Visual Basic 2008

Windows-Programmierung mit Visual Basic 9.0,
Visual Studio 2008 und .NET Framework 3.5

PETER MONADJEMI

Markt+Technik

KOMPENDIUM

Einführung | Arbeitsbuch | Nachschlagewerk

4 Visual Studio 2008

My home is my castle

Unbekannter Entwickler

In diesem Kapitel geht es ausschließlich um die Entwicklungsumgebung Visual Studio 2008, die als universelle »Allround-IDE« (*Integrated Development Environment*) neben Visual Basic und C# noch einige andere .NET-Programmiersprachen »beherbergen« kann und zudem im Rahmen der »Visual Studio Shell« frei zur Verfügung steht, sodass sie theoretisch jeder Entwickler für seine Zwecke nutzen kann. Visual Studio blickt als Microsoft-IDE auf eine längere Geschichte zurück, die als Entwicklungsumgebung für C++ bereits vor .NET begann. In diesem Kapitel erfahren Sie alles Wissenswerte über den Umgang mit der Visual Studio-IDE. Dazu zählen vor allem Themen wie der Umgang mit Projekten und die Rolle der zahlreichen Einstellungen und Anpassungen, die sich in der umfangreichen IDE vornehmen lassen. Dieses Kapitel werden Sie sicher nicht in einem Rutsch lesen, sondern portionsweise, um sich über die verschiedenen Aspekte beim Arbeiten mit der IDE gezielt zu informieren. Visual Studio, das sei als das vorgenommene Fazit für dieses Kapitel bereits vorweggenommen, ist eine der durchdachtesten Anwendungen aus dem Hause Microsoft, das die Herausforderung, sowohl Anfängern die lästigen Formalitäten abzunehmen als auch anspruchsvolle Profis zufriedenzustellen, mit Bravour meistert. Perfekt ist die IDE noch nicht, etwa was die Vorlagenverwaltung angeht, aber es wird mit an Sicherheit grenzender Wahrscheinlichkeit eine nächste Version geben.

Die Stichwörter für dieses Kapitel:

- Ein erster Rundgang durch die Visual Studio-IDE
- Der Umgang mit Projekten – die Projektverwaltung
- Projekte erstellen, ausführen und debuggen
- Arbeiten mit Quelltext
- Die Rolle der Vorlagen

- Einstellen von Optionen
- Die IDE über die Tastatur bedienen
- Tipps & Tricks für die Praxis

4.1 Ein erster Rundgang durch die IDE

Auch wenn die große Zahl an Menüs, Symbolleisten und »Einstellmöglichkeiten« am Anfang vielleicht etwas abschreckend wirken mag, Visual Studio ist eine benutzerfreundliche Anwendung, in der sich auch »Anfänger« schnell zurechtfinden und nach kurzer Zeit mit ihr produktiv arbeiten können. Gleichzeitig gibt es keine »Spielereien« oder Assistenten, die den Blick für das Wesentliche verstellen oder den Anwender bevormunden, sodass erfahrene und weniger erfahrene Anwender gleichermaßen zufriedengestellt werden.

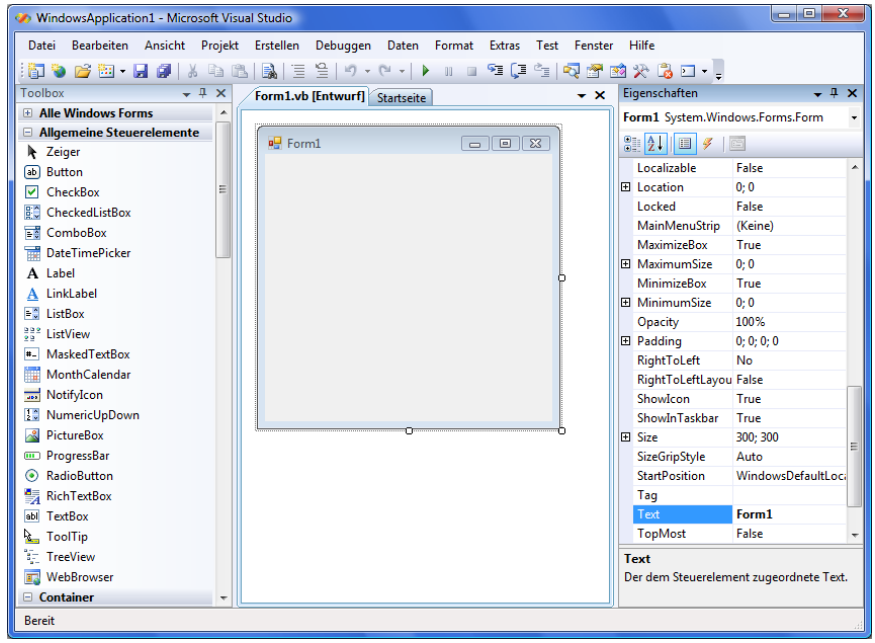


Abbildung 4.1: Visual Studio 2008 nach dem ersten Start

Nach dem ersten Aufruf steht in der Regel das Anlegen eines neuen Projekts oder das Öffnen eines vorhandenen Projekts an. Einfach ein paar Befehle einzutippen und zur Ausführung zu bringen, geht bei Visual Studio leider nicht, denn die Programmierung spielt sich stets in einem Projekt ab. Der erste Schritt ist daher, daran hat sich auch bei Visual Studio 2008 im Vergleich zu früheren Visual Basic-Versionen nichts geändert, die Auswahl einer Projektvorlage.

Station 1: Projektauswahl

Auch wenn Visual Studio 2008, insbesondere die Professional Edition, eine reichhaltige Auswahl an Projekttypen anbietet, in den meisten Fällen wird es ein neues Projekt vom Typ »Windows Forms-Anwendung« sein. Diese Vorlage ist die Grundlage für eine Windows-Anwendung, die aus einem oder mehreren Formularen mit Steuerelementen besteht und die Visual Basic-Programmierern bestens vertraut ist.

Station 2: Projektmappen-Explorer

Mit dem Anlegen eines neuen Projekts wird (sofern nichts anderes festgelegt wurde) ein Projektverzeichnis auf der Festplatte angelegt, dessen Inhalt im Projektmappen-Explorer angezeigt wird. Eine kleine Besonderheit des Projektmappen-Explorers ist, dass er am Anfang nicht alle Dateien und Unterverzeichnisse und nicht die Verweise des Projekts anzeigt. Viele angehende Visual Studio-Anwender ahnen daher nichts von der Existenz des *bin\debug*-Ausgabeverzeichnisses, in dem bei jedem Erstellen die resultierende .exe-Datei abgelegt wird. Dies ist eine der wenigen Stellen, wo die IDE den Anwender ein wenig bevormundet, zumal es offenbar keine Möglichkeit gibt einzustellen, dass von Anfang an alle Elemente angezeigt werden.

Manchmal findet man irgendwo (meistens ist dieses »Irgendwo« eine Webseite) eine einzelne Quellcodedatei mit der Erweiterung .Vb. *Ist der Code der Datei fehlerfrei, kann die Datei theoretisch direkt kompiliert werden. Jedoch nicht in Visual Studio, denn es muss erst ein Projekt angelegt, die Quelltextdatei hinzugefügt, sie zur Startdatei gemacht und die nicht benötigte Formular- bzw. Moduldatei gegebenenfalls erst entfernt werden. Da das alles ein wenig umständlich ist, gibt es bei Visual Studio (ein wenig versteckt) den Menübefehl* DATEI|NEU|PROJEKT AUS VORHANDENEM CODE. *Er startet einen kleinen Assistenten, über den das Verzeichnis mit den Quelltextdateien und der Typ des anzulegenden Projekts abgefragt werden. Doch Vorsicht vor der Einstellung* Unterordner einbeziehen. *Bleibt sie gesetzt, werden auch sämtliche Unterordner einbezogen, und Visual Studio »nudelt« sämtliche Dateien in allen Verzeichnissen durch, was nicht nur eine Weile dauern kann, sondern sich nicht abbrechen lässt. Auch hat Visual Studio gewisse Vorstellungen davon, wie eine Quelltextdatei aufgebaut sein muss, sodass unter Umständen noch Anpassungen erforderlich sind. Gut gedacht, aber nicht optimal umgesetzt.*

Kompatibilität zu VS 2005

Es hat Tradition, dass sich bei Visual Basic bzw. Visual Studio praktisch mit jeder neuen Version das Projektformat ändert. Auch wenn es nur Kleinigkeiten sind, die eine Projektdatei bei VS 2005 von einer Projektdatei bei VS 2008 unterscheidet, müssen mit älteren Visual Studio-Versionen erstellte Projekte beim Laden in Visual Studio 2008 erst einmal konvertiert werden, was ein Assistent souverän erledigt. Eingestellt werden muss nichts, das Anfertigen einer Sicherheitskopie dürfte nur etwas für übervorsichtige Anwender sein, und der angefertigte Report wandert bei einer Umstellung von VS 2005 auf die aktuelle Version im Allgemeinen ungelesen in den Papierkorb.

Station 3: Formulardesigner und Code-EditorCode-Editor

Wenn in diesem Buch vom Formulardesigner die Rede ist, so ist dies kein High-End-Werkzeug für besser betuchte Developer, sondern das, was Visual Basic-Programmierer bereits von der Version 1.0 an kennen: Die Oberfläche eines Formularfensters, auf dem sich Steuerelemente platzieren und verschieben lassen. Bereits mit Visual Studio 2005 wurde der Formulardesigner kräftig geliftet und z. B. mit den sehr praktischen Führungslinien ausgestattet, die dafür sorgen, dass die Feinabstimmung beim Anordnen von Steuerelementen kein Geduldspiel mehr ist.


TIPP

Möchte man lediglich ein paar Zeilen Quellcode ausprobieren, hat man nicht immer die Muße, dafür ein neues Projekt anzulegen. Für diese Zwecke ist der kleine Snippet Compiler *von* Jeff Key *gut geeignet (Download unter* http://www.sliver.com/dotnet/SnippetCompiler*). Das kleine Tool ist eine Mini-IDE, in der komplette Befehlsfolgen eingegeben und ausgeführt werden können, ohne dass dafür ein Projekt angelegt werden muss. Trotzdem gibt es ein wenig Visual Studio-Komfort durch Auswahllisten und andere Eingabehilfen.*

Station 4: Eingabehilfen

Die Eingabehilfe des Code-Editors beschränkt sich längst nicht mehr auf die Anzeige von Auswahllisten nach der Eingabe eines Punktes. Dieser Standardkomfort wurde inzwischen so weit perfektioniert, dass er auch für Typen angeboten werden kann, die im selben Befehl entstanden sind, wie es bei einer LINQ-Abfrage der Fall ist. Auch für Befehle gibt es inzwischen Auswahllisten, was sehr angenehm ist, da bereits für den ersten Schritt (der bekanntlich am schwersten ist) Auswahlhilfen angeboten werden. Der Code-Editor bietet auch bei Programmfehlern Eingabehilfen an, die oft sogar Korrekturvorschläge enthalten, die per Mausklick übernommen werden können. Der klassische Fall ist die Verwendung eines Shared-Members bei einer Instanz der Klasse, der zu einer Warnung des Compilers führt.

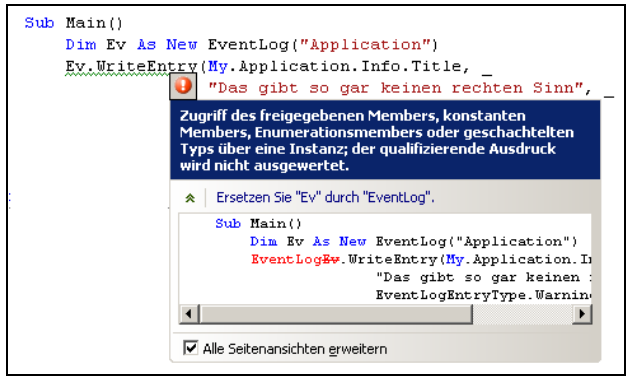


Abbildung 4.2: Die Eingabehilfe des Code-Editors bietet einen Korrekturvorschlag an.

Station 5: Datei-Menü und Standard-Symbolleiste

Ein neu angelegtes Projekt sollte als Erstes gespeichert werden. Das geschieht über den Menübefehl *Datei|Alle speichern* oder über das entsprechende Symbol in der Standard-Symbolleiste. Schauen Sie die Standard-Symbolleiste in Ruhe an, denn Sie finden hier alle Befehle, die für die tägliche Praxis benötigt werden.

Station 6: Das Erstellen-Menü

Ein Projekt wird nicht kompiliert, sondern erstellt. Daher heißt der entsprechende Befehl im *Erstellen*-Menü auch *<Projektname> erstellen* oder *<Projektname> neu erstellen*, wobei *<Projektname>* für den Namen des Projekts steht, das erstellt werden soll. Das Ergebnis ist, je nach Projekttyp, eine .exe- bzw. .dll-Datei, die im ausgewählten Ausgabeverzeichnis abgelegt wird. Alternativ kann gleich die gesamte Projektmappe erstellt werden. Bei kleinen Projekten macht dies keinen Unterschied, bei großen bis sehr großen Projekten spielt es durchaus eine Rolle, wenn Visual Studio einige Minuten damit beschäftigt ist, alle Quelltextdateien neu zu übersetzen.

Station 7: Das Debuggen-Menü

Viele Visual Basic-Programmierer haben über die Jahre das »F5-Prinzip« geradezu verinnerlicht. Befehle eintippen, F5 drücken, um das Ergebnis begutachten zu können, Änderung vornehmen, erneut F5 drücken, um das Ergebnis begutachten zu können, usw. Daran hat sich auch bei Visual Studio grundsätzlich nichts geändert, nur dass F5 nicht das Programm »startet«, sondern die vom Compiler erzeugte .exe-Datei im Debug-Modus zur Ausführung bringt und sie gegebenenfalls zuvor erstellt. Soll die .exe-Datei nicht im Debug-Modus, sondern »regulär« zur Ausführung gebracht werden, wird statt F5 ein Strg + F5 gedrückt (von den Unterschieden beider Modi wird noch die Rede sein). Das Wichtigste für einen Entwickler ist, dass er oder sie produktiv arbeiten kann. Visual Studio bietet zahlreiche Hilfestellungen an, um dieses Ziel erreichen zu können. Eine davon ist, dass durch F5 wie seit jeher das Projekt neu erstellt und, sofern es keine Fehler im Quelltext gab, die resultierende .exe-Datei gestartet wird.

Station 8: Der Fenster-Reiter

Alle geöffneten Fenster werden auf einem Reiter angeordnet. Pro Reiter kann immer nur ein Fensterinhalt angezeigt werden. Zwischen den Fenstern wird über den Reiterkopf umgeschaltet. Sind mehr Fenster geöffnet, als Reiterköpfe in einer Reiterzeile Platz haben, erscheint am linken Rand eine Auswahlliste. Es können jederzeit neue Reiter angelegt werden. Auf diese Weise ist es zum Beispiel möglich, ein Formular und den dazugehörigen Quellcode gleichzeitig zu betrachten.

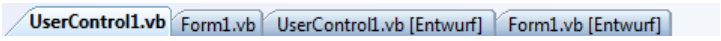


Abbildung 4.3: Über die Reiterleiste wird auf ein Fenster umgeschaltet.

4.1.1 Start unter Vista

Unter Windows Vista gelten leider etwas andere Spielregeln. Bei aktivierter Benutzerkontensteuerung muss Visual Studio in den meisten Fällen explizit unter dem Administratorkonto gestartet werden, damit es nicht aufgrund fehlender Berechtigungen zu lästigen Fehlermeldungen kommt. Dies geschieht am einfachsten, indem Sie den Visual Studio-Eintrag im Startmenü mit der rechten Maustaste anklicken und *Als Administrator ausführen* wählen.

