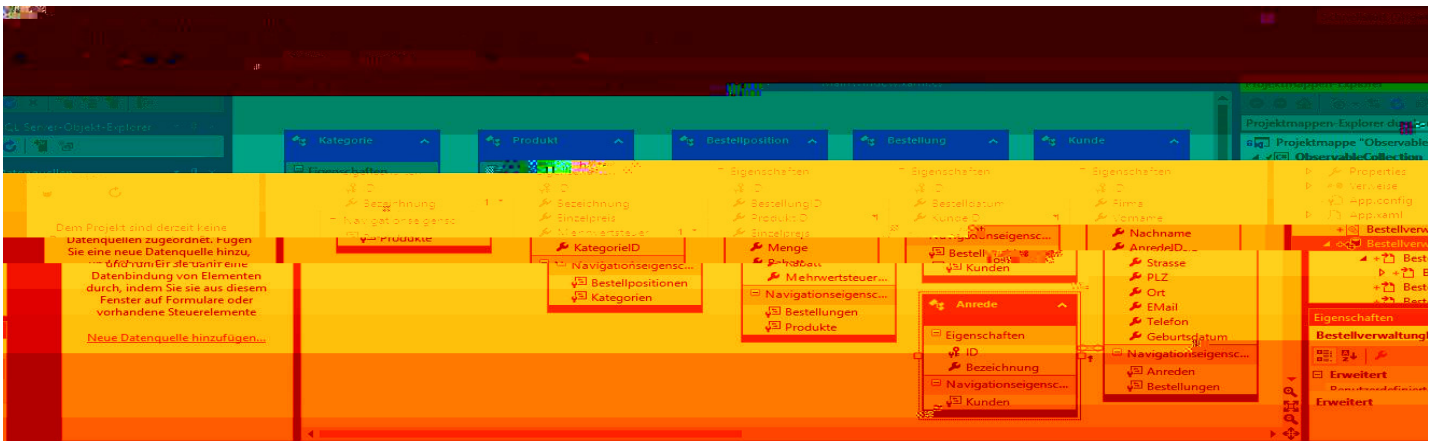
AUSGABE 06/2017

# DATENBANK ENTWICKLER

MAGAZIN FÜR DIE DATENBANKENTWICKLUNG MIT VISUAL STUDIO FÜR DESKTOP, WEB UND CO.




## TOP-THEMEN:



André Minhorst Verlag

## Impressum

DATENBANKENTWICKLER
© 2017 André Minhorst Verlag
Borkhofer Str. 17
47137 Duisburg

Redaktion: Dipl.-Ing. André Minhorst

# Visual Studio-Vorlagen erstellen

**Wer die bisherigen Artikel verfolgt und zuvor mit Access gearbeitet hat, stellt fest, dass es einige Aufgaben gibt, die sich unter Access einfacher durchführen ließen – zum Beispiel das Ausstatten eines Formulars mit einer Datenherkunft und das Hinzufügen der gebundenen Steuerelemente zu diesem Formular. Nun gibt es auch unter Visual Studio die Möglichkeit, Assistenten selbst zu programmieren. Damit könnten wir dann selbst die Helferlein erstellen, die uns bei Routineaufgaben unterstützen – wie beispielsweise das Anlegen eines Window- oder Page-Elements auf Basis einer Entität. Doch bis dahin ist es noch ein langer Weg – in diesem Artikel schauen wir uns zunächst an, wie Sie einfache Vorlagen für Visual Studio erstellen.**

## Standards erweitern

Microsoft-Produkte wie das Office-Paket inklusive Access oder auch Visual Studio sind tolle Produkte, die schon eine Menge Funktionen bieten und den Benutzer bestmöglich bei seinen Aufgaben unterstützen.

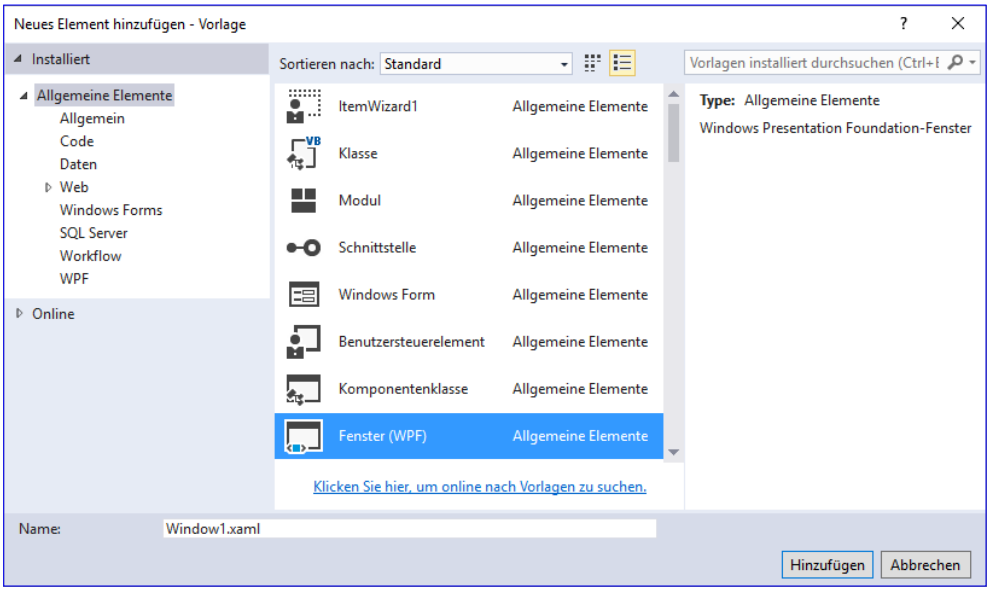



**Bild 1:** Standard-Vorlagen

Allerdings gelingt dies nicht immer so, wie man sich das vorstellt – was allerdings zum größten Teil daran liegt, dass es einfach so viele unterschiedliche Aufgaben gibt. Die Entwickler von Microsoft können einfach nicht alle denkbaren Anforderungen der Benutzer ihrer Programme vorausahnen – deshalb kommen die meisten Vorlagen in einer Minimalausstattung, die der Entwickler für seine Zwecke erweitern kann (siehe Bild 1). Das ist aber auch gar nicht weiter schlimm, denn sowohl für Office als auch für Visual Studio gibt es verschiedene Möglichkeiten, die Anwendungen um Vorlagen, benutzerdefinierte Add-Ins oder Wizards zu erweitern. Wir schauen uns in diesem Artikel an, wie Sie aus Ihren selbst gestalteten **Window**-, **Page**-, **Class**- und sonstigen Elementen Vorlagen erstellen können, die Sie dann beim Anlegen neuer Elemente nutzen können.

## Beispiele für Vorlagen

Wenn Sie mit WPF arbeiten und Windows-Anwendungen entwickeln, haben Sie vielleicht bestimmte Einstellungen, die für alle Steuerelemente verwendet werden sollen, wenn diese nicht gerade im Steuerelement selbst überschrieben werden. Solche Eigenschaften legt man beispielsweise wie folgt in einem Element namens **Window.Resources** an:

```
<Window.Resources>
    <Style TargetType="Button">
        <Setter Property="Margin" Value="3"></Setter>
        <Setter Property="Padding" Value="5"></Setter>
        <Setter Property="VerticalAlignment" Value="Top"></Setter>
        <Setter Property="HorizontalAlignment" Value="Left"></Setter>
    </Style>
</Window.Resources>
```

Wenn Sie solche Einstellungen immer wieder verwenden, müssen Sie diese jeweils in neue **Window**- oder **Page**-Element integrieren. Sie können aber auch eine Vorlage erstellen, der Sie die gewünschten Elemente hinzufügen, und diese dann bei Bedarf statt des herkömmlichen **Window**-Elements zum Projekt hinzufügen.

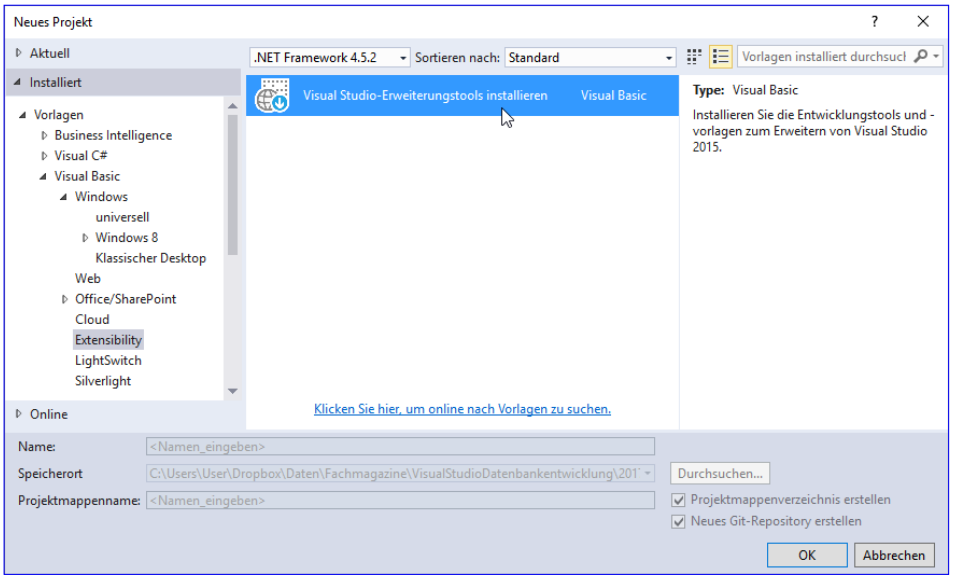



**Bild 2:** Erweiterungsvorlagen hinzufügen

## Vorbereitungen

Wie aber legen wir eine solche Vorlage an und welche Einstellungen sind nötig, damit die Vorlage dann auch beim Anlegen neuer Elemente im Dialog **Neues Element hinzufügen** angezeigt wird? Dazu ist etwas Vorarbeit erforderlich:

- Starten Sie Visual Studio.
- Öffnen Sie den Dialog **Neues Projekt**.
- Wechseln Sie zum Eintrag **Visual C#IExtensibility** oder **Visual BasicIExtensibility**.
- Klicken Sie auf den einzigen Eintrag **Visual Studio Erweiterungstools installieren** (siehe Bild 2).
- Dies startet die Installation (siehe Bild 3).

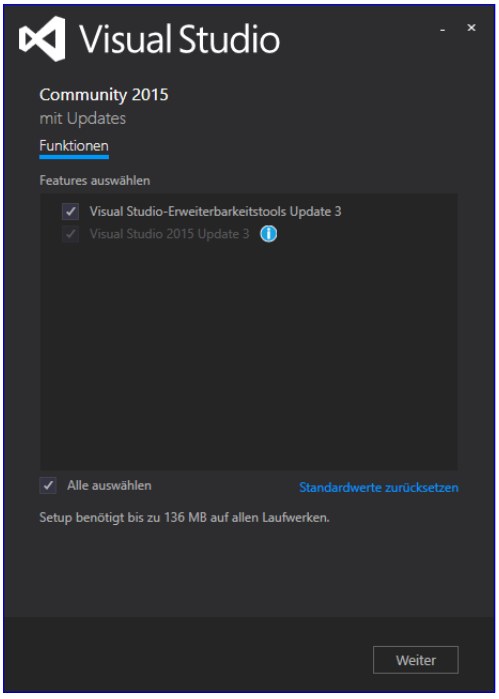



**Bild 3:** Installation der Erweiterungstools

Wenn Sie, gegebenenfalls nach einem Neustart, nun wieder ein neues Projekt anlegen wollen und dazu den Dialog **Neues Projekt** öffnen, fin-

den Sie im Bereich **Visual C#IExtensibility** oder **Visual BasicIExtensibility** die Einträge aus Bild 4 vor. Uns interessieren hier zunächst die folgenden beiden Einträge:

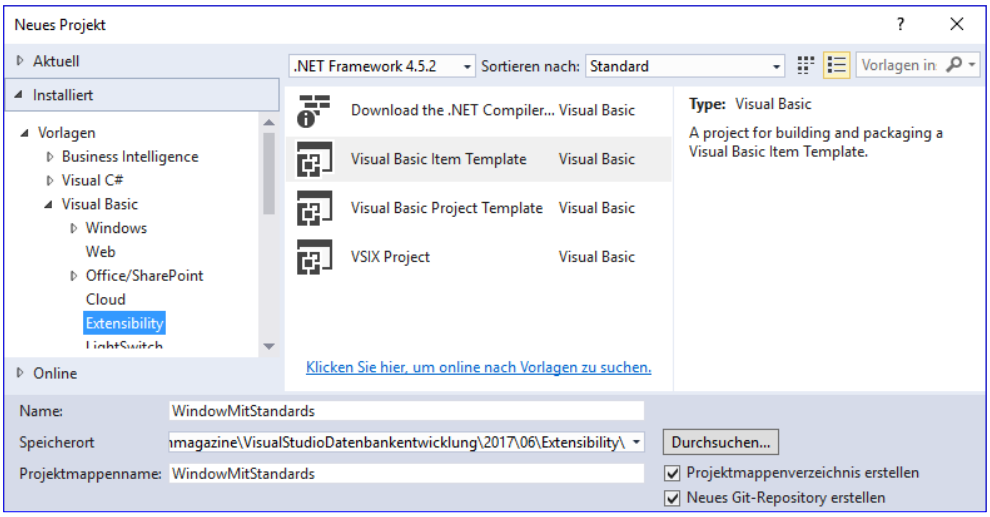



**Bild 4:** Erweiterungsvorlagen

- **Visual Basic Project Template**: Vorlage für einen Assistenten zum Anlegen neuer Projekte
- **Visual Basic Item Template**: Vorlage für einen Assistenten zum Anlegen eines neuen Elements in Visual Studio, also etwa ein **Class**-, ein **Window**- oder eine **Page**-Element

Wir wollen ein neues Element anlegen, vollständige Projektvorlagen benötigen wir aktuell noch nicht. Also geben wir den Namen des neuen Projekts ein (**WindowMitStandards**), wählen das gewünschte Zielverzeichnis aus und klicken auf die Schaltfläche **OK**.

Das neue Projekt führt sich mit der Anzeige der Datei **ItemWizard.vstemplate** ein (siehe Bild 5). Diese enthält die Konfiguration der frisch erstellten Vorlage. Des Weiteren finden wir im Projektmappen-Explorer einige weitere Dateien:

- **Class.vb**: Bisher einzige Klasse des Projekts.
- **ItemWizard.ico**: Icon-Datei für die Vorlage

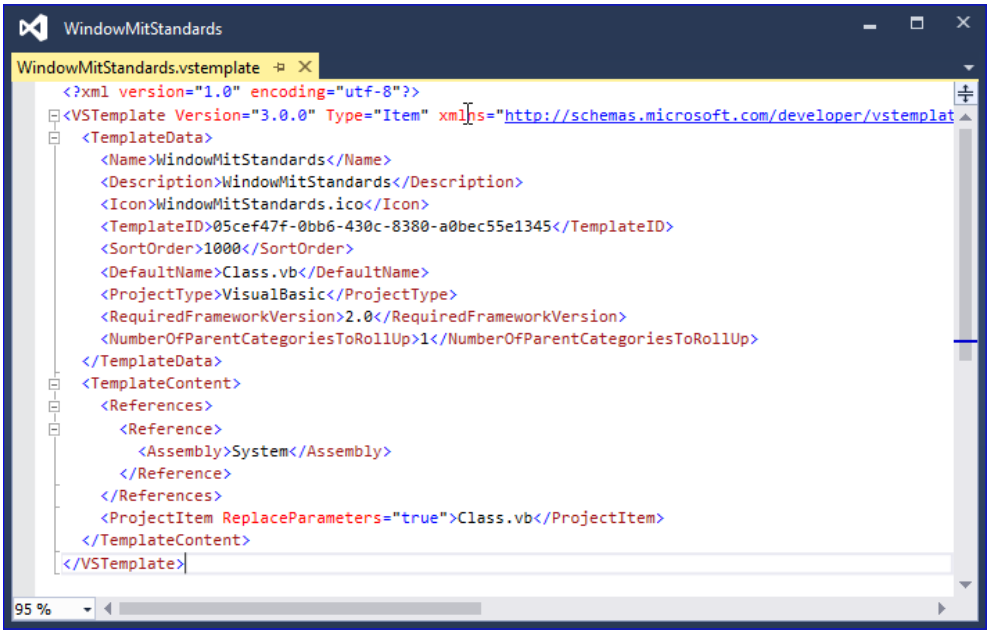



**Bild 5:** Die Konfigurationsdatei des neuen Projekts

Die Datei **Class.vb** enthält den folgenden Code:

```
Public Class $safeitemname$

End Class
```

Dies lässt bereits erahnen, dass es eine Reihe von Platzhaltern gibt, die wir in die Elemente der Vorlage in-

tegrieren können – wir schauen uns diese weiter unten an. Wozu benötigen wir diese? Nun, wenn Sie beispielsweise eine neue Klasse erstellen, geben Sie im Dialog **Neues Element hinzufügen** ja den Namen des neuen Elements ein. Dieser wird dann für einen der Platzhalter im Code der **.xaml**- oder **.xaml.cs**/**.xaml.vb**-Datei eingetragen.

## Vorgehensweise

Nachdem wir das Vorlagen-Projekt erstellt und gesichtet haben, stellt sich die Frage, welche Aufgaben nun vor uns liegen. Diese scheinen wie folgt zu lauten:

- Anlegen des Elements/der Elemente und Versehen des Codes mit Platzhaltern
- Anpassen der Konfigurationsdatei
- Exportieren der Vorlage

Warum »scheinen wie folgt zu lauten«? Ganz einfach: Weil die Konfigurationsdatei beim Exportieren des gewünschten Elements als Vorlage überhaupt nicht berücksichtigt wird. Visual Studio stellt in den nachfolgend beschriebenen Schritten selbst eine Konfigurationsdatei auf Basis der gemachten Angaben zusammen. Sie brauchen sich also nicht die Mühe zu machen, diese direkt in Visual Studio zu manipulieren. Das erledigen wir später, wenn nötig, an anderer Stelle.

## Test des Exports

Zu Testzwecken wollen wir einfach die Klasse **Class.vb** übernehmen und dieser als einzige Anpassung einen Kommentar hinzufügen:

```
Public Class $safeitemname$
    'Tolle neue Vorlagenklasse
End Class
```

Den Inhalt der Datei **WindowsMitStandards.vstemplate** wollen wir vorerst beibehalten. Nun folgt schon der entscheidende Schritt: Sie wählen aus dem Menü den Eintrag **Datei\Vorlage exportieren...** aus. Es erscheint der Dialog aus Bild 6, in dem Sie den zweiten Eintrag **Elementvorlage auswählen** und das Projekt **WindowsMitStandards** als Vorlagenquelle beibehalten.

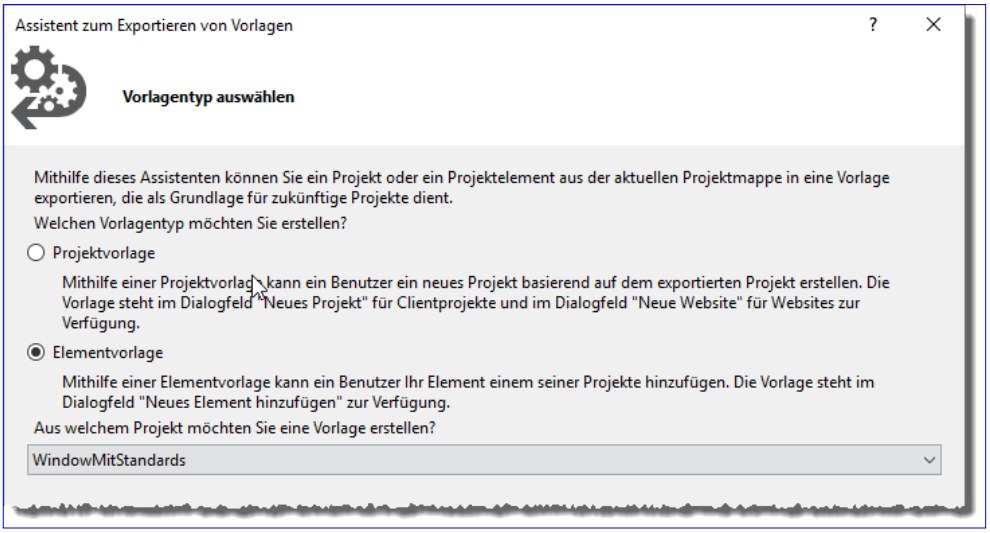



**Bild 6:** Export als Elementvorlage

Der nächste Schritt des Assistenten fragt, welches Element als Vorlagen-Element verwendet werden soll. Die Auswahl ist einfach, zumal Sie nur einen Eintrag auswählen können – nämlich die Klasse **Class.vb** (siehe Bild 7).

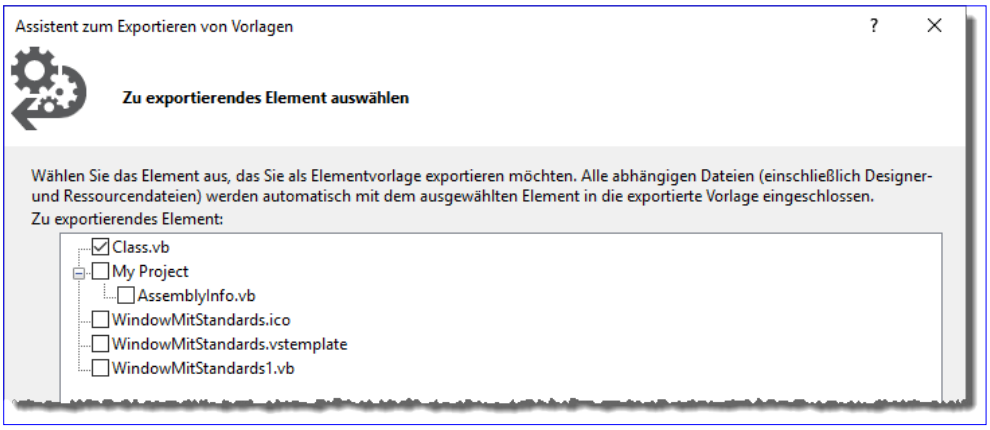



**Bild 7:** Festlegen des als Vorlage zu verwendenden Elements

Danach erscheint ein Dialog, mit dem Sie die Verweise auswählen können, die gegebenenfalls speziell für das Element benötigt werden. Der Dialog zeigt alle im Vorlagen-Projekt referenzierten Verweise an. Wenn Sie also nachträglich noch Verweise zum Projekt hinzufügen wie etwa den Verweis auf die Bibliothek **Microsoft.Office.Interop.Word**, dann wird auch dieser zur Auswahl angeboten (siehe Bild 8).

Wenn Sie wie in diesem Beispiel einen Verweis auswählen, der nicht mit Visual Studio vorinstalliert wird, weist der Dialog auf eventuelle Probleme hin – in diesem Fall, weil Microsoft Word ja vielleicht gar nicht auf dem Zielrechner der Vorlage installiert ist (bei genauerem Hinsehen liefert der Dialog diesen Hinweis auch für alle standardmäßig enthaltenen Verweise).

Im folgenden und letzten Schritt des Assistenten geben Sie den Namen der anzulegenden Vorlage und einige weitere Informationen an (siehe Bild 9).

Zum Abschluss öffnet Visual Studio dann noch den Ordner, in dem die Vorlagendatei angelegt wurde. Hier können Sie dann auch gleich das entsprechende Verzeichnis entnehmen – in diesem Fall beispielsweise **C:\Users\<Benutzername>\Documents\Visual Studio 2015\My Exported Templates**.

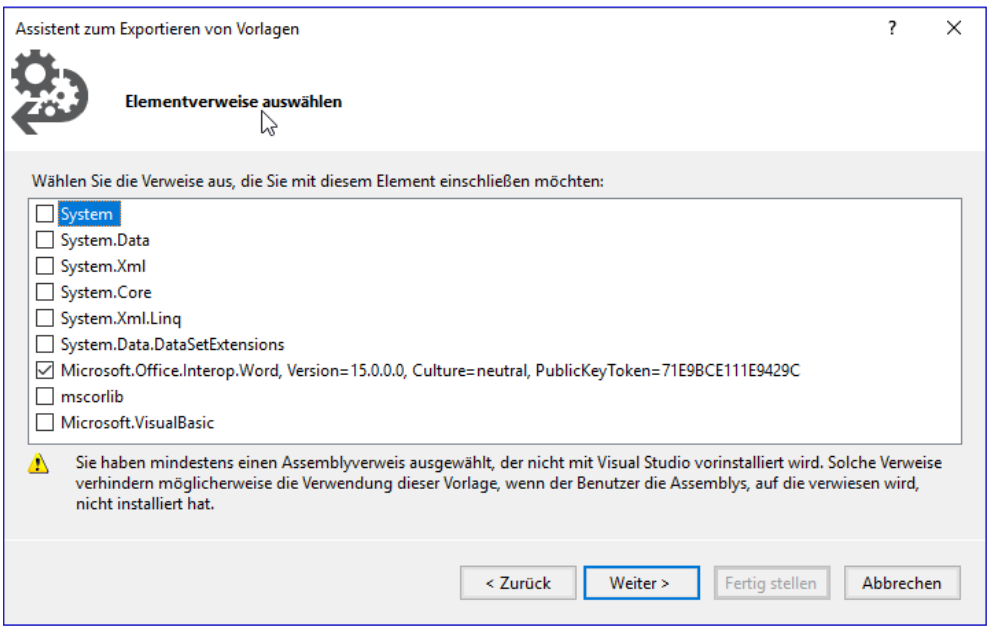



**Bild 8:** Auswählen der Projektverweise

## Neue Vorlage ausprobieren

Schließlich finden Sie, spätestens nach einem

Neustart von Visual Studio, die neue Vorlage im Dialog **Neues Element hinzufügen** vor (siehe Bild 10).

Da wir in der Konfigurationsdatei keine Änderungen vorgenommen haben, verwendet die Vorlage das vom Assistenten mitgelieferte Bild als Icon für die Vorlage. Der Name wurde ebenso übernommen wir der Beschreibungstext.

Klicken Sie nun auf **Hinzufügen**, legt Visual Studio die Klasse unter der angegebenen Bezeichnung an.

Die Klasse wird nun erstellt, allerdings enthält sie nicht den Code, den wir erwartet haben.

