AUSGABE 01/2020

DATENBANK ENTWICKLER

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

TOP-THEMEN:

André Minhorst Verlag

FlowDocument-Elemente mit XAML

Ein Element des Typs FlowDocument dient zusammen mit den drei Steuerelementen FlowDocumentScrollViewer, FlowDocumentTextViewer und FlowDocumentReader der Textdarstellung in WPF. Während auch andere Steuerelemente wie TextBox oder TextBlock Texte darstellen können und auch ihre Bearbeitung ermöglichen, bietet das FlowDocument viel mehr Möglichkeiten, was die Gestaltung des Texts angeht. Neben verschiedenen Absatzformatierungen und Zeichenformatierungen sind hier auch die Anzeige von Listen, Tabellen oder eingebauten Elementen wie Bildern möglich. Dieser Artikel stellt das FlowDocument-Element mit seinen Eigenschaften vor und zeigt auch, wie Sie solche Dokumente mit VB zur Laufzeit erstellen können. Letzteres ist interessant, wenn Sie Inhalte aus verschiedenen Feldern einer Datenbank strukturiert und formatiert darstellen wollen.

FlowDocuments benötigen entweder eines der drei Darstellungs-Steuerelemente **FlowDocumentScrollViewer**, **FlowDocumentTextViewer** oder **FlowDocumentReader** oder, wenn Sie den Inhalt bearbeiten wollen, das **RichTextBox**-Steuerelement. Im vorliegenden Artikel verwenden wir einfach das **FlowDocumentReader**-Steuerelement, damit wir die per WPF zusammengestellten formatierten Texte darstellen können. Auf das **RichTextBox**-Steuerelement gehen wir im Artikel **Das RichTextBox-Steuerelement** ein.

FlowDocument anlegen

Ein **FlowDocument**-Element können wir beispielsweise wie in Bild 1 in einem **FlowDocumentReader**-Steuerelement ausgeben. Und die WPF-Definition beginnt auch genau so: Wir legen in einem neuen, leeren Fenster ein **FlowDocumentReader**-Element an, dem wir ein **FlowDocument**-Element hinzufügen:

```
<FlowDocumentReader>
    <FlowDocument>
    ...
    </FlowDocument>
```

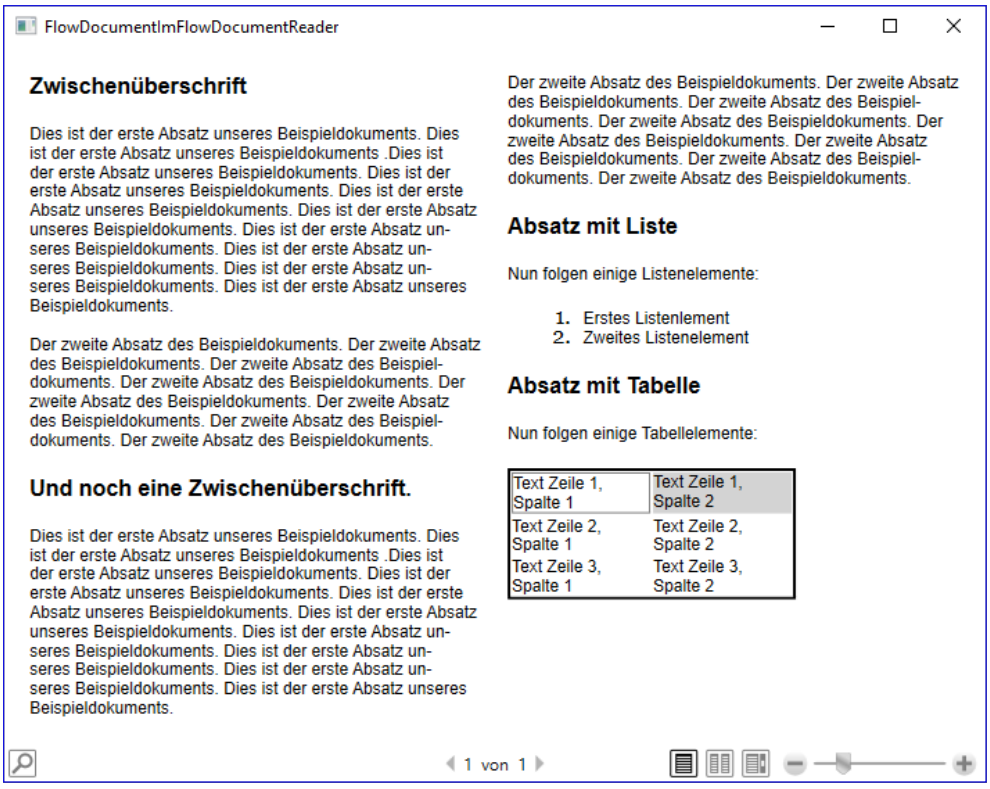

Bild 1: Ausgabe des Inhalts eines FlowDocuments im **FlowDocumentReader**-Steuerelement

```
</FlowDocumentReader>
```

Damit erhalten wir dann ein noch leeres **FlowDocumentReader**-Element im Entwurf der **.xaml**-Datei (siehe Bild 2).

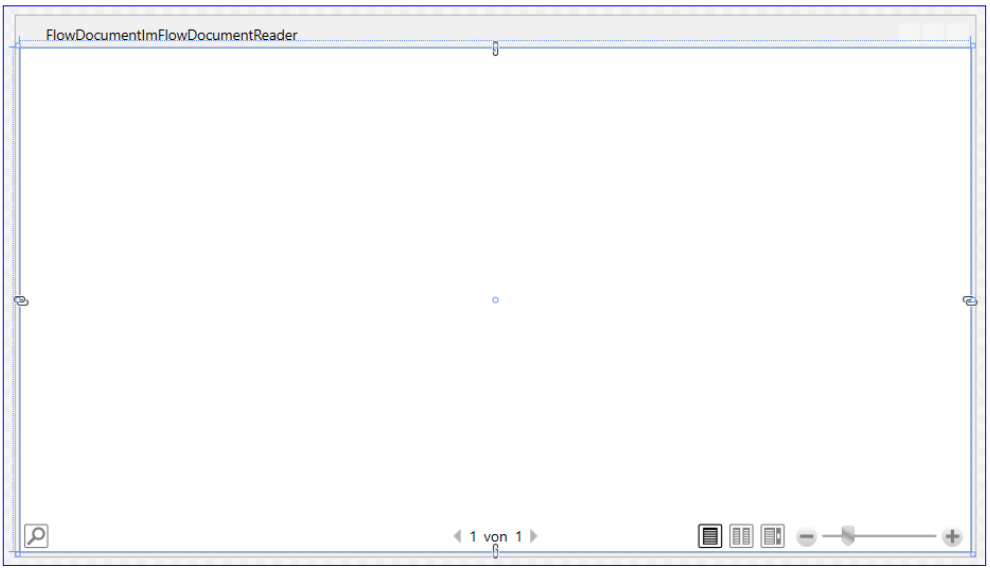

Bild 2: Das leere **FlowDocument** im **FlowDocumentReader**

Text zum FlowDocument hinzufügen

Das FlowDocument füllen wir sehr schnell, indem wir einfach ein **Paragraph**-Objekt mit einen beliebigen Text zum **FlowDocument**-Element hinzufügen:

```
<Paragraph>Ein neuer Text.</Paragraph>
```

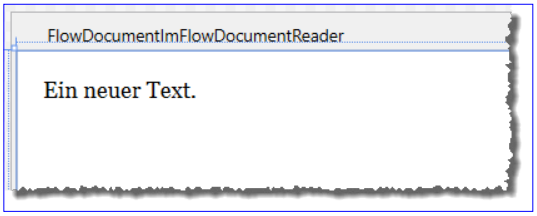

Bild 3: Ein Beispielabsatz

Praktischerweise zeigt der Entwurf die so hinzugefügten Inhalte direkt an (siehe Bild 3). So müssen wir nicht immer das Projekt starten, um den Fortschritt zu betrachten.

Absatzformatierungen

Das **Paragraph**-Element erlaubt einige Formatierungen, die für den kompletten Absatz gelten. Diese geben Sie als Attribute für das **Paragraph**-Element an. Die wichtigsten dienen der Formatierung des Texts:

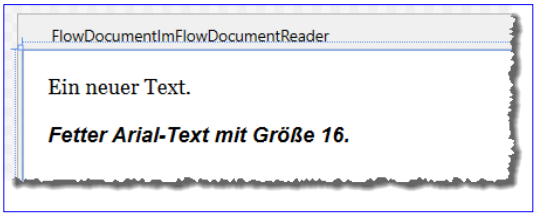

Bild 4: Formatierter Absatz

- **FontFamily**: Gibt den Namen der Schriftart für den Text des Absatzes an, zum Beispiel **Arial**.
- **FontSize**: Gibt an, welche Größe die Schrift haben soll (Angabe als numerischer Wert wie etwa in Word)
- **FontWeight**: Gibt an, ob der Text normal (Standardwert) oder fett gedruckt werden soll (Wert **Bold**)
- **FontStyle**: Für kursive Darstellung mit den Werten **Italic** oder **Oblique**.

Mit diesen Formatierungen erhalten wir bereits ein anderes Ergebnis als zuvor (siehe Bild 4):

```
<Paragraph FontWeight="Bold" FontFamily="Arial" FontSize="16" FontStyle="Italic">Fetter Arial-Text mit Größe 16.</Para-
graph>
```

Damit können wir verschiedene Absatzformatierungen durchführen und erhalten eine optisch brauchbare Formatierung – wie zum Beispiel mit den wie folgt definierten Absätzen:

```
<Paragraph FontSize="16" FontFamily="Arial" FontWeight="Bold">Zwischenüberschrift</Paragraph>
<Paragraph FontSize="12" FontFamily="Arial">Dies ist der erste Absatz unseres Beispieldokuments. ...</Paragraph>
<Paragraph FontSize="12" FontFamily="Arial">Der zweite Absatz des Beispieldokuments. ...</Paragraph>
<Paragraph FontSize="16" FontFamily="Arial" FontWeight="Bold">Und noch eine Zwischenüberschrift.</Paragraph>
<Paragraph FontSize="12" FontFamily="Arial">Dies ist der erste Absatz unseres Beispieldokuments. ...</Paragraph>
<Paragraph FontSize="12" FontFamily="Arial">Der zweite Absatz des Beispieldokuments. ...</Paragraph>
```

Diese Absätze sehen nach dem Starten des Projekts wie in Bild 5 aus. Wir haben das Fenster mit dem **FlowDocumentReader** ein wenig verkleinert, um zu zeigen, dass das Dokument dann auf zwei Seiten aufgeteilt wird, von der die erste angezeigt wird. Mit den Steuerelementen unten können Sie dann zur zweiten Seite blättern. Außerdem haben wir die Breite soweit verringert, dass das Fenster den Text in nur einer Spalte anzeigt. Wenn Sie das Fenster wieder breiter ziehen, erscheinen zunächst zwei Spalten, dann auch noch mehr als zwei Spalten.

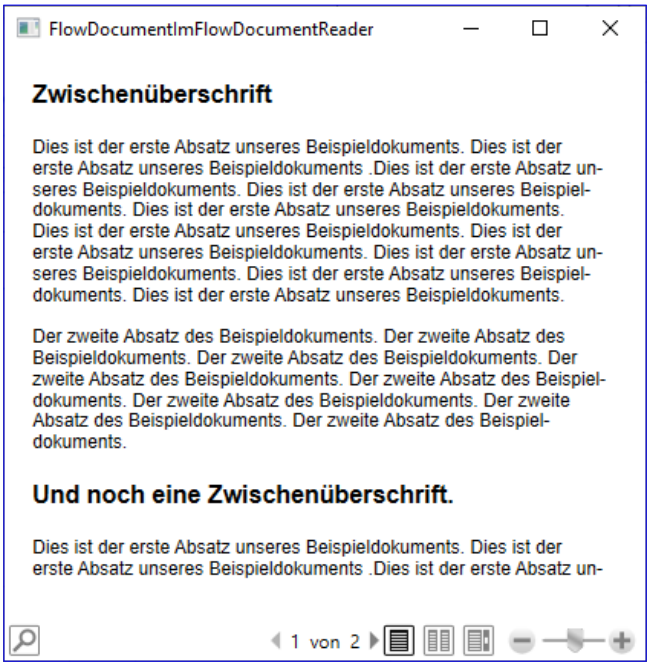

Bild 5: Ausgabe einiger Absätze im **FlowDocumentReader**-Element

Die Formatierung der Schrift müssen Sie nicht für jeden Absatz einzeln durchführen. Sie können die Attribute auch für die übergeordneten Elemente definieren. Die untergeordneten Elemente übernehmen beziehungsweise »erben« die Werte der übergeordneten Elemente.

Inline-Elemente

Sie können auch einzelne Elemente innerhalb eines Absatzes formatieren. Dazu nutzen Sie wie unter HTML sogenannte Inline-Formatierungen. Zum Formatieren des Textes mit fetten, kursiven oder unterstrichenen Zeichen verwenden Sie die folgenden Elemente:

- **Bold**: Formatiert den eingeschlossenen Text fett.
- **Italic**: Formatiert den eingeschlossenen Text kursiv.
- **Underline**: Unterstreicht den eingeschlossenen Text.