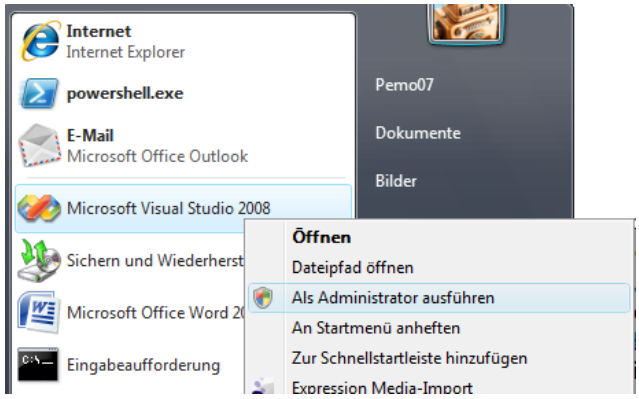


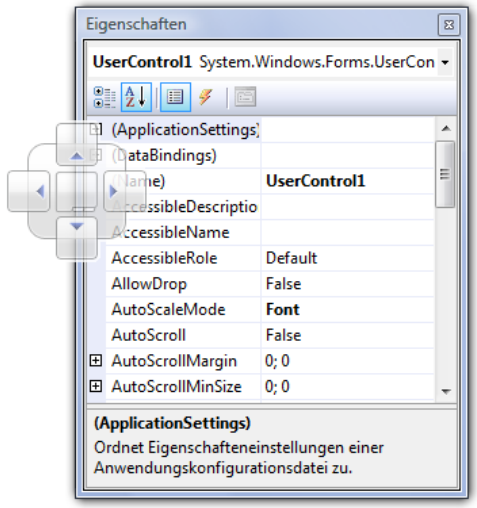
Abbildung 4.4: Unter Vista muss Visual Studio in der Regel explizit als Administrator ausgeführt werden.

4.2 Fensterparade

Visual Studio lebt von seinen Fenstern, von denen im Folgenden die wichtigsten »Exemplare« vorgestellt werden. Fenster, die von Anfang an nicht sichtbar sind, werden über das *Ansicht*-Menü oder über *Ansicht|Weitere Fenster* sichtbar gemacht. Nehmen Sie sich die Zeit, und schauen Sie sich jedes Fenster in Ruhe an, denn hier verbergen sich die wichtigsten Funktionen der IDE. Die Fenster besitzen nicht nur eine gegenüber der Vorgängerversion ein wenig modernisierte Optik, sie sind (vor allem im Vergleich zu VS.NET 2003) zudem noch ein wenig handlicher geworden. Die verschiedenen Fenster integrieren sich automatisch ineinander, zwischen ihnen kann über einen Registerreiter umgeschaltet werden. Beim Verschieben von Fenstern innerhalb der IDE haben sich die Microsoft-Entwickler etwas Besonderes einfallen lassen: Es erscheinen an allen vier Rändern sowie in der Mitte Markierungselemente – sogenannte Diamant-Führungssymbole –, die auf den ersten Blick ein wenig deplatziert wirken. Ihre Nützlichkeit offenbart sich, wenn ein Fenster über einen Pfeil bewegt wird, da der Bereich, den das Fenster beim Loslassen einnehmen würde, optisch hervorgehoben wird. Wird das Fenster losgelassen, dockt es, wie von Geisterhand gesteuert, zuverlässig in diesem Bereich an. Diese Maßnahme soll verhindern, dass ein Fenster wie bei früheren Visual Studio-Versionen wieder und wieder gegen einen Rand »geschoben« werden muss, damit es endlich dort andockt.

Welche Fenster angeboten werden, hängt von der Visual Studio-Edition ab. Die folgende Aufstellung bezieht sich auf die Professional Edition von Visual Studio 2008. In der Express Edition stehen nicht alle der beschriebenen Fenster zur Verfügung.

Abbildung 4.5: Beim Verschieben eines Fensters werden »Andockhilfen« eingeblendet.



4.2.1 Der Server-Explorer

Der Server-Explorer zeigt die »Server-Elemente« des aktuellen Computers sowie jener Computer im Netzwerk an, die nachträglich hinzugefügt wurden. Dazu gehören unter anderem die Dienste, die Ereignisprotokolle, die Leistungsindikatoren, die Meldungswarteschlangen und die WMI-Elemente. Der Server-Explorer ist nicht nur zum Betrachten da, jedes Element kann auf ein Formular (oder generell auf eine

Designerfläche) gezogen werden, was zur Folge hat, dass eine Komponente im Quellcode angelegt und im Komponentenfach angezeigt wird, durch die das betreffende Element im Programm angesprochen werden kann. Wird etwa der Dienst »Windows-Zeitgeber« auf einem Formular abgelegt, wird eine Komponente mit dem Namen »ServiceController1« (basierend auf der *ServiceController*-Klasse im Namespace *System.ServiceProcess*) angelegt, über die der Dienst gesteuert werden kann. Soll der Dienst während der Programmausführung etwa gestartet werden, genügt dazu der Befehl:

```
If ServiceController1.Status = _
 ServiceProcess.ServiceControllerStatus.Stopped Then
  ServiceController1.Start()
End If
```

Einfacher kann der Umgang mit Systemelementen nicht mehr werden. Soll der Server-Explorer den Inhalt eines Datenbankservers anzeigen, muss dafür eine Datenverknüpfung angelegt werden.

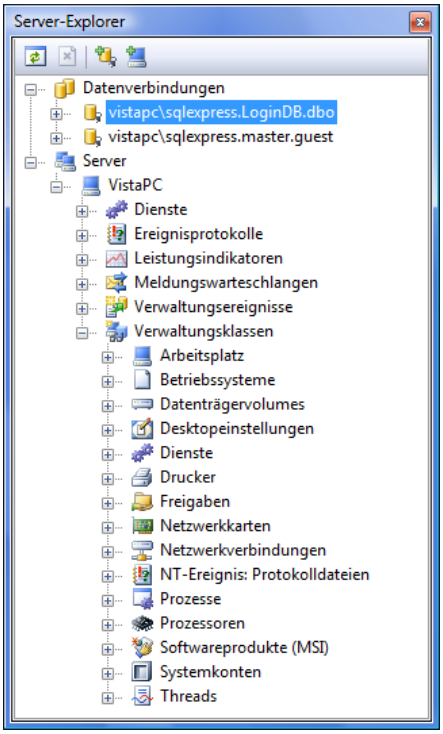


Abbildung 4.6: Der Server-Explorer zeigt neben Datenverbindungen auch die »Bewohner« eines Servers an.

4.2.2 Der Projektmappen-Explorer

Der Projektmappen-Explorer ist so etwas wie das Steuerpult eines Projekts, denn hier werden alle zum Projekt gehörenden Dateien angezeigt. Der Projektmappen-Explorer besitzt zwei Gesichter. Standardmäßig werden nicht alle Dateien des Projekts angezeigt. Die verschiedenen Designer-, Ressourcen- und Config-Dateien werden erst über einen Klick auf das Symbol *Alle Dateien anzeigen* sichtbar gemacht.

Das dürfte am Anfang nicht unbedingt auffallen: Der Projektmappen-Explorer selektiert automatisch jenes Modul, das über den Registerkartenreiter geöffnet wird. Das ist gerade bei großen Projekten sehr praktisch, die aus vielen Dateien und Ordnern bestehen. Dieses Verhalten wird über die Einstellung Aktives Element im Projektmappen-Explorer überwachen *in den IDE-Optionen gesteuert.*

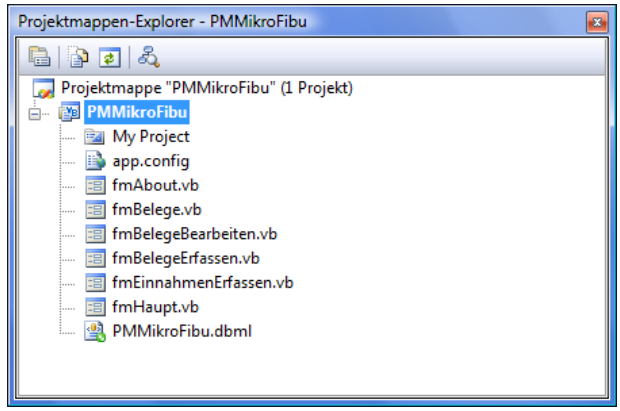


Abbildung 4.7: Der Projektmappen-Explorer zeigt alle Dateien eines Projekts an – die meisten erst auf Aufforderung.

4.2.3 Der Objektbrowser

Der Objektbrowser zeigt unter anderem alle Klassen an, die zu allen Projekt-Assemblies gehören, die im Moment entweder direkt bearbeitet werden oder auf welche die Projekte der Projektmappe Verweise gesetzt haben[1]. Der Objektbrowser ist eine rein passive Einrichtung, in der die Inhalte der einzelnen Assemblies nur angezeigt werden. Es ist am Anfang sehr wichtig zu verstehen, dass der Objektbrowser die vorhandenen Klassen entweder nach Assemblies (auch Container genannt) oder nach Namespaces unterteilt anzeigt (im Unterschied zu seinem Vorgänger kann dies eingestellt werden). Wenn Sie den Eintrag »System« öffnen, sehen Sie alle Klassen dieses Namespace, unabhängig davon, aus welcher Assembly sie stammen.

Es ist im Allgemeinen die bessere Wahl, den Inhalt des Objektbrowsers nach Namespaces – statt nach Bibliotheken – zu sortieren, da jede Klasse zu einem Namespace gehört, eine Assembly-Bibliothek aber Klassen aus mehreren Namespaces enthalten kann. Praktisch ist die Möglichkeit, über einen Button (rechts neben dem Suchknopf) einen Verweis auf jene Assembly-Bibliothek hinzufügen zu können, in der sich eine über die Suchfunktion gefundene Klasse befindet. Per Standardeinstellung werden alle Assemblies durchsucht, es lässt sich aber auch ein »benutzerdefinierter Komponentensatz« zusammenstellen, wenn nur bestimmte Bibliotheken durchsucht werden sollen.

Den Objektbrowser durchsuchen

Über eine Suchfunktion lässt sich jede Klasse, Schnittstelle, Eigenschaft usw. im Objektbrowser schnell lokalisieren. Die Suchergebnisse werden in der rechten Fensterhälfte aufgelistet, sodass die Ansicht nicht gewechselt werden muss. Soll das Ergebnis wieder »verschwinden«, muss die Suche gelöscht werden.

1 Eigentlich müsste er »Typbrowser« heißen, da natürlich keine Objekte, sondern Klassen, Konstantenlisten und Schnittstellen angezeigt werden.

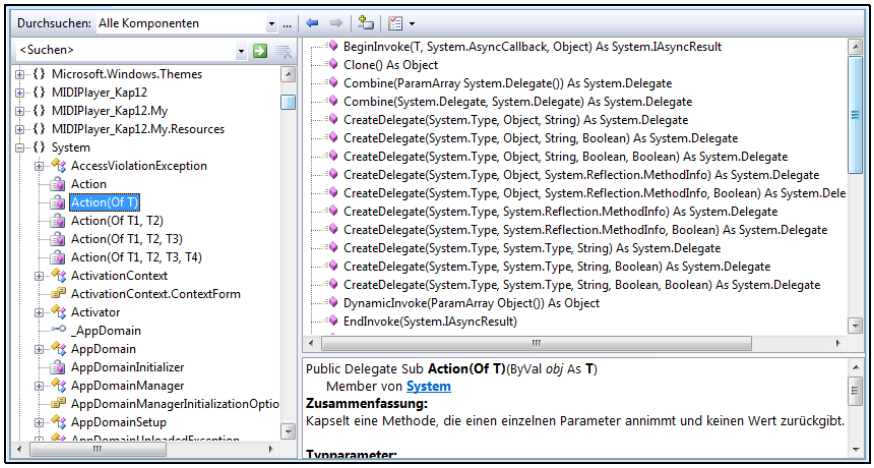


Abbildung 4.8: Der Objektbrowser zeigt die Klassen aller Assemblies an, auf die das Projekt Verweise hält.

Weitere Bibliotheken hinzufügen

Per Voreinstellung werden alle »Standard-Assembly-Bibliotheken« durchsucht. Sollen weitere Bibliotheken durchsucht werden, geschieht dies über das Anlegen einer benutzerdefinierten Komponentenauswahl über den Listeneintrag »Benutzerdefinierten Komponentensatz bearbeiten«.

Einstellungen beim Objektbrowser ändern

Damit es ein wenig übersichtlicher wird, werden von Anfang an nicht alle Mitglieder angezeigt. Dies kann in den Einstellungen des Objektbrowsers festgelegt werden.

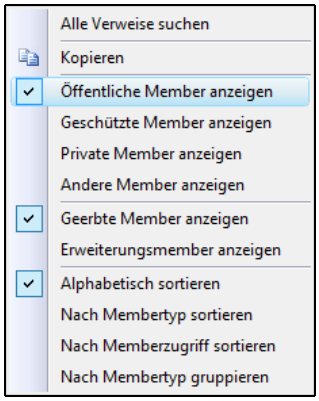
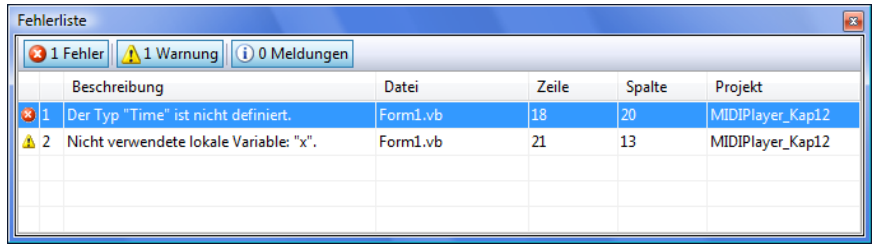


Abbildung 4.9: Der besseren Übersichtlichkeit halber werden beim Objektbrowser von Anfang an nicht alle Mitglieder angezeigt.

4.2.4 Die Fehlerliste

Die Fehlerliste zeigt alle aktuellen Fehler und Warnungen des Compilers bezogen auf den Quellcode des aktuellen Projekts.

Abbildung 4.10: Die Fehlerliste zeigt alle aktuellen Fehler und Warnungen im Quelltext an.

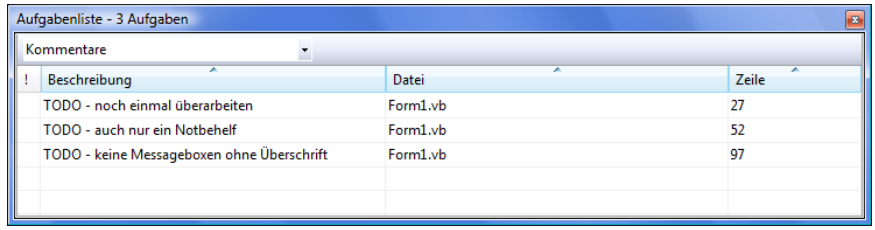


Klicken Sie einen Eintrag doppelt an, gelangen Sie an die Stelle im Quelltext, die den Fehler bzw. die Warnung verursacht hat.

4.2.5 Die Aufgabenliste

Die Idee der Aufgabenliste ist, dass Visual Studio hier Einträge einfügt, die der Programmierer (also Sie) erledigen soll. Allerdings denkt sich Visual Studio keine Aufgaben nach dem Motto »Mach mal zehn Liegestützen zum Warmwerden« aus, die Aufgaben werden z. B. vom Aktualisierungs-Assistenten nach der Aktualisierung eines Projekts eingetragen und besitzen keinen verbindlichen Charakter. Auch Kommentare erscheinen in der Aufgabenliste, wenn sie ein bestimmtes »Kommentartoken« wie z. B. »TODO« oder »HACK« enthalten. Die Aufgabenliste wird über ANSICHT|WEITERE FENSTER|AUFGABENLISTE (oder die Tastenkombination Strg + Alt + K bei VB6-Tastenbelegung und Strg + ^ + Strg + T bei Standardtastenbelegung) sichtbar gemacht.

