene hinzufügen (über das Kontextmenü der Toolbox). Außerdem ist es auch möglich, der Toolbox weitere Steuerelemente hinzuzufügen. Dabei kann es sich um selbst geschriebene Steuerelemente oder um Komponenten von Drittanbietern handeln. Zum Erweitern der Toolbox 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. die Komponente, die Sie suchen.

VERWEIS

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

ms-help://MS.VSCC.v80/MS.MSDN.v80/MS.VisualStudio.v80.de/dv_vwdgenref/html/35e9320d-fcbd-474b-8b8f-55705e9a1870.htm

3.2.3 Das Eigenschafts-/Ereignisfenster

Die Einstellungen der Komponenten und Steuerelemente auf dem Formular erfolgt über Eigenschaften. Über diese steuern Sie sowohl das Aussehen als auch in vielen Fällen das Standardverhalten eines Steuerelements. Das Eigenschaftsfenster ist daher das Fenster, das am häufigsten zum Einsatz kommt. Die wichtigsten Einstellungen ein Steuerelement betreffend können Sie ab dieser Version des Visual Studios 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 können selbst 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 auftritt. 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 Die Projekteigenschaften

In .NET 1.1 wurden die Projekteigenschaften noch in einem kleinen Dialog eingestellt. In .NET 2.0 hat sich das geändert. Alle Einstellungen lassen sich nun 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 3.3 zeigt die neu gestalteten Projekteigenschaften.

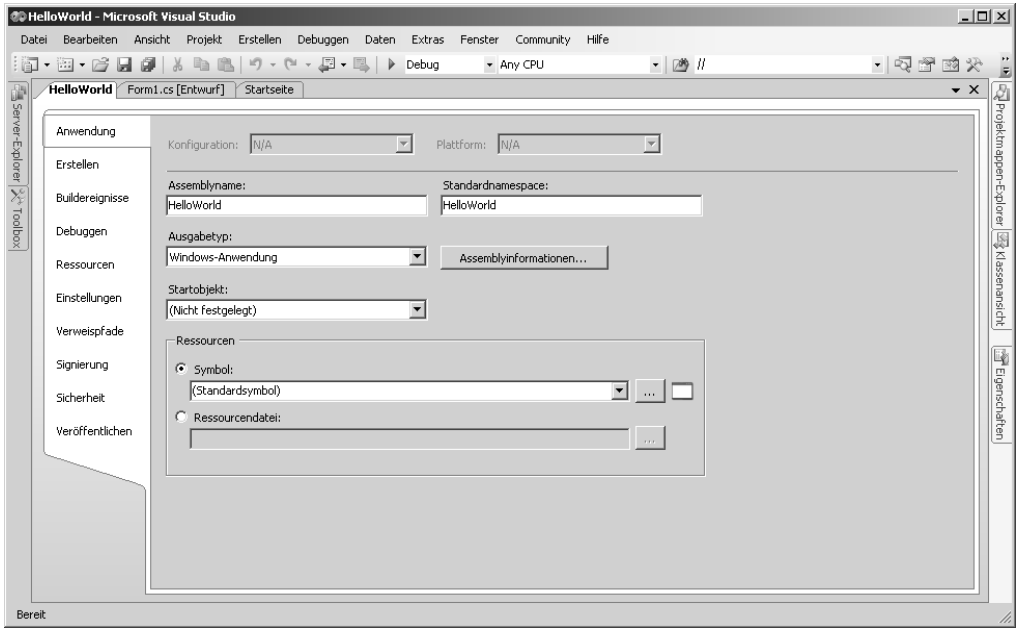

Abbildung 3.3: Die Projekteigenschaften im Visual Studio 2005

3.3 Der visuelle Designer

Die Gestaltung einer Anwendung bzw. der in der Anwendung enthaltenen Formulare war bereits im Visual Studio 2003 sehr komfortabel. Steuerelemente konnten per Drag&Drop auf das Formular gezogen werden und wurden dann an einem Grid ausgerichtet.