Ein Beispiel mit allen drei Formatierungen:

```
<Paragraph FontSize="12" FontFamily="Arial">Text mit <Bold>fettem</Bold>, <Italic>kursivem</Italic> und
<Underline>unterstrichenem</Underline> Inhalt.</Paragraph>
```

In den **.xaml**-Code eingebaute Zeilenumbrüche wirken sich übrigens nicht auf die Darstellung aus. Die folgende, vom **.xaml**-Editor automatischen hinzugefügten Zeilenumbrüche werden also nicht auf den Text übertragen:

```
<Paragraph FontSize="12" FontFamily="Arial">Text mit
    <Bold>fettem</Bold> ,
    <Italic>kursivem</Italic> und
    <Underline>unterstrichenem</Underline> Inhalt.
</Paragraph>
```

Zeilenumbrüche

Wenn Sie Zeilenumbrüche wünschen, müssen Sie diese explizit angeben. Unter HTML fügen Sie Zeilenumbrüche mit dem Zeichen **
** ein, unter WPF in **FlowDocument**-Elementen mit dem Element **<LineBreak />** (das abschließende Slash-Zeichen ist zwingend erforderlich):

```
<Paragraph FontSize="12" FontFamily="Arial">Text mit einem Zeilenumbruch
    <LineBreak /> vor diesem Teil.</Paragraph>
```

Abbildungen mit dem Figure-Element

Mit dem **Figure**-Element können Sie Abbildungen und andere Elemente in den Fließtext integrieren. Dazu fügen Sie in das **Paragraph**-Element ein **Figure**-Element ein. Dieses bietet Eigenschaften wie die folgenden:

- **HorizontalAnchor**: Horizontaler Anker, **ColumnRight** verankert beispielsweise am rechten Rand der aktuellen Spalte
- **VerticalAnchor**: Vertikaler Anker, **ParagraphTop** beispielsweise verankert am oberen Rand des Absatzes

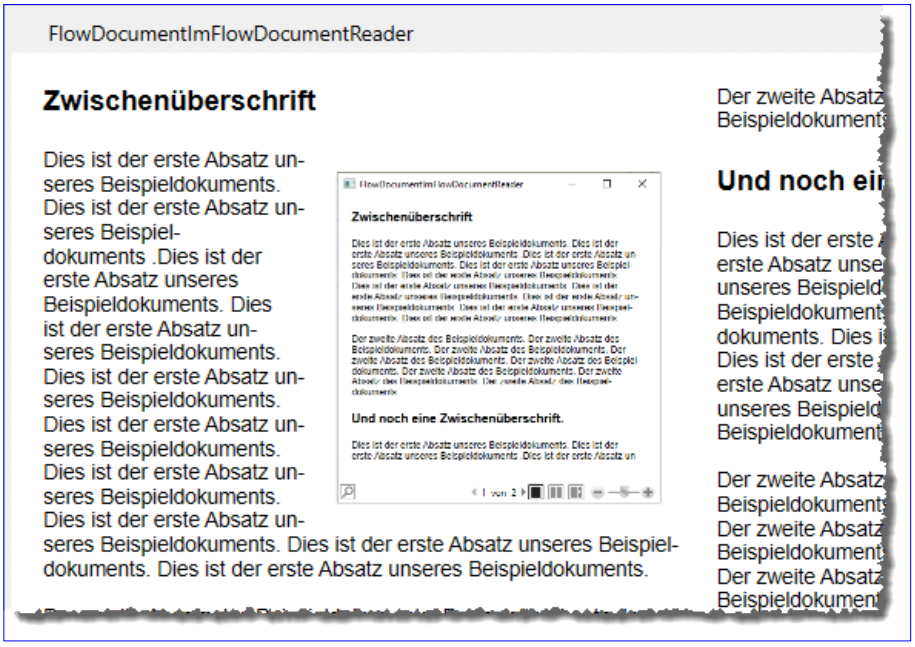

Bild 6: Einbetten einer Abbildung per **Figure**-Element

- **Width**/**Height**: Geben Breite und Höhe des Elements an.
- **VerticalOffset**: Gibt den vertikalen Versatz an.
- **HorizontalOffset**: Gibt den horizontalen Versatz an.

Mit diesen Eigenschaften lässt sich eine Abbildung wie in Bild 6 einbetten.

Dazu benötigen wir noch einige weitere Elemente, die wir in das **Figure**-Element einbauen – zunächst ein **BlockUIContainer**-Element ohne Attribute und darin den eigentlichen Inhalt, in diesem Fall ein **Image**-Element mit dem anzuzeigenden Bild:

```
<Paragraph FontSize="12" FontFamily="Arial">
    <Figure Width="200" HorizontalAnchor="ColumnRight" VerticalAnchor="ParagraphTop">
        <BlockUIContainer>
            <Image Source="images/pic005.png"/>
        </BlockUIContainer>
    </Figure>
Dies ist der erste Absatz unseres Beispieldokuments. ...</Paragraph>
```

Sie können dem Bild sogar noch eine Bildunterschrift hinzufügen. Dazu fügen Sie einfach noch ein **Paragraph**-Element mit dem gewünschten Text zwischen dem schließenden **BlockUIContainer**- und dem schließenden **Figure**-Element ein:

```
</BlockUIContainer>
    <Paragraph Padding="5">
        <Bold>Bild1: Beispielbild</Bold>
     </Paragraph>
</Figure>
```

Mit dem **Figure**-Element können Sie auch einfache Textkästen in den Fließtext integrieren – siehe Bild 7. Im folgenden Beispielcode haben wir dazu ein neues **Figure**-Element in das **Paragraph**-Element eingefügt, das den Kasten enthalten soll. Für das **Figure**-Element haben wir die Attribute so eingestellt, dass der Kasten rechts oben in der Textspalte mit dem entsprechenden Absatz verankert wird. Außerdem haben wir die Breite auf **200** festgelegt, einen

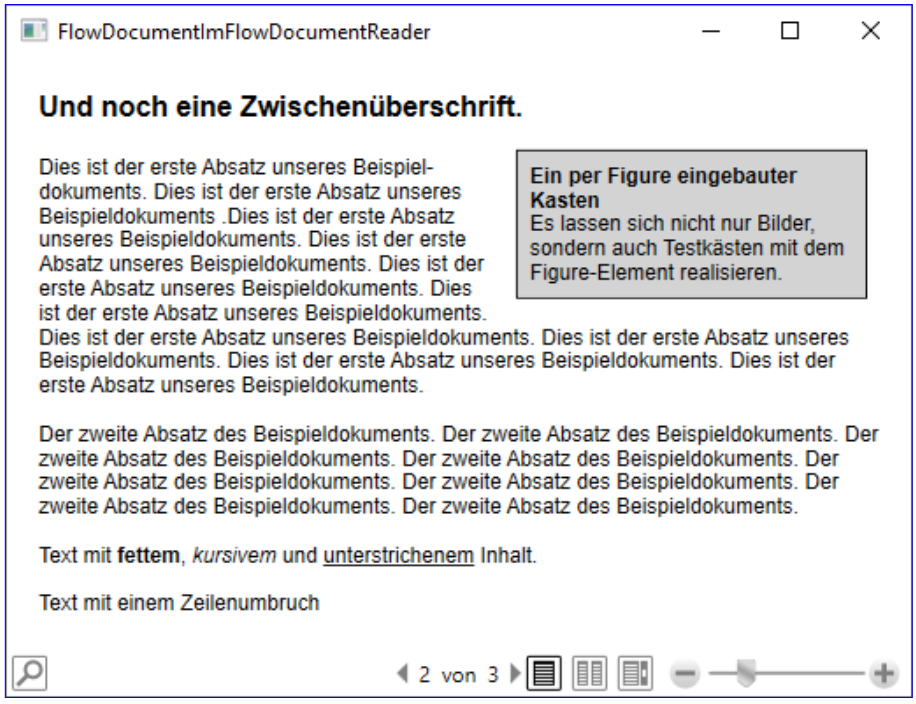

Bild 7: Einbetten eines Kastens mit Text in den Fließtext

Rahmen definiert und einen grauen Hintergrund. Im Kasten haben wir dann eine fett gesetzte Überschrift sowie einen normal gesetzten Text eingefügt:

```
<Paragraph FontSize="12" FontFamily="Arial">
    <Figure Width="200" VerticalOffset="-10" BorderBrush="Black" BorderThickness="1" Background="LightGray">
        <Paragraph FontWeight="Bold">Ein per Figure eingebauter Kasten</Paragraph>
        <Paragraph>Es lassen sich nicht nur Bilder, sondern auch Testkästen mit dem Figure-Element realisieren.</Paragraph>
    </Figure>Dies ist der erste Absatz unseres Beispieldokuments. ...
</Paragraph>
```

Weitere Formatierung mit dem Span-Element

Mit den drei Inline-Elementen **Bold**, **Italic** und **Underline** können Sie zwar bereits einige Hervorhebungen im Text unterbringen, aber manchmal bedarf es etwas mehr Anpassungen. Dazu können Sie das **Span**-Element nutzen. Dieses verwenden Sie prinzipiell wie unter HTML: Sie fassen den betroffenen Text in ein öffnendes und ein schließendes **Span**-Element ein und versehen das öffnende Element dann mit den gewünschten Attributen. Das sieht beispielsweise so aus:

```
<Paragraph FontSize="12" FontFamily="Arial">Der zweite Absatz des <Span FontWeight="Black" Foreground="Red">Beispieldokum
ents</Span>. Der zweite Absatz des Beispieldokuments. ...</Paragraph>
```

Sie können auch Text wie Hyperlinks formatieren und diese mit einer Link-Funktion versehen. Dazu nutzen Sie das **Hyperlink**-Element:

```
<Paragraph FontFamily="Arial" FontSize="12">Absatz mit URL:
    <Hyperlink>http://datenbankentwickler.net</Hyperlink></Paragraph>
```

Weitere Blöcke

Neben dem Absatz beziehungsweise dem **Paragraph**-Element als vermutlich am häufigsten verwendeten Element gibt es noch vier weitere Elemente:

- **Section**: Kann weitere Elemente wie **BlockUIContainer**, **List**, **Paragraph**, **Table** und auch weitere **Section**-Elemente aufnehmen und beispielsweise Attribute für alle untergeordneten Elemente festlegen.
- **List**: Erlaubt das Formatieren von Absätzen in Listenform.
- **Table**: Ermöglicht das Strukturieren von Texten und Elementen in Tabellen.
- **BlockUIContainer**: Mit dem **BlockUIContainer**-Element können Sie WPF-Steuerelemente in das FlowDocument einfügen.

Das RichTextBox-Steuerelement

Die beiden Steuerelemente TextBox und TextBlock reichen für die Darstellung einfacher Texte aus. Wenn die Texte jedoch formatiert oder sogar strukturiert dargestellt werden wollen, brauchen wir eine Alternative. Im Artikel »FlowDocument-Elemente mit XAML« haben Sie bereits das FlowDocument-Element kennen gelernt, das viele Möglichkeiten bietet einschließlich Absätze, Inline-Formatierungen, Listen, Tabellen und Abbildungen. Nun wollen wir uns ansehen, wie uns das RichTextBox-Steuerelement dabei unterstützen kann, diese Art von Dokumenten zu bearbeiten.

FlowDocument als Inhalt von RichTextBox-Steuerelementen

RichTextBox-Steuerelemente enthalten immer ein **FlowDocument**-Element als Inhalt. Wenn Sie also Text in ein solches Steuerlement eingeben, dann wird dieser Text im Hintergrund als Inhalt eines **FlowDocument**-Elements gespeichert. Ein **FlowDocument**, das haben wir bereits im Artikel **FlowDocument**-Elemente mit XAML beschrieben, ist im Prinzip ein XML-Dokument, das sowohl den Inhalt also auch die Formatierungen und die Strukturierung des Dokuments enthält.

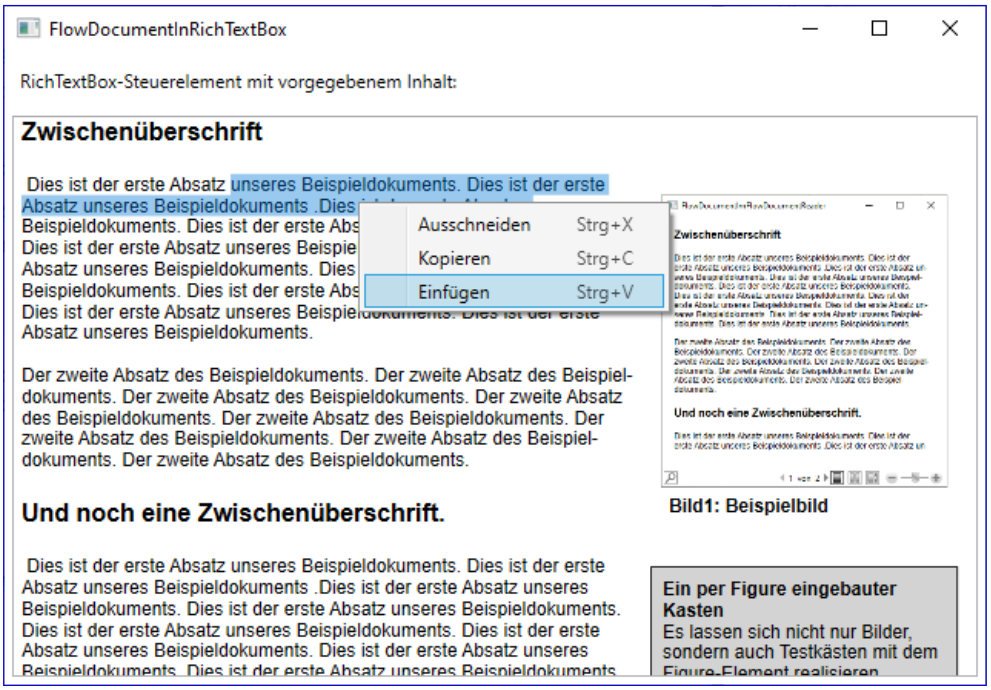

Bild 1: RichTextBox-Element mit formatiertem Text, der sich aber nicht formatieren lässt

Wir wollen ein Fenster erzeugen, dass in einem Grid ein Label und ein **RichTextBox**-Steuerelement enthält und den Inhalt eines der **FlowDocument**-Elemente, die wir im oben genannten Artikel erzeugt haben, enthält (siehe Bild 1).

