Produktivität erhöhen durch gute Vorlagen

Visual Studio .NET bietet ein Erweiterungssystem, das dem Entwickler die Anpassung der IDE nach eigenen Vorstellungen erlaubt. So können zum Beispiel individuelle Vorlagen für Projekte und Projektelemente angelegt werden. Im Folgenden werden die verschiedenen Anpassungs- und Erweiterungsmöglichkeiten beschrieben und einige Beispiele für benutzerdefinierte Vorlagen vorgestellt.

Beim Anlegen von Projekten oder Komponenten treten häufig die gleichen Arbeitsschritte auf. So müssen zum Beispiel beim Anlegen eines Dialogs stets die Eigenschaften *FormBorderStyle*, *MinimizeBox* und *MaximizeBox* verändert sowie entsprechende Schaltflächen für Bestätigung und Abbruch eingefügt werden. Eventuell sieht der Firmenstandard eine bestimmte Schrift vor, die ebenfalls zu Beginn geändert werden muss. Diese und andere Arbeiten können durch die Definition eigener Vorlagen eingespart werden.

Die Verwendung von Vorlagen ist vor allem in Projektteams sinnvoll. So können sehr leicht Codierungs-, Gestaltungs- und Dokumentationsregeln hinterlegt werden, die beim Einfügen von Projektelementen automatisch angewendet werden.

Visual Studio .NET bietet hierfür vielfältige Möglichkeiten, die von angepassten Code-Vorlagen bis hin zu eigenen Projekt-Assistenten reichen. Nachfolgend sind einige Beispiele für die Anpassung von Vorlagen aufgeführt:

- Vorgabe eines Standard-Namensraumes für Klassen,
- Vorbelegung von Klassen mit Ableitungen, Schnittstellen oder Attributen,
- Einfügen von speziellen Dokumentationsvorlagen,
- Einfügen von Code-Schachtelungen (Code Regions),
- Voreinstellen von Assembly-Attributen, wie *AssemblyCompany* oder *AssemblyCopyright*,
- Zuweisen eines Standard-CSS-Stylesheets für ASPX-Dateien.

Doch bevor es um das Anlegen eigener Vorlagen geht, wird zunächst das Vorlagenmodell von VS.NET kurz vorgestellt.

SUMMARY

Auf einen Blick
Der Artikel zeigt, wie wiederkehrende Arbeiten bei der Neuanlage von Projekten und Projektelementen durch Visual-Studio-.NET-Vorlagen automatisiert werden können.

Eingesetzte Anwendungen
Visual Studio .NET

Autor
Jörg Neumann ist Programmierer bei der Firma KEEP IT SIMPLE GmbH in Hamburg. Bei Fragen und Anregungen erreichen Sie ihn unter Joerg.Neumann@KEEPITSIMPLE.de.

Die Vorlagenverwaltung

Bei der Verwaltung von Vorlagen unterscheidet VS.NET zwischen sprachspezifischen und sprachneutralen Vorlagen. Sprachneutrale Vorlagen werden unter

```
...\Microsoft Visual Studio .NET\Common7\IDE
```

abgelegt. Dies sind zum Beispiel:

- Add-In-Projekte (*Extensibility Projects*),
- Makro-Projekte (*MacroProjects*),
- eine leere Projektmappe (*SolutionTemplates*),
- einzelne Dateien für Grafiken oder Skripte (*NewFileItems*).

Sogar die Startseite von VS.NET ist über eine Vorlage im Verzeichnis *...\Common7\IDE HTML* anpassbar.

Bei sprachspezifischen Vorlagen legt VS.NET die Vorlagen im Root-Verzeichnis der jeweiligen Programmiersprache ab. Dies ist:

- in Visual Basic .NET: *...\Microsoft Visual Studio .NET\Vb7*,
- in Visual C#: *...\Microsoft Visual Studio .NET\VC#*,
- in Visual C++: *...\Microsoft Visual Studio .NET\Vc7*.

Unterhalb des sprachspezifischen Verzeichnisses sind verschiedene Vorlagenverzeichnisse hinterlegt. Diese enthalten Definitionen für das Anlegen von neuen Projekten, Projektelementen und Assistenten.

Tabelle 1 enthält die wichtigsten Verzeichnisse für jede Sprache mit einer kurzen Beschreibung.

Vorlagen-Definitionsdateien

In den Verzeichnissen für Projekte und Projektelemente befinden sich Definitionsdateien mit der Endung *vsdir*. Sie enthalten neben dem Namen und der Beschreibung des zu erstellenden Elements auch eine Verknüpfung zu dem Assistenten, der für das Erzeugen zuständig ist.

