HAUFE.
Ausgabe 02/2021

ACCESS
IM UNTERNEHMEN

OUTLOOK IN ACCESS INTEGRIEREN

Erhalten Sie vollen Zugriff auf Outlook-Mails und Co. über die Benutzeroberfläche von Access! (ab S. 62)

In diesem Heft:

Outlook in Access integrieren

Outlook und Access – das sind zwei Office-Anwendungen, die man in der Regel getrennt voneinander verwendet. Outlook kümmert sich um Mails, Termine oder Kontakte und mit Access nutzen Sie Datenbankanwendungen. Gelegentlich tauschen die beiden auch Daten aus. In dieser Ausgabe von Access im Unternehmen schauen wir uns eine bisher recht unbekannte Möglichkeit für die Kooperation von Outlook und Access an: die Integration von Outlook-Elementen in das Formular einer Access-Anwendung.

Dieses Thema haben wir bereits vor einigen Jahren einmal in Angriff genommen. Es gibt seit Langem einige ActiveX-Steuerelemente mit Outlook-Bezug, die sich in Access-Formluare einfügen lassen. Jedoch schien damit kein Zugriff auf Outlook möglich zu sein. Vor kurzer Zeit jedoch wendete sich ein Leser mit der Frage an uns, ob wir nicht einmal etwas zu Steuerelementen wie **Outlook View Control** schreiben können. Und er lieferte gleich noch ein lauffähiges Beispiel mit. Nach weiteren Recherchen im Internet haben wir es dann hinbekommen: Ein Formular, dass in einem **Outlook View Control** die Inhalte verschiedener Outlook-Ordner wie Posteingang, Kalender, Aufgaben et cetera anzeigen konnte.

Diese Ausgabe konzentriert sich auf verschiedene Outlook-Techniken – mit Ausnahme des Beitrags von Bernd Jungbluth mit dem Titel **SQL Server-Security – Teil 5: Rechte für Abteilungen** (ab Seite 45). Hier erfahren Sie, wie Sie für die Mitarbeiter verschiedener Abteilungen eines Unternehmens die entsprechenden Berechtigungen festlegen.

Alle anderen Beiträge drehen sich um Microsoft Outlook. Der erste sogar ausschließlich – hier stellen wir eine Technik vor, die uns im Alltag sehr viel Arbeit spart. Es geht um das Verschiebenen von E-Mails aus dem Posteingang in verschiedene Ordner. Dies realisieren wir einfach durch markieren der E-Mail und anschließendes Betätigen einer Tastenkombination wie **Alt + 1**, **Alt + 2** und so weiter. Den Artikel finden Sie unter dem Titel **Outlook-Mails per Tastenkombination verschieben** ab Seite 2.

Im Beitrag **Kunde zu einer E-Mail öffnen** greifen wir ab Seite 12 diese Technik auf und nutzen diese, um per Tastenkombination den Absender der aktuell markierten E-Mail zu ermitteln und die Kundenverwaltung mit dem Kunden, der diese E-Mail-Adresse besitzt, zu öffnen. Damit springen Sie blitzschnell von einer E-Mail zum Kundendatensatz!

Die übrigen Beiträge drehen sich um das **Outlook View Control**. Der Beitrag **Outlook-Folder in Access anzeigen** zeigt ab Seite 19, wie Sie das Steuerelement überhaupt in einem Formular zum Laufen bringen.

Damit Sie den gewünschten Ordner komfortabel auswählen können, fügen wir mit der Anleitung aus dem Beitrag **TreeView für Outlook-Ordner** ab Seite 35 ein **TreeView**-Steuerelement zur Anzeige der Outlook-Ordner hinzu.

Schließlich bauen wir dieses Beispiel im Beitrag E-Mails verwalten mit dem Outlook View Control (ab Seite 62) zu einem Formular aus, mit dem Sie alle üblichen Techniken rund um das Abfragen, Lesen und Beantworten von E-Mails benötigen.

Viel Spaß beim Ausprobieren!

Ihr André Minhorst

Outlook-Mails per Tastenkombination verschieben

Nicht alle eingehenden E-Mails, die man erhält, müssen beantwortet werden. Beispiele sind Bestellbestätigungen, Rechnungen et cetera. Diese wollen Sie aber vielleicht aus dem Posteingangs-Ordner in einen anderen Ordner verschieben, der beispielsweise nur E-Mails enthält, die mit Bestellungen zu tun haben. Das gelingt per Drag and Drop relativ schnell. Noch besser wäre aber eine Tastenkombination, mit der wir die Mails in die Zielordner verschieben könnten. Dann brauchen Sie beim Durchgehen des Posteingangs die Hände gar nicht mehr von der Tastatur zu nehmen. Die hier vorgestellte Lösung berührt zwar nicht die Datenbankanwendung Microsoft Access, um die es eigentlich in diesem Magazin geht, aber diese Lösung für effizienteres Arbeiten wollen wir Ihnen nicht vorenthalten.

Es gibt sicher Menschen, für die es in Bezug auf die E-Mails im Posteingang von Outlook nur zwei Möglichkeiten gibt: Beantworten und löschen oder direkt löschen. Die meisten müssen aber vielleicht geschäftliche und andere E-Mails aufbewahren oder machen das einfach aus praktischen Gründen. Zum Beispiel ist es sinnvoll, alle E-Mails, die mit Onlinebestellungen zusammenhängen, aufzubewahren.

Dazu bietet es sich unter Outlook an, weitere Ordner unterhalb des Ordners **Posteingang** anzulegen und diesen Bezeichnungen wie beispielsweise **Bestellungen** zu geben (siehe Bild 1). E-Mails, die sich im Posteingang befinden und die mit Bestellungen in Zusammenhang stehen, können Sie dann mit der Maus per Drag and Drop in diesen Ordner verschieben.

Wenn Sie jedoch viele solcher E-Mails bekommen, die Sie schnell einem bestimmten Ordner unterhalb des Posteingang-Ordners zuweisen wollen, wird das ewige Drag and Drop schnell anstrengend.

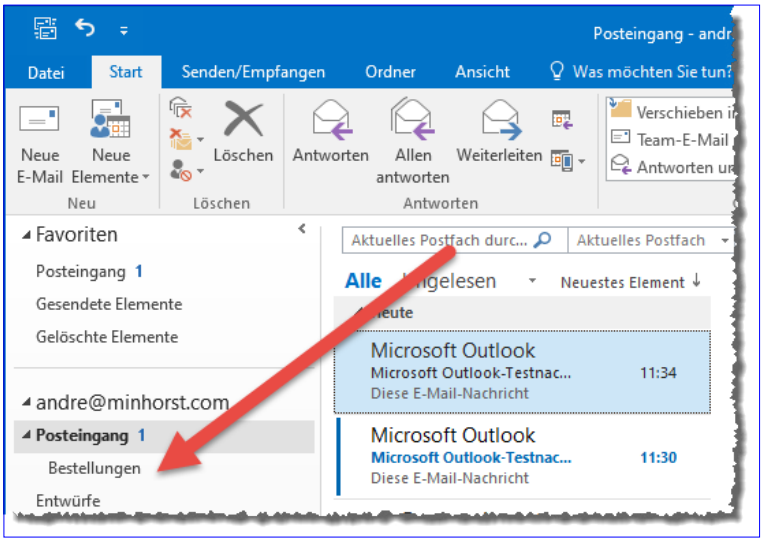

Bild 1: Bestellungen-Ordner in Outlook

E-Mails verschieben per Tastenkombination

Also haben wir uns überlegt, wie wir solche Arbeitsschritte vereinfachen können. Die naheliegendste Idee ist der Einsatz einer Tastenkombination, mit der wir eine von uns für diesen Zweck angelegte VBA-Prozedur aufrufen wollen. Die Idee mit der VBA-Prozedur können wir umsetzen, aber es gibt keine direkte Möglichkeit in Outlook, eine solche per Tastenkombination aufzurufen. Daher müssen wir einen kleinen Umweg gehen: Wir können VBA-Prozeduren nämlich mit benutzerdefinierten Schaltflächen im Ribbon aufrufen, und dieses kann mit einigen Einschränkungen per Tastenkombination gesteuert werden.

Wir werden in den nächsten Abschnitten zunächst die benötigten VBA-Prozeduren erstellen und dann die Ribbon-Einträge samt Tastenkombinationen hinzufügen.

Modul im VBA-Projekt von Outlook anlegen

Im Gegensatz zu Access, wo jede Datenbankdatei ein eigenes VBA-Projekt aufweist, gibt es für Outlook ein zentrales VBA-Projekt. Schließlich gibt es in Outlook keine zu öffnenden Dateien beziehungsweise Dokumente wie in den anderen Office-Anwendungen.

Den VBA-Editor mit diesem VBA-Projekt zeigen Sie von Outlook aus mit der Tastenkombination **Alt + F11** an. Hier finden Sie standardmäßig nur den Ordner **Microsoft Outlook Objekte** mit der Klasse **ThisOutlookSession**. Um Prozeduren anzulegen, fügen wir über den Menüeintrag **EinfügenIModul** ein neues Standardmodul hinzu und nennen dieses **mdlMailsAndFolders**. Dieses Modul wir dann im Ordner **Module** angezeigt.

E-Mail per VBA verschieben

Die geplante VBA-Prozedur soll alle zum Zeitpunkt des Aufrufs markierten Einträge des Posteingangs in den mit **strFolder** angegebenen Zielordner verschieben (siehe Listing 1).

Dazu deklarieren wir verschiedene Variablen, die wir anschließend füllen. Als Erstes referenzieren wir mit der Variablen **objExplorer** das mit **ActiveExplorer** ermittelte Fenster zur Ansicht von Ordnerinhalten unter Outlook.

Dann weisen wir den Standardordner des **Explorer**-Objekts der Variablen **objCurrentFolder** zu und prüfen, ob der Name des Ordners **Posteingang** lautet. Wenn Sie eine englische Version von Office nutzen, müssen Sie diese Zeile gegebenenfalls noch anpassen. Hat der Ordner den Namen **Posteingang**, arbeitet die Prozedur die Anweisungen in der **If...Then**-Bedingung ab.

Hier referenzieren wir die ausgewählten Elemente mit der Variablen **objSelection** und den aktuellen MAPI-Namespace mit **objNamespace**. Leider kann man nicht direkt über den Namen auf **Folder**-Objekte zugreifen, die nicht in der ersten Ebene liegen (also unterhalb von **Posteingang**).

Deshalb benötigen wir eine Hilfsfunktion namens **GetFolder**, um den mit **strTargetfolder** angegebenen Ordner zu erhalten. Dieser Funktion, die wir weiter unten beschreiben, übergeben wir die Variablen **objNamespace** und **strTargetfolder** und erhalten ein **Folder**-Objekt mit dem Zielordner zurück.

```
Public Sub MoveToFolder(strTargetfolder As String)
    Dim objMailItem As MailItem
    Dim objExplorer As Explorer
    Dim objSelection As Selection
    Dim objTargetFolder As Folder
    Dim objCurrentFolder As Folder
    Dim objNamespace As NameSpace
    Dim i As Integer
    Set objExplorer = Application.ActiveExplorer
    Set objCurrentFolder = objExplorer.CurrentFolder
    If objCurrentFolder.Name = "Posteingang" Then
        Set objSelection = objExplorer.Selection
        Set objNamespace = Application.GetNamespace("MAPI")
        Set objTargetFolder = GetFolder(objNamespace, strTargetfolder)
        For i = 1 To objSelection.count
            Select Case TypeName(objSelection.Item(i))
                Case "MailItem"
                    Set objMailItem = objSelection.Item(i)
                    objMailItem.Move objTargetFolder
            End Select
        Next i
    End If
End Sub
```

Listing 1: Verschieben der aktuell markierten E-Mails in den mit **strFolder** angegebenen Ordner

Danach durchläuft die Prozedur in einer **For...Next**-Schleife über die Zahlen von 1 bis zur Anzahl der markierten Elemente alle betroffenen Einträge. Nach einer Prüfung, ob es sich bei dem jeweiligen Eintrag um eines mit dem Typ **MailItem** handelt, weisen wir dieses der Variablen **objMailItem** zu. Für dieses rufen wir dann die Methode **Move** auf und geben mit **objTargetfolder** den Zielordner für das Verschieben an.

Die Funktion GetFolder

Die Funktion aus Listing 2 erwartet die Angabe des zu verwendenden **Namespace**-Objekts sowie des Pfades zum zu ermittelten Ordner als Parameter.

Dieser Pfad lautet beispielsweise für den Ordner **Posteingang** wie folgt:

```
\\Outlook\Posteingang
```

Wenn Sie einen Ordner namens **Bestellungen** verwenden wollen, der sich direkt im Ordner **Posteingang** befindet, lautet der Pfad:

```
\\Outlook\Posteingang\Bestellungen
```

Nachfolgend wollen wir die einzelnen Elemente des Pfades in ein Array übertragen und dabei die **Split**-Funktion nutzen, um den Pfad an den Backslash-Zeichen aufzuspalten. Dazu müssen wir zunächst die eventuell vorn angegebenen beiden Backslash-Zeichen entfernen. Ob diese Zeichen angegeben wurden, prüfen wir mit einer ersten **If...Then**-Bedingung und entfernen diese gegebenenfalls.

Um beim obigen Beispiel zu bleiben, erhalten wir nun die folgende Zeichenkette:

```
Outlook\Posteingang\Bestellungen
```

Mit der **Split**-Anweisung fügen wir nun die durch das Backslash-Zeiten getrennten Teilzeichenketten in die Elemente eines Arrays namens **strFolders**.

Nun weisen wir der Variablen **objFolder** den Ordner zu, der den Namen des ersten Elements des Arrays enthält, in diesem Fall **Outlook**. Sofern das Array **strFolders** mehr als ein Element enthält, was der Fall ist, wenn die Differenz aus dem Index des letzten Elements und dem Index des ersten Elements größer als **0** ist, enthält der Pfad einen oder mehrere Ebenen mit Unterordnern.

```
Public Function GetFolder(objNamespace As NameSpace, strSubfolder As String) As Folder
    Dim objFolder As Folder
    Dim strFolders() As String
    Dim i As Integer
    If Left(strSubfolder, 2) = "\\" Then
        strSubfolder = Mid(strSubfolder, 3)
    End If
    strFolders = Split(strSubfolder, "\")
    Set objFolder = objNamespace.Folders(strFolders(0))
    If UBound(strFolders) - LBound(strFolders) > 0 Then
        For i = LBound(strFolders) + 1 To UBound(strFolders)
            Set objFolder = objFolder.Folders(strFolders(i))
        Next i
    End If
    Set GetFolder = objFolder
End Function
```

Listing 2: Ermitteln eines Folder-Objekts anhand des Pfades

Diese wollen wir nun durcharbeiten, bis wir am letzten angegebenen Ordner angelangt sind. Die dazu angelegte **For...Next**-Schleife durchlaufen wir vom zweiten bis zum letzten Element des Arrays.

Darin weisen wir der Variablen **objFolder** jeweils den Ordner zu, der den Namen aus **strFolders(i)** enthält und sich in dem zuvor mit **objFolder** referenzierten **Ordner**-Element befindet.

In unserem Beispiel referenzieren wir im ersten Durchlauf der Schleife das Element **Posteingang** mit der Variablen **objFolder** und im zweiten Durchlauf das Element **Bestellungen**.

Der nach dem Verlassen der Schleife in **objFolder** befindliche Ordner wird schließlich als Funktionsergebnis zurückgeliefert.

Prozedur MoveToFolder aufrufen

Solange wir noch keine Ribbon-Schaltflächen und damit auch noch keine Tastenkombination zu ihrem Aufruf angelegt haben, testen wir die Prozedur vom VBA-Editor aus.

Dazu können Sie, nachdem Sie eine zu verschiebende E-Mail im Posteingang markiert haben, einen Aufruf wie den folgenden im Direktbereich des VBA-Editors absetzen:

```
MoveToFolder "\\Outlook\Posteingang\Bestellungen"
```

Dadurch wird die aktuell markierte E-Mail in den angegebenen Zielordner verschoben. Sie können testweise auch einmal mehrere Elemente gleichzeitig markieren und auf diese Weise verschieben.

Startprozedur hinzufügen

Für den Aufruf über eine Ribbon-Schaltfläche eignen sich nur öffentlich deklarierte Prozeduren ohne Parameter. Also benötigen wir noch eine Wrapper-Prozedur, welche den Parameter für uns übergibt:

```
Public Sub MoveToFolder_Bestellungen()
    MoveToFolder "\\Outlook\Posteingang\Bestellungen"
End Sub
```

Warum bauen wir den Zielordner nicht direkt in die Prozedur **MoveToFolder** ein? Weil Sie gegebenenfalls nicht nur einen Zielordner verwenden wollen, sondern vielleicht auch mehrere.

