Thomas Müller

Add-In-Entwicklung für Visual Studio

Thomas Müller

Add-In-Entwicklung für Visual Studio

entwickler.press

Thomas Müller
Add-In-Entwicklung für Visual Studio
ISBN: 978-3-86802-210-0

Ein Imprint der Software & Support Verlag GmbH

Bibliografische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.d-nb.de abrufbar.

Ihr Kontakt zum Verlag und Lektorat:
Software & Support Verlag GmbH
entwickler.press
Geleitsstraße 14
60599 Frankfurt am Main
Tel: +49(0) 69 63 00 89 - 0
Fax: +49(0) 69 63 00 89 - 89
lektorat@entwickler-press.de
http://www.entwickler-press.de

Inhaltsverzeichnis

Für MONI

Von ganzem Herzen Dankeschön!

Vorwort

2003 hatte mich mein sehr geschätzter Kollege Thomas Hemmer (CTO der conplement AG) mit meinem ersten Visual-Studio-Add-in-Projekt beauftragt. Leider konnte ich keine Lektüre, Onlinedokumentation oder Ähnliches finden, das die Add-in-Programmierung, unter Verwendung der Programmiersprache C#, ausführlich und durchgängig erklärt. Es gab bis zu diesem Zeitpunkt zwar schon zwei Bücher zu diesem Thema, doch wurde immer nur Makrocode verwendet, und einzelne Kapitel endeten genau dort, wo ich gern mehr erfahren hätte. Nach drei Jahren, in denen ich viel praktische Erfahrung in der Add-in-Programmierung mit C# gesammelt hatte, fasste ich den Entschluss, dieses Buch zu schreiben. Dieses halten Sie nun in Ihren Händen. Mit all den unbeschriebenen Funktionen, Eigenheiten und Erweiterungen, die bis dahin – nicht nur von mir selbst – in anderen Büchern oder Dokumentationen vermisst wurden. Mit diesem Buch möchte ich Ihnen ein Nachschlagewerk an die Hand geben, das Ihnen einen fundierten Einstieg in die Add-in-Programmierung für Visual Studio geben soll. Aber auch die Programmierer unter Ihnen, die sich schon mit der Add-in-Programmierung für Visual Studio beschäftigt haben, werden sicher viel Wissenswertes in diesem Buch vorfinden.

Ich wünsche Ihnen gutes Gelingen für Ihre eigenen Add-in-Projekte!

Warum sollten Sie dieses Buch lesen?

Vielleicht haben Sie ja schon einmal erste Schritte in der Add-in-Programmierung für Visual Studio gewagt. Dann könnte es Ihnen genauso wie mir ergangen sein und Sie haben sich mit den folgenden Fragen und Hindernissen herumärgern müssen:

- Sie möchten Toolfenster erstellen, die sich genauso wie die Toolfenster von Visual Studio verhalten?
- Sie haben eigene Symbole mit transparenten Bildbereichen, aber in den Toolfenstern von Visual Studio werden diese nicht angezeigt?
- Sie möchten auf Ereignisse in Visual Studio reagieren, wissen aber nicht, wie Sie die entsprechenden Event-Handler abonnieren können?
- Sie haben eigene benannte Befehle erstellt, die dann aber nur einmal angezeigt werden?
- Sie möchten neben benannten Befehlen eigene Befehlsleistensteuerelemente und Symbolleisten erstellen, finden aber nur Beispiele in Makrocode, die für Visual Studio Add-ins nicht anzuwenden sind?
- Sie möchten auf das Code-Modell in C#-Projekten zugreifen, ohne den gesamten Text einer C#-Datei zu parsen?

Auf diese und viele weitere spannende Fragen werden Sie in diesem Buch Antworten finden. Zusätzlich habe ich Ihnen für jedes Thema aus den einzelnen Kapiteln dieses Buchs praxisnahe Beispiele erstellt.

Für wen ist dieses Buch?

Dieses Buch richtet sich sowohl an Einsteiger, die bereits Erfahrung mit der C#-Programmierung haben, als auch an erfahrene Entwickler in der Add-in-Entwicklung für Visual Studio.

Als Leser dieses Buchs sollten Sie die folgenden Eigenschaften mitbringen:

- Gute Grundkenntnisse in der Programmiersprache C#,
- geübten Umgang mit Visual Studio ab der Version 2005,
- Neugier, Geduld und Spaß, den Funktionsumfang von Visual Studio zu erweitern.

Danksagung

So wie viele Autoren es schon vor mir gesagt haben, kann auch ich nur noch einmal den folgenden Satz betonen: Ohne ein starkes Team ist ein gutes Fachbuch nicht möglich!

Darum möchte ich mich mit einem festen Händedruck bei den folgenden Menschen bedanken.

Vielen Dank Erik Franz. Erik hat diesem Buch-Projekt die Initialzündung gegeben und immer an die Fertigstellung geglaubt.

Vielen Dank an Sandra Michel und Frau Möws, die mit großer Unterstützung als direkte Ansprechpartner für mich da waren. Natürlich richtet sich mein herzlichster Dank auch an das gesamte Team des Software & Support Verlages.

Ebenso möchte ich Rudolf Huttary für seine Unterstützung danken. Rudolf hat es mit seiner Erfahrung als hervorragender Fachautor geschafft, diesem Buch den wichtigen letzten Schliff zu geben.

Vielen Dank an meinen Chef Jürgen Lorenz (conplement AG), der mir mit gutem Rat und Verständnis in Zeiten großer Anstrengungen geholfen hat.

Als Letztes möchte ich allen Freunden und meiner Familie Dank für das Verständnis aussprechen. Danke für Eure Unterstützung.

Nürnberg, im Oktober 2008

Thomas Müller

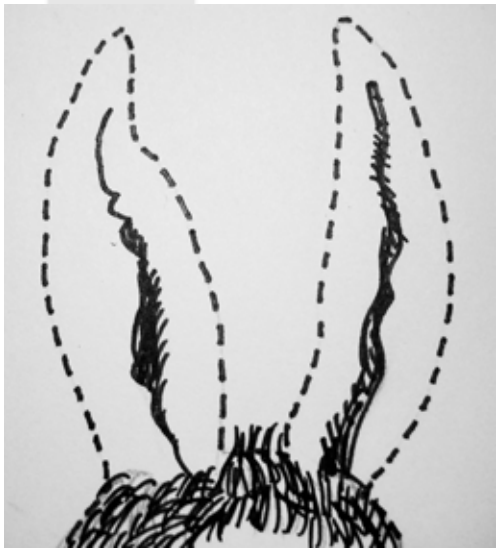
www.tom-mue.de
TOM_MUE@gmx.net

1 Einleitung

Mit dieser Einleitung möchte ich Ihnen den Aufbau dieses Buchs vorstellen und anschließend noch einige Tipps für weiterführende Literatur und die MSDN von Microsoft geben.

Der Aufbau dieses Buchs

Wenn Sie gerade erst mit der Programmierung von Visual Studio Add-ins begonnen haben, dann erhalten Sie sicher den besten Lerneffekt, indem Sie das Buch vom ersten bis zum letzten Kapitel durchlesen. Für diejenigen unter Ihnen, die schon Erfahrung in einem oder mehreren Bereichen der Add-in-Entwicklung haben, kann das Buch ebenso als Nachschlagewerk zu jedem einzelnen Thema verwendet werden. Wie auch immer Ihre Erfahrungen im Bereich der Add-in-Entwicklung sind, Sie können dieses Buch quer oder aber auch vom Anfang bis zum Ende lesen. Ich kann Sie herzlich dazu einladen, an den für Sie spannendsten Stellen Eselsohren zu verwenden.



