25 Der My-Namespace

715 Formulare ohne Instanzierung aufrufen

716 Auslesen der Befehlszeilenargumente mit My.Application.CommandLineArgs

718 Gezieltes Zugreifen auf Ressourcen mit My.Resources

721 Internationalisieren von Anwendungen mithilfe von Ressource-Dateien und dem My-Namespace

724 Vereinfachtes Durchführen von Dateioperationen mit My.Computer.FileSystem

727 Verwenden von Anwendungseinstellungen mit My.Settings

Der `My`-Namespace wurde in Visual Basic 2005 eingeführt, um den Umgang mit bestimmten Funktionalitäten zu vereinfachen – um sozusagen »Abkürzungen« zu bestimmten Zielen mit Funktionalitäten im .NET-Framework zur Verfügung zu stellen, die Sie normalerweise nur auf verschlungenen Pfaden erreichen würden.

So Sie das Beispielszenario aus ▶ Kapitel 3 komplett durchexerziert haben, sind Sie auch schon in den Genuss dieser Abkürzungen gekommen – Sie haben dort nämlich kennen gelernt, wie einfach Sie auf selbst speichernde Anwendungseinstellungen durch `My.Settings` zurückgreifen können.

Doch der `My`-Namespace verfügt über noch weit mehr wirklich coole Features. Welche das allerdings genau sind, lässt sich nicht verbindlich sagen, denn die Features, die Ihnen durch `My` zur Verfügung stehen, hängen von dem Projekttyp ab, den Sie gerade bearbeiten.

Generell teilt sich die Funktionalität im `My`-Namespace in die folgenden Kategorien auf:

- **My.Application:** Stellt Zugriff auf Informationen über die aktuelle Anwendung bereit, wie beispielsweise den Pfad zur ausführbaren Datei, die Programmversion, derzeitige Kulturinformationen oder den Benutzer-Authentifizierungsmodus.
- **My.Computer:** Ermöglicht Ihnen den Zugriff auf mehrere untergeordnete Objekte, die Ihnen wiederum das Abrufen von computerbezogenen Informationen gestatten, wie beispielsweise über das verwendete Dateisystem, über verwendete Audio- und Videospezifikationen, angeschlossene Drucker oder generelle I/O-Hardware wie Maus, Tastatur, Speicher, die verwendete Netzwerkumgebung, serielle Schnittstellen und Weiteres.
- **My.Forms:** Gestattet Ihnen das Abrufen von Standardinstanzen aller Formulare einer Windows Forms-Anwendung.

- **My.Resources:** Ermöglicht den einfachen Zugriff auf eingebettete Ressourcen, wie beispielsweise auf Zeichenkettentabellen, Bitmaps und Ähnliches.
- **My.Settings:** Ermöglicht einerseits, auf Anwendungseinstellungen Zugriff zu nehmen, und sorgt andererseits dafür, dass diese Anwendungseinstellungen im Bedarfsfall benutzerabhängig beim Programmstart wiederhergestellt und beim Programmende gesichert werden.
- **My.User:** Ermöglicht das Abrufen von Infos über den zurzeit angemeldeten Benutzer und erlaubt ferner das Implementieren benutzerdefinierter Authentifizierungsmechanismen.
- **My.WebServices:** Stellt eine Eigenschaft für jeden Webservice zur Verfügung, den das aktuelle Projekt referenziert, und erlaubt so auf Webservices Zugriff zu nehmen, ohne eine explizite Proxy-Klasse für den jeweiligen Webservice erstellen zu müssen.

In einer Konsolenanwendung, die nun einmal keine Formulare verwendet, steht Ihnen natürlich die Kategorie `My.Forms` nicht zur Verfügung. Und so hängt die tatsächliche Verfügbarkeit der `My`-Features durchweg von den Projekttypen ab, mit denen Sie es gerade zu tun haben, wie die folgende Tabelle zeigt:

My-Objekt	Windows-Anwendung	Klassen-bibliothek	Konsolen-anwendung	Windows-Steuerelement-bibliothek	Web-Steuerelement-bibliothek	Windows-Dienst
`My.Application`	Ja	Ja	Ja	Ja	Nein	Ja
`My.Computer`	Ja	Ja	Ja	Ja	Ja	Ja
`My.Forms`	Ja	Nein	Nein	Ja	Nein	Nein
`My.Resources`	Ja	Ja	Ja	Ja	Ja	Ja
`My.Settings`	Ja	Ja	Ja	Ja	Ja	Ja
`My.User`	Ja	Ja	Ja	Ja	Ja	Ja
`My.WebServices`	Ja	Ja	Ja	Ja	Ja	Ja

Nun finde ich, es würde an dieser Stelle keinen Sinn machen, die komplette Referenz aller Funktionalitäten jedes einzelnen Bereichs herunterzubeten – im Gegenteil: Schließlich dient `My` eben genau dazu, auch ohne großes Blättern und nur mithilfe von IntelliSense und dynamischer Hilfe, schnell an die gewünschte Funktionalität zu gelangen.

Stattdessen sollten Sie gerade von einem Entwicklerbuch erwarten können, dass Ihnen der Autor Sachverhalte im Rahmen einer Softwareentwicklung näher bringt – und das hat er für die Demonstration von `My` auch getan. Die in den folgenden Abschnitten beschriebenen `My`-Funktionalitäten erheben deshalb keinen Anspruch auf Vollständigkeit – sie sollen es auch gar nicht. Ziel ist es vielmehr, zum einen generelle Vorgehensweisen beim Umgang mit dem `My`-Namespace zu vermitteln. Die richtigen Funktionen für Ihre eigenen Bedürfnisse zu finden, wird dank IntelliSense und Online-Hilfe sicherlich nicht *das* Problem sein, und würde an dieser Stelle nur wertvollen Platz beschränken.

Es gibt zum anderen auch einiges Erwähnenswertes zum `My`-Namespace und seinen Funktionalitäten, was Sie nicht in der Online-Hilfe finden, und auch diesen Punkten widmen sich die folgenden Abschnitte.

BEGLEITDATEIEN: Viele der hier gezeigten Features können Sie sich direkt am Beispiel von DotNetCopy (vorgestellt im vorherigen Kapitel) anschauen, und innerhalb der Abschnitte werden Sie auch immer wieder Codeausschnitte aus diesem Beispiel finden. Sie finden dieses Projekt unter dem Namen *DotNetCopy.sln* im Verzeichnis *.\VB 2005 - Entwicklerbuch\ F - Vereinfachung\Kap25\DotNetCopy*. So Sie das vorherige Kapitel noch nicht gelesen haben, sollten Sie es zunächst tun, um die Beispiele besser nachvollziehen zu können.