Jede Zeile in der Datei steht dabei für eine Vorlage. Die Felder einer Definitionszeile werden durch ein Pipe-Zeichen (|) getrennt. Optionale Felder, die nicht ausgefüllt werden, können mit *0* oder Leerzeichen belegt werden.

Die Definition einer C#-Windows-Anwendung sieht zum Beispiel so aus (aus Gründen der Übersichtlichkeit sind die Felder untereinander dargestellt):

Tabelle 1 Die wichtigsten Vorlagenverzeichnisse von Visual Studio .NET.

C#-Verzeichnis	VB.NET-Verzeichnis	Beschreibung
CSharpProjectItems	VBProjectItems	Definitionen für Projektelemente
CSharpProjects	VBProjects	Definitionen für Projekttypen
VC#Wizards	VBWizards	Vorlagen und Skripte, die von den Assistenten verwendet werden.

```
CSharpEXE.vsz|
{FAE04EC1-301F-11d3-BF4B-00C04F79EFBC}|
#2318|
10|
#2319|
{FAE04EC1-301F-11d3-BF4B-00C04F79EFBC}|
4554|
|
WindowsApplication
```

Das erste Feld enthält den Namen der Assistenten-Definitionsdatei. Sie bestimmt, welcher Assistent für das Erstellen des Elements zuständig ist und welche Parameter übergeben werden müssen. Falls die Datei nicht im selben Verzeichnis liegt, ist zusätzlich eine Pfadangabe nötig, wobei relative Pfade unterstützt werden.

Außerdem sind der Name und ein Beschreibungstext der Vorlage notiert. Diese Informationen können entweder in Textform oder als Resource Identifier hinterlegt werden. In letzterem Fall muss jedoch zusätzlich die Klassen-ID der Komponente angegeben werden, welche die entsprechenden Ressourcen beinhaltet (wie im oberen Beispiel zu sehen). Der Verweis auf das Vorlagensymbol, das später im Dialog angezeigt werden soll, ist ebenfalls als Resource Identifier anzugeben. Für diesen Identifier muss eine zusätzliche Klassen-ID einer dll- oder exe-Datei angegeben werden.

In Tabelle 2 sind alle Felder mit einer kurzen Beschreibung aufgeführt. Eine ausführliche Dokumentation des Dateiformats ist unter [1] zu finden.

Assistenten-Definitionsdateien

Wie bereits erwähnt, sind alle Vorlagen mit Assistenten verknüpft. Jeder Assistent verfügt über eine eigene Definitionsdatei, die mit der Endung *vsz* versehen ist und alle Informationen beinhaltet, die zum Erzeugen des entsprechenden Elements nötig sind. Die Definitionsdatei des Assistenten für ein C#-Klassenbibliothek-Projekt (*CSharpDLLWiz.vsz*) sieht zum Beispiel so aus:

```
VSWIZARD 7.0
Wizard=VsWizard.VsWizardEngine
Param="WIZARD_NAME = CSharpDLLWiz"
Param="WIZARD_UI = FALSE"
Param="PROJECT_TYPE = CSPROJ"
```

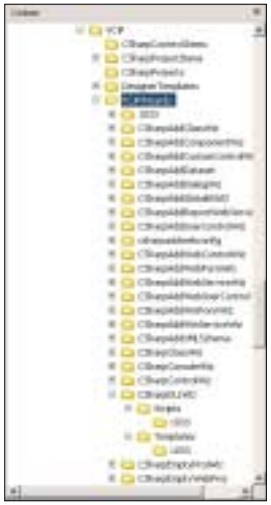


Abbildung 1 | Für jeden Assistenten existiert ein Verzeichnis, in dem die Definitions- und Vorlagendateien für ein Element hinterlegt sind.

Tabelle 2 Formatbeschreibung einer vsdir-Datei.

Feld	Beschreibung
Relativer Pfadname	Pflichtfeld. Es enthält den Pfad der zugehörigen Assistenten-Definitionsdatei.
{clsidPackage}	Optional. Klassen-ID einer DLL, die lokalisierte Ressourcen für den Typ bereitstellt.
Lokalisierter Name	Optional. Enthält den lokalisierten Namen der Vorlage, der im Dialog erscheint. String oder Resource Identifier (#ResID) sind möglich.
Sort-Priorität	Pflichtfeld. Eine Zahl, die für die Reihenfolge steht, in der die Vorlage im Dialog erscheint, wobei 1 am höchsten ist.
Beschreibung	Pflichtfeld. Eine lokalisierte Beschreibung der Vorlage, die im Dialog angezeigt wird. String oder Resource Identifier sind möglich.
DLL-Pfad oder {clsidPackage}	Pflichtfeld. Klassen-ID einer dll- oder exe-Datei, die das Projektsymbol enthält, welches über IconResourceId angegeben wird.
Icon-Resource-ID	Optional. Resource Identifier in der dll- oder exe-Datei, welcher das Vorlagensymbol angibt.
Flags	Pflichtfeld. Eine Beschreibung der einzelnen Flags finden Sie unter [1].
Vorzuschlagender Basisname	Pflichtfeld. Standardname, der im Namensfeld des Assistenten angezeigt wird. String oder Resource Identifier sind möglich.