Dann brauchen Sie für jeden neuen Zielordner nur jeweils eine neue Wrapper-Prozedur hinzuzufügen.

Zielordner ermitteln

Wenn Sie den Pfad zum Zielordner zuverlässig ermitteln wollen, können Sie die folgende Anweisung vom Direktbereich des VBA-Editors aus aufrufen:

```
? Application.GetNamespace("MAPI").PickFolder.FolderPath
```

Dies zeigt den Dialog aus Bild 2 an, mit dem Sie den gewünschten Ordner auswählen können. Die Anweisung gibt den Pfad zum angegebenen Ordner im Direktbereich aus.

Prozedur per Ribbon aufrufen

Auf dem Weg, die Prozedur durch eine Tastenkombination aufrufbar zu machen, benötigen wir eine Ribbon-Schaltfläche. Hier gibt es zwei Möglichkeiten:

- Wir fügen die Schaltfläche dem eigentlichen Ribbon hinzu, als in einer der Gruppen in den **Tab**-Elementen.
- Oder wir legen die Ribbon-Schaltfläche in der Quick Access Toolbar an, das ist die Leiste mit den kleinen Befehlsschaltflächen über dem eigentlichen Ribbon.

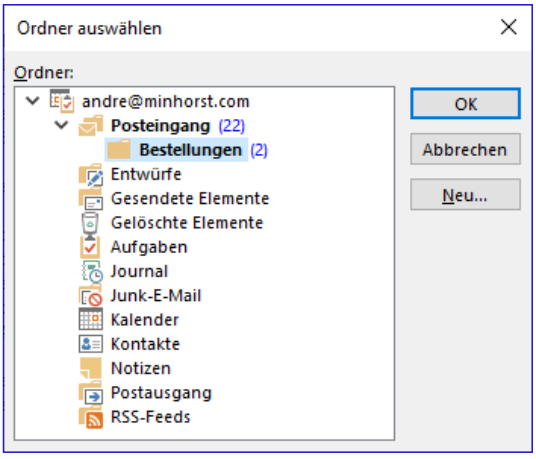

Bild 2: Auswahl eines Ordners

Der Unterschied liegt im Wesentlichen in der Art, wie wir die Befehle per Tastenkombination aufrufen können. Für das eigentlichen Ribbon benötigen wir eine Kombination aus mehreren mit der **Alt**-Taste verwendeten Buchstaben. Wenn Sie die **Alt**-Taste betätigen, werden die direkt verfügbaren Elemente des Ribbons ja mit Buchstaben versehen, über die Sie diese

Danach wollen wir wieder die Schaltflächen hinzufügen, mit denen die einzelnen Prozeduren zum Zuweisen der aktuell markierten Mails aus dem Posteingang zu den verschiedenen Ordnern aufgerufen werden sollen. Diese finden wir wieder, indem wir im Auswahlfeld über der linken Liste den Eintrag **Makros** auswählen. Danach erscheinen die öffentlichen deklarierten Prozeduren des VBA-Projekts in der Liste und wir können diese durch Markieren und Betätigen der Schaltfläche **Hinzufügen >>** zur gewünschten Gruppe im Ribbon hinzufügen. Danach erscheinen die Befehle wie in Bild 8 an der gewünschten Stelle.

Damit Sie dort auch die von uns angegebenen Buchstaben vorfinden, müssen Sie die Werte des Attributs imageMso entsprechend einstellen. Die Symbole zur Darstellung der Buchstaben finden wir nicht im Dialog zur Auswahl der Icons, also gehen wir wieder wie folgt vor:

- Exportieren der Definition mit der Schaltfläche **Importieren/Exportieren|Alle Anpassungen exportieren**
- Öffnen der Definition per Text- oder XML-Editor
- Hinzufügen des Attributs **imageMso** mit den jeweiligen Werten, also beispielsweise dem Buchstaben **B** (**imageMso="B"**)
- Speichern der geänderten Datei
- Importieren der Definition mit der Schaltfläche **Importieren/Exportieren|Anpassungsdatei importieren**
- Neustart von Outlook

Tastenkürzel definieren

Bei oben genanntem Vorgang können Sie auch gleich die Tastenkürzel mit dem Attribut **keytip** festlegen. Beachten Sie, dass Sie Großbuchstaben angeben müssen, damit es funktioniert.

Das Ergebnis sieht im XML-Editor wie in Bild 9 aus.

Solution1 - XMLFile1.xml*

XMLFile1.xml*

```
<mso:cmd app="olkexplorer" dt="0" slr="0" />
<mso:customUI xmlns:x1="http://schemas.microsoft.com/office/2009/07/customui/macro" xmlns:mso="http://
schemas.microsoft.com/office/2009/07/customui">
  <mso:ribbon>
    <mso:qat>
      ...
    </mso:qat>
    <mso:tabs>
      <mso:tab id="mso_c1.5696E3A" label="Zielordner" insertBeforeQ="mso:TabCalendarTableView" keytip="ZZ">
        <mso:group id="mso_c2.5696E3A" label="Zielordner" autoScale="true">
          <mso:button idQ="x1:Projekt1.MoveToFolder_Bestellungen_1" visible="true" label="Zu Bestellungen"
           imageMso="B" onAction="Projekt1.MoveToFolder_Bestellungen" keytip="B" />
          <mso:button idQ="x1:Projekt1.MoveToFolder_Themenvorschlaege_1" visible="true" label="Zu
           Themenvorschlaege" imageMso="T" onAction="Projekt1.MoveToFolder_Themenvorschlaege" keytip="T"/>
          <mso:button idQ="x1:Projekt1.MoveToFolder_Rechnungen_1" visible="true" label="Zu Rechnungen"
           imageMso="R" onAction="Projekt1.MoveToFolder_Rechnungen" keytip="R"/>
          <mso:button idQ="x1:Projekt1.MoveToFolder_FirstInkasso_1" visible="true" label="Zu FirstInkasso"
           imageMso="F" onAction="Projekt1.MoveToFolder_FirstInkasso" keytip="F"/>
        </mso:group>
      </mso:tab>
    </mso:tabs>
  </mso:ribbon>
</mso:customUI>
```

Bild 9: Erweiterung des Ribbons im XML-Editor

Kunde zu einer E-Mail öffnen

Viele Access-Benutzer dürften, wenn Sie schon das Office-Paket auf dem Rechner haben, auch Outlook nutzen – zumindest, um E-Mails zum empfangen, zu senden und zu bearbeiten. Vielen fehlt dabei die Möglichkeit einer stärkeren Interaktion zwischen Access und Outlook, beispielsweise bei Verwendung einer Kundenverwaltung. Eine pfiffige Möglichkeit wäre es, beim Anzeigen einer E-Mail von einem bestimmten Kunden direkt den Kundendatensatz in der Kundenverwaltung zu öffnen. Dies sollte dann auch noch auf möglichst einfache Art und Weise geschehen, beispielsweise per Tastenkombination. Wie das gelingt, zeigt der vorliegende Beitrag.

In einem anderen Beitrag mit dem Titel **Outlook-Mails per Tastenkombination verschieben** (**www.access-im-unternehmen.de/1290**) haben wir bereits gezeigt, wie Sie Aktionen für die aktuell markierte E-Mail per Tastenkombination ausführen können.

Im vorliegenden Beitrag wollen wir die Verbindung zwischen der dort beschriebenen reinen Outlook-Technik und Access herstellen. Dazu wollen wir, wenn der Benutzer eine bestimmte Tastenkombination für die aktuell angezeigte E-Mail betätigt, die folgenden Schritte ausführen:

- Die E-Mail-Adresse des Absenders dieser E-Mail ermitteln,
- prüfen, ob die Kundendatenbank bereits geöffnet ist und diese gegebenenfalls öffnen, wobei dann die E-Mail-Adresse übergeben werden soll und
- den Kunden mit der betreffenden E-Mail-Adresse im Kunden-Formular anzeigen.

Ob die Kunden-Datenbank bereits geöffnet ist oder nicht bringt noch eine wichtige Unterscheidung mit sich: Wenn sie noch nicht geöffnet ist, haben wir es etwas einfacher. Wir brauchen die Datenbank dann nämlich einfach nur zu öffnen, was relativ einfach ist, und dabei die entsprechende E-Mail-Adresse zu übergeben. Wenn die Access-Anwendung bereits geöffnet ist, wird es etwas komplizierter, denn dann müssen wir per VBA zunächst die Access-Anwendung referenzieren und diese dann so fernsteuern, dass sie den gewünschten Kunden anzeigt.

Wir wollen jedoch mit dem Aufruf beginnen.

E-Mail auswählen und den Prozess starten

Die Grundlagen zu diesem Schritt haben wir ausführlich in dem eingangs erwähnten Beitrag beschrieben. Dabei haben wir eine VBA-Prozedur erstellt, welche die aktuelle E-Mail ermittelt und dann etwas damit erledigt.

Einige der dort verwendeten Techniken übernehmen wir für die nachfolgend vorgestellte Prozedur namens **ShowCustomerInAccess**, die Sie in Listing 1 finden (die nachfolgend beschriebenen Routinen geben Sie in einem neuen Standardmodul des VBA-Projekts von Outlook ein, das Sie von Outlook aus mit der Tastenkombination **Alt + F11** öffnen). Diese Prozedur schreibt den Pfad zur Kundenverwaltungs-Datenbank in die Variable **strDatabase** und referenziert den aktuell im Outlook-Explorer-Bereich angezeigten Ordner mit der Variablen **objCurrentFolder**.

Heißt dieser Ordner **Posteingang**, wird der Rest der Prozedur ausgeführt. Wenn Sie nicht die deutschsprachige Outlook-Version verwenden, heißt der Ordner möglicherweise anders, in diesem Fall müssen Sie den Code entsprechend anpassen. Die Prozedur erfasst mit der

Variablen **objSelection** die aktuelle Auswahl in diesem Ordner. Wenn die mit **Count** ermittelte Anzahl der markierten Elemente nicht **1** lautet, erscheint eine entsprechende Meldung und die Prozedur wird beendet.

Anderenfalls prüft die Prozedur, ob es sich um ein Element des Typs **MailItem** handelt. Ist das der Fall, weist die Prozedur das markierte Objekt der Variablen **objMailItem** zu. Danach liest es die E-Mail-Adresse des Absenders aus der Eigenschaft **SenderEmailAddress** aus und schreibt sie in die Variable **strMail**.

Anschließend versucht die Prozedur, mit der Funktion **GetDatabase** auf eine laufende Instanz der Datenbank aus **strDatabase** zugreifen. Liefert diese kein Objekt zurück, bleibt **objAccess** leer. Die Prozedur durchläuft dann den **If**-Teil der Bedingung.

Hier erstellt sie mit **OpenDatabase** einen Verweis auf das **Database**-Objekt der Datenbank und referenziert dieses mit der Variablen **db**. Damit ruft sie die Funktion **CustomerExists** auf, die prüft, ob die Datenbank einen Kunden mit der angegebenen E-Mail-Adresse enthält.

```
Public Sub ShowCustomerInAccess()
    Dim objMailItem As MailItem
    Dim objExplorer As Explorer
    Dim objSelection As Selection
    Dim objCurrentFolder As Folder
    Dim objNamespace As NameSpace
    Dim objAccess As Access.Application
    Dim strMail As String
    Dim strDatabase As String
    Dim db As DAO.Database
    strDatabase = "C:\...\Kundenverwaltung.accdb"
    Set objExplorer = Application.ActiveExplorer
    Set objCurrentFolder = objExplorer.CurrentFolder
    If objCurrentFolder.Name = "Posteingang" Then
        Set objSelection = objExplorer.Selection
        Set objNamespace = Application.GetNamespace("MAPI")
        If Not objSelection.count = 1 Then
            MsgBox ("Bitte selektieren Sie genau eine E-Mail.")
            Exit Sub
        End If
        Select Case TypeName(objSelection.Item(1))
            Case "MailItem"
                Set objMailItem = objSelection.Item(1)
                strMail = objMailItem.SenderEmailAddress
                Set objAccess = GetDatabase(strDatabase)
                If objAccess Is Nothing Then
                    Set db = OpenDatabase(strDatabase)
                    If CustomerExists(db, strMail) Then
                        OpenDatabaseInAccess strDatabase, strMail
                    Else
                        MsgBox "Es existiert kein Kunde mit der E-Mail-Adresse '" _
                            & strMail & "'"
                    End If
                    Set db = Nothing
                Else
                    ShowCustomer objAccess, strMail
                End If
        End Select
    End If
End Sub
```

Listing 1: Prozedur zum Anzeigen eines Kunden in Access

Falls ja, ruft die Prozedur eine weitere Routine namens **OpenDatabaseInAccess** auf, welche die Kundenverwaltung öffnen und die E-Mail-Adresse des gesuchten Kunden als Parameter übergeben soll.

Liefert **CustomerExists** den Wert **False** zurück, zeigt die Prozedur eine Meldung an, dass kein Kunde mit der angegebenen E-Mail-Adresse gefunden werden konnte.

```
Public Function GetDatabase(strDatabase As String) As Access.Application
    Dim objAccess As Access.Application
    If IsDatabaseOpen(strDatabase) Then
        Set objAccess = GetObject(strDatabase)
    End If
    Set GetDatabase = objAccess
End Function
```

Listing 2: Funktion zum Holen eines **Access.Application**-Objekts

Es gibt noch den Fall, dass die Funktion **GetDatabase** einen Verweis auf eine Access-Anwendung zurückgeliefert hat. Diese wird im **Else**-Zweig der entsprechenden Bedingung behandelt. Dieser Teil ruft die Prozedur **ShowCustomer** auf und übergibt dieser einen Verweis auf die gefundene Access-Instanz und die E-Mail-Adresse des gesuchten Kunden.

```
Public Function IsDatabaseOpen(strPath As String) As Boolean
    Dim strLACCDB As String
    Dim bolIsDatabaseOpen As Boolean
    strLACCDB = Replace(strPath, ".accdb", ".laccdb")
    If Not Len(Dir(strLACCDB)) = 0 Then
        On Error Resume Next
        Kill strLACCDB
        If Not Err.Number = 0 Then
            bolIsDatabaseOpen = True
        End If
        On Error GoTo 0
    End If
    IsDatabaseOpen = bolIsDatabaseOpen
End Function
```

Listing 3: Funktion zum Prüfen, ob eine Datenbank geöffnet ist

Die hier aufgerufenen Funktionen und Prozeduren beschreiben wir in den folgenden Abschnitten.

Datenbank holen mit GetDatabase

Die erste Funktion, die wir benötigen, heißt **GetDatabase**. Sie erwartet den Pfad der zu holenden Datenbank als Parameter und liefert eine Objektvariable des Typs **Access.Application** zurück (siehe Listing 2).

Die Funktion ruft eine weitere Funktion namens **IsDatabaseOpen** auf, der ebenfalls der Pfad zu der betroffenen Datenbank übergeben wird. Diese Funktion, die wir im Anschluss beschreiben, prüft, ob die mit **strDatabase** angegebene Datenbank geöffnet ist. Ist das der Fall, verwendet die Funktoin **GetDatabase** die VBA-Funktion **GetObject** mit Angabe des Datenbankpfades, um einen Verweis auf das entsprechende **Access.Application**-Objekt zu holen. Der Inhalt der Variablen **objAccess** wird dann als Funktionswert der Funktion **GetDatabase** an die aufrufende Funktion zurückgegeben. Dabei kann **objAccess** an dieser Stelle auch leer sein, also den Wert **Nothing** enthalten.

Auf geöffnete Datenbank prüfen mit IsDatabaseOpen

Die Funktion **IsDatabaseOpen** erwartet mit dem Parameter **strPath** die Angabe der zu untersuchenden Datenbankdatei (siehe Listing 3). Der Ansatz ist, die beim Öffnen einer Datenbankdatei von Access erstellte Datei mit der Endung **.laccdb** zu untersuchen. Diese wird immer im gleichen Verzeichnis wie die **.accdb**-Datei erstellt. Außerdem ist diese Datei, wenn die Datenbank noch geöffnet ist, schreibgeschützt und kann nicht gelöscht werden. Manchmal kommt es vor, dass die Access-Datenbank geschlossen wird und die **.laccdb**-Datei noch vorhanden ist. Dann kann diese aber problemlos gelöscht werden, was wir in der Funktion ausnutzen.

Outlook-Folder in Access anzeigen

Es gibt sehr viele Möglichkeiten für Interaktion zwischen Outlook und Access. Sie können Termine, E-Mails, Kontakte oder Aufgaben zwischen den beiden Anwendungen abfragen, synchronisieren, erstellen oder bearbeiten. Sehr stiefmütterlich wurde bisher allerdings das Thema der Anzeige von Outlook-Elementen in Access behandelt. Zum Glück brachte mich neulich ein Leser auf die Idee, das Thema noch einmal aufzugreifen – und lieferte mir auf mein Zögern hin direkt noch ein Beispiel, wie die Integration funktioniert. Wir haben uns dies einmal genau angesehen. Das Ergebnis und die resultierenden Möglichkeiten finden Sie im vorliegenden Beitrag.

Bereits vor einigen Jahren habe ich aus Neugier einmal versucht, eines der interessant aussehenden Elemente des Dialogs zum Einfügen von ActiveX-Steuerelementen in Formulare zu nutzen (siehe Bild 1).

Allerdings bin ich damals nicht besonders weit gekommen – vermutlich war die Dokumentation dieser Elemente damals noch lückenhafter als heute.

Dank unseres Lesers Oliver Specht habe ich das Thema allerdings nochmal angesehen und dank seiner Vorarbeit lief es nun direkt wesentlich besser.

E-Mails im Access-Formular anzeigen

Die erste Idee war, den Outlook-Explorer mit den E-Mails einmal in einem Access-Formular anzuzeigen.

Dazu gehen Sie wie folgt vor:

- Legen Sie ein neues, leeres Formular an.
- Öffnen Sie mit dem Ribbon-Eintrag **Entwurf|Steuerelemente|ActiveX-Steuerelemente** den Dialog **ActiveX-Steuerelemente einfügen** und scrollen Sie zu den Einträgen, die mit **Outlook** beginnen.

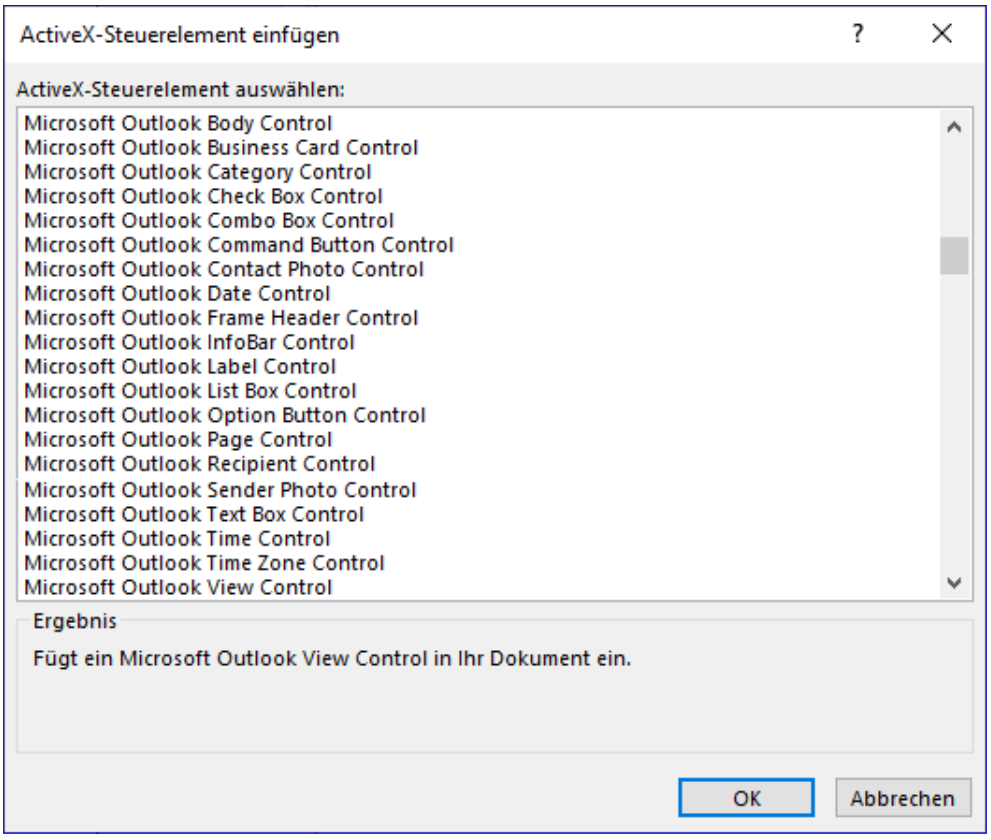

Bild 1: Als ActiveX-Steuerelemente einfügbare Outlook-Elemente

- Fügen Sie das ActiveX-Steuerelement **Microsoft Outlook View Control** in das Formular ein und passen Sie seine Größe nach Wunsch an.

Danach sieht das Formular wie in Bild 2 aus.

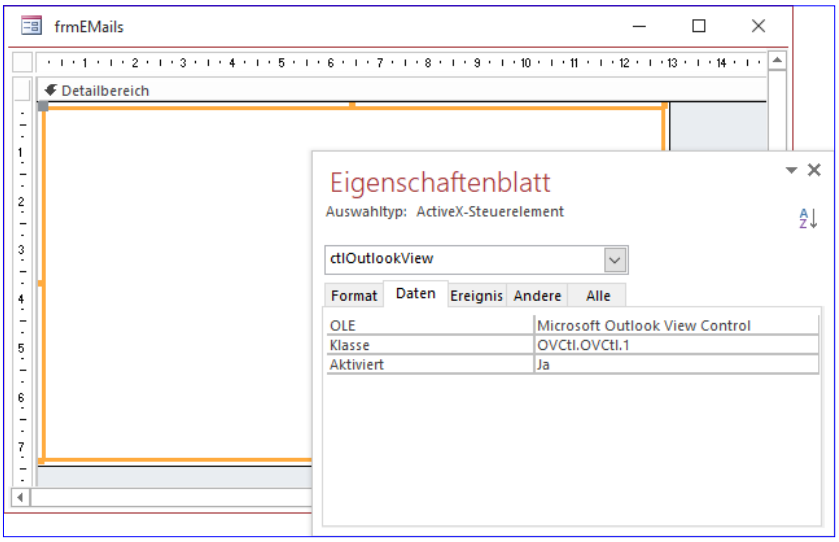

Bild 2: Das **Microsoft Outlook View Control** in einem Access-Formular

Wenn wir nun in die Formularansicht wechseln, erhalten wir die Meldung **In diesem Steuerelement befindet sich kein Objekt**.

Die gleiche Meldung erscheint auch nochmal beim Wechsel zurück zur Entwurfsansicht, danach aber nicht mehr.

Wenn wir das Steuerelement nochmal löschen und erneut einfügen, können Sie einen Test machen: Klicken Sie, bevor Sie in die Formularansicht wechseln, einmal in der Entwurfsansicht doppelt auf das hinzugefügte Steuerelement.

Dieses zeigt dann wie in Bild 3 die Liste der E-Mails an – und zwar die aktuellen, in Outlook angezeigten Einträge. Nach einem Wechsel in die Formularansicht erhalten wir allerdings das gleiche Ergebnis wie zuvor.

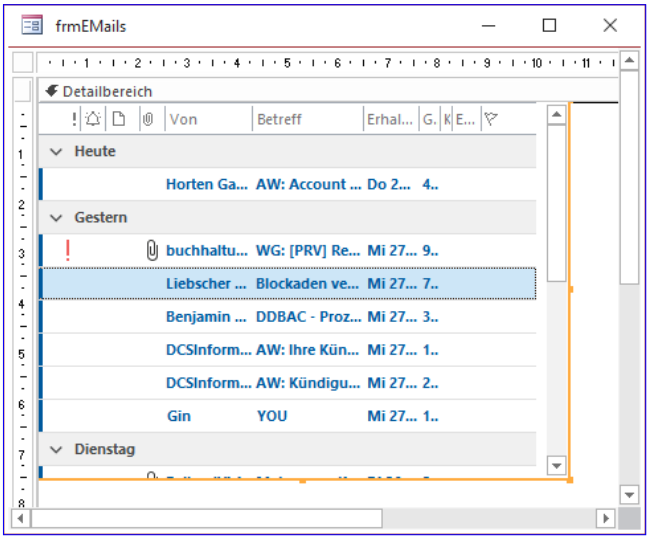

Bild 3: Anzeige von Outlook-Daten im Entwurf

Outlook-View in Frame

Die Lösung ist das Einbetten des **Outlook View Control**-Steuerelements in ein weiteres Steuerelement, nämlich das **Frame**-Steuerelement der **Forms**-Bibliothek.

Der Vorgang ist nicht besonders intuitiv, daher hier die ausführliche Beschreibung:

- Fügen Sie einem neuen, leeren Formular aus dem Dialog **ActiveX-Steuerelemente einfügen** das Element **Microsoft Forms 2.0 Frame** hinzu (siehe Bild 4).

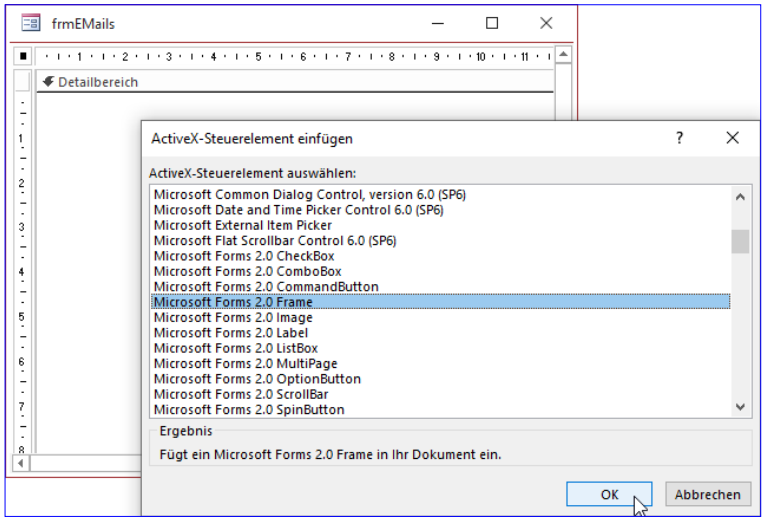

Bild 4: Hinzufügen eines Frames

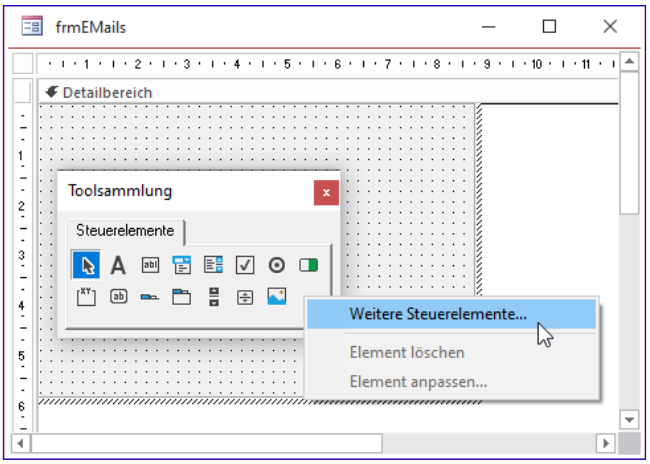

Bild 5: Freischalten weiterer Steuerelemente

- Danach klicken Sie doppelt auf das frisch eingefügte **Frame**-Steuerelement.

- Sollte die Toolbox nicht angezeigt werden, klicken Sie mit der rechten Maustaste in das Steuerelement und aktivieren Sie den Eintrag **Toolsammlung...**.

- Klicken Sie mit der rechten Maustaste in den Bereich mit den Steuerelementen und wählen Sie den Kontextmenü-Eintrag **Weitere Steuerelemente** aus (siehe Bild 5).

- Selektieren Sie im nun erscheinenden Dialog **Weitere Steuerelemente** den Eintrag **Microsoft Outlook View Control** (siehe Bild 6).

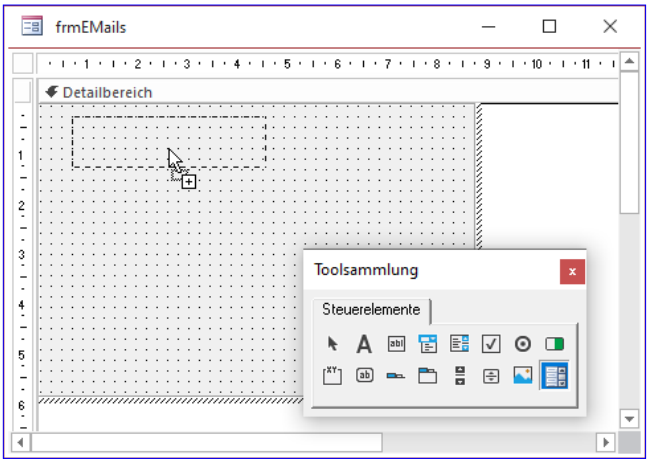

Bild 7: Hinzufügen des **ViewCtl**-Steuerelements per Drag and Drop

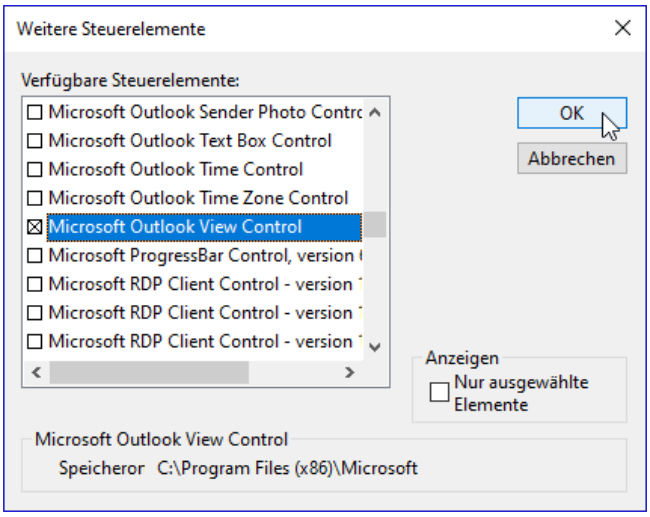

Bild 6: Auswahl des **Outlook View Control**-Steuerelements

- Dieses finden Sie danach in der Toolbox vor. Ziehen Sie es per Drag and Drop in das **Frame**-Objekt (siehe Bild 7).

- Schließlich ziehen Sie es auf die gewünschte Größe – so, dass es die gesamte Fläche des **Frame**-Steuerelements ausfüllt.

Wechseln Sie nun in die Formularansicht, haben wir das erste Zwischenziel erreicht: Das Formular zeigt die Liste der E-Mails an (siehe Bild 8). Sie können die angezeigten

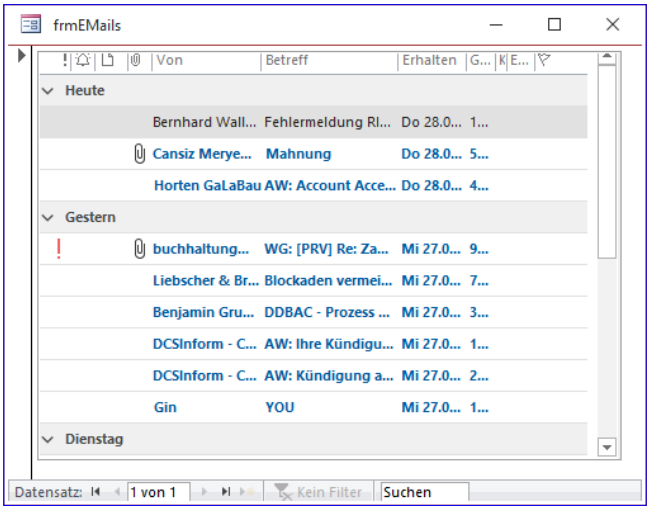

Bild 8: Anzeige der E-Mails im Formular

Elemente genau wie in Outlook nutzen, also die Spalten anpassen, E-Mails löschen oder diese per Doppelklick öffnen. Nach dem Öffnen einer E-Mail wird diese in dem von Outlook bekannten Fenster angezeigt.

View-Steuerelement per VBA referenzieren

Nun wollen wir herausfinden, wie wir per VBA auf das **Microsoft Outlook View Control** zugreifen können. Nun befindet sich das Steuerelement in einem weiteren Steuerelement des Typs **Frame**. Wir stellen einmal nach, wie wir herausfinden, wie das Steuerelement zu referenzieren ist.

Als Erstes wollen wir das **Frame**-Steuerelement per VBA-Variable referenzieren. Dazu müssen wir herausfinden, welchen Typ das **Frame**-Steuerelement hat.

Dazu öffnen wir das Formular, das aktuell nur das **Frame**-Steuerelement mit dem View-Steuerelement enthält, in der Formularansicht. Im Direktbereich können wir nun mithilfe der **TypeName**-Funktion den Typ des **Frame**-Steuerelements herausfinden. Dazu setzen Sie dort den folgenden Befehl ab:

```
Debug.Print TypeName(Screen.ActiveForm.Controls(0))
CustomControl
```

Wir referenzieren mit **Screen.ActiveForm** zunächst das aktuelle Formulare. Dieses könnten wir auch mit **Forms!<Formularname>** referenzieren, aber **Screen.ActiveForm** ist einfacher, weil wir den Formularnamen dazu nicht kennen müssen.

Da das Formular nur ein Steuerelement enthält, können wir dieses mit **Controls(0)** ansprechen. Das Ergebnis lautet **CustomControl**. Das ist nicht das gewünschte Ergebnis.

Der Grund ist: ActiveX-Steuerelemente spricht man immer noch zusätzlich über die **Object**-Eigenschaft an.

Das ergibt dann:

```
Debug.Print TypeName(Screen.ActiveForm.Controls(0).Object)
Frame
```

Damit ist bestätigt, dass wir eine Objektvariable für das **Frame**-Objekt beispielsweise in der Prozedur, die durch das Ereignis **Beim Laden** des Formulars ausgelöst wird, mit dem Datentyp **Frame** deklarieren können:

```
Private Sub Form_Load()
    Dim objFrame As Frame
    Set objFrame = Me!ctlFrame.Object
End Sub
```

Warum wollen wir dieses Steuerelement überhaupt per Objektvariable referenzieren? Weil wir so über IntelliSense auf seine Eigenschaften zugreifen können!

In einer weiteren Anweisung der obigen Prozedur können wir nun den Typ des einzigen Elements der **Controls**-Auflistung des **Frame**-Steuerelements ermitteln:

```
Debug.Print TypeName(objFrame.Controls(0).Object)
```

Dies liefert das Ergebnis **IViewCtl**, was eine Schnittstelle beschreibt. Wir verwenden allerdings **ViewCtl** und weisen das Steuerelement wie folgt zu:

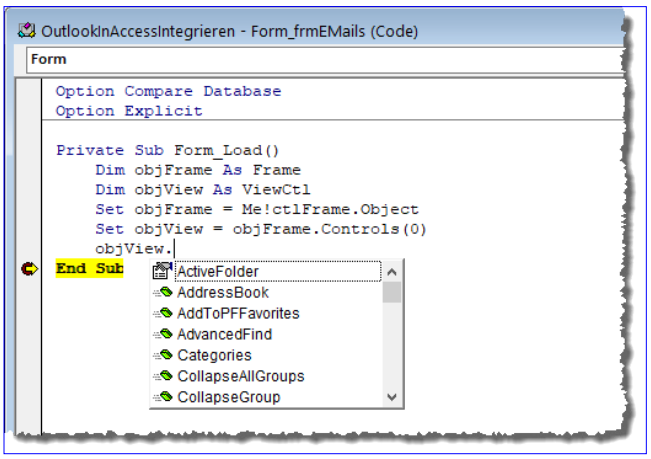

Bild 9: Zugriff per IntelliSense auf die Eigenschaften und Methoden des **View**-Steuerelements

```
Dim objFrame As Frame
Dim objView As ViewCtl
Set objFrame = Me!ctlFrame.Object
Set objView = objFrame.Controls(0)
```

Damit können wir dann per IntelliSense die Eigenschaften und Methoden dieses Steuerelements ansteuern (siehe Bild 9).

Eigenschaften und Methoden des View-Steuerelements

Damit können wir uns nun die Eigenschaften und Methoden ansehen und unter anderem die Ansicht des Steuerelements damit steuern. Einige davon können wir uns aber auch bereits über die Benutzeroberfläche ansehen. Das wollen wir hier noch vorziehen. Dazu zeigen Sie das Formular in der Entwurfsansicht an.

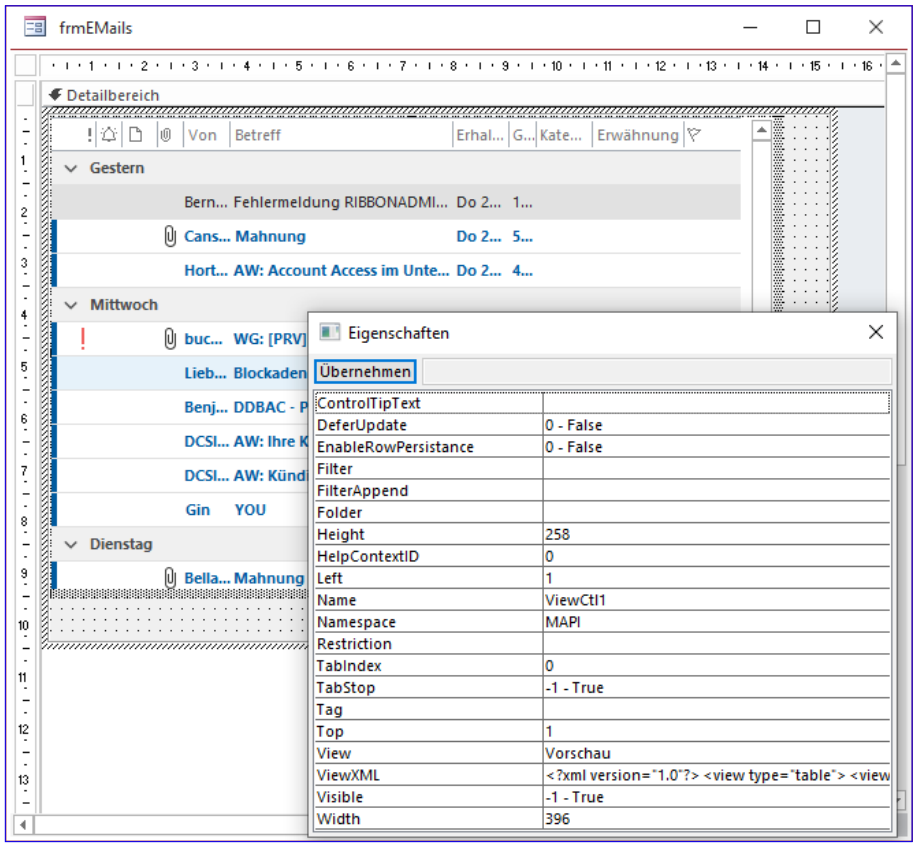

Bild 10: Eigenschaften des **View**-Steuerelements

Klicken Sie doppelt auf das **Frame**-Steuerelement, sodass das **View**-Steuerelement erscheint. Wählen Sie dann aus dem Kontextmenü des **View**-Steuerelements den Eintrag **Eigenschaften** aus. Dies liefert das Eigenschaften-Fenster aus Bild 10.

Eine interessante Eigenschaft lautet **ViewXML**. Diese liefert offensichtlich den Aufbau der Ansicht im XML-Format.

Wollen Sie dieses XML-Dokument betrachten, können Sie es einfach durch Hinzufügen des folgenden Befehls zur Prozedur **Form_Load** im Direktbereich des VBA-Editors ausgeben:

```
Private Sub Form_Load()
    ...
    Debug.Print objView.ViewXML
End Sub
```

In diesem XML-Dokument finden Sie einige Attribute, die das Layout der E-Mail-Liste beschreiben (siehe Listing 1). Am Beispiel der Sortierung können Sie prüfen, dass die XML-Definition jeweils an die aktuellen Einstellungen im **View**-Steuerelement angepasst wird. Wenn Sie beispielsweise eine andere Spalte als Sortierkriterium festlegen, schlägt sich dies direkt im Unterelement **orderby** nieder. Im folgenden Beispiel ist die Sortierreihenfolge auf die Spalte **Erhalten** festgelegt:

```
<orderby>
    <order>
        <heading>Erhalten</heading>
        <prop>urn:schemas:httpmail:datereceived</prop>
        <type>datetime</type>
        <sort>desc</sort>
    </order>
</orderby>
```

Ändern wir die Sortierung auf die Spalte **Von** und fragen die XML-Definition erneut ab, erhalten wir für das Element **orderby** den folgenden Code:

```
<orderby>
  <order>
    <heading>Von</heading>
    <prop>urn:schemas:httpμ
       mail:fromname</prop>
    <type>string</type>
    <sort>asc</sort>
  </order>
</orderby>
```

Funktion für den Zugriff auf das View-Steuerelement per VBA

In den folgenden Abschnitten werden wir einige Dinge zur Steuerung des View-Steuerelements per VBA über den Direktbereich des VBA-Editors ausprobieren. Da es wenig Spaß macht, immer den kompletten Ausdruck zum Referenzieren dieses Steuerelements im aktuell geöffneten Formular einzugeben, bauen wir uns zu diesem Zweck eine kleine Hilfsfunktion in einem Standardmodul.

Diese Funktion heißt **OutlookView** und Sie finden diese in Listing 2. Damit können Sie nun einfach mit **OutlookView** auf das

```
<?xml version="1.0"?>
<view type="table">
    <viewname>Vorschau</viewname>
    <viewstyle>font-family:Segoe UI;...;table-layout:fixed;width:100%</viewstyle>
    <viewtime>0</viewtime>
    <linecolor>8421504</linecolor>
    <linestyle>3</linestyle>
    <previewlines>0</previewlines>
    <previewlineschangenum>2</previewlineschangenum>
    <autopreview>1</autopreview>
    <previewunreadonly>1</previewunreadonly>
    <ensuredcategoriesfield>1</ensuredcategoriesfield>
    <collapsestate/>
    <rowstyle>background-color:White;color:Black</rowstyle>
    <headerstyle>background-color:#D3D3D3</headerstyle>
    <previewstyle/>
    <arrangement>
        <autogroup>1</autogroup>
        <enablexfc>1</enablexfc>
        <collapseclient>01000000</collapseclient>
        <collapseconv/>
        <upgradetoconvchangenum>2</upgradetoconvchangenum>
    </arrangement>
    ...
    <column>
        <heading>Von</heading>
        <prop>urn:schemas:httpmail:fromname</prop>
        <type>string</type>
        <width>38</width>
        <style>text-align:left;padding-left:3px</style>
        <editable>0</editable>
        <displayformat>1</displayformat>
    </column>
    ...
    <orderby>
        <order>
            <heading>Erhalten</heading>
            <prop>urn:schemas:httpmail:datereceived</prop>
            <type>datetime</type>
            <sort>desc</sort>
        </order>
    </orderby>
    <groupbydefault>2</groupbydefault>
    <previewpane>
        <markasread>0</markasread>
    </previewpane>
</view>
```

Listing 1: XML-Definition der aktuellen Ansicht

Steuerelement und seine Eigenschaften und Methoden zugreifen.

```
Public Function OutlookView() As ViewCtl
    Set OutlookView = Forms!frmEMails!ctlFrame.Object.Controls(0)
End Function
```

Listing 2: VBA-Prozedur für den Zugriff auf das **View**-Steuerelement

Verschiedene Ansichten einstellen mit Folder und View

Die XML-Definition liefert eine Menge Informationen, aber wir können dieser nicht entnehmen, welcher Ordner-Inhalt gerade angezeigt wird. Dies lässt sich allerdings mit der **Folder**-Eigenschaft realisieren.

Diese liefert, wenn Sie den aktuellen Wert ausgeben, für die aktuelle Ansicht allerdings keinen Wert – wie wir mit unserer Funktion **OutlookView** im Direktbereich des VBA-Editors belegen können:

```
? OutlookView.Folder
<leere Zeichenkette>
```

Eine zweite wichtige Eigenschaft lautet **View**. Diese liefert direkt nach dem Anzeigen bereits einen Wert:

```
? OutlookView.View
Vorschau
```

Interessanterweise erscheinen die Werte offensichtlich in der jeweiligen Landessprache. Woher aber bekommen wir die möglichen Werte für diese Eigenschaften? Der Objektkatalog liefert jedenfalls kein Element mit dem Text **Vorschau**. Und welche Werte können wir für die Eigenschaft **Folder** verwenden?

Folder-Werte herausfinden

Die **Folder**-Werte entsprechen schlicht den Ordnerbezeichnungen aus Outlook. Um alle möglichen Einträge zu sehen, brauchen Sie nur die Ansicht des Ordnerbereichs von Outlook auf **Ordner** umzustellen.

Das erledigen Sie mit einem Klick auf die Schaltfläche mit den drei Punkten unten im Ordnerbereich und anschließende Auswahl des Eintrags **Ordner** (siehe Bild 11).

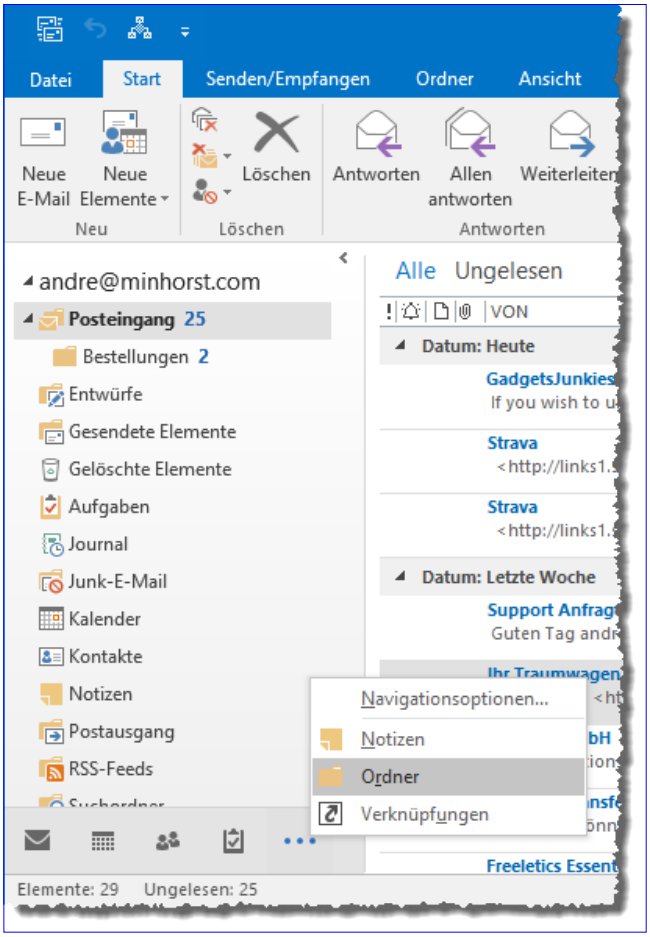

Bild 11: Ordner in Outlook

Danach brauchen Sie die Eigenschaft **Folder** nur noch auf den Namen des Ordners einzustellen, also beispielsweise wie folgt:

```
OutlookView.Folder = "Kalender"
OutlookView.Folder = "Aufgaben"
OutlookView.Folder = "Gesendete Elemente"
```

Wenn Sie dem Posteingang noch weitere Ordner hinzugefügt haben und ihren Inhalt anzeigen wollen, geben Sie

TreeView für Outlook-Ordner

Im Beitrag »Outlook-Folder in Access anzeigen« liefern wir die Grundlagen zur Anzeige von Outlook-Ordnern in Access-Formularen. Dabei haben wir die einzelnen Outlook-Ordner in einem einfachen Listenfeld zur Auswahl angeboten. Im Beitrag »E-Mails verwalten mit dem Outlook View Control« wollen wir dies ein wenig professioneller gestalten und die Outlook-Ordner in einem TreeView-Steuerelement anzeigen. Wie das gelingt, erfahren Sie im vorliegenden Beitrag.

Das **TreeView**-Steuerelement, in dem wir die Outlook-Ordner in der gleichen hierarchischen Anordnung anzeigen wollen wie im entsprechenden Bereich von Outlook, soll die Auswahl eines der Ordner ermöglichen. Der Inhalt dieses Ordners soll in einem **Outlook View Control** erscheinen. Unter Outlook sieht die hierarchische Anzeige wie in Bild 1 aus.

Für den Anfang wollen wir uns dabei bezüglich der Icons damit begnügen, für jeden Ordner das gleiche Icon anzuzeigen, nämlich ein einfaches Ordner-Icon.

Außerdem wollen wir nur die Ordner anzeigen, die einen bestimmten Typ von Elementen enthalten. In diesem Fall sind das die Ordner, die typischerweise E-Mails enthalten. Wie Sie diese Ordner von den anderen unterschieden, erläutern wir weiter unten.

Welche Schritte sind für die Umsetzung unseres Vorhabens nötig? Zunächst benötigen wir einige Steuerelemente:

- ein **TreeView**-Steuerelement und
- ein **ImageList**-Steuerelement zum Speichern der zu verwendenden Icons.

Außerdem benötigen wir VBA-Code, der beim Laden des Formulars ausgelöst wird und der die betreffenden Ordner aus Outlook einliest und entsprechende Einträge im **TreeView**-Steuerelement anlegt.

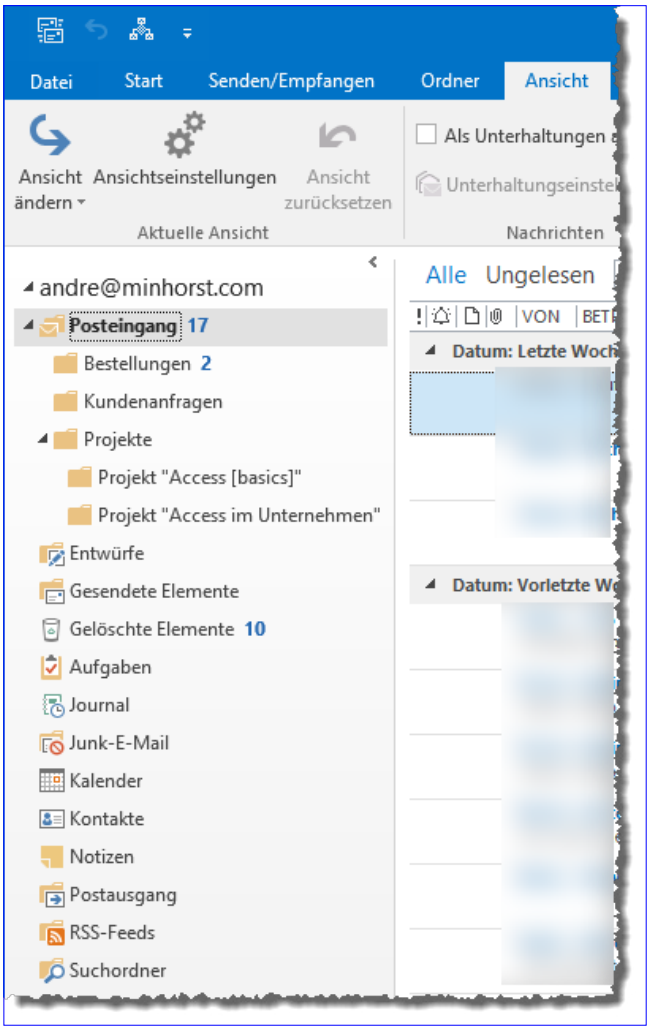

Bild 1: Anzeige der Ordner in Outlook

Steuerelemente hinzufügen

Die beiden benötigten Steuerelemente fügen Sie der Entwurfsansicht eines neuen Formulars namens **frmE-**

MailDetails hinzu, indem Sie über den Ribbon-Eintrag **EntwurfISteuerelementeIActiveX-Steuerelemente** den Dialog **ActiveX-Steuerelement einfügen** öffnen. Hier wählen Sie als Erstes das Steuerelement **Microsoft TreeView Control, version 6.0** aus und klicken **OK** (siehe Bild 2).

Danach führen Sie den gleichen Vorgang für das Steuerelement **Microsoft ImageList Control, version 6.0** aus.

Legen Sie für dieses Steuerelement den Namen **ctlImageList** fest und für das **TreeView**-Steuerelement den Namen **ctlTreeView**. Ziehen Sie das **TreeView**-Steuerelement so auf, dass es sich am linken Rand des Formulars befindet (siehe Bild 3).

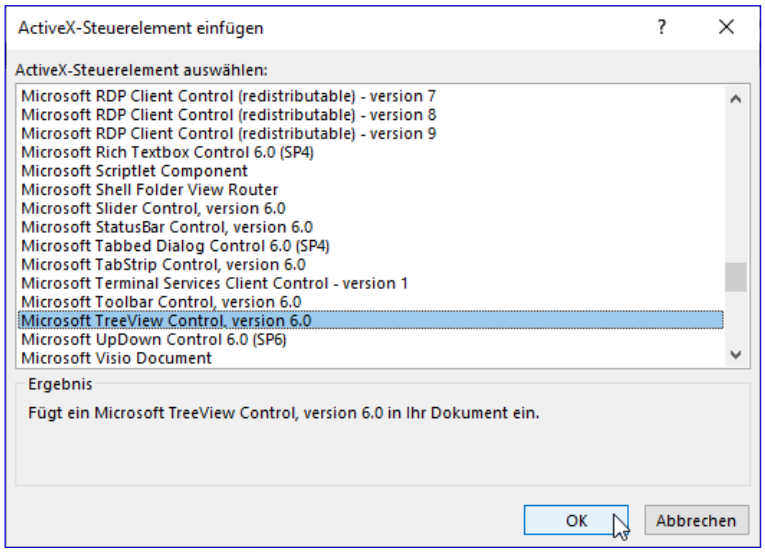

Bild 2: Hinzufügen der ActiveX-Steuerelemente

Icons zu ImageList-Steuerelement hinzufügen

Danach fügen wir das zunächst zu verwendende Icon zum **ImageList**-Steuerelement hinzu. Dazu klicken Sie doppelt auf das Steuerelement und erhalten den Dialog **Eigenschaften von ImageListCtrl**. Hier stellen Sie die Bildgröße auf der Registerseite **General** auf **16 x 16** ein (siehe Bild 4).

Danach wechseln Sie auf die Registerseite **Images** und klicken dort auf die Schaltfläche **Insert Picture...**, was einen **Dateiauswahl**-Dialog öffnet.

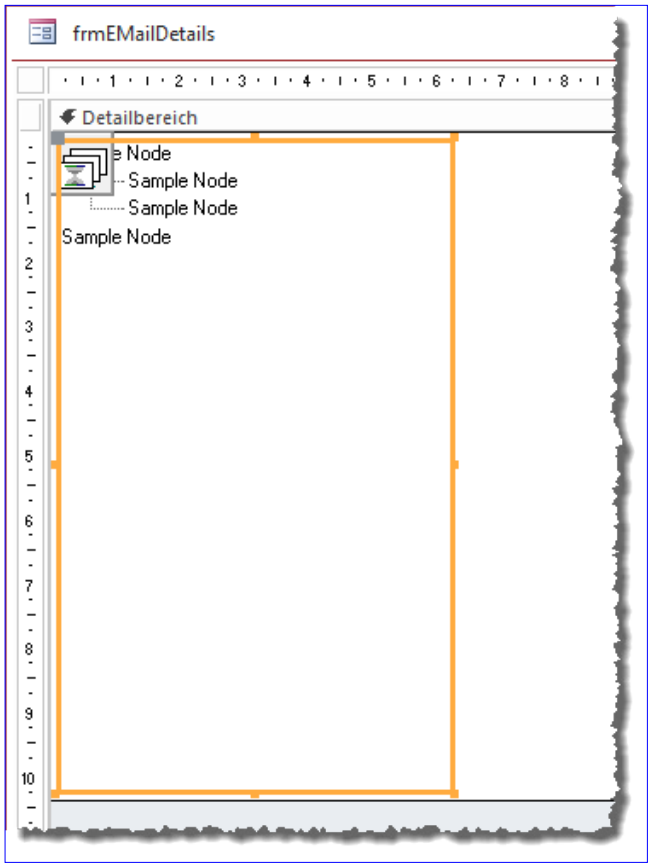

Bild 3: Die beiden neuen Steuerelemente im Formular

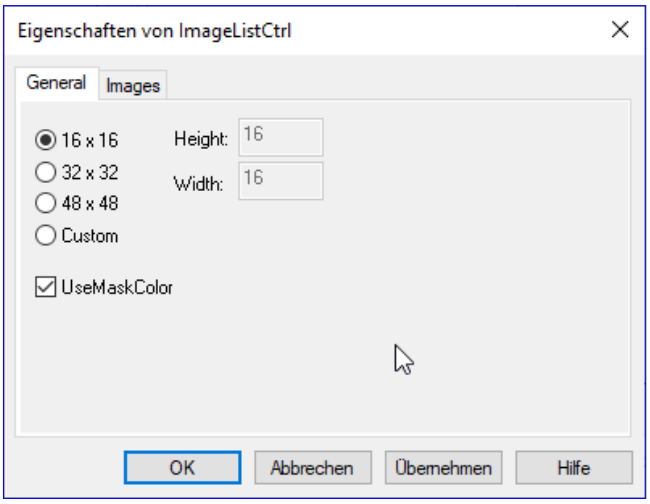

Bild 4: Festlegen der Icon-Größe

Hier fügen Sie nacheinander zwei Icons ein, die einen geschlossenen und einen geöffneten Ordner anzeigen (siehe Bild 5). Für diese beiden Elemente legen Sie im Feld **Key** die Schlüssel **folder_closed** und **folder_open** fest.

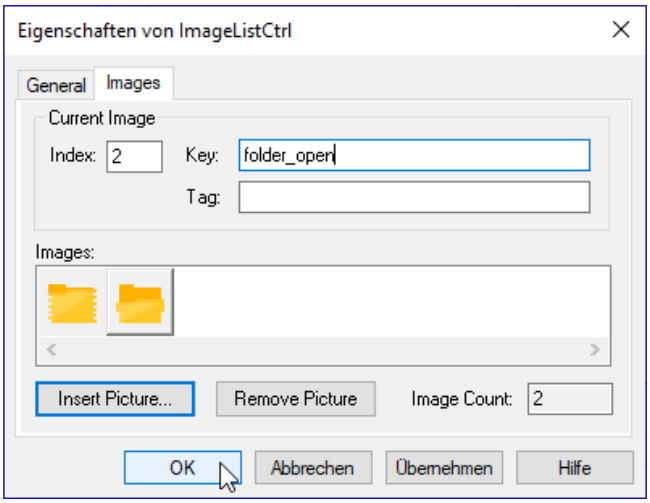

Bild 5: Hinzufügen und benennen der Icons

Einstellungen für das TreeView-Steuerelement

Danach nehmen wir die Einstellungen für das **TreeView**-Steuerelement vor. Das erledigen wir in der Regel per VBA, weil sich so ein Satz von Einstellungen recht leicht von einer Lösung zur nächsten übertragen lässt – das geht schneller, als wenn Sie die Einstellungen jedes Mal erneut im Eigenschaften-Fenster erledigen.

Die Einstellungen nehmen wir in einer eigenen Prozedur vor, die wir **InitializeTreeView** nennen. Diese rufen wir in der Ereignisprozedur auf, die beim Laden des Formulars ausgelöst wird:

```
Private Sub Form_Load()
    InitializeTreeView
    ...
End Sub
```

Die Prozedur **InitializeTreeView** sieht wie folgt aus:

```
Private Sub InitializeTreeView()
    Dim objTreeView As MSComctlLib.TreeView
    Set objTreeView = Me!ctlTreeView.Object
    With objTreeView
        .Appearance = ccFlat
        .BorderStyle = ccNone
        .Font.Name = "Calibri"
        .Font.Size = 9
        .Indentation = 200
        Set .ImageList = Me!ctlImageList.Object
        .LineStyle = tvwRootLines
        .Style = tvwTreelinesPlusMinusPictureText
        .Nodes.Clear
        FillTreeView objTreeView
        ExpandTreeViewNodes objTreeView
    End With
End Sub
```

Sie füllt zunächst die Variable **objTreeView** mit einem Verweis auf das **TreeView**-Steuerelement. Dann stellt sie einige Eigenschaften für das so referenzierte **TreeView**-Steuerelement ein. **Appearance** gibt an, ob es in einer 3D-Ansicht oder flach angezeigt werden soll.

Mit **BorderStyle** legen wir fest, dass kein Rahmen erscheinen soll – wenn, dann definieren wir diesen über die Steuerelementeigenschaften des Containers für das ActiveX-Steuerelement. Mit **Font** referenzieren wir das **Font**-Objekt, mit dessen Eigenschaften **Name** und **Size** wir die Schriftart und die Schriftgröße einstellen.

Den Standardeinzug reduzieren wir mit der Eigenschaft **Indentation** auf den Wert **200** (standardmäßig **567**). Die Eigenschaft **ImageList** legt das zu verwendende **ImageList**-Steuerelement fest, aus dem die Icons für die **TreeView**-Elemente bezogen werden. Hier müssen wir wie beim **TreeView**-Steuerelement mit **Object** auf das im ActiveX-Container enthaltene Steuerelement zugreifen. **LineStyle** gibt mit dem Wert **tvwRootLines** an, dass wir schon für die Root-Elemente Linien und somit auch Plus/Minus-Zeichen anzeigen wollen. Und mit **Style** und dem Wert **tvwTreelinesPlusMinusPictureText** bestimmen wir, dass Linien, Plus/Minus-Zeichen, Icons und Text für die

```
Private Sub FillTreeView(objTreeView As MSComctlLib.TreeView)
    Dim objOutlook As Outlook.Application
    Dim objNamespace As Outlook.Namespace
    Dim objFolder As Outlook.Folder
    Dim objNode As MSComctlLib.Node
    Set objOutlook = New Outlook.Application
    Set objNamespace = objOutlook.GetNamespace("MAPI")
    For Each objFolder In objNamespace.Folders
        If objFolder.DefaultItemType = olMailItem Then
            Set objNode = objTreeView.Nodes.Add(, , objFolder.FolderPath, objFolder.Name, "folder_closed")
            FillNode_Rek objTreeView, objNode, objFolder
        End If
    Next objFolder
End Sub
```

Listing 1: **TreeView**-Steuerelement füllen

einzelnen Elemente angezeigt werden sollen – also das komplette Programm.

Schließlich leeren wir die **Nodes**-Auflistung des Steuerelements, damit eventuell bei einer früheren Anzeige noch verbliebene Elemente verschwinden und rufen die Prozedur **FillTreeView** auf, der wir den Verweis auf das zu füllende **TreeView**-Steuerelement übergeben. Nachdem dies geschehen ist, wollen wir noch den Eingeklappt/Ausgeklappt-Zustand der Elemente wiederherstellen, der vor dem letzten Schließen des Formulars Bestand hatte. Das erledigen wir mit der Prozedur **ExpandTreeViewNodes**, dem wir ebenfalls einen Verweis auf das **TreeView**-Steuerelement übergeben.

Füllen des TreeView-Steuerelements mit den Outlook-Ordner

Die Prozedur **FillTreeView** nimmt mit dem Parameter **objTreeView** einen Verweis auf das zu füllende **TreeView**-Steuerelement entgegen (siehe Listing 1). Sie referenziert eine neue oder die aktive Outlook-Instanz mit der Variablen **objOutlook** und weist der Variablen **objNamespace** dann mit der **GetNamespace**-Methode den MAPI-Namespace zu. Anschließend durchläuft sie alle **Folder**-Objekte, die direkt im MAPI-Namespace enthalten sind. Typischerweise sind das Ordner wie Outlook oder solche, welche nach der E-Mail-Adresse benannt sind, für die eine eigene **.pst**-Datei angelegt wurde. Diese **Folder**-Objekte durchläuft die Prozedur in einer **For Each**-Schleife, wobei sie das jeweilige Objekt mit der Variablen **objFolder** referenziert.

Innerhalb der Schleife prüft die Prozedur den Typ der standardmäßig in diesem Ordner enthaltenen Elemente. Die mit der Eigenschaft **DefaultItemType** ermittelte Einstellung soll in diesem Fall **olMailItem** lauten, da wir nur die Mailordner anzeigen wollen.

Handelt es sich um einen Ordner für E-Mails, legt die Prozedur für diesen ein neues **Node**-Element im **TreeView**-Steuerelement an und referenziert dieses mit der Variablen **objNode**. Das Anlegen erledigen wir mit der **Add**-Methode der **Nodes**-Auflistung des Steuerelements. Dieser übergeben wir für den dritten Parameter den Wert der Eigenschaft **FolderPath** des Ordners, also beispielsweise **//Outlook**. Dieser Parameter nimmt den Wert für die Eigenschaft **Key** auf. Über den Schlüssel legen wir einen eindeutigen Bezeichner für jedes Element im **TreeView**-Steuerelement fest. Daher eignet sich der Wert der Eigenschaft **FolderPath** ausgezeichnet – dieser ist für jeden Ordner in Outlook eindeutig. Der vierte Parameter legt den Wert der Eigenschaft **Name** fest. Hier übergeben wir auch den Namen des jeweiligen Ordners. Der fünfte Parameter schließlich erwartet die Angabe des Schlüssels

SQL Server-Security – Teil 5: Rechte für Abteilungen

Bernd Jungbluth - Horn

Die aktuelle Rechtevergabe der Beispielapplikation erlaubt den Anwendern eine Anmeldung am SQL Server und den Zugang zur Datenbank WaWi_SQL. Innerhalb der Datenbank ist ihnen das Lesen und Schreiben der Daten sowie das Ausführen von Gespeicherten Prozeduren erlaubt. Ein solch pauschales Berechtigungskonzept beinhaltet viel zu viele Rechte. Den Anwendern stehen alle Daten zur Verfügung – sogar die Daten, die sie besser nicht lesen oder ändern sollten. Um das zu verhindern, bedarf es einer detaillierteren Rechtevergabe. Eine mögliche Variante zeigt Ihnen dieser Beitrag.

Warnung

Die beschriebenen Aktionen haben Auswirkungen auf Ihre SQL Server-Installation. Führen Sie die Aktionen nur in einer Testumgebung aus. Verwenden Sie unter keinen Umständen Ihren produktiven SQL Server!

Aktueller Stand

Das derzeitige Berechtigungskonzept basiert auf der SQL Server-Anmeldung **WaWiMa**. Mit dieser Anmeldung stellt die Access-Applikation **WaWi** die Verbindung zur SQL Server-Datenbank **WaWi_SQL** her. Ob nun eine Mitarbeiterin aus dem Vertrieb oder ein Mitarbeiter aus der Personalabteilung mit der Access-Applikation **WaWi** arbeitet, der Datenzugriff erfolgt immer über die gleiche Anmeldung. Die Zugriffsrechte werden folglich nicht über den Anwender, sondern über die Applikation gesteuert.

Welcher Anwender in der Applikation mit welchen Daten arbeiten darf, wird beim Start der Applikation bestimmt. Im Start-Formular ermittelt die VBA-Funktion **fBerechtigungen** den aktuellen Windows-Benutzer und aktiviert entweder die Schaltflächen für den Vertrieb oder die für die Personalverwaltung.

Auf diese Weise wird verhindert, dass ein Mitarbeiter des Vertriebs die Funktionen der Personalverwaltung nutzen kann. Allerdings schränkt dies nur den Zugriff auf die Funktionen ein, nicht aber den auf die Daten. Die Zugriffssteuerung findet ausschließlich im Start-Formular statt. Umgeht ein Anwender dieses Formular, stehen ihm alle Daten der Datenbank **WaWi_SQL** zur Verfügung. Dazu gehören auch die Daten der Personalverwaltung.

Das gilt es zu vermeiden. Dazu wird das Berechtigungskonzept um eine weitere Anmeldung im SQL Server ergänzt. Das Ziel ist eine Rechtevergabe auf Abteilungsebene.

Pro Abteilung eine Anmeldung

Das neue Berechtigungskonzept enthält zwei SQL Server-Anmeldungen, eine für die Installationen der Access-Applikation **WaWi** im Vertrieb und eine weitere für die Installationen in der Personalabteilung. Zur Umsetzung des neuen Berechtigungskonzepts ist zuerst eine Anmeldung im SQL Server anzulegen und der Datenbank **WaWi_SQL** zuzuordnen.

Der dabei erstellte Benutzer erhält in der Datenbank pauschale Lese- und Schreibrechte sowie das Recht zum Ausführen aller gespeicherten Prozeduren. Auf den Rechnern der Personalabteilung wird dann in den bestehenden ODBC-Datenquellen die neue Anmeldung eingetragen. Abschließend sind in der Access-Applikation die Tabellen neu einzubinden und die Verbindungsei-

genschaften der Pass-Through-Abfragen sowie die der ADO-Zugriffe anzupassen.

Auf den Rechnern des Vertriebs sind keine Änderungen notwendig. Die Verbindung von der Access-Applikation zur SQL Server-Datenbank erfolgt dort weiterhin über die Anmeldung **WaWiMa**. Allerdings wird diese in ihren Rechten eingeschränkt. Die Zugriffe auf die für die Personalverwaltung relevanten Tabellen und gespeicherten Prozeduren werden verweigert. Das Einschränken dieser Rechte findet im SQL Server statt.

Doch der Reihe nach. Als Erstes erstellen Sie die Beispielumgebung mit der SQL Server-Datenbank **WaWi_SQL**, der gleichnamigen ODBC-Datenquelle und der Access-Applikation **WaWi**. Die hierzu notwendigen Schritte sehen Sie in der Installationsanleitung am Ende dieses Beitrags.

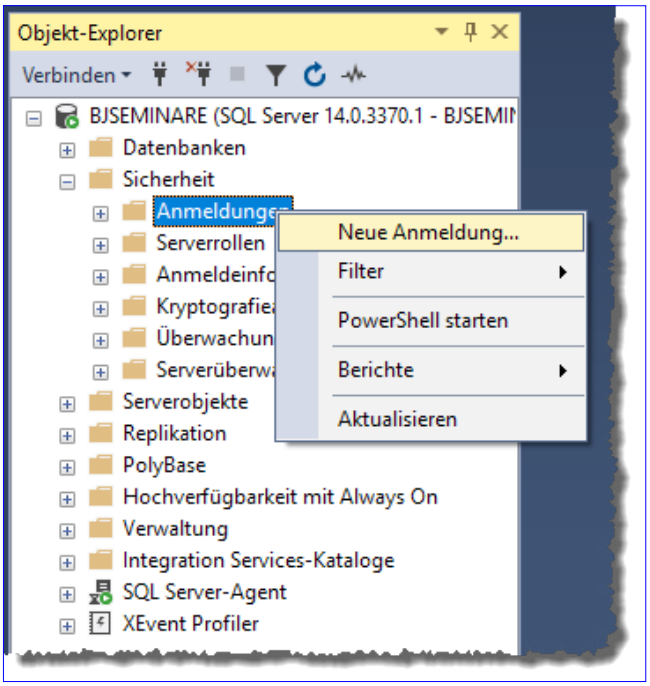

Bild 1: Der Weg zur neuen Anmeldung

Danach legen Sie die neue SQL Server-Anmeldung an. Öffnen Sie dazu das SQL Server Management Studio und verbinden Sie sich mit Ihrem SQL Server. Im Objekt-Explorer erweitern Sie den Ordner **Sicherheit** und wählen im Kontextmenü des Unterordners **Anmeldungen** den Eintrag **Neue Anmeldung** (siehe Bild 1). Im Dialog **Anmeldung - Neu** geben Sie der Anmeldung den Namen **WaWiPersonal** und aktivieren die Option **SQL Server-Authentifizierung**.

Das Kennwort zur Anmeldung tragen Sie in den beiden Eingabefeldern **Kennwort** und **Kennwort bestätigen** ein. Vergeben Sie ein gutes Kennwort. Eines, das nicht leicht zu erraten ist. Immerhin soll es die Daten der Personalverwaltung vor unbefugtem Zugriff schützen. Stellen Sie sich nur einmal die Diskussionen vor, wenn unerlaubterweise die Gehälter der Mitarbeiter veröffentlicht würden!

Die Option **Kennwortrichtline** unterstützt Sie bei der Vergabe des Kennworts. Ist die Option aktiviert, muss das Kennwort den in Ihrem Unternehmen gültigen Richtlinien zur Vergabe von Kennwörtern entsprechen. Sollten Sie keine Kennwortrichtlinien definiert haben, können Sie das Kennwort nach einem Muster erstellen oder Sie lassen sich eines in einem Passwort-Safe generieren. Ein mögliches Muster wie die Vor- und Nachteile beider Vorgehensweisen wurden im dritten Teil dieser Beitragsreihe beschrieben.

Kennwörter sollten Sie regelmäßig ändern. Bei aktivierter Option **Ablauf des Kennworts erzwingen** erinnert Sie der SQL Server gemäß Ihrer Kennwortrichtlinien an diese Empfehlung.

Diese Art der Erinnerung kann sich jedoch schnell kontraproduktiv auswirken. Ist das Kennwort abgelaufen, verlangt der nächste Anmeldeversuch die Vergabe eines neuen Kennworts. Ein neues Kennwort ist schnell vergeben.

Das ist aber nur ein Teil der Aufgabe. Danach sind noch alle Client-Applikationen, die diese Anmeldung verwenden, an das neue Kennwort anzupassen. Je nach Anzahl

der Client-Applikationen ist das schnell ein größerer Aufwand. Am besten deaktivieren Sie die Option **Ablauf des Kennworts erzwingen**. Das spart Ihnen unangenehme Überraschungen. Legen Sie sich stattdessen besser einen Termin an, der Sie an die Vergabe eines neuen Kennworts erinnert.

Das Entfernen des Häkchens in der Option **Ablauf des Kennworts erzwingen** deaktiviert sinnvollerweise ebenso die Option **Benutzer muss das Kennwort bei der nächsten Anmeldung ändern**.

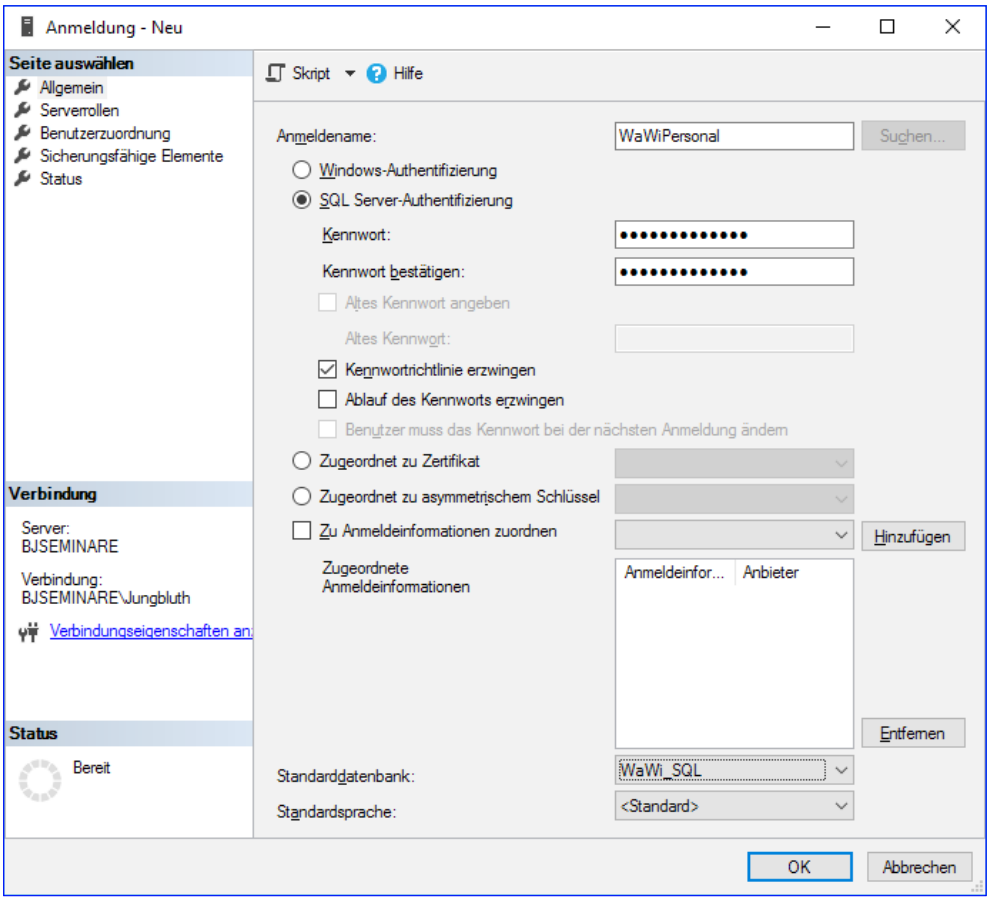

Bild 2: Die Anmeldung WaWiPersonal

Es macht keinen Unterschied, ob das Kennwort beim Ablauf oder bei der nächsten Anmeldung geändert werden muss. Mit der Kennwortänderung alleine ist es nicht getan. Es hat immer eine Anpassung der abhängigen Client-Applikationen zur Folge.

Als Nächstes legen Sie die Standarddatenbank der Anmeldung fest. Die Standarddatenbank wird verwendet, wenn beim Aufbau der Verbindung die Datenbank nicht explizit angegeben ist. Wählen Sie in der Auswahlliste **Standarddatenbank** den Eintrag **WaWi_SQL**.

Die Konfiguration der neuen Anmeldung wäre damit soweit abgeschlossen (siehe Bild 2). Dennoch sollten Sie nicht auf **OK** klicken. In der Seite **Benutzerzuordnung** können Sie jetzt nämlich direkt den zugehörigen Benutzer in der Datenbank **WaWi_SQL** anlegen und ihm die entsprechenden Rechte geben.

Wechseln Sie zur Seite **Benutzerzuordnung** und wählen Sie im oberen Bereich die Datenbank **WaWi_SQL** aus. Durch diese Auswahl wird der Benutzer **WaWiPersonal** in der Datenbank erstellt. Im unteren Bereich geben Sie dem Benutzer die Zugriffsrechte.

Aktivieren Sie hierzu die Systemdatenbankrollen **db_datareader** und **db_datawriter** plus die Benutzerdatenbankrolle **edb_execute** (siehe Bild 3). Diese Datenbankrolle wurde im vorheigen Beitrag dieser Reihe eingeführt (**SQL Server-Security, Teil 4: Schutz mit Datenbankrollen**, **www.access-im-unternehmen.de/1274**).

Sie beinhaltet die Rechte zur Ausführung aller gespeicherten Prozeduren. Um die Datenbankrolle **public** müssen Sie sich vorerst nicht kümmern. Hierbei handelt es sich um eine Standardzuordnung. Die Vorteile und vor

allem die Nachteile dieser Datenbankrolle lernen Sie in einem der nächsten Beiträge kennen.

Ein Klick auf **OK** erstellt zuerst die Anmeldung **WaWiPersonal** und dann in der Datenbank **WaWi_SQL** den zugehörigen Benutzer. Die Anmeldung sehen Sie im Unterordner **Anmeldungen** des Ordners **Sicherheit**. Den Benutzer finden Sie in der Datenbank **WaWi_SQL**. Dort gibt es ebenfalls einen Ordner namens **Sicherheit**. Dieser listet die Benutzer der Datenbank im Ordner **Benutzer** auf.

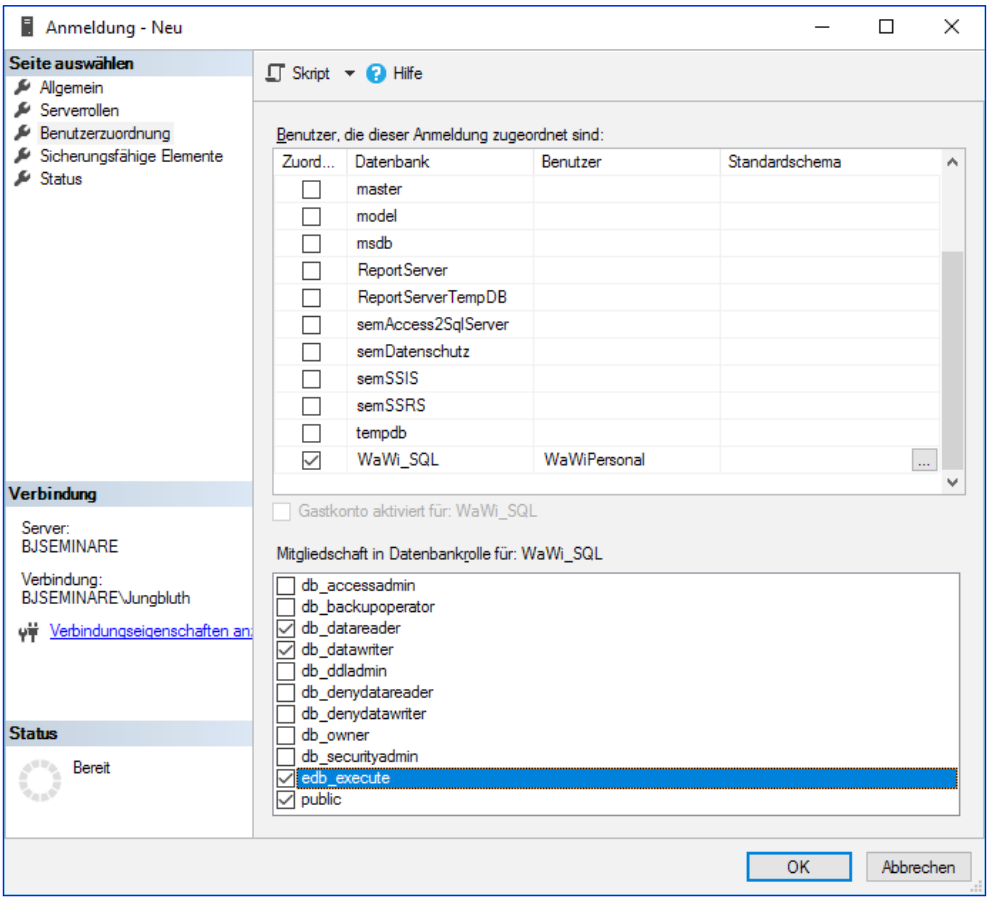

Bild 3: Der Benutzer WaWiPersonal

Das Berechtigungskonzept ist nun um eine Anmeldung reicher. Mit dieser Anmeldung authentifiziert sich der Anwender am SQL Server. Die Anmeldung ist in der Datenbank **WaWi_SQL** fest mit dem gleichnamigen Benutzer verbunden.

Diese Zuordnung autorisiert die Anmeldung und somit den Anwender für den Zugang zur Datenbank. Die Rechte für den Datenzugriff sind am Benutzer definiert. Dieser ist aktuell den Datenbankrollen **db_datareader**, **db_datawriter** und **edb_execute** zugeordnet. Der Anwender darf mit diesen Rechten in der Datenbank jegliche Daten lesen und schreiben sowie alle gespeicherten Prozeduren ausführen.

Soweit in Kurzform die Beschreibung der Anmeldung **WaWiPersonal** innerhalb der mehrstufigen Sicherheitsarchitektur von SQL Server.

Das ist aber noch nicht alles. Schließlich erfolgt der Anmeldevorgang am SQL Server aktuell nicht durch den Anwender, sondern automatisch in der Access-Applikation **WaWi**. Die muss jetzt noch an die neue Anmeldung angepasst werden.

Eine Anmeldung für die Personalabteilung

In der Realität würden Sie nun in die Personalabteilung gehen und dort auf den einzelnen Rechnern die Access-Applikation mitsamt der ODBC-Datenquelle anpassen.

Um dieses Szenario nachzustellen, kopieren Sie die Access-Applikation **WaWi** und geben der Kopie den Namen **WaWi Personal**. Als nächstes ändern Sie die ODBC-Datenquelle **WaWi_SQL**. Den ODBC-Datenquellen-Administrator starten Sie am besten über die Windows-Suche. Dazu drücken Sie die Windows-Taste und tippen **ODBC**.

Das Suchergebnis liefert Ihnen zwei Einträge. Wählen Sie je nach installierter Access-Version den Eintrag **ODBC-Datenquellen (32-Bit)** oder **ODBC-Datenquellen (64-Bit)**.

Im ODBC-Datenquellen-Administrator wechseln Sie zur Registerkarte **System-DSN**, markieren dort die ODBC-Datenquelle **WaWi_SQL** und klicken auf **Konfigurieren**. Den ersten Schritt des Assistenten überspringen Sie mit der Schaltfläche **Weiter**.

Im zweiten Schritt passen Sie die Anmeldedaten an. Geben Sie im Eingabefeld **Anmelde-ID** den Namen der neuen Anmeldung und im Feld **Kennwort** das zugehörige Kennwort ein (siehe Bild 4). Danach bestätigen Sie diesen sowie den nächsten Schritt des Assistenten mit einem Klick auf **Weiter** und beenden die Konfiguration im letzten Schritt mit der Schaltfläche **Fertig stellen**.

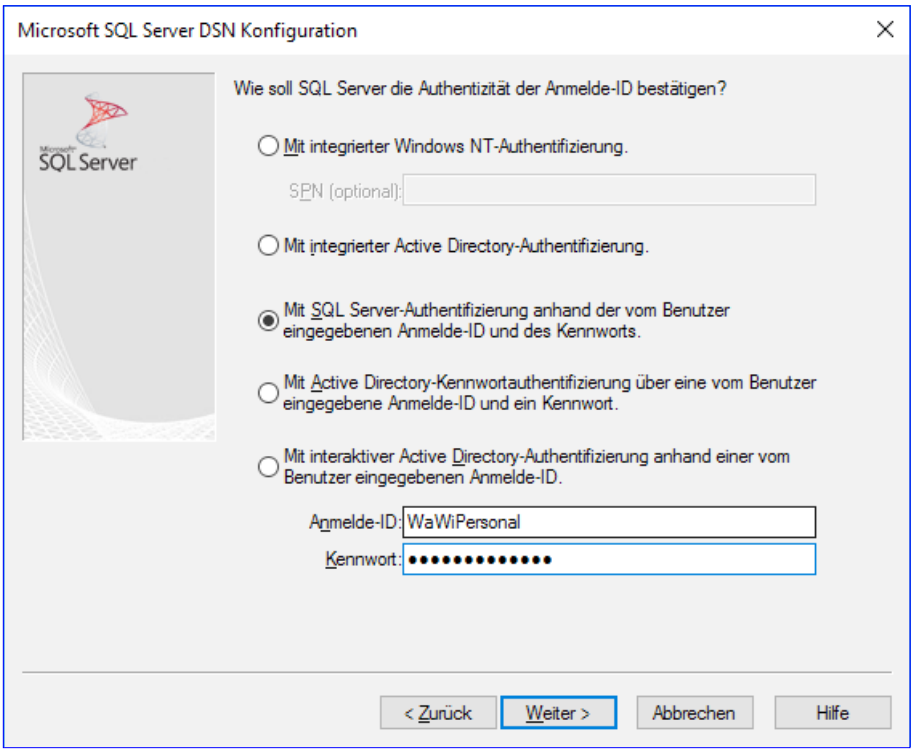

Bild 4: Ändern der ODBC-Datenquelle

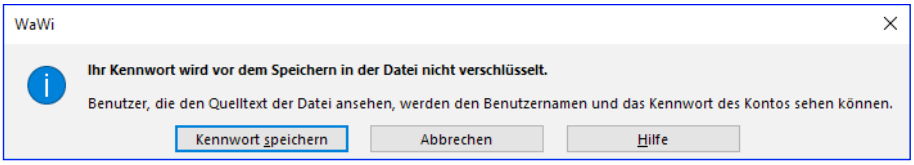

Bild 5: Speichern des Kennworts

Es folgt eine Zusammenfassung der Konfiguration und die Möglichkeit, die neuen Anmeldedaten zu testen. Sollte das Ergebnis nach einem Klick auf **Datenquelle testen** nicht erfolgreich sein, haben Sie sich vielleicht beim Kennwort vertippt. In diesem Fall wiederholen Sie die eben beschriebenen Schritte. Nach einem erfolgreichen Test ist die Änderung der ODBC-Datenquelle abgeschlossen.

Soweit so gut. Die ODBC-Datenquelle verwendet nun die Anmeldung **WaWiPersonal**. Jetzt passen Sie noch die Access-Applikation an die neue Anmeldung an. In der Realität wäre dies auf jedem Rechner der Personalabteilung erforderlich. In der Beispielumgebung öffnen Sie dazu die eben kopierte Access-Applikation **WaWi Personal**. Dort sind die Tabellen noch mit der Anmeldung **WaWiMa** eingebunden.

Aus diesem Grund entfernen Sie alle eingebundenen Tabellen und binden diese neu ein. Starten Sie über **Externe Daten – Neue Datenquelle – Aus Datenbank – Aus SQL Server** den Dialog **Externe Daten - ODBC-Datenbank** und aktivieren Sie dort die zweite Option.

Mit einem Klick auf **OK** kommen Sie zum nächsten Dialog, in dem Sie in der Registerkarte **Computerdatenquelle** die ODBC-Datenquelle **WaWi_SQL** per

E-Mails verwalten mit dem Outlook View Control

Im Beitrag »Outlook-Folder in Access anzeigen« haben wir gezeigt, wie Sie das Outlook View Control in ein Formular integrieren, um damit die Ordner von Outlook anzuzeigen. Im vorliegenden Beitrag bauen wir auf den dort vorgestellten Techniken auf und gehen genauer auf den Umgang mit dem E-Mail-Ordnern ein. Dabei wollen wir Details wie den Betreff, den Inhalt oder den Empfänger oder Absender der aktuell markierten E-Mail in entsprechenden Steuerelementen anzeigen. Außerdem wollen wir die Anzeige der Outlook-Ordner in ein TreeView-Steuerelement verlagern.

Das **Outlook View Control** ist eine tolle Erweiterung, wenn Sie beispielsweise die Outlook-Ordner zur Anzeige von E-Mails wie **Posteingang**, **Postausgang**, **Gesendete** Elemente oder auch benutzerdefinierte Ordner in einer Access-Anwendung anzeigen wollen.

Das **Outlook View Control** können Sie jedoch nicht einfach zu einem Access-Formular hinzufügen, sondern Sie müssen es in ein **Frame**-Steuerelement einbetten. Wie das gelingt, zeigen wir im Beitrag **Outlook-Folder in Access anzeigen** (**www.access-im-unternehmen.de/1292**).

In diesem Beitrag haben wir gezeigt, wie Sie die Outlook-Ordner in ein Listenfeld einlesen und ihre Inhalte durch Anklicken des jeweiligen Listeneintrags im **Outlook View Control** anzeigen.

Da die Anzeige in einem Listenfeld wenig professionell aussieht, haben wir in einem weiteren Beitrag namens **TreeView für Outlook-Ordner** ein **TreeView**-Steuerelement zum Formular hinzugefügt, das alle Ordner mit E-Mails anzeigt (**www.access-im-unternehmen.de/1293**). Der Zwischenstand nach diesem Beitrag sieht wie in Bild 1 aus.

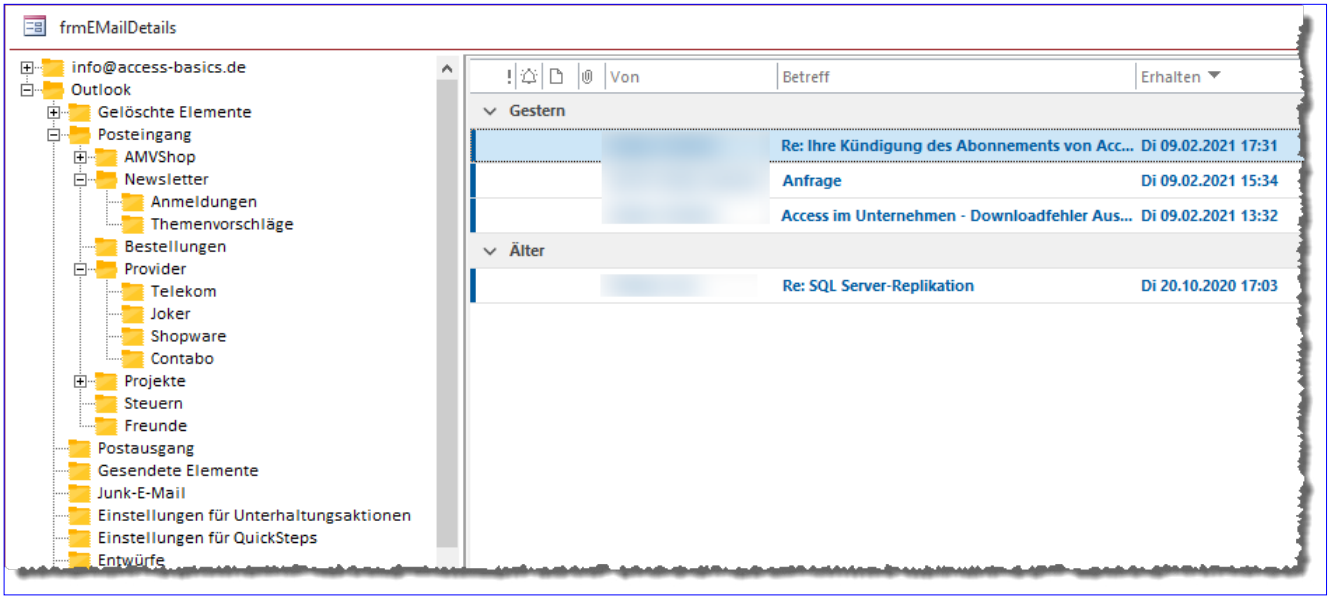

Bild 1: Outlook-Ordner und der Inhalt eines E-Mail-Ordners

Damit kommen wir zum Ziel des vorliegenden Beitrags. Hier wollen wir die übrigen Elemente, die für den Umgang mit E-Mails notwendig sind, hinzufügen.

Dazu gehören die folgenden:

- Gewählten Ordner aus dem **TreeView**-Steuerelement im **Outlook View Control** anzeigen
- Zuletzt angezeigten Ordner speichern und wiederherstellen
- Inhalt der aktuell markierten E-Mail anzeigen (Absender, Empfänger, Betreff, Inhalt, Eingangsdatum)
- Empfangen von E-Mails
- Erstellen einer neuen E-Mail im Outlook-Inspektor
- Öffnen einer E-Mail mit dem dazugehörigen Outlook-Inspektor
- Löschen von E-Mails

Gewählten Ordner im Outlook View Control anzeigen

Das Formluar **frmEMailDetails** enthält das **TreeView**-Steuerelement namens **ctlTreeView** sowie ein **Frame**-Objekt und ein **ViewCtl**-Objekt. Das **ViewCtl**-Objekt ist in das **Frame**-Objekt eingebettet.

Das **Frame**- und das **ViewCtl**-Objekt deklarieren wir im Kopf des Klassenmoduls des Formulars wie folgt:

```
Private objFrame As Frame
Private WithEvents objView As ViewCtl
```

In der beim Laden des Formulars ausgelösten Prozedur **Form_Load** initialisieren wir das **TreeView**-Steuerelement (siehe Beitrag **TreeView für Outlook-Ordner**) und weisen den beiden Variablen die Elemente zu:

```
Private Sub Form_Load()
    InitializeTreeView
    Set objFrame = Me!ctlFrame.Object
    Set objView = objFrame.Controls(0)
End Sub
```

Nun wollen wir dafür sorgen, dass der vom Benutzer im **TreeView**-Steuerelement angeklickte Ordner im **Outlook View Control** angezeigt wird. Dazu legen wir eine Ereignisprozedur an, die beim Anklicken eines der Elemente des **TreeView**-Steuerelements ausgelöst wird:

```
Private Sub ctlTreeView_NodeClick(ByVal Node As Object)
    Dim objNode As MSComctlLib.Node
    Set objNode = Node
    objView.Folder = objNode.Key
End Sub
```

Das wir das mit dem Parameter **Node** übergebene **Node**-Objekt noch einer Variablen des Typs **MSComctlLib.Node** zuweisen, liegt daran, dass wir so die IntelliSense-Funktion für diese Variable nutzen können. So können wir den in der **Key**-Eigenschaft des **Node**-Elements gespeicherten Outlook-Pfad wie zum Beispiel **\\Outlook\Posteingang** nutzen und diesen der **Folder**-Eigenschaft des **Outlook View Control**-Steuerelements zuweisen.

Zuletzt gewählten Ordner speichern und beim Öffnen wiederherstellen

Nun, da das **Outlook View Control** den Outlook-Ordner anzeigt, den der Benutzer im **TreeView**-Steuerelement ausgewählt hat, wollen wir diesen auch speichern, damit dieser beim nächsten Öffnen auch wieder angezeigt werden kann. Dazu legen wir eine Optionen-Tabelle namens **tblOptions** an. Diese soll lediglich zwei Felder enthalten – ein Primärschlüsselfeld namens **OptionID** und ein Textfeld namens **CurrentMailfolder**. Den Entwurf dieser Tabelle zeigen wir in Bild 2.

Wann speichern wir den zuletzt gewählten Outlook-Ordner in der Optionen-Tabelle? Am sichersten ist es, diesen nach

jeder Auswahl erneut zu speichern. Wir können den Speichervorgang also direkt in die oben vorgestellte Ereignisprozedur integrieren, die beim Anklicken eines der **Node**-Elemente im **TreeView**-Steuerelement ausgelöst wird. Diese erweitern wir um den Aufruf einer Prozedur namens **SaveCurrentMailfolder**, der wir den Pfad zum angeklickten Ordner als Parameter übergeben:

```
Private Sub ctlTreeView_NodeClick(ByVal
Node As Object)
    ...
    SaveCurrentMailFolder objNode.Key
End Sub
```

Die Prozedur **SaveCurrentMailfolder** finden Sie in Listing 1. Sie nimmt mit dem Parameter **strFolder** den zuletzt aufgerufenen Ordner entgegen. Dann führt sie eine **UPDATE**-Abfrage aus, die das Feld **CurrentMailfolder** der Tabelle **tblOptions** auf den übergebenen Wert einstellt.

Es kann sein, dass die Tabelle leer ist, weil Sie diese beispielsweise vor dem Weitergeben an einen Benutzer geleert haben. In diesem Fall liefert die folgende Abfrage der von der **UPDATE**-Abfrage betroffenen Datensätze den Wert **0**. In dem dann ausgelösten Zweig der **If...Then**-Bedingung legen wir diesen Datensatz dann einfach neu an und weisen dem Feld **CurrentMailfolder** den Wert aus **strFolder** zu.

