

# Benutzerdialoge

## Eigene Dialoge erstellen

11.1

Während Sie bisher den Einsatz von integrierten Word-Dialogen oder die eher rudimentäre Daten-Eingabe via *InputBox*-Funktion kennengelernt haben, geht es im Folgenden darum, eigene Dialoge zu programmieren. Dank eines grafischen Editors werden wesentliche Teile des Dialogs bequem gezeichnet und die zugehörige Programmier-Technik unterscheidet sich kaum vom herkömmlichen VBA-Code.

Die im amerikanischen Original als *userforms* bezeichneten Dialoge werden oft als *Formulare* benannt, etwa auch im Projekt-Fenster des VBA-Editors oben links. Ich werde bei der Bezeichnung *Dialoge* bleiben, weil sie erstens sonst mit Formuldokumenten verwechselt werden könnten und zweitens ohnehin typischerweise modal sind. Modale Bildschirm-Formulare, also solche, die zum Weiterarbeiten geschlossen werden müssen, werden im Windows-Sprachgebrauch generell als Dialoge bezeichnet.



**Formulare und Dialoge**

Das hier beispielhaft erstellte *docuMentor*-Projekt in diesem Teil soll Ihnen zusammen mit dem Vorwissen aus den bisherigen Kapiteln zeigen, wie eine Vorlagen- und Add-In-Programmierung mit Word aussieht. Dabei fügen sich die schon bekannten Programmiertricks zu einem gemeinsamen Projekt zusammen.

- 1 Erstellen Sie bitte zuerst eine leere Datei und speichern diese im Startup-Verzeichnis als Add-In unter dem Namen »docuMentor.dotm«. Auch der interne Name sollte gleich jetzt im VBA-Editor von Project auf eine eindeutige Bezeichnung wie *dM\_AddIn* geändert werden.
- 2 Mit dem Menübefehl *Einfügen/UserForm* legen Sie dann im VBA-Editor einen leeren Dialog an:

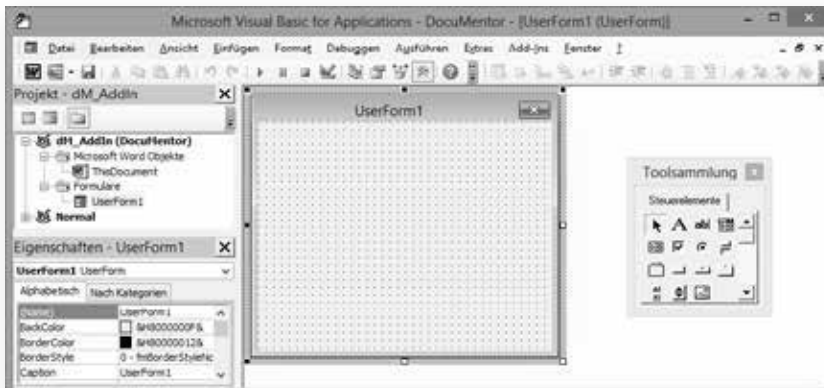


Abbildung 11.1: Ein gerade eingefügter Dialog im Entwurf



- 3 Solange der Dialog hier im Entwurfsmodus markiert ist, sehen Sie die zugehörige Symbolleiste *Toolsammlung*, sobald Sie in eines der linken Fenster klicken, verschwindet sie vorübergehend. Sie müssen nur wieder in den Dialog klicken, um sie erneut anzuzeigen.
- 4 Jeder Dialog enthält wenigstens eine Schaltfläche zum Schließen. Klicken Sie daher zuerst in der Toolsammlung auf das Symbol *Befehlsschaltfläche* und dann in den Dialog-Entwurf. Dadurch erscheint dort eine Schaltfläche mit der Beschriftung *CommandButton1*.
- 5 Wie Sie im zugehörigen Eigenschaften-Fenster links unten sehen können, sind die *Name*-Eigenschaft und die *Caption*-(Beschriftung-)Eigenschaft derzeit noch identisch. Das ist nicht sinnvoll, denn der Name sollte sich an die Ungarische Notation mit einem erläuternden Präfix halten und die Beschriftung sollte inhaltlich passen.
- 6 Ändern Sie die Name-Eigenschaft bitte auf *btnSchliessen*, da es sich um eine Schaltfläche handelt, deren englischer Name *button* als Präfix benutzt wird. Für die *Caption*-Eigenschaft tragen Sie bitte *Schließen* ein.

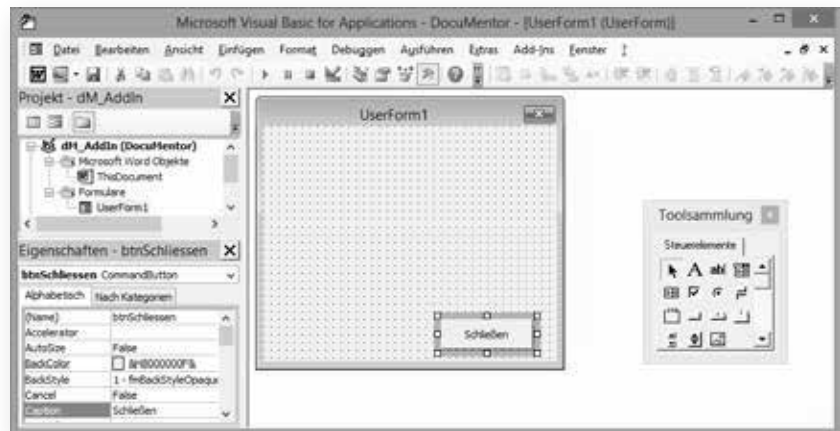


Abbildung 11.2: Die Schaltfläche wurde umbenannt und beschriftet



### Sonderzeichen vermeiden

Während in Zeichenketten (was auch die *Caption*-Eigenschaft einschließt) Umlaute und andere Sonderzeichen kein Problem sind, sollten Sie diese in Namen von Objekten oder Variablen unbedingt vermeiden. Spätestens in anderssprachigen Umgebungen müssen Sie sonst mit Problemen rechnen.

- 7 Damit die Schaltfläche später auch den Dialog schließen kann, machen Sie auf ihren derzeit schraffierten Rand einen Doppelklick. Sie kommen so nicht nur in das zugehörige Dialog-Modul, sondern erzeugen damit gleich die Ereignis-Prozedur *btnSchliessen\_Click*. Ergänzen Sie diese wie folgt:

```
Private Sub btnSchliessen_Click
    Unload Me
End Sub
```

- 8 Die *Unload*-Methode entlädt das übergebene Objekt und bei *Me* handelt es sich um eine automatisch erstellte Objekt-Variable wie beispielsweise auch *ActiveDocument*. Sie bezeichnet den Dialog, in dem sie steht, denn dieses Modul ist nicht Teil der Standardmodul-Auflistung, wie Sie vielleicht schon bemerkt haben, sondern ist im Dialog enthalten. Jeder Dialog hat eine grafische Oberfläche und ein Modul, zwischen denen Sie ein- und herschalten können. Dadurch ist *Me* in einem solchen Dialog-Modul immer eindeutig (und in Standard-Modulen deswegen auch nicht zulässig).
- 9 Wenn Sie Ihren Dialog nun testen wollen (Sie haben doch hoffentlich vorher gespeichert?!), können Sie ihn direkt mit der **[F5]**-Taste starten. Ein Mausklick auf die Schaltfläche schließt ihn dann erwartungsgemäß wieder.

Damit haben Sie Ihren ersten funktionsfähigen Dialog erzeugt, auch wenn dieser noch keine besonderen Fähigkeiten besitzt. Anschließend geht es vor allem darum, ihn mit passenden Steuerelementen auszustatten.

- 1 Da dieser Dialog später dazu dienen soll, das *docuMentor*-Logo anzuzeigen, können Sie im Entwurf noch ein Bild hinzufügen. Klicken Sie dazu in der Toolsammlung auf das *Anzeige*-Symbol und ziehen im Dialog ein Rechteck auf. Benennen Sie es als *imgLogo* und wählen in der *Picture*-Eigenschaft mit der dort erscheinenden Schaltfläche eine geeignete Bild-Datei aus:

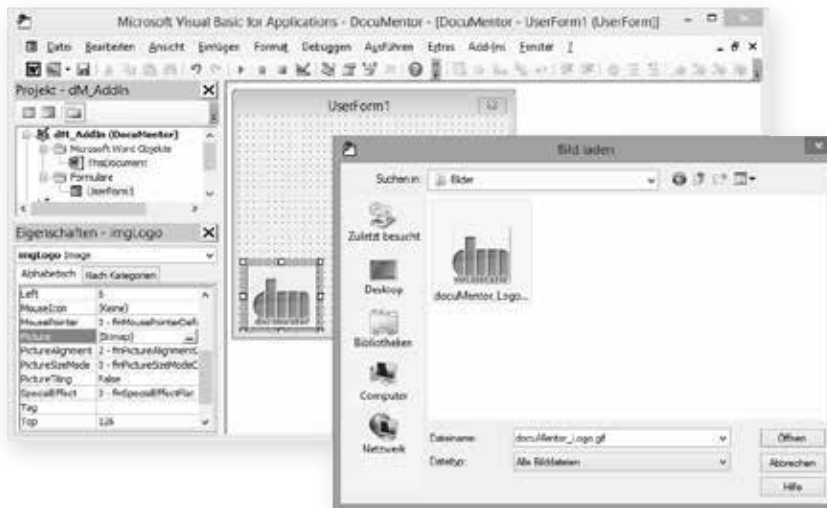


Abbildung 11.3: Die vorbereitete GIF-Datei wird geladen

Wenn Sie ein nicht rechteckiges Bild einsetzen wollen, sollte es teilweise transparent sein. Von den möglichen Bildformaten, die hier geladen werden können, erlauben nur *\*.gif*-, *\*.png*-, *\*.ico*- und *\*.wmf*-Bilder Transparenz. Diese muss bereits vorher in einem Bildbearbeitungsprogramm eingestellt worden sein.



**Welche Bilder sind geeignet?**



- 2 Je nach Größe wird das Bild nicht in die vorgesehene Anzeige-Fläche passen, daher sollten Sie entweder die Anzeige dem Bild anpassen (*AutoSize* auf *True* stellen) oder den *PictureSizeMode* auf *fmPictureSizeModeZoom* ändern, damit es verkleinert wird. Für ein Bild mit Transparenz wählen Sie *fmBackStyleTransparent* als *BackStyle* und *fmBorderStyleNone* als *BorderStyle*, sodass es nun im Entwurf so aussieht:

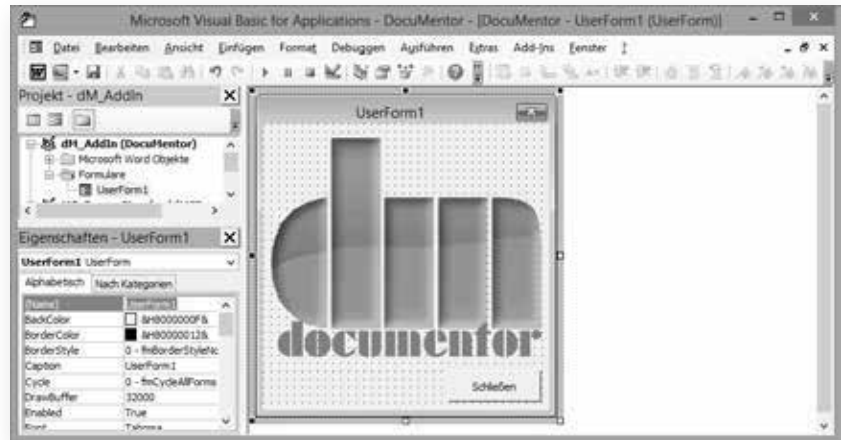


Abbildung 11.4: Das Bild passt auf den Dialog

- 3 Der Dialog selber heißt bisher noch *UserForm1* und hat eine entsprechende Beschriftung. Ebenso wie bei der Schaltfläche ändern Sie die *Name*- und die *Caption*-Eigenschaft. Klicken Sie auf den äußeren Rand des Dialog-Entwurfs und geben als Name **dlgLogo** und als Caption **Logo anzeigen** ein.
- 4 Damit ist der Dialog fertig und sieht aus wie im folgenden Bild, wenn Sie ihn mit **F5** starten:



Abbildung 11.5: Der fertige Dialog zur Laufzeit

## Daten anzeigen und eingeben

## 11.2

Ein weiterer Dialog soll auch die Dateneingabe erlauben und damit die Optionen verwalten. Alle bisher erstellten Formulare als Dokument-Vorlagen hatten ja am Anfang ein paar benutzertypische Standardwerte. Anstatt diese manuell in der Registry zu verwalten, soll der Benutzer sie über diesen Optionen-Dialog pflegen können.

- 1 Erstellen Sie im *docuMentor*-Add-In einen zweiten Dialog mit dem Menü *Einfügen/ UserForm*, der im unteren Bereich wieder ein transparentes Bild und Schaltflächen enthält. Darüber fügen Sie aus der Toolsammlung jeweils links ein *Bezeichnungsfeld* und rechts ein *Textfeld* ein. Benennen Sie diese wie im folgenden Bild:

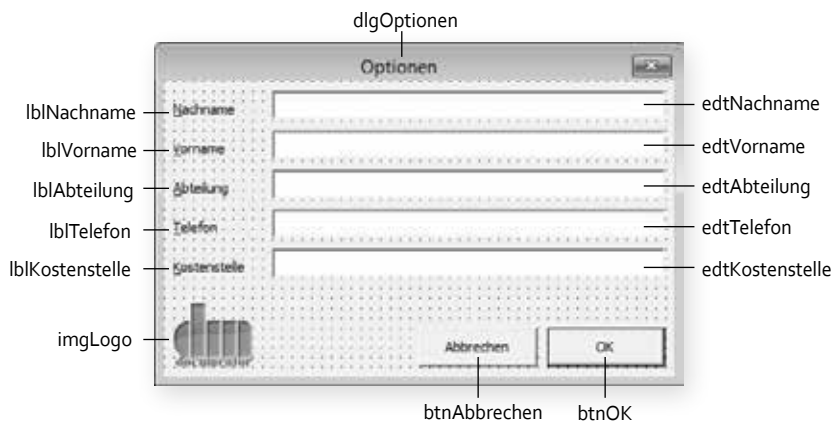


Abbildung 11.6: Der Entwurf des Optionen-Dialogs

Wie in anderen Windows-Dialogen auch lassen sich die Eingabezeilen per Tastaturkürzel anspringen. Dazu muss beim jeweiligen Bezeichnungsfeld die *Accelerator*-Eigenschaft mit dem gewünschten Buchstaben belegt werden. Der erste passende Buchstabe innerhalb der *Caption*-Eigenschaft wird dann unterstrichen (es muss aber keineswegs der erste Buchstabe des *Caption*-Textes sein). Ein Benutzer kann dann zur Laufzeit das betreffende Element aktivieren, indem er diesen Buchstaben zusammen mit der **Alt**-Taste drückt.

Damit das auch bei nachträglicher Einfügung von Steuerelementen noch gelingt, können Sie diese im Menü *Ansicht/Aktivierreihenfolge* anpassen. Dort müssen immer zuerst das *Label* und anschließend die zugehörige *Textbox* aufeinanderfolgen.



### Tastaturkürzel in Dialogen

- 2 Damit beim Öffnen des Dialogs direkt die Inhalte aus der Registry in die Textfelder geladen werden, brauchen Sie ein Ereignis, welches dies zur Laufzeit erledigt. Beim Laden jedes Dialogs wird immer *UserForm\_Initialize* ausgeführt, was daher sicherlich die am häufigsten benutzte Prozedur ist.
- 3 Machen Sie auf den Dialog-Hintergrund einen Doppelklick, sodass Sie damit automatisch in den VBA-Editor wechseln. Dort wird allerdings automatisch *UserForm\_Click* erstellt. Löschen Sie diese Prozedur, wählen aus den Listen *UserForm\_Initialize* aus und ergänzen Sie diese so:



```
Private Sub UserForm_Initialize()
    Me.edtNachname.Value = GetSetting("docuMentor", _
        "Standard\Optionen", "Nachname", "")
End Sub
```

- 4 Die Objektvariable *Me* steht dabei ja für den Dialog, in dem sich der Code befindet. Deren Unterobjekte werden ebenfalls automatisch gebildet, sodass das Textfeld *edtNachname* nun automatisch in der IntelliSense-Liste angeboten werden muss. Dessen *Value*-Eigenschaft entspricht dem angezeigten und vom Benutzer änderbaren Text. Die bereits vorgestellte *GetSetting*-Funktion liest die durch den Code aus den Dokument-Vorlagen schon gespeicherten Werte aus.
- 5 Das würde alles auch für die weiteren Textboxen funktionieren, ist aber etwas umständlich. Schreiben Sie besser in einem neuen Standard(!)-Modul *modAllgemein* zwei Property-Prozeduren, welche das Lesen und Schreiben solcher Optionen sinnvoll kapseln:

```
Public Const p_cstrName = "docuMentor"
Public Const p_cstrVersion = "1.00"
Public Const p_cstrRegStandardTexte = "Standard\Optionen "
```

```
Property Get OptionsText(strName As String) As String
    OptionsText = GetSetting(p_cstrName, p_cstrRegStandardTexte, _
        strName, "")
End Property
```

```
Property Let OptionsText(strName As String, strWert As String)
    SaveSetting p_cstrName, p_cstrRegStandardTexte, strName, strWert
End Property
```

- 6 Die Datei-öffentlichen Konstanten stellen sicher, dass es später keine Schreibfehler bei der Angabe des Registry-Zweigs gibt. Die Konstanten-Namen mögen Ihnen im Moment recht lang erscheinen, aber früher oder später werden eine Menge weitere hinzukommen, da ist diese Ausführlichkeit viel übersichtlicher.
- 7 Mit diesen Properties können Sie den Code in *UserForm\_Initialize* viel eleganter formulieren:

```
Private Sub UserForm_Initialize()
    Me.edtNachname.Value = OptionsText("Nachname")
    Me.edtVorname.Value = OptionsText("Vorname")
    Me.edtAbteilung.Value = OptionsText("Abteilung")
    Me.edtTelefon.Value = OptionsText("Telefon")
    Me.edtKostenstelle.Value = OptionsText("Kostenstelle")
End Sub
```

- 8 Wenn Sie den Dialog nun mit **F5** starten, enthalten alle Textboxen bereits Registry-Daten, weil diese im letzten Kapitel bereits mit Werten versehen wurden:

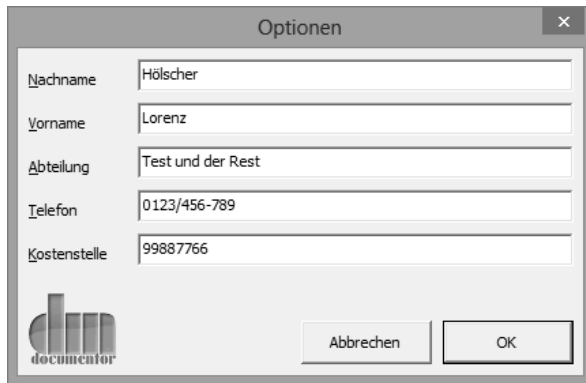


Abbildung 11.7: Der Optionen-Dialog ist vorläufig fertig

- 9 Jetzt müssen Sie nur noch sicherstellen, dass die Schaltflächen funktionieren. Die *Abbrechen*-Schaltfläche erhält den eben schon gesehenen *Unload Me*-Code und für das Klick-Ereignis der *OK*-Schaltfläche werden die Zuweisungen umgedreht, sodass nun automatisch die *Let*-Property zum Einsatz kommt:

```
Private Sub btnAbbrechen_Click()
    Unload Me
End Sub
```

```
Private Sub btnOK_Click()
    OptionsText("Nachname") = Me.edtNachname.Value
    OptionsText("Vorname") = Me.edtVorname.Value
    OptionsText("Abteilung") = Me.edtAbteilung.Value
    OptionsText("Telefon") = Me.edtTelefon.Value
    OptionsText("Kostenstelle") = Me.edtKostenstelle.Value
    Unload Me
End Sub
```

Damit enthält auch der zweite Dialog eigentlich schon alle Elemente eines typischen benutzerdefinierten Dialogs, um Daten zu verändern. Wie Sie sehen, ist der Aufwand recht gering. Daher soll der Dialog noch um ein paar Fähigkeiten erweitert werden.

## Weitere Dialogelemente einfügen

## 11.3

Zusätzlich zu den bisherigen Optionen für Standardwerte in Formular-Dokument-Vorlagen soll es möglich sein, die Sprache zu wechseln, in welcher der Dialog beschriftet ist. Außerdem werden früher oder später Hilfsdateien (für nachzuladende Bilder oder ähnliches) benötigt, deren Pfad benutzerseitig flexibel angegeben werden darf.



- 1 Klicken Sie auf den Hintergrund des Dialog-Entwurfs und vergrößern seine Fläche am unteren mittleren Anfassers nach unten. Markieren Sie dann mit dem Menü *Bearbeiten/Alles markieren* (oder einer mit der Maus über alle Elemente gezogenen Markierung) die bisher enthaltenen Kontroll-Elemente und ziehen diese an den unteren Rand.
- 2 Fügen Sie oberhalb der bisherigen Elemente ein Bezeichnungsfeld *lblSprache* mit einem Kombinationsfeld *cmbSprache* ein. Darunter folgt ein weiteres Bezeichnungsfeld *lblPfadDaten* mit einer Textbox *edtPfadDaten* sowie daneben einer Schaltfläche *btnPfadDaten*.
- 3 Diese Schaltfläche erhält wegen leerer *Caption*-Eigenschaft keine Beschriftung, sondern über die *Picture*-Eigenschaft ein Bild. Das ist zwar sehr viel kompakter, aber oft nicht selbsterklärend. Daher sollten Sie dann immer auch die *ControlTipText*-Eigenschaft mit einem erläuternden Text wie »Pfad auswählen...« versehen, der später automatisch als QuickInfo erscheint.
- 4 Da in *edtPfadDaten* keine manuelle Dateneingabe möglich sein soll, wird es mit *True* für die *Locked*-Eigenschaft gegen Änderungen gesperrt und die *BackStyle*-Eigenschaft auf *fmBackStyleTransparent* gestellt. Der Dialog-Entwurf sieht nun aus wie im folgenden Bild:

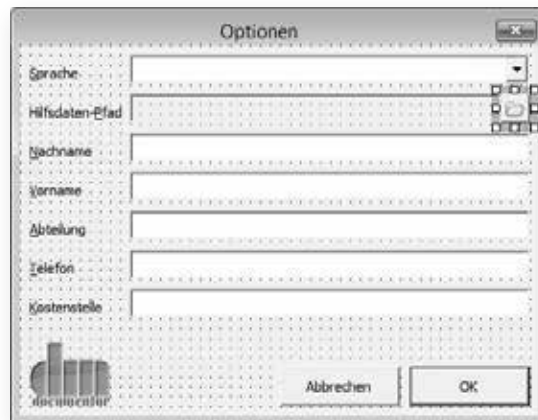


Abbildung 11.8: Die zusätzlichen Kontroll-Elemente auf dem Optionen-Dialog

- 5 Damit auch diese beiden neuen Inhalte aus der Registry als Optionen geladen werden, müssen Sie die *UserForm\_Initialize*-Prozedur wie folgt erweitern:

```
Private Sub UserForm_Initialize()
    'wie bisher
    Me.edtPfadDaten.Value = OptionsText("PfadHilfsdaten")
    With Me.cmbSprache
        .AddItem "Deutsch"
        .AddItem "English"
        .AddItem "Français"
        .ListIndex = Val(OptionsText("Sprache"))
    End With
End Sub
```





Mehr Platz für Texte

Neu eingefügte Textboxen haben typischerweise die *SelectionMargin*-Eigenschaft auf *True* stehen. Das führt dazu, dass der enthaltene Text nicht direkt links beginnt. Wenn Sie diese Eigenschaft auf *False* stellen, verzichten Textboxen und Kombinationsfelder Windows-üblich auf den linken Abstand.

- 6 Wie schon in den Formular-Dokumentvorlagen kann ein Kombinationsfeld (engl. combobox) keine Inhalte im Entwurf speichern, sodass es beim Laden des Dialogs jedes Mal neue Zeilen erhalten muss. Diese werden mit der *AddItem*-Methode hinzugefügt. Anschließend wird mit *ListIndex* die zuletzt ausgewählte Zeile auch direkt wieder markiert. Beim ersten Aufruf ist der Registry-Eintrag zwar leer, aber dank der *Val*-Funktion wird dann eine 0 zurückgegeben, was dem Index der ersten Zeile entspricht.
- 7 Beim Speichern werden diese beiden Werte durch die folgende Ergänzung ebenfalls berücksichtigt:

```
Private Sub btnOK_Click()
    'wie bisher
    OptionsText("Sprache") = Me.cmbSprache.ListIndex
    OptionsText("PfadHilfsdaten") = Me.edtPfadDaten.Value
    Unload Me
End Sub
```

- 8 Um mittels Klick auf die *edtPfadDaten*-Schaltfläche ein Verzeichnis übernehmen zu können, nutzen Sie einfach die bereits vorgestellten integrierten Dialoge. Da der *DateiÖffnen*-Dialog jedoch nur dann eine Auswahl erlaubt, wenn der Benutzer wirklich eine der angezeigten Dateien markiert, könnte er damit kein leeres Verzeichnis auswählen. Daher ruft die *btnPfadDaten\_Click*-Prozedur stattdessen einen *DateiKopieren*-Dialog auf, der auch leere Verzeichnisse akzeptiert:

```
Private Sub btnPfadDaten_Click()
    With Dialogs(wdDialogCopyFile)
        If .Display() Then
            Me.edtPfadDaten.Value = .Directory
        End If
    End With
End Sub
```

Damit ist dieser benutzerdefinierte Dialog voll funktionsfähig und sieht nach Start mit

F5 so aus:



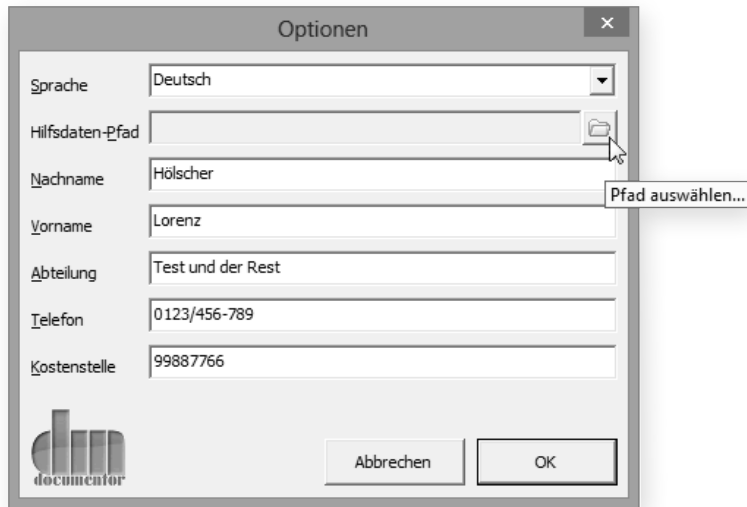



Abbildung 11.9: Der Optionen-Dialog zur Laufzeit

Sie werden beim Testen des Dialogs feststellen, dass die Reihenfolge der Eingabefelder wieder nicht stimmt, wenn Sie mit der -Taste wechseln. Das hängt mit der Aktivierreihenfolge zusammen, die normalerweise in der Reihenfolge des Einfügens erfolgt. Rufen Sie zur Korrektur im Entwurf *Ansicht/Aktivierreihenfolge* auf. Dort können Sie ein oder mehrere Objekte markieren und mit den Schaltflächen nach oben oder unten verschieben:

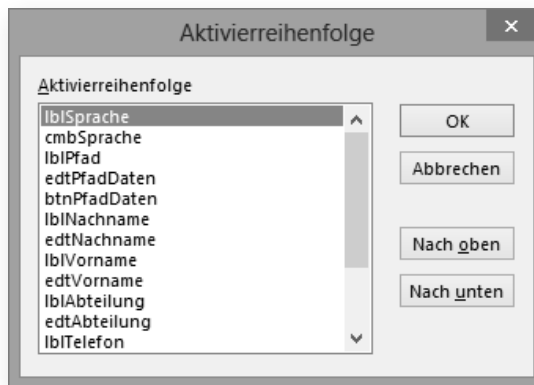


Abbildung 11.10: So passen Sie die Aktivierreihenfolge an

## 11.4 Dialoge dynamisch übersetzen

Die mögliche Auswahl einer Sprache hat schon darauf hingewiesen, dass das *docuMentor*-Projekt mehrsprachig wird. Dabei werden allerdings nicht die Dokumentvorlagen dynamisch übersetzt, weil es viel effektiver ist, direkt für jede Sprache eine eigene Datei bereitzustellen. Aber alle Bedienelemente auf den Dialogen werden zur Laufzeit umschaltbar sein, damit in verschiedenen Sprachen immer der gleiche Code läuft und Sie nicht mehrere Versionen pflegen müssen.

Bei einem echten mehrsprachigen Projekt kämen die Übersetzungen selbstverständlich aus einer Datenbank, was hier jedoch umfangreiche Erläuterungen zum Zugriff auf Datenbank-Objekte benötigen und den Rahmen sprengen dürfte.

In diesem Projekt werden die Übersetzungsdaten in der Registry gespeichert, sodass sich deren Inhalte bei Bedarf einfach mit einer \*.reg-Datei exportieren und importieren lassen. Wirklich implementiert sind hier auch nur Deutsch und Englisch. Wie Sie aber sehen werden, sind die Sprachen in der Registry einfach durch einen senkrechten Strich (das so genannte *pipe*-Zeichen: »|«) getrennt und dadurch beliebig erweiterbar:



Abbildung 11.11: So werden die gespeicherten Daten in der Registry aussehen

Der Zusatz »\_CTP« kennzeichnet dabei den zugehörigen *ControlTipText*, da für eine Übersetzung ja nur die Beschriftungen und QuickInfos (*ControlTipText*) angepasst werden müssen.

- 1 Ergänzen Sie zuerst das Modul *modAllgemein* um je eine Datei-öffentliche Konstante (für den Zweig in der Registry) und Variable (für die aktuelle Sprachauswahl):

```
Public Const p_cstrRegDialoge = "Dialoge"
Public p_intSprache As Integer
```

- 2 Außerdem erstellen Sie bitte ein neues Modul namens *modFuerDialoge*, in welchem verschiedene Hilfsfunktionen für Dialoge ausgelagert sein werden. Darin braucht es eine Funktion *Uebersetzen*, welche zu einem Dialognamen und einem Kontroll-Elementnamen den Übersetzungstext ermittelt:

```
Function Uebersetzen(strDialogName As String, strControlName As String, _
    strDeutsch As String) As String

    Dim varWert As Variant

    varWert = GetSetting(p_cstrName, p_cstrRegDialoge & "\" & strDialogName, _
        strControlName)
    If varWert = "" Then 'Eintrag also noch leer
        SaveSetting p_cstrName, p_cstrRegDialoge & "\" & strDialogName, _
            strControlName, strDeutsch & "|"
        Uebersetzen = strDeutsch
    Else
        varWert = Split(varWert, "|")
```



```

If p_intSprache > UBound(varWert) Then
    Uebersetzen = "(" & varWert(0) & ")"
Else
    Uebersetzen = varWert(p_intSprache)
    If Uebersetzen = "" Then 'falls andere Sprache noch fehlt
        If varWert(0) = "" Then
            Uebersetzen = ""
        Else
            Uebersetzen = "(" & varWert(0) & ")"
        End If
    End If
End If
End If
End Function

```

- 3 Zuerst liest die Funktion in *varWert* den kompletten Inhalt der Registry-Zeile ein. Ist dieser leer, gab es noch keine Übersetzung, also wird der Standardwert aus dem Dialog-Entwurf im Parameter *strDeutsch* in die Registry geschrieben und mit anschließendem Trennstrich versehen. Diese Daten können in der Registry dann später noch manuell übersetzt und eingetragen werden.
- 4 War ein Wert enthalten, prüft die Funktion vorsichtshalber, ob die Sprachnummer *p\_intSprache* eventuell größer ist als die zur Verfügung stehende Auswahl an Übersetzungen in dieser Zeile. Das wäre hier der Fall, wenn jemand Französisch auswählt, für das ja noch keine Übersetzungen enthalten sind.
- 5 Erst danach kann die Funktion sicher auf das passende Element im *varWert*-Array zugreifen. Bei unvollständigen Übersetzungen ist das Element zwar vorhanden, aber leer. Dann wird der deutsche Text in Klammern zurückgegeben, damit Sie wissen, welche Übersetzung noch fehlt.
- 6 Mit der *Uebersetzen*-Funktion lässt sich nun in einer *For Each/Next*-Schleife einfach jedes Kontroll-Element auf einem beliebigen Dialog übersetzen. Tatsächlich müssen Sie sogar nicht einmal jedes Kontroll-Element beachten, denn übersetzbare Texte stehen im Wesentlichen auf Bezeichnungsfeldern (label), Schaltflächen (command button) und Umschaltfeldern (toggle button):

```

Sub DialogUebersetzen(dlgDieser As Object)
    Dim ctIDieses As Control

    For Each ctIDieses In dlgDieser.Controls
        Select Case TypeName(ctIDieses)
            ,Groß-/Kleinschreibung beachten!
        Case "Label", "CommandButton", "ToggleButton"
            ctIDieses.Caption = Uebersetzen(dlgDieser.Name, ctIDieses.Name, _

```

```

        ctIDieses.Caption)
    ctIDieses.ControlTipText = Uebersetzen(dlgDieser.Name, _
        ctIDieses.Name & "_CTP", ctIDieses.ControlTipText)
End Select
Next
dlgDieser.Caption = Uebersetzen(dlgDieser.Name, Chr(0), _
    dlgDieser.Caption)
End Sub

```

### UserForm-Datentyp ist hier problematisch

Sicherlich haben Sie bemerkt, dass in der *DialogUebersetzen*-Funktion für den Parameter *dlgDieser* nicht der korrekte Datentyp `As UserForm`, sondern nur das allgemeine `As Object` vereinbart wurde. Die Deklaration `As UserForm` würde jedoch einen Laufzeitfehler für *dlgDieser.Name* erzeugen mit der (falschen!) Meldung, das *UserForm*-Objekt unterstütze die *Name*-Eigenschaft nicht. Für einen *Object*-Datentyp prüft der Compiler das einfach nicht und zur Laufzeit funktioniert es eben doch.



**UserForm-Datentyp ist hier problematisch**

- 7 Da der Dialog selber kein Kontroll-Element ist, muss seine *Caption*-Eigenschaft am Ende extra behandelt werden. Für diesen Eintrag wird der »(Standard)«-Wert im jeweiligen Registry-Zweig benutzt, der mit einem ASCII-o-Zeichen ausgewählt wird. Da sich das weder im VBA-Code noch überhaupt auf der Tastatur eingeben lässt, geben Sie das Zeichen mit der *Chr*-Funktion an.
- 8 Nun sind zwar alle Fähigkeiten zum Übersetzen programmiert, aber noch nirgends konkret eingesetzt. Es gibt zwei Anlässe, wann der Optionen-Dialog übersetzt werden muss: beim Laden und (nur, weil hier die Sprache geändert werden kann) bei der Sprachauswahl. Daher müssen Sie erstens in *UserForm\_Initialize* nachbessern und zweitens das *Change*-Ereignis beim Ändern eines Kombinationsfeld-Eintrags berücksichtigen:

```

Private Sub cmbSprache_Change()
    p_intSprache = Me.cmbSprache.ListIndex
    DialogUebersetzen Me
End Sub

Private Sub UserForm_Initialize()
    'wie bisher
    .ListIndex = Val(OptionsText("Sprache"))
    p_intSprache = Me.cmbSprache.ListIndex
End With
DialogUebersetzen Me
End Sub

```



- 9 Mit diesen Anpassungen lässt sich dieser und jeder andere Dialog zur Laufzeit einfach übersetzen:



Abbildung 11.12: Der Optionen-Dialog kann nun zur Laufzeit übersetzt werden

## 11.5 Weitere Kontrollelemente

Der nächste Beispiel-Dialog enthält noch mehr interaktive Elemente als die bisher bearbeiteten Dialoge. Inhaltlich sollen damit die Kopf- und Fußzeile sowie eine Falzmarke unserer Dokument-Vorlagen bearbeitet werden.

Damit nicht in jedem Dialog-Entwurf das Logo tatsächlich enthalten sein muss, brauchen wir zur Vorbereitung noch einen eigenen Dialog, der lediglich Speicher für solche Grafiken ist.

- 1 Fügen Sie einen Dialog namens *dlgBilder* ein, der wie im folgenden Bild lediglich zwei Anzeige-Elemente enthält: oben das *imgLogo* mit dem *docuMentor*-Logo und unten *imgFortschritt* mit einer beliebigen Grafik als zukünftiger Fortschrittsbalken:

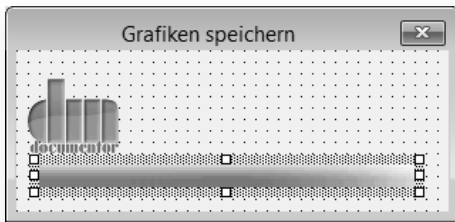


Abbildung 11.13: Dieser Dialog sammelt nur Grafiken

- 2 Der Vorteil dieses speziellen Dialogs zum Speichern von Grafiken ist, dass Sie nie überprüfen müssen, ob die Grafik-Datei vorhanden ist, in welchem Pfad sie gespeichert ist, ob Sie Leserechte haben und Ähnliches. Da die Grafiken Teil des Add-Ins sind, können Sie sich immer darauf verlassen, dass sie zur Verfügung stehen. Trotzdem lassen sie sich hier zentral pflegen, wenn es mal inhaltliche Änderungen gibt.

Alle anderen Dialoge können bei Bedarf auf diese Inhalte zugreifen. Daher enthält dieser Dialog auch keine VBA-Programmierung, weil diese (beispielsweise *UserForm\_Initialize*) ansonsten ausgeführt würde. Jetzt können wir uns um den eigentlich geplanten Dialog zur Änderung von Kopf- und Fußzeilen kümmern.

- 1 Fügen Sie im Add-In einen neuen Dialog *dlgKopfFusszeile* ein, der die im folgenden Bild gezeigten Kontroll-Elemente mit den passenden Namen enthält. Die mit »frm...« benannten Elemente sind Rahmen (engl. frame), die Sie vor(!) deren Inhalten zeichnen müssen.

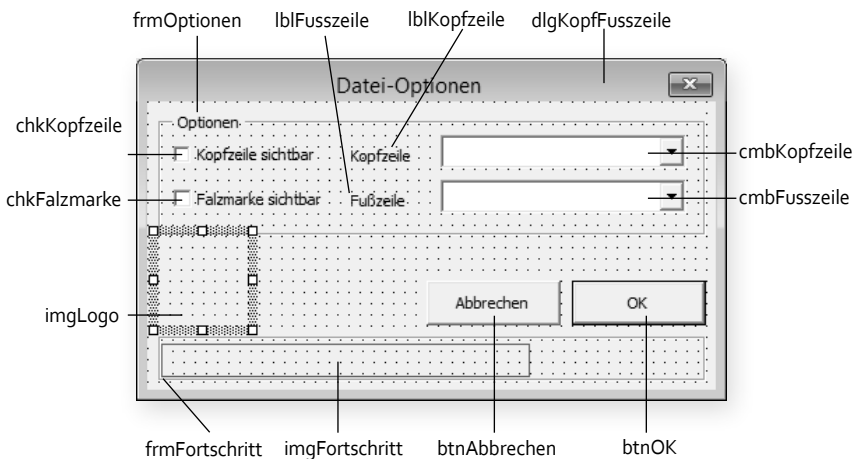


Abbildung 11.14: Der Kopf-/Fußzeilen-Dialog mit seinen Kontroll-Elementen

- 2 Anders als bisher gibt es zwar ein *Anzeige*-Element, aber dessen *Picture*-Eigenschaft wird ja nicht zur Entwurfszeit gefüllt. Jetzt müssen Sie nur noch dafür sorgen, dass die Grafiken auch von *dlgBilder* zu *dlgKopfFusszeile* kommen. Das ist extrem einfach, weil Sie in *dlgKopfFusszeile* nur deren *Picture*-Eigenschaft übergeben müssen:

```

Private Sub UserForm_Initialize()
    Me.imgLogo.Picture = dlgBilder.imgLogo.Picture
    Me.imgFortschritt.Picture = dlgBilder.imgFortschritt.Picture
End Sub

```

- 3 Damit zeigt der Kopf-/Fußzeilen-Dialog zur Laufzeit ohne Probleme bereits alle Grafiken an:

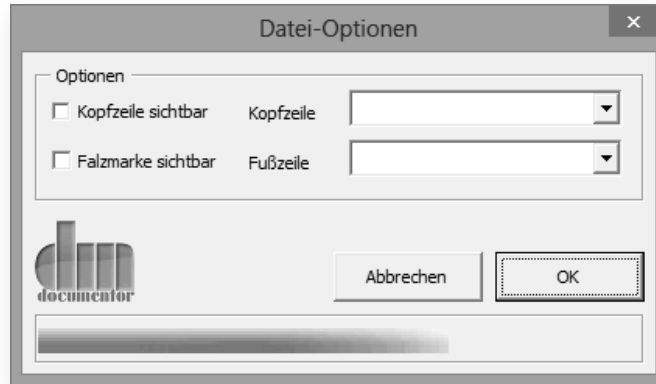


Abbildung 11.15: Der Kopf-/Fußzeilen-Dialog zeigt die benötigten Grafiken

- 4 Als Nächstes müssen die Kombinationsfelder mit Inhalten gefüllt und je nach Vorhandensein von Textmarken Teile des Dialogs deaktiviert werden. Die Textmarken enthalten die bereits benutzten Präfixe »dM\_«, um sie von anderen zufällig vorhandenen Textmarken unterscheiden zu können. Erweitern Sie die *UserForm\_Initialize*-Prozedur um den folgenden Code:

```

Private Sub UserForm_Initialize()
    Me.imgLogo.Picture = dlgBilder.imgLogo.Picture
    Me.imgFortschritt.Picture = dlgBilder.imgFortschritt.Picture
    Me.chkFalzmarke.Enabled = _
        ActiveDocument.Bookmarks.Exists("dM_Falzmarke")
    FueilleCombo Me.cmbKopfzeile, "kopf"
    If Not ActiveDocument.Bookmarks.Exists("dM_Kopfzeile") Then
        Me.chkKopfzeile.Enabled = False
        Me.lblKopfzeile.Enabled = False
        Me.cmbKopfzeile.Enabled = False
        Me.cmbKopfzeile.BackStyle = fmBackStyleTransparent
    End If
    FueilleCombo Me.cmbFusszeile, "fuss"
    If Not ActiveDocument.Bookmarks.Exists("dM_Fusszeile") Then
        Me.lblFusszeile.Enabled = False
        Me.cmbFusszeile.Enabled = False
        Me.cmbFusszeile.BackStyle = fmBackStyleTransparent
    End If
End Sub

```



- 5 Mit der *Enabled*-Eigenschaft steuert der Code, ob das jeweilige Kontroll-Element zur Laufzeit benutzbar ist. Für den Funktionstest müssen also die drei Textmarken in der Dokument-Vorlage *Allgemeiner Brief.dotm* eingefügt werden. Die *dM\_Kopfzeile* ist bereits vorhanden, *dM\_Fusszeile* wird eine offene Textmarke in der Fußzeile und *dM\_Falzmarke* umfasst ein Minuszeichen in einer Textbox am linken Rand.
- 6 Die *FuelleCombo*-Hilfsprozedur liest aus dem Pfad für die Hilfsdaten alle Dateien mit einem passenden Zusatz, für Fußzeilen also alle *\*\_fuss.\**-Dateien. Trotz des Variablen-Namens handelt es sich lediglich um das Ende des eigentlichen Dateinamens. Dadurch bleibt die Datei-Endung selber erhalten und die Dateien können weiterhin mit ihrem jeweiligen Originalprogramm bearbeitet werden.

```
Sub FuelleCombo(cmbDiese As ComboBox, strEndung As String)
    Dim strAktDatei As String

    With cmbDiese
        strAktDatei = Dir(OptionsText("PfadHilfsdaten") & "*_" & _
            strEndung & ".*")
        Do Until strAktDatei = ""
            .AddItem strAktDatei
            strAktDatei = Dir()
        Loop
        .AddItem "(wie Dokument)", 0
        .AddItem "(löschen)", 1
        .ListIndex = 0
    End With
End Sub
```

- 7 Beim Testen müssen Sie darauf achten, dass in der Registry im genannten Wert *PfadHilfsdaten* tatsächlich ein gültiger Pfad steht und sich in diesem auch Dateien befinden, die passend zum Suchmuster benannt sind.

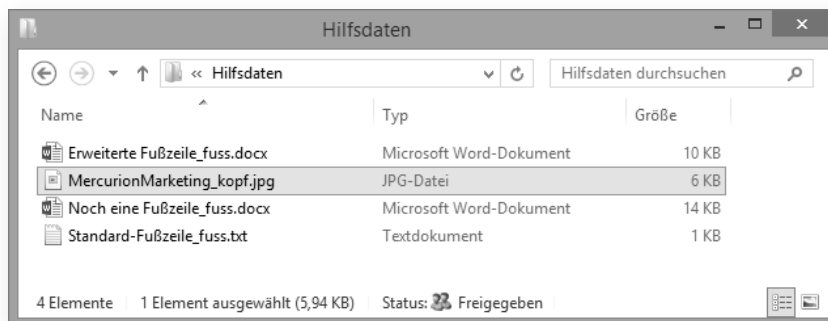


Abbildung 11.16: Das Verzeichnis enthält passende Dateien

- 8 Zuerst werden alle im Verzeichnis gefundenen Dateien mit dem Suchmuster als Zeilen für das Kombinationsfeld hinzugefügt und anschließend die Standard-Auswahlen »(wie Dokument)« beziehungsweise »(löschen)«. Der optionale zweite Parameter



von *AddItem* ermöglicht es, gezielt einen neuen Wert in einer beliebigen Zeile anzufügen. Je nach vorhandenen Dateien sieht der Dialog zur Laufzeit nun so aus:

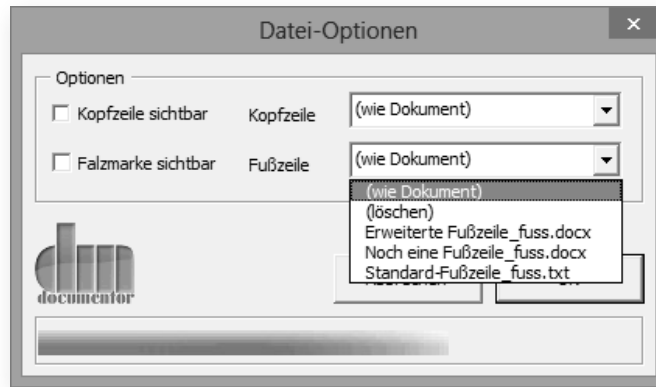


Abbildung 11.17: Der Dialog zeigt bestimmte Dateien des Verzeichnisses

Im Gegensatz zur Fußzeile kann die Kopfzeile vorübergehend unsichtbar gemacht werden, wenn abwechselnd auf Papier mit und ohne Logo gedruckt wird. Das bedeutet für den Dialog, dass das Kombinationsfeld *cmbKopfzeile* auch nur bei aktivierter Option *Kopfzeile sichtbar* aktiv sein darf.

- 1 Mit einem Doppelklick auf dieses Kontrollkästchen im Entwurfsmodus kommen Sie in die benötigte Ereignis-Prozedur:

```
Private Sub chkKopfzeile_Click()
    With Me.cmbKopfzeile
        If Me.chkKopfzeile.Value Then
            .Enabled = True
            .BackStyle = fmBackStyleOpaque
            Me.lblKopfzeile.Enabled = True
        Else
            .Enabled = False
            .BackStyle = fmBackStyleTransparent
            Me.lblKopfzeile.Enabled = False
        End If
    End With
End Sub
```

- 2 Eigentlich sollte es reichen, nur die *Enabled*-Eigenschaft für ein Kombinationsfeld umzustellen. Leider ändert sich dann optisch nichts daran, sodass ein Benutzer den Wechsel nicht bemerken würde.
- 3 Daher wird hier zusätzlich die *BackStyle*-Eigenschaft auf durchsichtig (*fmBackStyleTransparent*) oder undurchsichtig (*fmBackStyleOpaque*) geändert. Außerdem muss das jeweilige Bezeichnungsfeld einzeln deaktiviert werden, weil es technisch unabhängig vom Kombinationsfeld ist.

## Fortschrittsanzeige

## 11.6

Je umfangreicher Ihre VBA-Programmierung wird, desto mehr müssen Sie sich um die Wartezeit kümmern, die den Benutzer verärgern könnte. Oberstes Ziel ist es selbstverständlich, schnelle Prozeduren zu schreiben. Falls es sich aber nicht mehr beschleunigen lässt, muss der Benutzer wenigstens eine Rückmeldung darüber erhalten, wie weit das Makro ist oder ob es überhaupt noch arbeitet.

Wenn Sie sowieso schon einen (grafisch orientierten) Dialog auf dem Bildschirm haben, können Sie den Fortschritt darin mit einem klassischen Fortschrittsbalken anzeigen. Vom Einsatz der Statuszeile (via `Application.StatusBar`) rate ich Ihnen dringend ab, da Ihr Code dadurch extrem verlangsamt werden kann!

Der Dialog enthält schon die vorbereitende Grafik für die Fortschrittsanzeige. Das ist ein im Grunde völlig beliebiges Bild, dessen Breite kontinuierlich vergrößert wird. Da das geplante Einlesen von Bildern für die Kopfzeile oder Text-Dateien für die Fußzeile in Word eher länger dauern wird, soll der Benutzer so eine Rückmeldung erhalten, wie weit die Arbeit fortgeschritten ist.

Das Problem der Laufzeiten könnte auf mehreren Dialogen auftauchen, daher werden wir hier wie schon bei der Übersetzung direkt eine allgemeine Lösung erstellen. Sonst müssen Sie das jedes Mal wieder programmieren oder benötigen doppelten Code.

- 1 Wechseln Sie zuerst in das Modul *modFuerDialoge*, weil hier ja die Hilfsprozeduren stehen, die auf mehreren Dialogen eingesetzt werden. Fügen Sie dort diese neue Prozedur *FortschrittStart* ein:

```
Sub FortschrittStart(dlgDieser As Object)
    With dlgDieser
        .imgFortschritt.Picture = dlgBilder.imgFortschritt.Picture
        .imgFortschritt.Width = 0
        .Height = .frmFortschritt.Top - 2 + 30
    End With
End Sub
```

- 2 Mit dieser Prozedur werden erst einmal alle Startwerte eingestellt, also das Logo geladen (die gleichnamige Zeile in *UserForm\_Initialize* kann damit entfallen!) und die Breite der Fortschritts-Grafik auf 0 gestellt.
- 3 Da der ganze Fortschrittsbalken erst mit Klick auf *OK* sichtbar werden soll, wird die Höhe des Dialogs auf knapp über der *frmFortschritt*-Oberkante eingestellt. Die Überschrift des Dialogs wird bei *dlgDieser.Height* jedoch mitgerechnet, während der *dlgDieser.frmFortschritt.Top*-Wert diese ignoriert, daher müssen Sie je nach Betriebssystem noch ein paar Pixel hinzurechnen (Windows 8: 30 Pixel, Vista: 20 Pixel, Windows XP 26 Pixel).



- 4 Sie können in diesem Modul auch schon eine zweite Prozedur für die eigentliche Fortschrittsanzeige bereitstellen:

```
Sub FortschrittWeiter(dlgDieser As Object, intGesamt As Integer, _
    Optional booStart As Boolean)

    Static intAktuell As Integer

    With dlgDieser
        If booStart Then
            intAktuell = 1
            .Height = .frmFortschritt.Top + .frmFortschritt.Height + 2 + 30
        End If
        .imgFortschritt.Width = .frmFortschritt.Width * _
            intAktuell / intGesamt
        .frmFortschritt.Repaint
    End With
    intAktuell = intAktuell + 1
End Sub
```

- 5 Für die neue Höhe des Dialogs beim Anzeigen des Fortschrittsbalkens muss sowohl die *Top*-Position als auch die Höhe des Rahmens berücksichtigt werden. Dazu kommt wieder die Höhe der *Caption*-Zeile des Dialogs und ein kleiner Extra-Abstand von 2 Pixeln.
- 6 Damit beim Aufruf von *FortschrittWeiter* nicht jedes Mal der laufende Zähler als Parameter übergeben werden muss, ist hier eine *Static*-Variable eingesetzt, die ihren Wert ja über das Ende der Prozedur hinaus behält. Dafür müssen Sie umgekehrt angeben können, wann sie wieder zurückgesetzt werden muss, wofür der optionale *booStart*-Parameter dient.
- 7 Die Prozedur ist flexibel und unabhängig von der tatsächlichen Breite der Fortschrittsanzeige auf dem jeweiligen Dialog, denn sie rechnet nur den prozentualen Anteil des Fortschritts aus. Als Maßstab für die anzustrebende Gesamtbreite gilt einfach der umgebende Rahmen *frmFortschritt*.
- 8 Der Hauptgrund für den Rahmen ist aber die *Repaint*-Zeile. Sie ist nötig, weil sonst erst am Ende der aufrufenden Prozedur (hier also von *UserForm\_Initialize!*) überhaupt eine Bildschirmaktualisierung stattfindet. Ihr Benutzer sähe also den Fortschrittsbalken nie größer werden, sondern nur einmal kurz am Ende bei 100 %. Die *Repaint*-Methode erzwingt dafür ein Neuzeichnen des Bereichs.



**Nicht zu viel neu zeichnen lassen!**

Der Dialog selber hätte mit *Me.Repaint* auch eine Methode zum Neuzeichnen, aber das ergibt ein ganz furchtbares Geflacker, weil bei schnellen Schleifen zu große Bereiche betroffen sind. Einzelne Kontroll-Elemente hingegen besitzen keine *Repaint*-Methode, lassen sich also gar nicht einzeln aktualisieren. Der sinnvolle Kompromiss ist ein solcher Rahmen, der eine *Repaint*-Methode besitzt und alle in ihm enthaltenen Objekte neu zeichnet.

- 9 Um das anfängliche Zurücksetzen des Fortschrittsbalkens beziehungsweise der Höhe des Dialogs zu testen, müssen Sie nur in der `UserForm_Initialize`-Prozedur eine Zeile ergänzen:

```
Private Sub UserForm_Initialize()
    'wie bisher
    FortschrittStart Me
End Sub
```

- 10 Der Kopf-/Fußzeilen-Dialog präsentiert sich damit wie geplant ohne sichtbaren Fortschrittsbalken:

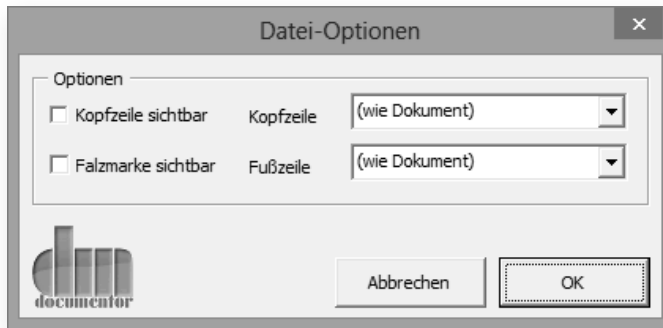


Abbildung 11.18: Der Fortschrittsbalken ist ausgeblendet

Damit wir erst einmal nur die Funktionsfähigkeit des Fortschrittsbalkens testen können, soll noch nicht die endgültige Funktionalität eingebaut werden.

- 1 Eine kleine Schleife wird beim Klick auf die `OK`-Schaltfläche hier einfach nur tausendmal die `FortschrittWeiter`-Prozedur aufrufen:

```
Private Sub btnOK_Click()
    Dim intZaehler As Integer

    FortschrittWeiter Me, 1000, True
    For intZaehler = 1 To 1000
        FortschrittWeiter Me, 1000, False
    Next
End Sub
```

- 2 Wenn Sie zur Laufzeit auf die `OK`-Schaltfläche klicken, sollten Sie den Fortschrittsbalken jetzt zügig größer werden sehen. Bei 100% stoppt er im weiterhin geöffneten Dialog, denn es gibt derzeit ja kein `Unload Me` im Code.

Sie fürchten, das aufwändige Neuzeichnen solch einer Grafik sei zu langsam? Dann vergleichen Sie es doch mal mit der Anzeige in der Statuszeile: Ersetzen Sie `FortschrittWeiter Me, 1000, False` innerhalb der Schleife durch `Application.StatusBar = intZaehler`. Das braucht seit Word 2007 ein Vielfaches an Zeit und flackert außerdem noch ganz entsetzlich.



**Grafik ist schneller als Statuszeile**



- 3 Die *OK*-Schaltfläche soll natürlich eine sinnvollere Funktion erhalten, als lediglich bis 1000 zu zählen. Schreiben Sie zuerst eine Hilfsprozedur im Modul *modFuerDialoge* wie folgt:

```
Sub DateiEinfuegen(strName As String, booGrafik As Boolean, _
    strDatei As String)

    Dim rngHier As Range

    With ActiveDocument.Bookmarks
        Set rngHier = .Item(strName).Range
        rngHier.Text = " " 'Achtung: Leerzeichen in Anführungszeichen!
        If strDatei <> "" Then
            .Add strName, rngHier
            If booGrafik Then
                rngHier.InlineShapes.AddPicture strDatei, , , rngHier
            Else
                rngHier.InsertFile strDatei
            End If
        End If
        .Add strName, rngHier
    End With
End Sub
```

- 4 Die Prozedur schreibt den gewünschten Inhalt in die für *strName* genannte Textmarke. Falls der Dateiname leer ist, soll der Inhalt einfach nur gelöscht werden. Andernfalls gibt der *booGrafik*-Parameter vor, ob eine Grafik oder eine Textdatei importiert werden soll, sodass sie entweder der *InlineShapes*-Auflistung oder direkt dem Inhalt hinzugefügt wird.
- 5 Diese *DateiEinfuegen*-Prozedur rufen Sie in *btnOK\_Click* des Dialogs mit den passenden Parametern auf. Ersetzen Sie den kompletten bisherigen Code darin, der ja nur zu Testzwecken diente:

```
Private Sub btnOK_Click()
    FortschrittWeiter Me, 3, True
    Select Case Me.cmbKopfzeile.ListIndex 'Kopfzeile importieren
        Case 0: 'wie Dokument, also nichts tun
        Case 1: DateiEinfuegen "dM_Kopfzeile", True, ""
        Case Else: DateiEinfuegen "dM_Kopfzeile", True, _
            OptionsText("PfadHilfsdaten") & Me.cmbKopfzeile.Value
    End Select
```

```

FortschrittWeiter Me, 3, False
Select Case Me.cmbFusszeile.ListIndex  'Fußzeile importieren
Case 0: 'wie Dokument, also nichts tun
Case 1: DateiEinfuegen "dM_Fusszeile", False, ""
Case Else: DateiEinfuegen "dM_Fusszeile", False, _
    OptionsText("PfadHilfsdaten") & Me.cmbFusszeile.Value
End Select

```

```

FortschrittWeiter Me, 3, False
If Me.chkFalzmarke.Enabled Then  'Falzmarke umschalten
    ActiveDocument.Bookmarks("dM_Falzmarke").Range.Font.Hidden = _
        Not Me.chkFalzmarke.Value
End If
Unload Me
End Sub

```

- 6 Während das Vorhandensein der *dM\_Falzmarke*-Textmarke indirekt anhand der *Enabled*-Eigenschaft überprüft wird, ist das für die Kombinationsfelder nicht nötig. Waren die zugehörigen Textmarken nämlich nicht vorhanden, steht deren Auswahl garantiert auf dem *ListIndex 0*, was der Auswahl »(wie Dokument)« entspricht und keine Aktion auslöst. Da die Kombinationsfelder dann deaktiviert sind, kann der Benutzer das auch nicht mehr ändern.

Nun funktioniert der Kopf-/Fußzeilen-Dialog wie geplant, sodass sich mit seiner Hilfe passende Briefkopf-Logos und Fußzeilen-Angaben auswählen lassen. Auch wenn der Fortschritt hier nur drei Schritte anzeigt, werden Sie feststellen, dass ein solcher Import durchaus Zeit braucht, sodass der Benutzer anhand der Anzeige wenigstens sieht, dass es noch vorangeht.