Dazu nutzen wir den folgenden Code:

```
<Grid>
    <Grid.RowDefinitions>
        <RowDefinition Height="Auto"></RowDefinition>
        <RowDefinition Height="*"></RowDefinition>
```

```
    </Grid.RowDefinitions>
    <Label Grid.Row="0" Margin="5">RichTextBox-Steuerelement mit vorgegebenem Inhalt:</Label>
    <RichTextBox Grid.Row="1" Margin="5">
        <FlowDocument IsOptimalParagraphEnabled="true" IsHyphenationEnabled="True" TextAlignment="Left">
            <Paragraph FontSize="16" FontFamily="Arial" FontWeight="Bold">Zwischenüberschrift</Paragraph>
            ...
        </FlowDocument>
    </RichTextBox>
</Grid>
```

Wir haben hier direkt den Beispielcode aus dem Artikel **FlowDocument-Elemente mit XAML** (**www.datenbankentwickler.net/203**) in den Code integriert, damit wir direkt Text zum Herumprobieren haben. Im Artikel **FlowDocument-Elemente mit VB** (**www.datenbankentwickler.net/210**) zeigen wir, wie Sie **FlowDocument**-Elemente per VB mit Inhalt füllen, diese auslesen oder bearbeiten.

Leider zeigt das **RichTextBox**-Element in der aktuellen Konfiguration keine Steuerelemente zum Formatieren des Textes an – zunächst keine offensichtlichen, aber auch keine über das Kontextmenü. Außerdem sehen wir in der Abbildung auch direkt, dass sich das **RichTextBox**-Element deutlich von den Steuerelementen zur Anzeige von **FlowDocument**-Elementen unterscheidet. Der Inhalt wird immer in einer Spalte angezeigt, es gibt keine Zoom- und auch keine Suchfunktion. Wir haben es also mit einem Steuerelement rein zum Bearbeiten des Inhalts zu tun. Damit Sie den enthaltenen Text auch in gewohnter Form bearbeiten können, fügen wir im Artikel **Format-Menü für die RichTextBox** (**www.datenbankentwickler.net/205**) ein passendes Menü hinzu.

RichTextBox-Steuerelement leeren

Wenn Sie mit einem leeren **RichTextBox**-Steuerelement fortfahren wollen, können Sie das schnell erledigen, indem Sie den kompletten Inhalt markieren (mit **Strg + A**) und dann die **Entf**-Taste betätigen. Sie finden dann ein leeres **RichTextBox**-Steuerelement vor (siehe Bild 2).

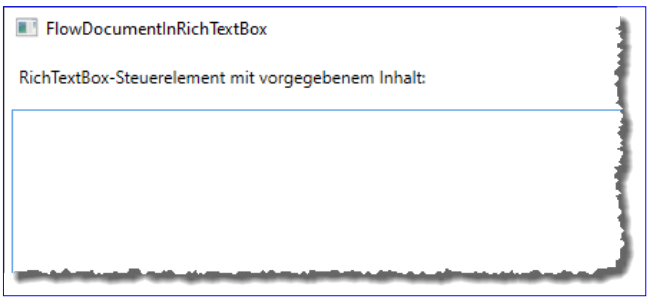

Bild 2: Geleertes **RichTextBox**-Steuerelement

Kompletten Inhalt markieren

Wir wollen zeigen, wie Sie per VB den kompletten Inhalt des **RichTextBox**-Steuerelements markieren. Dazu fügen wir eine erste Schaltfläche oberhalb des **RichTextBox**-Steuerelements hinzu, das wir wie folgt in ein **StackPanel**-Element mit horizontaler Ausrichtung zusammenfassen:

```
<StackPanel Orientation="Horizontal" Grid.Row="1">
    <Button x:Name="btnAllesMarkieren" Click="BtnAllesMarkieren_Click">Alles markieren</Button>
</StackPanel>
```

Die Ereignismethode enthält den Aufruf der **SelectAll**-Methode des **RichTextBox**-Steuerelements sowie die Anweisung zum Verschieben des Fokus auf dieses Steuerelement (siehe Bild 3).

Die **SelectAll**-Methode reicht zwar aus, um den Inhalt des **RichTextBox**-Steuerelements zu markieren, aber solange das Steuerelement nicht den Fokus hat, wird die Markierung nicht angezeigt. Daher müssen wir noch den Fokus auf das **RichTextBox**-Element verschieben.

```
Private Sub BtnAllesMarkieren_
Click(...)
    rtb.SelectAll()
    rtb.Focus()
End Sub
```

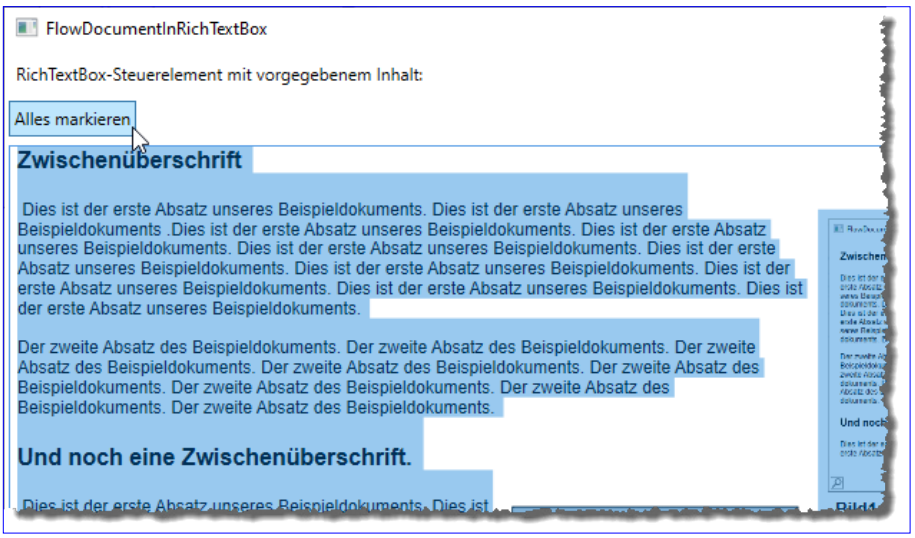

Bild 3: Markieren des vollständigen Inhalts des **RichTextBox**-Steuerelements

Text ausgeben

Um den vollständigen Text auszugeben, müssen wir einen etwas anderen Weg gehen als etwa bei einem Textfeld. Hier reicht es, über das **Text**-Attribut auf den Inhalt zuzugreifen.

Dazu deklarieren wir ein **TextRange**-Objekt namens **objTextRange** und füllen dieses mit einem neuen Objekt dieses Typs mit den Konstruktor-Parametern **rtb.Document.ContentStart** und **rtb.Document.ContentEnd**. Dann geben wir den Inhalt von **objTextRange** über die **Text**-Eigenschaft in einem Meldungsfenster aus:

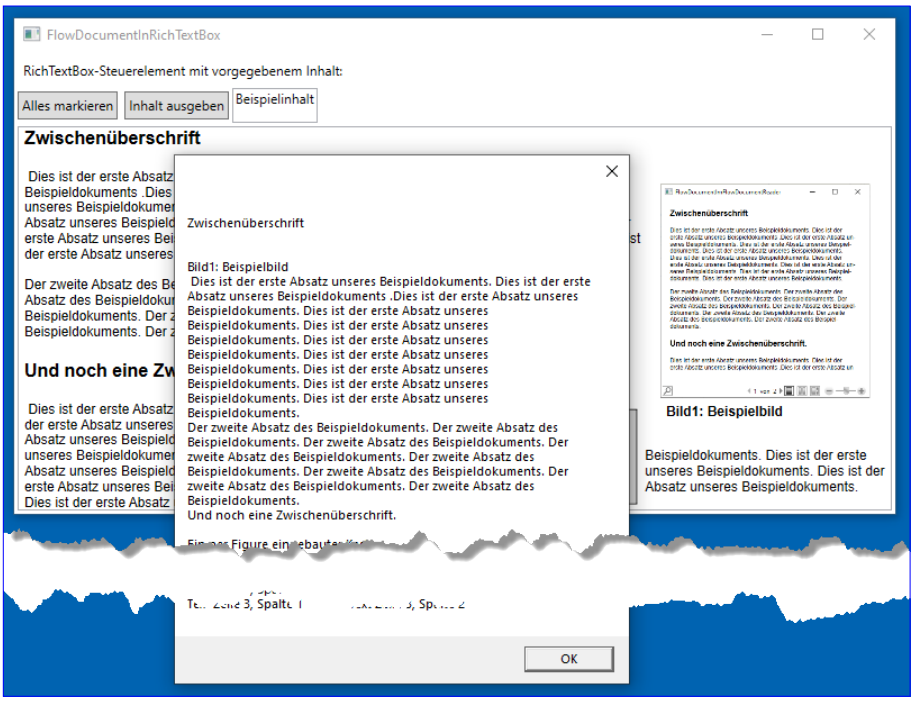

Bild 4: Ausgabe des Inhalts des **RichTextBox**-Steuerelements

```
Private Sub BtnInhaltAusgeben_Click(sender As Object, e As RoutedEventArgs)
    Dim objTextRange As TextRange
    objTextRange = New TextRange(rtb.Document.ContentStart, rtb.Document.ContentEnd)
    MessageBox.Show(objTextRange.Text)
End Sub
```

Das Ergebnis zeigt Bild 4. Hier ist zu erkennen, dass die Ausgabe ohne Formatierungen geliefert wird.

Markierten Text auslesen

Den aktuell markierten Text ermitteln wir mit der **Selection**-Eigenschaft des **RichTextBox**-Steuerelements. Dieses bietet mit der **Text**-Eigenschaft die Möglichkeit, den enthaltenen Text auszulesen. Die folgende Methode gibt den markierten Text aus:

```
Private Sub BtnMarkierterText_Click(sender As Object, e As RoutedEventArgs)
    Dim strSelection As String
    strSelection = rtb.Selection.Text
    MessageBox.Show(strSelection)
End Sub
```

Auf Änderung der Markierung reagieren

Manchmal will man auf das Setzen oder Ändern einer Markierung reagieren. Dazu gibt es ein passendes Ereignis, das wir durch Setzen des Attributs **SelectionChanged** auf den Namen der auszuführenden Ereignismethode aktivieren:

```
<RichTextBox x:Name="rtb" ... SelectionChanged="Rtb_SelectionChanged">
```

Das **Selection**-Ereignis feuert mehrmals, wenn Sie beispielsweise mit der Maus einen Text markieren und dazu an einer Stelle die Maustaste herunterdrücken und dann den Mauszeiger bis zum Ende der Markierung ziehen oder die Markierung mithilfe von Tastaturbefehlen erweitern. Daher geben wir den aktuell markierten Text im folgenden Beispiel in einem Textfeld aus:

```
Private Sub Rtb_SelectionChanged(sender As Object, e As RoutedEventArgs)
    Dim strSelection As String
    strSelection = rtb.Selection.Text
    txt.Text = strSelection
End Sub
```

Text an Markierung einfügen

Die folgende Schaltfläche soll eine Methode aufrufen, die den Text aus dem Textfeld an der Stelle der aktuellen Markierung im **RichTextBox**-Steuerelement einfügen soll:

```
<Button x:Name="btnTextEinfuegen" Click="BtnTextEinfuegen_Click">Text an Markierung einfügen:</Button>
<TextBox x:Name="txtEinfuegen" Text="Einfügen"></TextBox>
```

Die dadurch aufgerufene Ereignismethode erledigt das durch Einstellen des mit **Selection.Text** ermittelten Bereichs auf den Inhalt des Textfeldes:

```
Private Sub BtnTextEinfuegen_Click(sender As Object, e As RoutedEventArgs)
    Dim strText As String
    strText = txtEinfuegen.Text
    rtb.Selection.Text = strText
End Sub
```

Kompletten Inhalt durch neuen Text ersetzen

Wenn Sie den kompletten aktuellen Inhalt des **RichTextBox**-Steuerelements durch einen anderen Text ersetzen wollen, legen Sie den zu ersetzenden Teil mit einem **TextRange**-Element fest. Dieses füttern wir beim Initialisieren mit der Start- und der Endmarkierung des enthaltenen Dokuments und weisen dann der **Text**-Eigenschaft des **TextRange**-Objekts den einzusetzenden Text zu:

```
Private Sub BtnKomplettenTextErsetzen_Click(sender As Object, e As RoutedEventArgs)
    Dim objTextRange As TextRange
    Dim strText As String
    strText = txtEinfuegen.Text
    objTextRange = New TextRange(rtb.Document.ContentStart, rtb.Document.ContentEnd)
    objTextRange.Text = strText
End Sub
```

Texte speichern und laden

Wenn Sie eine kleine Textverarbeitung bauen wollen, möchten Sie die eingegebenen Texte auch speichern und wieder in das **RichTextBox**-Steuerelement laden können. Als Erstes wollen wir den aktuell im **RichTextBox**-Steuerelement enthaltenen Text in einer Datei speichern. Dazu benötigen wir einen Dialog zum Auswählen des Pfades der Datei, den wir mit dieser Funktion aufrufen:

```
Private Function GetSaveFilename() As String
    Dim objSaveFileDialog As SaveFileDialog
    Dim strFilename As String = ""
    objSaveFileDialog = New SaveFileDialog
    If (objSaveFileDialog.ShowDialog = True) Then
        strFilename = objSaveFileDialog.FileName
    End If
    Return strFilename
End Function
```

Diese Funktion nutzen wir in der Ereignismethode, die durch die Schaltfläche **btnSpeichern** ausgelöst wird. Hier ermitteln wir zunächst den Dateinamen mit der Funktion **SaveFilename**:

```
Private Sub BtnSpeichern_Click(sender As Object, e As RoutedEventArgs)
    Dim objTextRange As TextRange
    Dim objFileStream As FileStream
    Dim strDateiname As String
    strDateiname = GetSaveFilename()
```

Dann prüfen wir, ob ein Dateiname zum Speichern eingegeben wurde. Falls ja, erstellen wir ein neues **TextRange**-Objekt, das mit dem wir den kompletten Inhalt des **RichTextBox**-Steuerelements füllen. Dann erstellen wir ein **FileStream**-Objekt, das ein

Format-Menü für die RichTextBox

Das RichTextBox-Steuerelement unter WPF offeriert eine Reihe von Formatierungsmöglichkeiten. Damit Sie diese Formatierungen komfortabel anwenden können, bauen wir in diesem Artikel ein Menü mit den gängigsten Befehlen zur Formatierung von Texten im RichTextBox-Steuerelement. Dabei können wir gleich einige der Grundlagen, die wir im Artikel Das RichTextBox-Steuerelement kennengelernt haben, praktisch umsetzen.