Formulare ohne Instanzierung aufrufen

Die Änderungen an Visual Basic, um die es jetzt geht, haben in der bis dato etablierten VB.NET-Gemeinde für das meiste Aufsehen gesorgt,[1] denn: Es geht um die Vereinfachung von Formularaufrufen, und das Hauptargument dabei ist: Damit nähere sich Visual Basic politisch betrachtet wieder einer »Kindersprache«, bei der man dem »unmündigen« VB-Programmierer nicht zumuten kann, die buchstäblichen Basics der OOP zu kennen und anzuwenden.

Worum geht es genau?

Auch schon in VB6 mussten Sie, um mit einem Objekt arbeiten zu können, es zunächst instanzieren. Haben Sie also eine `Collection` verwendet, in der Sie andere Objekte speichern wollten, waren folgende Zeilen notwendig:

```
Dim auflistung As Collection
Set auflistung = New Collection
auflistung.Add 5
```

Es ist klar, dass diese Zeilen nicht funktioniert hätten:

```
Collection.Add 10
Collection.Add 15
Collection.Add 20
```

Bei VB6-Formularen passiert aber genau das: Hier können Klassenname und Instanz das gleiche sein. Diese Version funktioniert:

```
Dim f As Form1
f.Show
```

genau wie die Verwendung der Klasse `Form1` als Instanzvariable:

```
Form1.Show
```

Das entspricht natürlich nicht den Vorschriften zur objektorientierten Programmierung, war aber wahrscheinlich für viele VB6-Programmierer einfacher zu verstehen, und aus diesem Grund wurde der Basic-Compiler so frisiert, dass eine solche Verwendung einer Formularklasse möglich war.

Schon in Visual Basic 6 passierte »unter der Haube« dazu etwas, was nunmehr auch in VB2005 wieder möglich ist.

Der VB-Compiler sorgt nämlich bei der Kompilierung einer Windows Forms-Anwendung dafür, dass versteckter Quellcode, der im Compiler fest verdrahtet ist, mit in Ihre Anwendung kompiliert

[1] Vergleichen Sie dazu bitte auch den Einführungstext zu diesem Buchteil.

wird. Der VB-Compiler »erfindet« also quasi einen Haufen von Quellcode und fügt diesen, unsichtbar für den Entwickler, dem eigentlichen Windows-Projekt hinzu.

So können Sie auch in Visual Basic 2005 wieder folgende Zeile verwenden, um das unter dem Klassennamen `Form2` erstellte Formular beispielsweise als modalen Dialog ins Leben zu rufen:

```
Form2.ShowDialog
```

Doch es sieht hier nur so aus, als würden Sie eine statische Klassenmethode direkt verwenden, denn in Wirklichkeit ist `Form2` eine Eigenschaft, die eine *Instanz* vom Typ `Form2` zurückliefert. Denn was hier eigentlich passiert, ist, dass der Compiler diese Zeile nur vervollständigt, und er im Grunde genommen folgenden Quellcode für die Generierung der eigentlichen ausführbaren Datei kompiliert:

```
MyProject.Forms.Form2.ShowDialog
```

Die Klasse `MyProject` ist dabei allerdings eine Klasse, die Sie unter diesem Namen nicht erreichen können, weil sie mit entsprechenden Attributen so gekennzeichnet ist, dass IntelliSense sie nicht »sieht«. Sie ist aber dennoch vorhanden. Quasi versteckt legt der Compiler also für jedes Formular, das Sie Ihrem Projekt hinzufügen, eine Eigenschaft vom Typ des jeweiligen Formulars in der (ebenfalls unsichtbaren) `MyForms`-Klasse an, die eine eingebettete Klasse der Klasse `MyProject` selbst ist, deren Instanz durch die `Forms`-Eigenschaft der `MyProject`-Klasse abgerufen werden kann. Und erst diese Eigenschaft, die den Namen des Formulars trägt, sorgt dafür, dass die gewünschte Formularklasse vor Gebrauch instanziert und dann ohne Probleme (bzw. `NullReference`-Ausnahmen) verwendet werden kann.

Sie können das ausprobieren. Tippen Sie »**My.**«, dann werden Sie sehen, dass IntelliSense `MyProject` nicht zur Auswahl anbietet.

Komplettieren Sie jedoch die Zeile mit »**MyProject.Forms.Form2.ShowDialog**«, meldet der Visual Basic-Compiler dennoch keinen Fehler.

Zur Veranschaulichung: Die Programmzeile

```
MyProject.Forms.Form2.ShowDialog
```

bedeutet im Grunde genommen:

```
(MyProject-Klasse).(Forms-Eigenschaft der MyProject-Klasse vom Typ MyForms). _
(Form2-Eigenschaft der MyForms-Klasse vom Typ Form2).(ShowDialog-Methode der Form2-Klasse).
```

Auslesen der Befehlszeilenargumente mit My.Application.CommandLineArgs

Anwendungen, die unter Windows laufen, können Sie auf unterschiedliche Weisen starten. Die einfachste und bekannteste ist, sie durch einen Doppelklick auf eine Verknüpfung beispielsweise aus dem Startmenü zum Laufen zu bewegen. Sie können jede Windows- oder Befehlszeilenanwendung aber auch direkt durch die Angabe des Namens der ausführbaren Datei aus der Befehlszeile oder durch den *Ausführen*-Befehl des Start-Menüs heraus in Gang bekommen. Und viele Anwendungen erlauben es hier, weitere Parameter anzugeben, die das Startverhalten steuern. So starten Sie beispielsweise den Windows-Explorer mit

```
Explorer c:\
```

um zu definieren, dass der Explorer nicht nur gestartet wird, sondern dass er auch direkt den Inhalt des Wurzelverzeichnisses von Laufwerk *c:* anzeigt.

Parameter, die Sie Anwendungen auf diese Weise übergeben, nennt man Befehlszeilenargumente, und mithilfe des `My`-Namespace können Sie Befehlszeilenargumente auf einfache Weise ermitteln und auswerten.

Das `My`-Beispiel DotNetCopy arbeitet ebenfalls mit solchen Befehlszeilenargumenten, nämlich um die Anwendung in den Autostart- oder Silent-Modus zu versetzen. Der folgende Code, der sich im `StartUp`-Ereignis des `MyApplication`-Objektes befindet, und der ausgelöst wird, sobald die Anwendung startet, demonstriert den Umgang mit Befehlszeilenargumenten.

HINWEIS: Sie finden diesen Code in der Datei *ApplicationEvents.vb* des Projektes. Mehr zu Anwendungsereignissen erfahren Sie übrigens im nächsten Kapitel.

```
Partial Friend Class MyApplication

    Private Sub MyApplication_Startup(ByVal sender As Object, ByVal e As _
         Microsoft.VisualBasic.ApplicationServices.StartupEventArgs) Handles Me.Startup

        'Das Verzeichnis für die Protokolldatei beim ersten Mal setzen...
        If String.IsNullOrEmpty(My.Settings.Option_AutoSaveProtocolPath) Then
            My.Settings.Option_AutoSaveProtocolPath = _
                My.Computer.FileSystem.SpecialDirectories.MyDocuments & "\DotNetCopy Protokolle"
            Dim locDi As New DirectoryInfo(My.Settings.Option_AutoSaveProtocolPath)

            'Überprüfen und
            If Not locDi.Exists Then
                'im Bedarfsfall anlegen
                locDi.Create()
            End If

            'Settings speichern
            My.Settings.Save()
        End If

        Dim locFrmMain As New frmMain

        'Kommandozeile auslesen
        'Sind überhaupt Befehlszeilenargumente vorhanden?
        If My.Application.CommandLineArgs.Count > 0 Then
            'Durch jedes einzelne Befehlszeilenargument durchiterieren
            For Each locString As String In My.Application.CommandLineArgs

                'Alle unnötigen Leerzeichen entfernen und
                'Groß-/Kleinschreibung 'Unsensiblisieren'
                'HINWEIS: Das funktioniert nur in der Windows-Welt;
                'kommt die Kopierlistendatei von einem Unix-Server, bitte darauf achten,
                'dass der Dateiname dafür auch komplett in Großbuchstaben gesetzt ist,
                'da Unix- (und Linux-) Derivate Groß-/Kleinschreibung berücksichtigen!!!
                locString = locString.ToUpper.Trim
```

```
                If locString = "/SILENT" Then
                    locFrmMain.SilentMode = True
                End If

                If locString.StartsWith("/AUTOSTART") Then
                    locFrmMain.AutoStartCopyList = locString.Replace("/AUTOSTART:", "")
                    locFrmMain.AutoStartMode = True
                End If
            Next
        End If
.
.
.
```

Gezieltes Zugreifen auf Ressourcen mit My.Resources

Ressource-Dateien speichern bestimmte Elemente einer Anwendung in einer separaten Datei. Bei solchen Elementen handelt es sich beispielsweise um Bitmaps, Symbole, Zeichenketten oder Ähnliches. Diese Elemente können mithilfe von My.Resources zur Laufzeit aus Ressource-Dateien gelesen und anschließend im Kontext verwendet werden.

Der Vorteil dieser Vorgehensweise: Anwendungen können auch im Nachhinein angepasst werden, ohne dem Bearbeiter den eigentlichen Code der Anwendung zur Verfügung zu stellen. Denken Sie dabei nur zum Beispiel an die Lokalisierung einer Anwendung in eine andere Sprache. Wären die Texte der Anwendung fest verdrahtet, müsste man den Übersetzern die kompletten Quellen der Anwendung zur Verfügung stellen. Ein weiterer Nachteil wäre, dass es verschiedene Versionen des Quellcodes gäbe, von denen jede einzelne bei Änderungen oder Fehlerbehebungen in der Anwendung bearbeitet werden müsste – ein Aufwand, der bei größeren Projekten nicht zu leisten ist.

Anlegen und Verwalten von Ressource-Elementen

In unserer Beispielanwendung werden die Texte für die Spaltennamen der `ListView`, die die Kopierlisteneinträge beinhaltet, aus einer Ressource-Datei entnommen. Die Ressource-Datei selber wird ähnlich verwaltet, wie Sie es bei den Anwendungseinstellungen (den ApplicationSettings) bereits im ▶ 3. Kapitel kennen gelernt haben.

- Rufen Sie, um die Ressource-Datei einer Anwendung zu bearbeiten, einfach die Projekteigenschaften über das Kontext-Menü des Projektmappen-Explorers auf.
- Wählen Sie anschließend die Registerkarte Ressourcen.
- Wählen Sie aus der linken Schaltfläche, die Sie aufklappen können (siehe Abbildung 25.1), den Typ Ressource, den Sie einpflegen möchten. Bei Zeichenketten erfassen Sie die Texte wie bei Anwendungseinstellungen in Form von Texttabellen. Bei anderen Ressource-Typen, wie beispielsweise Bitmaps, Symbolen oder Audio-Clips, verwenden Sie die Schaltfläche Ressource hinzufügen, um eine entsprechende Datei der Ressource-Datei hinzuzufügen.

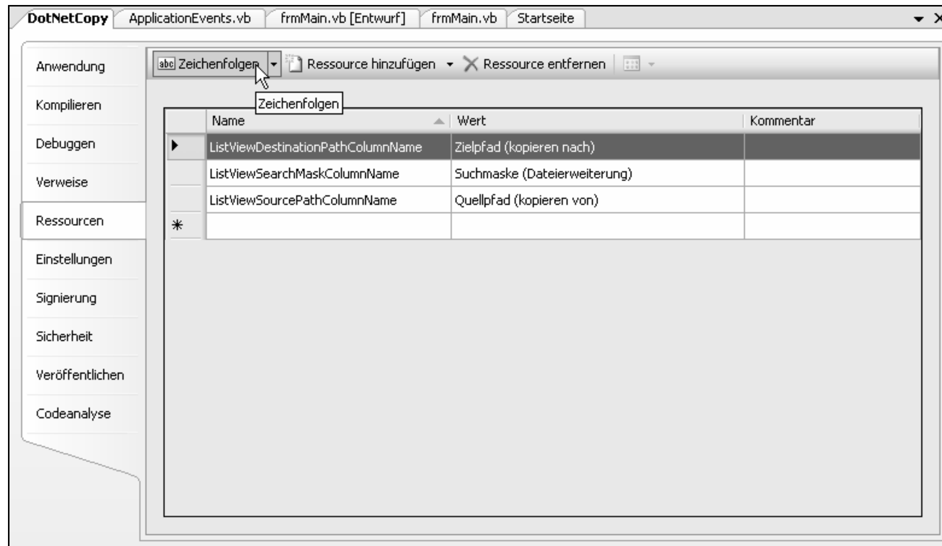

Abbildung 25.1: Bestimmen Sie wie hier den Typ Ressource, den Sie der Ressource-Datei hinzufügen möchten. Bei anderen Typen als Zeichenketten verwenden Sie Ressource hinzufügen, um Ressource-Elemente wie Bitmaps, Symbole oder Audio-Clips der Ressource-Datei hinzuzufügen.