Abbildung 4.11: Die Aufgabenliste zeigt auch Kommentare mit einem Kommentartoken an.



4.2.6 Das Eigenschaftenfenster

Das Eigenschaftenfenster zeigt die Eigenschaften (und Ereignisse, sofern vorhanden) des aktuell selektierten Objekts an. Am schnellsten wird es über F4 geöffnet, wenn das Element zuvor selektiert wurde. Besonders praktisch ist das Einfügen einer Ereignisprozedur, denn dazu muss das Ereignis lediglich doppelt angeklickt werden. Doch welches Ereignis? Jene Ereignisse, die bei einem Windows Forms-Formular in der Kategorie *Ereignisse* angezeigt werden. Auch das kann schnell übersehen werden. Das Eigenschaftenfenster zeigt auch die Attribute einer Klasse, Eigenschaften usw. an. Schalten Sie dazu auf das Codefenster um, setzen Sie die Textmarke zum Beispiel in einen *Class*-Befehl, und drücken Sie F4, um das Eigenschaftenfenster zu öffnen. Sie sehen dort die vordefinierten Attribute, die dem *Class*-Befehl zugeordnet werden können. Klicken Sie zum Beispiel *COM-sichtbar* doppelt an, wird das *ComVisible*-Attribut aus dem Namespace *System.Runtime.InteropServices* eingefügt (mehr zu den Attributen in Kapitel 12).

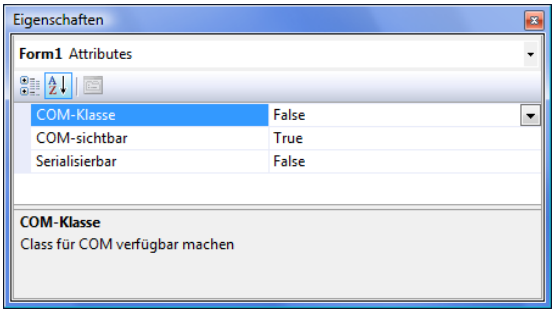


Abbildung 4.12: Im Eigenschaftenfenster werden auch Codeattribute angezeigt.

Das Eigenschaftenfenster steht unter dem Namen »PropertyGrid« auch als Control in der Toolbox zur Verfügung. Damit können Anwendungen das Einstellen von Eigenschaften ihrer internen Objekte zur Ausführungszeit auf die gleiche Weise anbieten, wie es Visual Studio tut.

TIPP

4.2.7 Die Toolbox

Die Toolbox von Visual Studio bietet alle Steuerelemente (engl. »Controls«) und Komponenten an, die zur Entwurfszeit auf der Oberfläche eines Formulars oder allgemein einer Designeroberfläche platziert werden können. Der besseren Übersichtlichkeit halber ist die Toolbox in verschiedene Sektionen unterteilt. Es lassen sich jederzeit neue Sektionen hinzufügen und Elemente zwischen den einzelnen Sektionen verschieben. Anders als bei früheren Visual Basic-Versionen ist das Platzieren von Steuerelementen über die Toolbox bei .NET lediglich eine Option, denn jedes Steuerelement und jede Komponente kann theoretisch auch direkt im Quellcode angelegt werden, wenngleich dies keine Vorteile bringt.

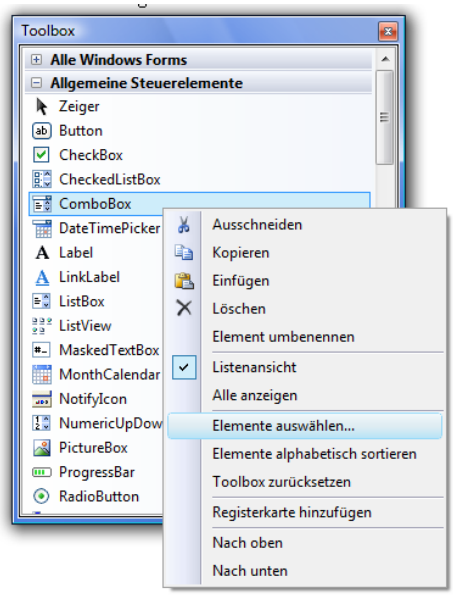


Abbildung 4.13: Die Toolbox ist erweiterbar.

Die Toolbox bietet über ihr Kontextmenü noch einige Kleinigkeiten an, die aus Platzgründen nur kurz erwähnt werden können:

- Es lassen sich alle Elemente über *Alle Anzeigen* auf einmal sichtbar machen (dann wird es richtig voll, da auch die alten Steuerelemente von VS.NET 2003 wieder erscheinen, ohne dass es unübersichtlich wird).
- Der »Grundzustand« der Toolbox wird über den *Toolbox zurücksetzen*-Befehl wiederhergestellt.
- Die Elemente lassen sich alphabetisch auflisten.
- Es lassen sich neue Registerkarten (Sektionen) anlegen und die Mitglieder einer Sektion per Maus in eine andere Sektion verschieben.

Bliebe noch zu erwähnen, dass, anders als bei VB6, die Toolbox-Belegung nicht mit dem Projekt gespeichert wird. Es stehen also immer alle Elemente zur Verfügung. Allerdings ist die Auswahl projektabhängig. In einem Webprojekt erscheinen (natürlich) andere Steuerelemente als in einem Windows Forms-Projekt. Gar nicht sichtbar ist die Toolbox in einer Konsolenanwendung, solange kein Formular hinzugefügt wurde.

Texte ablegen

In der Toolbox lassen sich auch Texte ablegen, was überaus praktisch ist, da sie so einfach wieder in den Quelltext eingefügt werden können. Das ist oft der einfachste und direkteste Weg, eine »Wiederverwendbarkeit« von Programmcode zu erreichen. Insbesondere immer wieder benötigte Variablendeklarationen sind hier gut aufgehoben. Tipp: Benennen Sie einen Eintrag um, damit wird klarer, was sich dahinter verbirgt.

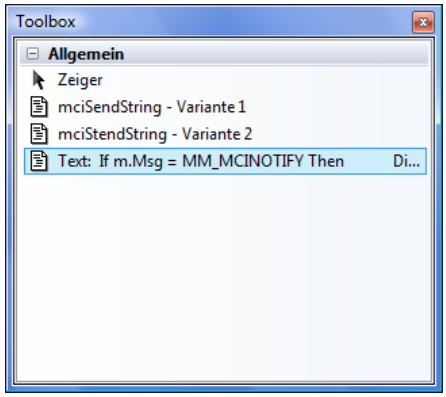


Abbildung 4.14: Die Toolbox kann auch Textausschnitte aufnehmen.

4.2.8 Das Lesezeichenfenster

Ein Lesezeichen ist eine Markierung im Quelltext, durch die sich die betreffende Stelle direkt aufrufen lässt. Gerade bei großen Quelltextmengen können diese Lesezeichen sehr praktisch sein. Alle gesetzten Lesezeichen werden im Lesezeichenfenster aufgelistet. Ein Doppelklick auf einen Eintrag bewirkt, dass die Stelle im Quellcode angesprungen wird, an welcher der Eintrag definiert ist. Dies ist oft der schnellste Weg, um eine Stelle im Quellcode zu lokalisieren.

4.2.9 Die Klassenansicht

Die Klassenansicht zeigt – das legt der Name nahe – alle Klassen eines Programms an. Da jedes Visual Basic-Programm ausschließlich aus Klassen besteht, gibt es auch die Programmstruktur schön wieder, da auch eine eventuell existierende Hierarchie der Klassen berücksichtigt wird.

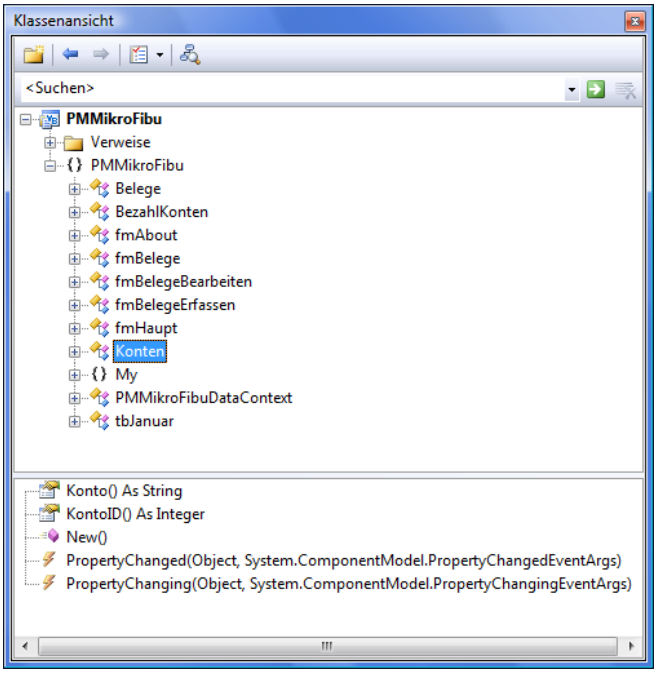


Abbildung 4.15: Die Klassenansicht zeigt alle Klassen und deren Mitglieder an.

4.2.10 Der Klassendesigner

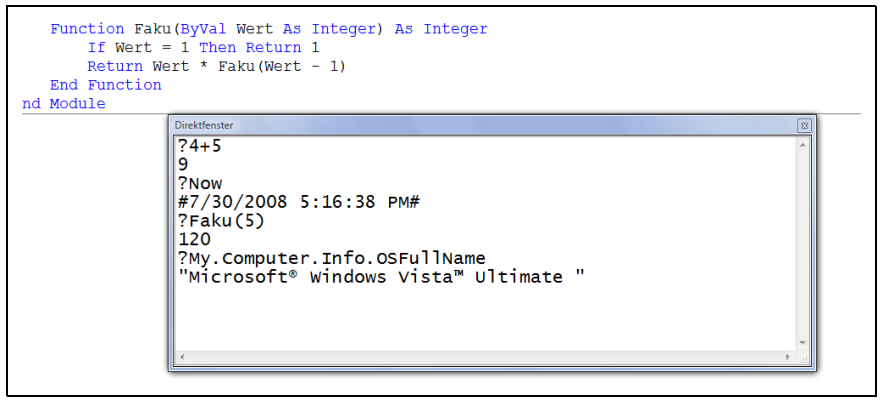
Einen Schritt weiter als die Klassenansicht geht der Klassendesigner, den es leider nicht in der Express Edition gibt. Der Klassendesigner ermöglicht es, die Klassen eines Programms und ihre Mitglieder visuell zu »designen«, wobei im Hintergrund der entsprechende Quellcode angelegt wird. Da sich aus einer bereits vorhandenen Klassenstruktur jederzeit ein Klassendiagramm ableiten lässt, spielt es keine Rolle, ob die Klassen einer Anwendung von Anfang an oder zu einem späteren Zeitpunkt mit dem Klassendesigner erstellt werden. Mehr zum Klassendesigner in Kapitel 6, wenn es um das Thema objektorientierte Programmierung geht.

4.2.11 Das Direktfenster

Das Direktfenster ist eine sehr praktische Einrichtung, denn hier können sowohl während einer Programmunterbrechung als auch im »Leerlauf«, wenn noch kein Programm gestartet wurde, Befehle eingegeben werden, die dann sofort ausgeführt werden. Allerdings gibt es im Vergleich zum regulären Ausführen eines Befehls ein paar kleinere Unterschiede. Voraussetzung ist, dass sich der Quelltext des Projekts fehlerfrei kompilieren lässt. Dann lassen sich im Direktfenster auch einzelne Funktionen des Programms aufrufen. Variablen können nur eingeschränkt deklariert wer-

den, ein *A = 123* ist erlaubt, nicht aber ein *Dim A As Byte = 123*. Dafür stehen z. B. die Shared-Member der Basisklassen zur Verfügung. Der universelle Ausgabebefehl ist das ?, sodass z. B. der Befehl *?Environment.GetEnvironmentVariable(»path«)* den Wert der Umgebungsvariablen *Path* ausgibt.

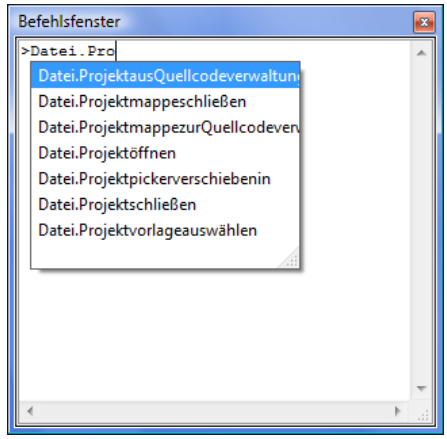
Abbildung 4.16: Im Direktfenster lassen sich Berechnungen durchführen und Programmfunktionen sowie Funktionen der Basisklassen aufrufen.



4.2.12 Das Befehlsfenster

Das Befehlsfenster erlaubt die Eingabe von IDE-Befehlen (und nicht Programmbefehlen). Die IDE-Befehle werden so eingegeben, wie sie sich durch den Menünamen und den entsprechenden Menüeintrag ergeben, wobei die passende Auswahl bereits nach Eingabe der ersten Buchstaben erscheint.

Abbildung 4.17: Die Visual Studio-IDE lässt sich über das Befehlsfenster steuern.



4.2.13 Das Ausgabefenster

Das Ausgabefenster zeigt sowohl jene Meldungen an, die vom Compiler beim Erstellen des Projekts produziert werden, als auch jene Debug- und Trace-Meldungen, die während der Programmausführung z. B. über die *WriteLine*-Methode der *Debug*-Klasse (Namespace *System.Diagnostics*) ausgegeben werden. Dass diese Meldungen scheinbar immer im Direktfenster landen, liegt an einer Einstellung in den Optionen der IDE, die diese vom Ausgabe- ins Direktfenster umleitet, wo sie beim Debuggen etwas besser aufgehoben sind, da das Direktfenster das wichtigste Fenster beim Debuggen ist.

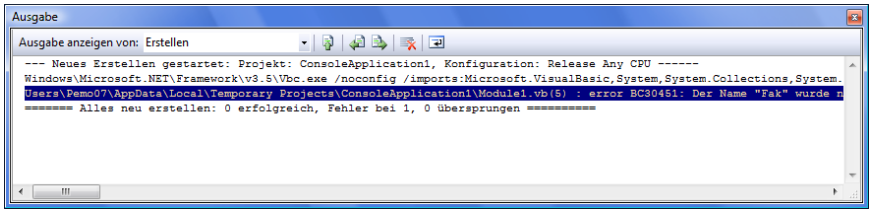


Abbildung 4.18: Im Ausgabefenster meldet sich der Compiler zu Wort.

4.3 Die Rolle der Projektvorlagen

Ein Projekt ist bei Visual Studio ein Name, der für alle Dateien steht, die durch das Projekt zusammengefasst und die durch den Compiler zu einer Assembly kompiliert werden (im Hintergrund arbeitet ein Tool mit dem Namen *MsBuild.exe*, das die »Anweisungen« der Projektdatei umsetzt und das in Kapitel 12 kurz vorgestellt wird). Damit das Grundgerüst eines Projekts nicht jedes Mal aufgebaut werden muss, gibt es die Projektvorlagen. Eine Projektvorlage umfasst neben den Dateien, die von Anfang an Teil des Projekts sind, auch die Projekteinstellungen, die in der IDE für jedes Projekt unter *Projekt*|*<Projektname>-Eigenschaften* getroffen werden. Visual Studio bietet von Anfang an eine große Auswahl an Projektvorlagen für alle wichtigen Anwendungstypen an. Die Liste ist natürlich erweiterbar, allerdings setzt das Erstellen einer neuen Projektvorlage ein wenig Erfahrung voraus und wird durch Visual Studio nicht unterstützt. Der einfachste Weg, an eine eigene Projektvorlage zu kommen, besteht daher darin, ein vorhandenes Projekt als Projektvorlage zu speichern, was über *Datei*|*Vorlage exportieren* eine Kleinigkeit ist. Tabelle 4.1 stellt die wichtigsten Projektvorlagen zusammen.