RichTextBox-Inhalte formatieren

Da das RichTextBox-Steuerelement keine offensichtlichen Steuerelemente zum Formatieren bietet, legen wir diese einfach selbst an. Das Ergebnis soll anschließend wie in Bild 1 aussehen. Wir wollen also Schaltflächen zum Ausrichten des Absatzes, zum Einstellen fetter, kursiver und unterstrichener Schrift, zum Umwandeln von Absätzen in Auflistungen oder nummerierte Listen sowie zum Einstellen der Schriftart und -größe vorsehen.

Die Steuerelemente platzieren wir in einer zusätzlichen Zeile des **Grid**-Elements. Damit wir nicht alle Befehle mit Texten ausstatten müssen und da es überdies für die meisten gängigen Formatierungsbefehle gängige Icons gibt, statten wir auch unsere Schaltflächen mit passenden Icons aus. Diese fügen wir dem Unterverzeichnis **images** unseres Projekts hinzu (siehe Bild 2).

Alternative Icons direkt von Microsoft finden Sie etwa unter folgendem Link:

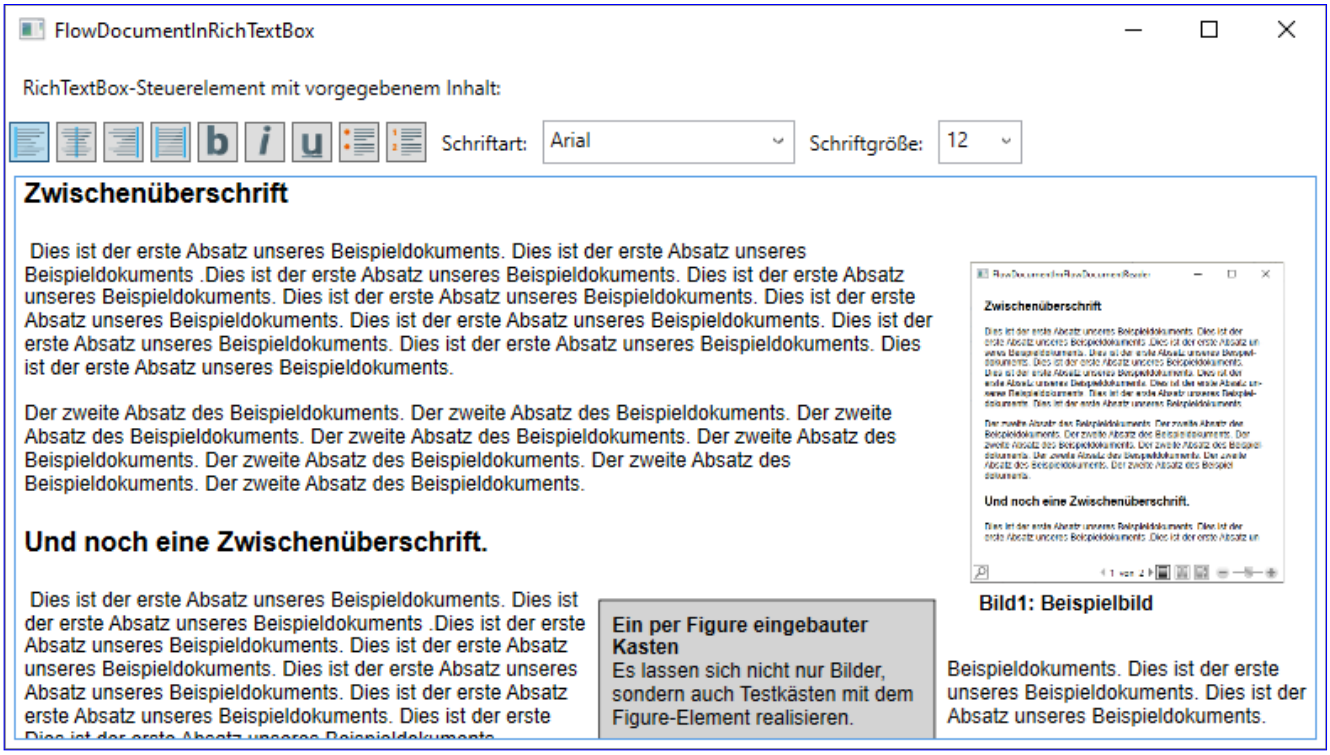

Bild 1: RichTextBox-Steuerelement mit Formatierungs-Menü

https://www.microsoft.com/en-us/download/details.aspx?id=35825

Wir verwenden jedoch die Icons von der Seite **www.iconexperience.com**.

Der neuen Zeile im **Grid**-Element fügen wie ein **StackPanel**-Element mit horizontaler Ausrichtung hinzu, das vier **Button**-Elemente aufnimmt. Diese statten wir jeweils mit einer Bezeichnung wie **btnTextAlignLeft**, **btnTextAlignCenter**, **btnTextAlignRight** und **btnTextAlignJustified** aus und legen für das **Click**-Ereignis einer jeden Schaltfläche eine passende Ereignismethode an:

```
<StackPanel Orientation="Horizontal" Grid.Row="1">
    <Button x:Name="btnTextAlignLeft" Margin="2" Width="24" Height="24" HorizontalAlignment="Left"
            Click="BtnTextAlignLeft_Click">
        <Image Source="images/text_align_left.ico"></Image>
    </Button>
    <Button x:Name="btnTextAlignCenter" Margin="2" Width="24" Height="24" HorizontalAlignment="Left"
            Click="BtnTextAlignCenter_Click">
        <Image Source="images/text_align_center.ico"></Image>
    </Button>
    <Button x:Name="btnTextAlignRight" Margin="2" Width="24" Height="24" HorizontalAlignment="Left"
            Click="BtnTextAlignRight_Click">
        <Image Source="images/text_align_right.ico"></Image>
    </Button>
    <Button x:Name="btnTextAlignJustified" Margin="2" Width="24" Height="24"
            HorizontalAlignment="Left" Click="BtnTextAlignJustified_Click">
        <Image Source="images/text_align_justified.ico"></Image>
    </Button>
</StackPanel>
```

Die dadurch ausgelösten Methoden implementieren wir im Code behind-Modul des XAML-Fensters:

```
Public Class FlowDocumentInRichTextBox
    Private Sub BtnTextAlignCenter_Click(sender As Object, _
            e As RoutedEventArgs)
        EditingCommands.AlignCenter.Execute(vbNull, Me.rtb)
    End Sub

    Private Sub BtnTextAlignJustified_Click(sender As Object, _
            e As RoutedEventArgs)
        EditingCommands.AlignJustify.Execute(vbNull, Me.rtb)
    End Sub
```

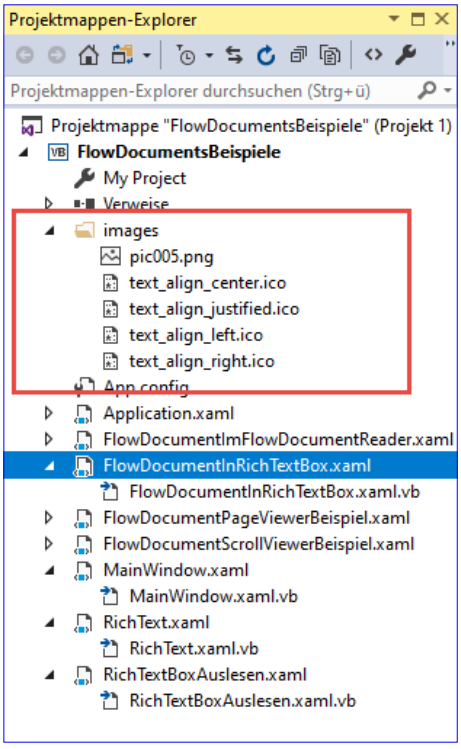

Bild 2: Icons im Projektmappen-Explorer

```
    Private Sub BtnTextAlignLeft_Click(sender As Object, e As RoutedEventArgs)
        EditingCommands.AlignLeft.Execute(vbNull, Me.rtb)
    End Sub

    Private Sub BtnTextAlignRight_Click(sender As Object, e As RoutedEventArgs)
        EditingCommands.AlignRight.Execute(vbNull, Me.rtb)
    End Sub
End Class
```

Die Methoden rufen jeweils eine Methode der Klasse **EditingCommands** auf, in diesem Fall **AlignCenter**, **AlignJustify**, **AlignLeft** und **AlignRight**. Diese wird mit der **Execute**-Methode ausgeführt, wobei wir mit dem zweiten Parameter dieser Methode das Ziel der Methode angeben – in diesem Fall das **RichTextBox**-Element.

Das gleiche Ergebnis erhalten wir übrigens, wenn wir die entsprechenden **Command**-Elemente dem Attribut **Command** der Schaltflächen zuweisen. Dazu fügen wir einen neuen Satz mit vier weiteren Schaltflächen zum **.xaml**-Code hinzu. Jede Schaltfläche erhält das Attribut **Command** mit dem Befehl, den wir zuvor in der Ereignisprozedur aufgerufen haben – zum Beispiel **EditingCommands.AlignLeft**.

Das allein bewirkt allerdings noch nichts, wenn wir nicht angeben, auf welches Steuerelement sich die Aktion auswirken soll. Dazu stellen wir das Attribut **CommandTarget** jeweils auf den Wert **{Binding ElementName=rtb}** ein, wobei **rtb** der Name des **RichTextBox**-Steuerelements ist:

```
<Button x:Name="btnTextAlignLeftCommand" Margin="2" Width="24" Height="24" HorizontalAlignment="Left"
Command="EditingCommands.AlignLeft" CommandTarget="{Binding ElementName=rtb}">
    <Image Source="images/text_align_left.ico"></Image>
</Button>
<Button x:Name="btnTextAlignCenterCommand" Margin="2" Width="24" Height="24" HorizontalAlignment="Left"
Command="EditingCommands.AlignCenter" CommandTarget="{Binding ElementName=rtb}">
    <Image Source="images/text_align_center.ico"></Image>
</Button>
<Button x:Name="btnTextAlignRightCommand" Margin="2" Width="24" Height="24" HorizontalAlignment="Left"
Command="EditingCommands.AlignRight" CommandTarget="{Binding ElementName=rtb}">
    <Image Source="images/text_align_right.ico"></Image>
</Button>
<Button x:Name="btnTextAlignJustifiedCommand" Margin="2" Width="24" Height="24" HorizontalAlignment="Left"
Command="EditingCommands.AlignJustify" CommandTarget="{Binding ElementName=rtb}">
    <Image Source="images/text_align_justified.ico"></Image>
</Button>
```

Das Ergebnis sieht wie in Bild 3 aus (hier zunächst nur mit dem ersten Satz der vier Schaltflächen – die übrigen ergänzen wir auf den folgenden Seiten).

Damit Sie schnell im **RichTextBox**-Element arbeiten können, können Sie die meisten Befehle auch per eingebauter Tastenkombination aufrufen – für die wichtigsten drei der vier oben vorgestellten Befehle beispielsweise mit den folgenden Tastenkombinationen:

- **AlignCenter** (**Strg + E**): Zentrierte horizontale Ausrichtung
- **AlignLeft** (**Strg + L**): Linkszentrierte horizontale Ausrichtung
- **AlignRight** (**Strg + R**): Rechtszentrierte horizontale Ausrichtung

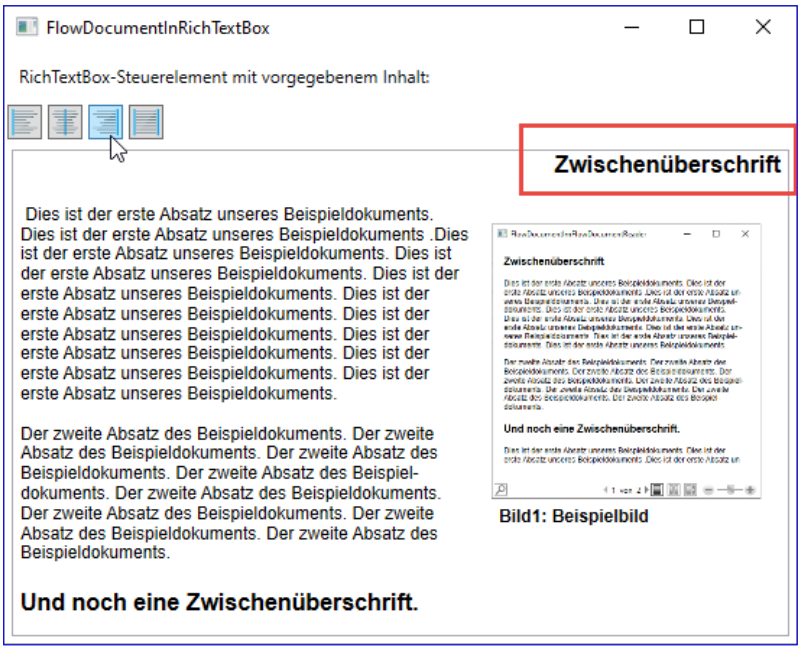

Bild 3: Ausrichten von Text mit selbstgebauten Schaltflächen

Die übrigen Befehle und Tastenkombinationen können Sie beispielsweise dieser Seite entnehmen:

```
https://docs.microsoft.com/en-us/dotnet/api/system.windows.documents.editingcommands?view=netcore-3.1
```

Neben den vier Schaltflächen zum Einstellen der Ausrichtung des aktuellen Absatzes benötigen wir noch weitere Steuerelemente. Zunächst wollen wir uns um die Steuerelemente zum Formatieren von Text kümmern (siehe Bild 4).