Snap Lines

Leider war es bisher nicht möglich, beispielsweise ein Label automatisch so auszurichten, dass sein enthaltener Text auf der gleichen Grundlinie läuft wie der Text eines daneben angeordneten Textfelds. Korrekte Abstände und die Länge von Elementen konnten vor allem bei umfangreicheren Formularen oft nur schlecht abgeschätzt werden – um wirklich detailliert richtig zu liegen war es meist nötig, selbst Hand anzulegen und die Positionsdaten über das Eigenschaftsfenster manuell festzulegen.

Mit dem neuen Designer ist das Grid als Standard-Anordnungsgrundlage verschwunden. Es lässt sich zwar über die Optionen noch einblenden, allerdings sind die neuen *Snap Lines* wesentlich komfortabler (sie wurden auch schon in Abschnitt 2.2.2 ab Seite 54 angesprochen). Mit ihnen lassen sich die Elemente schneller und komfortabler anordnen – und die Snap Lines arbeiten pixelgenau, beispielsweise bei der Ausrichtung von Labels auf der Grundlinie einer `TextBox`.

Aufgabenmenü (Task-Menü)

Ebenso neu ist die Möglichkeit, die wichtigsten Einstellungen eines Steuerelements bereits im Designer selbst festzulegen, statt die Eigenschaft erst noch im Eigenschaftsfenster suchen zu müssen. Viele der verfügbaren Steuerelemente ermöglichen dies durch ein neues Feature, das *Task-Menü* oder *Aufgabenmenü*. Nach dem Einfügen des Steuerelements erscheint oben rechts ein kleiner schwarzer Pfeil. Ein Klick darauf und das Aufgabenmenü wird eingeblendet. Bei manchen Steuerelementen, z.B. dem `DataGridView`, erfolgt die Einblendung gar automatisch. Abbildung 3.4 zeigt das Aufgabenmenü einer `ListBox`.

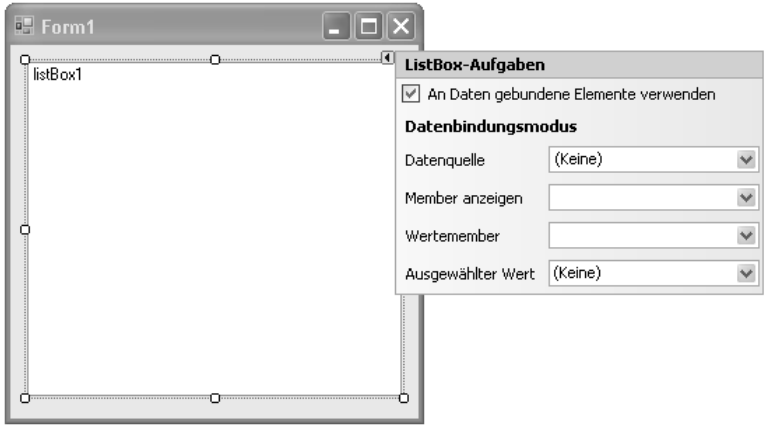

Abbildung 3.4: Das Aufgabenmenü einer ListBox. Die Felder für die Datenbindung erscheinen nur, wenn die ListBox auch als datengebundenes Steuerelement eingesetzt werden soll.

3.4 Der Editor

Der Texteditor des Visual Studio hilft durch zahlreiche kleine Hilfestellungen, die oft im Verborgenen liegen. Er unterstützt Syntaxhervorhebung, die im Vergleich zum Vorgänger noch erweitert wurde, 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 der Applikation springen können. Wieder dabei ist auch die IntelliSense-Hilfe, auch sie wurde erweitert. Hinzugekommen ist eine Unterstützung für Smarttags vor allem bei Klassennamen, die unbekannt sind.

3.4.1 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, 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. Und das nicht grundlegend wie in der vorherigen Version, sondern bis zum letzten Leerzeichen. 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, usw.