Projektvorlage	Welche Rolle spielt sieer?	Besonderheiten
Windows Forms-Anwendung	Standard-Projekttyp für Windows-Anwendungen.	Keine
Klassenbibliothek	Eine Klassenbibliothek ist eine Assembly, die aus mindestens einer öffentlichen Klasse besteht.	Keine
Konsolen-anwendung	Eine Konsolenanwendung läuft in einer Win32-Konsole und nimmt Eingaben über das StdIn-Gerät entgegen und führt Ausgaben über das StdOut-Gerät durch.	Auch eine Konsolenanwendung kann Fenster anzeigen.
Windows Forms-Steuerelement-bibliothek	Steht für ein eigenes Steuerelement.	Steuerelemente können auch projektbezogen definiert werden und bestehen in diesem Fall nur aus einer Klasse.
Windows-Dienst	Ein Windows-Dienst läuft als »unsichtbare« Anwendung im Hintergrund unter einem bestimmten Systemkonto und wird bereits mit dem Windows-Start aktiv.	Windows-Dienste besitzen keine Oberfläche, können aber ansonsten beliebige Aufgaben übernehmen.
Leeres Projekt	Enthält keine Dateien und keine Voreinstellungen.	Ist sehr viel praktischer, als es sich zunächst anhören mag.

Tabelle 4.1: Die wichtigsten Visual Studio-Projekttypen für Windows

4.3.1 Wo werden Projektdateien abgelegt?

Jedes Projekt wird in einem eigenen Verzeichnis abgelegt. Damit die Projektverzeichnisse nicht über die ganze Festplatte verstreut oder (wie bei früheren Visual Basic-Versionen) per Voreinstellung im Visual Studio-Verzeichnis landen, werden sie im Ordner *Visual Studio 2008\Projects* im Benutzerprofil des Anwenders abgelegt. Grundsätzlich spielt es keine Rolle, wo ein Projektverzeichnis angelegt wird. Der einzige Aspekt, der bei der Ablage berücksichtigt werden muss, ist der Umstand, dass Netzwerklaufwerke am Anfang keine gute Wahl sind, da ein solches Projekt weniger Berechtigungen erhält und eine entsprechende Codezugriffsrichtlinie angelegt werden müsste.

4.3.2 Speichern oder Verwerfen?

Wurde ein Projekt nicht offiziell gespeichert, gibt es beim Verlassen der IDE (oder beim Neuanlegen eines Projekts) die Gelegenheit, es zu »verwerfen« oder zu speichern. Verwerfen bedeutet, dass die temporär angelegten Projektdateien wieder gelöscht werden. Soll ein Projekt mit dem Erstellen gespeichert werden, muss dies über die Einstellung »Alle Änderungen speichern« in den Optionen der IDE unter *Projekte und Projektmappen|Erstellen und Ausführen* aktiviert werden.

Wundern Sie sich nicht darüber, dass manche Dateien scheinbar zwei Erweiterungen besitzen, z. B. »Form1.Designer.vb«. Da ist ein allgemeines Merkmal von Dateinamen bei Windows. Es wird nur relativ selten benutzt. Ein Dateiname kann beliebig viele Punkte besitzen. Als Erweiterung werden nur die Zeichen nach dem letzten Punkt interpretiert.

4.3.3 Automatische Sicherung der Projektdateien

Wie bei einer Textverarbeitung bietet auch Visual Studio die Möglichkeit, die Projektdateien regelmäßig zu sichern, sodass sie nach einem Absturz der IDE automatisch wiederhergestellt werden können[2]. Die Einstellung wird in den Optionen der IDE unter *Umgebung|Autowiederherstellen* vorgenommen (dazu müssen alle Einstellungen sichtbar sein) und sollte wahrgenommen werden.

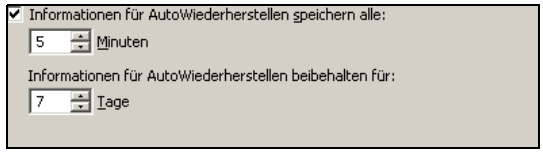


Abbildung 4.19: Für das automatische Sichern der Projektdateien kann ein Intervall eingestellt werden.

4.4 Der Aufbau eines Projekts

Ein Projekt besteht, je nach Typ, aus unterschiedlichen Dateien. Eine Windows Forms-Anwendung besteht aus einer Formularklasse *Form1.vb*, eine Konsolenanwendung aus einer Klasse *Module1.vb*. Mit der Vorlage Windows Forms-Anwendung werden für das Projekt insgesamt zehn Dateien angelegt, von denen aber nur zwei, die Formulardatei und die dazugehörige Designerdatei, wirklich wichtig sind. Regelrecht irritierend dürfte am Anfang sein, dass eine Formulardatei zunächst total aufgeräumt erscheint und lediglich aus einem unscheinbaren *Class*-Befehl besteht.

2 Leider ist die Absturztendenz von VS 2008, vor allem unter Vista, deutlich größer als noch bei der Vorgängerversion (die unter Vista offiziell gar nicht lief).

Schuld sind die partiellen Klassen, durch die eine Klasse auf mehrere Module verteilt werden kann. Die vom Designer generierten Befehle werden in einer speziellen Datei mit der Erweiterung *.Designer* untergebracht, die im Projektmappen-Explorer zunächst unsichtbar ist (und vom Programmierer auch nicht editiert werden sollte).

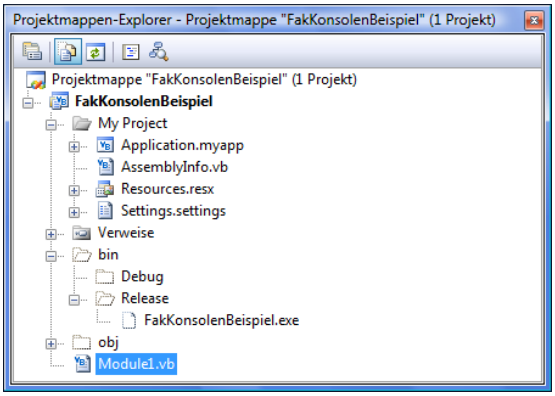


Abbildung 4.20: Der Projektmappen-Explorer zeigt alle Dateien an, die zu einem Projekt gehören.

4.4.1 Die Eigenschaften eines Projekts

Die umfangreichen Einstellungen eines Projekts werden bei Visual Studio in einem Projektunterverzeichnis mit dem Namen *My Project* in verschiedenen Dateien gespeichert (Tabelle 4.2 zeigt eine Übersicht). Diese werden von der IDE angelegt und sollten nicht geändert werden. Der Vorteil dieser Dateien ist, dass sie im Quelltext vorliegen und dass es sich bei den Dateien mit der Erweiterung *.Vb* um reguläre Quelltextdateien handelt, die zum Beispiel auch außerhalb der IDE kompiliert werden könnten. Dieses offene Prinzip zieht sich wie ein roter Faden durch alle Bereiche der IDE und ist ein angenehmer Kontrast zu früheren Visual Basic-Versionen, wo die meisten Einstellungen intern vorgenommen wurden.

Tabelle 4.2: In diesen Dateien werden Projekteinstellungen gespeichert.

Datei	Bedeutung
Application.Designer.vb	Quelltextdatei, in der Einstellungen gespeichert werden, die im Register Anwendung der Projekteigenschaften getroffen werden.
Application.myapp	Enthält die Einstellungen des Registers Anwendung der Projekteigenschaften im XML-Format.
AssemblyInfo.vb	Quelltextdatei, die allgemeine Projekteinstellungen wie Titel, Beschreibung und Versionsnummer in Gestalt von Attributen enthält. Diese Datei gab es bereits bei VS.NET 2003.
Resources.Designer.vb	Quelltextdatei, die den Namespace My.Resources um die Möglichkeit erweitert, auf die Ressourcen der Anwendung zuzugreifen.
Resources.resx	Ressourcendatei, in der die Ressourcen der Anwendung definiert sind.
Settings.Designer.vb	Quelltextdatei, die *Settings* im Namespace *My* um Eigenschaften erweitert, die einen Zugriff auf die im Register *Einstellungen* eingetragenen Werte erlauben.
Settings.settings	Datei im XML-Format, in der die im Register *Einstellungen* der Projekteigenschaften angelegten Variablen gespeichert werden.

Die Verweise (Referenzen) auf andere Assemblies werden im Projektmappen-Explorer nicht mehr von Anfang an angezeigt. Möchte man sie sehen, muss man auf *Alle Dateien anzeigen* klicken.

4.5 Der Umgang mit Projekten – die Projektverwaltung

Bei Visual Studio dreht sich alles um Projekte. Ein Projekt ist eng mit jenem Ordner verknüpft, der von Visual Studio beim Anlegen eines neuen Projekts automatisch angelegt wird. Visual Studio überwacht diesen Ordner (per Voreinstellung) und erkennt Änderungen, die außerhalb der IDE an einzelnen Dateien im Projektverzeichnis vorgenommen werden, sodass der Projektinhalt immer auf dem neusten Stand ist. Nicht jede Datei, die in dem Projektordner enthalten ist, muss zum Projekt gehören. Welche Datei im Projektordner zum Projekt gehört und welche nicht, bestimmt der Inhalt der Projektdatei. Das ist eine Textdatei mit der Erweiterung *.Vbproj*. Wer mit Notepad einen Blick in die Datei wirft, wird feststellen, dass sie nicht einfach die Namen der Dateien enthält, sondern sehr viel Text, der auf den ersten Blick recht verworren erscheinen mag. Dabei ist auch hier alles ganz einfach. Eine Projektdatei ist bereits seit Visual Studio 2005 eine XML-Datei im MSBuild-Format, das in Kapitel 12 vorgestellt wird. Da es nur selten einen Grund gibt, eine Vbproj-Datei direkt zu editieren, muss man sich für die Details nicht interessieren. Nicht jede Datei, die zum Projekt gehört, muss beim Erstellen des Projekts übersetzt werden. Über die Eigenschaft *BuildVorgang* wird eingestellt, ob eine Quelltextdatei kompiliert wird oder das Projekt lediglich begleitet.

4.5.1 Dateiüberwachung

Die IDE überwacht (per Voreinstellung) alle zum Projekt gehörenden Dateien, sodass Änderungen, die außerhalb der IDE an einer Datei durchgeführt werden, registriert werden und Sie die Möglichkeit erhalten, die Änderungen zu übernehmen oder zu ignorieren. Ist dieses Verhalten nicht erwünscht, kann es in den Projektoptionen deaktiviert werden.

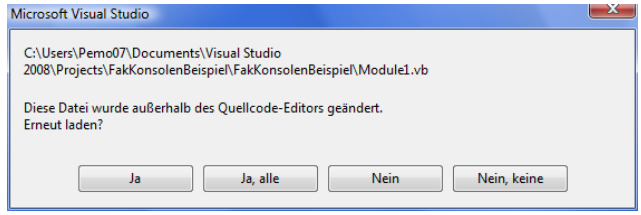


Abbildung 4.21: Diese Meldung erscheint, wenn eine Projektdatei außerhalb der IDE geändert wurde.

4.5.2 Anlegen von Verzeichnissen

Im Projektmappen-Explorer können durch Anklicken eines Eintrags mit der rechten Maustaste und Auswahl von *Hinzufügen|Neuer Ordner* neue Ordner angelegt werden. Das ist zum Beispiel sinnvoll, wenn bestimmte Datendateien gruppiert werden sollen (etwa Bitmaps, die nicht als Ressourcen Teil der Assembly sein sollen). Auch das Anlegen von Ordnern für Quelltextdateien (etwa Klassen) kann sinnvoll sein, da

es die Übersichtlichkeit erhöht und es etwas leichter macht, Gruppen von Dateien zwischen Projekten auszutauschen. Für die Frage, ob eine Quelltextdatei kompiliert wird, spielt es keine Rolle, in welchem Ordner sie sich befindet.

4.5.3 Dateien – nur dabei oder mittendrin?

Bei jeder Datei kann in ihren Eigenschaften eingestellt werden, ob sie Teil der Assembly werden soll oder nicht. Im Allgemeinen gibt es keinen Grund, die Voreinstellung zu ändern. Ausnahmen sind Textdateien mit Programmeinstellungen, die Teil der Assembly werden sollen, damit sie nicht als separate Dateien vorliegen, oder Quelltextdateien, die nicht Teil der Assembly sein sollen (sie dürfen dann aber auch nicht im Programm angesprochen werden). Dass eine Datei eingebettet wird, bedeutet aber nicht, dass sie als Datei angesprochen werden kann.

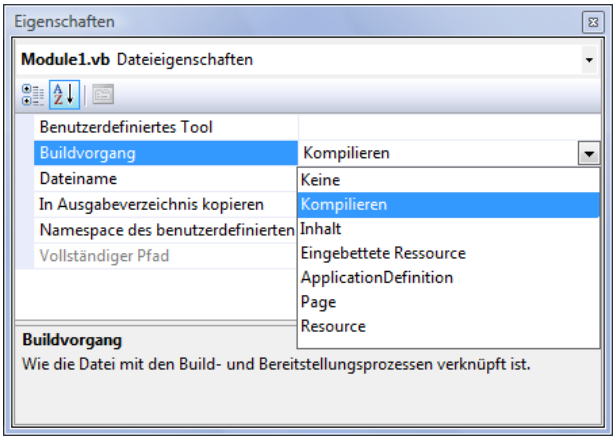


Abbildung 4.22: In den Dateieigenschaften wird festgelegt, ob eine Datei mitkompiliert werden soll oder nicht.

Soll eine Datendatei Teil der Assembly und der Inhalt dieser Datei während der Programmausführung eingelesen werden, muss für BuildVorgang *»Eingebettete Ressource« gewählt werden. Während der Programmausführung wird der Inhalt über die* GetManifestResourceStream*-Methode der* Assembly*-Klasse als Stream eingelesen und im folgenden Beispiel in ein DataSet gelesen, das wiederum mit einem DataGridView verbunden wird, sodass der Dateninhalt der XML-Datei im Grid angezeigt wird:*

```
Imports System.Reflection
Imports System.IO

Dim Ds As New DataSet
Using St As Stream =
Assembly.GetExecutingAssembly.GetManifestResourceStream("WindowsApplication2.Daten.xml"
)
  Ds.ReadXml(St)
End Using
DataGridView1.DataSource = Ds.Tables(0)
```

Der Name der Ressource setzt sich aus dem Stamm-Namespace der Assembly (wird in den Projekteigenschaften eingestellt und entspricht per Voreinstellung dem Projektnamen) und dem Namen der eingebetteten Datei zusammen.

4.5.4 Dateien von A nach B kopieren über Pre- und Postbuild-Events

Sollen vor oder nach dem Erstellen eines Projekts besondere Aktionen mit einzelnen Projektdateien ausgeführt werden, geschieht dies über die sog. *Buildereignisse*. Die Aktionen werden im Register *Kompilieren* der Projekteigenschaften über *Buildereignisse* eingestellt. Es gibt Ereignisse, die vor dem Erstellen, und Ereignisse, die nach dem Erstellen ausgeführt werden. Als Kommandos kommen alle Betriebssystemkommandos infrage, sodass es zum Beispiel möglich ist, nach dem Erstellen Dateien per *Copy*-Befehl an einen anderen Ort zu kopieren oder beliebige Programme auszuführen (dazu muss der *Copy*-Befehl indirekt über *Cmd.exe* ausgeführt werden). Für größere Projekte werden richtige Skripte festgelegt, die eine Reihe von Aktionen ausführen. Die benötigten Verzeichnispfade und Dateinamen werden über Platzhalter zur Verfügung gestellt.