Stattdessen wird nicht nur der Platzhalter **$safeitemname$** durch den neuen Klassennamen ersetzt, sondern auch die Bezeichnung **Class**:

```
Public WindowMitStandards1
WindowMitStandards1
    'Tolle neue Vorlagen-
klasse
End WindowMitStandards1
```

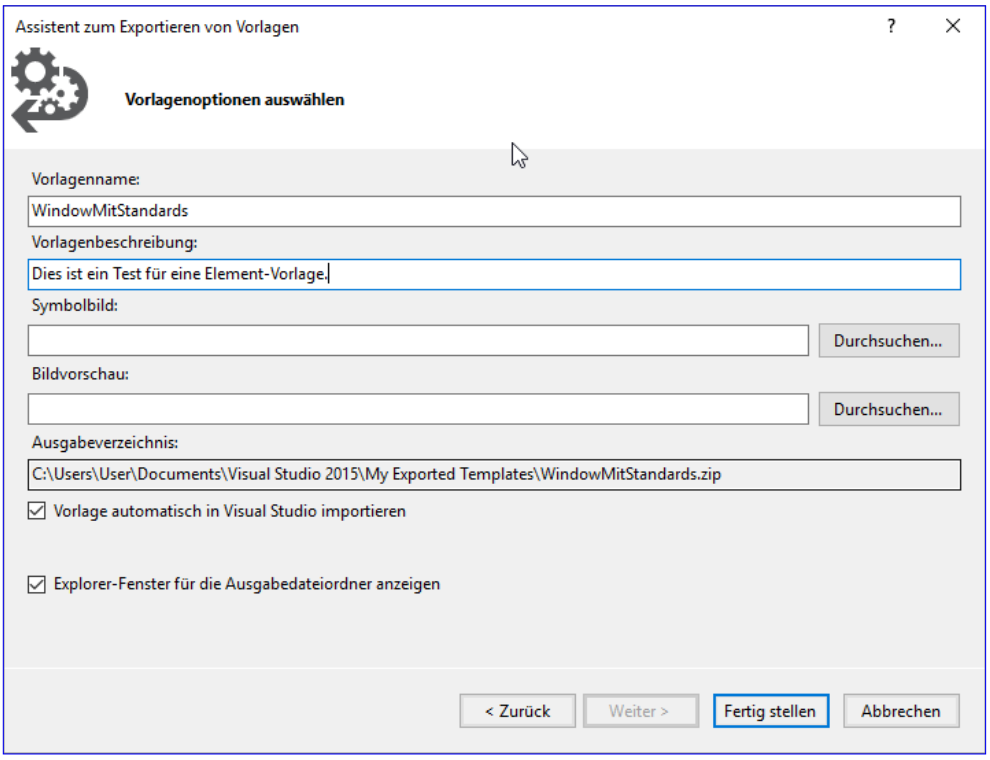



**Bild 9:** Abschließender Dialog

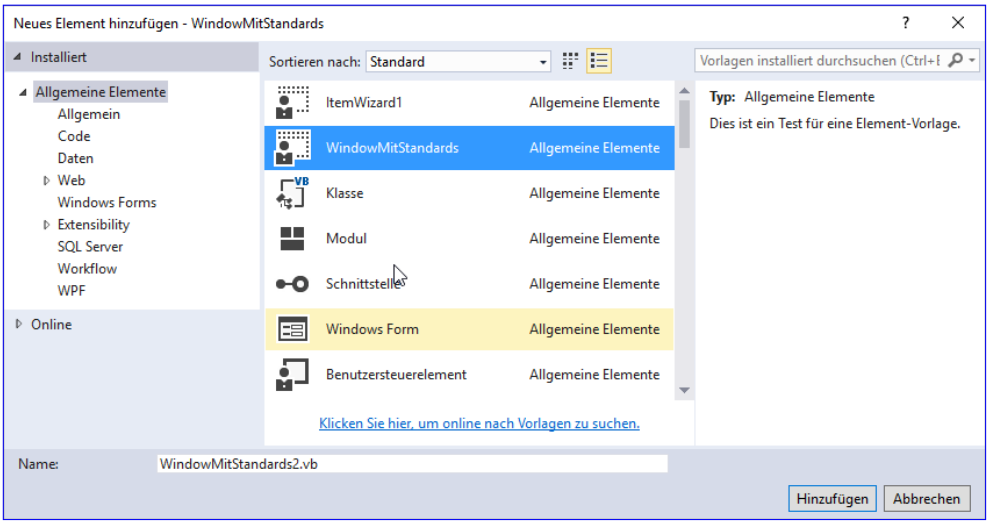



**Bild 10:** Die neue Element-Vorlage ist bereits verfügbar.

Warum geschieht dies? Dazu werfen wir einen Blick in die Vorlagendatei mit der Dateiendung **.zip** im Vorlagenverzeichnis. Diese enthält die drei Dateien aus Bild 11.

Wenn wir die Datei **Class.vb** öffnen, erhalten wir folgenden Code:

# Grid im Griff

**Das Grid-Element ist ein flexibles Steuerelement zum Platzieren der enthaltenen Steuerelemente. Seine Definition erfolgt entweder über XAML oder über den Entwurf der jeweiligen .xaml-Seite. Wie Sie das Grid-Element zur Anordnung von Steuerelementen mit XAML definieren, haben wir uns zum Teil schon im Artikel Steuerelemente anordnen angesehen. Im vorliegenden Artikel zeigen wir nun, wie Sie das Grid schnell mit der Maus so definieren, wie Sie es sonst per XAML tun – und sich damit eine Menge Zeit und Mühe sparen können.**

## Standardaufgaben vereinfachen

Wenn Sie Steuerelemente auf einem WPF-Fenster anordnen wollen, können Sie diese natürlich einfach so aus der Toolbox auf das Fenster ziehen und mit den absoluten Werten arbeiten, die dann automatisch für Position und Größe der Steuerelemente angelegt werden. Sie können diese allerdings auch in Elementen wie dem **Grid**, dem **StackPanel**, dem **WrapPanel**, dem **DockPanel** oder anderen Elementen platzieren. Wenn Sie die Steuerelemente alle direkt auf der Seite platzieren, wird es kompliziert, wenn es um verschiedene Bildschirmauflösungen oder in der Größe variable Fenster geht. Mit dem **Grid**-Element geht das alles viel besser: Sie können dann die Elemente in das Grid einfügen und durch Einstellen von Eigenschaften dafür sorgen, dass sich die Größe etwa eines Bezeichnungsfeldes, Textfeldes oder einer Schaltfläche in Abhängigkeit der jeweiligen Zelle im Grid anpasst. Erstmal wollen wir uns jedoch anschauen, wie dem Grid Zeilen und Spalten mit der Maus hinzufügen, also durch direkte Manipulation der XAML-Entwurfsansicht.

## Grid-Element

Das **Grid**-Element ist ja bei jedem neu angelegten **Window**- oder **Page**-Element automatisch vorhanden und wird im XAML-Code wie folgt definiert:

```
<Grid></Grid>
```

Im Entwurf fällt es auch deshalb kaum auf, weil seine Größe genau mit der des Fensters übereinstimmt. Allein wenn Sie es im Code markieren, wird es durch die Verankerungen sichtbar (siehe Bild 1).

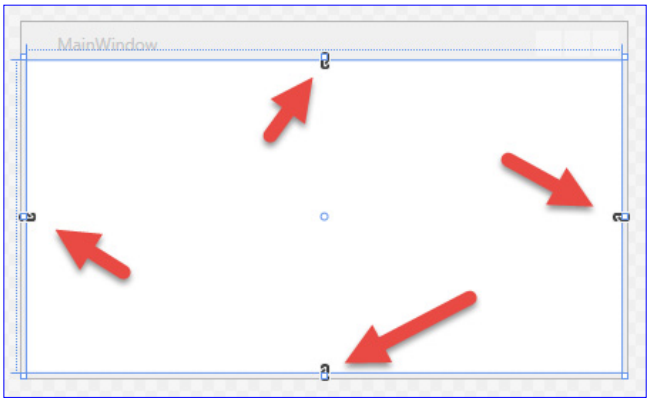



**Bild 1:** Einfaches Grid, standardmäßig an allen Seiten verankert

Sollten Sie ein Grid einmal versehentlich aus dem übergeordneten Element löschen, können Sie es einfach aus der Toolbox in das Zielelement ziehen – dann wird im Code es allerdings mit absoluten Eigenschaften versehen, etwa so – und sieht im Entwurf wie in Bild 2 aus:

```
<Grid HorizontalAlignment="Left" Height="100" Margin="20,20,0,0" VerticalAlignment="Top" Width="200"/>
```

Wenn das Grid sich über das komplette Fenster erstrecken soll und bei Größenänderungen ebenfalls angepasst werden soll, benötigen Sie all diese Eigenschaften jedoch nicht. Bevor Sie die unnötigen Attribute aus dem Element entfernt haben, haben

Sie jedoch schneller ein neues, leeres Element mit der Definition **<Grid></Grid>** angelegt.

## Anlegen von Spalten

Für eine bessere Sichtbarkeit der nachfolgend vorgenommenen Änderungen verwenden wir jedoch weiterhin das mitten im Fenster angelegte **Grid**-Element. Um dem **Grid**-Element eine neue Spalte hinzuzufügen, bewegen Sie den Mauszeiger nach dem Markieren des **Grid**-Elements (was am einfachsten durch Setzen der Einfügemarke in den Code des **Grid**-Elements gelingt) oben an die gestrichelte Linie, bis eine vertikale Linie wie in Bild 3 erscheint.

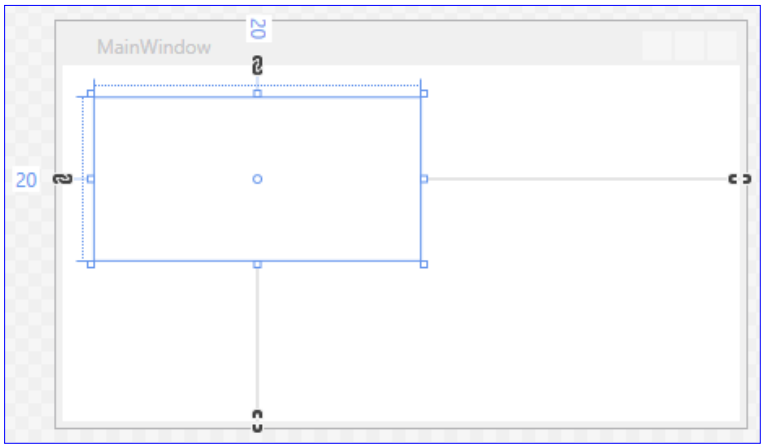



**Bild 2:** Ein aus der Toolbox hinzugefügtes Grid-Element

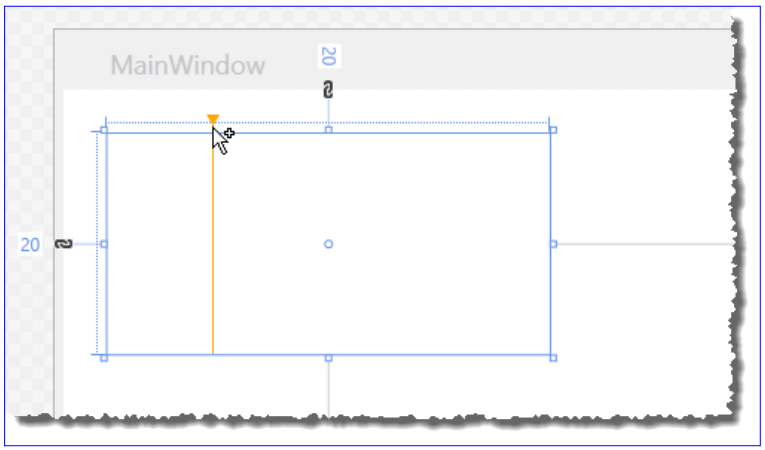



**Bild 3:** Hinzufügen einer Spalte zum **Grid**-Element

Bewegen Sie die Maus an die gewünschte Stelle und klicken Sie, wenn Sie diese erreicht haben. Die neue Spalte wird dann wie in Bild 4 angelegt. Der XAML-Code wird dadurch merklich erweitert, nämlich um die Definition der entsprechenden Spalten. Durch das Hinzufügen einer Spalte haben Sie nun zwei Spalten, denn eine Spalte war ja implizit bereits vorher vorhanden. Nun haben wir zwei durch **ColumnDefinition**-Elemente definierte Spalten, für die Visual Studio gleich die entsprechenden Breiten eingetragen hat:

```
<Grid ...>
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="41*"/>
        <ColumnDefinition Width="159*"/>
    </Grid.ColumnDefinitions>
</Grid>
```

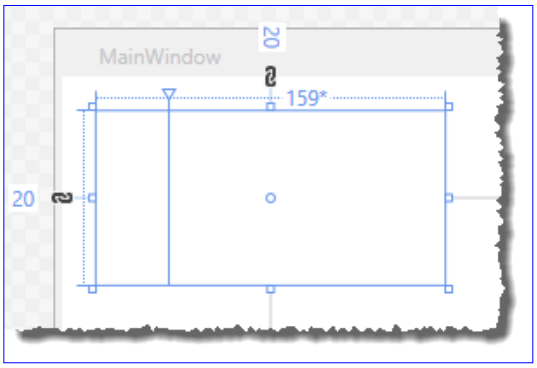



**Bild 4:** **Grid**-Element mit der neuen Spalte

Wenn Sie weitere vertikale Linien auf die oben gezeigte Art hinzufügen, werden die Elemente unter **Grid.ColumnDefinitions** entsprechend erweitert. Was bedeutet **41*** und **159*** doch gleich? Das ist das Verhältnis, in dem die Breiten der Spalten zueinanderstehen. Die Gesamtbreite entspricht hier 200*, also nimmt die linke Spalte 41/200 und die rechte 159/200 der Breite de **Grid**-Elements ein. Wenn Sie dies einfach anpassen wollen, bewegen Sie den Mauszeiger oben auf das nach unten zeigende Dreieck zwischen den beiden Spalten und ziehen

Sie dieses wie in Bild 5 nach links oder rechts. Halten Sie dabei die **Umschalt**-Taste gedrückt, ändert sich auch gleich noch die Größe des dem Grid übergeordneten Elements, hier dem **Window**-Element, mit.

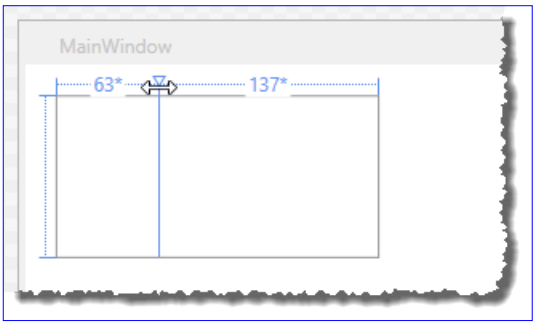



**Bild 5:** Ändern des Spaltenbreiten-Verhältnisses

Neben der hier beschriebenen Syntax mit der Zahl und dem folgenden Sternchen (*) wie **50*** gibt es natürlich noch andere Werte, die Sie gelegentlich verwenden wollen – zum Beispiel das Sternchen ohne führende Zahl oder der Wert **Auto**. Diese können Sie ebenfalls über die Entwurfsansicht manipulieren. Dazu fahren Sie mit der Maus wie oben beschrieben so in den oberen Bereich einer Spalte, als ob Sie eine neue Spalte anlegen wollen. Es erscheint dann eine Art Kombinationsfeld, das nach dem Anklicken der Schaltfläche am rechten Rand die Optionen wie in Bild 6 anzeigt.

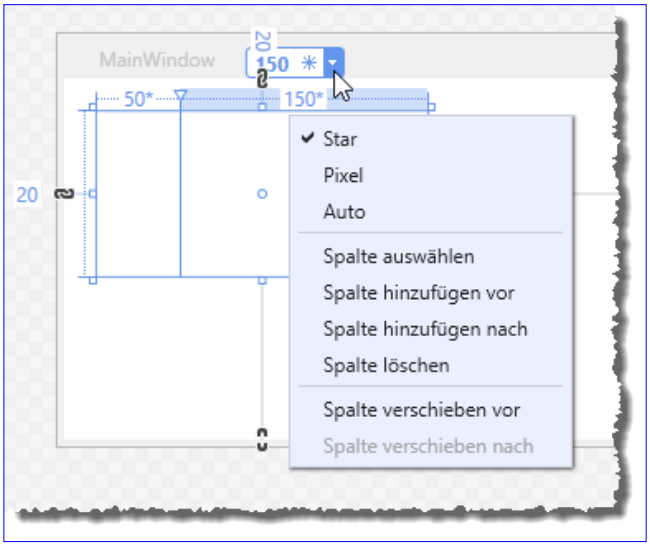



**Bild 6:** Spalten-Optionen

- **Star**: Entspricht der Standardeinstellung.
- **Pixel**: Trägt die aktuelle Breite in Pixeln für die Spaltenbreite ein.
- **Auto**: Trägt den Wert Auto für die Eigenschaft **Width** ein. Wenn keine Steuerelemente enthalten sind, führt dies zur Anzeige der Spalte mit der Breite **0**.

Die übrigen Einträge erlauben das Auswählen der gewählten Spalte, das Hinzufügen einer Spalte vor oder nach der gewählten Spalte oder das Löschen der aktuellen Spalte. Der Eintrag **Spalte verschieben vor** tauscht die Position der Spalte mit der links davon befindlichen Spalte, der Eintrag **Spalte verschieben nach** tauscht mit der rechts davon liegenden Spalte.

Sie können die Maus allerdings auch über die Zahl oder das Symbol im Kombinationsfeld fahren und dann auf eines der beiden Elemente klicken. Mit einem Klick auf die Zahl können Sie den Zahlenwert ändern, mit einem Klick auf das Symbol wechseln Sie durch die unterschiedlichen Werte, nämlich Sternchen (*), Pixelgröße (dargestellt durch ein Schloss) und **Auto** (siehe Bild 7).

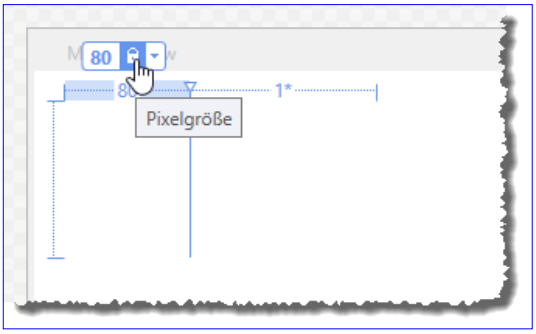



**Bild 7:** Direktes Manipulieren der Breite im Entwurf

## Zeilen hinzufügen

Das Hinzufügen von Zeilen gelingt analog zu den Spalten – Sie müssen nur an die entsprechende Stelle an der vertikalen gestrichelten Linie des **Grid**-Elements klicken. Dadurch fügen Sie dann zunächst eine neue Zeile hinzu, was wie in Bild 8 aussieht. Der XAML-Code des **Grid**-Elements wird dadurch wiederum stark erweitert, und zwar um ein **Grid.RowDefinitions**-Element und für jede Zeile ein **RowDefinition**-Element:

# Visual Basic: Bedingungen und Schleifen

**Wer bisher mit VBA oder C# gearbeitet hat und zu VB wechseln möchte, sieht sich bei der Entwicklung von WPF-Anwendungen einigen Änderungen gegenüber. Dieser Artikel liefert die Grundlagen zur Programmierung von Bedingungen wie If...Then oder Select Case und von Schleifen wie For...Next, For...Each oder Do While. Zum Experimentieren mit den Beispielen nutzen wir das Tool LINQPad 5.**

Wer von VBA kommt, wird wenig Probleme haben, sich in Visual Basic für .NET einzuarbeiten. Wer nur C# kennt oder sich daran gewöhnt hat, muss sich erst ein wenig einarbeiten. Hier kommen nämlich die wichtigsten Konstrukte bei der Programmierung von Methoden und Funktionen: Bedingungen und Schleifen. Die Beispiele dieses Artikels haben wir der Einfachheit halber wieder im Tool LINQPad5 ausprobiert, das wir im Artikel **LINQPad: LINQ, C# und VB einfach ausprobieren** bereits vorgestellt haben.

## Einfaches If...Then

Eine einfache **If...Then**-Bedingung prüft, ob der angegebene Ausdruck wahr ist und führt dann die zwischen der **If**- und der **End If**-Zeile enthaltenen Anweisungen aus:

```
Dim intZahl As Integer
intZahl = 10
If intZahl > 5 Then
    Debug.WriteLine("intZahl ist größer als 5")
End If
```

Von C# kommend ist das eine Umstellung, denn auch hier fallen die geschweiften Klammern weg und die Semikola am Ende der Befehlszeilen. Auch ist es hier nicht mehr möglich, eine einfache Zeile einfach ohne geschweifte Klammern hinter der **If**-Zeile einzufügen. Es gibt aber auch unter VB eine vereinfachte Schreibweise für einfache Anweisungen. Hier lassen Sie dann einfach die Zeile **End If** weg und fügen die auszuführende Zeile einfach an die Bedingung an:

```
If intZahl > 5 Then Debug.WriteLine("intZahl ist größer als 5")
```

## If...Then...Else-Bedingung

Nehmen wir noch einen **Else**-Zweig hinzu, um Anweisungen anzugeben, die bei Nichterfüllung der Bedingung ausgeführt werden sollen. Dann sieht die Bedingung wie folgt aus:

```
If intZahl > 5 Then
    Debug.WriteLine("intZahl ist größer als 5")
Else
    Debug.WriteLine("intZahl ist nicht größer als 5")
End If
```

Wenn der erste Teil der Bedingung hier nicht eintritt, wird auf jeden Fall der zweite Teil hinter dem Schlüsselwort **Else** ausgeführt. Sie können statt **Else** auch das Schlüsselwort **ElseIf** verwenden und eine zweite Bedingung angeben. Im folgenden Beispiel müssen wir allerdings beachten, dass im Falle von **intZahl = 5** keiner der beiden Zweige ausgeführt wird:

```
If intZahl > 5 Then
    Debug.WriteLine("intZahl ist größer als 5")
ElseIf intZahl < 5 Then
    Debug.WriteLine("intZahl ist nicht größer als 5")
End If
```

Für diesen Fall können wir aber auch noch einen finalen **Else**-Zweig anhängen, der auf jeden Fall ausgeführt wird. Der **Else**-Zweig kann allerdings nur als letzte Bedingung verwendet werden:

```
If intZahl > 5 Then
    Debug.WriteLine("intZahl ist größer als 5")
ElseIf intZahl < 5 Then
    Debug.WriteLine("intZahl ist nicht größer als 5")
Else
    Debug.WriteLine("Keine der angegebenen Bedinungen traf zu.")
End If
```

Sie können mehrere **ElseIf**-Zweige einsetzen. Das ist ein großer Unterschied zu C#, denn dort gibt es gar kein **ElseIf** (nur eine vereinfachte Schreibweise für verschachtelte **If**-Bedingungen mit **Else If**). Hier müssen Sie für solche Fälle mit verschachtelten Bedingungen arbeiten, wenn Sie **If...Then** nutzen wollen.

## Verschachtelte If...Then-Bedingung

**If...Then**-Bedingungen lassen sich ineinander verschachteln:

```
If intZahl > 5 Then
    Debug.WriteLine("intZahl ist größer als 5")
ElseIf
    If intZahl > 3 Then
        Debug.WriteLine("intZahl ist größer als 3 und kleiner als 5")
    Else
        Debug.WriteLine("intZahl ist kleiner gleich 3")
    End If
End If
```

## Verwendung von If...Then in einer Zeile mit IIf

Die von VBA bekannte Funktion **IIf** können Sie unter C# nicht verwenden, es gibt dort eine alternative Schreibweise mit gleicher Funktion. Unter VB jedoch steht **IIf** wie unter VBA zur Verfügung. Dabei erwartet die Funktion drei Parameter:

- Bedingung
- Wert, wenn die Bedingung wahr ist
- Wert, wenn die Bedingung falsch ist

Die obige **If...Then...Else**-Bedingung würde in einer Zeile wie folgt aussehen:

```
IIf(intZahl>5, "Zahl ist größer als 5", "Zahl ist kleiner gleich 5")
```

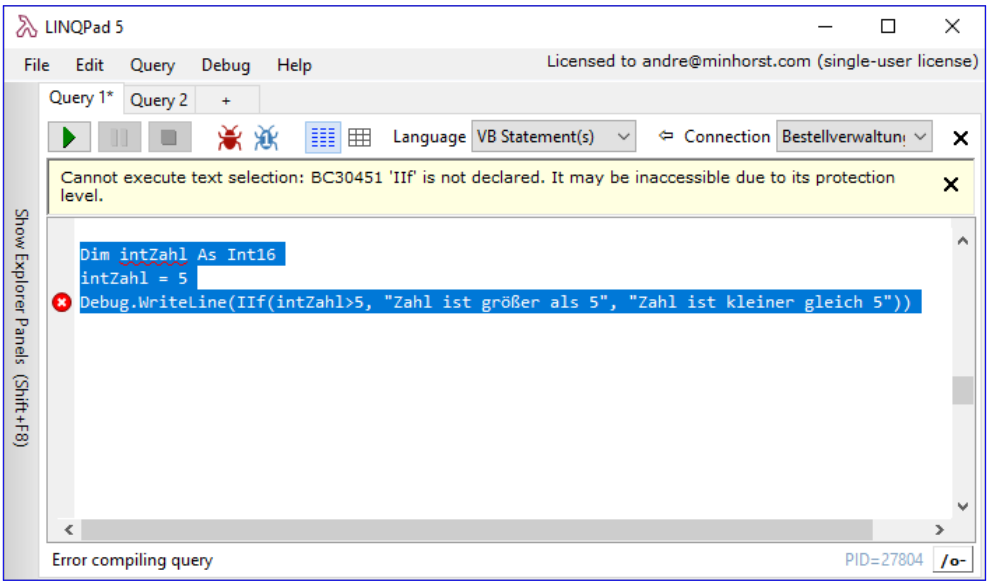



**Bild 1:** Fehler beim Ausführen der Funktion **IIf**

Hinweis: Wenn Sie **IIf** und anderen Visual Basic-Funktionen in LINQPad testen wollen, erhalten Sie normalerweise den Fehler aus Bild 1.

Um dies zu verhindern, müssen Sie den Namespace **Microsoft.VisualBasic** hinzufügen.