Diese Unmenge an Einstellungen nehmen Sie im Optionsdialog des Visual Studio vor (Menüpunkt EXTRAS|OPTIONEN). Wählen Sie dort den Eintrag TEXT-EDITOR|C#|FORMATIERUNG. Abbildung 3.5 zeigt den Dialog.

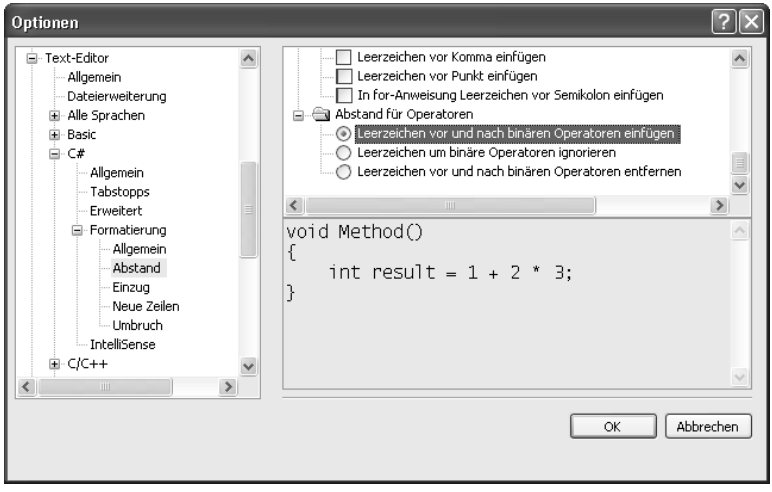

Abbildung 3.5: Der Dialog für die Formatierungsoptionen

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.

Die Einstellungen 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.

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, ist der erste Schritt immer die Festlegung so genannter Code-Conventions, die Benamung, Formatierung und Vorgehensweisen für das gesamte Team festlegen.

3.4.2 IntelliSense

Die IntelliSense-Hilfe wurde ebenfalls im Vergleich zum Vorgänger verbessert. Sie schlägt bereits nach den ersten eingegebenen Buchstaben passende Wörter vor. Dabei bezieht sie auch bekannte Klassennamen und reservierte Wörter mit ein. Das ist eine deutliche Verbesserung; im Vorgänger sprang die IntelliSense-Hilfe in der Regel erst nach Eingabe eines Punkts als qualifizierendem Operator ein.

3.4.3 Smarttags

Der Visual Studio Editor unterstützt nun auch *Smarttags*, die Sie vielleicht bereits aus den Office-Produkten kennen. Falls Sie beispielsweise innerhalb einer fertigen Klasse den Namen einer Variablen nachträglich ändern, können Sie über einen Smarttag diese Änderung für die gesamte Klasse (oder eine gesamte Methode) übernehmen. Dabei handelt es sich um ein so genanntes *Refactoring*; das Visual Studio unterstützt vor allem für C# zahlreiche Refactorings, wie sie in Fowlers gleichnamigem Buch beschrieben sind.

Besonders hilfreich ist dieses Feature dann, wenn Sie eine Klasse verwenden wollen, aber nicht wissen, in welchem Namespace sich die Klasse befindet. Über Smarttags können sie entweder diese Klasse vollständig qualifizieren (also den gesamten Klassennamen inklusive des Namespaces verwenden) oder den Namespace einbinden (siehe auch Abbildung 3.6). Das setzt allerdings voraus, dass die DLL, in der sich die Klasse und der Namespace befinden, referenziert ist.

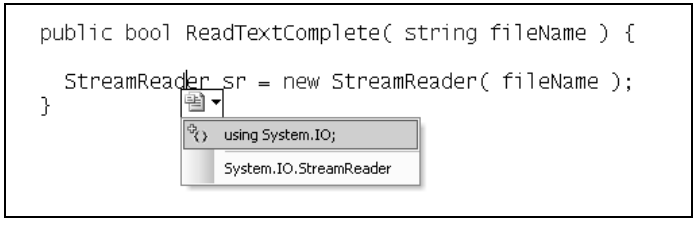

Abbildung 3.6: Smarttags im Visual Studio

Weitere Funktionalitäten, die über Smarttags ausgeführt werden können, sind z.B. Änderungen von Bezeichnern oder auch das Implementieren von Interfaces. Die Änderung eines Bezeichners entspricht einem Refactoring. Ändern Sie beispielsweise einen Methodennamen, erscheint sofort ein Smarttag, mit dem Sie das Refactoring durchführen und den Methodennamen auch an allen referenzierten Stellen ändern können.