Abrufen von Ressourcen mit My.Ressources

Wenn Sie Ressourcen auf die im letzten Abschnitt beschriebene Weise Ihrem Projekt hinzugefügt haben, ist das Abrufen der Ressource-Elemente zur Laufzeit dank My ein Einfaches. Im Beispielprojekt DotNetCopy wird das beispielsweise im Ereignisbehandler `form_Load` des Formulars gemacht; hier werden beispielhaft die Spaltentexte der `ListView` eingerichtet:

```
''' <summary>
''' Wird aufgerufen, wenn das Formular geladen wird, und enthält die
''' Initialisierung der ListView sowie das Anstoßen des Kopiervorgangs
''' (und das zuvor notwendige Laden der Kopierliste), wenn das Programm
''' im Autostart (aber nicht im Silent) -Modus läuft.
''' (Silent-Modus wird in ApplicationEvents.vb gehandelt).
''' </summary>
''' <param name="sender"></param>
''' <param name="e"></param>
''' <remarks></remarks>
Private Sub frmMain_Load(ByVal sender As Object, ByVal e As System.EventArgs) _
        Handles Me.Load
    'Listview einrichten
    With Me.lvwCopyEntries.Columns
        'Die Texte für die Listview-Spalten aus der Ressource-Datei entnehmen
        .Add(My.Resources.ListViewSourcePathColumnName)
        .Add(My.Resources.ListViewSearchMaskColumnName)
        .Add(My.Resources.ListViewDestinationPathColumnName)
```

```
            'Spalten ausrichten
            AlignColumns()
        End With
.
.
.
```

Natürlich könnte man dieses Verfahren auch beispielsweise auf die angezeigten Texte im Backup-Protokoll ausweiten. Der Einfachheit halber habe ich die Verwendung von Text-Ressourcen auf diesen Bereich beschränkt.

HINWEIS: Im Übrigen machen WinForms-Anwendungen beim Zuweisen von Texten oder Bilddaten ebenfalls implizit von Ressource-Dateien Gebrauch. Zwar verwendet der Windows Forms-Designer für den generierten Code nicht die Funktionalität aus dem `My`-Namespace, aber das Holen beispielsweise von Bitmap-Dateien aus einer Ressource, um dieser ein Symbol etwa einer Symbolleistenschaltfläche hinzuzufügen, funktioniert prinzipiell auf die gleiche Weise. Sie können sich selbst ein Bild davon machen: Wenn Sie im Projektmappen-Explorer die ausgeblendeten Dateien des Projektes mit dem Symbol *Alle Dateien anzeigen* (der Tooltip hilft beim Finden des Symbols) einblenden, können Sie einen Blick in den Code werfen, der zum Aufbau des Formulars führt. Dazu doppelklicken Sie anschließend auf die Datei *frmMain.Designer.vb*, die sich jetzt im Zweig unterhalb der Formulardatei *frmMain.vb* befindet.

```
    .
    .
    .
            '
            'tsbLoadCopyList – Auszug aus der FormsDesigner generierten Datei zum Zuweisen
            'eines Images an ein Symbol in DotNetCopy:
            Me.tsbLoadCopyList.DisplayStyle = System.Windows.Forms.ToolStripItemDisplayStyle.Image
            Me.tsbLoadCopyList.Image = CType(resources.GetObject("tsbLoadCopyList.Image"), _
                        System.Drawing.Image)

            Me.tsbLoadCopyList.ImageTransparentColor = System.Drawing.Color.Magenta
    .
    .
    .
```

Die Zuweisung eines Symbols mithilfe des `My`-Namespaces würde folgendermaßen ausschauen:

```
Me.tsbLoadCopyList.Image = My.Resources.LoadCopyList
```

Voraussetzung dafür wäre, dass eine entsprechende Image-Datei auf die zuvor beschriebene Weise der Ressource-Datei hinzugefügt wurde, etwa wie in Abbildung 25.2 zu sehen.

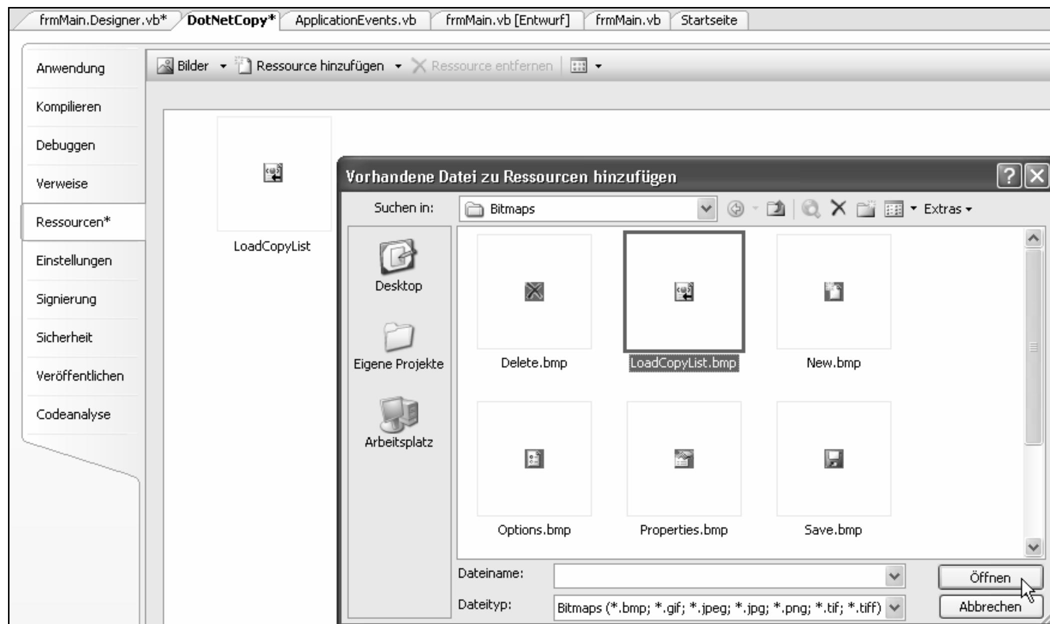

Abbildung 25.2: Klappen Sie die Schaltfläche *Ressource hinzufügen* auf (nicht darauf klicken!), und wählen Sie aus der Aufklappliste *Datei hinzufügen*, um beispielsweise eine Bitmap der Ressource-Datei hinzuzufügen, die Sie dann zur Laufzeit mit *My.Resource* und dem Element-Laden zuweisen können

Internationalisieren von Anwendungen mithilfe von Ressource-Dateien und dem My-Namespace

Der Einsatz von Ressource-Dateien macht natürlich nur dann Sinn, wenn Sie planen, eine Anwendung tatsächlich auch in einer anderen Sprache zur Verfügung zu stellen. Für diesen Vorgang des Lokalisierens müssen Sie weitere Ressource-Dateien erstellen, die auf Basis eines bestimmten Schemas benannt und dem Projekt hinzugefügt werden.

Leider gibt es in Visual Studio 2005 für Windows Forms-Anwendungen in Visual Basic keine Designer-Unterstützung, die Ihnen wirklich dabei hilft, neue Ressource-Dateien für andere Kulturen aus schon vorhandenen zu erstellen. Sie müssen dazu selbst Hand anlegen und die IDE auch ein wenig austricksen. Die Unterstützung im Designer beschränkt sich lediglich auf Formulare selbst, bei denen Sie einfach mit Ihrer `Localizable`-Eigenschaft bestimmen können, ob das Formular lokalisiert werden soll oder nicht.

HINWEIS: Wenn Sie die folgenden Schritte nachvollziehen möchten, empfehle ich Ihnen, die Projektdateien zum »Kaputtexperimentieren« zunächst an einen anderen Ort auf Ihrer Festplatte zu kopieren. Sollte bei den Versuchen dann etwas schief gehen, können Sie das Quellprojekt einfach wieder dort rüberkopieren, und ohne Verlust von vorne beginnen.

- Öffnen Sie als Erstes die »Experimentier-Kopie« von DotNetCopy und lassen Sie den Projektmappen-Explorer mit **Strg+Alt+L** darstellen, falls dieser noch nicht angezeigt wird.
- Klicken Sie auf das entsprechende Symbol im Projektmappen-Explorer, um alle Dateien des Projektes anzeigen zu lassen. Öffnen Sie den Zweig, der sich neben dem jetzt sichtbaren Eintrag *My Project* befindet.
- Die Datei *Resources.resx*, in der sich die Ressourcen für die deutsche Kultur befinden, wird nun neben anderen angezeigt.
- Klicken Sie mit der rechte Maustaste auf diese Datei, um das Kontextmenü zu öffnen, und wählen Sie dort den Eintrag *Kopieren*.
- Im nächsten Schritt müssen Sie die IDE ein wenig austricksen: Sie müssen nun die Datei exakt an der Stelle einfügen, und damit eine Kopie der Ressource-Datei erstellen, an der die Ausgangsdatei ebenfalls platziert war. Klicken Sie allerdings wieder mit der rechten Maustaste, dann befindet sich im Kontextmenü kein Eintrag namens *Einfügen*. Aus diesem Grund klicken Sie die Datei *Resources.resx* mit der *linken* Maustaste an, um sie zu selektieren, und wählen Sie anschließend *Einfügen* aus dem Menü *Bearbeiten* der Visual Studio IDE. Eine Kopie der Datei steht anschließend ebenfalls im Zweig *My Project*.
- Nun benennen Sie die Ressource-Datei um. Das .NET-Framework wird dann später, wenn es Ihr Programm ausführen muss, anhand des Namens der Ressource-Datei wissen, welche Datei es für welche zugrunde liegende Kultur verwenden soll. Auf deutschen Systemen wird, wenn Sie auf einer deutschen Windows-Plattform entwickelt haben, immer die Ressource-Datei verwendet, mit der Sie die Anwendung entwickelt haben. Für ein englisches System (bzw. US-amerikanisches) benennen Sie die gerade erstellte Kopie der Ressource-Datei in *Resources.en.resx* um, für französisch in *Resources.fr.resx*, für italienisch in *Resources.it.resx* usw.

TIPP: Die gängigen Sprachkürzel entsprechen denen, wie sie bei der Programmierung von kulturabhängigen Format Providern verwendet werden. Sie können deswegen die entsprechende Liste in ▶ Kapitel 17 (Tabelle 17.1) verwenden.

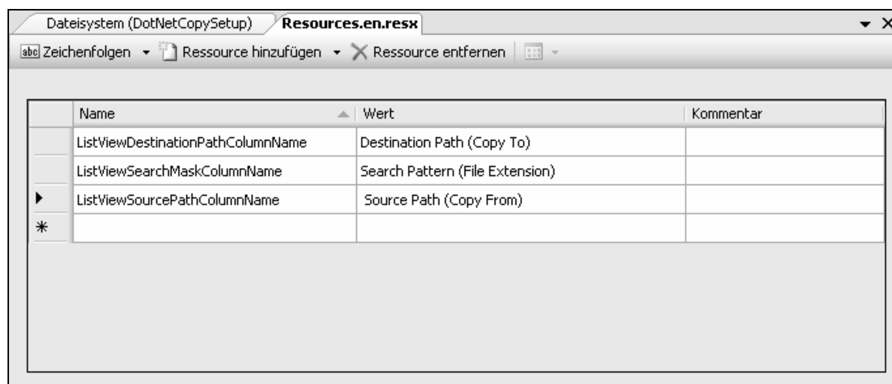

Abbildung 25.3: Wenn Sie die zusätzliche Ressource-Datei ins Projekt kopiert haben, können Sie sich an das Übersetzen der Ressourcen machen

- Wenn Sie die Datei umbenannt haben,[2] müssen Sie noch mit ein paar Handgriffen dafür sorgen, dass diese neue Ressource-Datei nicht dazu herhält, eine weitere Codedatei zu erzeugen, bei der dann, bedingt durch die schon vorhandene deutsche Version, Codeelemente für das Abrufen der Ressourcen doppelt definiert würden und so Fehler generierten. Dazu klicken Sie die umbenannte Datei an und löschen im Eigenschaftenfenster die Werte für *Benutzerdefiniertes Tool* und *Namespace des benutzerdefinierten Tools*.
- Doppelklicken Sie anschließend auf die unbenannte Datei, um die Einträge, die sie beinhaltet, zu lokalisieren – etwa wie in Abbildung 25.3 zu sehen.
- Speichern Sie die Änderungen anschließend und lassen Sie das komplette Projekt neu erstellen.

Sie werden anschließend sehen, dass im Verzeichnis der ausführbaren Dateien des Projektes (standardmäßig ist das der Pfad *Projekt\bin\Debug*) ein weiteres Unterverzeichnis hinzugekommen ist, das eine so genannte Satelliten-Assembly beinhaltet. Diese Satelliten-Assembly beinhaltet die lokalisierten Ressourcen für jede Ausgangsressource-Datei und wird vom .NET-Framework automatisch verwendet, wenn der Name des Unterordners mit dem entsprechenden Kulturkürzel der Kultur des Systems übereinstimmt, auf dem die Anwendung ausgeführt wird (*en* für Englisch, *fr* für Französisch usw.). Das bedeutet: Wenn Sie die Satelliten-Assemblies auf diese Weise erzeugt haben, dann brauchen Sie sich selbst um nichts Weiteres mehr zu kümmern – sorgen Sie lediglich dafür, dass sie in entsprechenden Unterverzeichnissen vorhanden sind.

Abbildung 25.4: Auf diesem ohne Zweifel englischen Betriebssystem wird automatisch – so vorhanden – die englische Satelliten-Assembly verwendet; in der Ressource-Datei zuvor lokalisierte Texte erscheinen dann auch in der Landessprache.

[2] Laut Aussagen des Entwicklerteams ist dieser Schritt in der finalen Version von Visual Studio 2005 nicht mehr nötig; ein Auslassen dieses Schrittes hat aber bei einigen Experimenten Probleme verursacht, und so kann es nicht schaden, diesen zusätzlichen Schritt dennoch zu erledigen.

Auf einem englischen System würde die Anwendung dann automatisch so erscheinen, wie es Abbildung 25.4 auch wiedergibt (achten Sie hier auf die Bezeichnungen der `ListView`-Spalten, denn nur diese werden beispielhaft bei DotNetCopy aus der Ressource-Datei entnommen).

Vereinfachtes Durchführen von Dateioperationen mit My.Computer.FileSystem

`FileInfo` und `DirectoryInfo` unterstützten Sie bei der Ausführung von Dateioperationen schon auf eine sehr komfortable Art und Weise. Doch bestimmte Funktionen im `My`-Namespace können das noch besser, und sie machen erst es möglich, dass unser Backup-Tool DotNetCopy trotz der vergleichsweise großen Flexibilität dennoch relativ wenig Code benötigt.

Die Funktionen, von denen DotNetCopy gerade bei der Durchführung des eigentlichen Backup-Vorgangs sehr intensiv Gebrauch macht, finden sich in `My.Computer.FileSystem`. Besonders bemerkenswert sind hier Funktionen, mit denen ganze Verzeichnisstrukturen bzw. die Namen der sich dort befindlichen Dateien rekursiv eingelesen werden können. Sie kennen das vielleicht von dem *Dir*-Befehl der Befehlszeile von Windows: Nutzen Sie diesen mit der Option */S*, werden nicht nur die Verzeichnisse bzw. Dateien des Verzeichnisses aufgelistet, das Sie angegeben haben bzw. in dem Sie sich derzeitig befinden, sondern es werden auch alle Unterverzeichnisse berücksichtigt.

Die eigentliche Kopierroutine von DotNetCopy macht von dieser `GetFiles`-Funktion Gebrauch, indem sie alle Unterverzeichnisse samt der darin enthaltenen Dateien ermittelt, zwischenspeichert und gemäß der so erstellten Liste im zweiten Schritt all diese Dateien kopiert. Der folgende Codeausschnitt demonstriert den Umgang:

```
    ''' <summary>
    ''' Die eigentliche Kopierroutine, die durch die My.Settings-Optionen gesteuert wird.
    ''' </summary>
    ''' <remarks></remarks>
    Public Sub CopyFiles()
.
.  'Aus Platzgründen gekürzt
.
        'In dieser Schleife werden zunächst alle Dateien ermittelt,
        'die es daraufhin zu untersuchen gilt, ob sie kopiert werden
        'müssen oder nicht.
        For locItemCount As Integer = 0 To lvwCopyEntries.Items.Count - 1
            Dim locCopyListEntry As CopyListEntry
            'CType ist notwendig, damit .NET weiß, welcher Typ
            'in der Tag-Eigenschaft gepspeichert war.
            locCopyListEntry = CType(lvwCopyEntries.Items(locItemCount).Tag, CopyListEntry)
            'UI aktualisieren
            lblCurrentPass.Text = "Pass 1: Kopiervorbereitung durch Zusammenstellung der Pfade..."
            lblSourceFileInfo.Text = "Unterverzeichnisse suchen in:"
            lblDestFileInfo.Text = ""
            lblProgressCaption.Text = "Fortschritt Vorbereitung:"
            lblCurrentSourcePath.Text = locCopyListEntry.SourceFolder.ToString
            LogInProtocolWindow(locCopyListEntry.SourceFolder.ToString)
            pbPrepareAndCopy.Value = locItemCount + 1
```

```
        'Windows die Möglichkeit geben, die Steuerelemente zu aktualisieren
        My.Application.DoEvents()

        'GetFiles aus My.Computer.FileSystem liefert die Namen aller
        'Dateien im Stamm- und in dessen Unterverzeichnissen.
        Dim locFiles As ReadOnlyCollection(Of String)
        Try
            locNotCaught = False
            locFiles = My.Computer.FileSystem.GetFiles( _
                    locCopyListEntry.SourceFolder.ToString, _
                    FileIO.SearchOption.SearchAllSubDirectories, _
                    locCopyListEntry.SearchMask)
        Catch ex As Exception
            LogError(ex)
            'Wenn schon beim Zusammenstellen der Dateien Fehler aufgetreten sind,
            'sollte das Verzeichnis ausgelassen werden. In diesem Fall sollte dieser
            'schwerwiegende Fakt im Anwendungsprotokoll protokolliert werden!
            My.Application.Log.WriteException(ex, TraceEventType.Error, _
                "DotNetCopy konnte das Verzeichnis " & _
                locCopyListEntry.SourceFolder.ToString & " nicht durchsuchen., " & _
                "Das Verzeichnis wurde daher nicht berüscksichtigt!")

            ' Es müsste wie "Catch" einen "NotCaught"-Zweig geben...
            locNotCaught = True
        End Try
.
.
.
```

Eine weitere Funktionalität, die ebenfalls sehr sinnvoll und praktisch ist, stellt `My.Computer.FileSystem` in Form der `CopyFile`-Methode zur Verfügung. Anders als die Methode `CopyTo` der `FileInfo`-Klasse greift diese exakt auf die gleiche Betriebssystemfunktion von Windows zurück, die auch beispielsweise der Windows-Explorer verwendet. Und das bedeutet – auch wenn DotNetCopy in diesem Beispiel davon keinen Gebrauch macht – dass Sie bei der Verwendung von `CopyFile` aus dem `My`-Namespace automatisch den bekannten Fortschrittsdialog anzeigen lassen können.

```
    ''' <summary>
    ''' Interne Kopierroutine, die den eigentlichen 'Job' des Kopierens übernimmt.
    ''' </summary>
    ''' <param name="SourceFile">Quelldatei, die kopiert werden soll.</param>
    ''' <param name="DestFile">Zielpfad, in den die Datei hineinkopiert werden soll.</param>
    ''' <remarks></remarks>
    Private Sub CopyFileInternal(ByVal SourceFile As FileInfo, ByVal DestFile As FileInfo)
        'Falls die Zieldatei noch gar nicht existiert,
        'dann in jedem Fall kopieren
        If Not DestFile.Exists Then
            Try
                'Datei kopieren, ohne Windows-Benutzeroberfläche anzuzeigen.
                My.Computer.FileSystem.CopyFile(SourceFile.ToString, DestFile.ToString)
                LogInProtocolWindow("Kopiert, OK: " & SourceFile.ToString)
            Catch ex As Exception
                LogError(ex)
            End Try
```

```
            Exit Sub
        End If
        'Datei nur kopieren, wenn Sie jünger als die zu überschreibende ist
        If My.Settings.Option_OnlyOverwriteIfOlder Then
            If SourceFile.LastWriteTime > DestFile.LastWriteTime Then
                'Wenn das Historien-Backup aktiviert ist...
                If My.Settings.Option_EnableBackupHistory Then
                    '...dieses durchführen.
                    ManageFileBackupHistory(DestFile)
                End If
                Try
                    'Datei kopieren.
                    My.Computer.FileSystem.CopyFile(SourceFile.ToString, DestFile.ToString)
                    LogInProtocolWindow("Kopiert, OK: " & SourceFile.ToString)
                Catch ex As Exception
                    LogError(ex)
                End Try
            End If
        End If
    End Sub
```

HINWEIS: Leider werden bei der Verwendung von `CopyFile` in Windows-Anwendungen, bei denen – wie es standardmäßig bei Windows Forms-Anwendungen geschieht – Formulare an eine Meldungswarteschlage gebunden werden, auch Fehlermeldungen als modale Dialoge dargestellt. Gerade bei Fehlern, die das Programm behandeln soll, kann das lästig und unerwünscht sein. Wichtig ist es aber auch zu wissen, dass bei Anwendungen, die keine Meldungswarteschlange initiieren, Fehlermeldungen nicht vom Betriebssystem in Form eines sichtbaren modalen Dialogs abgefangen werden, oder mit anderen Worten: Wenn Sie dafür sorgen, dass Ihr Programm keine sichtbare Benutzeroberfläche hat, dann erscheint auch keine sichtbare Fehlermeldung, selbst wenn bei `CopyFile` ein Fehler auftritt.

Im Falle unseres Beispiels DotNetCopy bedeutet das: Wir können im Silent-Modus keine sichtbaren Fehlermeldungen gebrauchen. Die gibt es aber auch bei `CopyFile` in diesem Modus sowieso nicht, denn es gibt keine sichtbare Benutzeroberfläche. Zwar verwenden wir die gleiche Formularklasse, die wir auch verwenden, wenn DotNetCopy nicht im Silent-Modus läuft, aber wir zeigen das Formular nicht an.

Der folgende Code, der sich im `StartUp`-Ereignis des `MyApplication`-Objektes befindet, und der ausgelöst wird, sobald die Anwendung startet, demonstriert das:

```
'Hinweis: locFrmMain wurde weiter oben im Code schon instanziert, aber nicht dargestellt!
'Silentmode bleibt nur "an", wenn AutoStart aktiv ist.
locFrmMain.SilentMode = locFrmMain.SilentMode And locFrmMain.AutoStartMode

'Und wenn Silentmode, erfolgt keine Bindung des Formulars an den Anwendungskontext!
If locFrmMain.SilentMode Then
    'Alles wird in der nicht sichtbaren Instanz des Hauptforms durchgeführt,
    locFrmMain.HandleAutoStart()
    'und bevor das "eigentliche" Programm durch das Hauptformular gestartet wird,
    'ist der ganze Zauber auch schon wieder vorbei.
    e.Cancel = True
Else
    'Im Nicht-Silent-Modus wird das Formular an die Anwendung gebunden,
```

```
            'und los geht's!
            My.Application.MainForm = locFrmMain
        End If
    End Sub
End Class
```

Verwenden von Anwendungseinstellungen mit My.Settings

DotNetCopy verwendet Anwendungeinstellungen (*ApplicationSettings*), um die Einstellungen auch nach dem Beenden des Programms zu erhalten, die der Anwender vornehmen kann, wenn er aus dem Menü *Extras* den Menüpunkt *Optionen* auswählt. Da schon ▶ Kapitel 3 sich intensiv mit Anwendungseinstellungen auseinandergesetzt hat, erfolgt an dieser Stelle lediglich eine kompakte Zusammenfassung des dazu Bekannten:

Anwendungseinstellungen pflegen Sie – ähnlich wie die Zeichenkettentabellen der Ressourcen – über eine Tabelle, die Sie über die Projekteigenschaften erreichen.

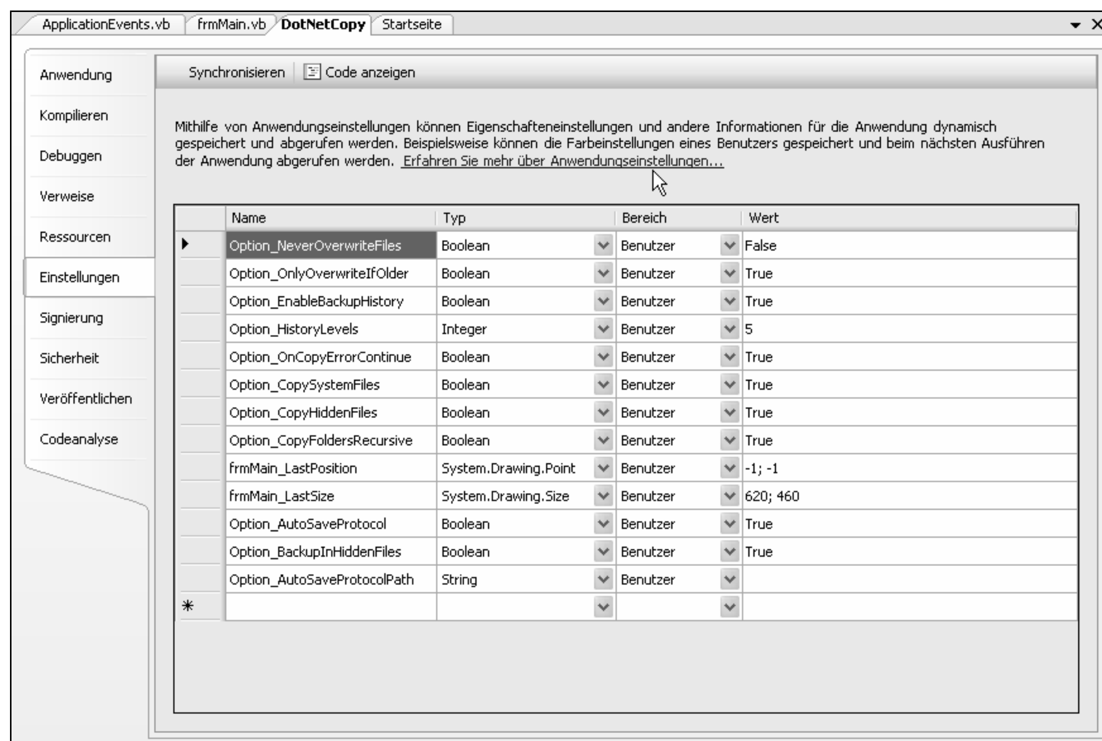

Abbildung 25.5: Anwendungseinstellungen, die Sie über *My.Settings* abrufen können, verwalten Sie wie Ressourcen in den Projekteigenschaften unter der Registerkarte *Einstellungen*

Für jeden Namen, den Sie in der Tabelle anlegen, wird dabei eine über `My.Settings` zu erreichende Objektvariable von dem Typ angelegt, den Sie in der Tabelle unter Typ bestimmt haben. Sie können für die Variablen, die Sie an dieser Stelle anlegen, aus zwei Bereichen auswählen: *Benutzer* wählen Sie, wenn Sie den Variableninhalt zur Laufzeit verändern wollen, und der neue Inhalt nach Programmende auch automatisch gesichert werden soll; *Anwendung* wählen Sie, wenn es sich um eine Konstante handeln soll, die nur Sie zur Entwurfszeit in den Projekteigenschaften (in der in Abbildung 25.5 gezeigten Tabelle) einstellen können, die man aber zur Laufzeit nicht verändern darf.

Das Abspeichern oder Auslesen von Anwendungseinstellungen ist schließlich eine Kleinigkeit. Der folgende Code zeigt, wie von Anwendungseinstellungen intensiv Gebrauch gemacht wird, wenn der Anwender im Optionsdialog Änderungen vornimmt. Die folgende Routine, die aufgerufen wird, wenn der Optionsdialog geladen wird, sorgt zunächst dafür, dass die Steuerelemente die Anwendungseinstellungen widerspiegeln:

```
Private Sub frmOptions_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles Me.Load
    'Anwendungseinstellungen im Dialog widerspiegeln lassen:
    'Auch dazu wird My.Settings verwendet.
    chkCopyFoldersRecursive.Checked = My.Settings.Option_CopyFoldersRecursive
    chkCopyHiddenFiles.Checked = My.Settings.Option_CopyHiddenFiles
    chkCopySystemFiles.Checked = My.Settings.Option_CopySystemFiles
    chkEnableBackupHistory.Checked = My.Settings.Option_EnableBackupHistory
    nudHistoryLevels.Value = My.Settings.Option_HistoryLevels
    chkNeverOverwriteFiles.Checked = My.Settings.Option_NeverOverwriteFiles
    chkOnCopyErrorContinue.Checked = My.Settings.Option_OnCopyErrorContinue
    chkOnlyOverwriteIfOlder.Checked = My.Settings.Option_OnlyOverwriteIfOlder
    chkBackupInHiddenFiles.Checked = My.Settings.Option_BackupInHiddenFiles
    chkAutoSaveProtocol.Checked = My.Settings.Option_AutoSaveProtocol
    lblProtocolPath.Text = My.Settings.Option_AutoSaveProtocolPath
End Sub
```

Beim Verlassen des Dialogs mit OK erfolgt der umgekehrte Weg: Die Werte oder Zustände der Steuerelemente werden ausgelesen und in den Anwendungseinstellungen gespeichert.

```
Private Sub btnOK_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnOK.Click
    'Die Einstellungen des Options-Dialogs in den Anwendungseinstellung speichern.
    'Auch dazu wird My.Settings verwendet.
    My.Settings.Option_CopyFoldersRecursive = chkCopyFoldersRecursive.Checked
    My.Settings.Option_CopyHiddenFiles = chkCopyHiddenFiles.Checked
    My.Settings.Option_CopySystemFiles = chkCopySystemFiles.Checked
    My.Settings.Option_EnableBackupHistory = chkEnableBackupHistory.Checked
    My.Settings.Option_HistoryLevels = CInt(nudHistoryLevels.Value)
    My.Settings.Option_NeverOverwriteFiles = chkNeverOverwriteFiles.Checked
    My.Settings.Option_OnCopyErrorContinue = chkOnCopyErrorContinue.Checked
    My.Settings.Option_OnlyOverwriteIfOlder = chkOnlyOverwriteIfOlder.Checked
    My.Settings.Option_BackupInHiddenFiles = chkBackupInHiddenFiles.Checked
    My.Settings.Option_AutoSaveProtocol = chkAutoSaveProtocol.Checked
    My.Settings.Option_AutoSaveProtocolPath = lblProtocolPath.Text

    'Das Setzen des Dialogergebnisses bei einem modal aufgerufenen Dialog
    'bewirkt gleichzeitig, dass dieser geschlossen wird!
    Me.DialogResult = Windows.Forms.DialogResult.OK
End Sub
```

Speichern von Anwendungseinstellungen mit dem Bereich Benutzer

Den Code zum Abspeichern der Anwendungseinstellungen werden Sie übrigens im Optionsdialog vergebens suchen – es gibt ihn nicht, da er im Grunde genommen nicht notwendig ist. Wenn eine Windows Forms-Anwendung beendet wird, für die sowohl das Anwendungsframework als auch die Option *Eigene Einstellungen beim Herunterfahren speichern* in den Projekteigenschaften unter Anwendung aktiviert ist (siehe auch ▶ Kapitel 26), dann regelt das Framework das Speichern der Anwendungseinstellungen für den Bereich *Benutzer* automatisch.

In einigen Fällen kann es aber sinnvoll sein, die Anwendungseinstellungen selbst oder schon vor dem Beenden des Programms zu speichern. In DotNetCopy ist das beispielsweise der Fall, wenn das Programm zum allerersten Mal gestartet wird. In diesem Fall werden die Einstellungen für die Position und die Größe des Hauptformulars festgelegt, und direkt gespeichert – und das erfolgt mit der Methode `My.Settings.Save`, wie das folgende Beispiel zeigt:

```
Private Sub frmMain_Load(ByVal sender As Object, ByVal e As System.EventArgs) _
.
. ' Aus Platzgründen ausgelassen
.
    'Fenstergröße und -position wiederherstellen, wenn
    'nicht im Silent-Modus
    If Not mySilentMode Then
        'Beim ersten Aufruf ist die Größe -1, dann bleibt die Default-
        'Position, die Settings werden aber sofort übernommen...
        If My.Settings.frmMain_LastPosition.X = -1 Then
            My.Settings.frmMain_LastPosition = Me.Location
            Me.Size = My.Settings.frmMain_LastSize
            My.Settings.Save()

            'anderenfalls werden die Settings-Einstellungen in die Formular-
            'Position übernommen
        Else
            Me.Location = My.Settings.frmMain_LastPosition
        End If
        Me.Size = My.Settings.frmMain_LastSize
    End If
```