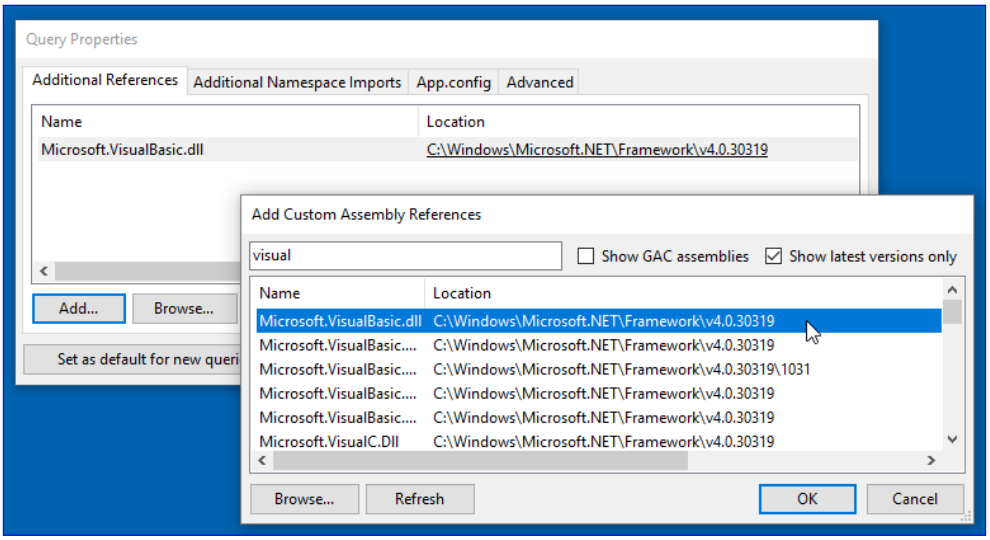



**Bild 2:** Hinzufügen der Referenz auf die Visual Basic-Bibliothek

Das erledigen Sie über einen Dialog, den Sie mit dem Menübefehl **Query|Namespace Imports** öffnen. Der Dialog heißt **Query Properties**. Im Bereich **Additional Reference** klicken Sie auf die Schaltfläche **Add**, was den Dialog **Add Custom Assembly Reference** öffnet. Hier geben Sie im Suchfeld **VisualBasic** ein und fügen den Eintrag wie in Bild 2 mit einem Klick auf die **OK**-Schaltfläche hinzu.

Nun fehlt nur noch ein Schritt: Zurück im Dialog **Query Properties** wechseln Sie zum Bereich **Additional Namespace Imports** und geben dort wie in Bild 3 gezeigt **Microsoft.VisualBasic** ein. Da wir zuvor einen Verweis auf die entsprechende Bibliothek hinzugefügt haben, können Sie in der Liste oben rechts den Eintrag **Microsoft.VisualBasic.dll** auswählen.

Darunter erscheinen dann alle verfügbaren Namespaces, von denen Sie **Microsoft.VisualBasic** auswählen und per Klick auf die Schaltfläche **< Add Selected Namespaces** zur Liste im rechten Bereich hinzufügen.

Danach können Sie **IIf** nicht nur ohne Fehlermeldung nutzen, sondern **IIf** und die übrigen Elemente des Namespaces werden per IntelliSense angezeigt.

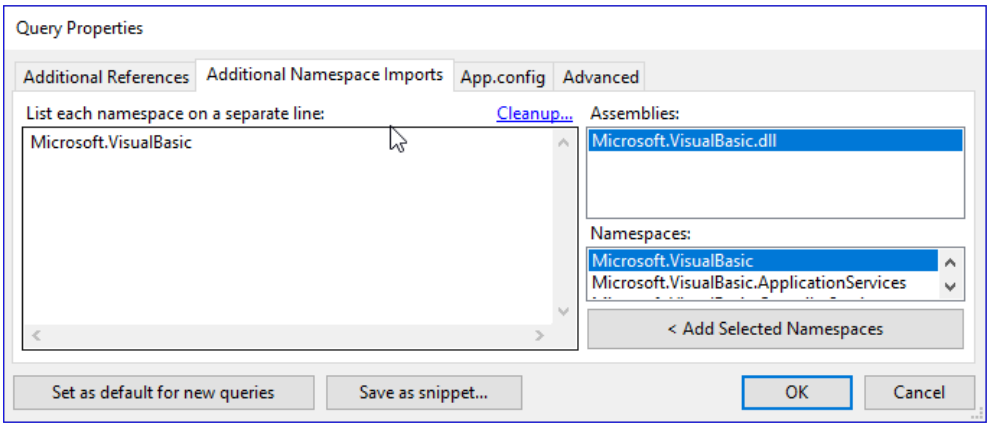



**Bild 3:** Hinzufügen des Namespaces **Microsoft.VisualBasic**

## Mehrere Bedingungen komfortabel mit Select Case prüfen

Wenn Sie unter VBA mehrere Bedingungen prüfen wollen, können Sie das durchaus mit **If...ElseIf...Else...End If** erledigen. Allerdings gibt es auch noch die **Select Case**-Bedingung, mit der Sie Bedingungen mit mehreren Zweigen einfacher abbilden können. **Select Case** ist genauso wie unter VBA aufgebaut. Unter C# setzt man zu diesem Zweck die **switch**-Bedingung ein. **Select Case** funktioniert so:

```
Dim intZahl As Integer
intZahl = 5
Select Case intZahl
    Case >5
        Debug.WriteLine ("Zahl ist größer als 5.")
    Case <5
        Debug.WriteLine ("Zahl ist kleiner als 5.")
    Case Else
        Debug.WriteLine ("Zahl ist gleich 5.")
End Select
```

Wir geben also den ersten Teil des Vergleichsausdrucks hinter der **Select Case**-Zeile an. Der Rest der jeweiligen Vergleichsausdrücke landet jeweils hinter den **Case**-Zweiten. Es gibt einfache **Case**-Zeilen und maximal einen **Case Else**-Zweig, der alle Fälle auffängt, die in den vorherigen Fällen nicht berücksichtigt wurden.

Sie müssen auch nicht wie bei der **switch**-Bedingung unter C# für jede Bedingung eine Anweisung einfügen, welche die Bedingung verlässt, wenn einer der Zweige angesteuert wird. Wenn der erste Case-Zweig angesteuert wurde, werden die übrigen automatisch nicht mehr ausgeführt.

Bei der **switch**-Bedingung unter C# konnten Sie auch einen **case**-Zweig leerlassen. In diesem Fall wurden, wenn dieser **case**-Zweig angesteuert wurde, die Anweisungen des folgenden **case**-Zweiges ausgeführt. Das ist bei **Select Case** nicht der Fall. Enthält ein **Case**-Zweig keine Anweisungen, geschieht auch nichts.

**Select Case**-Anweisungen, die nur eine Anweisung in einem **Case**-Zweig enthalten, können Sie auch wie folgt vereinfacht schreiben:

```
Dim intZahl As Integer
intZahl = 5
Select Case intZahl
        Case > 5: Debug.WriteLine("Zahl ist größer als 5.")
        Case < 5: Debug.WriteLine("Zahl ist kleiner als 5.")
        Case Else: Debug.WriteLine("Zahl ist gleich 5.")
End Select
```

Grundsätzlich können Sie mehrere Anweisungen durch einen Doppelpunkt voneinander getrennt in einer Zeile aufführen.

### Mögliche Ausdrücke für die Case-Zweige

Sie können dem **Case**-Zweig flexiblere Vergleichsausdrücke hinzufügen wie einer **If...Then**-Bedingung.

Wenn Sie beispielsweise in einer **If...Then**-Bedingung die gleiche Variable mit drei verschiedenen Werten vergleichen wollen, müssen Sie dies wie folgt formulieren:

```
Dim intZahl As Integer
intZahl = 5
If intZahl = 1 Or intZahl = 3 Or intZahl = 5 Then
        Debug.WriteLine("intZahl ist 1, 3 oder 5")
End If
```

Hier arbeiten wir ja noch mit einem einfachen Ausdruck. Dieser kann aber auch komplizierter ausfallen, wodurch sich die Schreibarbeit und somit auch die Fehleranfälligkeit massiv erhöhen. Mit **Select Case** geht das viel eleganter:

```
Dim intZahl As Integer
intZahl = 5
Select Case intZahl
    Case 1, 3, 5
        Debug.WriteLine("intZahl ist 1, 3 oder 5")
End Select
```

Das hier hingegen würde nicht funktionieren:

```
Select Case intZahl
    Case 1 Or 3 Or 5
        Debug.WriteLine("intZahl ist 1, 3 oder 5")
End Select
```

Das Ergebnis lautet nämlich 7. Was, 7? Nun: **Or** macht einen binären Vergleich, addiert also die binären Werte: **1** ist **001**, **3** ist **011**, **5** ist **101**. Der **Or**-Vergleich liefert dann **111**, was dezimal **7** entspricht.

# E-Mails mit Outlook verschicken

**Wer Anwendungen mit Visual Studio programmiert, die Daten wie etwa die von Kunden verwaltet, kommt früher oder später nicht um eine Funktion zum Versenden von E-Mails herum. Da gibt es nun zwei Möglichkeiten: Sie verwenden eine eigene SMTP-Klasse, um die E-Mails zu versenden. Das klappt mit .NET-Projekten viel einfacher als etwa unter Access, weil es hier schon fertige Klassen für diesen Anwendungszweck gibt. Allerdings werden die gesendeten Mails dann nicht in Outlook im Ordner »Gesendete Elemente« gespeichert. Deshalb schauen wir uns in diesem Artikel den Versand von Mails per Outlook an. In einem anderen Artikel gehen wir dann auf die Verwendung einer SMTP-Klasse ein.**

## Beispielanwendung

Wir wollen als Beispiel für diesen Artikel eine kleine WPF-Anwendung programmieren, mit der Sie die Daten des E-Mail-Empfängers sowie Betreff, Inhalt und weitere Informationen eingeben können. Eine Schaltfläche soll dann die eingegebenen Daten an eine Outlook-Instanz schicken und die E-Mail versenden.

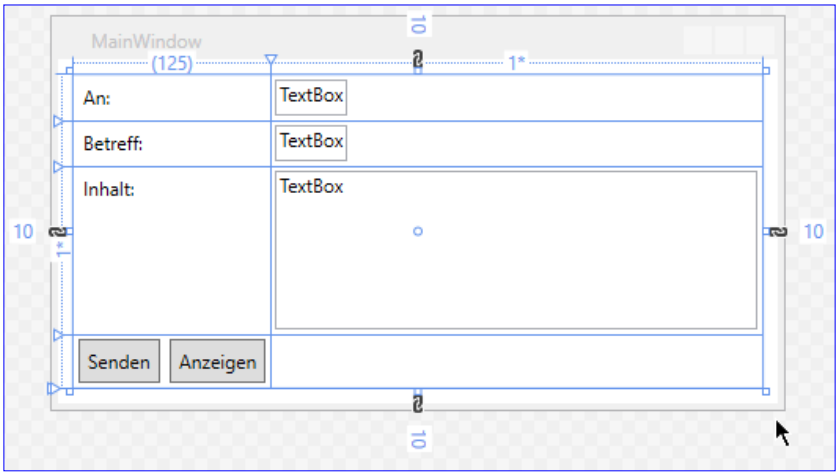



**Bild 1:** Einfaches Formular zum Versenden von E-Mails

Also legen Sie zunächst eine WPF-Anwendung in der von Ihnen bevorzugten Sprache an (also etwa VB oder C#). Das Fenster **MainWindow.xaml** ist dann auch gleich das Element, das wir mit den benötigten Steuerelementen versehen wollen. Dieses gestalten wir wie in Bild 1.

Der dazu notwendige XAML-Code sieht gekürzt so aus (es gibt noch einige Steuerelement-spezifische Eigenschaftsdefinitionen, die wir nicht abbilden):

```
<Window x:Class="MainWindow" ... Title="MainWindow" Height="250" Width="466">
    <Grid HorizontalAlignment="Stretch" Margin="10" VerticalAlignment="Stretch" ClipToBounds="true" >
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="Auto"/>
            <RowDefinition/>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="Auto"/>
        </Grid.RowDefinitions>
```

```
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="Auto"/>
            <ColumnDefinition Width="*"/>
        </Grid.ColumnDefinitions>
        <Label x:Name="label1" Content="An:" />
        <Label x:Name="label2" Content="Betreff:" Grid.Row="1"/>
        <Label x:Name="label3" Content="Inhalt:" Grid.Row="2"/>
        <TextBox x:Name="txtAn" Grid.Column="1" HorizontalAlignment="Stretch"/>
        <TextBox x:Name="txtBetreff" Grid.Row="1" Grid.Column="1" HorizontalAlignment="Stretch"/>
        <TextBox x:Name="txtInhalt" Grid.Row="2" TextWrapping="Wrap" Grid.Column="1" VerticalAlignment="Stretch"
            HorizontalAlignment="Stretch" Height="Auto" AcceptsReturn="True"/>
        <StackPanel Orientation="Horizontal" Grid.Row="3">
            <Button x:Name="btnSenden">Senden</Button>
            <Button x:Name="btnAnzeigen">Anzeigen</Button>
        </StackPanel>
    </Grid>
</Window>
```

Dadurch, dass wir für die dritte Zeile den Wert der Eigenschaften **Height** und **Width** auf **Auto** eingestellt haben, wird diese Zeile gemeinsam mit dem Fenster vergrößert. Damit wir im Textfeld **txtInhalt** mehrzeilige Texte eingeben können, haben wir die Eigenschaft **TextWrapping** auf **Wrap** und **AcceptsReturn** auf **True** eingestellt.

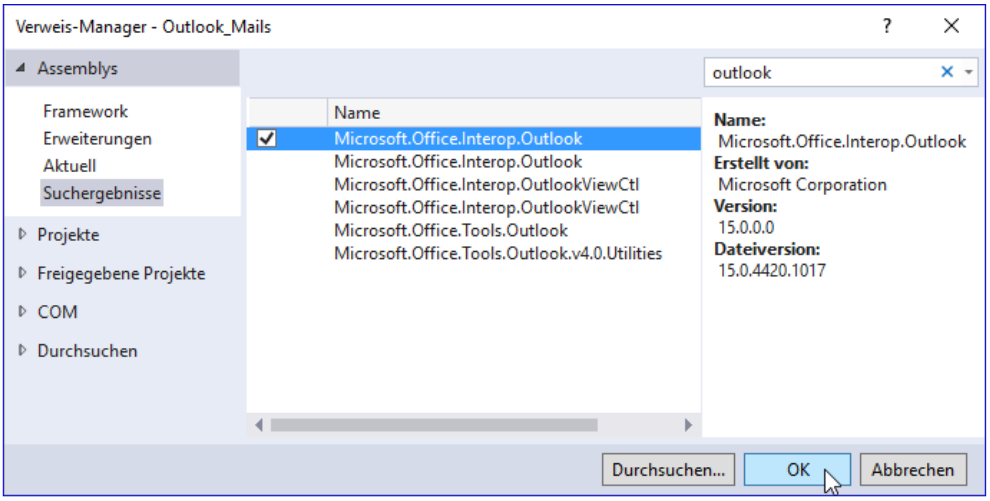



**Bild 2:** Erweitern der Verweise des Projekts um die Outlook-Bibliothek

## Verweis auf Outlook hinzufügen

Damit wir Outlook von einer WPF-Anwendung aus programmieren können, fügen wir den entsprechenden Verweis auf die Outlook-Bibliothek **Microsoft.Office.Interop.Outlook** hinzu. Dazu öffnen Sie den **Verweise-Manager** über den Menüeintrag **Projekt|Verweis hinzufügen...** und suchen im nun erscheinenden Dialog nach dem Wort **Outlook**.

Selektieren Sie den Eintrag **Microsoft.Office.Outlook.Interop** (siehe Bild 2) und schließen Sie den Dialog mit **OK**. Sollten Sie, wie im obigen Dialog der Fall, mehrere gleichnamige Einträge vorfinden, handelt es sich um verschiedene Versionen. Wählen Sie dann die aktuellere Version aus.

Dann statten wir die Schaltfläche **cmdSenden** mit der Ereigniseigenschaft **Click** aus. Nach der Eingabe des Gleichheitszeichens können Sie mit der Tabulator-Taste automatisch die entsprechende Ereignismethode anlegen, indem Sie aus der dann erscheinenden Auswahlliste den Eintrag **<Neuer Ereignishandler>** auswählen:

```
<Button x:Name="btnSenden" Click="btnSenden_Click">Senden</Button>
```

In der Code behind-Klasse erscheint dann die folgende Methode:

```
Private Sub btnSenden_Click(sender As Object, e As RoutedEventArgs)

End Sub
```

In der Klasse fügen wir nun zunächst eine **Imports**-Anweisung hinzu, mit der wir den Outlook-Namespace verfügbar machen:

```
Imports Outlook = Microsoft.Office.Interop.Outlook
```

Damit greifen wir über das Schlüsselwort **Outlook** auf Elemente dieses Namespaces zu.

## Textfelder mit Beispiel füllen

Bevor wir uns auf die Outlook-Programmierung stürzen, legen wir noch eine Ereignismethode für das Ereignis **Loaded** des **Window**-Elements an. Diese soll die drei Textfelder mit Beispieltexten füllen, damit das Fenster beim Öffnen gleich wie in Bild 3 aussieht:

```
Private Sub Window_Loaded(sender As Object, e As RoutedEventArgs)
    txtAn.Text = "andre@minhorst.com"
    txtBetreff.Text = "Testmail"
    txtInhalt.Text = "Hallo Herr Minhorst," & vbCrLf _
        & vbCrLf _
        & "dies ist eine Testmail." & vbCrLf _
        & vbCrLf _
        & "Mit freundlichen Grüßen" & vbCrLf _
        & "André Minhorst"
End Sub
```

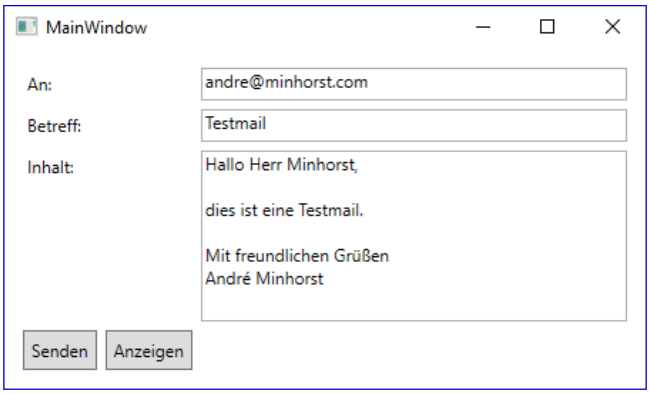



**Bild 3:** Beispielfenster zum Versenden von E-Mails

## Outlook-Mail erzeugen

Danach kümmern wir uns um den Code, der die Mail mit den angegebenen Daten versenden soll. Dazu füllen wir die bereits angelegte Ereignismethode **btnSenden_Click** wie folgt:

```
Private Sub btnSenden_Click(sender As Object, e As RoutedEventArgs)
    Dim objOutlook As Outlook.Application
```

```
    Dim objMailItem As Outlook.MailItem
    objOutlook = New Outlook.Application
    objMailItem = objOutlook.CreateItem(Outlook.OlItemType.olMailItem)
    With objMailItem
        .To = txtAn.Text
        .Subject = txtBetreff.Text
        .Body = txtInhalt.Text
        .Send()
    End With
End Sub
```

Wir erstellen also ein Objekt des Typs **Microsoft.Office.Interop.Outlook.Application**, den wir hier aufgrund unserer **Imports**-Anweisung vereinfacht mit **Outlook.Application** referenzieren können, und speichern einen Verweis darauf in der Variablen **objOutlook**. Die zweite Variable namens **objMailItem** mit dem Datentyp **MailItem** nutzen wir, um ein mit der **CreateItem**-Methode von **objOutlook** erzeugtes **MailItem**-Objekt zu referenzieren.

Dies dient dem Zweck, seine Eigenschaften **To**, **Subject** und **Body** mit den Inhalten der drei Textfelder **txtAn**, **txtBetreff** und **txtInhalt** zu füllen. Die letzte Anweisung heißt **Send** und sorgt dafür, dass die Mail direkt verschickt wird.

Wenn Sie die Anwendung nun starten und die Mail mit einem Klick auf die Schaltfläche **cmdSenden** verschicken, können Sie diese anschließend im Ordner **Gesendete Objekte** von Outlook einsehen.

### E-Mail erst anzeigen

Etwas transparenter ist das Ganze natürlich noch, wenn Sie die E-Mail betrachten können, bevor Sie diese durch einen Klick auf die dafür vorgesehene Schaltfläche absenden. Dazu legen wir für die Schaltfläche **cmdAnzeigen** ebenfalls die **Click**-Eigenschaft an:

```
<Button x:Name="btnAnzeigen" Click="btnAnzeigen_Click">Anzeigen</Button>
```

Den Code dieser Methode brauchen wir gegenüber der zuvor beschriebenen nur leicht zu variieren:

```
Private Sub btnAnzeigen_Click(sender As Object, e As RoutedEventArgs)
    ...
    With objMailItem
        ...
        .Display()
    End With
End Sub
```

Hier rufen wir nämlich statt der **Send**-Methode die **Display**-Methode auf. Dies zeigt beim Ausprobieren die Mail aus Bild 4 an. Nach dem Betrachten schickt man die E-Mail dann mit einem Klick auf die **Senden**-Schaltfläche ab.

# Absender von Outlook-Mails einstellen

**Im Artikel »E-Mails mit Outlook verschicken« haben wir gezeigt, wie Sie von einer .NET-Anwendung aus die Daten für eine E-Mail zusammenstellen und diese dann per Outlook verschicken können – unter anderem mit Empfänger, CC, BCC, Anlagen oder Priorität. Was noch fehlt, ist die Absenderadresse. Diese braucht man grundsätzlich nicht explizit einzustellen, denn Outlook verwendet dann die Adresse des Standardkontos von Outlook (vorausgesetzt, Sie haben mehrere Konten). Wenn Sie jedoch eine Mail mit einem anderen Konto als dem Standardkonto verschicken wollen, wird es kompliziert. Die Lösung für diese Aufgabe finden Sie im vorliegenden Artikel.**

## Verschiedene Outlook-Konten

Wenn Sie wie im **Artikel E-Mails mit Outlook verschicken** ein **MailItem**-Objekt erzeugen und seine Eigenschaften wie **To**, **CC**, **BCC**, **Subject** oder **Body** füllen, werden Sie vergeblich ein Feld etwa namens **From** suchen. Dieses Feld gibt es nämlich nicht – Sie können also nicht einfach irgendeine Absenderadresse zu Ihrer E-Mail hinzufügen.

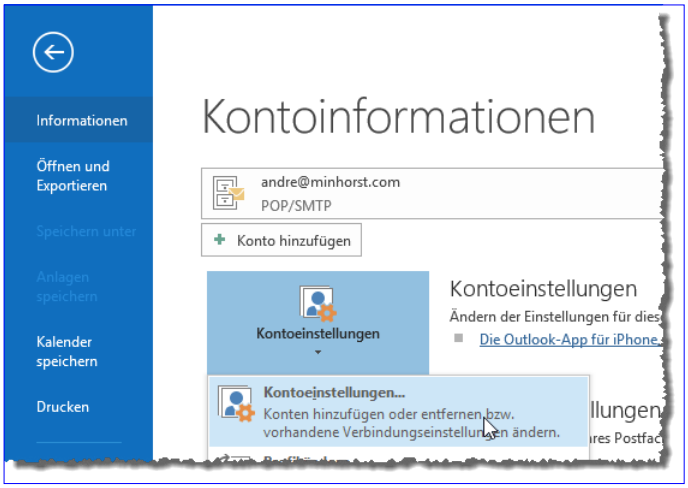



**Bild 1:** Anzeigen der Kontoeigenschaften

Woher weiß Outlook dann aber, welche Absenderadresse überhaupt zu verwenden ist? Ganz einfach: Es gibt in Outlook ein oder mehrere Konten, von denen eines als Standard festgelegt ist. Wann immer Sie direkt von Outlook aus eine neue E-Mail erstellen, wird die zum Standardkonto gehörende E-Mail-Adresse als Absenderadresse eingestellt. Wo finden Sie diese Konten und Einstellungen? Dazu öffnen Sie Outlook, klicken auf den Ribbon-Tab **Datei** und dann im Bereich **Informationen** auf die Schaltfläche **Kontoeinstellungen|Kontoeinstellungen...** (siehe Bild 1).

Hier finden Sie dann wie in Bild 2 die angelegten Outlook-Konten vor. Hier können Sie beispielsweise doppelt auf einen der Einträge klicken, um Eigenschaften wie den SMTP-Server oder die E-Mail-Adresse anzupassen oder ein anderes als das aktuelle Konto als Standardkonto festzulegen. Haben Sie nur ein einziges Outlook-Konto, was bei Verwendung eines Rechners für private und geschäftliche Zwecke unwahrscheinlich ist, haben Sie natürlich keine Wahl bei der Absenderadresse einer E-Mail. Wir gehen jedoch davon aus, dass viele Benutzer mindestens zwei Outlook-Konten in Outlook verwalten.

## Einstellen des Absenderkontos

Outlook wählt also standardmäßig die E-Mail-Adresse des als Standard festgelegten Kontos beim Anlegen neuer E-Mails. Auf die gleiche Weise verfährt Outlook, wenn Sie wie im oben genannten Artikel eine neue E-Mail von einer .NET-Anwendung aus erstellen. Die einfachste Weise, das Absenderkonto zu ändern, erhalten Sie, wenn Sie die neu erstellte E-Mail mit der Display-

Methode anzeigen (siehe Bild 3). Das ist aber natürlich keine Option, wenn Sie die E-Mail direkt nach der Eingabe der Daten wie im Artikel **E-Mails mit Outlook verschicken** versenden wollen oder gar automatisiert E-Mails mit vorgefertigtem Inhalt an einen oder mehrere Empfänger gehen sollen. In diesem Fall benötigen Sie eine Möglichkeit, den Absender per Code einzustellen oder diesen zumindest gleich beim Zusammenstellen der Mail in der Anwendung festzulegen.

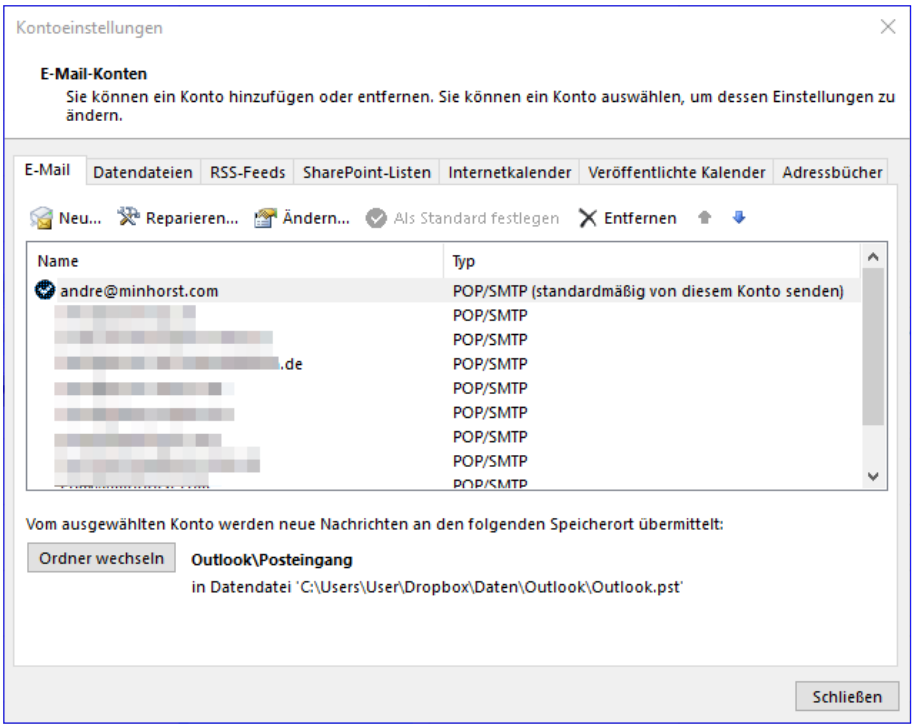



**Bild 2:** E-Mail-Konten unter Outlook

## Absenderadresse festlegen

Wir werden uns zwei Varianten ansehen, um die Absenderadresse festzulegen. Bei der ersten geben wir die Absenderadresse einfach in ein Textfeld ein und versuchen dann beim Zusammenstellen der E-Mail, das betroffene Outlook-Konto zu ermitteln und als Absenderkonto einzustellen. Im zweiten Anlauf wollen wir dann vorab alle Outlook-Konten auslesen und die E-Mail-Adressen per Kombinationsfeld zur Auswahl bereitstellen.

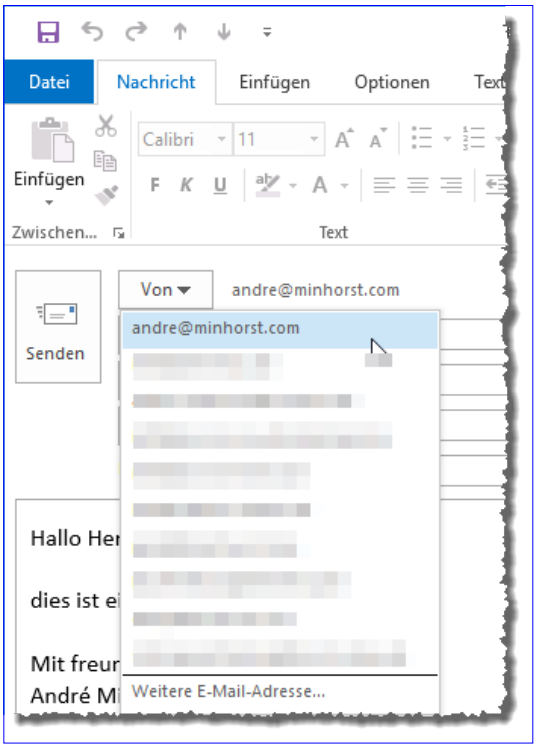



**Bild 3:** Auswahl des Absenders in der E-Mail

Wir bauen die Beispiellösung auf der Anwendung auf, die wir im Artikel **E-Mails mit Outlook verschicken** beschrieben haben. Dazu fügen wir zunächst ein Textfeld namens **txtAbsender** zum Fenster **MainWindow.xaml** hinzu:

```
<TextBox x:Name="txtAbsender" Grid.Column="1"></TextBox>
```

Damit dieses gleich beim Öffnen des Fensters gefüllt wird, fügen wir eine Zeile zur Methode **Window_Loaded** hinzu:

```
Private Sub Window_Loaded(sender As Object, e As RoutedEventArgs)
    txtAbsender.Text = "info@amvshop.de"
    ...
End Sub
```

## Absenderadresse per Code einstellen

Das war der einfache Teil. Nun müssen wir, wenn der Benutzer auf die **Senden**- oder **Anzeigen**-Schaltfläche klickt, neben den bereits bekannten Informationen, die wir der E-Mail einfach über entsprechende Eigenschaften zuweisen können, noch die Absenderadresse einstellen. Dazu deklarieren wir zunächst eine Objektvariable namens **objAccount** mit dem Typ **Microsoft.Office.Interop.Account**. Nach dem Erstellen des **MailItem**-Objekts wollen wir gleich den Absender festlegen. Dazu benötigen wir noch eine weitere Variable namens **bolAccountEingestellt** mit dem Typ **Boolean**, deren Funktion wir gleich erläutern.

In einer **For Each**-Schleife durchlaufen wir nun alle Outlook-Konten, die wir über die Auflistung **objOutlook.Session.Accounts** erhalten. Das aktuelle Konto landet dabei in der Laufvariablen **objAccount**. Für dieses prüfen wir zunächst in einer **If...Then**-Bedingung, ob es sich um ein POP3-Konto handelt. Falls ja, vergleichen wir die Absenderadresse des Kontos, die wir über die Eigenschaft **DisplayName** ermitteln, mit der von uns gewünschten Absenderadresse. Stimmen beide überein, weisen wir das aktuell in **objAccount** befindliche Konto der Eigenschaft **SendUsingAccount** zu und stellen die Variable **bolAccountEingestellt** auf **True** ein. Auf diese Weise durchlaufen wir alle Einträge der **Accounts**-Auflistung der aktuellen Session, bis wir ein passendes Konto gefunden haben – oder eben nicht: In diesem Fall verlassen wir die Schleife, ohne die Variable **bolAccountEingestellt** auf den Wert **True** gesetzt zu haben.

Die folgende **If...Then**-Bedingung prüft, ob **bolAccountEingestellt** den Wert **False** aufweist und damit gegebenenfalls kein Account zur Absenderadresse gefunden wurde. In diesem Fall erscheint eine Meldung, die den Benutzer darauf hinweist, dass das gewünschte Konto zur Absenderadresse nicht existiert, und schlägt ihm zwei Alternativen vor: entweder, mit dem Standardkonto von Outlook zu senden oder den Vorgang abzubrechen:

```
Private Sub btnAnzeigen_Click(sender As Object, e As RoutedEventArgs)
    Dim objOutlook As Outlook.Application
    Dim objMailItem As Outlook.MailItem
    Dim objAccount As Outlook.Account
    Dim strAnhang As String
    Dim bolAccountEingestellt As Boolean
    objOutlook = New Outlook.Application
    objMailItem = objOutlook.CreateItem(Outlook.OlItemType.olMailItem)
    With objMailItem
        For Each objAccount In objOutlook.Session.Accounts
            If objAccount.AccountType = Outlook.OlAccountType.olPop3 Then
                If objAccount.DisplayName = txtAbsender.Text Then
                    .SendUsingAccount = objAccount
                    bolAccountEingestellt = True
                    Exit For
                End If
            End If
        Next
        If bolAccountEingestellt = False Then
            If MessageBox.Show("Für die angegebene Absenderadresse '" & txtAbsender.Text _
```

# E-Mails ohne Outlook versenden

**Im Artikel »E-Mails mit Outlook verschicken« haben wir gezeigt, wie Sie von einer .NET-Anwendung aus die Daten für eine E-Mail zusammenstellen und diese dann per Outlook verschicken können. Was aber, wenn der Benutzer kein Outlook auf dem Rechner hat? Für diesen Fall bietet .NET eigene Bibliotheken. Damit können Sie E-Mails ohne Zugriff auf Outlook verschicken. Dieser Artikel zeigt, wie dies gelingt.**

Wir wollen in diesem Artikel das gleiche Fenster verwenden, mit dem wir auch in der Beispiellösung zum Artikel **E-Mails mit Outlook verschicken** gearbeitet haben (siehe Bild 1). Einziger Unterschied: Die Schaltfläche **btnAnzeigen** fehlt, denn wir arbeiten ja nicht mit Outlook und können die Mail vor dem Versenden auch nicht mehr separat anzeigen. Dafür haben wir eine Schaltfläche namens Konfiguration hinzugefügt. Diese soll einen weiteren Dialog öffnen, der die Konfigurationsdaten für den Mailserver speichert.

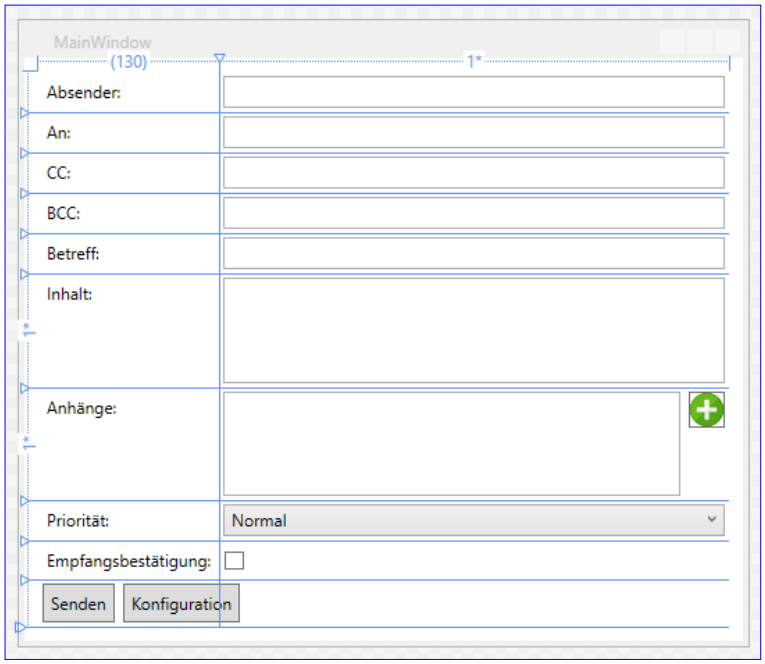



**Bild 1:** Fenster zur Eingabe der Mail-Daten

Für die Konfigurationsdaten legen wir einige Einträge im Bereich **Einstellungen** des aktuellen Projekts an, und zwar die folgenden (siehe Bild 2):

- **Host**: Name/IP des SMTP-Servers
- **Port**: Port des SMTP-Servers
- **Username**: Benutzername für die Anmeldung am SMTP-Server
- **Password**: Kennwort für die Anmeldung am SMTP-Server

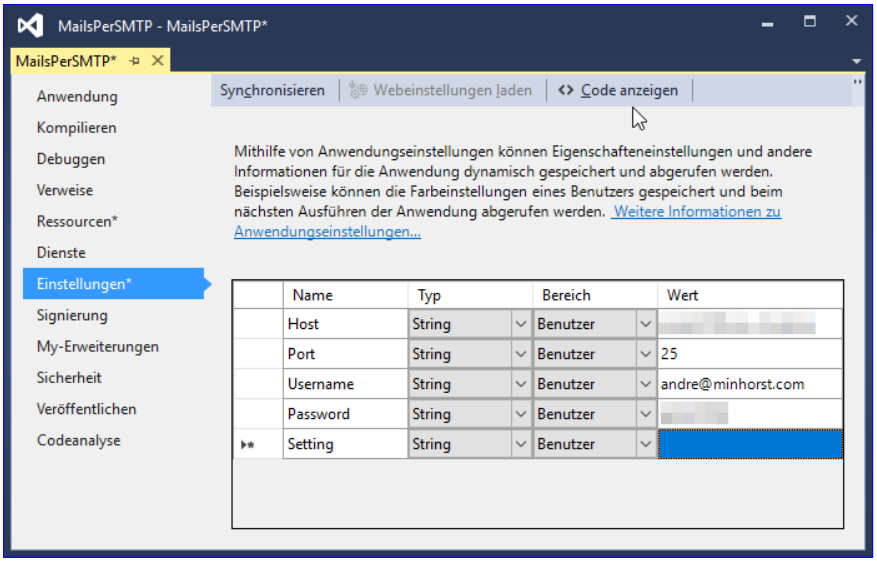



**Bild 2:** Anlegen der Konfigurationsdaten

Wenn Sie die Eigenschaften im Bereich **Einstellungen** angelegt haben und diese speichern, erstellt Visual Studio eine Klasse namens **MySettings**. Diese können Sie dann initialisieren und damit direkt die angelegten Eigenschaften ansprechen, also etwa **objSetting.User-**

**name** – mehr dazu weiter unten. Die Ereignismethode **Window_Load** füllt die Felder **txtAbsender**, **txtAn**, **txtBetreff** und **txtInhalt** mit Beispieldaten. Dies war beim Testen des Beispiels hilfreich, Sie werden diese Felder jedoch auf andere Weise füllen wollen – vorausgesetzt, Sie wollen überhaupt Mails über ein Fenster versenden und nicht direkt etwa mit den Daten aus einer Datenbank.

### Erstellen und Senden der E-Mail

Die durch die Schaltfläche **btnSenden** ausgelöste Ereignismethode stellt die E-Mail zusammen und verschickt diese dann. Da wir hier nicht Outlook als Mail-Client verwenden, sondern die Klasse **MailMessage**, benötigen wir etwas mehr Code für die Durchführung der Aufgabe. Das liegt vorrangig daran, dass wir die Daten des SMTP-Servers sowie die Benutzerdaten erst noch angeben müssen, die ja bei Benutzung von Outlook als Mail-Client bereits im jeweiligen Outlook-Konto gespeichert sind. Um die nachfolgend verwendeten Klassen nutzen zu können, benötigen wir noch Verweise auf die folgenden Namespaces:

```
Imports System.Net.Mail
Imports System.Net
```

Die Methode deklariert zunächst einige Variablen und initialisiert dann ein neues Objekt auf Basis der Klasse **MailMessage**:

```
Private Sub btnSenden_Click(sender As Object, e As RoutedEventArgs)
    Dim objMail As MailMessage
    Dim objMailAddress As MailAddress
    Dim objAttachment As Attachment
    Dim objSMTPClient As SmtpClient
    Dim objNetworkCredentials As NetworkCredential
    Dim objSettings As MySettings
    objMail = New MailMessage
```

Dann beginnt die Methode, die Eigenschaften des neuen **MailMessage**-Objekts zu füllen. Als Absender geben wir für die Eigenschaft **From** ein Objekt des Typs **MailAddress** an, dem wir beim Initialisieren den Inhalt des Textfeldes **txtAn** übergeben:

```
    With objMail
        objMailAddress = New MailAddress(txtAn.Text)
        .From = objMailAddress
```

Die Empfängeradresse weisen wir mit der **Add**-Methode der Eigenschaft **To** zu. Betreff und Inhalt landen wie auch bei Outlook in den Eigenschaften **Subject** und **Body**:

```
        .To.Add(txtAn.Text)
        .Subject = txtBetreff.Text
        .Body = txtInhalt.Text
```

Die Anlagen werden auch etwas anders zur E-Mail hinzugefügt wir unter Verwendung von Outlook. Wir können der **Attachments**-Auflistung hier nicht einfach den Namen der hinzuzufügenden Datei zuweisen, sondern müssen erst ein Objekt des Typs

# Mit Ressourcen arbeiten

**In einer .NET-Anwendung können Sie auch ohne Verwendung einer Datenbank Dateien und andere Daten ablegen. Dazu stellt Microsoft die sogenannten Ressourcen (englisch Resources) zur Verfügung. Dabei handelt es sich um die Möglichkeit, Dateien, Texte oder andere Daten entweder über die Benutzeroberfläche von Visual Studio zur Anwendung hinzuzufügen und sie per Code auszulesen und gegebenenfalls auch wieder ins Dateisystem zu kopieren. Im konkreten Fall wollen wir eine Word-Dokumentvorlage in der Anwendung vorhalten, um diese bei Bedarf ins Dateisystem zu kopieren und als Vorlage zu nutzen.**

## Beispiel Word-Dokumentvorlage

Anlass zu diesem Artikel ist ein weiterer Artikel namens **Briefe mit Word erstellen**, in dem wir auf Basis einer Word-Dokumentvorlage ein neues Word-Dokument erstellen wollen.

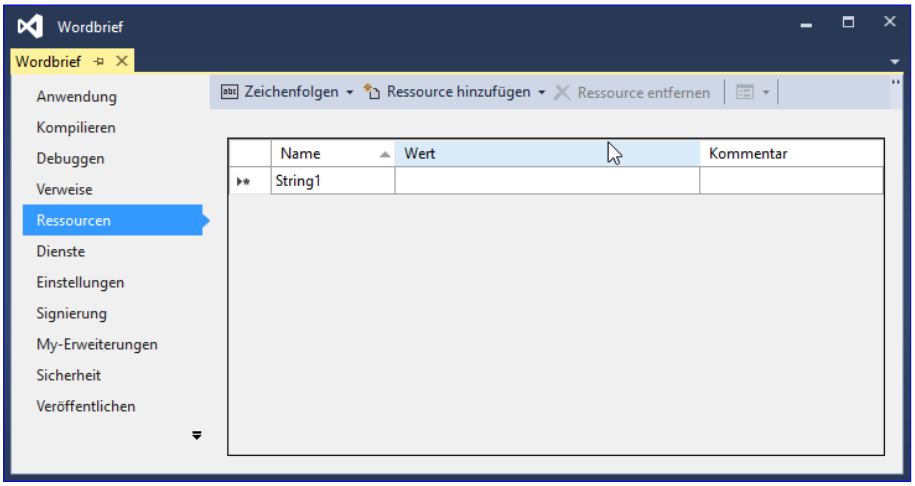



**Bild 1:** Der Dialog zum Verwalten der Ressourcen über Visual Studio

Für den Fall, dass der Benutzer die Anwendung frisch auf dem Rechner installiert hat, soll die Dokumentvorlage vor dem erstmaligen Erstellen von der Anwendung ins Dateisystem kopiert werden, damit der Code zum Erstellen des Word-Dokuments auf Basis dieser Vorlage auf die Vorlage zugreifen kann.

## Ressourcen in Visual Studio verwalten

Der einfachste Weg, die Ressourcen zu verwalten, findet sich in der Benutzeroberfläche von Visual Studio. Wenn Sie ein Projekt angelegt haben, klicken Sie doppelt auf den Eintrag **My Project** im Projektmappen-Explorer, um die Eigenschaften der Anwendung anzuzeigen.

Diesen Bereich öffnen Sie auch mit dem Menübefehl **Projekt|Eigenschaften**. Hier wechseln Sie dann durch einen Klick auf den Eintrag **Ressourcen** in der linken Leiste zum gleichnamigen Bereich. In einem frischen Projekt ist die Liste der Ressourcen noch leer (siehe Bild 1).

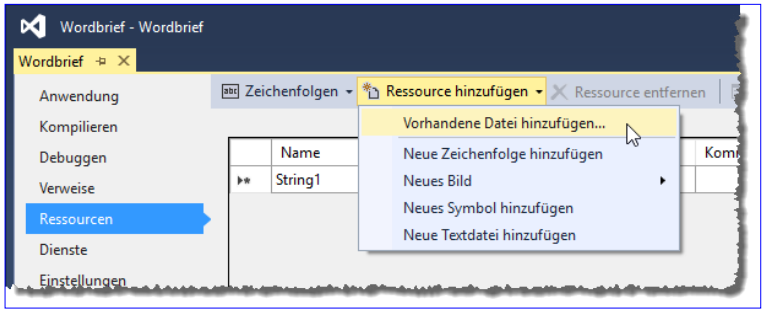



**Bild 2:** Hinzufügen einer Datei als Ressource

Um eine Datei als Ressource hinzuzufügen, klicken Sie auf den **Nach unten**-Pfeil der Schaltfläche **Ressource hinzufügen** im oberen Bereich und wählen im nun erscheinenden Menü den Eintrag **Vorhandene Datei hinzufügen...** aus (siehe Bild 2).

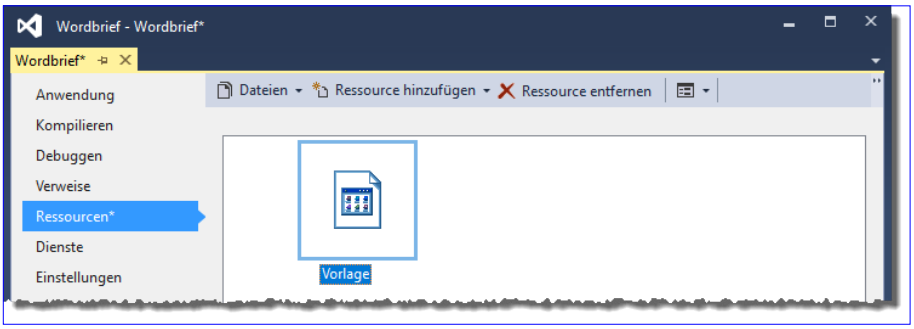



**Bild 3:** Anzeige der neuen Ressource in den Projekteigenschaften

Im folgenden Dialog **Vorhandene Datei zu Ressourcen hinzufügen** wählen Sie nun die hinzuzufügende Datei aus, in unserem Fall eine Datei namens **Vorlage.dotx**.

Nach dem Hinzufügen zeigt der **Ressourcen**-Bereich die neue Ressource wie in Bild 3 an.

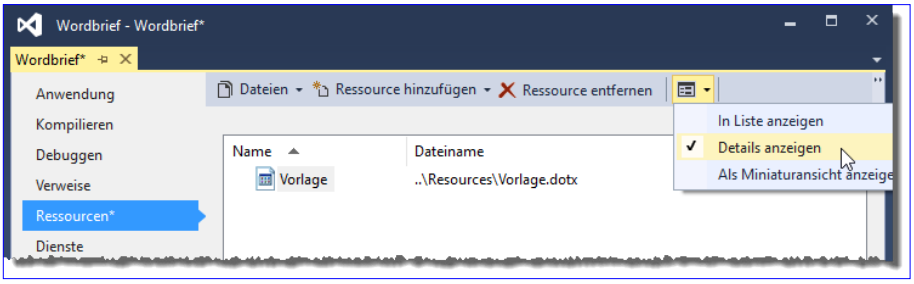



**Bild 4:** Detaillierte Liste der Ressourcen

Diese Ansicht ist nicht besonders aussagekräftig, deshalb wechseln wir wie in Bild 4 über die Auswahl der Ansichten zur Ansicht **Details anzeigen**. Damit befindet sich die Ressource bereits in der Anwendung. Sie müssen nun noch die Änderungen speichern, am einfachsten mit der Tastenkombination **Strg + S**.

## Ressource per Projektmappen-Explorer hinzufügen

Nach dem Hinzufügen der ersten Ressource zeigt der Projektmappen-Explorer einen Ordner für die Ressourcen an (siehe Bild 5). Dieser bietet auch ein Kontextmenü, über dessen Eintrag **Hinzufügen|Neues Element...** Sie den Dialog **Neues Element hinzufügen** öffnen. Hier können Sie allerdings nicht beliebige Dateien hinzufügen, sodass der Weg über den Ressourcen-Bereich in den Projekteigenschaften die einfachere Variante ist.

## Zur Laufzeit mit Ressourcen arbeiten

Nun ist das primäre Ziel, diese Ressource zur Laufzeit aus der Anwendung ins Dateisystem zu kopieren. Dies wollen wir als Nächstes realisieren. Damit erschöpfen sich die Möglichkeiten auch: Die als Ressource

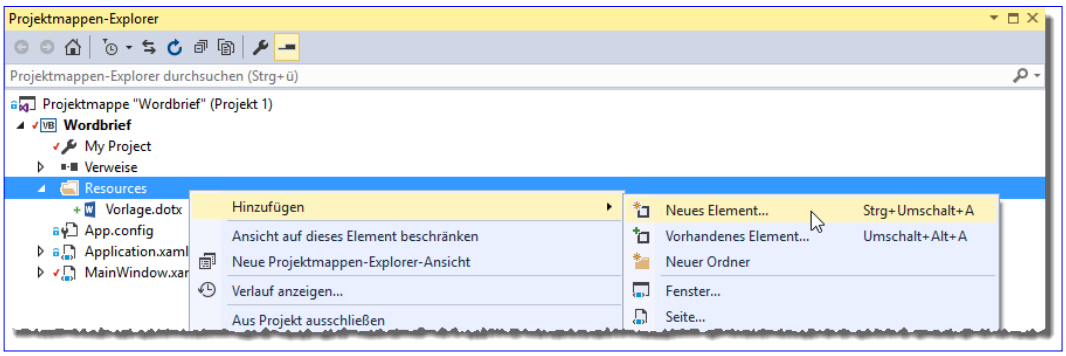



**Bild 5:** Ressource über den Projektmappen-Explorer hinzufügen

hinzugefügten Dateien werden nämlich beim Kompilieren in die **.exe**-Datei integriert. Sie können also nicht etwa zur Laufzeit weitere Elemente zu den Ressourcen einer Anwendung hinzufügen.

## Schaltfläche zum Exportieren der Ressource

Für unsere erste Aufgabe fügen wir dem Fenster **MainWindow.xaml** eine Schaltfläche namens **btnRessourceExportieren** hinzu und legen für diese eine Ereignismethode an, die durch das Ereignis **Click** ausgelöst wird. Die Definition der Schaltfläche sieht so aus:

```
<Button x:Name="btnRessourceExportieren"
Click="btnRessourceExportieren_Click">
Ressource exportieren</Button>
```

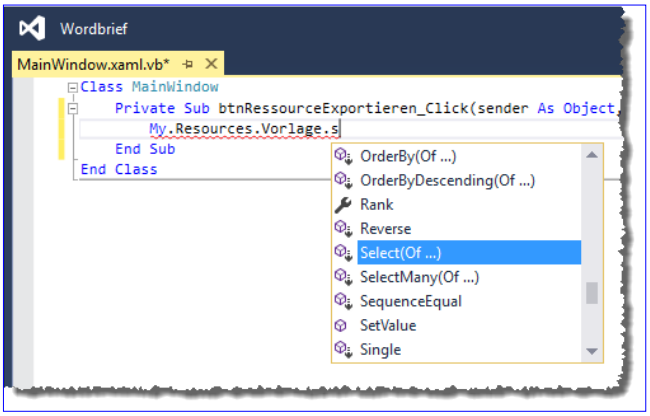



**Bild 6:** Zugriff auf die Ressource per Code

Wenn Sie nun zum Code-Editor für die Klasse **MainWindow.xaml.vb** wechseln, können Sie über **My.Resources** auf alle in den Ressourcen gespeicherten Elemente zugreifen, also auch auf unser Element mit der Bezeichnung **Vorlage**. Dieses bietet wiederum eine Reihe von Eigenschaften und Methoden an, unter denen sich aber leider nicht direkt eine Eigenschaft namens **SaveAs** findet (siehe Bild 6).

Wie bekommen wir die als Ressource gespeicherte Datei **Vorlage.dotx** nun dennoch in unser Dateisystem? Dazu brauchen wir die Methode lediglich wie folgt zu erweitern:

```
Private Sub btnRessourceExportieren_Click(sender As Object, e As RoutedEventArgs)
    Dim objVorlage As Byte()
    objVorlage = My.Resources.Vorlage
    My.Computer.FileSystem.WriteAllBytes(AppDomain.CurrentDomain.BaseDirectory + "\Vorlage.dotx", objVorlage, False)
End Sub
```

Die erste Zeile deklariert ein Byte-Array, die zweite weist diesem den Inhalt der Ressource zu, die wir mit **My.Resources.Vorlage** referenzieren. Die dritte speichert die Ressource unter dem Dateinamen **Vorlage.dotx** im aktuellen Verzeichnis der **.exe**-Anwendung.

## Zielverzeichnis definieren

Nun wollen wir dem Benutzer noch die Möglichkeit geben, die Ressource in einem von ihm ausgewählten Verzeichnis zu speichern. Dazu legen wir eine zweite Schaltfläche namens **btnRessourceExportierenAls** an, für die wir die folgende Methode hinterlegen:

```
Private Sub btnRessourceExportierenAls_Click(sender As Object, e As RoutedEventArgs)
    Dim objVorlage As Byte()
    Dim objSaveFileDialog As SaveFileDialog
```

# Brief mit Word erstellen

**Heute war es wieder soweit: Ich musste ein Anschreiben erstellen. Heutzutage geht zwar vieles per E-Mail, aber hier und da wird doch noch nach einem korrekten Anschreiben mit Briefkopf, Adresse, Ort und Datum, Betreff und dem Text selbst verlangt. Da dies so selten passiert, finde ich meist meine Wordvorlage für diesen Zweck nicht: Seit dem letzten Anschreiben kann es nämlich gut sein, dass ich einen neuen oder neu aufgesetzten Rechner habe und ich vergessen habe, meine Vorlagen vom alten Rechner auf den neuen zu übertragen. Wie schön wäre es doch, wenn man eine kleine .NET-Anwendung hätte, die einem ein paar Textfelder für die wichtigsten Daten bereitstellt und nach der Eingabe per Mausklick das gewünschte Dokument im .docx-Format anlegt! Wie das gelingt, zeigt Ihnen der vorliegende Artikel.**

Wenn Sie ein Dokument wie oben beschrieben erstellen möchten, erledigen Sie das üblicherweise auf Basis einer Dokumentvorlage, die bereits einige wichtige Informationen enthält – zum Beispiel, wo die einzelnen Elemente wie Briefkopf, Anschrift, Ort und Datum, Betreff und Inhalt angelegt werden. Genau diese Dokumentvorlage benötigen wir auch für die Beispiellösung zu diesem Beitrag. Dumm nur, dass sich genau das Mitnehmen solcher Dokumente beim Aufsetzen eines neuen Rechners bei mir als Problem herausstellte und ich dann, statt die Vorlage auf dem alten Rechner zu suchen, lieber schnell von Hand ein neues Anschreiben zusammengesetzt habe. Also sollte die Lösung auch die Möglichkeit bieten, zumindest eine Dokumentvorlage zu speichern und diese dann bei Bedarf bereitzustellen. Wie dies gelingt, zeigen wir ausführlich im Artikel **Mit Resourcen arbeiten**.

Die grobe Vorgehensweise sieht so aus: Wir haben eine Anwendung, welche eine Word-Dokumentvorlage als Ressource enthält. Die Benutzeroberfläche stellt Textfelder bereits, mit denen Sie die im Anschreiben auszugebenden Informationen eintippen können. Nach der Eingabe können Sie per Mausklick einen Vorgang starten, bei dem die Anwendung die Dokumentvorlage im Dateisystem speichert (sofern diese noch nicht vorhanden ist) und ein neues Dokument auf Basis dieser Vorlage erstellt und mit den in der Anwendung eingegebenen Daten füllt. Das Dokument soll anschließend geöffnet werden, damit der Benutzer es beispielsweise ausdrucken kann. Alternativ wollen wir noch eine Schaltfläche anlegen, mit welcher der Benutzer direkt ein PDF-Dokument auf Basis des erstellten Word-Dokuments erzeugen kann.

## Dokumentvorlage vorbereiten

Der erste Schritt ist das Vorbereiten einer Dokumentvorlage. Das heißt, dass wir in Microsoft Word ein neues Dokument anlegen, die gewünschten Bereiche inklusive Platzhalter definieren und beispielsweise den Briefkopf hinzufügen und das Dokument dann unter der Dateiendung **.dotx** speichern. Letzteres ist besonders wichtig, damit wir neue Dokumente auf Basis dieser Vorlage anlegen können. In welcher Form wollen wir die Platzhalter anlegen? Am einfachsten ist es, einfach entsprechende Texte in eckigen Klammern zu den jeweiligen Elementen hinzuzufügen, also beispielsweise **[Betreffzeile]**.

## Vorlage nach DIN 5008

Wenn wir schon eine neue Vorlage anlegen, wollen wir diese auch halbwegs DIN-gerecht erstellen. Deshalb schauen wir uns dabei die aktuellen Vorgaben entsprechend DIN 5008 an, die etwa unter **www.din-5008-richtlinien.de** schön zusammenge-

fasst sind. Wir beginnen mit dem Anschriftenfeld, für das die DIN-Norm angibt, dass es 40 x 85 mm groß ist und einen Abstand von 45 mm zum oberen und 20 mm zum linken Rand aufweist (es gibt noch eine Variante mit 27 mm zum oberen Rand, aber wir wollen die Version für Briefumschläge mit Adressfenster nutzen).

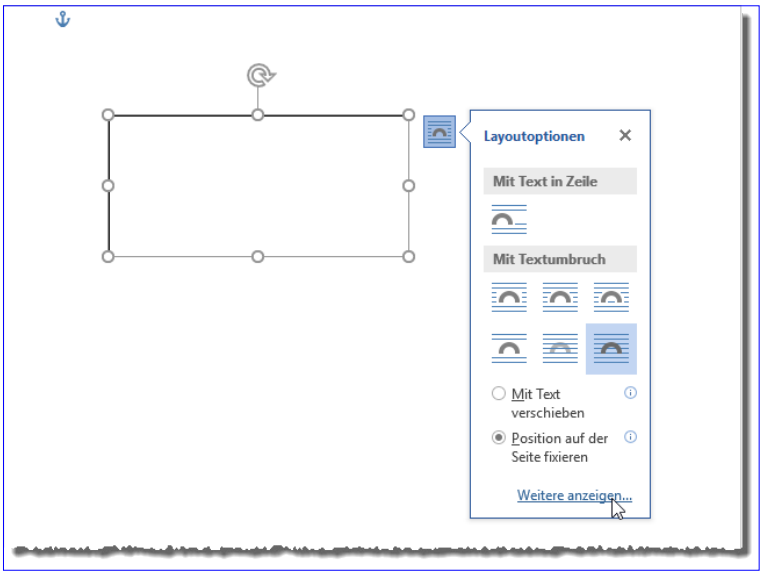



**Bild 1:** Anzeigen der Textfeld-Eigenschaften

## Anschriftenfeld

Wir öffnen also Word und legen ein neues, leeres Dokument an. Für das Anschriftenfeld fügen Sie dann ein Textfeld hinzu. Dies erledigen wir mit dem Ribbon-Eintrag **Einfügen|Text|Textfeld|Textfeld** erstellen. Ziehen Sie nach Betätigung dieses Befehls einen Rahmen auf, der etwa den gewünschten Abmessungen entspricht. Klicken Sie dann auf die Schaltfläche, die rechts oberhalb vom neuen Textfeld erscheint, und wählen Sie unten den Befehl **Weitere anzeigen** aus (siehe Bild 1).

Bevor wir auf den Dialog **Layout** schauen, noch die Information, dass Word mitunter etwas verwirrend ist (Version 2016). Word zeigt nämlich standardmäßig den verfügbaren Bereich ohne Seitenränder an – zumindest ohne die oberen und unteren Seitenränder. Wenn man das weiß, kann man die Einstellungen im Layout-Dialog wie in Bild 2 ohne Rücksicht darauf vornehmen, dass die Position des Textfeldes in der Ansicht Drucklayout nicht so aussieht, wie sie später auf dem Papier erscheinen wird – also mit 2 cm Abstand zur Seite horizontal und mit 4,5 cm Abstand zur Seite vertikal.

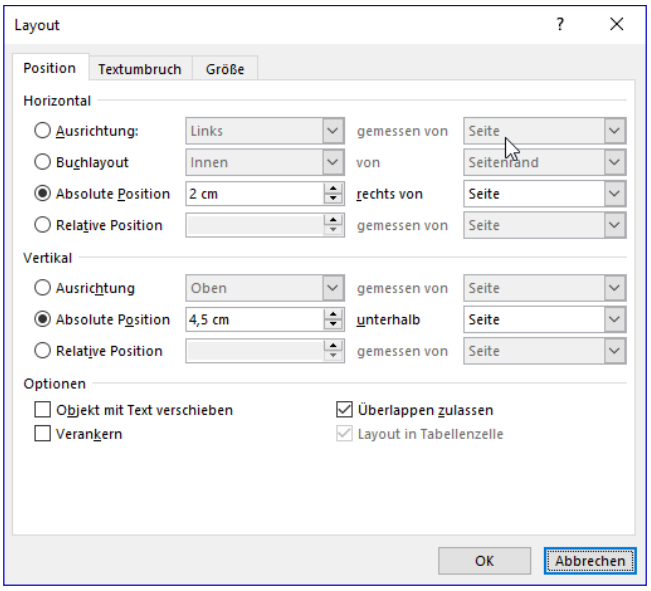



**Bild 2:** Position des Textfelds

Die Größe des Textfeldes stellen wir analog im Bereich Größe des gleichen Dialogs ein – und zwar wie von DIN 5008 vorgegeben auf 4 cm x 8,5 cm (siehe Bild 3). Die Größe können Sie allerdings auch über die beiden Steuerelemente in der Gruppe **Format|Größe** des Ribbons einstellen.

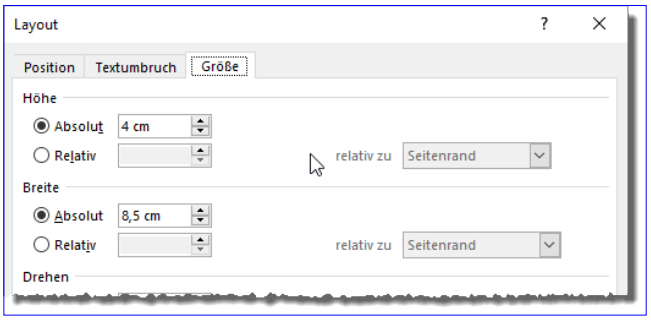



**Bild 3:** Größe des Textfelds

Schließlich weist das Textfeld noch einen Rahmen auf, den wir transparent machen wollen. Dazu wählen Sie beim markiertem Textfeld den Ribbon-Eintrag **Format|Formenarten|Formkontur|Keine Kontur** aus.


**Bild 4:** Einfügen einer Textmarke über das Ribbon

Ganz oben im Anschriftenfeld fügen wir nun eine Zeile mit der Rücksendeadresse ein, also der Adresse, an die der Brief zurückgesendet werden soll, wenn dieser nicht zugestellt werden konnte. Für diese Zeile legen Sie eine recht kleine Schriftgröße fest, zum Beispiel **8**. Darunter soll die eigentliche Anschrift landen – wie, erfahren Sie in den nächsten Abschnitten.

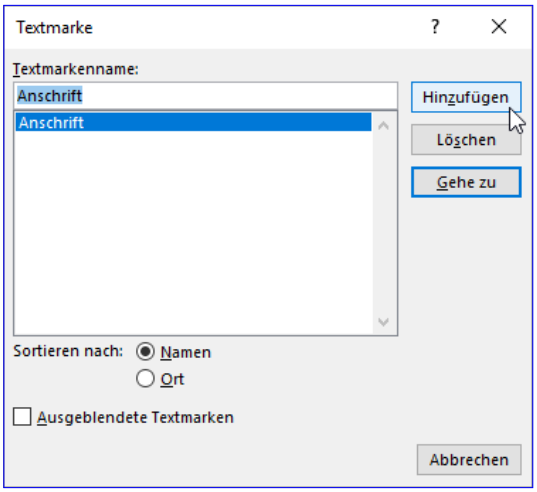



**Bild 5:** Einfügen einer Textmarke

## Textmarke einfügen

Später wollen wir den Textfeldern Texte hinzufügen. Dazu hinterlegen wir in den Textfeldern jeweils eine Textmarke, die wir dann später per Code leicht auffinden und durch die gewünschten Texte ersetzen können. Um die Textmarke hinzuzufügen, platzieren Sie die Einfügemarke im Textfeld für die Anschrift gleich unter der Rücksendeadresse und stellen die Schriftart auf die gewünschte Größe ein, beispielsweise auf **10**. Danach fügen wir die Textmarke ein. Dies erledigen wir mit dem Ribbon-Befehl **Einfügen|Links|Textmarke** (siehe Bild 4).

Dies öffnet den Dialog aus Bild 5, mit dem wir den Namen der neuen Textmarke eingeben und diese dann mit der Schaltfläche **Hinzufügen** an der Position der Einfügemarke einfügen. Im Dokument ist die Textmarke nun zunächst nicht zu erkennen. Das Vorhandensein dokumentiert lediglich der neue Eintrag im Dialog **Textmarke**.

Zur Sicherheit wollen wir die Textmarke jedoch noch sichtbar machen. Dazu öffnen Sie über das Ribbon den Dialog **Word-Optionen** und klicken dort links auf den Eintrag **Erweitert**. Im rechten Bereich finden Sie dann unter Dokumentinhalt anzeigen die Option **Textmarken anzeigen**, die Sie nun aktivieren (siehe Bild 6).

Wenn Sie den Dialog nun mit einem Klick auf die Schaltfläche

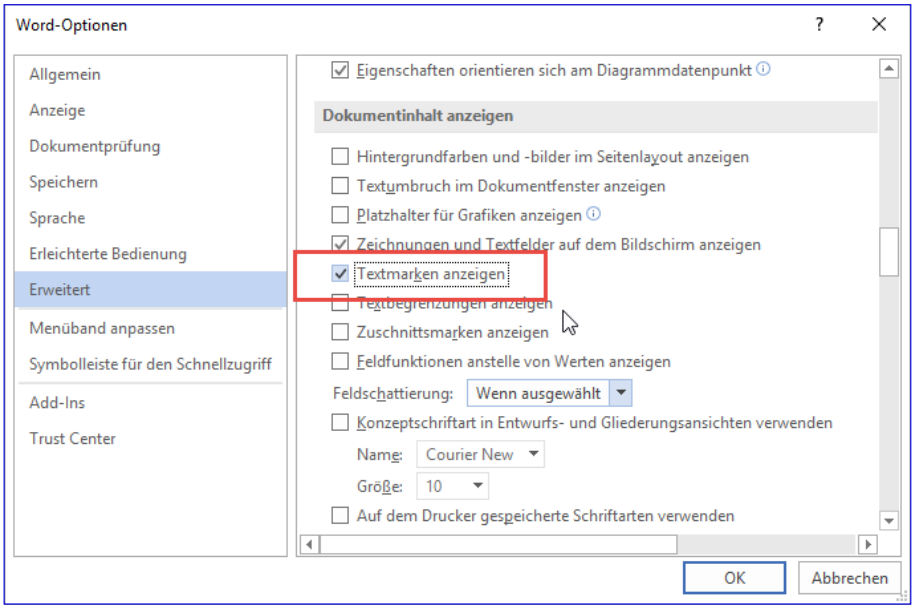



**Bild 6:** Textmarken einblenden

**OK** schließen, finden Sie im Textfeld für die Anschrift die Textmarke wie in Bild 7 vor.

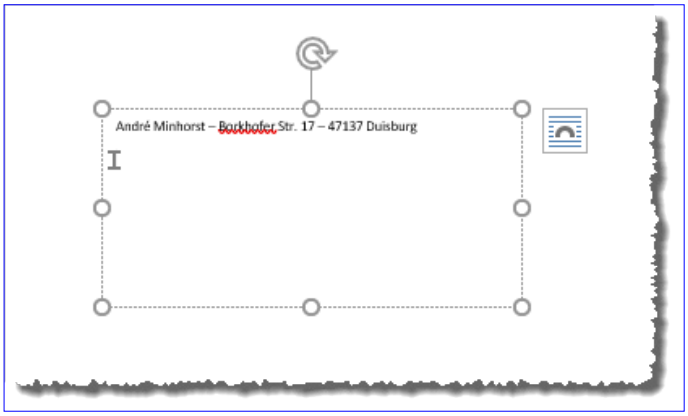



**Bild 7:** Markierung für eine Textmarke

Wenn Sie später alle Textfelder sichtbar machen wollen, können Sie einfach in einen leeren Bereich des Dokuments klicken und alle Elemente mit der Tastenkombination **Strg + A** markieren.

Das Textfeld können Sie nun gleich kopieren und als Basis für das Feld für die Betreffzeile nutzen – so brauchen Sie den Rahmen nicht wieder wie bei einem neuen Textfeld auf transparent einzustellen. Die Betreffzeile soll zwei Zeilen unterhalb des Anschriftenfeldes liegen, was wir als ca. 10 mm interpretieren. Also stellen wir den Abstand vom oberen Rand auf 10,5 cm ein, den Abstand vom linken Rand auf 2 cm und die Größe auf 1 x 12 cm. Löschen Sie den mitkopierten Text, stellen Sie die Schriftgröße auf 10 ein und fügen Sie auch diesem Textfeld einen Platzhalter hinzu – diesmal mit der Bezeichnung **Betreff**.

Rechts oben wollen wir den Briefkopf unterbringen. Dafür legen wir ein Textfeld mit einem Abstand von 2 cm vom rechten Rand und 2,5 cm vom oberen Rand und mit der Größe 5 x 5 cm an. Dazu müssen wir ein wenig rechnen, denn wir können nur Abstände von oben und links eintragen. Ein DIN A4-Blatt hat eine Breite von 19,7 cm, also müssen wir 2 cm für den Abstand von rechts und 5 cm für die Breite abziehen und landen bei einem Abstand von 12,7 cm vom linken Rand. Den Briefkopf füllen Sie mit den Absenderdaten aus und mit weiteren Daten, die Sie angeben möchten – Telefon, E-Mail, Webseite, Bankverbindung, Steuernummern et cetera.

Das Datum des Anschreibens fügen wir in einem Textfeld ein, das die Position 12,7 cm vom linken Rand und 9 cm vom oberen Rand einnimmt und eine Größe von 5 x 1 cm aufweist. Das Textfeld füllen wir mit einer weiteren Textmarke, diesmal mit der Bezeichnung **Datum** in Schriftgröße 10.

Das letzte Textfeld soll den Inhalt des Anschreibens aufnehmen. Es hat einen Abstand von 2 cm zum linken Rand und 13 cm zum oberen Rand und eine Größe von 15,7 x 12 cm. Hier fügen wir die Textmarke mit der Bezeichnung **Inhalt** ein.

An dieser Stelle der Hinweis, dass wir hier nur in Textfeldern schreiben und daher unabhängig von den Einstellungen der Seitenränder des Dokuments sind. Damit erhalten wir allerdings die Einschränkung, dass wir auch nur den Platz des dafür vorgesehenen Textfeldes für den Inhalt des Anschreibens haben. Für mehrseitige Dokumente müssten wir allerdings auch noch etwas mehr Aufwand betreiben – das soll jedoch nicht Thema dieses Artikels sein. Das Dokument sieht schließlich wie in Bild 8 aus. Schließlich speichern Sie die Word-Datei unter dem Namen **Vorlage.dotx**, wozu Sie den entsprechenden Dateityp beim Speichern auswählen müssen.

## Vorlage wieder öffnen

Wenn Sie die Dokumentvorlage nach dem Schließen nochmals öffnen wollen, um diese beispielsweise nochmals anzupassen, können Sie dies nicht einfach per Doppelklick auf die Datei **Vorlage.dotx** erledigen. Dies öffnet nämlich nicht die Dokument-

vorlge, sondern erzeugt ein neues Dokument auf Basis dieser Vorlage.

Dieses sieht dann zwar genauso aus wie die Vorlage selbst, aber wenn Sie diese speichern, stellen Sie fest, dass es sich hier um ein einfaches Word-Dokument handelt.

Wenn Sie die Vorlagendatei öffnen wollen, starten Sie zunächst Word und öffnen die Vorlagendatei dann über den **Datei öffnen**-Dialog von Word.

## Steuerelemente zum Eingeben der Briefdaten

Damit kommen wir zurück zum Visual Studio-Projekt. Diesem fügen wir nun zunächst die fertige Word-Vorlage als Ressource hinzu. Dazu öffnen Sie die Projekteigenschaften, wählen den Eintrag **Ressourcen** und legen die Word-Dokumentvorlage als neue Ressource an. Details dazu finden Sie im Artikel **Mit Ressourcen arbeiten**.

Wir machen es auf die Schnelle so, dass wir einfach die Datei **Vorlage.dotx** aus dem Windows Explorer in die Liste der Ressourcen hineinziehen (siehe Bild 9).

Sie können natürlich auch noch weitere Vorlagen anlegen, die beispielsweise für andere Familienmitglieder genutzt werden oder für geschäftliche Zwecke.

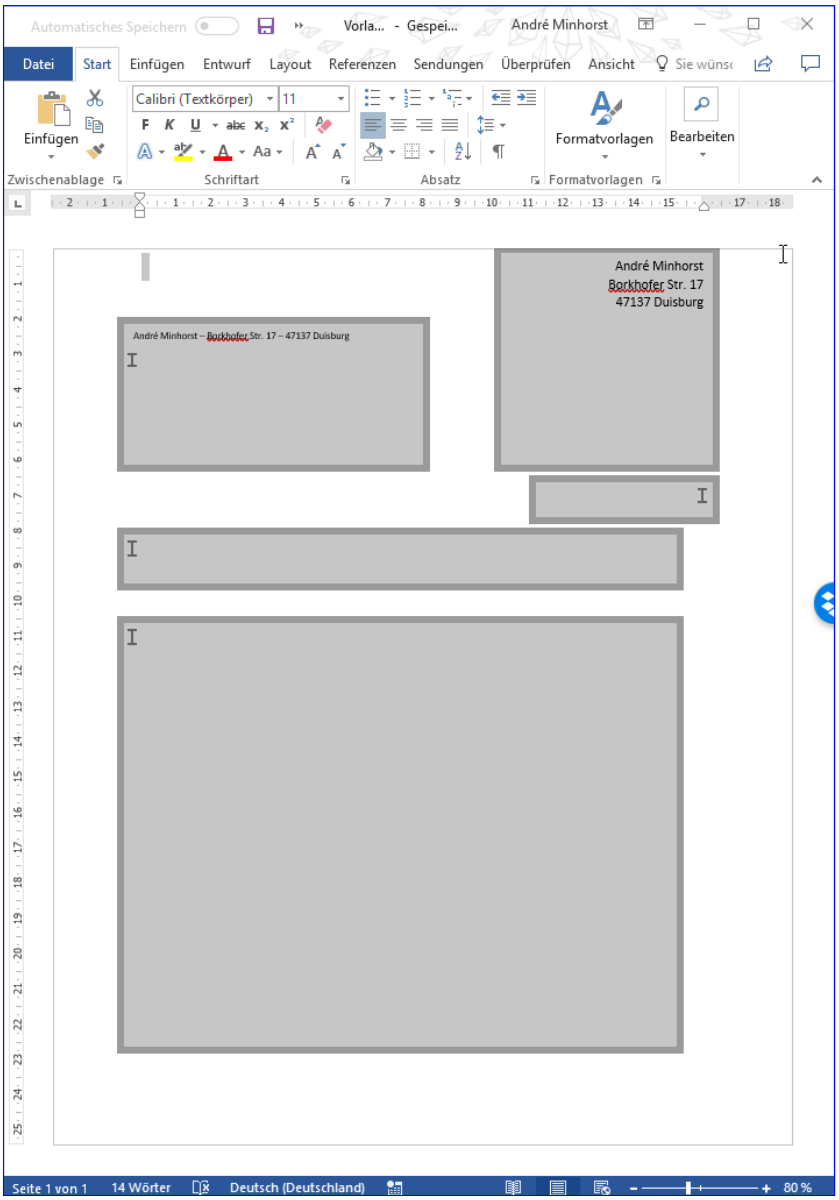



**Bild 8:** Anordnung der Textfelder

## Anzeige und Auswahl der Vorlagen

Ganz oben im Fenster wollen wir ein Kombinationsfeld zur Auswahl der Vorlage einfügen. Das Label und das Kombinationsfeld definieren wir wie folgt:

```
<ComboBox x:Name="cboVorlagen" Grid.RowSpan="2" Grid.Column="1"/>
<Button x:Name="btnGewaehlteRessourcenExportieren" Click="btnGewaehlteRessourcenExportieren_Click"
    Grid.Row="8">Gewählte Ressource exportieren</Button>
```

Damit das Kombinationsfeld mit den verfügbaren Vorlagen gefüllt wird, legen wir für das Window-Element die Ereigniseigenschaft **Loaded** an:

```
<Window x:Class="MainWindow" ...
Title="MainWindow" Height="450"
Width="525" Loaded="Window_Loaded">
```

Für diese hinterlegen wir die folgende Methode:

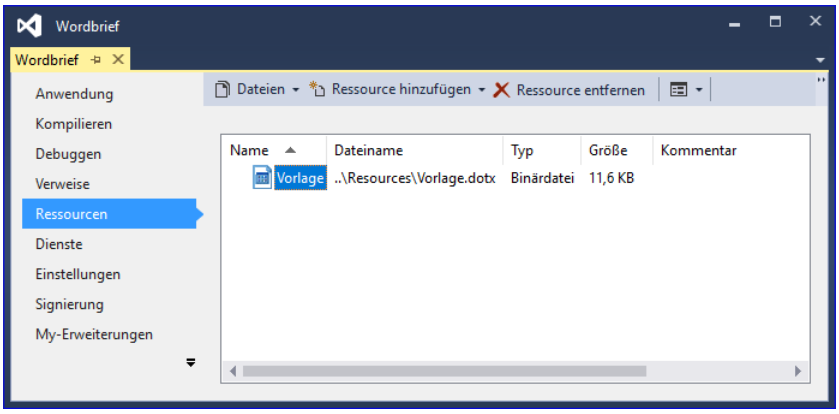



**Bild 9:** Voraussetzung: Eine Word-Dokumentvorlage als Ressource

```
Private Sub Window_Loaded(sender As Object, e As RoutedEventArgs)
    Dim objResourceManager As ResourceManager
    Dim objResourceSet As ResourceSet
    Dim strResource As String
    objResourceManager = New ResourceManager("Wordbrief.Resources", Assembly.GetExecutingAssembly())
    objResourceSet = objResourceManager.GetResourceSet(CultureInfo.CurrentCulture, True, True)
    Dim objDictionaryEntry As DictionaryEntry
    cboVorlagen.Items.Add("<Auswählen>")
    For Each objDictionaryEntry In objResourceSet
        strResource = objDictionaryEntry.Key.ToString()
        cboVorlagen.Items.Add(strResource)
    Next
    cboVorlagen.SelectedIndex = 0
End Sub
```

Die Methode erstellt ein **ResourceManager**-Objekt auf Basis der in der Anwendung befindlichen Ressourcen und referenziert die Elemente mit der Variablen **objResourceSet**. Dann fügt sie dem Kombinationsfeld **cboVorlagen** den Eintrag **<Auswählen>** hinzu und anschließend in einer Schleife über die Elemente von **objResourceSet** die Namen der verfügbaren Ressourcen. Schließlich stellt sie das Kombinationsfeld auf den Eintrag **<Auswählen>** ein.

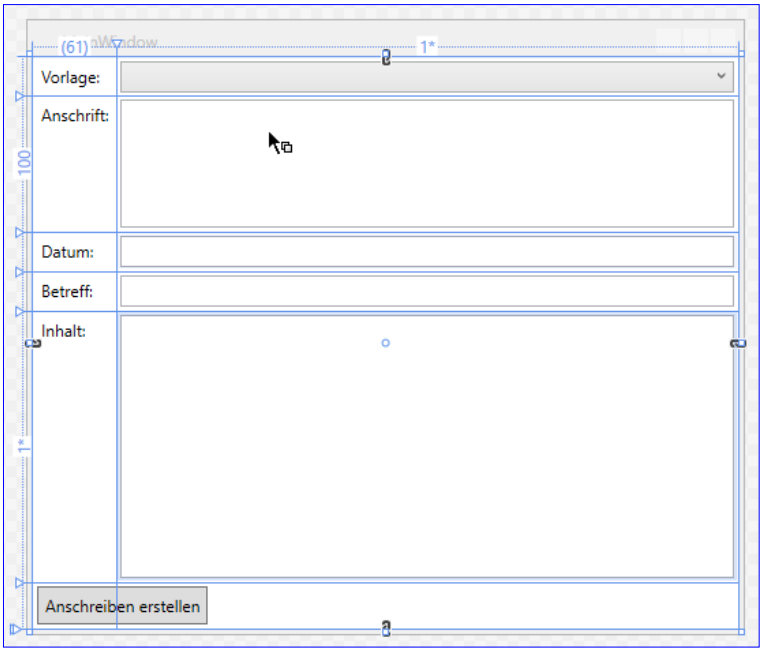



**Bild 10:** Steuerelemente zum Eingeben der Eigenschaften des Dokuments

Direkt danach legen wir weitere Steuerelemente an, und zwar die **TextBox**-Elemente **txtAnschrift**, **txtDatum**, **txtBetreff** und **txtInhalt** mit den jewei-

ligen **Label**-Elementen. Für die beiden **TextBox**-Elemente **txtAnschrift** und **txtBetreff**, die ja aus mehreren Zeilen bestehen können, stellen wir die Eigenschaft **AcceptsReturn** auf **True** ein, damit der Benutzer mit der Eingabetaste neue Zeilen anlegen kann und **TextWrapping** auf **Wrap**, damit Zeilen im Textfeld umgebrochen werden. Außerdem fügen wir in der untersten Zeile noch eine Schaltfläche namens **btnAnschreibenErstellen** hinzu (siehe Bild 10).

Der XAML-Code hierzu sieht gekürzt wie folgt aus:

```
<Window x:Class="MainWindow" ... Title="MainWindow" Height="450" Width="525" Loaded="Window_Loaded">
    <Window.Resources>...Basis-Einstellungen für die Steuerelemente ... </Window.Resources>
    <Grid>
        <Grid.ColumnDefinitions>...</Grid.ColumnDefinitions>
        <Grid.RowDefinitions>...</Grid.RowDefinitions>
        <Label>Vorlage:</Label>
        <Label Grid.Row="1">Anschrift:</Label>
        <Label Grid.Row="2">Datum:</Label>
        <Label Grid.Row="3">Betreff:</Label>
        <Label Grid.Row="4">Inhalt:</Label>
        <ComboBox x:Name="cboVorlagen" Grid.RowSpan="2" Grid.Column="1"/>
        <TextBox x:Name="txtAnschrift" Grid.Row="1" Grid.Column="1" VerticalAlignment="Stretch"
            Height="Auto" AcceptsReturn="True" TextWrapping="Wrap"></TextBox>
        <TextBox x:Name="txtDatum" Grid.Row="2" Grid.Column="1"></TextBox>
        <TextBox x:Name="txtBetreff" Grid.Row="3" Grid.Column="1"></TextBox>
        <TextBox x:Name="txtInhalt" Grid.Row="4" Grid.Column="1" VerticalAlignment="Stretch" Height="Auto"
            AcceptsReturn="True" TextWrapping="Wrap"></TextBox>
        <Button x:Name="btnAnschreibenErstellen" Click="btnAnschreibenErstellen_Click" Grid.Row="5"
            Grid.ColumnSpan="2">Anschreiben erstellen</Button>
    </Grid>
</Window>
```

## Word-Dokument auf Basis der Vorlage erstellen und füllen

Damit kommen wir zur Programmierung der Schaltfläche **btnAnschreibenErstellen**. Die dadurch ausgelöste Methode hat im Überblick die folgenden Aufgaben:

- Anlegen eines neuen Word-Dokuments auf Basis der gewählten Vorlage
- Füllen der Textmarken **Anschrift**, **Datum**, **Betreff** und **Inhalt** mit den Daten aus den Textfeldern
- Speichern des Dokuments unter dem gewählten Speicherort und Namen

Dazu kommen noch ein paar Detailaufgaben. So soll die Vorlage jeweils zum Erstellen eines Dokuments im gleichen Verzeichnis wie die **.exe**-Datei extrahiert werden, dann als Vorlage dienen und schließlich wieder gelöscht werden.

# E-Mails mit Outlook verschicken

**Wer Anwendungen mit Visual Studio programmiert, die Daten wie etwa die von Kunden verwaltet, kommt früher oder später nicht um eine Funktion zum Versenden von E-Mails herum. Da gibt es nun zwei Möglichkeiten: Sie verwenden eine eigene SMTP-Klasse, um die E-Mails zu versenden. Das klappt mit .NET-Projekten viel einfacher als etwa unter Access, weil es hier schon fertige Klassen für diesen Anwendungszweck gibt. Allerdings werden die gesendeten Mails dann nicht in Outlook im Ordner »Gesendete Elemente« gespeichert. Deshalb schauen wir uns in diesem Artikel den Versand von Mails per Outlook an. In einem anderen Artikel gehen wir dann auf die Verwendung einer SMTP-Klasse ein.**

## Beispielanwendung

Wir wollen als Beispiel für diesen Artikel eine kleine WPF-Anwendung programmieren, mit der Sie die Daten des E-Mail-Empfängers sowie Betreff, Inhalt und weitere Informationen eingeben können. Eine Schaltfläche soll dann die eingegebenen Daten an eine Outlook-Instanz schicken und die E-Mail versenden.

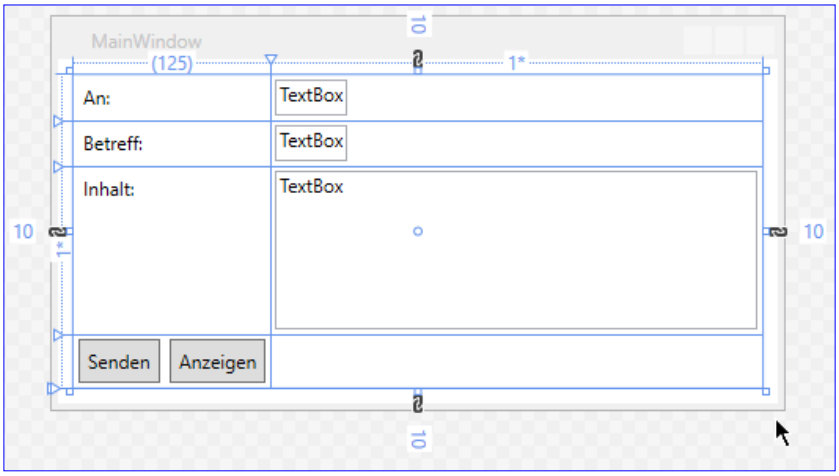



**Bild 1:** Einfaches Formular zum Versenden von E-Mails

Also legen Sie zunächst eine WPF-Anwendung in der von Ihnen bevorzugten Sprache an (also etwa VB oder C#). Das Fenster **MainWindow.xaml** ist dann auch gleich das Element, das wir mit den benötigten Steuerelementen versehen wollen. Dieses gestalten wir wie in Bild 1.

Der dazu notwendige XAML-Code sieht gekürzt so aus (es gibt noch einige Steuerelement-spezifische Eigenschaftsdefinitionen, die wir nicht abbilden):

```
<Window x:Class="MainWindow" ... Title="MainWindow" Height="250" Width="466">
    <Grid HorizontalAlignment="Stretch" Margin="10" VerticalAlignment="Stretch" ClipToBounds="true" >
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="Auto"/>
            <RowDefinition/>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="Auto"/>
        </Grid.RowDefinitions>
```

```
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="Auto"/>
            <ColumnDefinition Width="*"/>
        </Grid.ColumnDefinitions>
        <Label x:Name="label1" Content="An:" />
        <Label x:Name="label2" Content="Betreff:" Grid.Row="1"/>
        <Label x:Name="label3" Content="Inhalt:" Grid.Row="2"/>
        <TextBox x:Name="txtAn" Grid.Column="1" HorizontalAlignment="Stretch"/>
        <TextBox x:Name="txtBetreff" Grid.Row="1" Grid.Column="1" HorizontalAlignment="Stretch"/>
        <TextBox x:Name="txtInhalt" Grid.Row="2" TextWrapping="Wrap" Grid.Column="1" VerticalAlignment="Stretch"
            HorizontalAlignment="Stretch" Height="Auto" AcceptsReturn="True"/>
        <StackPanel Orientation="Horizontal" Grid.Row="3">
            <Button x:Name="btnSenden">Senden</Button>
            <Button x:Name="btnAnzeigen">Anzeigen</Button>
        </StackPanel>
    </Grid>
</Window>
```

Dadurch, dass wir für die dritte Zeile den Wert der Eigenschaften **Height** und **Width** auf **Auto** eingestellt haben, wird diese Zeile gemeinsam mit dem Fenster vergrößert. Damit wir im Textfeld **txtInhalt** mehrzeilige Texte eingeben können, haben wir die Eigenschaft **TextWrapping** auf **Wrap** und **AcceptsReturn** auf **True** eingestellt.

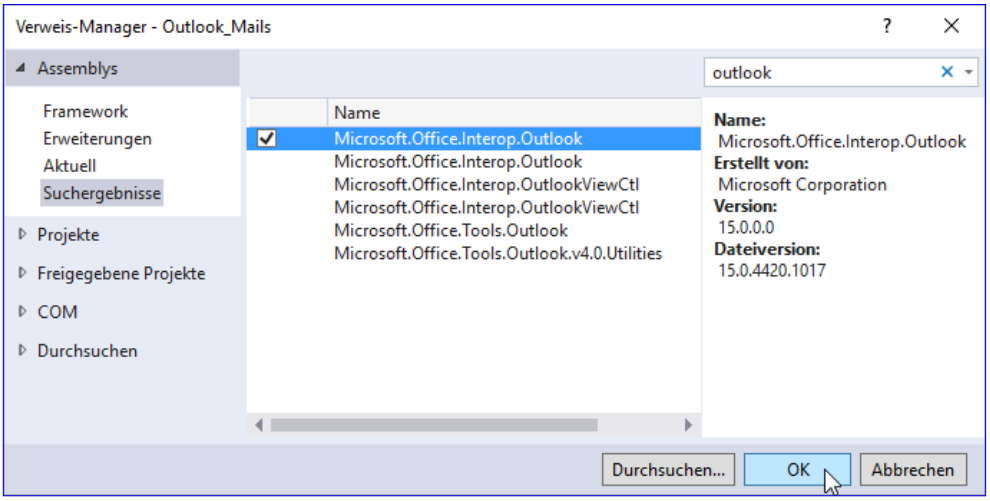



**Bild 2:** Erweitern der Verweise des Projekts um die Outlook-Bibliothek

## Verweis auf Outlook hinzufügen

Damit wir Outlook von einer WPF-Anwendung aus programmieren können, fügen wir den entsprechenden Verweis auf die Outlook-Bibliothek **Microsoft.Office.Interop.Outlook** hinzu. Dazu öffnen Sie den **Verweise-Manager** über den Menüeintrag **Projekt|Verweis hinzufügen...** und suchen im nun erscheinenden Dialog nach dem Wort **Outlook**.

Selektieren Sie den Eintrag **Microsoft.Office.Outlook.Interop** (siehe Bild 2) und schließen Sie den Dialog mit **OK**. Sollten Sie, wie im obigen Dialog der Fall, mehrere gleichnamige Einträge vorfinden, handelt es sich um verschiedene Versionen. Wählen Sie dann die aktuellere Version aus.

Dann statten wir die Schaltfläche **cmdSenden** mit der Ereigniseigenschaft **Click** aus. Nach der Eingabe des Gleichheitszeichens können Sie mit der Tabulator-Taste automatisch die entsprechende Ereignismethode anlegen, indem Sie aus der dann erscheinenden Auswahlliste den Eintrag **<Neuer Ereignishandler>** auswählen:

```
<Button x:Name="btnSenden" Click="btnSenden_Click">Senden</Button>
```

In der Code behind-Klasse erscheint dann die folgende Methode:

```
private void btnSenden_Click(object sender, RoutedEventArgs e) {

}
```

In der Klasse fügen wir nun zunächst eine **Imports**-Anweisung hinzu, mit der wir den Outlook-Namespace verfügbar machen:

```
using Outlook = Microsoft.Office.Interop.Outlook;
```

Damit greifen wir über das Schlüsselwort **Outlook** auf Elemente dieses Namespaces zu.

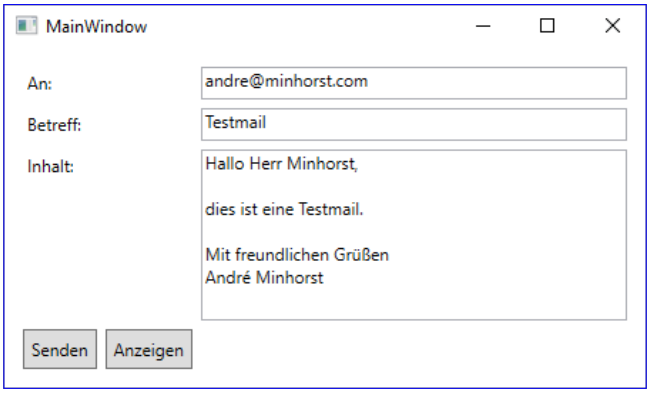



**Bild 3:** Beispielfenster zum Versenden von E-Mails

## Textfelder mit Beispiel füllen

Bevor wir uns auf die Outlook-Programmierung stürzen, legen wir noch eine Ereignismethode für das Ereignis **Loaded** des **Window**-Elements an. Diese soll die drei Textfelder mit Beispieltexten füllen, damit das Fenster beim Öffnen gleich wie in Bild 3 aussieht:

```
private void Window_Loaded(object sender, RoutedEventArgs e) {
    txtAn.Text = "andre@minhorst.com";
    txtBetreff.Text = "Testmail";
    txtInhalt.Text = "Hallo Herr Minhorst," +  Environment.NewLine + Environment.NewLine + "dies ist eine Testmail."
        + Environment.NewLine + Environment.NewLine + "Mit freundlichen Grüßen" + Environment.NewLine + "André Minhorst";
}
```

## Outlook-Mail erzeugen

Danach kümmern wir uns um den Code, der die Mail mit den angegebenen Daten versenden soll. Dazu füllen wir die bereits angelegte Ereignismethode **btnSenden_Click** wie folgt:

```
private void btnSenden_Click(object sender, RoutedEventArgs e) {
    Outlook.Application objOutlook;
    Outlook.MailItem objMailItem;
    objOutlook = new Outlook.Application();
```

```
    objMailItem = objOutlook.CreateItem(Outlook.OlItemType.olMailItem);
    objMailItem.To = txtAn.Text;
    objMailItem.Subject = txtBetreff.Text;
    objMailItem.Body = txtInhalt.Text;
    objMailItem.Send();
}
```

Wir erstellen also ein Objekt des Typs **Microsoft.Office.Interop.Outlook.Application**, den wir hier aufgrund unserer **Using**-Anweisung vereinfacht mit **Outlook.Application** referenzieren können, und speichern einen Verweis darauf in der Variablen **objOutlook**. Die zweite Variable namens **objMailItem** mit dem Datentyp **MailItem** nutzen wir, um ein mit der **CreateItem**-Methode von **objOutlook** erzeugtes **MailItem**-Objekt zu referenzieren. Dies dient dem Zweck, seine Eigenschaften **To**, **Subject** und **Body** mit den Inhalten der drei Textfelder **txtAn**, **txtBetreff** und **txtInhalt** zu füllen. Die letzte Anweisung heißt **Send** und sorgt dafür, dass die Mail direkt verschickt wird. Wenn Sie die Anwendung nun starten und die Mail mit einem Klick auf die Schaltfläche **cmdSenden** verschicken, können Sie diese anschließend im Ordner **Gesendete Objekte** von Outlook einsehen.

## E-Mail erst anzeigen

Etwas transparenter ist das Ganze natürlich noch, wenn Sie die E-Mail betrachten können, bevor Sie diese durch einen Klick auf die dafür vorgesehene Schaltfläche absenden. Dazu legen wir für die Schaltfläche **cmdAnzeigen** ebenfalls die **Click**-Eigenschaft an:

```
<Button x:Name="btnAnzeigen" Click="btnAnzeigen_Click">Anzeigen</Button>
```

Den Code dieser Methode brauchen wir gegenüber der zuvor beschriebenen nur leicht zu variieren:

```
private void btnSenden_Click(object
sender, RoutedEventArgs e) {
    ...
    objMailItem.Send();
}
```

Hier rufen wir nämlich statt der **Send**-Methode die **Display**-Methode auf. Dies zeigt beim Ausprobieren die Mail aus Bild 4 an. Nach dem Betrachten schickt man die E-Mail dann mit einem Klick auf die **Senden**-Schaltfläche ab.

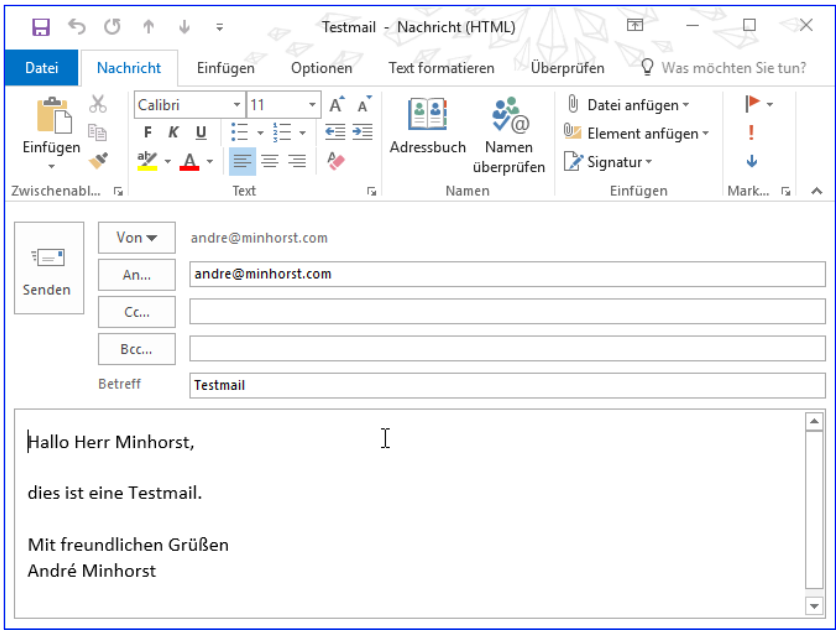



**Bild 4:** E-Mail vor dem Versenden ansehen

## Anhänge hinzufügen

Nun wollen wir dem Benutzer ermöglichen, einen oder mehrere Dateien zur

# Absender von Outlook-Mails einstellen

**Im Artikel »E-Mails mit Outlook verschicken« haben wir gezeigt, wie Sie von einer .NET-Anwendung aus die Daten für eine E-Mail zusammenstellen und diese dann per Outlook verschicken können – unter anderem mit Empfänger, CC, BCC, Anlagen oder Priorität. Was noch fehlt, ist die Absenderadresse. Diese braucht man grundsätzlich nicht explizit einzustellen, denn Outlook verwendet dann die Adresse des Standardkontos von Outlook (vorausgesetzt, Sie haben mehrere Konten). Wenn Sie jedoch eine Mail mit einem anderen Konto als dem Standardkonto verschicken wollen, wird es kompliziert. Die Lösung für diese Aufgabe finden Sie im vorliegenden Artikel.**

## Verschiedene Outlook-Konten

Wenn Sie wie im **Artikel E-Mails mit Outlook verschicken** ein **MailItem**-Objekt erzeugen und seine Eigenschaften wie **To**, **CC**, **BCC**, **Subject** oder **Body** füllen, werden Sie vergeblich ein Feld etwa namens **From** suchen. Dieses Feld gibt es nämlich nicht – Sie können also nicht einfach irgendeine Absenderadresse zu Ihrer E-Mail hinzufügen.

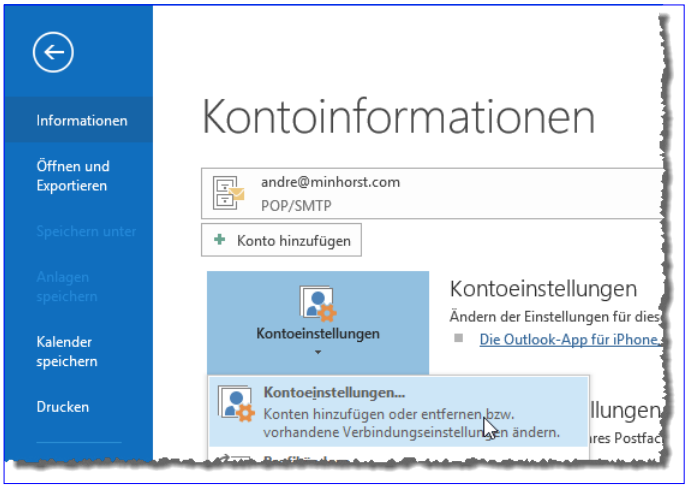



**Bild 1:** Anzeigen der Kontoeigenschaften

Woher weiß Outlook dann aber, welche Absenderadresse überhaupt zu verwenden ist? Ganz einfach: Es gibt in Outlook ein oder mehrere Konten, von denen eines als Standard festgelegt ist. Wann immer Sie direkt von Outlook aus eine neue E-Mail erstellen, wird die zum Standardkonto gehörende E-Mail-Adresse als Absenderadresse eingestellt. Wo finden Sie diese Konten und Einstellungen? Dazu öffnen Sie Outlook, klicken auf den Ribbon-Tab **Datei** und dann im Bereich **Informationen** auf die Schaltfläche **Kontoeinstellungen|Kontoeinstellungen...** (siehe Bild 1).

Hier finden Sie dann wie in Bild 2 die angelegten Outlook-Konten vor. Hier können Sie beispielsweise doppelt auf einen der Einträge klicken, um Eigenschaften wie den SMTP-Server oder die E-Mail-Adresse anzupassen oder ein anderes als das aktuelle Konto als Standardkonto festzulegen. Haben Sie nur ein einziges Outlook-Konto, was bei Verwendung eines Rechners für private und geschäftliche Zwecke unwahrscheinlich ist, haben Sie natürlich keine Wahl bei der Absenderadresse einer E-Mail. Wir gehen jedoch davon aus, dass viele Benutzer mindestens zwei Outlook-Konten in Outlook verwalten.

## Einstellen des Absenderkontos

Outlook wählt also standardmäßig die E-Mail-Adresse des als Standard festgelegten Kontos beim Anlegen neuer E-Mails. Auf die gleiche Weise verfährt Outlook, wenn Sie wie im oben genannten Artikel eine neue E-Mail von einer .NET-Anwendung aus erstellen. Die einfachste Weise, das Absenderkonto zu ändern, erhalten Sie, wenn Sie die neu erstellte E-Mail mit der Display-

Methode anzeigen (siehe Bild 3). Das ist aber natürlich keine Option, wenn Sie die E-Mail direkt nach der Eingabe der Daten wie im Artikel **E-Mails mit Outlook verschicken** versenden wollen oder gar automatisiert E-Mails mit vorgefertigtem Inhalt an einen oder mehrere Empfänger gehen sollen. In diesem Fall benötigen Sie eine Möglichkeit, den Absender per Code einzustellen oder diesen zumindest gleich beim Zusammenstellen der Mail in der Anwendung festzulegen.

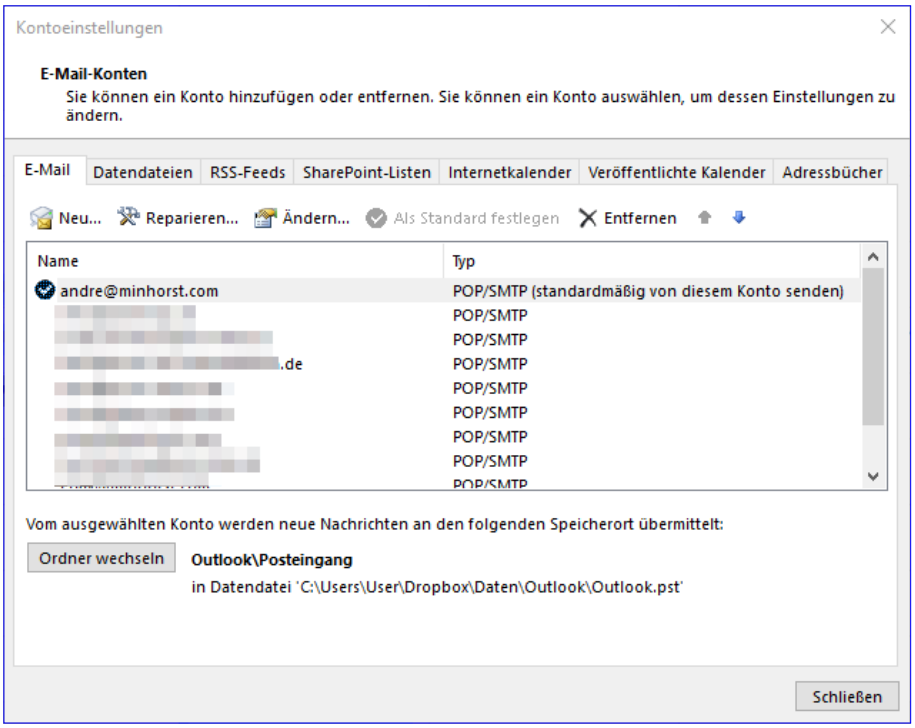



**Bild 2:** E-Mail-Konten unter Outlook

## Absenderadresse festlegen

Wir werden uns zwei Varianten ansehen, um die Absenderadresse festzulegen. Bei der ersten geben wir die Absenderadresse einfach in ein Textfeld ein und versuchen dann beim Zusammenstellen der E-Mail, das betroffene Outlook-Konto zu ermitteln und als Absenderkonto einzustellen. Im zweiten Anlauf wollen wir dann vorab alle Outlook-Konten auslesen und die E-Mail-Adressen per Kombinationsfeld zur Auswahl bereitstellen.

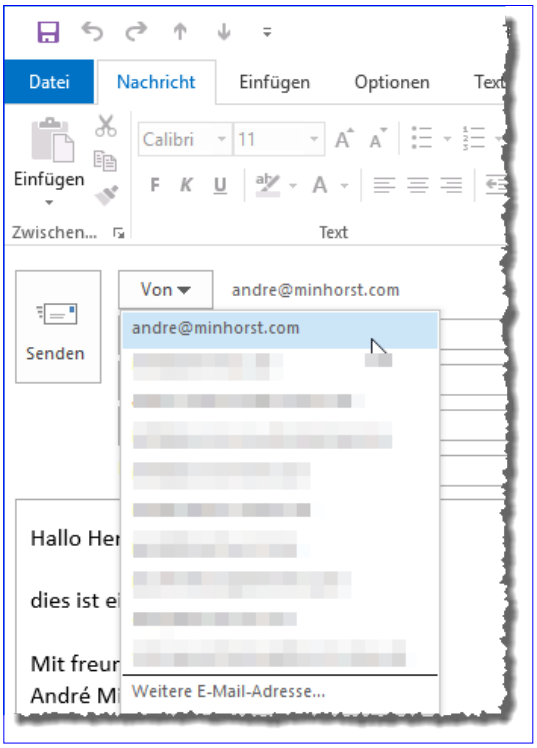



**Bild 3:** Auswahl des Absenders in der E-Mail

Wir bauen die Beispiellösung auf der Anwendung auf, die wir im Artikel **E-Mails mit Outlook verschicken** beschrieben haben. Dazu fügen wir zunächst ein Textfeld namens **txtAbsender** zum Fenster **MainWindow.xaml** hinzu:

```
<TextBox x:Name="txtAbsender" Grid.Column="1"></TextBox>
```

Damit dieses gleich beim Öffnen des Fensters gefüllt wird, fügen wir eine Zeile zur Methode **Window_Loaded** hinzu:

```
private void Window_Loaded(object sender, RoutedEventArgs e) {
    ...
    txtAbsender.Text = "info@amvshop.de";
    ...
}
```

## Absenderadresse per Code einstellen

Das war der einfache Teil. Nun müssen wir, wenn der Benutzer auf die **Senden**- oder **Anzeigen**-Schaltfläche klickt, neben den bereits bekannten Informationen, die wir der E-Mail einfach über entsprechende Eigenschaften zuweisen können, noch die Absenderadresse einstellen. Dazu deklarieren wir zunächst eine Objektvariable namens **account** mit dem Typ **Microsoft.Office.Interop.Account**. Nach dem Erstellen des **MailItem**-Objekts wollen wir gleich den Absender festlegen. Dazu benötigen wir noch eine weitere Variable namens **bolAccountEingestellt** mit dem Typ **Boolean**, deren Funktion wir gleich erläutern.

In einer **foreach**-Schleife durchlaufen wir nun alle Outlook-Konten, die wir über die Auflistung **outlook.Session.Accounts** erhalten. Das aktuelle Konto landet dabei in der Laufvariablen **account**. Für dieses prüfen wir zunächst in einer **if**-Bedingung, ob es sich um ein POP3-Konto handelt. Falls ja, vergleichen wir die Absenderadresse des Kontos, die wir über die Eigenschaft **DisplayName** ermitteln, mit der von uns gewünschten Absenderadresse. Stimmen beide überein, weisen wir das aktuell in **account** befindliche Konto der Eigenschaft **SendUsingAccount** zu und stellen die Variable **bolAccountEingestellt** auf **true** ein. Auf diese Weise durchlaufen wir alle Einträge der **Accounts**-Auflistung der aktuellen Session, bis wir ein passendes Konto gefunden haben – oder eben nicht: In diesem Fall verlassen wir die Schleife, ohne die Variable **bolAccountEingestellt** auf den Wert **true** gesetzt zu haben.

Die folgende **if**-Bedingung prüft, ob **bolAccountEingestellt** den Wert **false** aufweist und damit gegebenenfalls kein Account zur Absenderadresse gefunden wurde. In diesem Fall erscheint eine Meldung, die den Benutzer darauf hinweist, dass das gewünschte Konto zur Absenderadresse nicht existiert, und schlägt ihm zwei Alternativen vor: entweder, mit dem Standardkonto von Outlook zu senden oder den Vorgang abzubrechen:

```
private void btnAnzeigen_Click(object sender, RoutedEventArgs e) {
    Outlook.Application outlook;
    Outlook.MailItem mailItem;
    Boolean bolAccountEingestellt = false;
    outlook = new Outlook.Application();
    mailItem = outlook.CreateItem(Outlook.OlItemType.olMailItem);
    foreach (Outlook.Account account in outlook.Session.Accounts) {
        if (account.AccountType == Outlook.OlAccountType.olPop3) {
            if (account.DisplayName == txtAbsender.Text) {
                mailItem.SendUsingAccount = account;
                bolAccountEingestellt = true;
                break;
            }
        }
    }
    if (bolAccountEingestellt == false) {
        if (MessageBox.Show("Für die angegebene Absenderadresse '" + txtAbsender.Text
              + "' existiert kein Outlook-Konto. Mit Standardkonto senden?", "Konto fehlt",
              MessageBoxButton.YesNo) == MessageBoxResult.No) {
            return;
```

# E-Mails ohne Outlook versenden

**Im Artikel »E-Mails mit Outlook verschicken« haben wir gezeigt, wie Sie von einer .NET-Anwendung aus die Daten für eine E-Mail zusammenstellen und diese dann per Outlook verschicken können. Was aber, wenn der Benutzer kein Outlook auf dem Rechner hat? Für diesen Fall bietet .NET eigene Bibliotheken. Damit können Sie E-Mails ohne Zugriff auf Outlook verschicken. Dieser Artikel zeigt, wie dies gelingt.**

Wir wollen in diesem Artikel das gleiche Fenster verwenden, mit dem wir auch in der Beispiellösung zum Artikel **E-Mails mit Outlook verschicken** gearbeitet haben (siehe Bild 1). Einziger Unterschied: Die Schaltfläche **btnAnzeigen** fehlt, denn wir arbeiten ja nicht mit Outlook und können die Mail vor dem Versenden auch nicht mehr separat anzeigen. Dafür haben wir eine Schaltfläche namens Konfiguration hinzugefügt. Diese soll einen weiteren Dialog öffnen, der die Konfigurationsdaten für den Mailserver speichert.

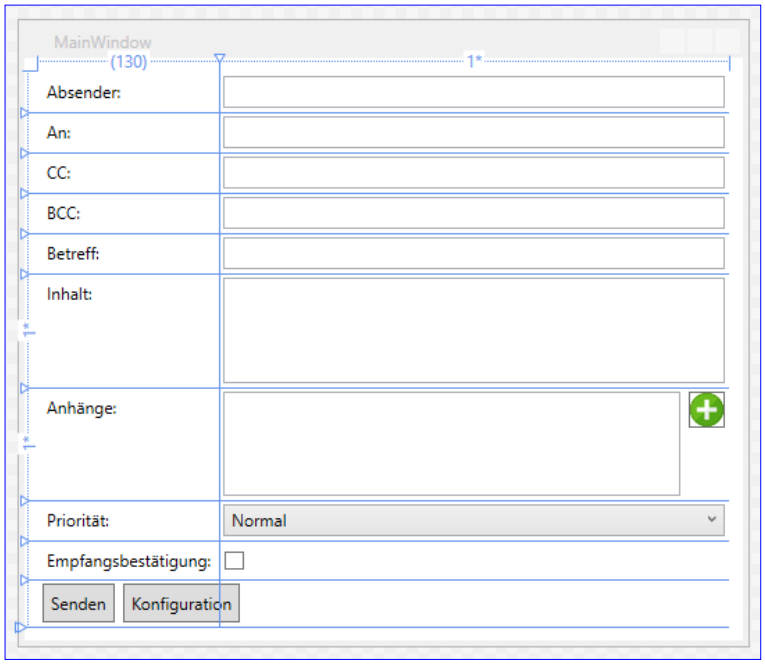



**Bild 1:** Fenster zur Eingabe der Mail-Daten

Für die Konfigurationsdaten legen wir einige Einträge im Bereich **Einstellungen** des aktuellen Projekts an, und zwar die folgenden (siehe Bild 2):

- **Host**: Name/IP des SMTP-Servers
- **Port**: Port des SMTP-Servers
- **Username**: Benutzername für die Anmeldung am SMTP-Server
- **Password**: Kennwort für die Anmeldung am SMTP-Server

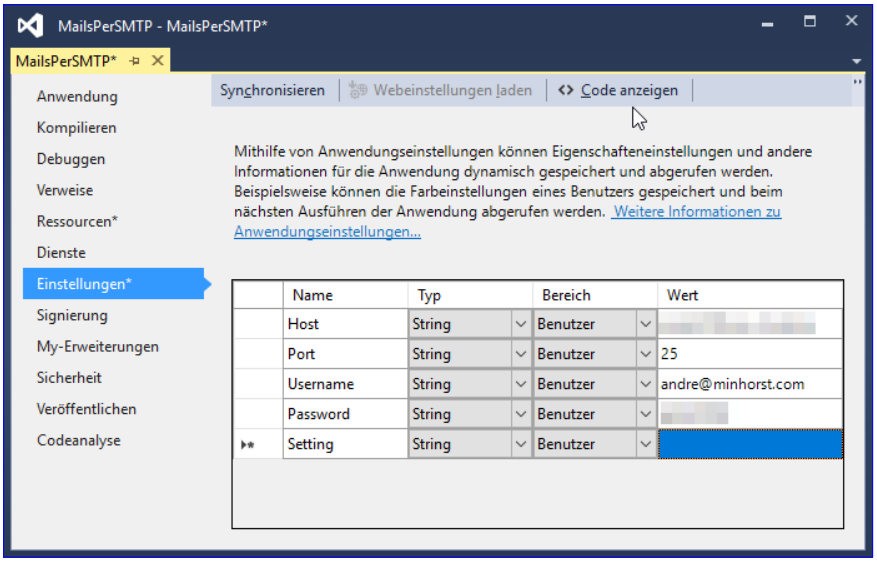



**Bild 2:** Anlegen der Konfigurationsdaten

Wenn Sie die Eigenschaften im Bereich **Einstellungen** angelegt haben und diese speichern, erstellt Visual Studio eine Klasse namens **Properties.Settings**. Diese können Sie dann initialisieren und damit direkt die angelegten Eigenschaften ansprechen, also etwa **setting.User-**

**name** – mehr dazu weiter unten. Die Ereignismethode **Window_Load** füllt die Felder **txtAbsender**, **txtAn**, **txtBetreff** und **txtInhalt** mit Beispieldaten. Dies war beim Testen des Beispiels hilfreich, Sie werden diese Felder jedoch auf andere Weise füllen wollen – vorausgesetzt, Sie wollen überhaupt Mails über ein Fenster versenden und nicht direkt etwa mit den Daten aus einer Datenbank.

### Erstellen und Senden der E-Mail

Die durch die Schaltfläche **btnSenden** ausgelöste Ereignismethode stellt die E-Mail zusammen und verschickt diese dann. Da wir hier nicht Outlook als Mail-Client verwenden, sondern die Klasse **MailMessage**, benötigen wir etwas mehr Code für die Durchführung der Aufgabe. Das liegt vorrangig daran, dass wir die Daten des SMTP-Servers sowie die Benutzerdaten erst noch angeben müssen, die ja bei Benutzung von Outlook als Mail-Client bereits im jeweiligen Outlook-Konto gespeichert sind. Um die nachfolgend verwendeten Klassen nutzen zu können, benötigen wir noch Verweise auf die folgenden Namespaces:

```
using System.Net.Mail;
using System.Net;
```

Die Methode deklariert zunächst einige Variablen und initialisiert dann ein neues Objekt auf Basis der Klasse **MailMessage**:

```
private void btnSenden_Click(object sender, RoutedEventArgs e) {
    MailMessage mailMessage;
    MailAddress mailAddress;
    SmtpClient smtpClient;
    NetworkCredential networkCredentials;
    Attachment attachment;
    Properties.Settings settings;
    mailMessage = new MailMessage();
```

Dann beginnt die Methode, die Eigenschaften des neuen **MailMessage**-Objekts zu füllen. Als Absender geben wir für die Eigenschaft **From** ein Objekt des Typs **MailAddress** an, dem wir beim Initialisieren den Inhalt des Textfeldes **txtAn** übergeben:

```
    mailAddress = new MailAddress(txtAn.Text);
    mailMessage.From = mailAddress;
```

Die Empfängeradresse weisen wir mit der **Add**-Methode der Eigenschaft **To** zu. Betreff und Inhalt landen wie auch bei Outlook in den Eigenschaften **Subject** und **Body**:

```
    mailMessage.To.Add(txtAn.Text);
    mailMessage.Subject = txtBetreff.Text;
    mailMessage.Body = txtInhalt.Text;
```

Die Anlagen werden auch etwas anders zur E-Mail hinzugefügt wir unter Verwendung von Outlook. Wir können der **Attachments**-Auflistung hier nicht einfach den Namen der hinzuzufügenden Datei zuweisen, sondern müssen erst ein Objekt des Typs

# Mit Ressourcen arbeiten

**In einer .NET-Anwendung können Sie auch ohne Verwendung einer Datenbank Dateien und andere Daten ablegen. Dazu stellt Microsoft die sogenannten Ressourcen (englisch Resources) zur Verfügung. Dabei handelt es sich um die Möglichkeit, Dateien, Texte oder andere Daten entweder über die Benutzeroberfläche von Visual Studio zur Anwendung hinzuzufügen und sie per Code auszulesen und gegebenenfalls auch wieder ins Dateisystem zu kopieren. Im konkreten Fall wollen wir eine Word-Dokumentvorlage in der Anwendung vorhalten, um diese bei Bedarf ins Dateisystem zu kopieren und als Vorlage zu nutzen.**

## Beispiel Word-Dokumentvorlage

Anlass zu diesem Artikel ist ein weiterer Artikel namens **Briefe mit Word erstellen**, in dem wir auf Basis einer Word-Dokumentvorlage ein neues Word-Dokument erstellen wollen.

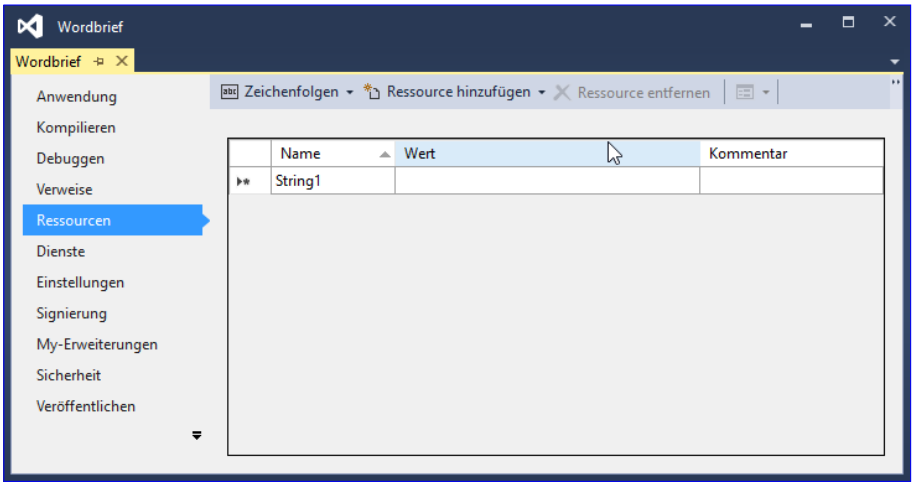



**Bild 1:** Der Dialog zum Verwalten der Ressourcen über Visual Studio

Für den Fall, dass der Benutzer die Anwendung frisch auf dem Rechner installiert hat, soll die Dokumentvorlage vor dem erstmaligen Erstellen von der Anwendung ins Dateisystem kopiert werden, damit der Code zum Erstellen des Word-Dokuments auf Basis dieser Vorlage auf die Vorlage zugreifen kann.

## Ressourcen in Visual Studio verwalten

Der einfachste Weg, die Ressourcen zu verwalten, findet sich in der Benutzeroberfläche von Visual Studio. Wenn Sie ein Projekt angelegt haben, klicken Sie doppelt auf den Eintrag **My Project** im Projektmappen-Explorer, um die Eigenschaften der Anwendung anzuzeigen.

Diesen Bereich öffnen Sie auch mit dem Menübefehl **Projekt|Eigenschaften**. Hier wechseln Sie dann durch einen Klick auf den Eintrag **Ressourcen** in der linken Leiste zum gleichnamigen Bereich. In einem frischen Projekt ist die Liste der Ressourcen noch leer (siehe Bild 1).

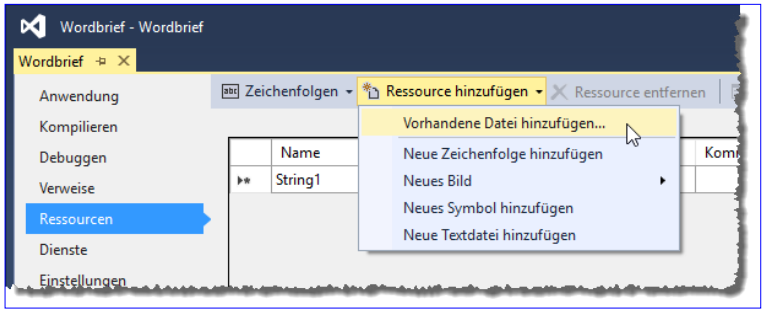



**Bild 2:** Hinzufügen einer Datei als Ressource

Um eine Datei als Ressource hinzuzufügen, klicken Sie auf den **Nach unten**-Pfeil der Schaltfläche **Ressource hinzufügen** im oberen Bereich und wählen im nun erscheinenden Menü den Eintrag **Vorhandene Datei hinzufügen...** aus (siehe Bild 2).

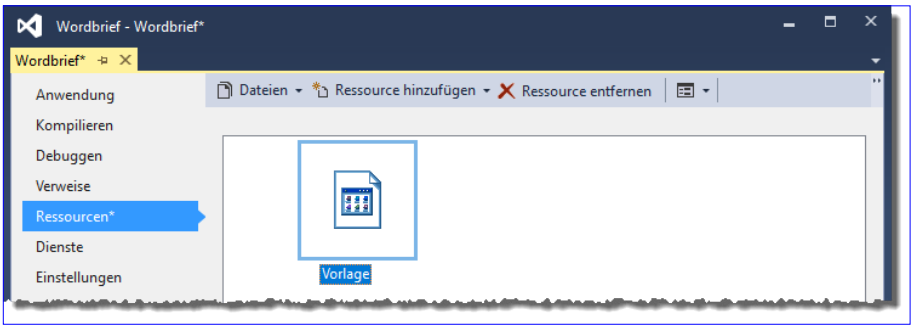



**Bild 3:** Anzeige der neuen Ressource in den Projekteigenschaften

Im folgenden Dialog **Vorhandene Datei zu Ressourcen hinzufügen** wählen Sie nun die hinzuzufügende Datei aus, in unserem Fall eine Datei namens **Vorlage.dotx**.

Nach dem Hinzufügen zeigt der **Ressourcen**-Bereich die neue Ressource wie in Bild 3 an.

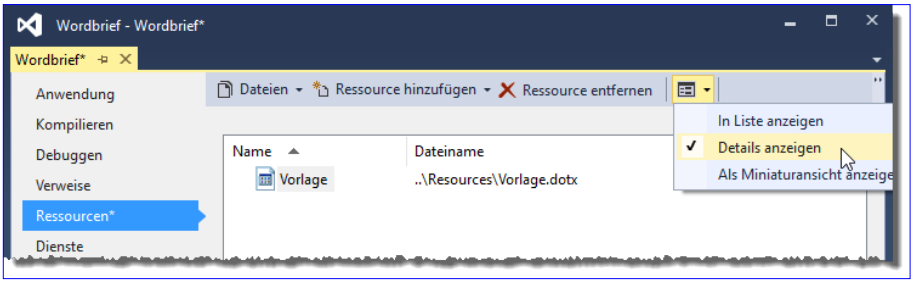



**Bild 4:** Detaillierte Liste der Ressourcen

Diese Ansicht ist nicht besonders aussagekräftig, deshalb wechseln wir wie in Bild 4 über die Auswahl der Ansichten zur Ansicht **Details anzeigen**. Damit befindet sich die Ressource bereits in der Anwendung. Sie müssen nun noch die Änderungen speichern, am einfachsten mit der Tastenkombination **Strg + S**.

## Ressource per Projektmappen-Explorer hinzufügen

Nach dem Hinzufügen der ersten Ressource zeigt der Projektmappen-Explorer einen Ordner für die Ressourcen an (siehe Bild 5). Dieser bietet auch ein Kontextmenü, über dessen Eintrag **Hinzufügen|Neues Element...** Sie den Dialog **Neues Element hinzufügen** öffnen. Hier können Sie allerdings nicht beliebige Dateien hinzufügen, sodass der Weg über den Ressourcen-Bereich in den Projekteigenschaften die einfachere Variante ist.

## Zur Laufzeit mit Ressourcen arbeiten

Nun ist das primäre Ziel, diese Ressource zur Laufzeit aus der Anwendung ins Dateisystem zu kopieren. Dies wollen wir als Nächstes realisieren. Damit erschöpfen sich die Möglichkeiten auch: Die als Ressource

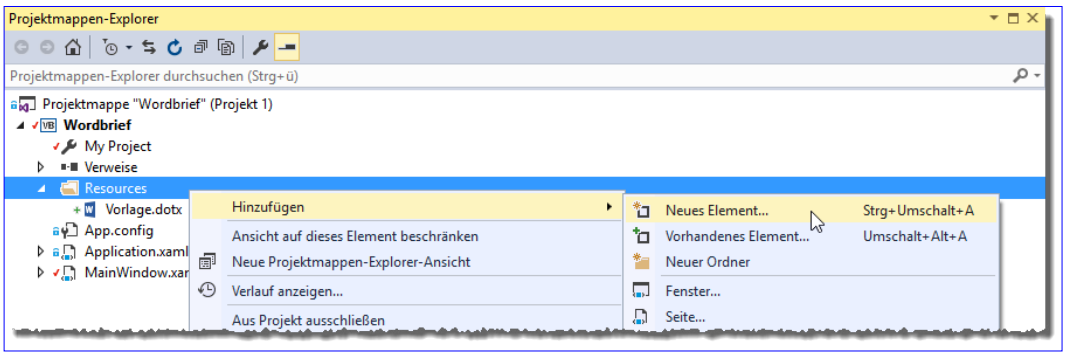



**Bild 5:** Ressource über den Projektmappen-Explorer hinzufügen

hinzugefügten Dateien werden nämlich beim Kompilieren in die **.exe**-Datei integriert. Sie können also nicht etwa zur Laufzeit weitere Elemente zu den Ressourcen einer Anwendung hinzufügen.

## Schaltfläche zum Exportieren der Ressource

Für unsere erste Aufgabe fügen wir dem Fenster **MainWindow.xaml** eine Schaltfläche namens **btnRessourceExportieren** hinzu und legen für diese eine Ereignismethode an, die durch das Ereignis **Click** ausgelöst wird. Die Definition der Schaltfläche sieht so aus:

```
<Button x:Name="btnRessourceExportieren"
Click="btnRessourceExportieren_Click">
Ressource exportieren</Button>
```

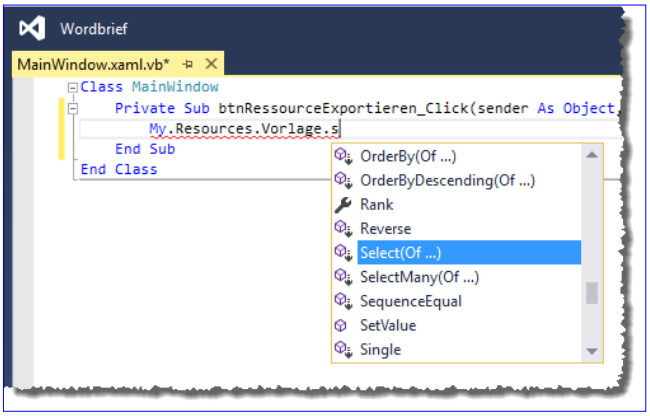



**Bild 6:** Zugriff auf die Ressource per Code

Wenn Sie nun zum Code-Editor für die Klasse **MainWindow.xaml.vb** wechseln, können Sie über **My.Resources** auf alle in den Ressourcen gespeicherten Elemente zugreifen, also auch auf unser Element mit der Bezeichnung **Vorlage**. Dieses bietet wiederum eine Reihe von Eigenschaften und Methoden an, unter denen sich aber leider nicht direkt eine Eigenschaft namens **SaveAs** findet (siehe Bild 6).

Wie bekommen wir die als Ressource gespeicherte Datei **Vorlage.dotx** nun dennoch in unser Dateisystem? Dazu brauchen wir die Methode lediglich wie folgt zu erweitern:

```
private void btnRessourceExportieren_Click(object sender, RoutedEventArgs e) {
    byte[] vorlage;
    vorlage = Properties.Resources.Vorlage;
    File.WriteAllBytes(AppDomain.CurrentDomain.BaseDirectory + "\\Vorlage.dotx", vorlage);
}
```

Die erste Zeile deklariert ein Byte-Array, die zweite weist diesem den Inhalt der Ressource zu, die wir mit **Properties.Resources.Vorlage** referenzieren. Die dritte speichert die Ressource unter dem Dateinamen **Vorlage.dotx** im aktuellen Verzeichnis der **.exe**-Anwendung.

## Zielverzeichnis definieren

Nun wollen wir dem Benutzer noch die Möglichkeit geben, die Ressource in einem von ihm ausgewählten Verzeichnis zu speichern. Dazu legen wir eine zweite Schaltfläche namens **btnRessourceExportierenAls** an, für die wir die folgende Methode hinterlegen:

```
private void btnRessourceExportierenAls_Click(object sender, RoutedEventArgs e) {
    Byte[] vorlage;
    SaveFileDialog saveFileDialog;
```

# Brief mit Word erstellen

**Heute war es wieder soweit: Ich musste ein Anschreiben erstellen. Heutzutage geht zwar vieles per E-Mail, aber hier und da wird doch noch nach einem korrekten Anschreiben mit Briefkopf, Adresse, Ort und Datum, Betreff und dem Text selbst verlangt. Da dies so selten passiert, finde ich meist meine Wordvorlage für diesen Zweck nicht: Seit dem letzten Anschreiben kann es nämlich gut sein, dass ich einen neuen oder neu aufgesetzten Rechner habe und ich vergessen habe, meine Vorlagen vom alten Rechner auf den neuen zu übertragen. Wie schön wäre es doch, wenn man eine kleine .NET-Anwendung hätte, die einem ein paar Textfelder für die wichtigsten Daten bereitstellt und nach der Eingabe per Mausklick das gewünschte Dokument im .docx-Format anlegt! Wie das gelingt, zeigt Ihnen der vorliegende Artikel.**

Wenn Sie ein Dokument wie oben beschrieben erstellen möchten, erledigen Sie das üblicherweise auf Basis einer Dokumentvorlage, die bereits einige wichtige Informationen enthält – zum Beispiel, wo die einzelnen Elemente wie Briefkopf, Anschrift, Ort und Datum, Betreff und Inhalt angelegt werden. Genau diese Dokumentvorlage benötigen wir auch für die Beispiellösung zu diesem Beitrag. Dumm nur, dass sich genau das Mitnehmen solcher Dokumente beim Aufsetzen eines neuen Rechners bei mir als Problem herausstellte und ich dann, statt die Vorlage auf dem alten Rechner zu suchen, lieber schnell von Hand ein neues Anschreiben zusammengesetzt habe. Also sollte die Lösung auch die Möglichkeit bieten, zumindest eine Dokumentvorlage zu speichern und diese dann bei Bedarf bereitzustellen. Wie dies gelingt, zeigen wir ausführlich im Artikel **Mit Resourcen arbeiten**.