Eselsohren zum Ausschneiden

Mir persönlich fehlte immer eine Zusammenfassung der Schnittstellen, Typen und Member aus den unterschiedlichen Bereichen der Add-in-Entwicklung. Dafür habe ich Ihnen als Referenz für die Kapitel 4-9 und auch zu weiterführenden Themen das Kapitel 10 erstellt. In Kapitel 10 finden Sie eine Zusammenfassung des Automatisierungsobjektmodells von Visual Studio bis einschließlich Visual Studio 2008. Wenn Sie zum Beispiel wis-

sen möchten, welche Schnittstellen Sie für den Bereich der benannten Befehle, Befehlsleistensteuerelemente und Symbolleisten verwenden können, finden Sie eine entsprechende Zusammenfassung und die dazugehörige Beschreibung im entsprechenden Abschnitt des Kapitels 10. Sehen Sie sich bitte dazu auch die Abbildung 1.1 an.

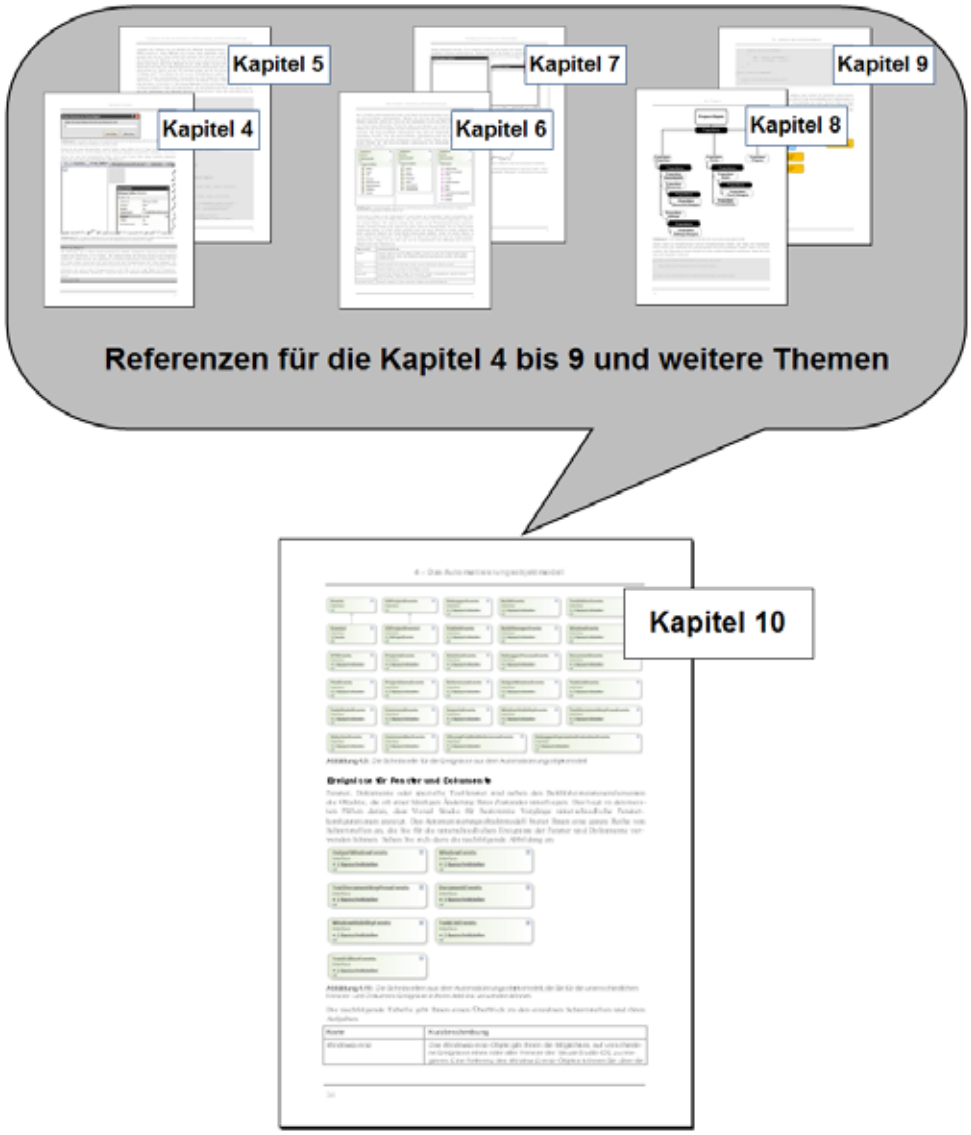


Abbildung 1.1: Das Kapitel 10 dieses Buchs stellt eine Referenz für die Themen aus Kapiteln 4 bis 9 und zu weiterführenden Bereichen des Automatisierungsobjektmodells zur Verfügung.

Ich kann es sehr gut verstehen, wenn Sie sich bei Ihrem ersten Blick in Kapitel 10 gegenüber den Tabellen skeptisch fragen, ob hier vielleicht nur der Text aus der MSDN kopiert wurde. Dem ist natürlich nicht so. Auch in den Tabellen habe ich darauf geachtet, dass die wichtigsten und weiterführenden Informationen enthalten sind. Was nützt es Ihnen, wenn ich in einer Tabelle unterschiedliche Enumerationen aufzähle, aber keinen Hinweis gebe, wo Sie diese in Ihren Add-ins verwenden können. Versuchen Sie also, Ihre Skepsis etwas zu überwinden ☺.

Code-Beispiele zum Buch

Für die Kapitel 4 bis 9 habe ich Ihnen auf dem Web-Blog *http://vsaddinbook.spaces.live.com* die passenden praktischen Beispiele und Informationsdateien zur Verfügung gestellt.

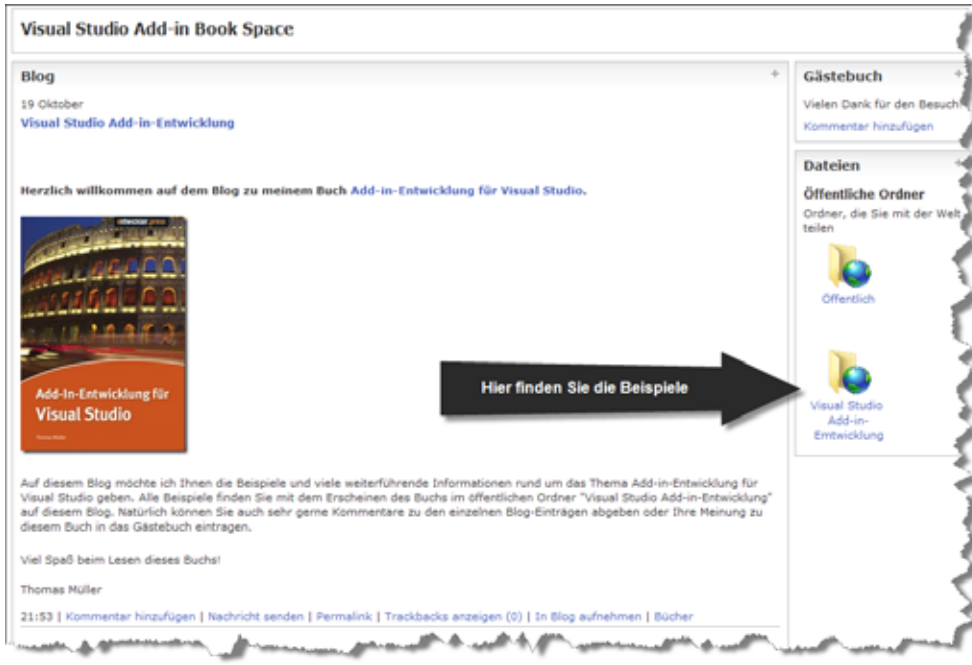


Abbildung 1.2: Auf dem Blog zu diesem Buch finden Sie auch die passenden Add-in-Beispiele.