In der ersten Zeile steht die Versionsnummer des Vorlagenformats. Sie ist abhängig von der installierten Visual-Studio-Version. Danach folgt die Prog-ID oder Klassen-ID (CLSID) des zugehörigen Assistenten. Optional können darunter beliebig viele Param-Zeilen folgen, die Parameter enthalten, die dem Assistenten beim Start übergeben werden.

Eine ausführliche Dokumentation des Dateiformats ist unter [2] zu finden.

Assistenten-Steuerungsskripte

Das eigentliche Generieren des Projekts wird nicht direkt vom Assistenten vorgenommen, sondern durch die JScript-Datei *Default.js*, die im *Scripts*-Verzeichnis unterhalb des jeweiligen Assistentenverzeichnisses liegt (siehe Abbildung 1).

Sie enthält verschiedene Funktionen, die vom Assistenten aufgerufen werden, zum Beispiel wenn der Projekterstellungsdialog geschlossen wird. Das Skript legt das Projekt an und fügt alle Dateien ein, die im Vorlagenordner hinterlegt sind. Dafür verwendet es unter anderem globale Funktionen, die in der allgemeinen Skriptdatei *common.js* definiert sind. Diese Datei befindet sich in einem sprachspezifischen Ordner unterhalb des Assistentenhauptverzeichnisses. Bei einer deutschen Installation ist dies *...\VC#Wizards\1031* für C#, beziehungsweise *...\VBWizards\1031* für VB.NET. Der Aufbau und die Anpassung dieser Skriptdateien werden später näher beschrieben, wenn es um eigene Vorlagen geht.

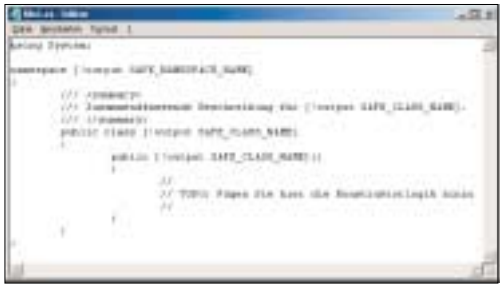


Abbildung 2 | Dateivorlage für eine Klasse.

Vorhandene Vorlagen anpassen

Im Verzeichnis *VC#Wizards* (C#) beziehungsweise *VBWizards* (VB.NET) befindet sich für jeden Projekt- und Projektelementtyp ein Unterverzeichnis, das die Definitionen und Vorlagendateien enthält, auf deren Grundlage ein Element erstellt wird (siehe Abbildung 1). Der Verzeichnisname entspricht hierbei dem Inhalt des *WIZARD_NAME*-Eintrags in der *Param*-Auflistung der Assistenten-Definitionsdatei (siehe oben). Für ein C#-Klassenbibliothek-Projekt heißt das Verzeichnis zum Beispiel *CSharpDLLWiz*.

In diesen Verzeichnissen gibt es jeweils die zwei Unterverzeichnisse *Scripts* und *Templates*, welche die Assistenten-Steuerungsdateien beziehungsweise die Element-Vorlagendateien enthalten. Unterhalb dieser Verzeichnisse befindet sich je ein sprachabhängiges Verzeichnis, das beispielsweise bei einer deutschen Installation mit *1031* und bei einer englischen mit *1033* benannt ist.

Das Verzeichnis *Templates* beinhaltet, im Falle von C#, eine Datei mit dem Namen *Templates.inf*. Sie enthält die Namen der Dateien, die beim Erstellen eines Elements dieses Typs erzeugt werden sollen. Außerdem sind die Dateien, die in *Templates.inf* aufgeführt sind, im Verzeichnis enthalten. In VB.NET-Vorlagen ist dies genauso, allerdings existiert hier keine *Templates.inf*-Datei.

Abbildung 3 | Vorlagen für Projektelemente, wie sie in VS.NET angezeigt werden.