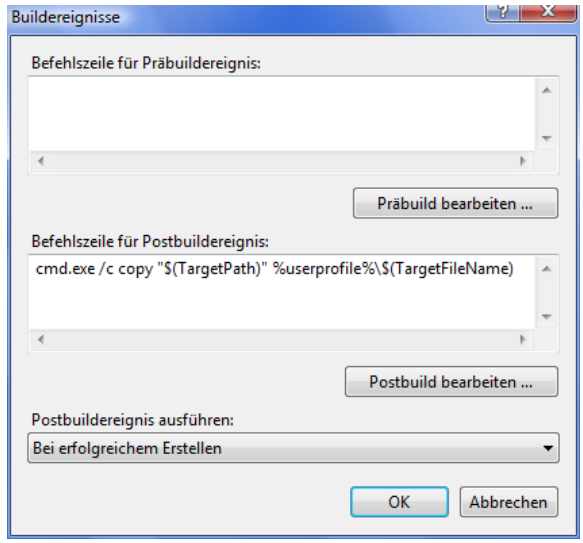


Abbildung 4.23: Über Buildereignisse kann eingestellt werden, was vor oder nach dem Erstellen eines Projekts passieren soll.

4.5.5 Der Konfigurationsmanager

Jedes Projekt kann in zwei Modi erstellt werden: Debug und Release. Der wichtigste Unterschied ist, dass im Debug-Modus symbolische Informationen in die Assembly eingefügt werden, die ein Debuggen auf Quelltextebene ermöglichen. Visual Studio legt für jedes Projekt auch eine Projektmappendatei (Erweiterung *.sln*) an. Allerdings wird die Projektmappe nur angezeigt, wenn dies in den Projekteigenschaften festgelegt wird. Auch der *Konfigurationsmanager* muss erst über die Option *Erweiterte Buildkonfigurationen anzeigen* unter *Projekte und Projektmappen|Allgemein* im Menü *Extras\Optionen* aktiviert werden. Über ihn kann pro Projekt eingestellt werden, ob und in welchem Modus es erstellt werden soll. Der Konfigurationsmanager wird über *Erstellen|Konfigurationsmanager* sichtbar gemacht.

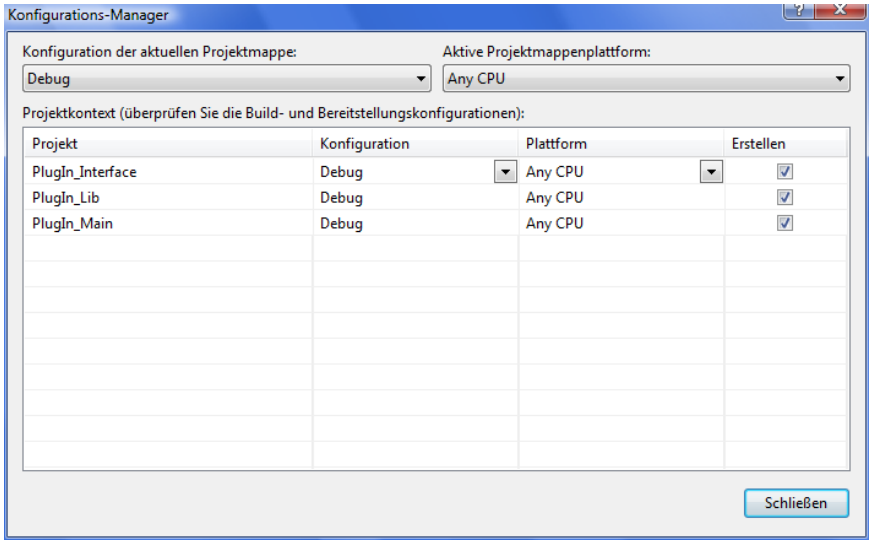


Abbildung 4.24: Der Konfigurationsmanager erlaubt zu jedem Projekt der Projektmappe die Auswahl einer Konfiguration.

Auch wenn es im Konfigurationsmanager möglich ist, neue Projektkonfigurationen anzulegen, hat dies für die Praxis keine echte Bedeutung. Auch die Auswahl der aktiven Projektmappenplattform beschränkt sich auf »Any CPU«.

INFO

Eine leere Projektmappe wird durch Auswahl des Projekttyps »Andere Projekttypen|Visual Studio-Projektmappen« angelegt.

INFO

Festlegen der Buildreihenfolge

Enthält eine Projektmappe mehrere Projekte, ist es über *Projekt|Projektbuildreihenfolge* möglich, die Reihenfolge festzulegen, in der die einzelnen Projekte übersetzt (nicht ausgeführt) werden. Dies geschieht aber nicht direkt, sondern indirekt, in dem im Register *Abhängigkeiten* angegeben wird, von welchen anderen Projekten das ausgewählte Projekt abhängig ist. Dadurch wird die Buildreihenfolge so festgelegt, dass die Projekte zuerst übersetzt werden, von denen andere Projekte abhängig sind.

Festlegen der Startreihenfolge

Auch das Festlegen der Startreihenfolge ist möglich. Dies geschieht in den Eigenschaften der Projektmappe. Hier kann auch eingestellt werden, dass gleich mehrere Startprojekte mit dem Drücken von F5 starten.

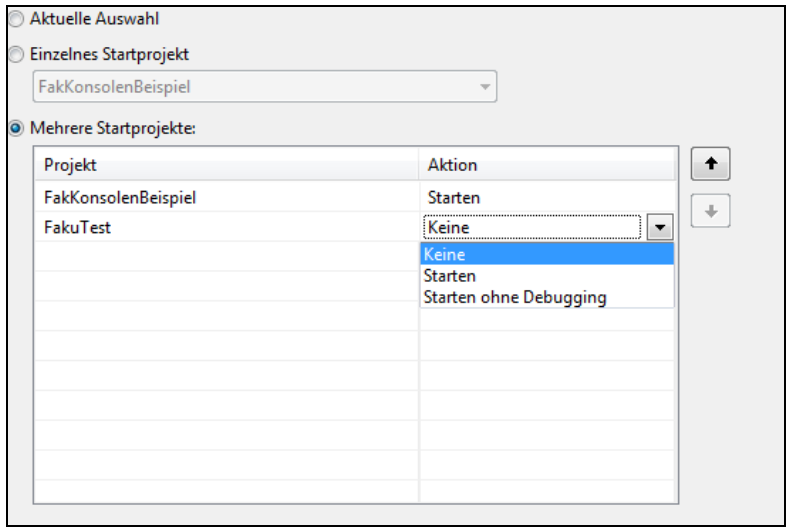


Abbildung 4.25: In den Eigenschaften der Projektmappe kann auch die Startreihenfolge der einzelnen Projekte festgelegt werden.

Die Projektmappendateien (Solution)

Neben den Projektdateien gibt es bei Visual Studio die Projektmappendateien (Erweiterung *.sln*). Sie fassen ein oder mehrere Projekte zusammen und haben nur eine geringere Bedeutung, da sich jedes Projekt auch einzeln laden ließe. Ein kleiner Vorteil ist, dass sich über den Konfigurationsmanager einstellen lässt, welche Projekte beim Einrichten der Projektmappe erstellt werden sollen (es ist ferner möglich einzustellen, welche Projekte auch ausgeführt werden sollen). Ein wenig verwirrend ist, dass Visual Studio auch dann eine Projektmappendatei anlegt, wenn nur ein Projekt im Spiel ist, diese aber in der IDE zunächst nicht anzeigt (dies kann in den Optionen eingestellt werden). Vielen Neulingen ist daher am Anfang die Rolle der Solution-Datei nicht klar.

Man könnte mal wieder sagen: typisch Microsoft, dass der Projektmappen-Explorer am Anfang nur die »halbe Wahrheit« anzeigt. Über den Button *Alle Dateien anzeigen* im Projektmappen-Explorer muss dies bei jedem Projekt geändert werden.

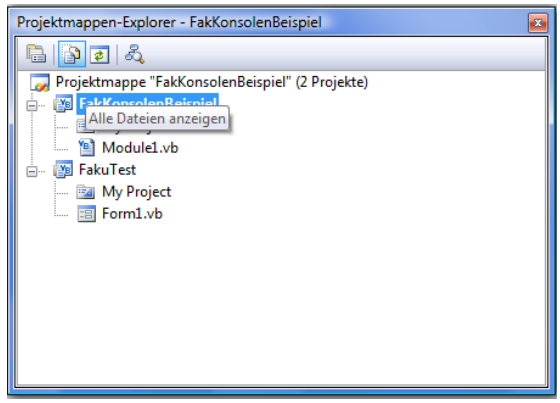


Abbildung 4.26: Der Projektmappen-Explorer zeigt am Anfang nicht alle Dateien an.

Projektmappe immer anzeigen

Soll die Projektmappe im Projektmappen-Explorer immer angezeigt werden, muss in den Optionen der IDE in der Kategorie *Projekte und Projektmappen|Allgemein* die Einstellung *Projektmappe immer anzeigen* gesetzt werden.

4.6 Projekte erstellen und debuggen

Einen Ausführen-Befehl gibt es bei Visual Studio nicht. Ein Projekt wird entweder nur erstellt oder erstellt und danach ausgeführt (wobei beim Erstellen nur jene Quelltextdateien berücksichtigt werden, die seit dem letzten Erstellen geändert wurden). Der Befehl zum Erstellen heißt entweder *Erstellen|Projektmappe erstellen*, wenn die gesamte Projektmappe erstellt werden soll (und diese im *Projektmappen-Explorer* angezeigt wird), oder *Erstellen|<Name des Projekts> erstellen*. Soll das Projekt gleich ausgeführt werden, wählt man den Befehl *Debuggen|Debbuging starten* oder drückt einfach die F5-Taste wie immer.

Hinter dem Drücken von F5 steht eine Reihe von Teilschritten:

- Alle Projekte der Projektmappe werden erstellt (über den Konfigurationsmanager können einzelne Projekte davon ausgenommen werden).
- Es entsteht, je nach Projekttyp, eine .exe- oder .dll-Datei, die im festgelegten Ausgabeverzeichnis abgelegt wird. Je nach Konfiguration ist dies das Projektunterverzeichnis *bin/debug* oder *bin/release*.
- Die als Startprojekt festgelegte Assembly wird gestartet, was standardmäßig nur bei .exe-Dateien möglich ist. In den Projekteigenschaften kann ein Programm festgelegt werden, mit dem die Assembly gestartet wird – dies ist auch für Klassenbibliotheken möglich. Außerdem können im Register *Debuggen* Befehlszeilenargumente eingetragen werden, die beim Aufruf übergeben werden, (und die per *Environment.Commandline* abgefragt werden), und es lässt sich ein Arbeitsverzeichnis festlegen, auf das sich alle relativen Dateipfade innerhalb der Anwendung beziehen.[3]

Erstellen, neu erstellen oder bereinigen?

Ein Projekt kann nicht nur erstellt, sondern auch neu erstellt, bereinigt und veröffentlicht werden. Normalerweise kompiliert der Compiler nur jene Bereiche des Programms, die sich seit dem letzten Kompilieren geändert haben. Soll das Projekt komplett neu erstellt werden, geschieht dies entsprechend über *Neu erstellen*. Bereinigen bedeutet, dass alle eventuell bereits vorhandenen Ausgabedateien gelöscht werden, sodass sich das Projekt wieder in jenem Zustand befindet, in dem es sich befand, bevor es zum ersten Mal erstellt wurde. Ein Projekt zu veröffentlichen bedeutet, dass alle für die Installation erforderlichen Dateien (inkl. der .NET-Laufzeit, wenn gewünscht) in ein (Netzwerk-)Verzeichnis kopiert werden, sodass die Anwendung von dort lokal installiert werden kann. Mehr dazu in Kapitel 24, wenn es um das Ausliefern einer Visual Basic-Anwendung geht.

3 Dies ist jenes Verzeichnis, das z. B. von der CurDir-Funktion geliefert wird.


INFO

Projekte lassen sich nicht ohne Weiteres auf ein Netzwerklaufwerk auslagern, da für sie dann die deutlich niedrigeren Sicherheitseinstellungen der Intranet-Zone *gelten und sie nicht mehr korrekt ausgeführt werden. Das gilt auch für Assemblies, die von einem Netzwerklaufwerk aus aufgerufen werden. Sie erhalten ebenfalls automatisch weniger Berechtigungen, als wenn sie von einem lokalen Laufwerk gestartet würden.*

Debug oder Build?

Die IDE erstellt ein Projekt stets in einem von mehreren Konfigurationsmodi. Von Anfang an gibt es die Modi *Debug* und *Release*, es können jederzeit weitere Modi angelegt werden. Jeder Modus steht für einen eigenen Satz an Projekteinstellungen. Debug- und Release-Modus unterscheiden sich im Wesentlichen dadurch, dass beim Debug-Modus die Symbolinformationen generiert werden, sodass ein Debuggen auf Quelltextebene möglich ist. Außerdem legt der Modus das Ausgabeverzeichnis fest, das beim Debug-Modus *bin\debug* und beim Release-Modus *bin\release* lautet.

Es ist leider ein wenig verwirrend, wie die IDE die beiden Modi verwaltet, da es zwei Orte gibt, an denen die Konfiguration unabhängig voneinander ausgewählt werden kann: das Register *Debuggen* in den Projekteigenschaften (aber nur, wenn die erweiterten Build-Konfigurationen angezeigt werden) und der Konfigurationsmanager. Bei der Frage, in welchem Ausgabeverzeichnis die Assembly abgelegt wird, ist die aktive Konfiguration maßgeblich, die im Konfigurationsmanager festgelegt wird und Vorrang vor der Einstellung in den Projekteigenschaften hat. Es kann daher auch beim Erstellen und Ausführen ohne Debuggen über Strg + F5 passieren, dass die Assembly in *bin\debug* landet, wenn *Debug* die aktive Konfiguration ist.

Ein wenig ungünstig ist ferner, dass der Konfigurationsmanager erst auf Anforderung angezeigt wird. Dafür kann man sich jederzeit weitere Konfigurationsmodi anlegen, was den Vorteil bringt, dass es z. B. verschiedene Ausgabeverzeichnisse für die Assembly geben kann und es theoretisch denkbar ist, auch ein Netzwerklaufwerk als Ausgabeverzeichnis festzulegen.

Der IDE bei der Arbeit zusehen

Im Ausgabefenster kann verfolgt werden, wie ein Projekt kompiliert wird und welche Assembly-Bibliotheken dabei verarbeitet werden. Wenn Sie einen Blick in eines der Verzeichnisse werfen möchten, aus denen diese Assemblies stammen, kopieren Sie den Pfad aus dem Ausgabefenster in das Windows-Eingabefeld, das über *Start|Ausführen* erscheint. Außer einer unscheinbaren .dll-Datei sehen Sie dort aber nichts.

4.7 Arbeiten mit Quelltext

Programmiert wird bei Visual Studio im Code-Editor, der zwar bei allen Programmiersprachen derselbe ist, dessen Fähigkeiten, eingegebene Befehle erkennen und entsprechende Auswahllisten anbieten zu können, aber von der gewählten Programmiersprache abhängen. Zu seinen Komforteinrichtungen gehören unter anderem:

- Das automatische Einfärben der Sprachelemente, je nach Typ. Die Farben können in den Optionen der IDE eingestellt werden.
- Der Umstand, dass nach der Eingabe eines Punktes eine Auswahlliste mit allen Mitgliedern der Klasse oder des Objekts angeboten wird. Dabei wird die Liste in häufig benötigte und alle Elemente unterteilt, und es werden nur jene Namen angezeigt, die mit den bereits eingegebenen Buchstaben übereinstimmen.

- Der Umstand, dass bereits vor der ersten Eingabe eine Auswahlliste mit allen zur Verfügung stehenden Befehlen und Schlüsselwörtern angeboten wird und diese kontextsensitiv ist, also immer nur jene Elemente angeboten werden, die zur aktuellen Situation passen.
- Befehle werden unmittelbar nach der Eingabe ausgewertet und Fehler sofort angezeigt. Das ist möglich, da der Compiler immer aktiv ist und jede Eingabe sofort in IL-Code umsetzt (der aber nur im Arbeitsspeicher gehalten wird).

Vielleicht haben Sie sich über den grünen oder gelben Randstreifen im Code-Editor-Code-Editor gewundert. Mit dem gelben Randstreifen macht die IDE Zeilen kenntlich, die seit dem letzten Abspeichern geändert wurden. Grün bedeutet, dass die Zeile gesichert wurde. Diese praktische Einrichtung ist eventuell nicht sichtbar, denn sie kann in den Optionen der IDE unter Text-Editor|Allgemein *über die Einstellung* Änderungen nachverfolgen *auch abgeschaltet werden.*

Fortgeschrittenere Entwickler profitieren von der Möglichkeit, den Code durch Umstrukturierung optimieren zu können, ohne dass Seiteneffekte auftreten, oder von der Möglichkeit, per Mausklick ein Testprojekt anzulegen, das Testroutinen enthält, die bei Methoden des Programms prüfen, ob sie die richtigen Resultate liefern.

4.7.1 Gliederungen

Am Anfang nimmt man sie vermutlich nicht wahr, doch sie erfüllt eine wichtige Funktion. Die Rede ist von der automatischen Gliederung des Quellcodes, die dazu führt, dass sich z. B. Klassen- oder Methodendefinitionen ein- und ausklappen lassen. Eine Gliederung wird am linken Rand durch ein – oder + angezeigt, das von einem kleinen Kasten umgeben ist. Über den Eintrag *Gliedern* im Kontextmenü kann die Art der Gliederung ausgewählt oder ganz abgeschaltet werden. Praktisch ist die Einstellung *Nur Definitionen anzeigen*, da in diesem Fall nur die Methodenköpfe angezeigt werden (wenn Sie den Mauszeiger auf den Kopf bewegen, wird die komplette Definition angezeigt).

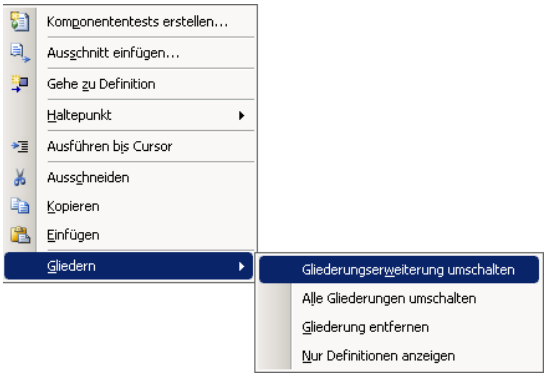


Abbildung 4.27: Das Gliedern führt zu einer besseren Übersicht im Quellcode.

Über die Kommandos #Region *und* #End Region *lassen sich eigene Gliederungsebenen einfügen (siehe Abschnitt 4.7.3).*

4.7.2 Fehler werden sofort erkannt

Da ständig ein Hintergrundcompiler aktiv ist, wird jede Eingabe sofort ausgewertet, und Eingabefehler werden sofort angezeigt. Wie in einem Textverarbeitungsprogramm werden Passagen mit einem Syntaxfehler blau gewellt unterstrichen. Warnungen werden grün untermalt (in den Projekteigenschaften kann eingestellt werden, ob eine Situation im Quellcode zu einer Warnung oder einem Fehler führt oder ignoriert wird), und für Änderungen, die während des Debuggens dazu führen, dass ein Neustart erforderlich ist, ist die Farbe Lila vorgesehen.

Warnungen und Fehler werden in der Fehlerliste angezeigt. Ein Doppelklick auf einen Eintrag führt dazu, dass die entsprechende Stelle im Quelltext angesprungen wird.

Eine Warnung ist lediglich ein Hinweis des Compilers, der Quelltext kann trotzdem kompiliert werden.


TIPP

Wen es stört, dass Warnungen angezeigt werden, kann dies in den Projekteigenschaften im Register Kompilieren *deaktivieren (hier kann auch eingestellt werden, dass eine Warnung wie ein Fehler behandelt werden soll).*

4.7.3 Regionen machen den Quellcode übersichtlich

Vielleicht sind Ihnen am linken Rand die Minus- und Pluszeichen aufgefallen. Damit werden bei Visual Studio die Regionen gekennzeichnet. Eine Region ist ein aus- und einklappbarer Bereich im Code-Editor. Das ist sehr praktisch, da auf diese Weise nicht der komplette Quellcode sichtbar sein muss, sondern nur jener Teil, der aktuell von Interesse ist. Visual Studio fügt z. B. jede Klassendefinition automatisch in eine Region ein. Über das Kommando *#Region* wird ein eigener Gliederungsbereich definiert, der sich auf- und zuklappen lässt:

```
#Region "MontagsCode"

#End Region
```

Auf die Programmausführung haben die Regionen natürlich keinen Einfluss. Das heißt, die Befehle in einer eingeklappten Region werden ebenfalls kompiliert.

4.7.4 Suchen im Quelltext

Wie eine Textverarbeitung besitzt auch Visual Studio eine leistungsfähige Funktion für das Suchen und Ersetzen von Text. Neben einfachen Suchbegriffen können auch reguläre Ausdrücke eingesetzt werden, mit deren Hilfe sich Textmuster finden lassen. Es ist wichtig anzugeben, welche Bereiche des Projekts durchsucht werden sollen. Bei kleinen Projekten spielt dies im Allgemeinen keine Rolle, da man in etwa weiß, wo sich das gesuchte Element aufhält, bei Projekten mit mehreren Hundert Modulen und entsprechend Tausenden von Befehlszeilen wäre es zu ineffektiv, stets das komplette Projekt zu durchsuchen. Die Suche wird über Strg + F gestartet. Es öffnet sich ein zunächst noch kleiner Suchdialog. Hier ist vor allem die Auswahlliste *Suchen in* von Bedeutung, denn standardmäßig beschränkt sich die Suche auf das aktuelle Dokument. Oft möchte man jedoch sämtliche Dateien des Projekts durchforsten und muss daher die Suche ausweiten.

Über den Button Lesezeichen *werden alle Stellen im Quelltext, an denen der Suchbegriff vorkommt, mit einem Lesezeichen versehen, sodass sie sich direkt ansteuern lassen. Das Lesezeichen-Fenster wird über das* Ansicht*-Menü geöffnet.*

Über *Suchoptionen* stehen weitere Varianten zur Auswahl, welche die Suche eingrenzen. Eine wichtige Suchoption kann leicht übersehen werden. Normalerweise wird das erste Vorkommen des Suchbegriffs angezeigt, und weitere Vorkommnisse müssen durch Fortsetzen der Suche aufgespürt werden. Es gibt aber auch einen Modus, in dem alle Ergebnisse in einer Ergebnisliste aufgelistet werden, was oft ein wenig praktischer ist. Dieser Modus wird über die unscheinbare Symbolleiste des Suchen-Dialogs mit dem Button *In Dateien suchen* aktiviert. Damit nicht die unwichtigen (vom Designer generierten) Dateien durchsucht werden, lässt sich die Suche auf Dateitypen beschränken.

4.7.5 Suchen mit regulären Ausdrücken

Bei der Suche mit regulären Ausdrücken werden Treffer nicht über ein konkretes Wort gefunden, sondern über ein Suchmuster, das für eine (große) Auswahl an Sonderzeichen zusammengesetzt wird. Ein [0-9]^4-Muster steht z. B. für eine beliebige Zeichenfolge bestehend aus vier Ziffern. Es versteht sich von alleine, dass solche Suchmuster insbesondere bei großen Projekten deutlich leistungsfähiger sind als konkrete Suchbegriffe, die nur mit einfachen Platzhaltern arbeiten, zumal sich mit regulären Ausdrücken auch das Ersetzen von Textelementen erledigen lässt. Reguläre Ausdrücke sind eine leistungsfähige Suchoption, die zum Glück in der Visual Studio-Hilfe ausführlich beschrieben ist.

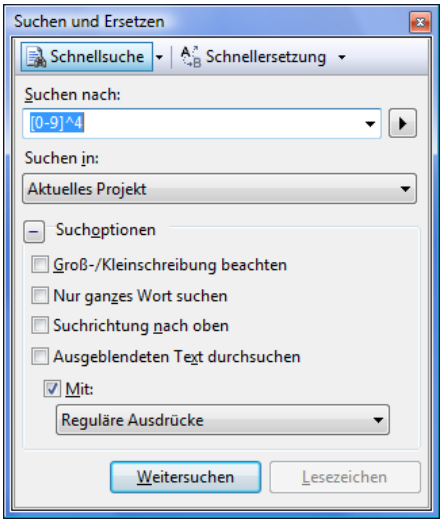


Abbildung 4.28: Die Suche kann auch mithilfe regulärer Ausdrücke durchgeführt werden.

4.7.6 Suchen mit Platzhaltern

Neben regulären Ausdrücken, die am Anfang ein wenig zu »speziell« sein können, steht natürlich auch eine Suche mit Platzhaltern als eine einfachere Alternative zur Verfügung. Wählen Sie dazu aus der Liste *Mit* bei *Suchoptionen* »Platzhalter« aus.

Jetzt ist der Pfeil rechts neben dem Sucheingabefeld aktiv und zeigt die zur Verfügung stehenden Platzhalter an. Ein »Function Pro*« findet z. B. alle Funktionsdefinitionen, die mit »Pro« beginnen.

4.7.7 Die inkrementelle Suche

Bei der inkrementellen Suche, die vor allem bei großen Textdateien sehr nützlich ist, wird das Suchergebnis bereits während der Eingabe angezeigt. Sie wird über Alt + I (oder über *Bearbeiten|Erweitert|Inkrementelle Suche*) gestartet. Der Mauszeiger nimmt das Symbol eines Fernrohrs mit einem Pfeil an, der die Suchrichtung symbolisiert. Die Eingabe eines Zeichens führt dazu, dass das nächste Textelement angesteuert wird, das mit den bereits eingegebenen Zeichen übereinstimmt. Der aktuelle Status der Suche wird in der Statusleiste angezeigt.

4.8 Mehr zu den Vorlagen

Wenn es einen »Schwachpunkt« bei Visual Studio gibt, dann ist es die Vorlagenverwaltung. Vielen Benutzern dürfte am Anfang nicht klar sein, dass sowohl die bei der Projektauswahl als auch beim Hinzufügen eines Elements angezeigten Vorlagen auf dem Inhalt von Zip-Dateien basieren, die sich in bestimmten Verzeichnissen befinden müssen, damit sie in der Auswahl erscheinen. Sind diese Dateien nicht mehr da, werden auch keine Vorlagen angezeigt. Auch wenn dies am Anfang sehr selten vorkommen dürfte, besteht diese Möglichkeit. Spätestens wenn man mal wieder die englische Version von Visual Studio nachinstallieren muss, um eine Vorabversion testen zu können, die nur mit dem englischen Visual Studio läuft, steigt die Wahrscheinlichkeit, dass die Vorlagenverwaltung durcheinandergerät. Und spätestens dann sollte man in etwa wissen, wo man etwas »reparieren« kann.

Eine kleine »Warnung« vorweg. Der Autor empfiehlt (aus eigener Erfahrung), nicht planlos mit den Vorlagen zu experimentieren, da es schnell passieren kann, dass gar keine Vorlagen mehr angezeigt werden. Im ungünstigsten Fall muss Visual Studio durch erneuten Aufruf des Setup-Programms »repariert« werden, was im Allgemeinen aber relativ schnell geht.

4.8.1 Was ist eine Vorlage?

Eine Vorlage ist das, was bei beim Anlegen eines neuen Projekts und beim Hinzufügen eines neuen Elements über *Projekt|Neues Element* angeboten wird. Hinter jeder Vorlage steckt (ab VS 2005) eine reguläre Zip-Datei, in der alle zur Vorlage gehörenden Dateien enthalten sind. Es ist grundsätzlich kein Problem, die Vorlagen zu bearbeiten (etwa damit in einer Klasse bereits *IDisposable* implementiert wird) oder neue Vorlagen anzulegen. Seit VS 2005 kann zudem innerhalb der IDE jedes Projekt in eine Vorlage umgewandelt werden. Es gibt auch »Vorlagen«, nach deren Öffnen ein Assistent startet, diese Sorte wird in diesem Kapitel nicht besprochen.

4.8.2 Wo befinden sich die Vorlagen?

Das ist ein heikler Punkt. Zum einen sind die Vorlagenverzeichnisse in der Visual Studio-Verzeichnishierarchie ein wenig versteckt, zum anderen gibt es mehrere potenzielle Vorlagenverzeichnisse.

Die Visual Basic-Vorlagenverzeichnisse einer Visual Studio 2008-Installation befinden sich im Verzeichnis *%Programfiles%\Microsoft Visual Studio 9.0\Common7\IDE\ProjectTemplates\VisualBasic*. Die Vorlagen eines Anwenders befinden sich dagegen (bei Windows XP) im Anwenderprofil, das heißt konkret im Verzeichnis *%userprofile%\Eigene Dateien\Visual Studio 2008\Templates\ProjectTemplates\VisualBasic*.

Welches Vorlagenverzeichnis herangezogen wird, kann in Visual Studio in den Optionen unter *Projekte und Projektmappen|Allgemein* eingestellt werden.

Abbildung 4.29: Der Speicherort der Projektvorlagen wird in den Optionen der IDE eingestellt.

4.9 Einstellen von Optionen

Visual Studio bietet weit über 100 unterschiedliche Einstellungen, die über *Extras|Optionen* vorgenommen werden. Es soll in diesem Kapitel daher gar nicht der Versuch unternommen werden, eine halbwegs vollständige Übersicht zu präsentieren. Einige Einstellungen sind sehr speziell, andere selbsterklärend. Zwingend eingestellt werden muss nach der Installation von Visual Studio nichts. Tabelle 4.3 stellt ein paar nützliche Einstellungen zusammen.

Tabelle 4.3: Nützliche Einstellungen bei der IDE

Einstellung	Kategorie	Was bewirkt sie?
Erweiterte Build-Konfiguration anzeigen	Projekte und Projektmappen|Allgemein	Der Konfigurationsmanager wird immer angezeigt.
Neue Projekte beim Erstellen speichern	Projekte und Projektmappen|Allgemein	Beim Anlegen eines neuen Projekts kann ein Projektverzeichnis ausgewählt werden. Ansonsten (und das ist die bequemere Variante) werden die Projektdateien zunächst in einem temporären Verzeichnis gespeichert.
Benutzer bei nicht vertrauenswürdigem Speicherort warnen	Projekte und Projektmappen|Allgemein	Diese Option hört sich dramatischer an, als sie ist. Ein nicht vertrauenswürdiger Speicherort ist zum Beispiel ein Netzwerklaufwerk, da Assemblies, die über ein Netzwerklaufwerk starten, automatisch einer niedrigeren Sicherheitsstufe zugeordnet werden.
Aktives Element im Projektmappen-Explorer überwachen	Projekte und Projektmappen|Allgemein	Recht praktisch, da dadurch im Projektmappen-Explorer stets jene Datei selektiert wird, die aktuell bearbeitet wird.
Vor Erstellen	Projekte und Projektmappen|Erstellen und Ausführen	Hier wird eingestellt, ob vor jedem Erstellen automatisch alle Projektdateien gespeichert werden, was im Allgemeinen empfehlenswert ist.

Tabelle 4.3: Nützliche Einstellungen bei der IDE (Forts.)