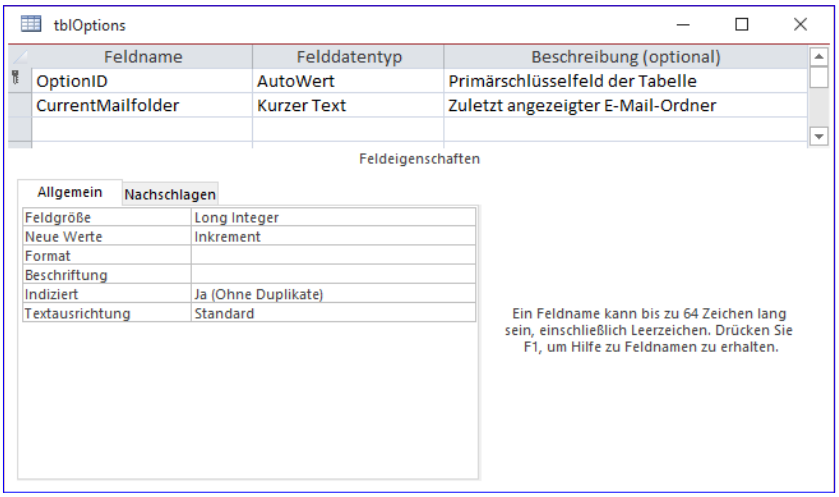

Bild 2: Tabelle zum Speichern von Optionen

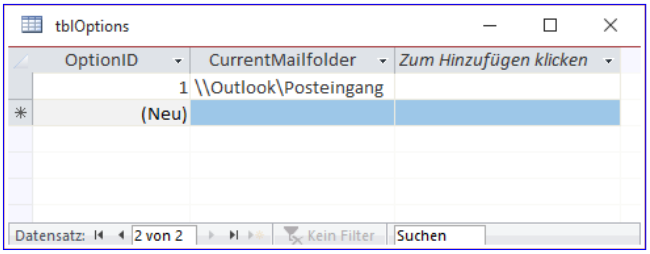

Bild 3: Zuletzt verwendeter Ordner in der Tabelle **tblOptions**

Nach dem Speichern des Wertes sieht die Tabelle **tblOptions** beispielsweise wie in Bild 3 aus.

Zuletzt gewählten Ordner beim Öffnen markieren und anzeigen

Die so gespeicherte Information wollen wir natürlich beim nächsten Öffnen des Formulars nutzen, indem wir den entsprechenden Eintrag im **TreeView**-Steuerelement selektieren und im **Outlook View Control** den passenden

```
Private Sub SaveCurrentMailfolder(strFolder As String)
    Dim db As DAO.Database
    Set db = CurrentDb
    db.Execute "UPDATE tblOptions SET CurrentMailfolder = '" & strFolder & "'", dbFailOnError
    If db.RecordsAffected = 0 Then
        db.Execute "INSERT INTO tblOptions(CurrentMailfolder) VALUES('" & strFolder & "')", dbFailOnError
    End If
End Sub
```

Listing 1: Prozedur zum Speichern des aktuell gewählten Mail-Ordners

Ordner anzeigen. Dazu legen wir eine neue Prozedur namens **SetCurrentMailfolder** an, den wir in der Prozedur **Form_Load** beim Laden des Formulars aufrufen:

```
Private Sub Form_Load()
    ...
    SetCurrentMailfolder
End Sub
```

```
Private Sub SetCurrentMailfolder()
    Dim strFolder As String
    Dim objTreeView As MSComctlLib.TreeView
    strFolder = Nz(DLookup("CurrentMailfolder", "tblOptions"), "")
    If Len(strFolder) = 0 Then
       strFolder = GetDefaultFolder(olFolderInbox)
    End If
    objView.Folder = strFolder
    Set objTreeView = ctlTreeView.Object
    objTreeView.SelectedItem = objTreeView.Nodes(strFolder)
    Me!ctlTreeView.SetFocus
End Sub
```

Listing 2: Prozedur zum Auswählen des zuletzt betrachteten E-Mail-Ordners

Diese Prozedur liefert Listing 2. Die Prozedur ermittelt zunächst den Wert des Feldes **CurrentMailFolder** aus der Tabelle **tblOptions** und schreibt diesen in die Variable **strFolder**. Wenn noch kein Eintrag in dieser Tabelle vorliegt oder das Feld leer ist, sorgt die **Nz**-Funktion dafür, dass **strFolder** mit einer leeren Zeichenkette gefüllt wird.

```
Private Function GetDefaultFolder(intFolder As Outlook.OlDefaultFolders) As String
    Dim objOutlook As Outlook.Application
    Dim objNamespace As Outlook.Namespace
    Dim objFolder As Outlook.Folder
    Set objOutlook = New Outlook.Application
    Set objNamespace = objOutlook.GetNamespace("MAPI")
    Set objFolder = objNamespace.GetDefaultFolder(intFolder)
    GetDefaultFolder = objFolder.FolderPath
End Function
```

Listing 3: Funktion zum Ermitteln eines Standardordners

Die folgende **If...Then**-Bedingung prüft, ob **strFolder** eine leere Zeichenkette enthält und somit kein zuletzt geöffneter E-Mail-Ordner in der Tabelle **tblOptions** vorliegt. In diesem Fall füllt sie **strFolder** mit dem Ergebnis der Funktion **GetDefaultFolder**. Diese finden Sie in Listing 3.

GetDefaultFolder erwartet eine der Konstanten der Auflistung **Outlook.OlDefaultFolders** als Parameter. Es gibt bereits eine Methode namens **GetDefaultFolder**, die zum **Namespace**-Objekt gehört.

Um dieses zu referenzieren, benötigen wir zuvor noch ein **Outlook.Application**-Objekt. Die von uns definierte Funktion **GetDefaultFolder** vereinfacht den Zugriff auf einen der Standardordner von Outlook, indem es das Erstellen einer Instanz von **Outlook.Application** und das Zuweisen des MAPI-Namespace an die Variable **objNamespace** kapselt und das **Folder**-Objekt ermittelt, das dem Parameter **intFolder** entspricht, also beispielsweise **olFolderInbox** für den Posteingang. Die Funktion **GetDefaultFolder** gibt schließlich den Pfad zu dem angegebenen Ordner zurück.

Dadurch, dass wir den Parameter mit dem Typ **Outlook.OlDefaultFolders** deklarieren, können wir beim Aufruf per IntelliSense aus der Liste der verfügbaren Werte auswählen (siehe Bild 4).

Diesen nimmt die Prozedur **SetCurrentMailFolder** entgegen und speichert ihn in der Variablen **strFolder**. Damit stellt sie nun zunächst die **Folder**-Eigenschaft des **Outlook View Controls** ein. Danach referenziert sie das **TreeView**-Steuerelement mit der Variablen **objTreeView**

und stellt die Eigenschaft **SelectedItem** auf das **Node**-Element ein, dass die **Nodes**-Auflistung für den als Parameter verwenden Wert aus **strFolder** (zum Beispiel **\\Outlook\Posteingang**) liefert.

Schließlich wird der Fokus auf das **TreeView**-Steuerelement verschoben, damit der Benutzer direkt den selektierten Ordner erkennen kann.

Müssen wir die übergeordneten Ordner aufklappen, damit der markierte Eintrag sichtbar wird? Es kann ja auch sein, dass der Benutzer einen Ordner auswählt und dann alle Ordner einklappt, sodass der gewählte Ordner nicht mehr im **TreeView**-Steuerelement sichtbar ist.

Hier besteht kein Grund zur Sorge: Das **TreeView**-Steuerelement klappt automatisch alle Elemente bis zum aktuell selektierten Element auf.

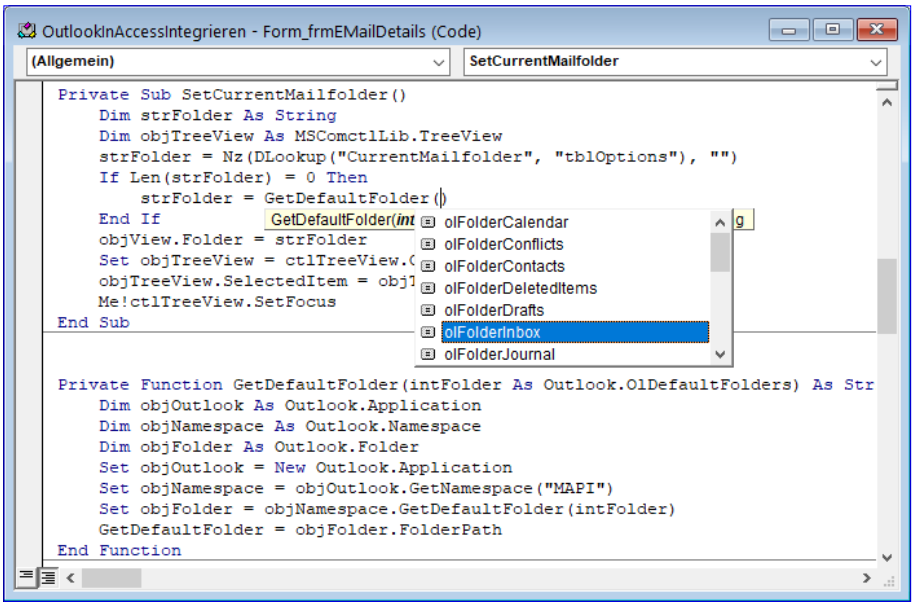

Bild 4: Auswahl des gewünschten Standardordners per IntelliSense

Wir wollen uns dabei auf folgende Elemente beschränken:

- Absender
- Empfänger

Inhalt der aktuell markierten E-Mail anzeigen

Nun wollen wir die Inhalte der aktuell im **Outlook View Control** ausgewählten E-Mail in dafür vorgesehenen Steuerelementen anzeigen.

In Outlook sieht der Bereich wie in Bild 5 aus. Die Ansicht im **Outlook View Control** liefert leider nur den oberen Teil mit der Liste der E-Mails, sodass wir den Rest selbst anlegen müssen.

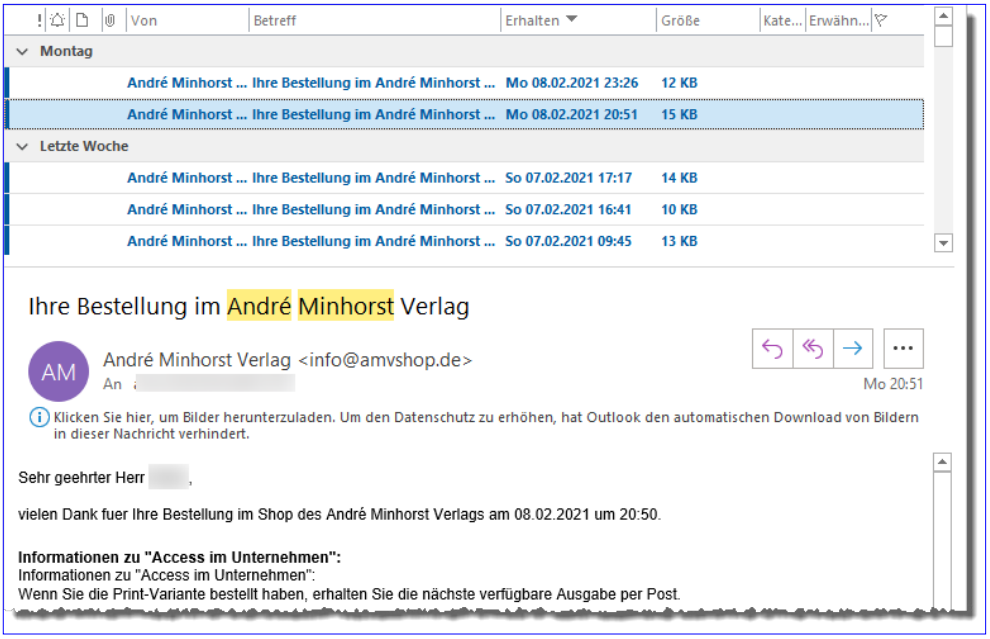

Bild 5: Anzeige einer E-Mail in Outlook

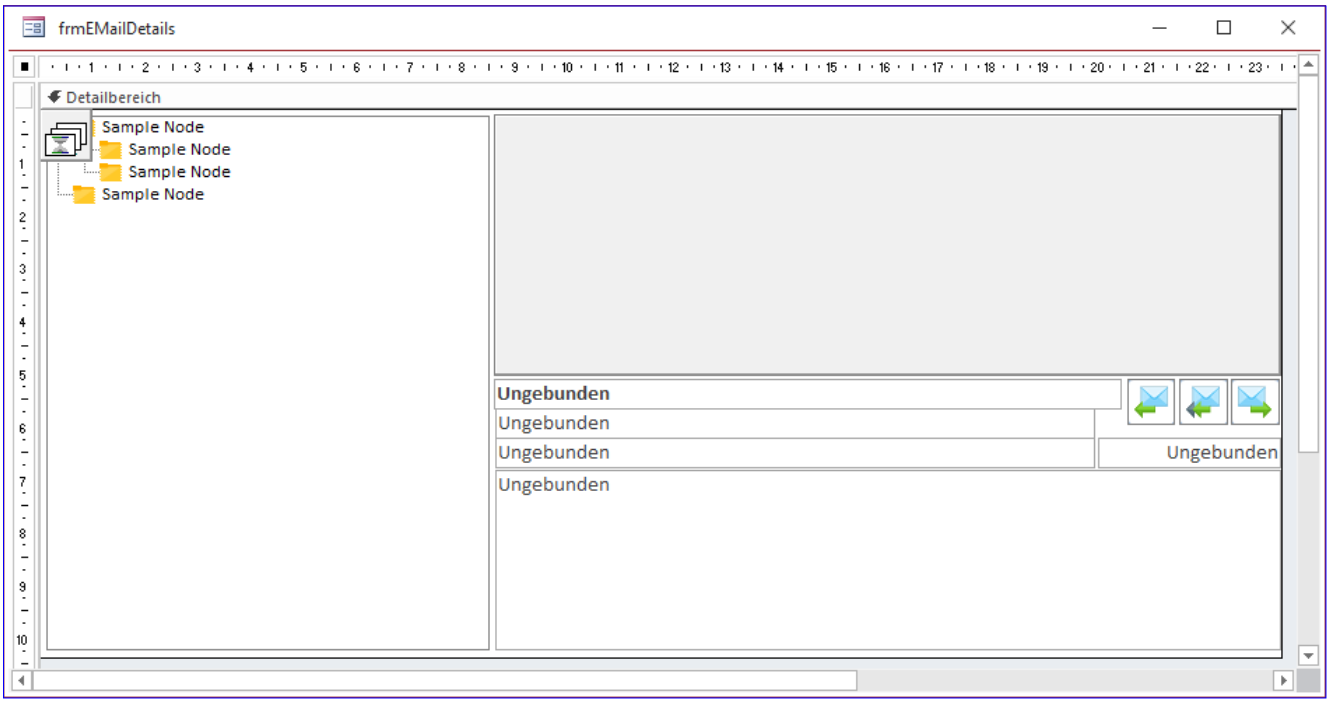

Bild 6: Hinzufügen von Steuerelementen zum Anzeigen einer E-Mail

- Betreff
- Inhalt
- Datum und Zeit des Eingangs
- Schaltflächen zum Antworten an einen Empfänger, an alle Empfänger und zum Weiterleiten

Um auch nur eine dieser Informationen anzuzeigen, benötigen wir nicht nur die entsprechenden Steuerelemente, sondern wir müssen auch erst einmal an die betreffenden Daten kommen.

Sprich: Wir benötigen das **MailItem**-Objekt, das sich hinter dem selektierten Eintrag befindet.

Doch zuerst zu den Steuerelementen. Diese fügen wir wie in Bild 6 hinzu. Die Textfelder heißen von oben nach unten **txtSubject**, **txtFrom**, **txtTo** und **txtBody**. Rechts befindet sich **txtReceived**. Die drei Schaltflächen heißen **cmdReply**, **cmdReplyAll** und **cmdForward**.

Anschließend kümmern wir uns um eine Prozedur, die beim Auswählen eines der Einträge des **Outlook View Control**-Steuerelements ausgelöst wird. Diese fügen wir hinzu, indem wir im Codefenster des Klassenmoduls des Formulars im linken Kombinationsfelder den Eintrag **objView** auswählen und im rechten den Eintrag **SelectionChange**. Diese füllen wir lediglich mit einem einzigen Befehl, der eine weitere Prozedur namens **ShowMail** aufruft.

```
Private Sub objView_SelectionChange()
    ShowMail
End Sub
```

Die Prozedur **ShowMail** finden Sie in Listing 4. Sie ermittelt mit der Funktion **GetFirstSelectedMailItem** die aktuell selektierte E-Mail. Ist **objMailItem** danach nicht leer, wurde also ein **MailItem**-Element markiert, stellt die Prozedur die Textfelder des Formulars auf die Eigenschaften des **MailItem**-Elements ein. Dabei füllt sie **txtFrom** mit dem Namen des Absenders gefolgt von der E-Mail-Adresse in Klammern. Die übrigen Textfelder werden mit