Bei einem C#-Klassenbibliothek-Projekt enthält zum Beispiel das Verzeichnis *CSharpDLLWiz* folgende Dateien:

- *Templates.inf*
- *assemblyinfo.cs*
- *file1.cs*

assemblyinfo.cs und *file1.cs* können nun nach eigenen Vorstellungen verändert werden. So könnte etwa das *AssemblyCompany*-Attribut in *assemblyinfo.cs* mit dem eigenen Firmennamen vorbelegt werden.

In der Datei *file.cs*, die in Abbildung 2 dargestellt ist, existieren zudem einige Textmarken, die für die Benennung von Namensräumen und Klassen durch den Assistenten verwendet werden. So ersetzt der Assistent zum Beispiel die Textmarke *[!output SAFE_NAMESPACE_NAME]* durch den angegebenen Projektnamen. Eine sinnvolle Anpassung der Klassendatei wäre etwa die Vorbelegung mit Code-Regions oder der Zuweisung bestimmter Attribute.

Vorlagen für Projektelemente anlegen

Neben der gezeigten Methode, die vorhandenen Vorlagen anzupassen, besteht auch die Möglichkeit, eigene Vorlagen für die verschiedenen Projektelemente zu erstellen. Diese werden zum Beispiel im Dialog *Neues Element hinzufügen* angezeigt, der erscheint, wenn über das Projekt-Menü ein Element eingefügt wird (siehe Abbildung 3).

Hierfür existiert unterhalb des Projektelementvorlagen-Verzeichnisses (*CSharpProjectsItems* für C#, *VBProjectsItems* für VB.NET) je Anwendungstyp ein Unterverzeichnis, welches die Projektvorlagen für Windows- beziehungsweise Webanwendungen enthält. Je nach verwendeter Programmiersprache heißen diese Verzeichnisse:

- *LocalProjectItems* und *WebProjectItems* (C#),
- *Local Project Items* und *Web Project Items* (VB.NET).

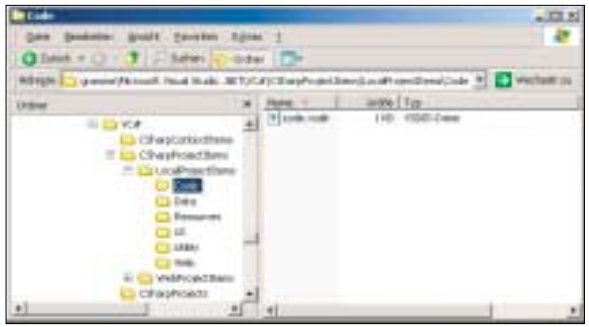
Darin befinden sich *vsdir*-Dateien, die Definitionen der Elemente enthalten, die angezeigt werden, wenn das Root-Element im Dialog ausgewählt wird, wie dies in Abbildung 3 zu sehen ist. Zusätzlich existieren mehrere Unterverzeichnisse, die zur Gruppierung der verschiedenen Elemente wie Code-Dateien, Datenzugriff oder Web-Klassen dienen.

In ihnen sind, analog zum Hauptverzeichnis, wieder *vsdir*-Dateien hinterlegt, die Definitionen zu den einzelnen Elementen enthalten (siehe Abbildung 4). Nach diesem Prinzip lassen sich auch eigene Gruppierungsverzeichnisse erstellen, die zum Beispiel auf firmeninterne Klassenvorlagen verweisen.

In Tabelle 3 sind die wichtigsten Verzeichnisse mit den entsprechenden Vorlagen-Definitionsdateien aufgeführt.

Beispielvorlage Standarddialog

Eine sinnvolle Vorlage für Windows-Anwendungen ist die Definition eines Standarddialogs. Die Vorlage besteht lediglich aus einem Fenster, das als Dialog formatiert wurde und die Schaltflächen *OK* und *Abbrechen* enthält (siehe Abbildung 5). Durch die Definition



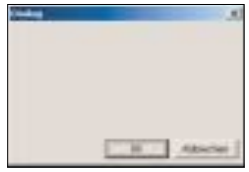


Abbildung 5 | Ein Standarddialog als Projektelementvorlage.

Abbildung 4 | Vorlagen für Projektelemente, wie sie im Dateisystem angelegt sind.

des Dialogs als Vorlage werden die Arbeitsschritte bei deren Verwendung eingespart. Außerdem sehen die Schaltflächen in allen Dialogen immer gleich aus.

Das Fenster könnte zum Beispiel von einer Dialogklasse abgeleitet werden. So könnte die Optik aller Dialoge auch noch im Nachhinein über die Basisklasse geändert werden.