Wenn Sie die Beispiele von der Webseite auf Ihren Computer geladen haben, dann können Sie die ZIP-Datei mit dem Passwort zU39Jkl& öffnen. In der obersten Verzeichnisebene der Beispiele finden Sie dann eine Liesmich-Textdatei, in der Ihnen der Aufbau der Beispiele kurz erklärt wird. Diese Datei sollten Sie in jedem Fall beachten, da ich in den AddIn-Registrierungsdateien relative Pfadangaben verwendet habe. Die Add-in-Assemblies sollten also passend zu den jeweiligen Pfadangaben in den jeweiligen AddIn-Registrierungsdateien auf Ihrem System abgespeichert sein.

Hilfe zur Selbsthilfe

Wenn Sie – neben dem Blog zu diesem Buch – weitere Informationen oder Hilfe zu weiterführenden Themen benötigen, dann kann ich Ihnen die folgenden Webseiten empfehlen.

Webseite und URL	Kurzbeschreibung
mztools *http://www.mztools.com*	Auf dieser Webseite finden Sie in der Rubrik „For Add-in Developers" sehr viele Beispiele und Bug-Berichte rund um das Thema Add-in-Entwicklung für Visual Studio. Alle Beispiele sind in Visual Basic geschrieben.
VSX Team Blog *http://blogs.msdn.com/vsxteam/*	Blog des Microsoft Visual Studio Extensibility-Teams. Hier finden Sie News, Code-Beispiele und Kontaktmöglichkeiten für Fragen zum Thema Visual Studio Add-in-Entwicklung.
Deutsches Visual Studio Extensibility-Center von Microsoft *http://www.microsoft.com/germany/msdn/vstudio/extend/default.mspx*	Einstieg zu weiterführenden Informationen, aber auch Basisinformationen rund um das Thema Visual Studio Erweiterung und Automatisierung.
Englisches Visual Studio Extensibility-Center von Microsoft *http://msdn.microsoft.com/en-us/vsx/default.aspx*	Einstieg zu weiterführenden Informationen, aber auch Basisinformationen rund um das Thema Visual Studio Erweiterung und Automatisierung. Immer mit aktuelleren Informationen und Beispielen als auf den deutschen Webseiten.

Tabelle 1.1: Webseiten für Hilfe und Informationen rund um das Thema Visual Studio Add-in-Entwicklung

Wenn Sie in der MSDN-Library nach Code-Beispielen für die Entwicklung von Visual Studio Add-ins suchen, dann kann es vorkommen, dass Sie auf den ersten Blick enttäuscht werden. Die Erklärungen und Anwendungs-Beispiele zu den unterschiedlichen Membern und Typen des Visual-Studio-Automatisierungsobjektmodells sind oft nicht sehr aussagekräftig. Verschiedentlich befinden sich aber an anderen Stellen in der MSDN sehr gute Erläuterungen, die nur darauf warten, gefunden zu werden. Gehen Sie bei der Suche nach solchen Beispielen und Erläuterungen für ein besseres Suchergebnis wie folgt vor.

Geben Sie in das Suchfeld der Suchmaschine Ihrer Wahl den entsprechenden Begriff mit dem Zusatz *site:microsoft.com* ein. Wenn Sie also nach einer Beschreibung für die Schnittstelle *DTE2* suchen, dann können Sie die folgende Suchanweisung angeben:

C# DTE2 site:microsoft.com

Dadurch werden nur Ergebnisse auf den Webseiten von Microsoft zu den entsprechenden Angaben gesucht. Somit erhalten Sie also die Suchergebnisse von alle Microsoft-Webseiten, müssen sich aber nicht mit den Suchergebnissen aus dem WWW herumquälen.

Zum Abschluss dieser Einleitung möchte ich Ihnen viel Spaß beim Lesen dieses Buchs wünschen. Ich hoffe, dass Ihnen dieses Buch immer ein guter Ratgeber für Ihre eigenen Add-in-Projekte ist. Wenn Sie Anmerkungen haben, würde ich mich natürlich sehr freuen, wenn Sie mir diese an die Mailadresse *vsaddinbook@live.de* schreiben könnten.

2 Erweiterbarkeit und Automatisierung von Visual Studio

Visual Studio ist eine sehr weit verbreitete und anerkannte Entwicklungsumgebung. Gerade die große Verbreitung bringt verschiedenste Ansprüche an eine Erweiterbarkeit (Extensibility) mit sich. Diesem Anspruch kann Visual Studio mit einer großen Bandbreite an Erweiterungsmöglichkeiten gerecht werden und braucht sich dabei nicht vor anderen Produkten zu verstecken. Die Möglichkeiten, Visual Studio in seinem Funktionsumfang zu erweitern, sind sehr groß und wachsen ständig weiter. In den letzten Jahren wurde bei Microsoft besonders großer Wert darauf gelegt, die Erweiterbarkeit von Produkten mit größerem Funktionsumfang zu vereinfachen. Besonders für Visual Studio 2005 wurden die Möglichkeiten der Erweiterung in vielen Bereichen vereinfacht.

Einige dieser Erweiterungsmöglichkeiten, die es neben der Add-in-Entwicklung gibt, möchte ich Ihnen in den nachfolgenden Abschnitten kurz vorstellen. Auf meiner Webseite zu diesem Buch steht Ihnen zum Download eine HTML-Datei mit dem Namen *Automation AndExtensibility* zur Verfügung. Diese Datei enthält weiterführende Links zu den verschiedenen Themen der Erweiterbarkeit und Automatisierung von Visual Studio.

2.1 Projekt- und Projektelement-Vorlagen exportieren

Oft gibt es Projekte oder Projektelemente, die immer wieder gleich aufgebaut sind und trotzdem von den Standardprojekt- oder Projektelementvorlagen abweichen. Seit Visual Studio 2005 können Sie mit einem dialoggeführten Export-Assistenten eigene Projekt- oder Projektelementvorlagen erstellen. Alles, was Sie zum Erstellen einer Vorlage benötigen, ist ein lauffähiges Projekt in einer der Programmiersprachen C#, Visual Basic, .NET oder J#. Haben Sie ein eigenes Projekt mit allen notwendigen Verweisen, Implementierungen und Projektelementen erstellt, können Sie den Assistenen zum Exportieren von Projektvorlagen in Visual Studio aufrufen. Über das Menü DATEI | VORLAGE EXPORTIEREN starten Sie den Assistenten.

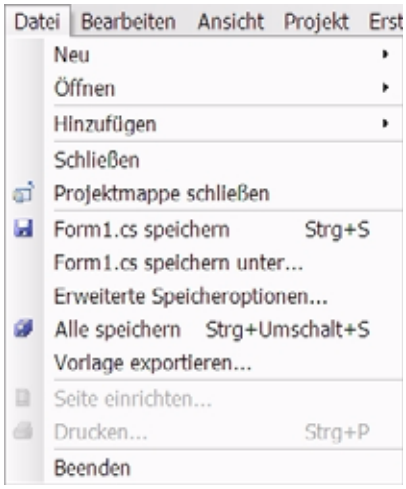


Abbildung 2.1: Das Menü zum Starten des Vorlagen-Assistenten

2.1.1 Projektvorlagen

Im Startdialog des Export-Assistenten können Sie sich entscheiden, ob Sie das gesamte Projekt als Vorlage exportieren wollen oder nur einen bestimmten Bestandteil des Projektes. Stellen Sie sich vor, Sie haben ein C#-Projekt mit einer Anwendungslogik für einen Explorer erstellt. Für diesen Aufbau des Projektes soll nun eine Vorlage erstellt werden. Dieses Projekt soll für zukünftige Projekte als Projektvorlage die Grundstruktur eines Explorers mit dem Projektstart zur Verfügung stellen. Abbildung 2.2 zeigt Ihnen den Startdialog des Vorlagen-Assistenten.