3.4.4 Änderungen innerhalb einer Sitzung

Ebenfalls mitunter 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 komfortabel.

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.

3.4.5 Refactoring

Refactoring, das Verbessern des Quellcodes, ist ein ständiger Vorgang. Jeder ernsthafte Programmierer ist ständig 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 Felds in einer Eigenschaft, die Extraktion einer Methode, 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 bzgl. Refactoring ist das gleichnamige Buch von Martin Fowler. Auf Deutsch ist es bei Addison-Wesley erschienen. Die ISBN lautet 3-8273-2278-2. Obwohl für absolute Einsteiger noch etwas schwierig ist es doch schon für den fortgeschrittenen Programmierer zu empfehlen; nach der Lektüre dieses Buchs werden Sie definitiv besseren Code schreiben.

3.5 Tools und Hilfsmittel

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

3.5.1 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 in der Version 2005 ein Tool namens *Klassendiagramm*, mit dem genau dies 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 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 3.7 zeigt das Klassendiagramm des Visual Studios. 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 natürlich Klassen, aber auch Strukturen (`struct`), Aufzählungen (`enum`), Interfaces oder abstrakte Klassen.

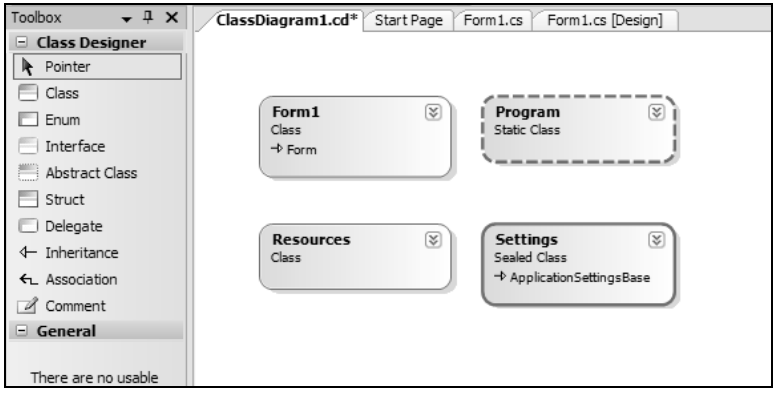

Abbildung 3.7: Der Klassendesigner des Visual Studio. Die Toolbox zeigt die verfügbaren Elemente.

3.5.2 Das Objekttestcenter

Aus dem Klassendesign heraus haben Sie auch Zugriff auf das Objekttestcenter, eine weitere Neuerung in Visual Studio 2005. Softwaretests sind allgemein Usus und es gibt auch zahlreiche Programme dafür. Unter anderem wäre da *nUnit* zu nennen, das für .NET recht populär ist.

Das Visual Studio bringt in der Team Edition eine Testumgebung mit, die das gleiche kann wie nUnit, aber eben in das Visual Studio integriert ist. Doch nicht immer muss es so groß sein – manchmal soll nur schnell ein einziges Objekt bzw. eine einzige Klasse getestet werden. Zu diesem Zweck existiert in den Visual-Studio-Versionen die *Object Test Bench* (oder Objekttestcenter), die genau das ermöglicht. Zugriff darauf erhalten Sie aus dem Klassendesigner oder aus der Klassenansicht (ANSICHT|KLASSENANSICHT, Strg+⇧+C). Über das Kontextmenü einer Klasse können Sie diese instanzieren (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 3.8 zeigt das Objekttestcenter mit einem erzeugten Objekt, von dem eine Methode aufgerufen wird.

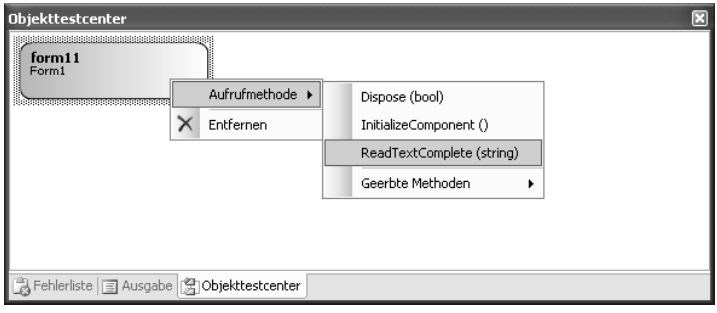

Abbildung 3.8: Das Objekttestcenter des Visual Studio

Es versteht sich von selbst, dass die Klasse, die instanziiert werden soll, auch kompilierbar sein muss, also keine Fehler enthalten darf. Das Ganze funktioniert natürlich nur, wenn sie kompiliert werden kann – dann aber richtig. Selbst für eine evtl. Parameterübergabe ist gesorgt, wie in Abbildung 3.9 zu sehen. Die Methode erwartet in diesem Fall einen Parameter vom Typ `String`, also eine Zeichenkette. Dieser muss daher in Anführungszeichen gesetzt werden.

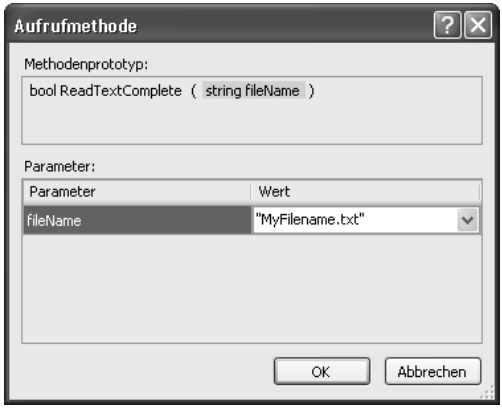

Abbildung 3.9: Übergabe eines Parameters für das Objekttestcenter

3.5.3 Code Snippets (Codeausschnitte)

Code Snippets sind kleine Codefragmente, die immer wieder benötigt werden. In den Vorgängerversionen konnten solche Fragmente in der Toolbox abgelegt werden (das geht immer noch), in der Version 2005 gibt es hierfür den *Codeausschnitte Manager* (oder im Original *Snippet Manager*). In ihm können Sie beliebig viele Codeausschnitte unterbringen, die innerhalb des Editors dann über entweder ein Tastenkürzel oder über das Kontextmenü eingefügt werden können.

Leider hat diese Sache auch einen Wermutstropfen. Der Codeausschnitte-Manager, erreichbar über das Menü EXTRAS, sieht zwar schön aus – dennoch gibt es derzeit noch keine Möglichkeit, einen eigenen Codeschnipsel auf einfache Art und Weise hinzuzufügen. Sie müssen in der Tat (in der Hilfe ist es beschrieben) eine XML-Datei schreiben und diese mit der Endung *.snippet* abspeichern. Diese XML-Datei müssen Sie zu allem Überfluss auch noch von Hand schreiben – hier wäre eine bessere Lösung angebracht gewesen.

3.6 Fazit

In einem kurzen Abschnitt innerhalb eines Buches können freilich nicht alle Vorzüge einer Entwicklungsumgebung dargestellt werden – schon gar nicht, wenn es sich um eine so umfangreiche Software wie dem Visual Studio 2005 handelt. Sie dürften allerdings bereits erkannt haben, dass diese Entwicklungsumgebung wirklich zahlreiche nützliche Tools enthält, die Ihnen die Arbeit stark erleichtern. Wenn Sie damit arbeiten – gleich mit welcher Version – werden Sie sicherlich auch feststellen, dass derzeit kein vergleichbares Tool existiert. Das Visual Studio ist definitiv die beste Wahl für die Entwicklung unter Microsoft .NET.