Einstellung	Kategorie	Was bewirkt sie?
Fensterlayout	Umgebung|Allgemein	Hier kann eingestellt werden, ob alle Fenster auf einem Reiter oder alle unabhängig voneinander, wie bei VB6, angezeigt werden.
Tokenliste	Umgebung|Aufgabenliste|Token	An dieser Stelle lassen sich Kommentartoken definieren, die, wenn sie in einer Kommentarzeile auftauchen, bewirken, dass die Kommentarzeile in der Aufgabenliste erscheint.
Informationen für AutoWiederherstellen speichern	Umgebung|AutoWiederherstellen	Hier wird das Intervall eingestellt, in dem Visual Studio die Dateien sichert.
Ermitteln, ob Datei außerhalb der Umgebung geändert wurde	Umgebung|Dokumente	Im allgemeinen praktisch, da verhindert wird, dass Dateien außerhalb der IDE geändert und diese Änderungen automatisch berücksichtigt werden.
Hilfe anzeigen mit	Umgebung|Hilfe|Allgemein	Hier wird eingestellt, ob die Hilfe in der IDE oder in einem separaten Fenster angezeigt wird.
Schriftarten und Farben	Umgebung|Schriftarten und Farben	Hier werden die Schriftattribute für alle Elemente des Code-EditorCode-Editors eingestellt.
Beim Start	Umgebung|Start	Hier wird eingestellt, was nach dem Start der IDE angezeigt wird.
Tastenkombination für ausgewählten Befehl	Umgebung|Tastatur	Hier kann jedem Menübefehl eine Tastenkombination zugeordnet werden.
Vorschläge für Fehlerkorrekturen aktivieren	Text-Editor|Basic|VB-spezifisch	Ist diese Option gesetzt, erscheint die bereits mit Visual Studio 2005 eingeführte SmartTag-Hilfe bei Codefehlern.

4.10 Die IDE über die Tastatur bedienen

Entwickler sind nicht gerade Mausfetischisten und betrachten sie oft eher als notwendiges Übel. Per Strg + Alt + X öffnet sich die Toolbox oder per F4 das Eigenschaftenfenster in der Regel schneller als über das *Ansicht*-Menü, zumal jeder Menübefehl eine eigene Tastenkombination erhalten kann. Die IDE kann aber nicht nur über Tastenkürzel, sondern im Direktfenster auch direkt (liegt nahe) über die Tastatur gesteuert werden. Dazu muss durch Eingabe von *>Cmd* im Direktfenster das Befehlsfenster geöffnet werden. Hier stehen sämtliche Menübefehle durch Eingabe des Menü- und Kommandonamens zur Verfügung, wobei für die Eingabe Auswahllisten angeboten werden. Besteht das Erstellen eines Projekts aus mehreren Schritten oder soll im Anschluss die IDE beendet werden, geht dies nach ein wenig Übung mit Sicherheit schneller als per Maus (und es beeindruckt mit Sicherheit Verwandte und Kollegen). Zurück in das Direktfenster und damit in den »normalen« Eingabemodus geht es durch Eingabe von »Immed«.

Die zweite Möglichkeit, Visual Studio über die Tastatur zu steuern, besteht darin, es in der Kommandozeile mit einem der zahlreichen Schalter zu starten. Hinter Visual Studio steckt eine unscheinbare Programmdatei mit dem Namen *Devenv.exe*. Doch was soll das bringen? Ganz einfach, manchmal soll ein Projekt nur übersetzt, dazu

aber die IDE nicht gestartet werden. Dies lässt sich mit dem direkten Aufruf von *Devenv.exe* schneller erreichen. Tabelle 4.4 stellt die wichtigsten Kommandozeilenparameter der IDE zusammen.

Tabelle 4.4: Die wichtigsten Kommmandozeilenschalter von Visual Studio

Schalter	Was bewirkt er?
/Build	Erstellt das über /Project angegebene Projekt in der angegebenen Konfiguration (z. B. Debug).
/Command	Startet die IDE und führt das angegebene Kommando aus.
/Log	Legt eine Logdatei fest, in der eventuell auftretende Probleme beim Arbeiten mit der IDE protokolliert werden.
/Deploy	Veröffentlicht das z. B. über /Build angegebene Projekt.

Tabelle 4.5: Die wichtigsten Tastatur-Shortcuts bei Visual Studio (VB6-Tastenbelegung)

Shortcut	Bedeutung
F1	Ruft eine kontextsensitive Hilfe auf.
⇧ + F8	Führt im Haltemodus einen Prozedurschritt aus.
F8	Einzelschritt.
F2	Ruft den Objektbrowser auf.
F3	Wiederholt die zuletzt durchgeführte Suche.
F4	Zeigt das Eigenschaftenfenster für das selektierte Element an.
F5	Kompiliert das als Starteinstellung vereinbarte Projekt und führt es aus.
F6	–
F7	Schaltet auf das Programmcodefenster um.
F9	Setzt einen Haltepunkt oder hebt ihn wieder auf.
Strg + Alt + S	Öffnet den Server-Explorer (nicht bei Express).
Strg + Alt + X	Öffnet die Toolbox.
Strg + F	Ruft den Suchen und Ersetzen-Dialog auf.
Strg + F6	Schaltet auf das nächste Fenster in der Registerkartengruppe um.
Strg + G Strg + Alt + I	Öffnet das Direktfenster.
Strg + ⇧ + N	Legt ein neues Projekt an.
Strg + O	Öffnet ein vorhandenes Projekt.
Strg + R	Öffnet den Projektmappen-Explorer.
Strg + S	Speichert ein Modul.
Strg + ⇧ + S	Speichert das gesamte Projekt.

Tabelle 4.5: Die wichtigsten Tastatur-Shortcuts bei Visual Studio (VB6-Tastenbelegung) (Forts.)

Shortcut	Bedeutung
Strg + Z	Macht die letzte Änderung rückgängig (bezieht sich auch auf Änderungen am Formular).
⇧ + F2	Zeigt die Definition im Quelltext des Bezeichners an, auf dem sich die Textmarke gerade befindet.
⇧ + F7	Schaltet vom Programmcodefenster auf das Formular um.
Strg + F5	Startet ein Projekt im Release-Modus.
Strg + ⇧ + F2	Springt wieder zu jener Stelle zurück, von der aus per ⇧ + F2 gesprungen wurde.

4.11 Tastatur-Shortcuts ändern

Jeder Menübefehl kann einen eigenen Tastatur-Shortcut besitzen, der sich jederzeit ändern lässt. Eine der ersten Einstellungen, die Sie nach der (Neu-)Installation von Visual Studio vornehmen sollten, besteht darin, den Menübefehl *Extras|Optionen* mit einem Tastenkürzel, z. B. F12, zu belegen, sodass Sie nicht jedes Mal das *Extras*-Menü öffnen und *Optionen* selektieren müssen. Gehen Sie dazu wie folgt vor:

1. Wählen Sie den Menübefehl *Extras|Optionen* (noch einmal auf die umständliche Variante).
2. Wählen Sie in der Kategorie *Umgebung* den Untereintrag *Tastatur* (dazu muss *Alle Einstellungen anzeigen* aktiviert sein).
3. Wählen Sie aus der oberen Auswahlliste ein Tastaturschema aus (z. B. »Standard«).
4. Geben Sie in dem Eingabefeld »Befehle mit folgendem Inhalt anzeigen« nun »Extras.Optionen« ein, oder wählen Sie den Befehl aus der Liste darunter aus.
5. Setzen Sie den Cursor unten in das Feld »Tastenkombination drücken«, und betätigen Sie die neue Taste oder eine Tastenkombination. Daraufhin erscheint das Kürzel für die Taste bzw. für die Tastenkombination in diesem Feld.
6. Klicken Sie auf *Zuweisen*, damit die Tastenbelegung aktiv wird, und bestätigen Sie die Änderung mit *OK*.

Wenn Sie nun F12 drücken, sollte der Optionendialog angezeigt werden. Nach diesem Schema erhalten sämtliche Menübefehle eine individuelle Tastenkombination.

4.12 Hilfe in allen Lebenslagen

Visual Studio bietet natürlich eine umfangreiche Hilfe, die neben der IDE auch alle Bereiche der Programmierung mit Visual Basic und dem .NET Framework umfasst. Die Hilfe wird entweder über F1 oder über das *Hilfe*-Menü aufgerufen.

4.12.1 Suchen in der Hilfe

Die Suche wird in Visual Studio über den Menübefehl *Hilfe|Suchen* oder über Strg + Alt + F3 gestartet. Bevor Sie zum ersten Mal einen Suchlauf starten, sollten Sie sich mit dem (neuen) Auswahlmechanismus der zu durchsuchenden Bereiche vertraut machen. In den drei Kategorien *Sprache*, *Technologie* und *Inhaltstyp*

legen Sie fest, was durchsucht werden soll. Wenn Sie ausschließlich in Visual Basic programmieren, sollten Sie keine Zeit damit verschwenden, zum Beispiel C++- oder JScript-Themen zu durchsuchen. Bei den Technologien können Sie speziellere Bereiche wie *Win32 und COM* und *Server-Technologien* abwählen und sich gegebenenfalls nur auf die Themen *.NET Development* und *Visual Studio* beschränken. Und beim Inhaltstyp müssen Sie am Anfang zum Beispiel die *Knowledge Base* (die Supportdatenbank von Microsoft, die auch über *http://support.microsoft.com* online erreichbar ist) nicht unbedingt durchsuchen. Je enger Sie die Auswahl treffen, desto genauer werden die Suchergebnisse. Generell ist die Visual Studio-Dokumentation sehr gut, sodass sich der geringe Aufwand lohnt, die Hilfe so zu präparieren, dass .NET- und Visual Basic-Themen bevorzugt angezeigt werden.

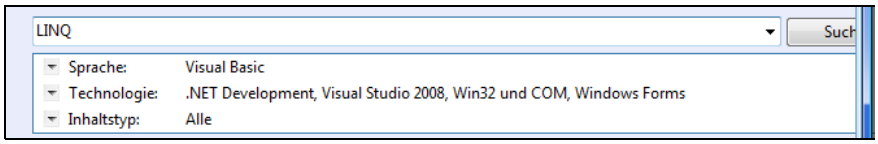


Abbildung 4.30: Bei der Eingabe des Suchbegriffs sollte der zu durchsuchende Bereich eingegrenzt werden.

Achten Sie während der Suche darauf, wie sich die einzelnen Kategorien mit Treffern füllen. Da die Ergebnisse nach Kategorien sortiert angeboten werden, lässt sich unter anderem erkennen, ob ein Treffer zur lokalen Hilfe oder zum MSDN Online-Angebot auf Deutsch oder auf Englisch gehört.

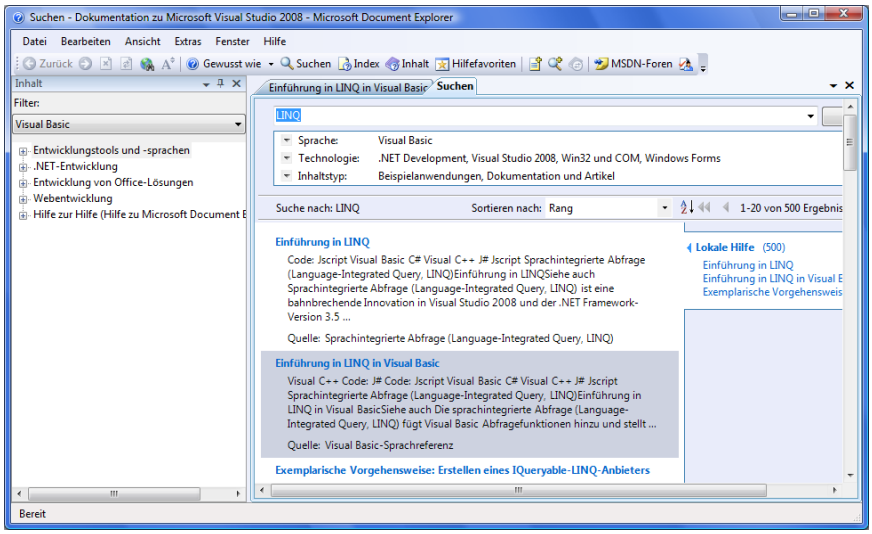


Abbildung 4.31: Das Ergebnis einer Suche ist im Allgemeinen sehr umfassend.

Deaktivieren Sie über Extras|Optionen *die Online-Hilfe. Es gibt normalerweise keinen Grund, dass bei der Suche eine Internetverbindung erforderlich ist. Es sei denn, die MSDN-Dokumentation wurde aus irgendeinem Grund nicht installiert (das ist bei der Visual Studio-Installation ein separater Schritt[4]).*

TIPP

4 Kein Wunder, bei mehreren GByte Umfang.

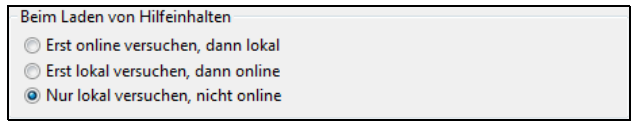


Abbildung 4.32: Die Hilfe muss dieselben Inhalte, die es auf der Festplatte gibt, nicht unbedingt aus dem Internet holen.

4.12.2 Die Indexsuche

Bei der Indexsuche geht es mehr um das Lokalisieren von Begriffen, die aus den Hilfedokumenten extrahiert wurden. Sie ist praktisch, wenn Sie gezielt nach einem Begriff suchen und sich nicht allgemein über ein Thema informieren möchten.

4.12.3 Die dynamische Hilfe

Die dynamische Hilfe wird über den Menübefehl *Hilfe|Dynamische Hilfe* oder einfacher über Strg + Alt + F4 geöffnet (bei Visual Basic 6-Tastaturbelegung, die in den Optionen der IDE jederzeit geändert werden kann). Das Besondere an diesem Hilfefenster ist, dass sich sein Inhalt dynamisch an die Stelle innerhalb der IDE anpasst, an der sich die Einfügemarke gerade befindet bzw. die momentan ausgewählt ist. Ist zum Beispiel etwas im Projektmappen-Explorer selektiert, erscheinen im Hilfefenster Einträge zum Thema Projektmappen-Explorer. Wenn das Codefenster eines Windows-Formulars aktiv oder darin etwas markiert ist, werden entsprechend Informationen zu diesen Themen angezeigt. Ist die Einfügemarke im Programmeditor zum Beispiel innerhalb eines *Imports*-Befehls positioniert, erscheint automatisch ein Eintrag, der Erklärungen zu diesem Befehl abruft. Das funktioniert zum einen, weil ein Hintergrundthread laufend die Begriffe sammelt, die sich in der Umgebung der Einfügemarke befinden. Zum anderen sind die Schlüsselwörter, die in den Programmeditor eingegeben werden können, mit (unsichtbaren) Attributen versehen, mit deren Hilfe eine Zuordnung zu einem Hilfethema hergestellt wird.

4.12.4 Einstellungen bei der Hilfe

Alle Einstellungen werden über den Menübefehl *Extras|Optionen* vorgenommen. Einen zwingenden Grund, Änderungen vorzunehmen, gibt es im Allgemeinen nicht. Zu den wichtigsten Einstellungen gehören:

- ob zu jedem Suchtreffer auch eine Zusammenfassung angezeigt werden soll.
- ob englischsprachige Themen durchsucht werden sollen (das scheint aber keine Auswirkung auf die Resultate zu haben).
- ob bei der Suche die Online-Themen einbezogen werden sollen (wird nicht empfohlen, da es nichts bringt).
- welche »Inhaltsanbieter« zusätzlich durchsucht werden sollen.

4.12.5 Hilfefavoriten und Suche speichern

Damit Sie interessante Seiten schneller wiederfinden, legen Sie für den Eintrag einen Hilfefavoriten an. Alle diese Merker werden in einem eigenen Fenster angezeigt, das sich normalerweise mit dem Index- und Inhaltsfenster kombiniert. Etwas weniger offensichtlich ist der Button *Suche speichern*, der eine Suche mit ihren Suchbegriffen, aber nicht deren Ergebnisse speichert.

4.13 Tipps & Tricks für die Praxis

Visual Studio ist eine komplexe Anwendung mit vielen Möglichkeiten. Das volle Potenzial zu erschließen kann, je nach Häufigkeit und Intensität der Benutzung, Monate, wenn nicht Jahre dauern. In diesem Abschnitt geht es um Bereiche der IDE, die man am Anfang leicht übersehen kann, und um die eine oder andere Arbeitserleichterung.

4.13.1 Codeausschnitte

Codeausschnitte sind eine tolle Sache. Geben Sie einmal Property ein, um eine Eigenschaft zu definieren, und drücken Sie einmal die ↹-Taste. Sie werden feststellen, dass der komplette Rahmen für die Property-Definition eingefügt wird, inkl. eines privaten Feldes.

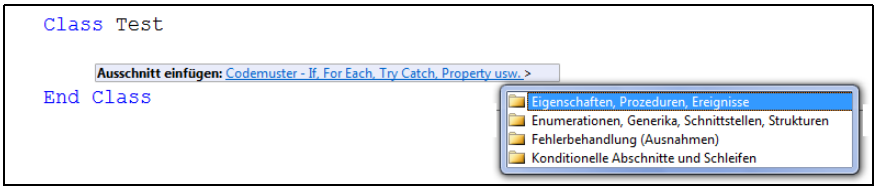


Abbildung 4.33: Codeausschnitte vermeiden das Eintippen stets gleicher Befehlsfolgen.

Die Sammlung der Codeausschnitte ist über den Menübefehl *Extras|Codeausschnitt-Manager* beliebig erweiterbar, sowohl durch heruntergeladene Snippet-Dateien (diese können im Prinzip von jedermann zur Verfügung gestellt werden und müssen lediglich ein bestimmtes Format einhalten) als auch durch eine direkte Suche im Internet im Rahmen der Visual Studio-Hilfe, wobei bei der Suche auch sog. Community-Sites gleichberechtigt miteinbezogen werden.

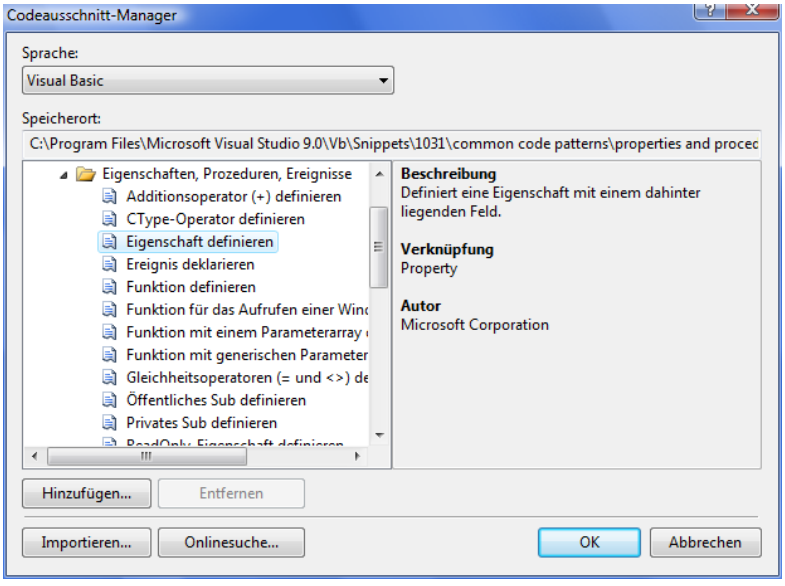


Abbildung 4.34: Die Sammlung der Codeausschnitte kann jederzeit erweitert werden.

4.13.2 Leerzeichen sichtbar machen

Über *Bearbeiten|Erweitert|Leerstelle anzeigen* lässt sich, wie bei einer Textverarbeitung, erreichen, dass die Leerzeichen optisch hervorgehoben werden.

4.13.3 Externe Tools einbinden

Eine der einfachsten Möglichkeiten, die IDE zu erweitern, geht über neue Einträge im *Extras*-Menü. Hinzugefügt werden diese Einträge über *Extras|Externe Tools*. Es erscheint ein Dialogfeld, in dem der Name und vor allem der komplette Pfad des Tools angegeben werden. Auf diese Weise lassen sich z. B. *Ildasm* zum Disassemblieren einer Assembly, Kommandozeilentools aus dem Windows SDK, wie z. B. *MageUI.exe* zum Editieren einer Manifestdatei, aus dem .NET Framework SDK bzw. dem *bin*-Verzeichnis von Visual Studio einbauen. Jetzt kommt ein wichtiger Punkt: Da viele Tools auf das Projektverzeichnis umschalten, dieses aber immer anders lauten kann, stehen entsprechende Variablen in der Auswahlliste *Argumente* bereit. Soll z. B. *Ildasm* mit der Assembly-Datei im Ausgabeverzeichnis als Argument aufgerufen werden, lautet dieses einfach nur *$(TargetPath)*.

Gehen Sie wie folgt vor, um Tools wie *Ildasm.exe* aus dem Framework SDK oder *Reflector.exe* in das *Extras*-Menü einzubinden.

1. Selektieren Sie *Extras|Externe Tools*.
2. Wählen Sie *Hinzufügen*.
3. Geben Sie für Titel z. B. »Ildasm« ein, klicken Sie auf den Button mit den drei Punkten bei *Befehl*, und wählen Sie die .exe-Datei *Ildasm.exe* in ihrem Verzeichnis aus (z. B. unter *%programfiles%\Microsoft SDKs\Windows\v6.0A\bin\ildasm.exe*).
4. Jetzt wird es etwas knifflig: Welche der in der Auswahlliste bei *Argumente* angebotenen Konstanten ist die richtige? Es stellt sich heraus, dass es ganz einfach ist. Anstatt den Pfad über seine Bestandteile *$(BinDir)*, *$(TargetName)* und *$(TargetExt)* zusammenzusetzen, genügt ein *$(TargetPath)*.

Wenn Sie über das *Extras*-Menü den Kommandoprompt anzeigen möchten, fügen Sie ein externes Kommando mit »Cmd.exe« als Befehl ein, und wählen Sie *$(TargetDir)* oder *$(ProjectDir)* als Ausgangsverzeichnis. Ein kleiner Nachteil dieses Prompts ist, dass die *Path*-Umgebungsvariable nicht erweitert wurde, sodass sich die Kommandozeilentools wie z. B. *Ildasm.exe* nicht, ohne den Pfad voranstellen zu müssen, aufrufen lassen. Das Erweitern der *Path*-Variablen und das Setzen von Umgebungsvariablen (also das Einrichten einer »Umgebung«) übernimmt die Stapeldatei *Vsvars32.bat* im Verzeichnis *Common7\Tools*. Wenn Sie der Ehrgeiz packen sollte und Sie einen »perfekten« Kommandozeilenprompt möchten, legen Sie folgende Einstellungen für *Befehl*, *Argumente* und *Ausgangsverzeichnis* fest.

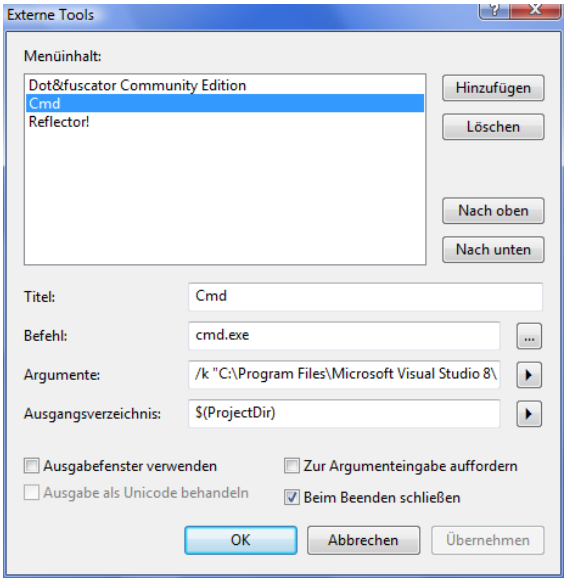


Abbildung 4.35: Cmd.exe wird über das Extras-Menü eingebunden.

Für *Befehl*:

```
%comspec%
```

Für *Argumente*:

```
/k "%programfiles%\Microsoft Visual Studio 9.0\Common7\Tools\vsvars32.bat"
```

Für *Ausgangsverzeichnis* (wird aus der Liste ausgewählt):

```
$(TargetDir)
```

Weitere Einstellungen müssen nicht gemacht werden. Sie erhalten einen Prompt, bei dem alle Pfade gesetzt sind und der auf das Ausgabeverzeichnis umschaltet[5].

TIPP

Einen Visual Studio-Prompt mit gesetzten Pfaden rufen Sie unter der Bezeichnung »Visual Studio 2008-Eingabeaufforderung« über das Startmenü in der Kategorie Alle Programme|Visual Studio 2008|Visual Studio Tools *auf.*

Unter http://www.codeplex.com/VSCmdShell *gibt es einen praktischen Kommandoprompt, der sich nahtlos in die IDE integriert.*

4.14 Zeilennummern einblenden

Zeilennummern, die am linken Rand einblendet werden, erleichtern nicht nur die Orientierung, sondern korrespondieren auch direkt mit den Zeilennummern in einer Fehlermeldung. Zeilennummern werden in den Optionen der IDE in der Kategorie *Text-Editor* entweder für alle Sprachen oder für einzelne Sprachen eingestellt.

5 Auch wenn es anscheinend nicht immer auf Anhieb funktioniert.

4.14.1 Programmbereiche vertikal markieren

Möchten Sie im Code-EditorCode-Editor einen rechteckigen Bereich markieren, halten Sie beim Markieren mit der Maus die Alt-Taste gedrückt.

4.14.2 Auf eine andere Sprache umstellen

Visual Studio »spricht« mehrere Sprachen, neben Deutsch und Englisch z. B. Japanisch. In der Regel wird man zwischen Deutsch und Englisch umschalten wollen. Voraussetzung ist allerdings, dass die englische Version von Visual Studio regulär installiert wurde (die Fähigkeit zur Mehrsprachigkeit besitzt Visual Studio also nicht von Anfang an). Das Umschalten geht sehr einfach in den Optionen der IDE über *Umgebung|Internationale Einstellungen* (diese Einstellung ist aber nur sichtbar, wenn die Option *Alle Einstellungen anzeigen* aktiv ist).

4.14.3 Die Rolle von Vshost.exe

Projekte, die in einer .exe-Datei resultieren, werden durch ein Hilfsprogramm mit dem Namen *Vshost.exe* ausgeführt. Das kann in den Projekteigenschaften (*Debuggen*-Register) über die Einstellung *Visual Studio-Hostprozess aktivieren* deaktiviert werden. *Vshost.exe* soll unter anderem die Ausführung nach dem Drücken von F5 etwas beschleunigen.

4.14.4 Einfügen von XML-Kommentaren

Das Kommentieren des eigenen Quellcodes ist eine wichtige, wenngleich ein wenig lästige Angelegenheit. Ungeschriebenen Konventionen zufolge sollten jede Klasse und jede Methode mit einem Kommentarkopf eingeleitet werden, der eine kurze Zusammenfassung enthält. Damit Programmierer diesen Kopf nicht immer wieder erneut eintippen müssen, gibt es seit Visual Studio 2005 eine praktische Abkürzung. Tippen Sie einfach drei Apostrophzeichen ein, und schon wird ein Kommentarkopf eingefügt. Da die Kommentarzeilen die typische XML-Struktur besitzen, werden sie auch als *XML-Kommentare* bezeichnet. XML wurde nicht zufällig gewählt. Externe Tools, wie zum Beispiel *SandCastle* von Microsoft, können aus diesen Kommentaren eine Dokumentation des Programmcodes zusammenstellen. Andere Tools wie *GhostDoc* können aus der Aufrufsyntax heraus XML-Kommentarköpfe ableiten.

4.14.5 Listings drucken

Es gab Zeiten, da sah man Programmierer stundenlang über grün-weiß-gestreifte Listings brüten. So etwas hat heutzutage eher Seltenheitswert, die allermeiste Zeit werden die Listings am Bildschirm bearbeitet. Dennoch spielt der Ausdruck nach wie vor eine Rolle, und sei es zu Dokumentationszwecken, denn was man schriftlich hat, geht nicht ganz so schnell verloren. Leider sind die Druckmöglichkeiten im Visual Studio auch in der aktuellen Version nicht gerade berauschend, aber ausreichend. Gedruckt wird das aktuelle Modul über *Datei|Drucken*, wobei der übliche Standarddruckerdialog erscheint. Praktisch ist die Option, nur die aktuelle Auswahl drucken zu können, da ein komplettes Modul relativ lang werden kann (auf eine Seite passen etwa 70 Zeilen, und auf einem Farbdrucker wird die Syntaxfärbung berücksichtigt). Über *Datei|Seitenansicht* kann die Abmessung des Druckbereichs eingestellt werden, wobei es im Allgemeinen keinen Grund geben dürfte, daran

etwas zu ändern. Das Drucken von Formularen ist aus der IDE heraus nicht möglich (aber es ist eine Kleinigkeit, ein Formular per Alt + Druck in die Zwischenablage und von dort in Ms Paint&Co zu übertragen, um das Bild von dort zu drucken).

4.15 Makros

Visual Studio bietet nicht nur Makros, sondern auch eine komplette Entwicklungsumgebung, die Visual Studio »täuschend ähnlich« sieht[6]. Über Makros lassen sich Vorgänge innerhalb der IDE automatisieren, die ansonsten über eine Folge von Menübefehlen umgesetzt werden müssten.

Das folgende Makro exportiert die aktuellen Tastatur-Shortcuts in eine Html-Datei, die Zeile für Zeile geschrieben wird.

```
Public Sub ExportVSShortcuts()
  Using Sw As New StreamWriter("VS2008Shortcuts.html")
    Sw.WriteLine("<HTML>")
    Sw.WriteLine("<TITLE>Visual Studio Shortcuts</TITLE>")
    Sw.WriteLine("<BODY>")
    Sw.WriteLine("<H3>Visual Studio Shortcuts</H3>")
    Sw.WriteLine("<TABLE Border='1'>")
    For Each Cmd As EnvDTE.Command In DTE.Commands
        If Cmd.Name <> "" Then
           Dim Bindings As System.Array
           Bindings = CType(Cmd.Bindings, System.Array)
           For i As Integer = 0 To Bindings.Length - 1
               Sw.WriteLine("<TR>")
               Sw.WriteLine("<TD>" & Cmd.Name & "</TD>")
               Sw.WriteLine("<TD>" & Bindings(i) & "</TD>")
               Sw.WriteLine("</TR>")
           Next
        End If
    Next
    Sw.WriteLine("</TABLE>")
    Sw.WriteLine("</BODY>")
    Sw.WriteLine("</HTML>")
    Sw.Flush()
  End Using
End Sub
```

4.16 Zusammenfassung

Visual Studio ist eine Anwendung, die man als Entwickler einfach lieben muss. Sie spricht unerfahrene Entwickler (ein Zustand, der sich im Allgemeinen sehr schnell ändert) durch verschiedene Eingabehilfen genauso an wie erfahrene Profis, die bereits viele Jahre (oder Jahrzehnte) als »Coder« auf dem Buckel haben. Mit dem aktuellen Visual Studio 2008 wurde die IDE durch den optimierten Background-Compiler noch reaktionsfreudiger gemacht. Die meisten Komfortverbesserungen gibt es für ASP.NET-Entwickler, wie z. B. ein komfortabler Umgang mit CSS, Eingabehilfen für JavaScript, die Möglichkeit, clientseitige JavaScripts debuggen zu können, indem im JavaScript-Code einfach ein Haltepunkt gesetzt wird, und natürlich die Integration von AJAX. (Das Thema ASP.NET wird in diesem Buch allerdings nur am Rande behandelt, daher wurden diese Neuerungen in diesem Kapitel auch nicht erwähnt.)

6 Es handelt sich natürlich tatsächlich um ein Visual Studio.