Hier wählen Sie die Option ***Projektvorlage*** (Project Template) aus. Umfasst die Projektmappe mehrere Projekte, wählen Sie im Kombinationsfeld das Projekt aus, das zur Projektvorlage umfunktioniert werden soll. Im zweiten Dialogfenster des Assistenten geben Sie Ihrer Projektvorlage einen Namen, eine Beschreibung und optional auch ein eigenes Projektsymbol (Project Icon). Ich habe hier als Beispiel das Symbol eines Monitors gewählt. Dieses Dialogfenster des Assistenten enthält zwei Kontrollkästchen (Checkboxes). Mit dem ersten Kontrollkästchen können Sie festlegen, ob Ihre neue Projektvorlage direkt in Visual Studio importiert werden soll. Das andere bietet die Möglichkeit, sich die exportierten Dateien der neuen Projektvorlage später im Windows Explorer anzeigen zu lassen.

Abbildung 2.2: Die Optionen zum Auswählen des Vorlagen-Typs

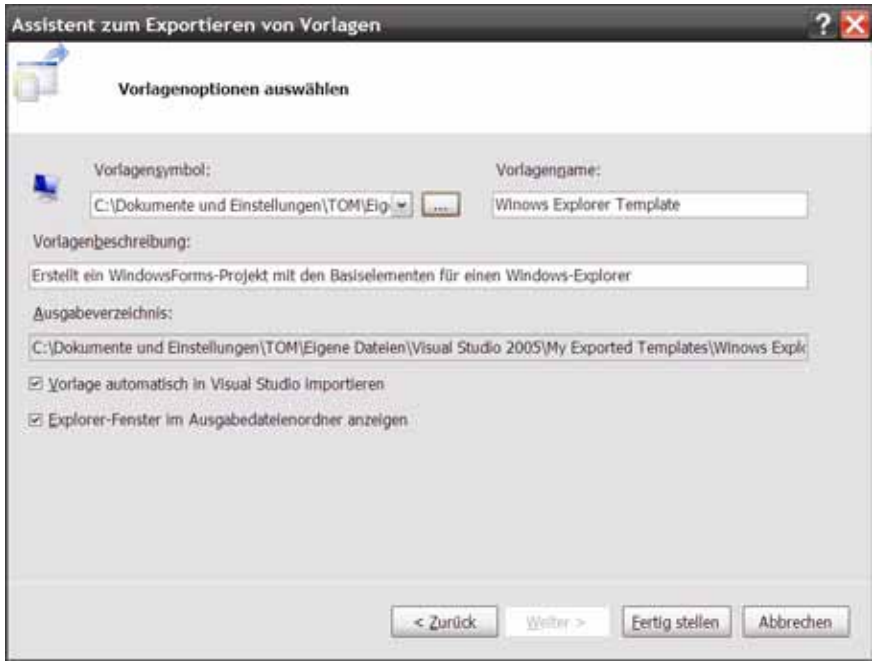


Abbildung 2.3: Beschreibung und Name der Projektvorlage

Wenn Sie alle Einstellungen in den einzelnen Dialogen des Assistenten getroffen haben, brauchen Sie nur noch auf den Button FERTIG STELLEN (Finish) zu klicken. Der Assistent erstellt nun Ihre neue Projektvorlage, eine Datei vom Typ *VSTemplate*. In dieser Datei stehen alle wichtigen Informationen zur generierten Projekt- oder Projektdateivorlage. Ich werde Ihnen im Laufe dieses Kapitels die *VSTemplate*-Dateien noch etwas genauer vorstellen. Die zur Projektvorlage gehörigen Projektbestandteile packt der Assistent in eine ZIP-Datei. Sofern die neue Projektvorlage direkt in Visual Studio importiert wurde, findet sich nun ein Eintrag dafür unter den Standardprojektvorlagen, wie Abbildung 2.4 zeigt.

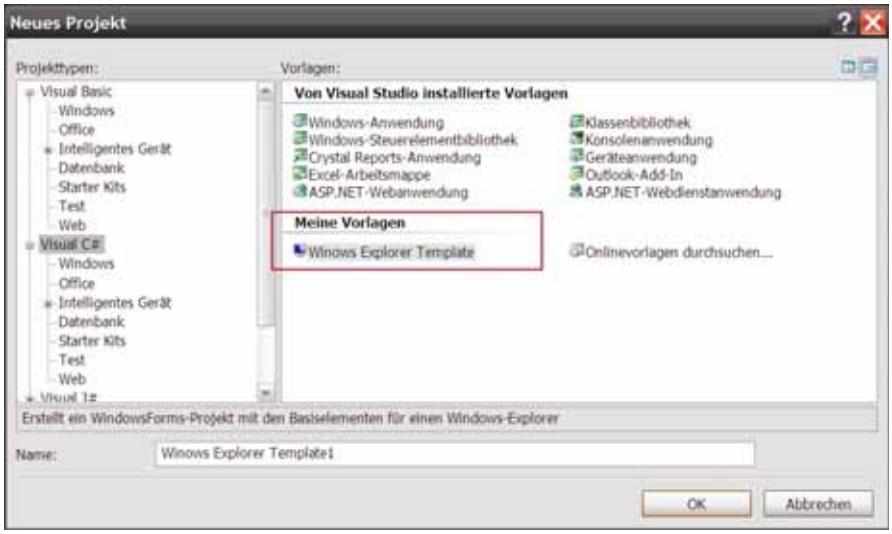


Abbildung 2.4: Die neue Projektvorlage zur Auswahl im Dialog von Visual Studio

2.1.2 Projektelementvorlagen

Wenn Sie nur einen Teil eines Projektes als Projektelementvorlage exportieren wollen, dann können Sie im ersten Dialogfenster des Assistenten die Option für eine Symbolvorlage auswählen, entsprechend Abbildung 2.5. Im Kombinationsfeld des Dialogfensters wählen Sie wieder das Projekt aus, aus dem das Projektelement als Vorlage exportiert werden soll. Anders als beim Export ganzer Projekte zeigt der Assistent vor dem Export zwei weitere Dialoge an.

Abbildung 2.5: Zu exportierende Elemente auswählen

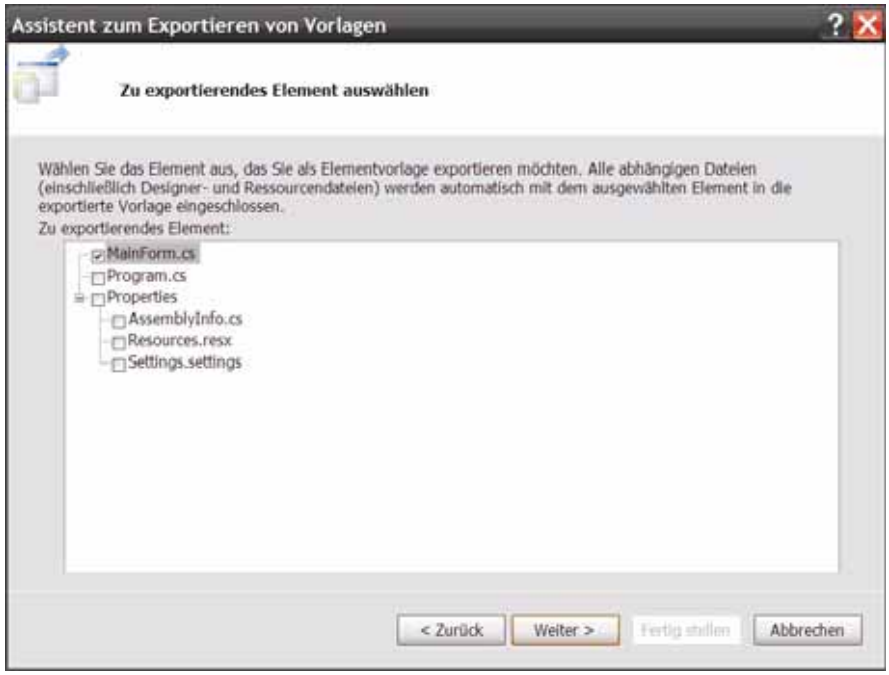


Abbildung 2.6: Elemente, die aus dem Projekt als Vorlage exportiert werden sollen