Die grobe Vorgehensweise sieht so aus: Wir haben eine Anwendung, welche eine Word-Dokumentvorlage als Ressource enthält. Die Benutzeroberfläche stellt Textfelder bereits, mit denen Sie die im Anschreiben auszugebenden Informationen eintippen können. Nach der Eingabe können Sie per Mausklick einen Vorgang starten, bei dem die Anwendung die Dokumentvorlage im Dateisystem speichert (sofern diese noch nicht vorhanden ist) und ein neues Dokument auf Basis dieser Vorlage erstellt und mit den in der Anwendung eingegebenen Daten füllt. Das Dokument soll anschließend geöffnet werden, damit der Benutzer es beispielsweise ausdrucken kann. Alternativ wollen wir noch eine Schaltfläche anlegen, mit welcher der Benutzer direkt ein PDF-Dokument auf Basis des erstellten Word-Dokuments erzeugen kann.

## Dokumentvorlage vorbereiten

Der erste Schritt ist das Vorbereiten einer Dokumentvorlage. Das heißt, dass wir in Microsoft Word ein neues Dokument anlegen, die gewünschten Bereiche inklusive Platzhalter definieren und beispielsweise den Briefkopf hinzufügen und das Dokument dann unter der Dateiendung **.dotx** speichern. Letzteres ist besonders wichtig, damit wir neue Dokumente auf Basis dieser Vorlage anlegen können. In welcher Form wollen wir die Platzhalter anlegen? Am einfachsten ist es, einfach entsprechende Texte in eckigen Klammern zu den jeweiligen Elementen hinzuzufügen, also beispielsweise **[Betreffzeile]**.

## Vorlage nach DIN 5008

Wenn wir schon eine neue Vorlage anlegen, wollen wir diese auch halbwegs DIN-gerecht erstellen. Deshalb schauen wir uns dabei die aktuellen Vorgaben entsprechend DIN 5008 an, die etwa unter **www.din-5008-richtlinien.de** schön zusammenge-

fasst sind. Wir beginnen mit dem Anschriftenfeld, für das die DIN-Norm angibt, dass es 40 x 85 mm groß ist und einen Abstand von 45 mm zum oberen und 20 mm zum linken Rand aufweist (es gibt noch eine Variante mit 27 mm zum oberen Rand, aber wir wollen die Version für Briefumschläge mit Adressfenster nutzen).

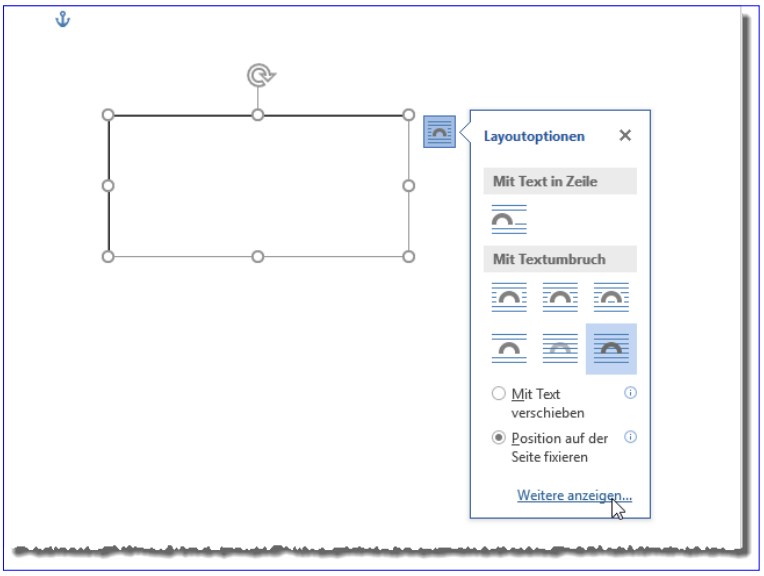



**Bild 1:** Anzeigen der Textfeld-Eigenschaften

## Anschriftenfeld

Wir öffnen also Word und legen ein neues, leeres Dokument an. Für das Anschriftenfeld fügen Sie dann ein Textfeld hinzu. Dies erledigen wir mit dem Ribbon-Eintrag **Einfügen|Text|Textfeld|Textfeld** erstellen. Ziehen Sie nach Betätigung dieses Befehls einen Rahmen auf, der etwa den gewünschten Abmessungen entspricht. Klicken Sie dann auf die Schaltfläche, die rechts oberhalb vom neuen Textfeld erscheint, und wählen Sie unten den Befehl **Weitere anzeigen** aus (siehe Bild 1).

Bevor wir auf den Dialog **Layout** schauen, noch die Information, dass Word mitunter etwas verwirrend ist (Version 2016). Word zeigt nämlich standardmäßig den verfügbaren Bereich ohne Seitenränder an – zumindest ohne die oberen und unteren Seitenränder. Wenn man das weiß, kann man die Einstellungen im Layout-Dialog wie in Bild 2 ohne Rücksicht darauf vornehmen, dass die Position des Textfeldes in der Ansicht Drucklayout nicht so aussieht, wie sie später auf dem Papier erscheinen wird – also mit 2 cm Abstand zur Seite horizontal und mit 4,5 cm Abstand zur Seite vertikal.

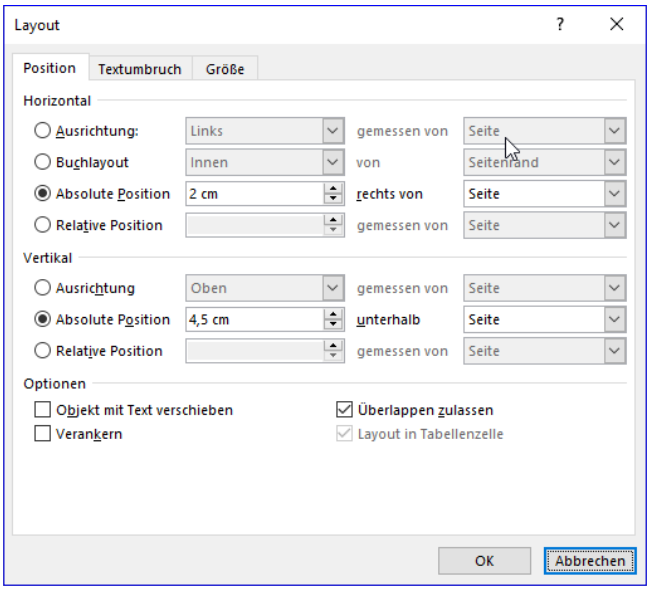



**Bild 2:** Position des Textfelds

Die Größe des Textfeldes stellen wir analog im Bereich Größe des gleichen Dialogs ein – und zwar wie von DIN 5008 vorgegeben auf 4 cm x 8,5 cm (siehe Bild 3). Die Größe können Sie allerdings auch über die beiden Steuerelemente in der Gruppe **Format|Größe** des Ribbons einstellen.

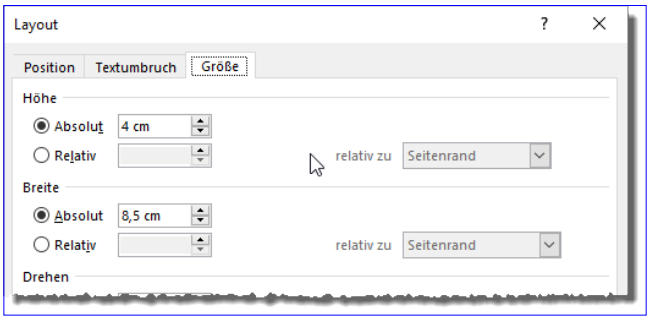



**Bild 3:** Größe des Textfelds

Schließlich weist das Textfeld noch einen Rahmen auf, den wir transparent machen wollen. Dazu wählen Sie beim markiertem Textfeld den Ribbon-Eintrag **Format|Formenarten|Formkontur|Keine Kontur** aus.


**Bild 4:** Einfügen einer Textmarke über das Ribbon

Ganz oben im Anschriftenfeld fügen wir nun eine Zeile mit der Rücksendeadresse ein, also der Adresse, an die der Brief zurückgesendet werden soll, wenn dieser nicht zugestellt werden konnte. Für diese Zeile legen Sie eine recht kleine Schriftgröße fest, zum Beispiel **8**. Darunter soll die eigentliche Anschrift landen – wie, erfahren Sie in den nächsten Abschnitten.

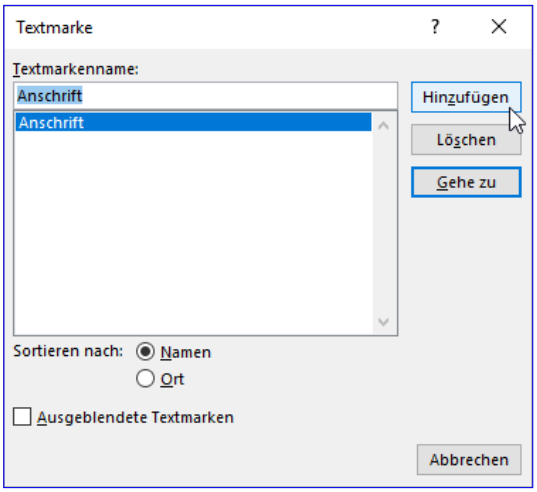



**Bild 5:** Einfügen einer Textmarke

## Textmarke einfügen

Später wollen wir den Textfeldern Texte hinzufügen. Dazu hinterlegen wir in den Textfeldern jeweils eine Textmarke, die wir dann später per Code leicht auffinden und durch die gewünschten Texte ersetzen können. Um die Textmarke hinzuzufügen, platzieren Sie die Einfügemarke im Textfeld für die Anschrift gleich unter der Rücksendeadresse und stellen die Schriftart auf die gewünschte Größe ein, beispielsweise auf **10**. Danach fügen wir die Textmarke ein. Dies erledigen wir mit dem Ribbon-Befehl **Einfügen|Links|Textmarke** (siehe Bild 4).

Dies öffnet den Dialog aus Bild 5, mit dem wir den Namen der neuen Textmarke eingeben und diese dann mit der Schaltfläche **Hinzufügen** an der Position der Einfügemarke einfügen. Im Dokument ist die Textmarke nun zunächst nicht zu erkennen. Das Vorhandensein dokumentiert lediglich der neue Eintrag im Dialog **Textmarke**.

Zur Sicherheit wollen wir die Textmarke jedoch noch sichtbar machen. Dazu öffnen Sie über das Ribbon den Dialog **Word-Optionen** und klicken dort links auf den Eintrag **Erweitert**. Im rechten Bereich finden Sie dann unter Dokumentinhalt anzeigen die Option **Textmarken anzeigen**, die Sie nun aktivieren (siehe Bild 6).

Wenn Sie den Dialog nun mit einem Klick auf die Schaltfläche

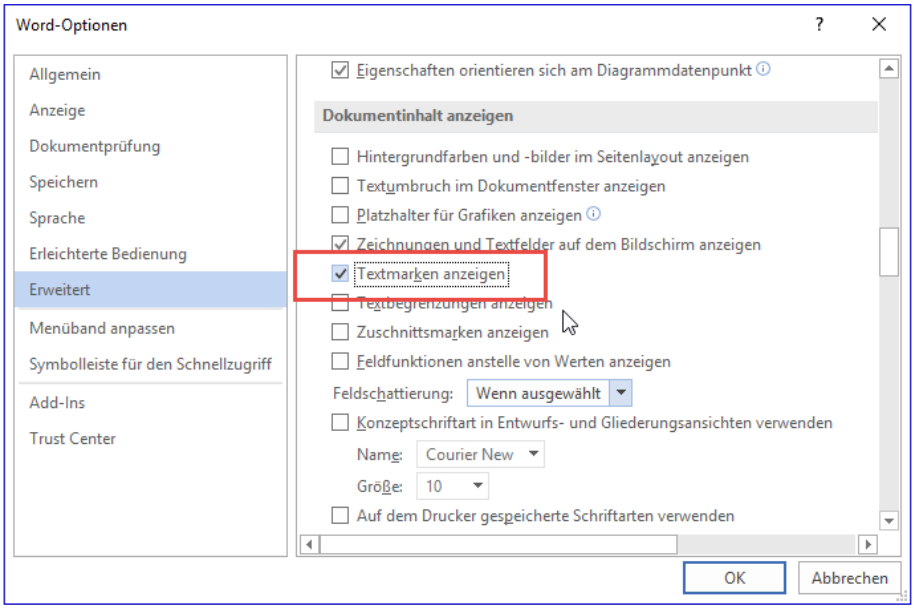



**Bild 6:** Textmarken einblenden

**OK** schließen, finden Sie im Textfeld für die Anschrift die Textmarke wie in Bild 7 vor.

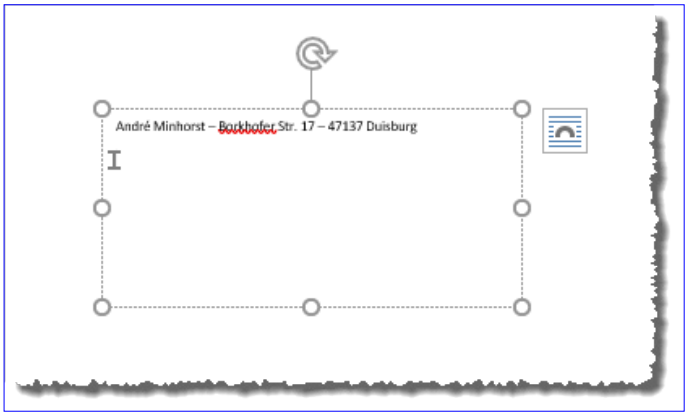



**Bild 7:** Markierung für eine Textmarke

Wenn Sie später alle Textfelder sichtbar machen wollen, können Sie einfach in einen leeren Bereich des Dokuments klicken und alle Elemente mit der Tastenkombination **Strg + A** markieren.

Das Textfeld können Sie nun gleich kopieren und als Basis für das Feld für die Betreffzeile nutzen – so brauchen Sie den Rahmen nicht wieder wie bei einem neuen Textfeld auf transparent einzustellen. Die Betreffzeile soll zwei Zeilen unterhalb des Anschriftenfeldes liegen, was wir als ca. 10 mm interpretieren. Also stellen wir den Abstand vom oberen Rand auf 10,5 cm ein, den Abstand vom linken Rand auf 2 cm und die Größe auf 1 x 12 cm. Löschen Sie den mitkopierten Text, stellen Sie die Schriftgröße auf 10 ein und fügen Sie auch diesem Textfeld einen Platzhalter hinzu – diesmal mit der Bezeichnung **Betreff**.

Rechts oben wollen wir den Briefkopf unterbringen. Dafür legen wir ein Textfeld mit einem Abstand von 2 cm vom rechten Rand und 2,5 cm vom oberen Rand und mit der Größe 5 x 5 cm an. Dazu müssen wir ein wenig rechnen, denn wir können nur Abstände von oben und links eintragen. Ein DIN A4-Blatt hat eine Breite von 19,7 cm, also müssen wir 2 cm für den Abstand von rechts und 5 cm für die Breite abziehen und landen bei einem Abstand von 12,7 cm vom linken Rand. Den Briefkopf füllen Sie mit den Absenderdaten aus und mit weiteren Daten, die Sie angeben möchten – Telefon, E-Mail, Webseite, Bankverbindung, Steuernummern et cetera.

Das Datum des Anschreibens fügen wir in einem Textfeld ein, das die Position 12,7 cm vom linken Rand und 9 cm vom oberen Rand einnimmt und eine Größe von 5 x 1 cm aufweist. Das Textfeld füllen wir mit einer weiteren Textmarke, diesmal mit der Bezeichnung **Datum** in Schriftgröße 10.

Das letzte Textfeld soll den Inhalt des Anschreibens aufnehmen. Es hat einen Abstand von 2 cm zum linken Rand und 13 cm zum oberen Rand und eine Größe von 15,7 x 12 cm. Hier fügen wir die Textmarke mit der Bezeichnung **Inhalt** ein.

An dieser Stelle der Hinweis, dass wir hier nur in Textfeldern schreiben und daher unabhängig von den Einstellungen der Seitenränder des Dokuments sind. Damit erhalten wir allerdings die Einschränkung, dass wir auch nur den Platz des dafür vorgesehenen Textfeldes für den Inhalt des Anschreibens haben. Für mehrseitige Dokumente müssten wir allerdings auch noch etwas mehr Aufwand betreiben – das soll jedoch nicht Thema dieses Artikels sein. Das Dokument sieht schließlich wie in Bild 8 aus. Schließlich speichern Sie die Word-Datei unter dem Namen **Vorlage.dotx**, wozu Sie den entsprechenden Dateityp beim Speichern auswählen müssen.

## Vorlage wieder öffnen

Wenn Sie die Dokumentvorlage nach dem Schließen nochmals öffnen wollen, um diese beispielsweise nochmals anzupassen, können Sie dies nicht einfach per Doppelklick auf die Datei **Vorlage.dotx** erledigen. Dies öffnet nämlich nicht die Dokument-

vorlge, sondern erzeugt ein neues Dokument auf Basis dieser Vorlage.

Dieses sieht dann zwar genauso aus wie die Vorlage selbst, aber wenn Sie diese speichern, stellen Sie fest, dass es sich hier um ein einfaches Word-Dokument handelt.

Wenn Sie die Vorlagendatei öffnen wollen, starten Sie zunächst Word und öffnen die Vorlagendatei dann über den **Datei öffnen**-Dialog von Word.

## Steuerelemente zum Eingeben der Briefdaten

Damit kommen wir zurück zum Visual Studio-Projekt. Diesem fügen wir nun zunächst die fertige Word-Vorlage als Ressource hinzu. Dazu öffnen Sie die Projekteigenschaften, wählen den Eintrag **Ressourcen** und legen die Word-Dokumentvorlage als neue Ressource an. Details dazu finden Sie im Artikel **Mit Ressourcen arbeiten**.

Wir machen es auf die Schnelle so, dass wir einfach die Datei **Vorlage.dotx** aus dem Windows Explorer in die Liste der Ressourcen hineinziehen (siehe Bild 9).

Sie können natürlich auch noch weitere Vorlagen anlegen, die beispielsweise für andere Familienmitglieder genutzt werden oder für geschäftliche Zwecke.

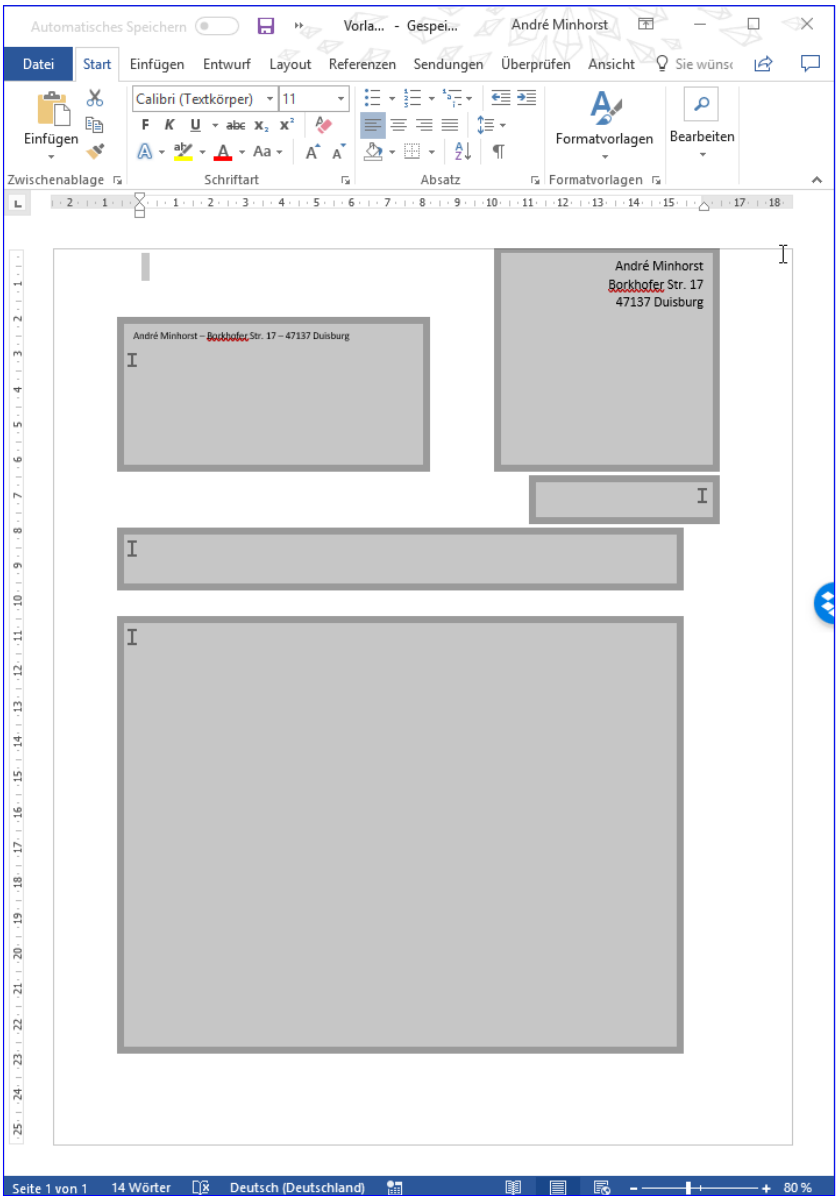



**Bild 8:** Anordnung der Textfelder

## Anzeige und Auswahl der Vorlagen

Ganz oben im Fenster wollen wir ein Kombinationsfeld zur Auswahl der Vorlage einfügen. Das Label und das Kombinationsfeld definieren wir wie folgt:

```
<ComboBox x:Name="cboVorlagen" Grid.RowSpan="2" Grid.Column="1"/>
<Button x:Name="btnGewaehlteRessourcenExportieren" Click="btnGewaehlteRessourcenExportieren_Click"
    Grid.Row="8">Gewählte Ressource exportieren</Button>
```

Damit das Kombinationsfeld mit den verfügbaren Vorlagen gefüllt wird, legen wir für das Window-Element die Ereigniseigenschaft **Loaded** an:

```
<Window x:Class="MainWindow" ...
Title="MainWindow" Height="450"
Width="525" Loaded="Window_Loaded">
```

Für diese hinterlegen wir die folgende Methode:

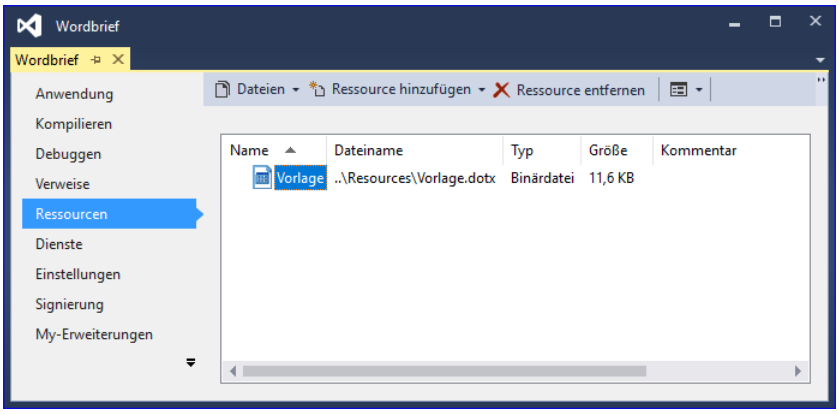



**Bild 9:** Voraussetzung: Eine Word-Dokumentvorlage als Ressource

```
private void Window_Loaded(object sender, RoutedEventArgs e) {
    ResourceManager resourceManager;
    ResourceSet resourceSet;
    string strResource;
    resourceManager = Properties.Resources.ResourceManager;
    resourceSet = resourceManager.GetResourceSet(CultureInfo.CurrentCulture, true, true);
    cboVorlagen.Items.Add("<Auswählen>");
    foreach (DictionaryEntry dictionaryEntry in resourceSet) {
        strResource = dictionaryEntry.Key.ToString();
        cboVorlagen.Items.Add(strResource);
    }
    cboVorlagen.SelectedIndex = 0;
}
```

Die Methode erstellt ein **ResourceManager**-Objekt auf Basis der in der Anwendung befindlichen Ressourcen und referenziert die Elemente mit der Variablen **objResourceSet**. Dann fügt sie dem Kombinationsfeld **cboVorlagen** den Eintrag **<Auswählen>** hinzu und anschließend in einer Schleife über die Elemente von **objResourceSet** die Namen der verfügbaren Ressourcen. Schließlich stellt sie das Kombinationsfeld auf den Eintrag **<Auswählen>** ein.

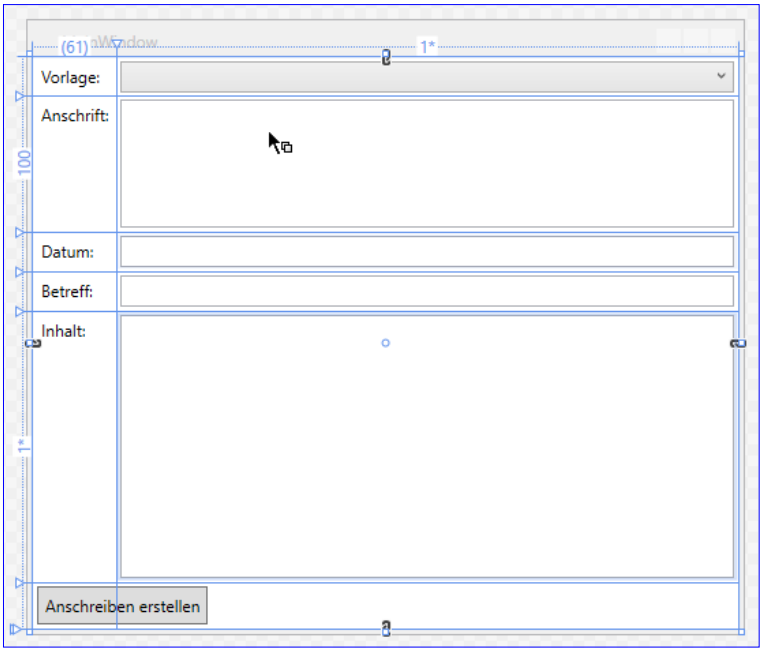



**Bild 10:** Steuerelemente zum Eingeben der Eigenschaften des Dokuments

Direkt danach legen wir weitere Steuerelemente an, und zwar die **TextBox**-Elemente **txtAnschrift**, **txtDatum**, **txtBetreff** und **txtInhalt** mit den jeweiligen **Label**-Elementen. Für die beiden **TextBox**-