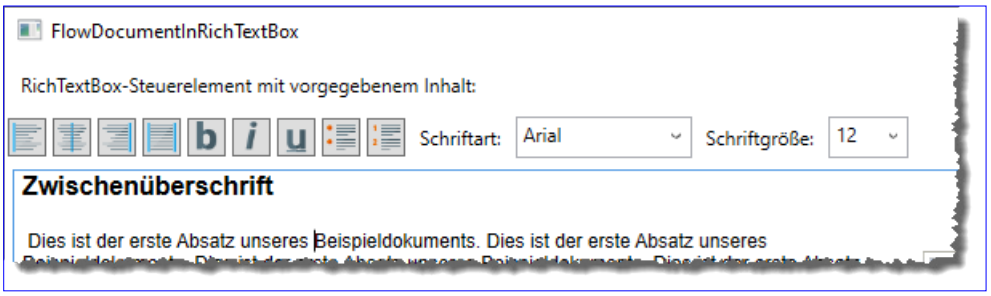

Bild 4: Weitere Steuerelemente zum Formatieren von Text

Diese fügen wir wie folgt hinzu und statten diese mit den entsprechenden **Command**-Befehlen aus. Wir starten mit den drei Schaltflächen für die fette, kursive und unterstrichene Zeichenformatierung, die wir mit folgendem XAML-Code definieren:

```
<StackPanel Orientation="Horizontal" Grid.Row="1">
    ...
    <Button x:Name="btnTextBold" Margin="2" Width="24" Height="24" HorizontalAlignment="Left"
            Command="EditingCommands.ToggleBold" CommandTarget="{Binding ElementName=rtb}">
        <Image Source="images/font_style_bold.ico"></Image>
    </Button>
```

```
<Button x:Name="btnTextItalic" Margin="2" Width="24" Height="24" HorizontalAlignment="Left"
        Command="EditingCommands.ToggleItalic" CommandTarget="{Binding ElementName=rtb}">
    <Image Source="images/font_style_italics.ico"></Image>
</Button>
<Button x:Name="btnTextUnderline" Margin="2" Width="24" Height="24" HorizontalAlignment="Left"
        Command="EditingCommands.ToggleUnderline" CommandTarget="{Binding ElementName=rtb}">
    <Image Source="images/font_style_underline.ico"></Image>
</Button>
```

Hier sind keine umfangreichen Erläuterungen nötig – die Elemente arbeiten wie die zuvor für die Ausrichtung der Absätze angelegten Elemente. Das gleiche gilt für die folgenden beiden Schaltflächen, welche die Auflistungen und die Nummerungen ermöglichen:

```
<Button x:Name="btnBullets" Margin="2" Width="24" Height="24" HorizontalAlignment="Left"
        Command="EditingCommands.ToggleBullets" CommandTarget="{Binding ElementName=rtb}">
    <Image Source="images/list_style_bullets.ico"></Image>
</Button>
<Button x:Name="btnNumbering" Margin="2" Width="24" Height="24" HorizontalAlignment="Left"
        Command="EditingCommands.ToggleNumbering" CommandTarget="{Binding ElementName=rtb}">
    <Image Source="images/list_style_numbered.ico"></Image>
</Button>
```

Spannender wird es bei den folgenden beiden Elementen, den Kombinationsfelder zur Auswahl der Schriftart und der Schriftgröße. Zunächst das **ComboBox**-Element **cboSchriftarten**, das wir mit den Schriften des aktuellen Rechners füllen wollen. Es soll nach dem Wechseln eines Eintrags das Ereignis **SelectionChanged** auslösen, für das wir eine Ereignismethode namens **CboSchriftarten_SelectionChanged** hinterlegt haben – diese schauen wir uns gleich im Anschluss an:

```
<Label Margin="2">Schriftart:</Label>
<ComboBox x:Name="cboSchriftarten" Margin="2" Width="150" IsEditable="True"
    SelectionChanged="CboSchriftarten_SelectionChanged"></ComboBox>
```

Das **ComboBox**-Element für die Auswahl der Schriftgröße sieht ähnlich aus. Auch hier hinterlegen wir eine Ereignismethode für das Ereignis **SelectionChanged**:

```
<Label Margin="2">Schriftgröße:</Label>
<ComboBox x:Name="cboSchriftgroessen" Margin="2" Width="50" IsEditable="True"
    SelectionChanged="CboSchriftgroessen_SelectionChanged"></ComboBox>
</StackPanel>
```

Zunächst aber wollen wir die beiden Kombinationsfelder einmal füllen. In der Code behind-Klasse der **.xaml**-Datei legen wir eine Konstruktor-Methode namens **New** an, in der wir zunächst die Komponente initialisieren. Dann füllen wir die beiden

Kombinationsfelder über die **ItemsSource**-Eigenschaft. Die des **comboBox**-Elements **cboSchriftarten** füllen wir mit der Systemauflistung **System.Windows.Media.Fonts.SystemFontFamilies**, welche alle Schriften des aktuellen Systems liefert (siehe Bild 5).

Im Code sieht das wie folgt aus:

```
Public Class FlowDocumentInRichTextBox
    Public Sub New()
        InitializeComponent()
        cboSchriftarten.ItemsSource = _
System.Windows.Media.Fonts.SystemFontFamilies
```

Danach füllen wir die Eigenschaft **ItemsSource** des Kombinationsfeldes **cboSchriftgroessen** mit dem Ergebnis der Property **Schriftgroessen**:

```
        cboSchriftgroessen.ItemsSource = _
            Schriftgroessen
    End Sub
```

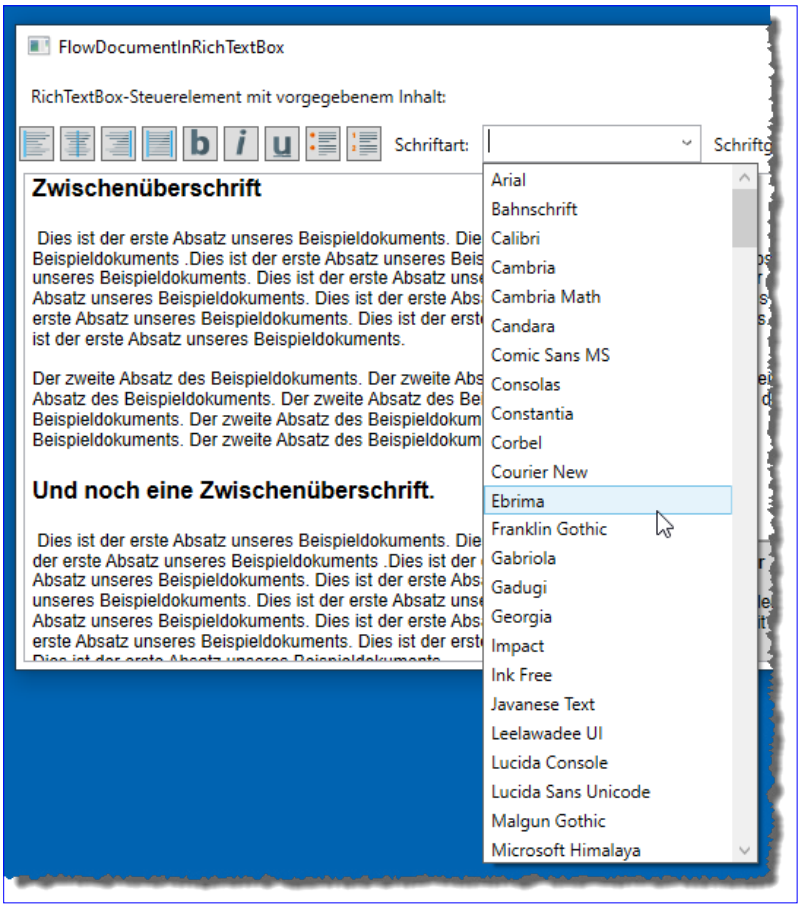

Bild 5: Einstellen der Schriftart

Diese Property deklarieren wir wie folgt. Sie liefert die gängigen Schriftgrößen:

```
    Private _schriftgroessen() As Double

    Public ReadOnly Property Schriftgroessen As Double()
        Get
            Return New Double() {6, 7, 8, 9, 10, 11, 12, 14, 16, 18, 20, 24, 28, 32, 36, 42, 48, 56, 64, 128}
        End Get
    End Property
```

Nun benötigen wir noch die Ereignismethoden, die durch das Auswählen neuer Werte in einem der beiden Kombinationsfelder **cboSchriftarten** und **cboSchriftgroessen** ausgelöst werden. Die für die Schriftart sieht wie folgt aus:

```
    Private Sub CboSchriftarten_SelectionChanged(sender As Object, e As SelectionChangedEventArgs)
        Dim objFontFamily As FontFamily = CType(e.AddedItems(0), FontFamily)
        TextformatierungEinstellen(TextElement.FontFamilyProperty, objFontFamily)
    End Sub
```

Codeschnipsel in Visual Studio

Visual Studio kommt seit einigen Versionen mit einer sehr coolen Funktion, nämlich den Code-Snippets. Diese können Sie auf verschiedene Arten in den Code einfügen. Zum Beispiel, indem Sie einen oder mehrere Buchstaben eingeben und dann die Tabulator-Taste betätigen, um den durch diese Buchstaben gekennzeichneten Codeschnipsel im Code einzufügen. In diesem Artikel schauen wir uns an, welche Codeschnipsel es schon gibt und vor allem, wie Sie selbst IntelliSense-Code-Snippets definieren können.

IntelliSense-Code-Snippets fallen Ihnen vermutlich nie auf, wenn Sie nicht über einen Tipp im Internet darüber stolpern. Deshalb wollen wir dieses spannende Thema hier aufgreifen!

Wie funktionieren diese IntelliSense-Code-Snippets also nun? Das können Sie einfach ausprobieren, indem Sie etwa in einem VB-Modul einmal die Zeichenfolge **Property** eingeben (siehe Bild 1).

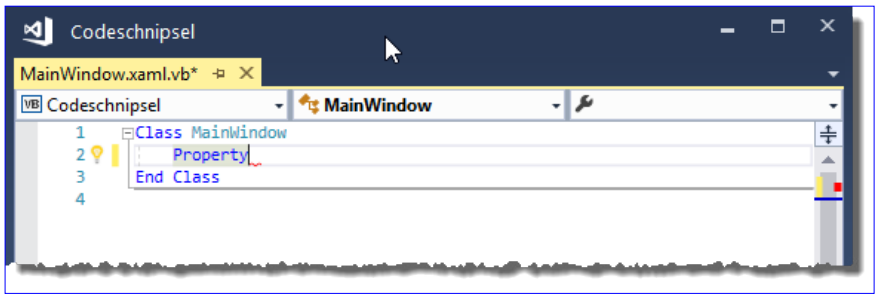

Bild 1: Vorbereiten des Code-Snippets ...

Wenn Sie dann direkt die Tabulator-Taste betätigen, wird das Schlüsselwort durch den für das Code-Snippet angegebenen Code ersetzt – siehe Bild 2.

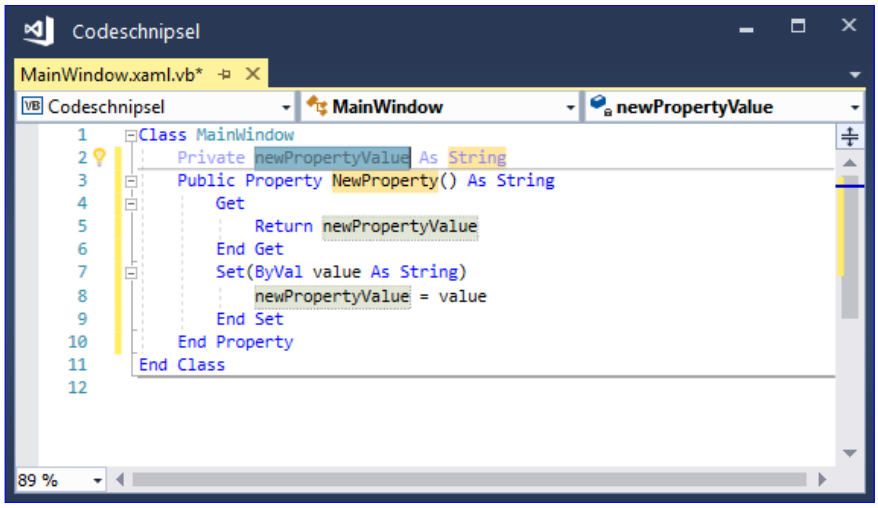

Bild 2: ... und Hinzufügen der gewünschte Code-Elemente

Weitere Einsatzmöglichkeiten von Code-Snippets

Wie und wo können wir Code-Snippets einsetzen? Es gibt zum Beispiel die folgenden Möglichkeiten:

- Einfügen von Code per Kürzel und Betätigen der Tabulator-Taste
- Betätigen der rechten Maustaste und Auswählen des zu verwendenden Codesnippets über den Kontextmenü-Eintrag **Ausschnitt\|Ausschnitt einfügen**.
- Umschließen des markierten Codes mit dem Kontextmenü-Befehl **Umschließen** (nur XAML und C#) und anschließender Eingabe des Kürzels

Umschließen des markierten Codes

Im Code möchte man gelegentlich unter VB eine oder mehrere Anweisungen in Code-Strukturen wie eine Schleife oder eine Bedingung einfassen oder unter XAML ein paar Elemente in ein übergeordnetes Element – beispielsweise, um diese in einem **StackPanel** anzuordnen. Dazu bietet Visual Studio die Möglichkeit an, den markierten Code von den gewünschten Elementen einfassen zu lassen. Unter Visual Basic funktioniert das leider nicht, sondern nur unter XAML (und unter alternativen Programmiersprachen zu Visual Basic, beispielsweise C#).