Im Dialogfenster aus Abbildung 2.6 können Sie die Struktur des Projektes sehen, aus dem ein oder mehrere Projektelemente exportiert werden sollen. Hier können Sie nun das Element auswählen, das als Projektelementvorlage exportiert werden soll. Wenn Sie wie in Abbildung 2.6 ein Windows-Form-Element ausgewählt haben, werden Ihnen die Designer- und die Ressourcen-Dateien der Form nicht mit angezeigt. Wenn Sie sich für das Windows Form zum Exportieren entschieden haben, fügt der Assistent die beiden Dateien automatisch zur Vorlage hinzu. Ist das Element aus dem Projekt zum Exportieren ausgewählt, müssen Sie im nächsten Dialogfenster des Assistenten für Ihre Projektelementvorlage die nötigen Verweise angeben. Stellen Sie sich vor, ein Entwickler fügt Ihre Windows-Form-Vorlage einem Projekt ohne die nötigen Verweise hinzu. Ohne die notwendigen Verweise für Ihre Projektelementvorlage würde der Compiler einen Fehler melden. Abbildung 2.7 zeigt Ihnen den Dialog zur Auswahl der Verweise.

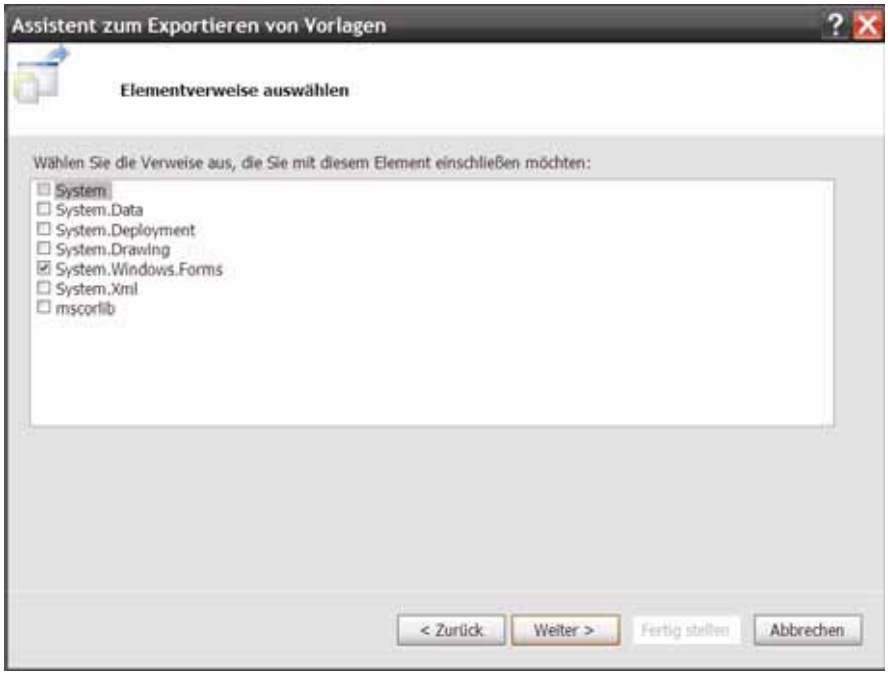


Abbildung 2.7: Auswahl der notwendigen Elementverweise

Nachdem Sie die erforderlichen Verweise angegeben haben, können Sie wie beim Erstellen einer Projektvorlage den Namen, eine Beschreibung und ein optionales Vorlagensymbol angeben. Auch beim Exportieren einer Elementvorlage können Sie sich nach dem Export die *VSTemplate*-Datei und die erstellte ZIP-Datei im Windows-Explorer anzeigen lassen. Ebenso ist der direkte Import der Projektelementvorlage in Visual Studio möglich.

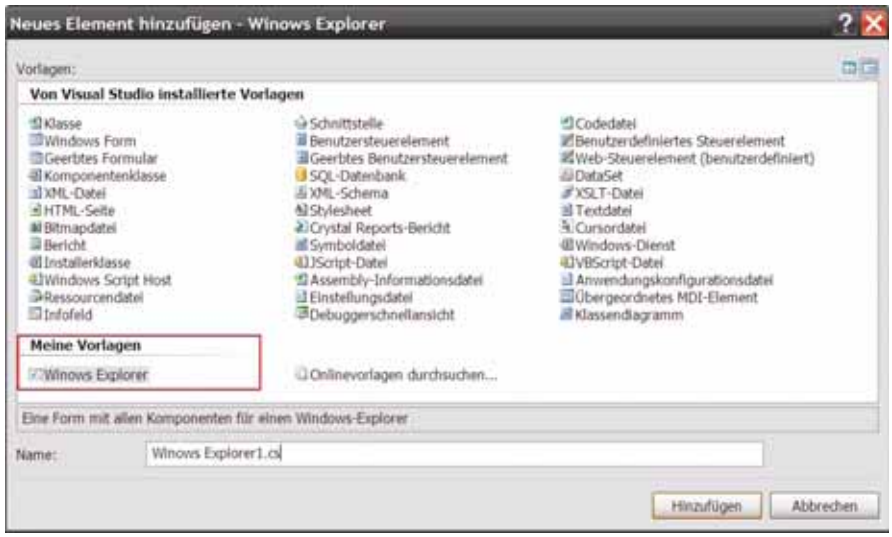


Abbildung 2.8: Die Vorlage für ein neues Projektelement

2.1.3 Dynamischer Export

Mit dem Assistenten zum Exportieren von Projekt- oder Projektelementvorlagen können Sie beliebig viele und beliebig komplexe Vorlagen für .NET-Projekte erstellen. Theoretisch sind Ihnen bei den Vorlagen keine Grenzen gesetzt. Hin und wieder kann es aber vorkommen, dass Sie die Projektvorlagen manuell oder innerhalb eines dynamischen Prozesses erstellen müssen. Egal, ob Sie so eine Projekt- oder Projektelementvorlage erstellen wollen: Damit Visual Studio die Vorlage später verwenden kann, braucht es eine *VSTemplate*-Datei. Diese *VSTemplate*-Datei enthält alle wichtigen Informationen zu einer Projekt- oder Projektelementvorlage. Wie solche *VSTemplate*-Dateien manuell oder dynamisch erstellt werden, finden Sie im MSDN in einer sehr ausführlichen Online-Dokumentation. Einige wichtige Links zu MSDN-Dokumenten finden Sie in dem *AutomationAndExtensibility* HTML-Dokument auf meiner Webseite. Informationen zum Link finden Sie in der Einleitung des Buchs.

2.2 Codeausschnitte

Codeausschnitte (*Code Snippets*) sind wohl die raffiniertesten kleinen Helfer, die Sie zur Automatisierung von immer wiederkehrenden Arbeitsschritten in Visual Studio verwenden können. Code, der immer wieder in derselben Grundform geschrieben wird, kann mithilfe der Codeausschnitte schnell und einfach in Codedateien eingefügt werden. Nehmen wird als Beispiel die Eigenschaften (*Properties*) in C#. Sie können eine Eigenschaft natürlich immer und immer wieder von Hand eingeben. Wenn Sie aber einen Codeausschnitt verwenden, sind es zum Erstellen einer fertigen Eigenschaft nicht mehr als fünf kleine Schritte.

In Abbildung 2.9 sehen Sie eine sehr einfache Klasse mit dem Namen *Person*. Dieser Klasse soll nun eine Eigenschaft mit dem Namen *FirstName* hinzugefügt werden. Die Eigenschaft soll dabei einen *Getter* und einen *Setter* enthalten.

```
1 using System;
2 using System.Collections.Generic;
3 using System.Text;
4
5 class Person
6 {
7
8 }
```

Abbildung 2.9: Die Klasse *Person*, der die Eigenschaft *FirstName* hinzugefügt werden soll

Wenn Sie im Code-Editor von Visual Studio innerhalb der Klasse *Person* die Buchstaben `prop` eintippen, öffnet sich automatisch das Autovervollständigungsfenster des C#-Code-Editors. In diesem kleinen Fenster wird Ihnen dann der Eintrag `prop` angeboten. Die Abbildung 2.10 zeigt Ihnen das Autovervollständigungsfenster. Der Eintrag *prop* steht als Abkürzung für das Schlüsselwort *property*.

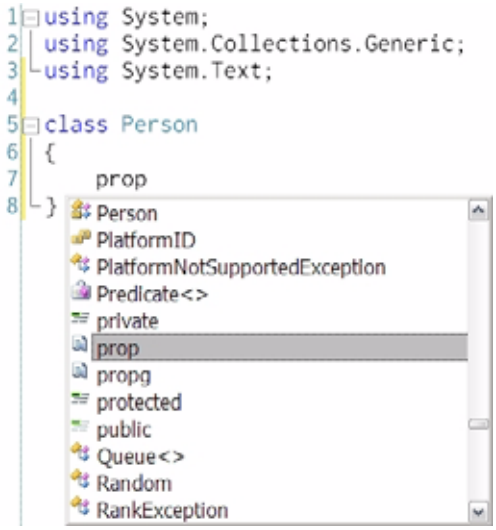


Abbildung 2.10: Der *prop*-Codeausschnitt in der Autovervollständigung

Ist der Eintrag *prop* im Autovervollständigungsfenster markiert, brauchen Sie nur noch zweimal die Taste ↹ drücken. Dadurch fügt Ihnen Visual Studio das Gerüst einer C#-Eigenschaft in den Code der Klasse *Person* ein. In Abbildung 2.11 können Sie das Gerüst der neuen Eigenschaft sehen.

```
using System;
using System.Collections.Generic;
using System.Text;

class Person
{
    private int myVar;

    public int MyProperty
    {
        get { return myVar; }
        set { myVar = value; }
    }

}
```

Abbildung 2.11: Das Grundgerüst einer Eigenschaft, die mit einem Codeausschnitt eingefügt wurde

Wenn Sie die Codeausschnitte das erste Mal verwenden, werden Ihnen sicher die unterschiedlichen Hintergrundfarben an den markierten Stellen der eingefügten Eigenschaft auffallen. Der Code-Editor markiert in einem Codeelement, das mit einem Codeausschnitt eingefügt wurde, alle bearbeitbaren Felder. Sie können mit der ↹-Taste von einem Feld zum nächsten Feld wechseln. Natürlich ist der Rückwärtsgang mit ⇧+↹ auch möglich.

```
using System;
using System.Collections.Generic;
using System.Text;

class Person
{
    private string myVar;

    public string MyProperty
    {
        get { return myVar; }
        set { myVar = value; }
    }

}
```

Abbildung 2.12: Im ersten bearbeitbaren Feld wurde der Typ geändert.

In Abbildung 2.12 wurde der Typ der Eigenschaft bereits geändert. Danach springt man mit der Taste ↹ zum Namen der Variablen für die Eigenschaft. Der Name der Variablen wird dann automatisch im Getter und Setter der Eigenschaft angepasst. Sehen Sie sich dazu Abbildung 2.13 an.

```
using System;
using System.Collections.Generic;
using System.Text;

class Person
{
    private string firstName;

    public string FirstName
    {
        get { return firstName; }
        set { firstName = value; }
    }

}
```

Abbildung 2.13: Der letzte Schritt für das Erstellen der Eigenschaft *FirstName*

Wenn Sie den Namen der Variablen angepasst haben, kommen Sie mit einem weiteren Klick auf die ⇥-Taste auf den Namen der neuen Eigenschaft. Nachdem Sie den Namen der Eigenschaft angepasst haben, reicht ein Klick auf die ↵-Taste, und die Eigenschaft ist fertig.

```
using System;
using System.Collections.Generic;
using System.Text;

class Person
{
    private string firstName;

    public string FirstName
    {
        get { return firstName; }
        set { firstName = value; }
    }

}
```

Abbildung 2.14: Die fertige Eigenschaft *FirstName*

Visual Studio bietet eine große Zahl von Codeausschnitten an. Diese Codeausschnitte können Sie für die unterschiedlichsten Bereiche in der Entwicklung nutzen. Als Programmiersprachen werden die .NET-Sprachen C#, Visual Basic und J# unterstützt. Bitte beachten Sie, dass die Sprache J# ab Visual Studio 2008 nicht mehr unterstützt wird. Natürlich können Sie Standard-Codeausschnitte nach Ihren ganz speziellen Wünschen erweitern und anpassen. Die Codeausschnitt-Datei für eine C#-Eigenschaft befindet sich in einem Unterverzeichnis von Visual Studio. Bei Visual Studio 2008 ist das der Pfad: *… \ Programme \ Microsoft Visual Studio 9.0 \ VC# \ Snippets \ 1033 \ Visual C#*. Die Codeausschnitt-Dateien haben die Dateierweiterung *.snippet* und enthalten XML, wie Abbildung 2.15 zeigt.

```
<?xml version="1.0" encoding="utf-8" ?>
<CodeSnippets  xmlns="http://schemas.microsoft.com/VisualStudio/2005/CodeSnippet">
        <CodeSnippet Format="1.0.0">
                <Header>
                        <Title>prop</Title>
                        <Shortcut>prop</Shortcut>
                        <Description>Code snippet for an automatically implemented property</Description>
                        <Author>Microsoft Corporation</Author>
                        <SnippetTypes>
                                <SnippetType>Expansion</SnippetType>
                        </SnippetTypes>
                </Header>
                <Snippet>
                        <Declarations>
                                <Literal>
                                        <ID>type</ID>
                                        <ToolTip>Property type</ToolTip>
                                        <Default>int</Default>
                                </Literal>
                                <Literal>
                                        <ID>property</ID>
                                        <ToolTip>Property name</ToolTip>
                                        <Default>MyProperty</Default>
                                </Literal>
                        </Declarations>
                        <Code Language="csharp"><![CDATA[public $type$ $property$ { get; set; }$end$]]>
                        </Code>
                </Snippet>
        </CodeSnippet>
</CodeSnippets>
```

Abbildung 2.15: Die XML-Struktur des Eigenschaften-Codeausschnitts

Natürlich können Sie auch benutzerdefinierte Codeausschnitt-Dateien erstellen. Dazu möchte ich Sie wieder auf die Datei *AutomationAndExtensibility* auf meiner Webseite aufmerksam machen. Hier finden Sie Links auf Inhalte der MSDN-Dokumentation, in denen sehr detailliert erklärt wird, wie Sie eigene Codeausschnitte erstellen.

