PROGRAMMER'S CHOICE

Michael Kofler, Frank Eller,
Alexander Beyer, Holger Schwichtenberg

Visual Basic 2010

Grundlagen, ADO.NET, Windows Presentation Foundation

ADDISON-WESLEY

3

Die Entwicklungsumgebung Visual Studio

Der Platz reicht nicht aus, um in diesem Kapitel alle Bedienungsdetails der Visual Basic- bzw. Visual Studio-Entwicklungsumgebung zu beschreiben. Und selbst wenn es mehr Platz gäbe, könnte dieser sicherlich besser genutzt werden, als auf vielen Seiten die Menüs, Symbolleisten und Fenster aufzuzählen: Die Entwicklungsumgebung ist zwar überladen, aber ansonsten intuitiv zu bedienen. Sie werden nach wenigen Tagen auch ohne langatmige Erklärungen damit zurechtkommen. Daher beschränkt sich dieses Kapitel auf Neuerungen und Aspekte, die nicht sofort ins Auge springen.

Obwohl sich dieses Kapitel im Einleitungsteil des Buchs befindet, sind einige der hier behandelten Themen schon recht fortgeschritten. Lassen Sie sich davon nicht abschrecken bzw. werfen Sie in ein paar Wochen noch einmal einen Blick in dieses Kapitel – dann verstehen Sie die Grundlagen und erkennen das Potenzial der Funktionen besser.

HINWEIS

Einzelne Teile der Entwicklungsumgebung werden in den Kapiteln behandelt, in die sie thematisch passen (der Objektkatalog also beispielsweise in dem Kapitel, das die Konzepte objektorientierter Programmierung erklärt):

Objektkatalog:	Abschnitt 6.4
Debugging-Elemente:	Abschnitt 8.5
WPF-Designer:	Abschnitt 16.2
Bearbeitung von Setup-Projekten:	Kapitel 22
Visual Data Tools:	Kapitel 28
Windows Forms-Designer:	Windows Forms-E-Book auf der Webseite zum Buch

3.1 Tipps zur Bedienung der Entwicklungsumgebung

Layout der Entwicklungsumgebung

Die Entwicklungsumgebung setzt sich aus Dutzenden von Fenstern zusammen, von denen die meisten an einem der vier Ränder des Hauptfensters angedockt sind. Die eigentlichen Dokumentenfenster (Programmcode, Formulare etc.) werden in dem verbleibenden Innenbereich angezeigt (siehe z.B. Abbildung 1.7 auf Seite 30). Das ermöglicht nur dann ein komfortables Arbeiten, wenn Sie einen ausreichend großen Monitor besitzen.

Wenn Sie mit dem Standardlayout nicht zufrieden sind, können Sie die Position, Größe und das Docking-Verhalten fast aller Fenster selbst konfigurieren. Meine Experimente endeten allerdings immer wieder damit, dass ich zum Standardlayout zurückgekehrt bin (Fenster|Fensterlayout zurücksetzen).

Menüs und Symbolleisten

Wie ein Menü zu bedienen ist, brauche ich an dieser Stelle sicher nicht zu erklären. Ebenso ist eine Beschreibung der zahlreichen Menükommandos hier nicht sinnvoll. Dennoch erscheinen mir einige nicht ganz selbstverständliche Hinweise angebracht:

» **Das Menü ist kontextabhängig!** Viele Menükommandos stehen nur dann zur Verfügung, wenn gerade ein bestimmtes Fenster oder ein bestimmter Fensterbereich der Entwicklungsumgebung aktiv ist. Zum Teil muss in diesem Fenster(bereich) sogar noch ein ganz bestimmtes Element aktiviert sein! Teilweise sind diese kontextabhängigen Menüeinträge ganz praktisch, weil sie vermeiden, dass das Menü noch unübersichtlicher wird, als dies ohnehin schon der Fall ist. Manchmal wirkt die Entwicklungsumgebung aber auch schlicht unlogisch.

» **Auch die Symbolleisten und andere Elemente der Entwicklungsumgebung sind kontextabhängig:** Was für das Menü gilt, gilt auch für andere Elemente der Entwicklungsumgebung. Symbolleisten tauchen auf und verschwinden, je nachdem, was Sie gerade tun bzw. welches Fenster aktiv ist. In der Toolbox sind nur dann Steuerelemente zu finden, wenn gerade ein Formularfenster geöffnet ist. Die Debugging-Fenster stehen erst zur Verfügung, wenn Sie tatsächlich ein Programm ausführen etc.

» **Menüs und Symbolleisten können verändert werden:** Mit Extras|Anpassen können Sie das Hauptmenü und die Symbolleisten verändern. Solange der Dialog aktiv ist, können Sie Menükommandos und Buttons zwischen den Symbolleisten verschieben bzw. kopieren (Strg-Taste). Eine schier endlose Liste aller zur Verfügung stehenden Kommandos finden Sie im Dialogblatt Befehle.

Tastenkürzel

Die Benutzeroberfläche kann durch unzählige Tastenkürzel sehr effizient bedient werden – wenn man die Kürzel einmal erlernt hat. Bevor Sie sich damit auseinandersetzen, sollten Sie sich zuerst für eine der möglichen Standardbelegungen entscheiden. Zur Auswahl stehen unter anderem `Standardeinstellung`, `Emacs`, `Visual Basic 6` und `Visual Studio 6`. Das Schema kann beim ersten Start der Entwicklungsumgebung sowie im Dialogblatt Extras|Optionen|Umgebung|Tastatur ausgewählt werden. Dieses Dialogblatt steht nur dann im Optionen-Dialog zur Auswahl, wenn Sie vorher die Option Alle Einstellungen anzeigen angeklickt haben. Davon gehe ich im weiteren Verlauf dieses Kapitels aus.

Darüber hinaus kann jedem der zahllosen Kommandos der Benutzeroberfläche ein eigenes Tastenkürzel zugewiesen werden. Die so veränderte Tastaturbelegung wird dann in einem eigenen Tastaturschema gespeichert. Alle Einstellungen erfolgen im gerade erwähnten Dialogblatt.

Codeeinrückung

Die Einrückung von Code erfolgt in der Regel automatisch. Die einzige Ausnahme sind mehrzeilige Anweisungen, deren Zeilenenden durch das Zeichen _ gekennzeichnet sind. Die Parameter für die automatische Einrückung können mit Extras|Optionen|Text-editor|Basic|Tabstopps eingestellt werden. Beispielsweise verwendet der gesamte in diesem Buch abgedruckte Code eine Einrücktiefe von nur zwei Zeichen statt der Standardeinstellung von vier Zeichen.

Wenn der Editor bei Änderungen der Codestruktur mit der automatischen Einrückung durcheinanderkommt, markieren Sie einfach die ganze Codepassage und drücken ⇆. Damit wird der gesamte Bereich neu (und korrekt) eingerückt.

#-Kommandos (Direktiven)

#-Kommandos beginnen mit dem Zeichen #. Es handelt sich dabei nicht um Visual Basic-Kommandos, sondern um Kommandos, die vom Compiler bzw. vom Editor ausgewertet werden. Sie dienen beispielsweise dazu, manche Codeteile je nach Abhängigkeit einer Konstante zu berücksichtigen oder Codeteile zu einer Gruppe zusammenzufassen, die vollständig zusammengeklappt werden kann.

#-Kommandos	

`#Region "name"`
`... code`
`#End Region`
	gruppiert Code zu einer Region, die zusammengeklappt werden kann. So definierte Regionen werden beim Laden eines Projekts automatisch zusammengeklappt.

`#Const constname = "xy"`	definiert eine Konstante. Die Konstante ist nur für andere #-Kommandos sichtbar und gilt nur für die jeweilige Codedatei.

`#If ausdruck1 Then`
`  ... code1`
`#ElseIf ausdruck2 Then`
`  ... code2`
`#Else`
`  ... code3`
`#End If`
	definiert eine Verzweigung. Der Compiler berücksichtigt nur den Codeabschnitt, dessen zugeordneter Ausdruck zutrifft. In den Ausdrücken können selbst definierte Konstanten sowie die drei vordefinierten Konstanten `Config`, `Debug` und `Trace` ausgewertet werden. `Debug` und `Trace` (jeweils `True/False`) sowie alle weiteren Konstanten können im Eigenschaftsdialog des Projekts eingestellt werden.

`#ExternalSource(filename, n)`
`  [error]`
`#End External Source`
	stellt eine Verbindung zu einer externen Datei her (z.B. bei **.aspx*-Dateien). Das Kommando ist nur für den internen Einsatz durch die Entwicklungsumgebung gedacht. `n` gibt eine Zeilennummer in der externen Datei an. `error` kann optional angeben, in welcher Zeile der externen Datei ein Fehler aufgetreten ist.

Projekteigenschaften

Die meisten Eigenschaften von Steuerelementen und anderen Elementen eines Projekts stellen Sie im Eigenschaftsfenster ein, das normalerweise rechts unten angezeigt wird. Bei Bedarf blenden Sie das Fenster mit Ansicht|Eigenschaftsfenster ein.

Ausgenommen von dieser Regel sind allerdings die Projekteigenschaften: Für deren Einstellung gibt es eigene Dialogblätter, die Sie am einfachsten durch einen Doppelklick auf My Project im Projektmappen-Explorer öffnen (siehe Abbildung 3.1). Bei Projekten, die Sie ursprünglich mit älteren VB-Versionen erstellt haben, gibt es den Eintrag My Project möglicherweise nicht. In diesem Fall klicken Sie den Projektnamen an und führen das Kontextmenükommando Eigenschaften aus.

Die Dialogblätter betreffen alle möglichen Projektdetails, die im weiteren Verlauf dieses Buchs behandelt werden. An dieser Stelle sind nur die Grundeinstellungen in den Dialogblättern Anwendung und Kompilieren von Interesse.

Anwendungseinstellungen

» Assemblyname: Dieser Punkt gibt an, wie die resultierende Programmdatei heißen soll. An diesen Namen wird noch *.exe* oder *.dll* angehängt. Eine Assembly ist – ein wenig vereinfacht ausgedrückt – die aus einem Projekt resultierende Programm- oder Bibliotheksdatei. Beachten Sie, dass eine Assembly bei komplexen Projekten aber auch aus mehreren Dateien bestehen kann.

Anwendungstyp: Dieser Punkt bestimmt die Art der Assembly. Zur Auswahl stehen unter anderem Konsolenanwendung, WPF-Anwendung, Windows Forms-Anwendung, Klassenbibliothek, Windows-Dienst oder Websteuerelementebibliothek. Vorhandene Projekte können allerdings nicht in einen beliebigen anderen Typ umgewandelt werden.

Je nach Anwendungstyp gilt auch ein besonderes Startverhalten: Konsolenanwendungen werden normalerweise mit der Prozedur `Main` gestartet, Windows- und WPF-Anwendungen durch das Anzeigen des ersten Fensters, Klassenbibliotheken überhaupt nicht. Sie können nur von anderen Projekten verwendet bzw. getestet werden, indem dort eine Objektinstanz einer der Klassen erzeugt wird.

Ein weiterer Unterschied besteht darin, dass Klassenbibliotheken zu **.dll*-Dateien kompiliert werden, die beiden anderen Typen zu **.exe*-Dateien.

» Start-URI/Startformular: Mit diesem Listenfeld geben Sie an, wo die Programmausführung startet. Je nach Projekttyp kann das ein Fenster oder die Prozedur `Main` einer Klasse oder eines Moduls sein.

» Stammnamespace: Dieses Feld bestimmt den Standardnamensraum für das gesamte Projekt. Dieser Name ist nur dann von Bedeutung, wenn es sich bei dem Projekt um eine Klassenbibliothek handelt. In diesem Fall gibt der Namensraum an, wie im Projekt definierte Klassen von außen angesprochen werden. Wenn Sie also in Ihrem Projekt die Klasse `Class1` definiert haben und der Standardnamensraum `hello_world` lautet, dann kann die Klasse von externen Projekten unter dem Namen `hello_world.Class1` angesprochen werden. (Innerhalb des Projekts besteht die Möglichkeit, weitere Unternamensräume mit dem Schlüsselwort `Namespace` zu definieren – siehe Abschnitt 1.3.)

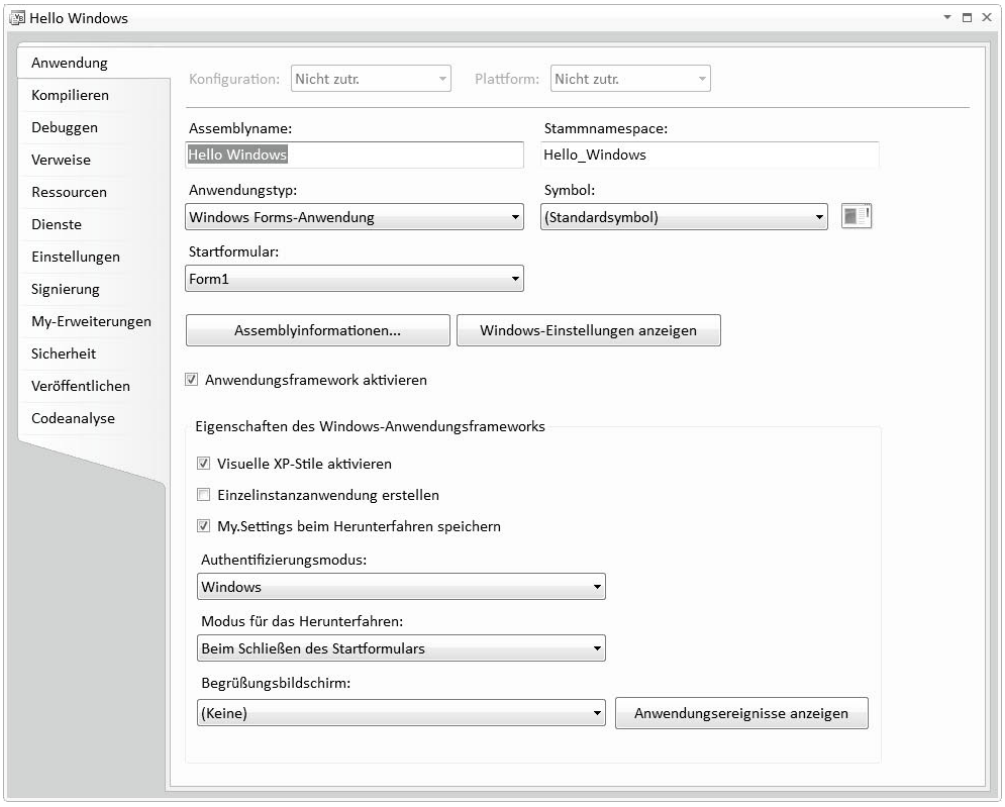


Abbildung 3.1: Einstellung der Projekteigenschaften

» Symbol: Hier können Sie das Icon für das Programm einstellen. Bei Windows-Programmen können Sie alternativ auch die `Icon`-Eigenschaft des Formulars verändern.

» Assemblyinformationen: Dieser Button führt zu einem weiteren Dialog zur Eingabe von Copyright-Informationen, Versionsnummern etc.

» Einstellungen für die Benutzerkontensteuerung anzeigen: Dieser Button öffnet die XML-Datei *app.manifest*, in der diverse Einstellungen durchgeführt werden, die unter anderem die Benutzerkontensteuerung (UAC) von Windows Vista betreffen.

» Eigenschaften des Windows-Anwendungsframeworks: Diese Optionen steuern, wann ein Windows-Programm endet, ob ein Splash-Bildschirm angezeigt werden soll (nur Windows Forms) etc.

Die Standardeinstellungen für Assemblyname und Stammnamespace werden vom Projektnamen übernommen, den Sie beim Start bzw. beim ersten Speichern eines neuen Projekts angeben. Im Namespace-Namen werden gegebenenfalls unzulässige Sonderzeichen durch _ ersetzt.

Achten Sie darauf, dass der Projektname nicht mit dem Namen einer .NET-Klasse oder eines .NET-Namenraums übereinstimmt – sonst gibt es Probleme beim Zugriff auf diese Klasse! Diese Probleme beheben Sie, indem Sie im Eigenschaftsdialog den Namespace-Namen ändern.

Kompilieren-Einstellungen

Im Dialogblatt Kompilieren können Sie diverse Compiler-Optionen und -Warnungen einstellen, wobei nur selten Änderungen an den Standardeinstellungen erforderlich sind. Hintergrundinformationen zu `Option Explicit`, `Strict`, `Compare` und `Infer` finden Sie in den Abschnitten 4.7 und 10.3.

Zu den wirklich interessanten Compiler-Einstellungen führt der Button Erweiterte Kompilierungsoptionen (siehe Abbildung 3.2). Dort können Sie unter anderem diverse Optimierungen und Überlaufkontrollen ein- und ausschalten, die Ziel-CPU und das Zielframework einstellen.

32- versus 64-Bit-Kompilate: Bereits seit Visual Studio 2005 kann der VB-Compiler 64-Bit-Code erzeugen. Es gibt drei Einstellungen für die Ziel-CPU: x86, x64 und AnyCPU. x64-Kompilate können nur auf einem 64-Bit-Betriebssystem ausgeführt werden, x86- und AnyCPU-Kompilate sowohl auf 32- als auch auf 64-Bit-Betriebssystemen. Der Unterschied zwischen diesen beiden Varianten besteht darin, dass x86-Kompilate auf einem 64-Bit-System mit der WOW64-Emulationsschicht ausgeführt werden, AnyCPU-Kompilate dagegen direkt durch die 64-Bit-Version des .NET Frameworks.

Die Standardeinstellung AnyCPU ist somit die optimale Wahl, wenn Ihr Programm sowohl auf 32- als auch auf 64-Bit-Systemen laufen soll. Für x64 sollten Sie sich nur entscheiden, wenn Ihr Programm sehr hohe Speicheranforderungen stellt und die 64-Bit-Adressierung somit von Vorteil ist. Aber auch die Einstellung x86 hat ihre Daseinsberechtigung: Nur in dieser Einstellung funktioniert Edit & Continue, also das Verändern eines laufenden Programms während des Debugging-Prozesses.

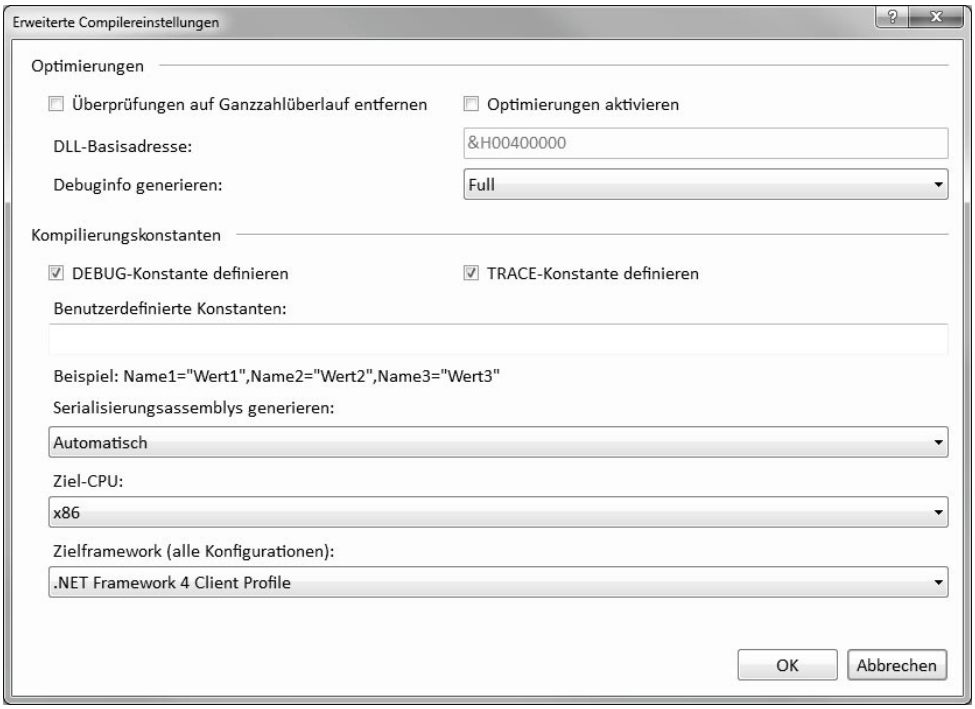


Abbildung 3.2: Erweiterte Compilereinstellungen

.NET Framework: Seit VB2008 können Sie in den Projekteigenschaften die erforderliche .NET-Version einstellen (2.0, 3.0, 3.5 oder 4.0): Je kleiner Sie die Version wählen, desto höher wird die Kompatibilität Ihres Programms zu älteren Betriebssystemen. Damit verlieren Sie aber natürlich auch die neuen Funktionen von VB2010: WPF-Anwendungen setzen zumindest .NET 3.0 voraus, Programme, die LINQ-Funktionen nutzen, .NET 3.5. In der Praxis werden Sie also zumeist die Standardeinstellung .NET 4.0 belassen. Die Auswahl einer kleineren .NET-Version ist aber interessant, wenn Sie ein altes Projekt mit VB2010 weiter warten möchten, ohne dabei Kompatibilitätsprobleme zu verursachen.

3.2 XML-Dokumentation

In C# konnten Sie bereits in der ersten Version direkt in den Code Kommentare im XML-Format einbetten. Derartige Kommentare haben den Vorteil, dass die Dokumentation zum Code nicht von diesem getrennt werden muss und gleichzeitig formale Regeln erfüllt. Die XML-Kommentare werden unter anderem von der IntelliSense-Funktion ausgewertet und im Objektkatalog angezeigt. Sie erleichtern damit insbesondere die Anwendung von Klassenbibliotheken.

Seit VB2005 unterstützt auch Visual Basic diese Form der Dokumentation. Zur Eingabe eines XML-Kommentars geben Sie einfach drei Apostrophe ein (also ›'‹). Die Entwicklungsumgebung macht aus Ihrer Eingabe automatisch eine Schablone, deren Felder Sie anschließend ausfüllen. Üblicherweise

fügen Sie XML-Kommentare unmittelbar oberhalb von Klassen- oder Variablendefinitionen, Prozeduren, Methoden, Eigenschaften etc. ein. Die Schablone berücksichtigt automatisch alle Parameter der Prozedur oder Methode.

Geben Sie beispielsweise oberhalb der Funktion `ItemsText` drei Apostrophe ein:

```
' Beispiel oo-programmierung\linkedlist4
'''
Public Function ItemsText(Optional ByVal max As Integer = 10, _
  Optional ByVal delimitor As String = " ") As String
```

Die Entwicklungsumgebung macht daraus:

```
''' <summary>
'''
''' </summary>
''' <param name="max"></param>
''' <param name="delimitor"></param>
''' <returns></returns>
''' <remarks></remarks>
```

Jetzt fügen Sie den Text für die diversen XML-Elemente hinzu, beispielsweise so:

```
''' <summary>
''' Verbindet mehrere LinkedList-Elemente zu einer Zeichenkette.
''' Die Methode liefert standardmäßig maximal 10 Einträge und
''' trennt die Zeichenketten durch Leerzeichen.
''' </summary>
''' <param name="max">Gibt an, wie viele LinkedList-Elemente
''' maximal berücksichtigt werden sollen.</param>
''' <param name="delimitor">Gibt das gewünschte Trennzeichen
''' zwischen den Zeichenketten an.</param>
''' <returns>Die Methode liefert die zusammengesetzte Zeichenkette zurück.</returns>
''' <remarks></remarks>
```

Wenn Sie die Methode nun im Objektkatalog ansehen, zeigt dieser genaue Informationen über den Zweck der Methode und über seine Parameter an. Außerdem erscheinen die Informationen in der IntelliSense-Hilfe.

Visual Studio macht aus den XML-Kommentaren beim Kompilieren des Projekts eigene XML-Dateien, die im selben Verzeichnis wie die **.exe*- bzw. **.dll*-Datei gespeichert werden. Das Erstellen der XML-Dokumentationsdateien kann im Dialogblatt Kompilieren der Projekteigenschaften ein- bzw. ausgeschaltet werden.

Neben den im obigen Beispiel verwendeten XML-Elementen sieht Visual Studio eine Reihe weiterer Elemente vor, z.B. <example>, <exception> (gibt an, welche `Exceptions` die Methode auslösen kann) und <permission> (gibt an, welche Rechte zur Ausführung benötigt werden. Eine Referenz aller Dokumentationselemente gibt die Online-Hilfe. Drücken Sie einfach innerhalb eines XML-Kommentars F1.

3.3 Code-Snippets (Codeausschnitt-Manager)

Zusammen mit VB werden viele vorgefertigte Codeschablonen mitgeliefert, die in der englischen Visual-Studio-Version als `code snippets` bezeichnet werden. In der deutschen Version ist zwar offiziell von `Codeausschnitten` die Rede, ich werde aber im weiteren Verlauf dieses Kapitels den etwas saloppen Begriff `Codeschnipsel` verwenden.

Codeschnipsel eingeben

Codeschnipsel stellen Lösungsvorschläge für häufig benötigte Aufgaben zur Verfügung. Zur Eingabe geben Sie zuerst ein Fragezeichen ein und drücken dann [Tab]. Es erscheint nun zuerst eine Kategorieliste, die nach Ihrer Auswahl durch eine Detailliste ersetzt wird.

Bei einigen Codeschnipseln sind nach dem Einfügen einige Codepassagen grün hervorgehoben. An diesen Stellen müssen Sie in der Regel eigenen Text einfügen oder einen Namen verändern. Zur Navigation zwischen den grünen Passagen verwenden Sie [Tab].

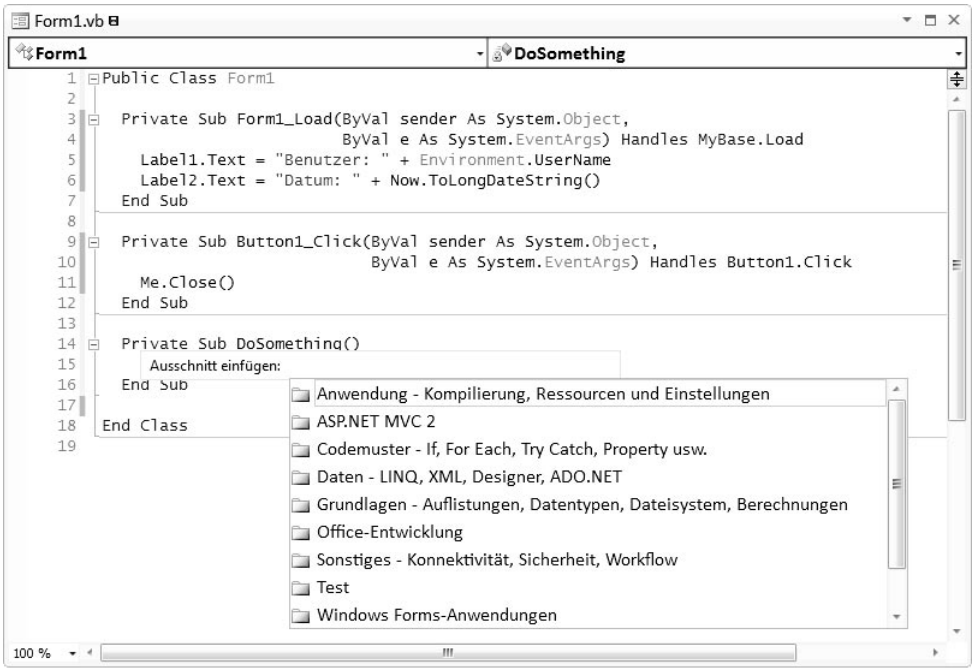


Abbildung 3.3: Codeschnipsel einfügen

Alle Codeschnipsel haben einen Namen, der im Codeausschnitt-Manager im Feld VERKNÜPFUNG angezeigt wird. Wenn Sie den Namen eines Schnipsels kennen, geben Sie diesen im Editor ein (z.B. `form Resize`) und drücken dann [Tab]. Das Schnipsel wird sofort eingefügt, und Sie ersparen sich die Navigation durch die verschiedenen Kategorien. Diese Vorgehensweise funktioniert auch, wenn Sie nur

die Anfangsbuchstaben kennen: In diesem Fall geben Sie die Anfangsbuchstaben und ein Fragezeichen ein und drücken dann [⇆]. Es erscheint nun eine Auswahlliste mit allen passenden Schnipseln.

HINWEIS

Weder die Auswahl der mitgelieferten Codeschnipsel noch deren Qualität haben mich begeistert. Bei den längeren Schnipseln hat mich zudem die oft fehlende Dokumentation gestört. Ich gehöre wohl nicht zur Zielgruppe dieses Werkzeugs ...

Der Codeausschnitt-Manager

Das Menükommando Extras|Codeausschnitt-Manager führt in den Dialog, den Sie in Abbildung 3.4 sehen. Dort wird eine hierarchische Liste aller verfügbaren Codeschnipsel angezeigt. Mit Importieren erweitern Sie die Kollektion um eine **.snippet*-Datei mit einem weiteren Codeschnipsel. Alternativ nehmen Sie mit Hinzufügen ein ganzes Verzeichnis mit **.snippet*-Dateien in Ihre Schnipselkollektion auf. Passende **.snippet*-Dateien können Sie selbst erstellen oder im Internet suchen. Bei den **.snippet*-Dateien handelt es sich um XML-Dateien, deren Format in der Online-Hilfe dokumentiert ist.

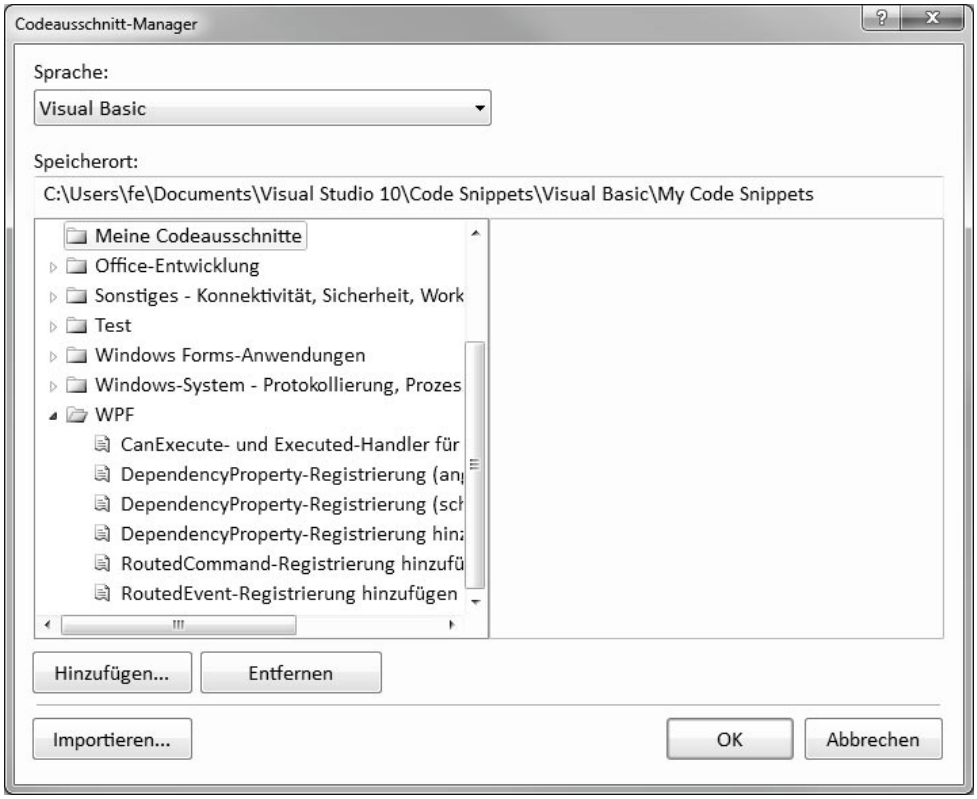
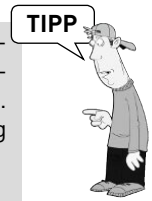


Abbildung 3.4: Der Codeausschnitt-Manager



Unbegreiflicherweise sieht Visual Studio keine einfache Möglichkeit vor, eigene Codeschnipsel zu definieren: Die Online-Hilfe empfiehlt, entsprechende XML-Dateien selbst zu erstellen, aber diese Vorgehensweise ist so umständlich, dass Sie spätestens nach dem dritten Schnipsel aufgeben werden. Besser ist es, aus dem Internet den kostenlosen Code Snippet Editor als Visual Studio-Erweiterung herunterzuladen.

http://msdn2.microsoft.com/en-us/vbasic/bb973770.aspx = http://tinyurl.com/34bh26

3.4 Dotfuscator Community Edition

Visual Basic-Code wird vom Compiler in die Zwischensprache MSIL übersetzt (siehe auch Abschnitt 2.4). Das hat viele Vorteile, aber auch einen großen Nachteil: Es ist denkbar einfach, den Code eines kompilierten .NET-Programms zu rekonstruieren. Der kostenlose .NET Reflector, erhältlich bei Redgate, übersetzt MSIL-Code zurück nach Visual Basic, C# oder Delphi. Dazu laden Sie die **.exe*- oder **.dll*-Datei Ihres Programms bzw. Ihrer Bibliothek, suchen eine Klasse oder Methode aus und starten per Doppelklick den Disassembler. In einem Listenfeld müssen Sie die gewünschte Sprache – also Visual Basic – angeben. Die Rückübersetzung gelingt für beinahe alle Sprachkonstrukte. Einzig einige Visual Basic-Besonderheiten (z.B. statisch deklarierte Variablen) werden so übersetzt, wie sie intern realisiert werden. Auch auf Kommentare müssen Sie natürlich verzichten, diese werden nicht mitkompiliert.

http://www.red-gate.com/products/reflector/

Abbildung 3.5 macht klar, dass Sie in .NET-Programmen nichts verbergen können. (Im Prinzip galt das schon immer, aber so einfach war es noch nie, fremden Code zu lesen.) Ob Verschlüsselungsalgorithmen oder Kopierschutzfunktionen – der Code muss so programmiert sein, dass auch die Kenntnis des Codes den Schutzmechanismus nicht beeinträchtigen kann!

Der .NET Reflector kann auch auf die .NET-Bibliotheken angewendet werden. Wenn Sie wissen wollen, wie die `ConcatArray`-Methode der `String`-Klasse funktioniert, zeigt der .NET Reflector Ihnen den kompletten Code an. Allerdings funktioniert das nicht für Methoden, die extern, also außerhalb der eigentlichen .NET-Bibliothek, realisiert sind. Gerade die .NET-Basisbibliotheken greifen zum Teil intensiv auf diverse Betriebssystemfunktionen zurück, die nicht als .NET-Code vorliegen.

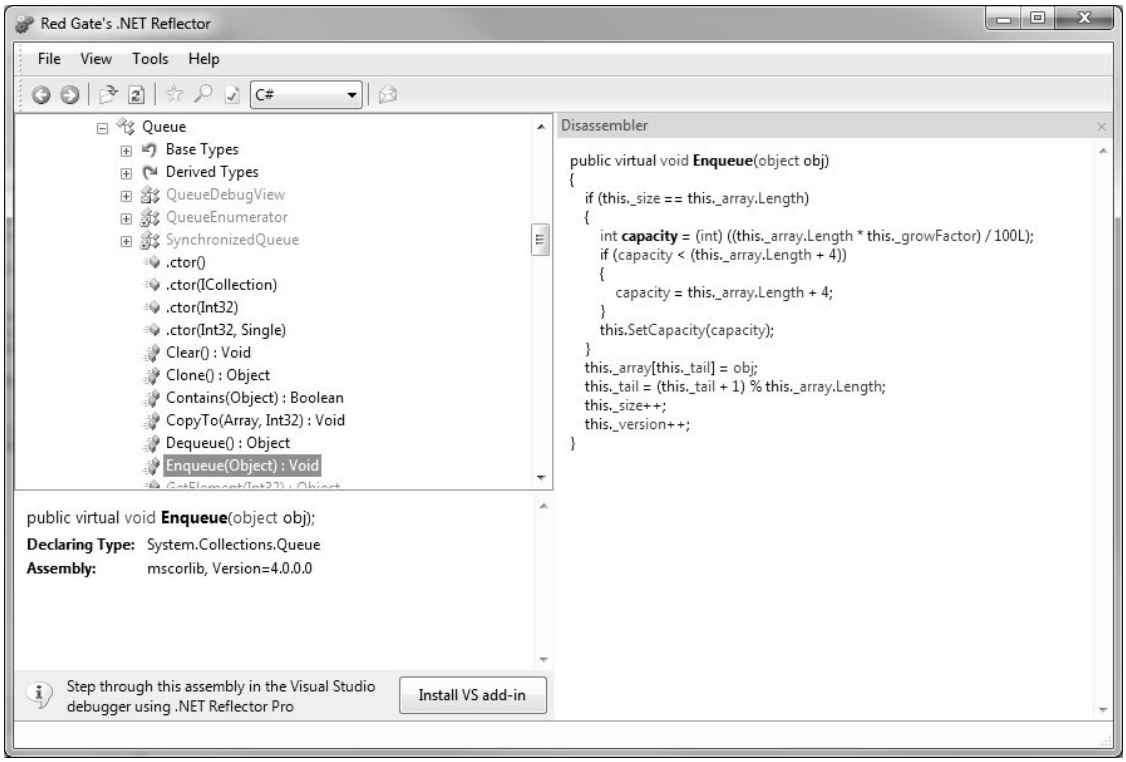


Abbildung 3.5: Der .NET Reflector als Visual Basic-Disassembler

Codeverschleierung durch Obfuscatoren

Mit einem gewissen Zynismus könnte man meinen, Microsoft hätte die ideale Open-Source-Programmiersprache entwickelt. Das Merkmal von Open-Source-Programmen ist ja, dass der Quellcode kostenlos zur Verfügung gestellt werden muss. Bei .NET-Programmen ist das gar nicht mehr notwendig – da wird der Quellcode quasi schon im Programm mitgeliefert! (Ganz stimmt das natürlich auch nicht – selbst mit Werkzeugen wie dem .NET Reflector ist es mit viel Arbeit verbunden, den kompletten Quellcode eines Programms zu rekonstruieren.)

So viel Liebe zur Open-Source-Bewegung darf nicht sein – das hat auch Microsoft erkannt. Außerdem sehen es viele kommerzielle Programmierer nicht gern, wenn jeder den Quellcode ihrer Programme lesen kann: Da wird die Konkurrenz ja geradezu eingeladen, wichtige Algorithmen einfach zu kopieren. Daher wird schon seit Visual Studio 2003 ein kostenloser Obfuscator mitgeliefert.

Der Zweck eines Obfuscators besteht darin, den MSIL-Code so zu verändern, dass die Funktion des Programms zwar erhalten bleibt, der Code aber schwieriger zu lesen ist. Eine einfache Maßnahme besteht etwa darin, alle Namen von Klassen, Methoden, Variablen etc. zu ändern (soweit es sich nicht um öffentliche Elemente einer Bibliothek handelt) – etwa in `a`, `b`, `c` etc. Das erschwert die Orientierung

im Code ganz erheblich. Andere Maßnahmen sind das Entfernen unbenutzten Codes, das Verschlüsseln von Zeichenketten, die Veränderung der Reihenfolge von MSIL-Anweisungen (soweit dadurch die Wirkung des Codes nicht verändert wird) etc.

Eine Google-Suche nach `dotnet obfuscator` liefert zahlreiche Treffer. Es stehen mehrere kommerzielle und sogar einige kostenlose Obfuscatoren zur Auswahl.

3.5 Refactoring

Der Begriff `Refactoring` ist im Zuge der Diskussion um `extreme programming` populär geworden und bezeichnet die Umgestaltung des Codes eines Programms unter Beibehaltung seiner Funktionen. Das Ziel des Refactorings ist es zumeist, einen klareren, besser wartbaren und besser erweiterbaren Code zu bekommen. Ein ganz einfaches Beispiel für Refactoring ist das Umbenennen einer Variablen. Schon aufwendiger ist die Zerlegung langer Codeabschnitte in mehrere Prozeduren, Methoden oder Eigenschaften.

http://en.wikipedia.org/wiki/Refactoring

Grundsätzlich benötigen Sie zum Refactoring keine besonderen Werkzeuge. Allerdings werden Sie rasch feststellen, dass sich beim Refactoring immer wieder dieselben Arbeitsschritte wiederholen: die Neudeklaration von Prozeduren, die Neudefinition lokaler Variablen, der Aufruf der gerade geschaffenen Prozedur etc. Während Microsoft für Visual C# selbst Refactoring-Werkzeuge entwickelt hat, ist dies für Visual Basic nicht der Fall. Für VB2008 existierte ein kostenloses Add-In, das Sie unter der folgenden URL finden:

http://msdn2.microsoft.com/en-us/vbasic/bb693327.aspx = http://tinyurl.com/2wzaev

Für das Visual Studio 2010 existiert von diesem Tool leider nur eine Trial-Version der Pro-Variante. Wir hoffen, dass sich das noch ändert. Über den Erweiterungsmanager finden Sie aber noch weitere interessante Tools.

Die Refactoring-Erweiterung kann nicht unter Visual Basic Express installiert werden. Sie benötigen mindestens Visual Studio Standard. Nach der Installation müssen Sie Visual Studio neu starten.