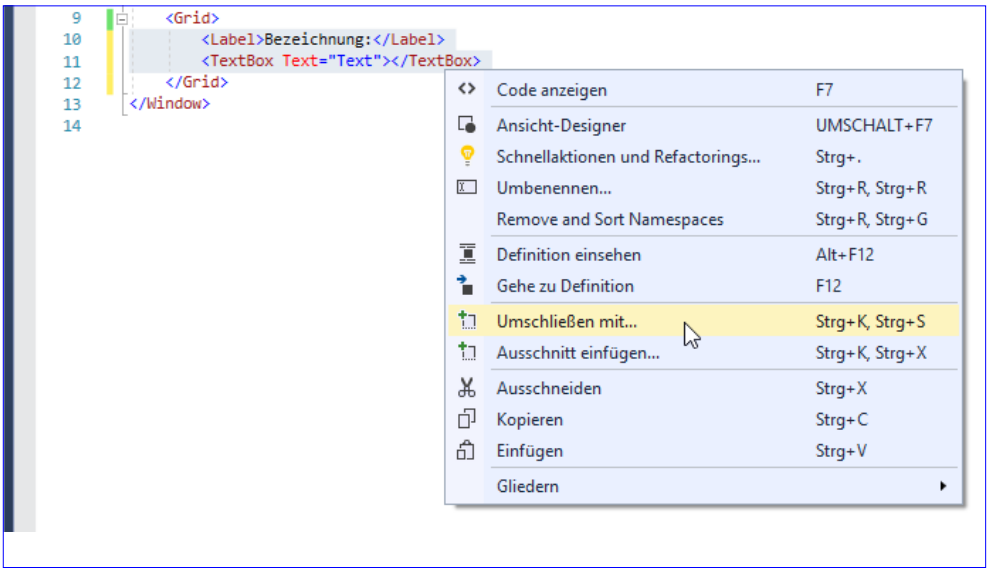

Bild 3: Umschließen von XAML-Elementen

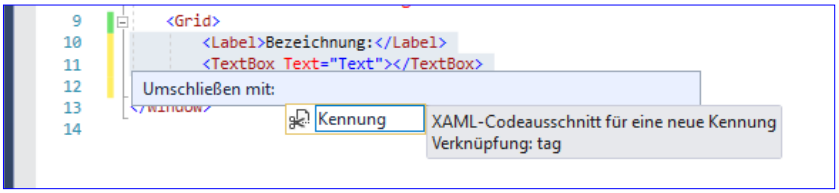

Bild 4: Auswählen des gewünschten Elements

Wenn Sie aber in XAML ein Element einfassen wollen, wie oben beschrieben durch Start- und Endtag eines **StackPanel**-Objekts, dann gehen Sie wie folgt vor:

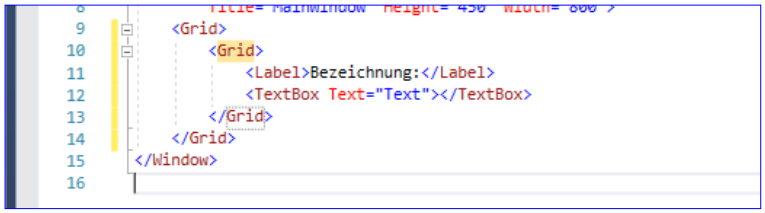

Bild 5: Der gelb markierte Text kann nun durch den Namen des gewünschten Elements ersetzt werden.

- Markieren Sie den einzufassenden Code.
- Öffnen Sie das Kontextmenü für den markierten Bereich (siehe Bild 3).
- Wählen Sie den Eintrag **Umschließen mit...** aus. Alternativ zu diesem und dem vorherigen Schritt betätigen Sie die Tastenkombination **Strg + K**, **Strg + S** (**Strg** gedrückt halten und **K** und **S** nacheinander eingeben).

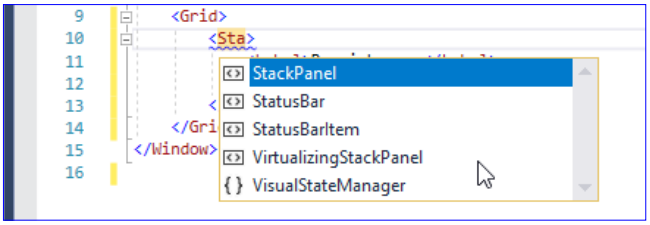

Bild 6: Beim Ersetzen unterstützt dann IntelliSense wieder.

- Es erscheint dann das Eingabefeld **Kennung**. Hier geben Sie die Kennung **tag** ein, denn es gibt nur dieses eine Element (siehe Bild 4).
- Danach wird der Name des öffnenden Elements allerdings mit gelbem Hintergrund angezeigt. Sie können dann den tatsächlich gewünschten Namen dort eingeben und die Bezeichnung **Grid** dadurch ersetzen (siehe Bild 5).

- Beim Ersetzen der markierten Elementbezeichnung hilft dann wieder IntelliSense (siehe Bild 6).

Code-Snippets verwalten

Die Code-Snippets können Sie in einem Dialog namens **Code-ausschnitt-Manager** verwalten, den Sie über den Menüeintrag **Extras|Codeausschnitt**-Manager verwalten oder mit der Tastenkombination **Strg + K**, **Strg + B** öffnen.

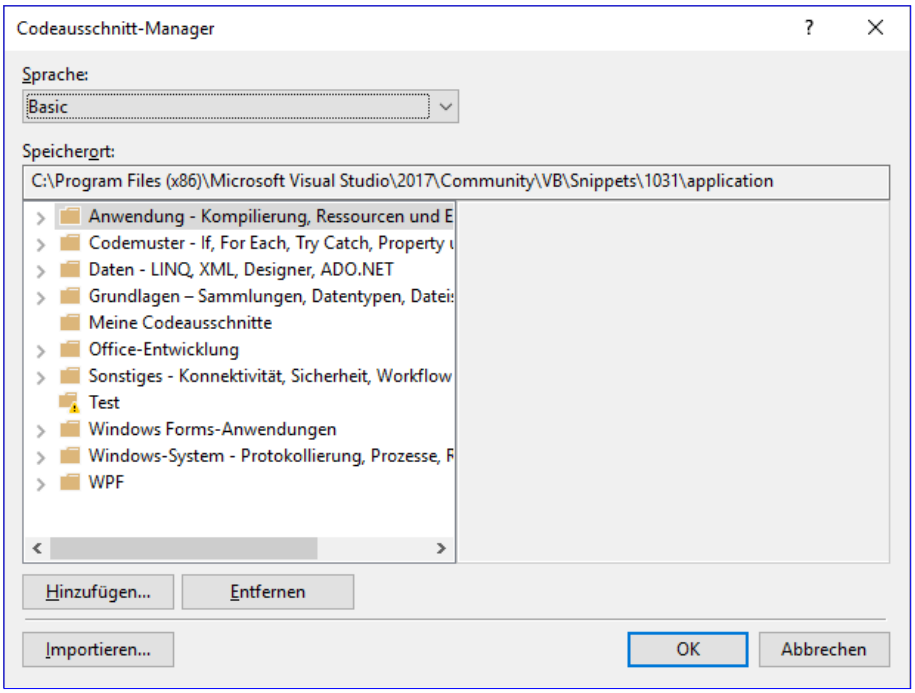

Bild 7: Der Codeausschnitt-Manager direkt nach dem Öffnen

Der Codeausschnitt-Manager sieht direkt nach dem Öffnen wie in Bild 7 aus. Sie finden hier zunächst ein Auswahlfeld für die Auswahl der Sprache, für welche die Code-Snippets angezeigt werden sollen. Der Codeausschnitt-Manager stellt direkt die aktuell verwendete Sprache ein, in diesem Fall **Basic** (für Visual Basic).

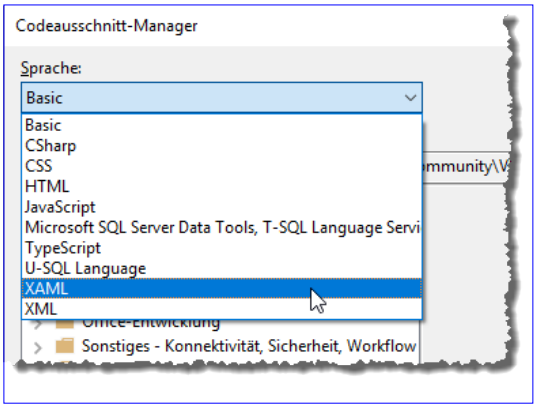

Bild 8: Auswahl der Sprache

Die Auswahl erstreckt sich von C# über Visual Basic bis hin zu **XAML** und **XML** (siehe Bild 8). Je nachdem, welche Sprache Sie auswählen, werden mehr oder weniger Ordner in der darunter befindlichen Liste angezeigt. Für XAML gibt es beispielsweise in Visual Studio 2017 erst ein Snippet namens **Kennung**.

Wenn Sie dieses Snippet auswählen, zeigt das Fenster im rechten Bereich einige Eigenschaften des Snippets an (siehe Bild 9).

Dazu gehören die folgenden Elemente:

- **Beschreibung**: Beschreibungstext zu diesem Code-Snippet
- **Verknüpfung**: Text, den Sie im Editor eingeben, um das Snippet einzufügen.

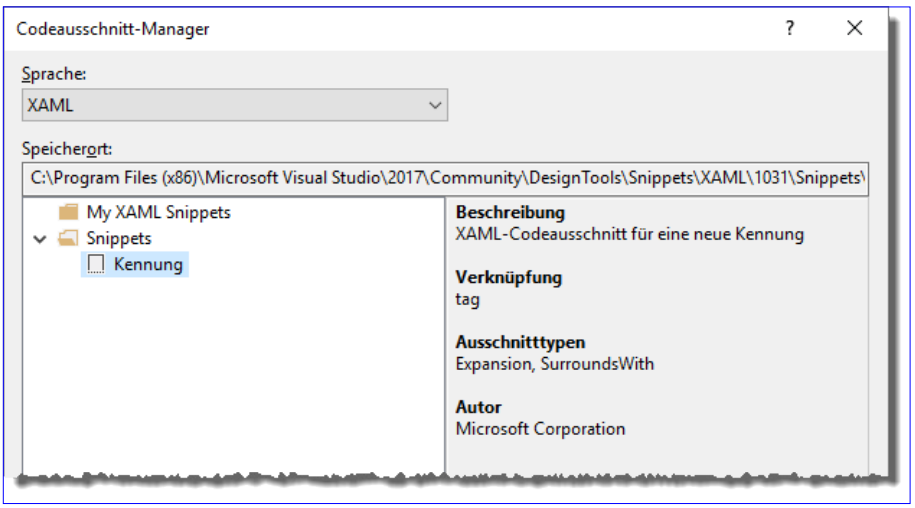

Bild 9: Informationen zu einem Snippet

- **Ausschnittstypen**: Art, wie das Code-Snippet eingefügt wird.
- **Autor**: Autor des Code-Snippets

Im unteren Bereich finden Sie noch einige Schaltflächen:

- **Hinzufügen...**: Öffnet einen Verzeichnisauswahl-Dialog. Damit können Sie alle Codesnippets dieses Verzeichnisses auf einen Rutsch importieren.
- **Entfernen**: Entfernt das aktuell angezeigten Code-Snippet aus der Sammlung der Code-Snippets.
- **Importieren...**: Öffnet einen Dateiauswahl-Dialog zur Auswahl von Dateien mit der Endung **.snippet**, die zum Codeausschnitt-Manager hinzugefügt werden sollen.

Anlegen eines eigenen Code-Snippets

Die Basis für ein neues Code-Snippet ist eine XML-Datei. Diese sieht wie folgt aus:

```
<?xml version="1.0" encoding="utf-8"?>
<CodeSnippets xmlns="http://schemas.microsoft.com/VisualStudio/2005/CodeSnippet">
    <CodeSnippet Format="1.0.0">
        <Header>
            <Title></Title>
        </Header>
        <Snippet>
            <Code Language="">
                <![CDATA[]]>
            </Code>
        </Snippet>
    </CodeSnippet>
</CodeSnippets>
```

Wir werfen einen kurzen Blick auf diese Elemente und wo Sie Informationen einfügen müssen. Die erste Lücke finden wir im Element **Title**. Hier fügen wir testweise einfach den Wert **Beispielschnipsel** ein. Als nächstes sind die Anführungszeichen des Attributes **Language** des Code-Elements leer. Hier tragen Sie die Bezeichnung der Programmiersprache ein, also etwas **CSharp** für C#, **VB** für Visual Basic oder **XAML**. Schließlich fehlt noch der einzufügende Code. Diesen tragen Sie zwischen die inneren eckigen Klammern des **CDATA**-Elements ein.

Im **Header**-Bereich können Sie noch die beiden folgenden Elemente ergänzen:

- **Author**: Gibt den Namen des Autors an.

- **Description**: Erwartet einen Beschreibungstext.

- **Shortcut**: Gibt die Abkürzung an, mit der Sie den Codeschnipsel hinzufügen können – also beispielsweise, indem Sie den Shortcut eingeben und die Tabulator-Taste betätigen.

Diese werden dann im Codeausschnitt-Manager für das entsprechende Element angezeigt.

Wir wollen ein Codesnippet für Visual Basic erstellen, das eine leere **MessageBox.Show("")**-Methode hinzufügt. Also gestalten wir das XML-Dokument wie folgt:

```
<?xml version="1.0" encoding="utf-8"?>
<CodeSnippets xmlns="http://schemas.microsoft.com/VisualStudio/2005/CodeSnippet">
    <CodeSnippet Format="1.0.0">
        <Header>
            <Title>Messagebox</Title>
            <Author>Andre Minhorst</Author>
            <Description>Fuegt eine MessageBox-Anweisung hinzu.</Description>
            <Shortcut>msg</Shortcut>
        </Header>
        <Snippet>
            <Code Language="VB">
                <![CDATA[MessageBox.Show("")]]>
            </Code>
        </Snippet>
    </CodeSnippet>
</CodeSnippets>
```

An dieser Stelle durften wir lernen, dass Umlaute oder Sonderzeichen nicht funktionieren – das Codesnippet wird dann nicht akzeptiert.