2.3 Makros für Visual Studio

Makros sind Automatisierungsskripte, die Ihnen die Möglichkeit geben, die Funktionalitäten der Visual-Studio-IDE zu nutzen, sie zu erweitern und zu automatisieren. Visual Studio benutzt Visual Basic als Makrosprache, wodurch Sie die gesamte Funktionalität des .NET Framework zur Verfügung haben. Zum Erstellen von Makros können Sie entweder das Makromodul in Visual Studio verwenden oder Sie verwenden die Makro-IDE zum manuellen Erstellen von Makros. In beiden Fällen haben Sie die Möglichkeit, häufig verwendete Arbeitsschritte in Visual Studio als Makro abzuspeichern und bei Bedarf über die Visual Studio DIE aufzurufen. Wenn Sie Makros mit dem eingebauten Makromodul von Visual Studio erstellen, wird das Makro standardmäßig als Methode *MyMacros.RecordingModule.TemporaryMacro* gespeichert. Makros, die über das Makromodul erstellt wurden, sind nur für eine kurze Verwendungsdauer gedacht. Wenn sie das Makromodul erneut aufrufen, wird der Code des zuvor aufgenommen Makros überschrieben. Wollen Sie also ein Makro dauerhaft abspeichern, müssen Sie über EXTRAS | MAKROS | TEMPORARYMACRO SPEICHERN das aufgezeichnete Makro abspeichern. Der dabei erscheinende Dialog gibt Ihnen Gelegenheit, einen Namen für das neue Makro anzugeben. Die Datei, in der das Makro landet, wird standardmäßig von Visual Studio in dem Verzeichnis *Eigene Dateien \ Visual Studio 200x \ Projects \ VSMacros80* abgespeichert. Das x steht hier für eine beliebige Visual-Studio-Version: also 2005 oder 2008. Die Makrodatei hat die Dateierweiterung *.vsmacros*.

Das Makromodul können Sie in Visual Studio mit der Tastenkombination Strg + ⇧ + R starten. Als Beispiel für das Makromodul können wir wieder die Klasse *Person* verwenden. In den meisten Fällen besteht eine Klasse nicht nur aus Eigenschaften und Variablen, sondern enthält auch Methoden, Ereignis-Handler und vieles mehr. Für eine bessere Übersicht im Code-Editor ist es sehr komfortabel, wenn man diese Elemente einer Klasse mit einzelnen Regionen zusammenfassen kann. Abbildung 2.16 zeigt die Klasse Person mit der Implementierung von Regionen. Regionen sind eine Funktion des Code-Editors, die mit Visual Studio .NET 2003 eingeführt wurde. Mit Regionen können Sie ganze Codebereiche im Code-Editor zusammenfassen und wie einen Knoten in einer Baumansicht zusammenklappen.

```
5  class Person
6  {
7      #region fields
8      #endregion
9      #region properties
10         #endregion
11     #region private methods
12         #endregion
13     #region public methods
14         #endregion
15     #region event handler
16         #endregion
17 }
```

Abbildung 2.16: Die Klasse *Person* mit den einzelnen Regionen für Methoden, Eigenschaften etc.

2.3.1 Makros aufzeichnen

Damit Sie nicht für jede neue Klasse von vorn mit der Implementierung der einzelnen Regionen beginnen müssen, können Sie den Vorgang mit dem Makromodul automatisieren:

1. Setzen Sie den Cursor in eine leere Klasse.
2. Rufen Sie das Makromodul mit der Tastenkombination Strg + ⇧ + R auf.
3. Schreiben Sie jetzt alle Regionen in die Klasse, die Sie als Standardregionen für andere Klassen verwenden möchten.
4. Wenn Sie alle Regionen eingefügt haben, beenden Sie das Makromodul mit der Tastenkombination Strg + ⇧ + R.
5. Löschen Sie die Regionen aus der Klasse und testen Sie das aufgenommene Makro mit der Tastenkombination Strg + ⇧ + P.

Abbildung 2.17: Das gestartete Makromodul

Wie Sie bereits erfahren haben, existiert das Makro jetzt als temporäres Makro, das beim nächsten Start des Makromoduls wieder überschrieben wird. Rufen Sie darum jetzt den Befehl EXTRAS | MAKROS | TEMPORARYMACRO SPEICHERN zum Speichern des temporären Makros auf.

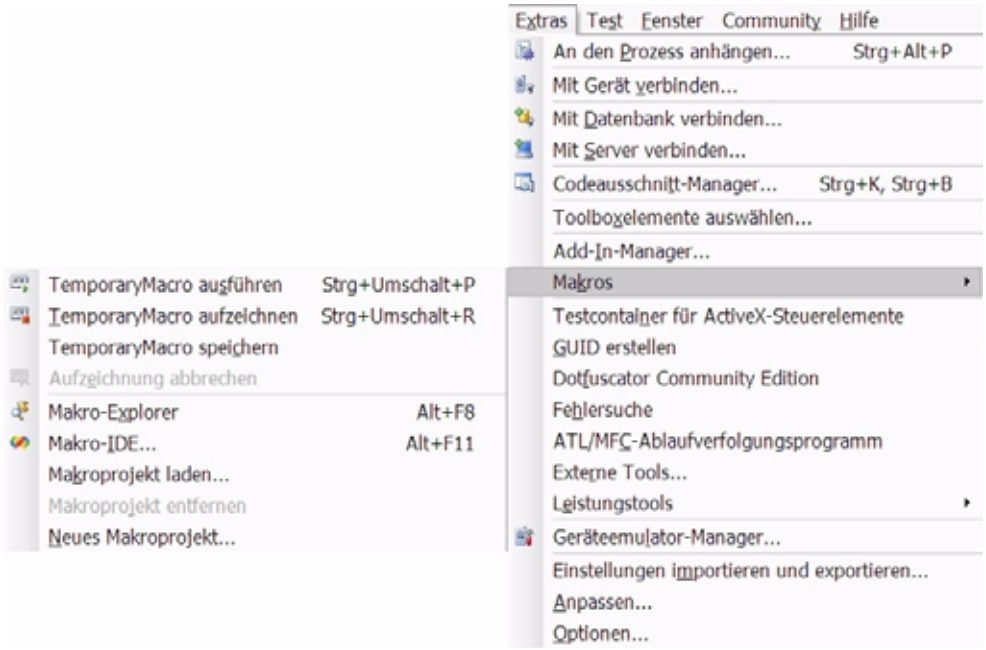


Abbildung 2.18: Der Befehl zum Abspeichern temporärer Makros

Wenn Sie den Befehl zum Speichern aufrufen, öffnet sich automatisch der Makro-Explorer in Visual Studio. In diesem Makro-Explorer können Sie dann das Makro zum automatisierten Einfügen von Regionen, das Sie eben erstellt haben, umbenennen. Damit wird es dann auch dauerhaft abgespeichert. Für dieses Makrobeispiel habe ich den Namen *InsertRegions* verwendet.

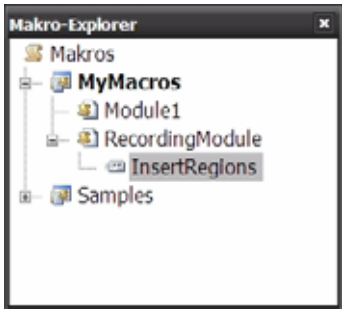


Abbildung 2.19: Der Makro-Explorer mit dem neuen Makro *InsertRegions*

Um dieses Makro auf andere Klassen anzuwenden, können Sie es nun im Makro-Explorer mit einem Doppelklick starten. Das ist natürlich nicht sehr komfortabel. Etwas mehr Komfort bedeutet die Möglichkeit, jedes Makro, das Sie in Visual Studio mit dem Makromodul oder über die Makro-IDE erzeugt haben, an eine Tastenkombination zu binden. So können Sie Ihr Makro bequem mit Ihrer eigenen Tastenkombination aus Visual Studio heraus starten. Öffnen Sie dazu über EXTRAS | OPTIONEN den Optionsdialog von Visual Studio. Aktivieren Sie in der Baumansicht unter UMGEBUNG | TASTATUR den Dialog für die Tastaturschemas und Tastaturkombinationen (Abbildung 2.20).