Um nun den Dialog als Vorlage zu definieren, muss zunächst im Verzeichnis

```
...\VC#\CSharpProjectItems\LocalProjectItems\UI
```

beziehungsweise

```
...\Vb7\VBProjectItems\Local Project Items\UI
```

ein neuer Eintrag in der Datei *ui.vst* (C#) beziehungsweise *LocalUIProjectItems.vsdir* (VB.NET) hinterlegt werden. Hierbei ist es am einfachsten, die Zeile eines vorhandenen Elements gleichen Typs zu kopieren. Da es sich in diesem Beispiel um ein Windows-Form-Element handelt, bietet sich folgende Zeile an, die meist am Anfang der Datei steht:

```
..\..\CSharpAddWinFormWiz.vsz|{FAE04EC1-301F-11d3-BF4B-
00C04F79EFBC}|#2237|10|#2264|{FAE04EC1-301F-11d3-BF4B-
00C04F79EFBC}|4535|0|Form.cs
```

Nachdem die Zeile an das Ende der Datei kopiert wurde, müssen die folgenden Änderungen durchgeführt werden (fett markiert):

➤ **C#:**

```
..\..\CSharpAddDialogWiz.vsz|{FAE04EC1-301F-11d3-BF4B-
00C04F79EFBC}|Dialog|10|Standarddialog mit ‚OK'- und ‚Abbrechen'-
Schaltfläche|{FAE04EC1-301F-11d3-BF4B-00C04F79EFBC}|4535|0|Dialog.cs
```

➤ **VB.NET:**

```
..\..\Dialog.vsz|{164B10B9-B200-11D0-8C61-
00A0C91E29D5}|Dialog|10|Standarddialog mit ‚OK'- und ‚Abbrechen'-
Schaltfläche|{164B10B9-B200-11D0-8C61-00A0C91E29D5}|4527| |Dialog.vb
```

Angepasst wurden hier eine Assistentendatei, der Titel der Vorlage mit Beschreibungstext und der Standarddateiname.

Nun muss die angegebene Assistenten-Definitionsdatei erstellt werden. Dazu kann die vorhandene Datei *CSharpAddWebFormWiz.vsz* im Verzeichnis *...\CSharpProjectItems* kopiert und in *CSharpAddDialogWiz.vsz* umbenannt werden. In VB.NET ist die Datei *WinForm.vsz* aus dem Verzeichnis *...\VBProjectItems* zu kopieren und in *Dialog.vsz* umzubenennen.

Die Änderungen in den Dateien sehen wie folgt aus:

➤ **C#:**

```
Param="WIZARD_NAME = CSharpAddDialogWiz"
```

➤ **VB.NET:**

```
Param="WIZARD_NAME = Dialog"
```

Tabelle 3 | Die verschiedenen Typen bei Komponentenvorlagen.

Typ-Verzeichnis	Beschreibung	C#-Datei	VB-Datei
Code	Vorlagen für verschiedene Klassentypen (Komponentenklasse, Windows-Dienst, usw.)	code.vsdir	LocalCodeProjectItems.vsdir
Data	Vorlagen für Datenzugriffs- und XML-Klassen	data.vsdir	LocalDataProjectItems.vsdir
Resources	Vorlagen für Bitmaps, Cursor, Icons und Assembly-Ressourcen	LocalResources.vsdir/ WebResources.vsdir	(bei VB nicht vorhanden)
UI	Vorlagen für Formulare und Steuerelemente	ui.vsdir	LocalUIProjectItems.vsdir
Utility	Sonstige Vorlagen für Scripting und Reporting	utility.vsdir	LocalUtilityProjectItems.vsdir
Web	Vorlagen für HTML-Dateien, Stylesheets, Framesets, usw.	web.vsdir	LocalWebProjectItems.vsdir

Im Assistentenverzeichnis (siehe Abbildung 1) wird ein Vorlagenordner benötigt, der die Source- und Steuerungsdateien der Vorlage enthält. Hierfür kann der vorhandene Ordner *CsharpAddWinFormWiz* kopiert und in *CsharpAddWinFormWiz* umbenannt werden. In VB.NET ist das Verzeichnis *WinForm* zu kopieren und in *Dialog* umzubenennen. Im neuen Verzeichnis ist zunächst eine Änderung der Skriptdatei *Default.js* unter *...\Scripts\1031* vorzunehmen. Im Falle von C# sind alle *NewWinForm.cs*-Einträge durch *Dia-*

Abbildung 6 | Der Dialog zum Einfügen von Projektelementen mit der Standarddialog-Vorlage.

log.cs zu ersetzen. Dies ist der Name der Source-Datei für den Standarddialog. Im Falle von VB.NET sind alle *Form.vb*-Einträge durch *Dialog.vb* zu ersetzen.

Schließlich sind alle Vorlagendateien in das *Templates*-Verzeichnis zu kopieren. Im Falle des Standarddialogs ist dies nur die Datei *Dialog.cs*. Die vorhandenen Dateien der alten Vorlage können gelöscht werden. In C# muss zusätzlich die Datei *Templates.in* mit den Dateinamen gefüllt werden.

Als Letztes sind noch in *Dialog.cs* Namespace- und Klassenname durch Textmarken zu ersetzen. Der Assistent fügt an diesen Stellen später gültige Namen ein. Folgende Änderungen sind nötig:

• in der Namespace-Zeile

```
namespace [!output SAFE_NAMESPACE_NAME]
```

• in der Definitionszeile der Klasse

```
public class [!output SAFE_ITEM_NAME] : System.Windows.Forms.Form
```

• im Konstruktor

```
public [!output SAFE_ITEM_NAME]()
```

• und schließlich in der Methode *InitializeComponent* der Eintrag

```
this.Name = "[!output SAFE_ITEM_NAME]";
```

Jetzt kann die Vorlage in der IDE verwendet werden. Hierbei ist jedoch zu beachten, dass Änderungen, die durchgeführt werden, während die IDE läuft, nicht berücksichtigt werden, da VS.NET die Dateien beim Starten zwischenspeichert.

Die *Standarddialog*-Vorlage kann über das Projektmenü oder über das Kontextmenü im Projektfenster mit *Add new Item* eingefügt werden (siehe Abbildung 6). Der Namespace des Dialogs wird dabei durch den Namespace des Projektes und der Klassenname durch den angegebenen Dateinamen ersetzt.

Projektvorlagen erstellen

Das Erstellen von Projektvorlagen verläuft fast auf die gleiche Weise wie bei den Projektelementen. Im Folgenden werden die Schritte beschrieben, die für eine Erstellung eines Windows-Exe-Projekts erforderlich sind. Zunächst ändern Sie die entsprechende *vsdir*-Datei, die im Verzeichnis *CSharpProjects* beziehungsweise *VBProjects* zu finden ist. Dort sind in den Dateien *CSharp.vsdir* und *CSharpEx.vsdir* alle Projektdefinitionen notiert. Erstere enthält Standardprojekte wie Windows-, Web- und Konsolenanwendungen, während in Letzterer spezielle Projekte wie Steuerelemente, Klassenbibliotheken und Services enthalten sind. In VB.NET heißen diese Dateien *projects_std.vsdir* und *Projects.vsdir*.

Um ein Windows-Exe-Projekts anzulegen, kopieren Sie in der Datei *CSharp.vsdir* beziehungsweise *projects_std.vsdir* die Definitionszeile für diesen Projekttyp. Bei C# ist dies die Zeile, die mit *CSharpExe.vs* beginnt, bei VB.NET *WindowsApplication.vsz*. Die Änderungen sehen wie folgt aus:

➤ **C#:**

```
CSharpMyEXE.vsz|{FAE04EC1-301F-11d3-BF4B-
00C04F79EFBC}|MyWindowsApp|10|Benutzerdefinierte Windows-
Anwendung|{FAE04EC1-301F-11d3-BF4B-00C04F79EFBC}|4554| |MyWindows-
Application
```

➤ **VB.NET:**

```
MyWindowsApplication.vsz|{164B10B9-B200-11D0-8C61-
00A0C91E29D5}|MyWindowsApp|10|Benutzerdefinierte Windows-
Anwendung|{164B10B9-B200-11D0-8C61-00A0C91E29D5}|4507| |MyWindows-
Application
```

Die angegebene Assistentenvorlage *CSharpMyEXE.vsz* kann durch Kopieren der Datei *CSharpEXE.vsz*, die im gleichen Verzeichnis wie die *vsdir*-Datei liegt, erstellt werden. Bei VB.NET ist die Datei *WindowsApplication.vsz* zu kopieren und in *MyWindowsApplication.vsz* umzubenennen. In der Datei muss wieder der Name des zugehörigen Assistenten geändert werden.

➤ **C#:**

```
Param="WIZARD_NAME = CSharpMyEXEWiz"
```

➤ **VB.NET:**

```
Param="WIZARD_NAME = MyWindowsApplication"
```

Richten Sie wie schon bei der *Standarddialog*-Vorlage ein Assistentenverzeichnis ein, das die Projektdateien enthält. Dazu kopieren Sie das Verzeichnis *...\VC#Wizards\CSharpEXEWiz* und benennen es in *CSharpMyEXEWiz* um. In VB.NET kopieren Sie das Verzeichis *...\VBWizards\WindowsApplication* und benennen es in *\VBWizards\MyWindowsApplication* um.

Im Unterverzeichnis *Templates\1031* können daraufhin die Standarddateien der Projektvorlage bearbeitet werden. Abbildung 7 zeigt das entsprechende Verzeichnis für C#.

Sollen zusätzliche Dateien in das Projekt aufgenommen werden, so sind diese einfach in das Verzeichnis zu kopieren und im Fall von C# ist ein entsprechender Eintrag in der *Templates.inf* zu hinterlegen. Außerdem sind die Klassen- und Namespace-Angaben durch entsprechende Textmarken zu ersetzen, wie dies bereits am Beispiel des Standarddialogs beschrieben wurde. Beim Hinzufügen von neuen Dateien, beziehungsweise dem Umbenennen oder Löschen vorhandener Dateien muss zudem das Projektskript (*Default.js*) geändert werden. Hier findet sich in Zeile 16 (C#) beziehungsweise Zeile 17 (VB.NET) ein Verweis auf eine Projektdatei.

➤ **C#:**

```
var proj = CreateCSharpProject(
strProjectName, strProjectPath,
"DefaultWinExe.csproj");
```

➤ **VB.NET:**

```
var strTemplateFile = strTemplatePath +
"\\WindowsApplication.vbproj";
```

Diese Datei wird bei der Projekterstellung als Vorlage verwendet. Im Falle von C# liegt sie im Assistentenhauptverzeichnis (*VC#Wizards*(C#)) und enthält Voreinstellungen zum Beispiel für Debug- und Release-Optionen. Möchten Sie eigene Einstellungen hinterlegen, kopieren Sie die Datei, nehmen die Änderungen vor und tragen den Namen der neuen Datei im Skript ein. In VB.NET liegt diese Datei im jeweiligen Vorlagenverzeichnis.

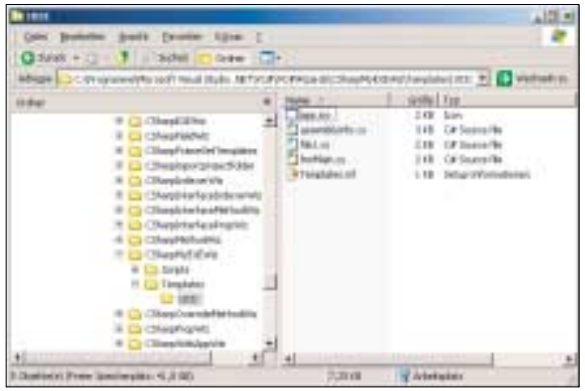


Abbildung 7 | Das Templates-Verzeichnis einer Projektvorlage.

Das Skript fügt nun alle Dateien im Verzeichnis dem Projekt hinzu. Das sieht in VB.NET wie folgt aus (Zeile 37 bis 46):

```
strTemplateFile = strTemplatePath + "\\Form.vb";
item = AddFileToVSProject("Form1.vb", project, project.ProjectItems,
strTemplateFile, false);
if( item )
{
 item.Properties("SubType").Value = "Form";
 project.Properties("StartupObject").Value = project.Properties(
"RootNamespace").Value + ".Form1";

 editor = item.Open(vsViewKindPrimary);
 editor.Visible = true;
}
```

Hier wird in der Funktion *OnFinish* der Pfad zur entsprechenden Datei erzeugt und dem Projekt über *AddFileToVSProject* hinzugefügt. Dabei kann im ersten Parameter der Dateiname angegeben werden, unter dem die Datei angelegt werden soll.

Im Anschluss wird der Typ des Elements definiert und das Formular als Startobjekt der Anwendung festgelegt. Die letzten beiden Zeilen bewirken, dass das Formular angezeigt wird, nachdem das Projekt angelegt wurde.

Soll ein zusätzliches Formular ins Projekt aufgenommen werden, müssen unterhalb des gezeigten Blocks folgende Zeilen eingefügt werden:

```
strTemplateFile = strTemplatePath + "\\MyForm.vb";
item = AddFileToVSProject("MyForm.vb", project, project.ProjectItems,
strTemplateFile, false);
if( item )
{
 item.Properties("SubType").Value = "Form";
}
```

Im Falle von Klassendateien muss die *SubType*-Eigenschaft mit *Code* belegt werden. Bei C#-Projekten ist dies ein wenig anders. Hier erledigt das Skript die Aufnahme der Dateien ins Projekt automatisch, indem es die *Templates.inf*-Datei ausliest.

In der Funktion *GetCSharpTargetName* werden für die entsprechenden Dateien Zielnamen generiert.

```
function GetCSharpTargetName(strName, strProjectName)
{
 var strTarget = strName;

 switch (strName)
 {
    case "readme.txt":
       strTarget = "ReadMe.txt";
       break;
    case "File1.cs":
       strTarget = "Form1.cs";
       break;
    case "assemblyinfo.cs":
       strTarget = "AssemblyInfo.cs";
       break;

 }
 return strTarget;
}
```

Hier könnten nun weitere *case*-Zweige für eigene Dateien angelegt und entsprechende Zieldateinamen hinterlegt werden.

In der Funktion *DoOpenFile* können Sie darüber hinaus festlegen, welche Dateien beim Anlegen des Projekts geöffnet werden sollen. Standardmäßig ist dies nur die Datei *Form1.cs*.

```
function DoOpenFile(strName)
{
 var bOpen = false;
 switch (strName)
 {
    case "Form1.cs":
       bOpen = true;
       break;
 }
 return bOpen;
}
```

Für die Festlegung der Objekttypen ist die Funktion *SetFileProperties* zuständig. Hier werden den Dateien, analog zu

Abbildung 8 | Der Dialog zum Erstellen von Projekten mit der eigenen Projektvorlage.

VB.NET, über die *SubType*-Eigenschaft die entsprechenden Typen zugewiesen.

```
function SetFileProperties(oFileItem, strFileName)
{
      if(strFileName == "File1.cs")
      {
            oFileItem.Properties("SubType").Value = "Form";
      }
}
```

Nachdem alle Änderungen durchgeführt wurden, sollte im *Neues-Projekt*-Dialog ein Eintrag für die eigene Projektvorlage erscheinen, wie dies in Abbildung 8 zu sehen ist.

Einfügen von Referenzen

Auch die Referenzen, die einem neuen Projekt hinzugefügt werden, lassen sich anpassen. Bei einem Windows-Forms-Projekt ist in der Skriptdatei der Methodenaufruf *AddReferencesForWinForm* für die Aufnahme der Referenzen zuständig. Bei anderen Projekttypen kann die Methode unter Umständen anders heißen (zum Beispiel *AddReferencesForClass* bei einer Klassenbibliothek).

Die Methoden sind im *Common.js*-Skript definiert, welches bei einer deutschen Installation unter *...\VC#\VC#Wizards\1031* zu finden ist.

Hier werden die Referenzen ins übergebene Projekt aufgenommen.

```
function AddReferencesForWinForm(oProj)
{
 var refmanager = GetCSharpReferenceManager(oProj);
 var bExpanded = IsReferencesNodeExpanded(oProj)
 refmanager.Add("System");
 refmanager.Add("System.Data");
 refmanager.Add("System.Drawing");
 refmanager.Add("System.Windows.Forms");
 refmanager.Add("System.XML");
 if(!bExpanded)
    CollapseReferencesNode(oProj);
}
```

Methoden sollten nicht verändert werden, da sie unter Umständen von mehreren Vorlagen verwendet werden. Stattdessen sollten Sie diese kopieren und umbenennen. Anschließend ist der Aufruf aus dem *Default.js*-Skript entsprechend zu ändern.

Weiterführende Themen

Neben den hier vorgestellten Arbeitserleichterungen können Sie auch mit Hilfe von eigenen Assistenten eine Effizienzsteigerung erreichen. Wie diese Sie beim Anlegen von Projekten und Projektelementen unterstützen, lesen Sie im vorhergehenden Beitrag. Auch das Anlegen von Top-Level-Ordnern im *Neues-Projekt*-Dialog ist möglich. In diesem Fall würde ein Ordner auf der gleichen Ebene wie VB.NET-Projekte und C#-Projekte eingefügt. Eine Beschreibung solcher Anpassungen finden Sie unter [3].

Darüber hinaus ist es möglich, für Vorlagen bestimmte Richtlinien zu hinterlegen, die Regeln für das Erstellen, das Codieren und für die Dokumentation enthalten können. Diese Vorlagen werden in VS.NET als Enterprise Templates bezeichnet. [4] bietet eine gute Einführung in das Thema. |||||||

[1] Formatbeschreibung der VSDIR-Dateien
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vsintro7/html/vxconvsdirfiles.asp

[2] Formatbeschreibung der VSZ-Dateien
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vsintro7/html/vxconvsdirfiles.asp

[3] Adding Custom Folders for Enterprise Template Projects to Visual Studio .NET Dialog Boxes
http://msdn.microsoft.com/library/en-us/dv_vstechart/html/vstchAddingCustomFoldersForEnterpriseTemplateProjectsToVisualStudioNETDialogBoxes.asp

[4] Enterprise Templates: Building an Application Construction Kit
http://msdn.microsoft.com/library/en-us/dv_vstechart/html/vstchEnterpriseTemplatesBuildingApplicationConstructionKit.asp