Nachdem Sie das Dokument mit der Dateiendung **.snippet** an der gewünschten Stelle gespeichert haben, können Sie es mithilfe des Codeausschnitt-Managers importieren. Dazu betätigen Sie die **Importieren**-Schaltfläche und wählen den Pfad zur **.snippet**-Datei aus. Sofern die Syntax korrekt ist, erscheint der

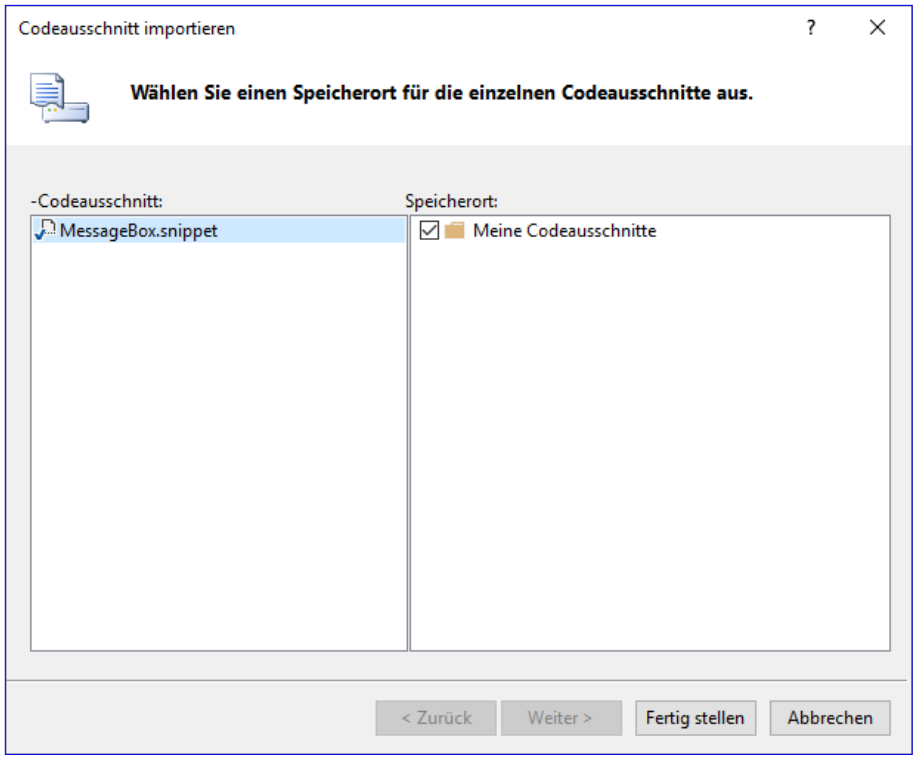

Bild 10: Auswählen des Speicherorts für den Codeschnipsel

Dialog **Codeausschnitte importieren** (siehe Bild 10).

Da sich hier keine allzu großen Auswahlmöglichkeiten ergeben, klicken wir einfach auf **Fertigstellen**. Danach erscheint das Codesnippet unter **Meine Codeausschnitte** in der Liste der Codesnippets (siehe Bild 11). Die für **Description**, **Author** und **Shortcut** angegebenen Werte erscheinen im rechten Bereich, während der Text aus dem **Title**-Element in der Liste erscheint.

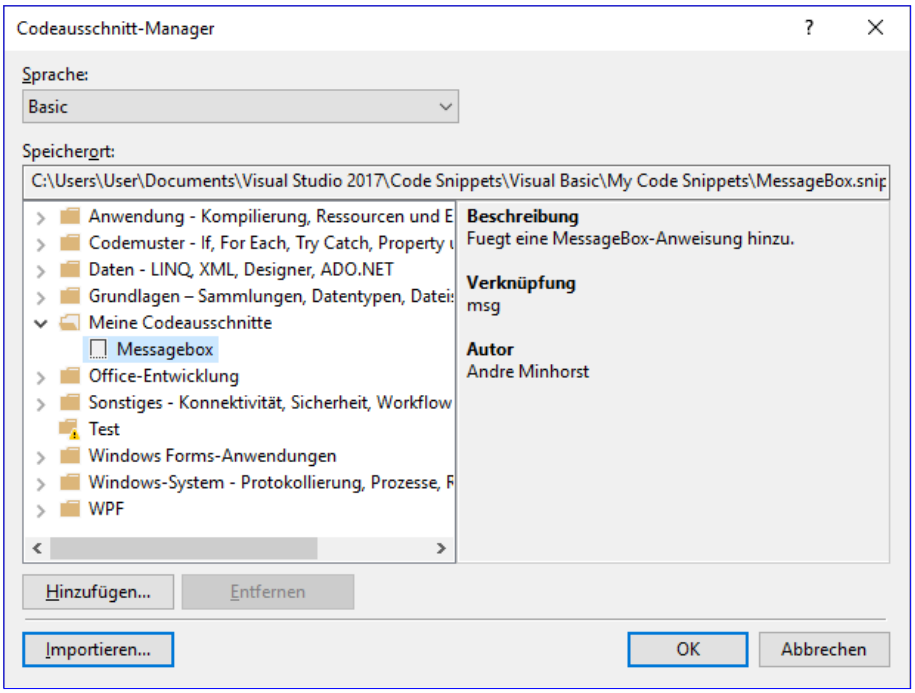

Bild 11: Unser selbst hinzugefügtes Codesnippet

Benutzerdefiniertes Codesnippet testen

Nun wollen wir das Codesnippets ausprobieren. Dazu geben wir in einer Methode eines VB-Moduls die drei Buchstaben **msg** ein und schließen mit der Tabulator-Taste ab. Als Ergebnis erhalten wir allerdings nur:

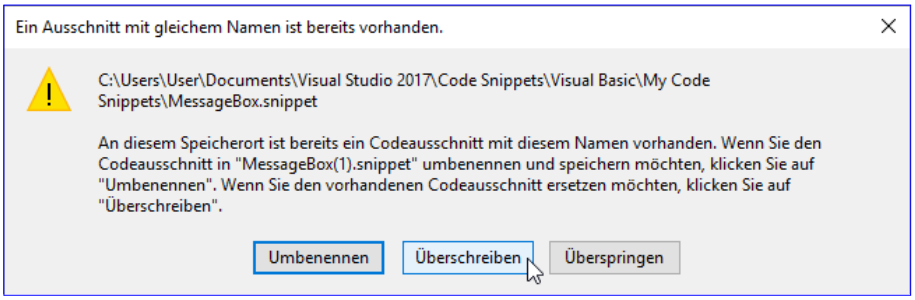

Bild 12: Meldung beim Importieren eines bereits vorhandenen Codesnippets

```
MsgBox
```

Wo ist der Rest geblieben? Wie sich zeigt, gibt es bereits ein anderes Codesnippet mit diesem Kürzel. Also ändern wir unseres in der Originaldatei in **msgs** um, importieren diese erneut und stoßen dabei auf die Meldung aus Bild 12. Hier wählen wir die Option **Überschreiben** aus.

Anschließend probieren es dann mit **msgs** aus. Aber wieder ist der Shortcut bereits vorbelegt – diesmal mit der Klasse **MsgBoxStyle**. Man muss also etwas Fantasie mitbringen, um einen noch freien Shortcut zu finden ... Wir sind schließlich mit **mesx** erfolgreich und können unser Codesnippet damit anwenden:

```
MessageBox.Show("")
```

Zu ersetzende Parameter

Ein weiteres Feature für Codesnippets sind Parameter. Dabei handelt es sich um eine Vereinfachung der Eingabe der Parameter beispielsweise für die **MessageBox**-Anweisung. Wir wollen nun nicht mehr nur eine leere Zeichenfolge als ersten Parameter der **MessageBox.Show**-Methode ausgeben, sondern dem Benutzer direkt die Möglichkeit geben, diese selbst einzugeben.

Onlinebanking mit DDBAC: Testdaten

Wenn Sie auf Basis der DDBAC-Bibliothek und der hier vorgestellten Techniken eine eigene Onlinebanking-Lösung programmieren, wollen Sie vermutlich nicht ständig Testüberweisungen auf Ihren privaten oder geschäftlichen Konten ausführen, um die Funktionalität zu prüfen. In diesem Fall gibt es gute Nachrichten: Die DDBAC stellt eine Testbank mit zwei Testkonten zur Verfügung, die Sie für diese Zwecke nutzen können. Allerdings gibt es ein paar Einschränkungen bei der Nutzung. Welche diese sind und wie Sie an die Daten der Testkonten gelangen, erfahren Sie in diesem Artikel.

Die Testbank und die beiden Testkonten können Sie beispielsweise nutzen, indem Sie für die eine Testbank einen Account im **Administrator für Homebanking Kontakte** hinzufügen, den Sie nach der Installation der DDBAC über die Systemsteuerung öffnen, und die andere als Ziel einer Überweisung angeben.

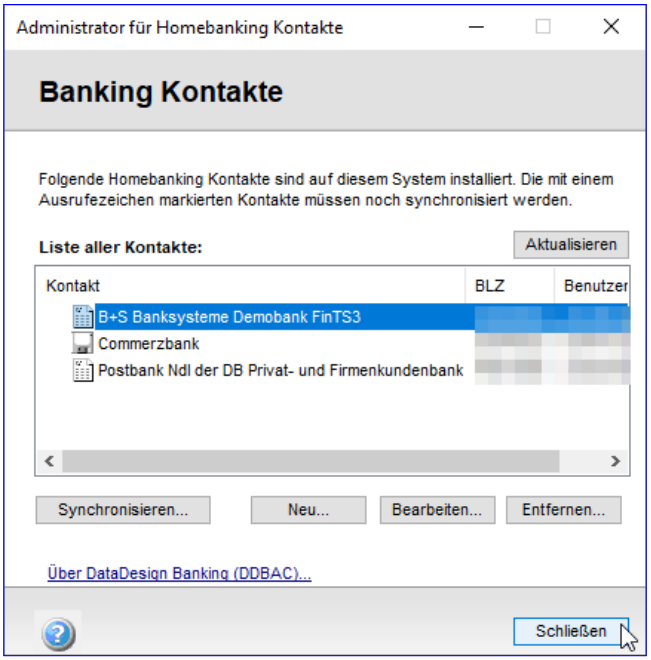

Bild 1: Testzugang im **Administrator für Homebanking Kontakte**

Wenn Sie unter folgendem Link eine DDBAC-Lizenz erworben haben, können Sie per E-Mail an **info@amvshop.de** die Daten der Testkonten anfordern:

```
https://shop.minhorst.com/access-tools/295/ddbac-jahreslizenz?c=78
```

Sie erhalten dann eine Bankleitzahl sowie zwei Kontonummern. Mit der Kombination aus der Bankleitzahl und der ersten Kontonummer legen Sie dann einen neuen Kontakt im Dialog **Banking Kontakte** an. Für das Synchronisieren benötigen Sie auch eine PIN und eine TAN – auch diese erhalten Sie nach Anforderung per E-Mail an **info@amvshop.de**. Nach dem Anlegen sieht der Testzugang im **Administrator für Homebanking Kontakte** wie in Bild 1 aus.

Die Bedingungen für die Nutzung der Testzugänge lauten:

- Sie verwenden nur Verwendungszwecke, die eindeutig als zu Testzwecken erstellt erkennbar sind.
- Sie fügen den Verwendungszwecken ein eindeutiges Element hinzu, damit Ihre Testüberweisungen et cetera identifizieren können, zum Beispiel eine immer gleiche Zahlen- oder Buchstabenkombination.
- Keine Werbung oder strafbaren Inhalte im Verwendungszweck.

Onlinebanking mit DDBAC: FinTS-Registrierung

Wenn Sie Onlinebanking-Lösungen, die Sie mit der DDBAC-Bibliothek realisiert haben, auch in Zukunft ohne Probleme weiterverwenden wollen, müssen Sie zwei zusätzliche Schritte gehen. Diese hängen, wie viele der aktuellen Änderungen im Bereich des Onlinebankings, mit Sicherheitsgründen zusammen. In diesem Fall müssen alle Onlinebanking-Anwendungen bei einer Registrierungsstelle gemeldet werden. Sie erhalten dann eine sogenannte FinTS-Registrierungsnummer, die auch an die Banken verteilt wird. Bei jedem Zugriff über Ihre selbst programmierte Onlinebanking-Anwendung müssen Sie diesen Schlüssel dann an den Bankserver übermitteln, wodurch dieser erkennen kann, welche Anwendung von welchem Hersteller hier zum Einsatz kommt.

Onlinebanking-Anwendung registrieren

Der erste Schritt ist das Programmieren einer eigenen Onlinebanking-Software. Wie das gelingt, erläutern wir in den übrigen Artikeln der Beitragsreihe **Onlinebanking mit DDBAC**.

Wenn Sie eine Onlinebanking-Software eines anderen Herstellers verwenden, wird dieser in der Regel alle notwendigen Schritte erledigen, um diese für die Übermittlung der FinTS-Nummer vorzubereiten. Die Lösung aus der Beitragsreihe **Onlinebanking mit DDBAC** ist noch nicht registriert, da wir die Lösung mit Quellcode liefern.

Die eigene Registrierungsnummer sollte man jedoch nicht weitergeben, da diese gegebenenfalls missbraucht werden kann. Wenn Sie den hier vorgestellten Code also für eine eigene Lösung nutzen wollen, müssen Sie eine eigenen Registrierungsnummer beantragen.

Das ist dann der zweite Schritt, in dem Sie die Registrierungsnummer beantragen. Wie das gelingt, erfahren Sie weiter unten. Nach der Registrierung wird die Nummer auch an die Banken weitergegeben, damit diese Ihre Anwendung an der Registrierungsnummer erkennen können.

Der dritte Schritt ist das Einbauen der Registrierungsnummer so in den Code, sodass diese beim Anmelden an den Bankserver an diesen übergeben wird. Auch diesen Schritt erläutern wir weiter unten im Detail.

Beantragen einer FinTS-Registrierungsnummer

Die FinTS-Registrierungsnummer beantragen Sie unter folgendem Link:

```
https://www.hbci-zka.de/register/prod_register.htm
```

Hier finden Sie ein PDF-Formular, das Sie ausfüllen und an die angegebene E-Mail-Adresse senden. Die Webseite gibt an, dass Sie das Formular, ergänzt um die FinTS-Registrierungsnummer, nach fünf bis zehn Tagen zurückerhalten. Danach kann es allerdings noch wenige Tage dauern, bis diese Registrierungsnummer auch bei allen Banken bekannt ist.