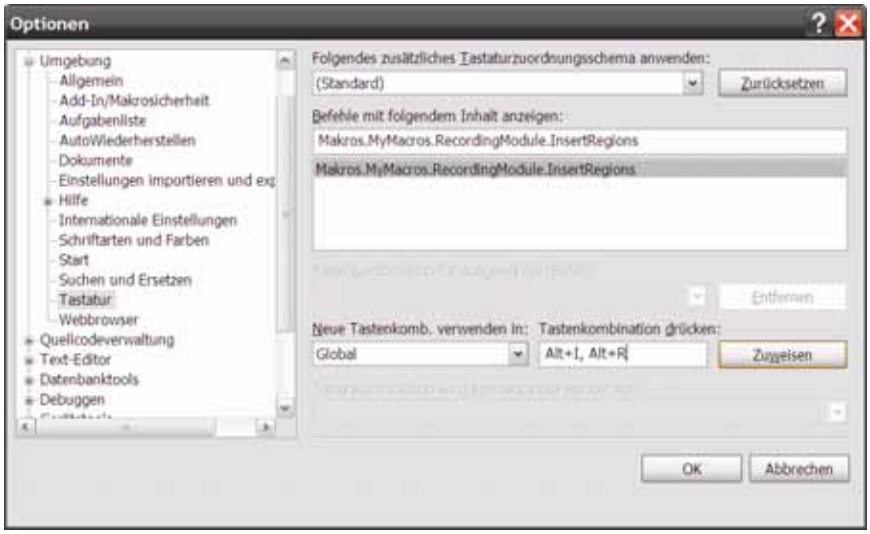


Abbildung 2.20: Der Optionsdialog für Tastaturschemas und Tastaturkombinationen

Geben Sie nun in das Eingabefeld *Befehle mit folgendem Inhalt anzeigen* den Namen Ihres Makros ein. Beachten Sie dabei, dass Sie für ein Makro immer den vollständigen Namen eingeben müssen. Setzen Sie nun den Fokus auf das Feld *Tastenkombination drücken* und drücken Sie die Tastenkombination Alt+I+R. Wenn Sie nun den Button *Zuweisen* klicken, dann ist Ihr Makro mit der von Ihnen gewählten Tastenkombination verbunden. Mit dem Kombinationsfeld *Neue Tastenkomb. verwenden in* könnten Sie außerdem das Makro bestimmten Werkzeugen von Visual Studio zuordnen. Mit der Auswahl *Global* gilt das Makro für jeden Bereich in Visual Studio. Sinnvoller ist hier die Auswahl *Text-Editor*, da das Einfügen von Regionen im *Windwos-Forms-Designer* von Visual Studio keinen Sinn ergeben würde. Sehen Sie sich dazu auch Abbildung 2.21 an.

Das Makromodul ist sicher eine sehr komfortable Möglichkeit, einfache Aufgaben und Vorgänge in Visual Studio zu automatisieren, lässt aber nur einen begrenzten Funktionsumfang zu. So können Sie zum Beispiel über das Makromodul nicht festlegen, dass die Regionen nur dann in den Code eingefügt werden sollen, wenn die .NET-Sprache C# verwendet wird.

Abbildung 2.21: Das Makro mit zugewiesener Tastenkombination und mit Begrenzung der Gültigkeit auf den Bereich des Text-Editors

Wenn Sie das Makro in seiner Funktion erweitern möchten, können Sie es in der Makro-IDE bearbeiten. Die Makro-IDE können Sie in Visual Studio mit EXTRAS | MAKROS | MAKRO-DIE… (Alt + F11) öffnen. Abbildung 2.22 zeigt Ihnen die Makro-IDE mit dem *Projektexplorer* auf der linken Seite.

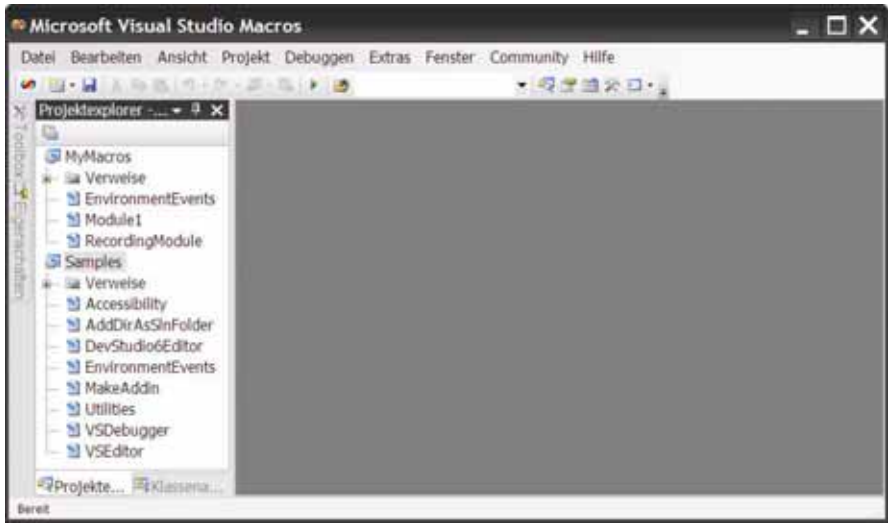


Abbildung 2.22: Die Makro-IDE

Im *Projektexplorer* finden Sie die Projektdatei *RecordingModule* und darin das Makro *InsertRegions*. Mit einem Doppelklick auf die Datei *RecordingModule* öffnet sich der Makroeditor. Listing 2.1 zeigt den Code des *InsertRegions*-Makros.

```
Public Module RecordingModule
    Sub InsertRegions()
        DTE.ActiveDocument.Selection.Text = "#region fields"
        DTE.ActiveDocument.Selection.NewLine()
        DTE.ActiveDocument.Selection.Text = "#endregion"
        DTE.ActiveDocument.Selection.NewLine()
        DTE.ActiveDocument.Selection.Text = "#region properties"
        DTE.ActiveDocument.Selection.NewLine()
        DTE.ActiveDocument.Selection.Text = "#endregion"
        DTE.ActiveDocument.Selection.NewLine()
        DTE.ActiveDocument.Selection.Text = "#region private methods"
        DTE.ActiveDocument.Selection.NewLine()
        DTE.ActiveDocument.Selection.Text = "#endregion"
        DTE.ActiveDocument.Selection.NewLine()
        DTE.ActiveDocument.Selection.Text = "#region public methods"
        DTE.ActiveDocument.Selection.NewLine()
        DTE.ActiveDocument.Selection.Text = "#endregion"
        DTE.ActiveDocument.Selection.NewLine()
        DTE.ActiveDocument.Selection.Text = "#region event handler"
        DTE.ActiveDocument.Selection.NewLine()
        DTE.ActiveDocument.Selection.Text = "#endregion"
    End Sub
End Module
```

Listing 2.1: Der Code des Makros *InsertRegion*

2.3.2 Ausführung von Makros

Alle Aktionen in diesem Makro werden über das so genannte *DTE*-Objekt gestartet. Es stellt die Schnittstelle zum Automatisierungsobjektmodell von Visual Studio dar und ermöglicht den Zugriff auf weite Teile von Visual Studio. Mit einer einfachen *If*-Anweisung können Sie den Code des Makros so erweitern, dass die Regionen nur noch dann eingefügt werden, wenn die .NET-Sprache C# verwendet wird. Sehen Sie sich dazu Listing 2.2 an.

```
Public Module RecordingModule
    Sub InsertRegions()
        Dim language As String
```

Listing 2.2: Das Makro wurde um eine *If*-Anweisung erweitert und tritt nur noch in Aktion, wenn es sich aktuell um die .NET-Programmiersprache C# handelt.