Onlinebanking mit DDBAC II: Überweisungen

Onlinebanking ist aktuell in aller Munde, da es für erhöhte Sicherheit einige Änderungen in den Abläufen gegeben hat. Viele Leser kennen die DDBAC-Bibliothek bereits, denn wir haben in Access im Unternehmen schon darüber berichtet und es ist auch ein Buch zu diesem Thema im André Minhorst Verlag erschienen. Wer diese Bibliothek nutzt, braucht neben einer aktuellen Version auch noch eine kleine Änderung, ohne die Vorgänge nicht mehr durchgeführt werden können. Nachdem wir im ersten Teil der Beitragsreihe das Einlesen von Kontoständen und Umsätzen beschrieben haben, kümmern wir uns nun um das Durchführen von Überweisungen.

Voraussetzungen

Voraussetzung für das Umsetzen der Lösung dieses Artikels ist das Vorhandensein einer DDBAC-Lizenz. Die damit nutzbaren Komponenten sind leider nicht mehr kostenlos verfügbar, sondern müssen lizensiert werden, bevor man diese in seine Produkte einbauen kann. Lizenzen finden Sie im Onlineshop unter **https://shop.minhorst.com/access-tools/295/ddbac-jahreslizenz?c=78**.

Download und Installation

Nach dem Erwerben einer Lizenz erhalten Sie einen Download in ihrem Kundenkonto. Dort finden sie eine Datei namens **DDBACNetWrapper-Version-5-7-65-0.zip** (oder neuer), die Sie entpacken. Darin enthalten ist eine **DDBACSDK.exe**, die Sie auf Ihrem Rechner installieren. Anschließend finden Sie im Verzeichnis **C:\Program Files (x86)\DataDesign\DDBACSDK** die installierten Elemente.

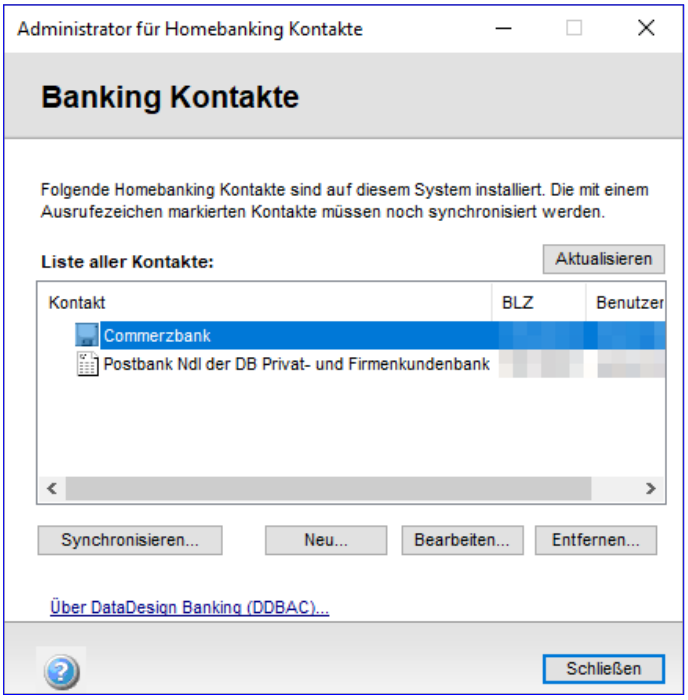

Bild 1: Verwaltung der Homebanking-Kontakte

Test des Quellcodes

Beim Erstellen des Codes für diesen Artikel haben wir das Tool **LinqPad5** zur Programmierung und zum Testen der Funktionalität genutzt. Dieses Tool ist kostenlos. Wir haben dazu im Tool eine neue Datei erstellt und für Language den Wert **VB Program** ausgewählt. Die Methode **Main** wird beim Ausführen aufgerufen, hier haben wir die Aufrufe der einzelnen Beispielmethoden untergebracht.

Überweisung durchführen

Zum Durchführen einer Überweisung verwenden wir diesmal eine **Function**-Methode, da wir einen Wert zurückliefern wollen, der über den Erfolg oder Misserfolg der Überweisung entscheidet. Dieser Funktionswert soll den Datentyp **Boolean** aufweisen. Die benötigten Parameter für eine Überweisung sind die folgenden:

- **lngContact**: Index des Kontakts, für den die Überweisung durchgeführt werden soll (entspricht dem 0-basierten Index der Position des zu wählenden Kontakts aus dem Dialog **Banking Kontakte**, siehe Bild 1)
- **lngAccount**: Index des Kontos des Auftraggebers (entspricht dem 0-basierten Index der Position des zu wählenden Kontos aus dem Dialog Konten verwalten, siehe Bild 2)
- **strEmpfaengername**: Name des Empfängers
- **strEmpfaengerIBAN**: IBAN des Empfänger-Kontos
- **strEmpfaengerBIC**: BIC des Empfänger-Kontos
- **curBetrag**: Zu überweisender Betrag
- **strVerwendungszweck**: Verwendungszweck der Überweisung
- **strMeldung**: Antwort des Systems (mit **ByRef** deklarieren!)

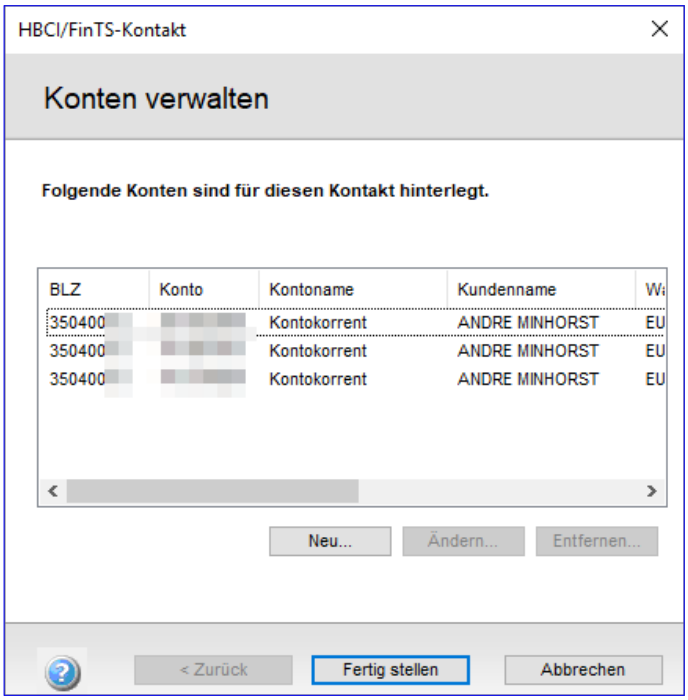

Bild 2: Verwaltung der Konten eines Homebanking-Kontakts

Die nackte Funktion, die wir nachfolgend füllen, sieht nun wie folgt aus:

```
Public Function Ueberweisung(lngContact As Long, lngAccount As Long, strEmpfaengername As String, strEmpfaengerIBAN As String, strEmpfaengerBIC As String, decBetrag As Decimal, strVerwendungszweck As String, ByRef strMeldung As String) As Boolean

End Function
```

BACContact-Objekt für den Auftraggeber ermitteln

Mit dem Parameter **lngContact** wollen wir das **BACContact**-Objekt für den Eintrag im Dialog **Banking Kontakte** auswählen, dessen Index dem Wert dieses Parameters entspricht. Diesen speichern wir in der Variablen **objContact**, die wir wie folgt deklarieren:

```
Dim objContact As BACContact
```

Diese Variable füllen wir dann mit der folgenden Anweisung:

```
objContact = GetContacts.Item(lngContact)
```

Die Funktion **GetContacts** haben wir bereits im ersten Teil der Beitragsreihe namens **Onlinebanking mit DDBAC: Saldo und Umsätze** erläutert. Sie liefert eine Auflistung aller **BACContact**-Elemente, die über die Systemsteuerung hinzugefügt wurden. Über die **Item**-Eigenschaft greifen wir auf das Element mit dem mit **lngContact** übergebenen Indexwert zu.

BACAccount-Objekt für den Auftraggeber ermitteln

Danach benötigen wir das **BACAccount**-Objekt für das Konto, von dem aus die Überweisung durchgeführt werden soll. Dazu legen wir die Variable **objAuftraggeber** an:

```
Dim objAuftraggeber As BACAccount
```

Das **BACAccount**-Objekt ermitteln wir einfach über die **Accounts**-Auflistung des zuvor ermittelten **BACContacts**-Objekts:

```
objAuftraggeber = objContact.Accounts(lngAccount)
```

BACAccount für das Konto des Empfängers erstellen

Während wir das **BACAccount**-Objekt für das Konto des Auftraggebers aus den über die Systemsteuerung abgelegten Daten auslesen konnten, müssen wir das **BACAccount**-Objekt für das Konto des Empfängers neu erstellen und mit den entsprechenden Daten füllen.

Dabei handelt es sich um den Namen, die IBAN und die BIC des Auftraggebers, die wir einem neuen **BACAccount**-Objekt zuweisen. Um diese Daten in einem neuen **BACAccount**-Objekt zusammenzustellen, verwenden wir die folgende Hilfsfunktion **AccountErstellen**:

```
Public Function AccountErstellen(strName1 As String, strIban As String, strBIC As String) As BACAccount
    Dim objAccount As BACAccount
    objAccount = New BACAccount
    With objAccount
        .NameEins = strName1
        .IBAN = strIban
        .BIC = strBIC
    End With
    Return objAccount
End Function
```

Diese erwartet den Namen des Empfängers, die IBAN und die BIC als Parameter und erstellt zunächst ein neues **BACAccount**-Objekt. Dann weist es die Werte der übergebenen Parameter den Eigenschaften **NameEins**, **IBAN** und **BIC** zu. Das so gefüllte **BACAccount**-Objekt wird dann mit der **Return**-Anweisung als Ergebnis der Funktion zurückgegeben.

Die Variable, in der das **BACAccount**-Objekt gespeichert werden soll, heißt **objEmpfaenger** und wird wie folgt deklariert:

```
Dim objEmpfaenger As BACContact
```

Dieses füllen wir über einen Aufruf der zuvor beschriebenen Funktion **AccountErstellen**. Dabei übergeben wir die Werte der drei Parameter **strEmpfaengername**, **strEmpfaengerIBAN** und **strEmpfaengerBIC**:

```
objEmpfaenger = AccountErstellen(strEmpfaengername, strEmpfaengerIBAN, strEmpfaengerBIC)
```

Segmenttyp festlegen und Version ermitteln

Danach speichern wir in einer Variablen namens **strSegment** den Namen des zu verwenden Segments, in diesem Fall **HKCCS**, das der Einreichung einer Einzelüberweisung entspricht:

```
Dim strSegment As String

...
strSegment = "HKCCS"
```

In einer Variablen namens **objSegmentVersion** wollen wir die Version des Segments speichern:

```
Dim objSegmentVersion As BACSegment
```

Dieses ermitteln wir über die Methode **FindOptimalBPDSegment** des **BACAccount**-Elements **objAuftraggeber**, also für die Auftraggeberbank:

```
objSegmentVersion = objAuftraggeber.FindOptimalBPDSegment(strSegment, 0)
```

Abhängig davon, ob ein passendes Segment gefunden werden konnte, steigen wir dann im **If**-Teil der folgenden Abfrage in die eigentliche Überweisung ein oder geben eine Meldung zurück, dass das Segment nicht gefunden werden konnte:

```
If Not objSegmentVersion Is Nothing Then
    Debug.Print ("Segmentversion: " + objSegmentVersion.Version.ToString())
    Return True
Else
    strMeldung = "Segment '" + strSegment + "' konnte nicht gefunden werden."
    Return False
End If
```

HKCCS-Segment erstellen

Nachdem wir die aktuellste Version des **HKCCS**-Segments ermittelt haben, erstellen wir auf dieser Basis das eigentliche **HKCCS**-Segment. Dafür deklarieren wir zwei Variablen – eine für das **BACBanking**-Objekt und eine für das **BACSegment**-Objekt:

```
Dim objBanking As BACBanking
Dim objSegment As BACSegment
```

Das **BACBanking**-Objekt füllen wir mithilfe der Funktion **GetBanking**, die wir gegenüber der Version aus dem ersten Teil der Artikelserie noch geändert haben – wie, lesen Sie im Artikel **Onlinebanking mit DDBAC: FinTS-Registrierung**:

```
Set objBanking = GetBanking
```

Dann nutzen wir die Methode **NewSegment** des **BACBanking**-Objekts und übergeben dieser den Namen des zu erzeugenden Segments, nämlich **HKCCS**, und die soeben ermittelte Version:

```
Set objSegment = objBanking.NewSegment(strSegment, objSegmentVersion.Version)
```

Erfassen der Auftragsdaten

Die Auftragsdaten verarbeiten wir in einem Objekt des Typs **BACSepaMessage**. Dafür deklarieren wir die folgende Objektvariable namens **objSepaMessage**:

```
Dim objSepaMessage As BACSepaMessage
```

Im Code instanzieren wir diese wie folgt:

```
Set objSepaMessage = New BACSepaMessage
```

Das **BACSepaMessage**-Objekt erstellen wir und füllen dann seine Eigenschaft **Segment** mit dem Auftragssegment aus **objSegment**, das wir vorher auf Basis des zu verwendenden Kontos angelegt haben:

```
objSepaMessage = New BACSepaMessage
objSepaMessage.Segment = objSegment
```

Außerdem legen wir dafür über die Eigenschaft **FromAccount** das Konto fest, von dem aus das Geld überwiesen werden soll. Dieses befindet sich in der Variablen **objAuftraggeber**:

```
objSepaMessage.FromAccount = objAuftraggeber
```

Dem **BACSepaMessage**-Objekt fügen wir wiederum ein **BACSepaOrder**-Element hinzu, und zwar mit der **Add**-Methode der **Orders**-Auflistung von **objSepaMessage**. Das Ergebnis referenzieren wir mit der Variablen **objSepaOrder**, die wir zuvor noch deklarieren:

```
Dim objSepaOrder As BACSepaOrder
....
Set objSepaOrder = objSepaMessage.Orders.Add
```

Das **BACSepaOrder**-Objekt füllen wir außerdem noch mit den typischen Informationen für eine Überweisung, nämlich den zu überweisenden Betrag mit der Eigenschaft **Amount**, die Währung mit **CurrencyCode** und den Verwendungszweck mit **Purpose**: