

PROGRAMMER'S CHOICE



André Minhorst



Access 2007

Das Grundlagenbuch für Entwickler

Leseprobe

Kapitel 12 - Ribbon

Weitere Informationen und Bestellung:
<http://www.access-entwicklerbuch.de/2007>

 ADDISON-WESLEY

Komplettes Buch als eBook
und alle Beispiele





12 Ribbon

Die neue Menüleiste der Microsoft Office-Anwendungen heißt Ribbon. Die deutsche Bezeichnung für Ribbon lautet Multifunktionsleiste (wörtlich übersetzt: »Band«), in diesem Buch wird jedoch ausschließlich von Ribbon die Rede sein – das ist erstens kürzer und zweitens wesentlich cooler. Das Ribbon ist Teil der runderneuten Benutzeroberfläche und übernimmt die Rolle von Menü- und Symbolleisten gleichzeitig.

Für Sie als Entwickler ist natürlich besonders interessant, wie Sie das Ribbon für Ihre Zwecke anpassen können, um etwa die eingebauten Befehle auszublenden und eigene Tabs, Groups und Steuerelemente hinzuzufügen.

Deshalb hält sich dieses Kapitel nicht mit einer Beschreibung der Bedienung des Ribbons auf (wenn Sie es bis zu diesem Kapitel geschafft haben, sollten Sie das schon drauf haben), sondern konzentriert sich auf die Anpassung und Programmierung des Ribbons.

Vorab zu Ihrer Beruhigung: Das Anpassen funktioniert ohne Weiteres, aber erstens völlig anders und zweitens nicht so einfach wie bei den *CommandBars* älterer Access-Versionen. Der Hauptgrund dafür, dass es nicht so leicht geht, ist die Tatsache, dass Microsoft noch keine grafische Benutzeroberfläche zum Anpassen des Ribbons mitliefert.

Es gibt zwar schon einige freie und kostenpflichtige Tools am Markt, aber Microsoft ist zur Zeit der Drucklegung dieses Buchs noch nicht mit einer eigenen Lösung herausgekommen. Es ist allerdings zu erwarten, dass dies früher oder später geschehen wird.



Beispieldatenbank: Eine Datenbank mit vorbereitetem Datenmodell finden Sie auf der Buch-CD unter `\Kap_12\Ribbon.accdb`.

12.1 Definition des Ribbons

Wie schwer Ihnen das Anpassen des Ribbons fällt, hängt in erster Linie davon ab, ob Sie bereits XML-Grundkenntnisse besitzen oder sich diese aneignen wollen. Sie müssen allerdings kein XML-Guru sein, um das Ribbon anzupassen.

Microsoft unternimmt in Office 2007 den Versuch, nicht nur Dokumente, sondern auch sehr viele Elemente der Oberfläche über XML-Dateien zu steuern. Diese XML-Steuerdateien befinden sich meist in den Dokumenten selbst. Im Falle von Access handelt es sich dabei üblicherweise um Tabellen.

Die Anpassungen legen Sie grundsätzlich in einem XML-Dokument fest, das Sie Access auf verschiedenen Wegen zugänglich machen können. Damit das alles funktioniert, hat Microsoft eine XML-Schema-Datei (XSD) spendiert, die Vorgaben für den Aufbau einer Ribbon-XML-Definition liefert. Mit einem vernünftigen XML-Editor ausgestattet können Sie die notwendige XML-Datei leicht erstellen. »Vernünftig« bedeutet dabei, dass der Editor IntelliSense- oder eine ähnliche Funktion mitbringt, die Ihnen die möglichen Elemente und Werte für das XML-Dokument vorschlägt – dazu jedoch später mehr. Wenn die von Ihnen erstellte XML-Datei nämlich nicht exakt zu den Vorgaben der Schemadefinitionen passt, dann meldet Access einen Fehler des XML-Parsers. Sie sollten also peinlich genau darauf achten, dass die XML-Datei »wellformed« und schema-konform ist.

Zunächst bekommen Sie noch eine Definition dessen mit auf den Weg, was das Ribbon nun eigentlich ist: Und dabei hilft die oben erwähnte XSD-Datei. Sie legt nämlich das *Ribbon*-Element als Root-Element fest und sieht darunter vier verschiedene Elemente vor:

- ▶ *qat* (Schnellzugriffsleiste): Kleine Leiste rechts neben dem Office-Button und standardmäßig über der Ribbon-Leiste, enthält Befehle, die schnell zugänglich sein sollen und lässt sich per Benutzeroberfläche anpassen (Kontextmenüeintrag *Symbolleiste für den Schnellzugriff anpassen*)
- ▶ *tabs*: machen das »eigentliche« Ribbon aus, können über die »Registerreiter« ausgewählt werden und enthalten in Gruppen aufgeteilte Steuerelemente
- ▶ *contextualTabs*: Tabs, die in Zusammenhang mit bestimmten Objekten eingeblendet werden – etwa das Entwurf-Tab beim Anzeigen eines Formulars in der Entwurfsansicht

► *officeMenu*: Menü, das sich beim Klicken auf den Office-Button öffnet

Auf der gleichen Ebene wie der des *ribbon*-Elements können Sie auch noch ein *commands*-Element einfügen, mit dem Sie die durch Steuerelemente eingebauter Tabs ausgeführten Funktionen durch den Aufruf benutzerdefinierter Funktionen ersetzen können.

Ribbon-Definition mit XML Notepad 2007

Offensichtlich ist das »Ribbon« also mehr als nur die neue Menüleiste von Access- und anderen Office-Anwendungen. Das können Sie sich komfortabel etwa im kostenlosen XML Notepad 2007 von Microsoft ansehen [1]. Abbildung 12.1 zeigt den schematischen Aufbau eines XML-Dokuments mit allen vier oben aufgelisteten Bereichen.

Die Bedienung des XML Notepads ist gewöhnungsbedürftig, weil Sie nicht direkt mit dem Editor im Text des XML-Dokuments arbeiten, aber dafür prüft das Tool Ihr Dokument direkt anhand der passenden XSD-Datei. Diese können Sie dem XML-Notepad über das Menü *View|Schemas* und den dortigen Eintrag *File|Add schemas* bekannt machen. Sie finden die Schemadefinition *customui.xsd* unter [2] oder auf der Buch-CD unter `\Kap12\customUI.xsd`.

Die Abbildung veranschaulicht vor allem den Aufbau der eigentlichen Ribbon-Leiste: Sie enthält eine *tabs*-Auflistung mit mehreren *tab*-Elementen, die wiederum eine oder mehrere Gruppen mit einem oder mehreren Steuerelementen enthält (in der Abbildung gibt es aus Platzgründen nur je ein Tab und eine Gruppe, dafür aber mit allen möglichen Steuerelementen).

Zu den Elementen *qat*, *contextualTabs* und *officeMenu* finden Sie im Laufe dieses Kapitels weitere Informationen.

Wenn Sie das Ribbon anpassen möchten, schreiben Sie genau ein XML-Dokument, das alle Änderungen enthält. Am wichtigsten ist dabei zunächst, ob Sie die vorhandenen Tabs beibehalten oder ausblenden möchten. Und danach können Sie richtig loslegen und Tabs, Gruppen und Steuerelemente anlegen oder einen der anderen Bereiche beackern.

12.2 Symbolleiste für den Schnellzugriff

Bevor Sie zum XML-Editor greifen, noch schnell die Beschreibung eines Elements des Ribbon, das Sie über die Benutzeroberfläche anpassen können: die »Symbolleiste für den Schnellzugriff« (im Folgenden kurz »Schnellzugriffsleiste« genannt). Sie ergänzt die eigentliche Ribbon-Leiste und enthält im Auslieferungszustand drei Einträge – einen zum Speichern, einen zum Wiederholen und einen zum Rückgängigmachen von Aktionen. Weitere Einträge fügen Sie auf verschiedene Weise hinzu:

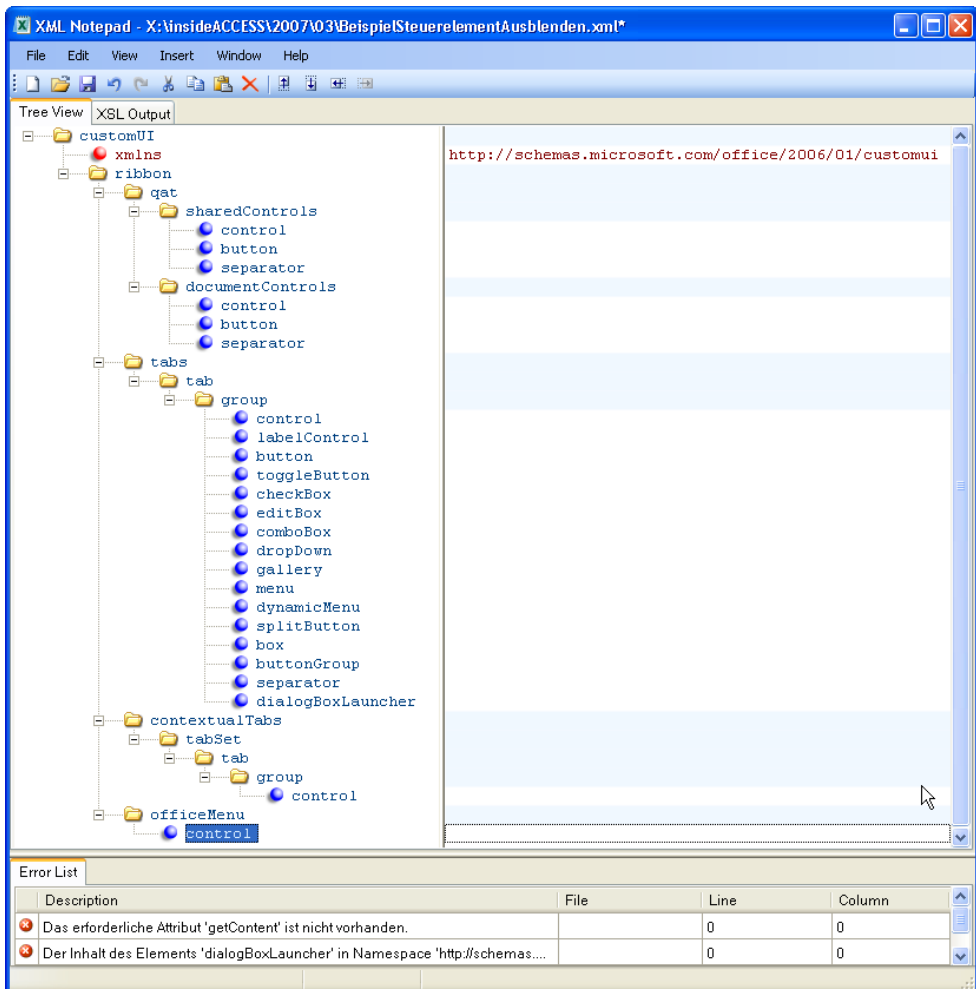


Abbildung 12.1: Die Struktur und einige mögliche Elemente einer Ribbon-XML-Dokumentation

- ▶ Über das mit der nebenan liegenden Schaltfläche zu öffnende Menü: Damit können Sie die vorhandenen Einträge abwählen und andere hinzufügen (siehe Abbildung 12.2).
- ▶ Beliebige Befehle aus den vorhandenen Ribbons fügen Sie der Schnellzugriffsleiste über den Eintrag *Zu Symbolleiste für den Schnellzugriff hinzufügen* des Kontextmenüs des jeweiligen Befehls hinzu (siehe Abbildung 12.3).
- ▶ Im Bereich *Anpassen* des Dialogs *Access-Optionen* können Sie alle in den Ribbons vorhandenen Befehle in der Übersicht anzeigen und der Schnellstartleiste hinzufügen. Dies macht Sinn, wenn man sich etwa eine immer sichtbare *Drucken*-Schaltfläche auf

den Schirm zaubern möchte. Den passenden Dialog (siehe Abbildung 12.4) öffnen Sie entweder über den herkömmlichen Weg (*Office-Button* | *Access-Optionen*), über den Eintrag *Weitere Befehle...* des Schnellzugriffsleisten-Menüs oder den Eintrag *Symbolleiste für den Schnellzugriff anpassen...* des Kontextmenüs eines Ribbon-Steuerelements.

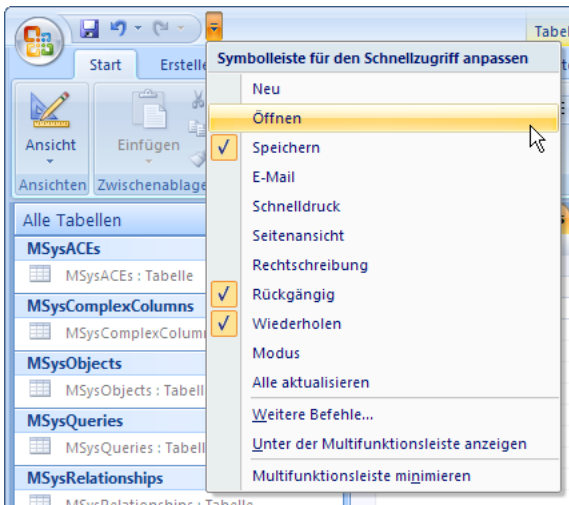


Abbildung 12.2: Die Symbolleiste für den Schnellzugriff und das Menü für ihre Anpassung

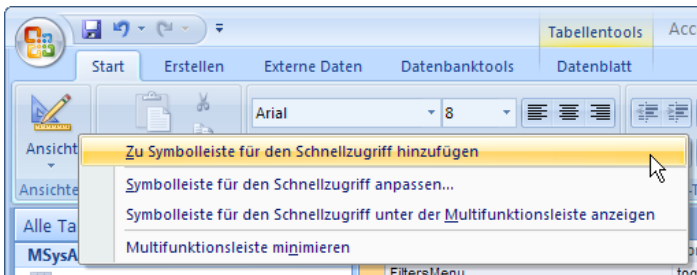


Abbildung 12.3: Per Kontextmenü können Sie beliebige Befehle zur Schnellzugriffsleiste hinzufügen

Schnellzugriffsleiste positionieren

Die standardmäßig neben dem Office-Button befindliche Schnellzugriffsleiste können Sie auch unterhalb des Ribbons platzieren. Dazu wählen Sie etwa den Eintrag *Unter der Multifunktionsleiste anzeigen* des Menüs der Schnellzugriffsleiste. Mit dieser Einstellung vergrößert sich der Ribbon-Bereich allerdings um einen zusätzlichen Balken.

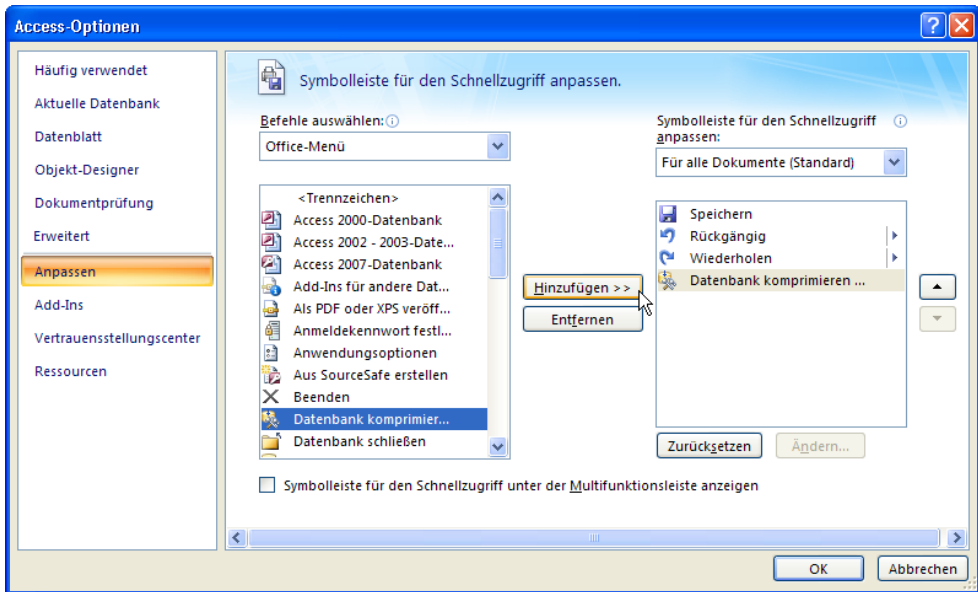


Abbildung 12.4: Dialog zum Anpassen der Schnellzugriffsleiste

Anwendungsspezifische Schnellzugriffsleiste

Sie können der Schnellzugriffsleiste für jede Datenbankanwendung ein eigenes Gesicht verpassen. Dazu klappen Sie im Dialog zum Anpassen der Schnellzugriffsleiste das Kombinationsfeld auf der rechten Seite auf und wählen dort die aktuell geöffnete Datenbank aus. Access zeigt nun eine leere Liste an, die Sie mit den gewünschten Befehlen füllen können. Alle Befehle, die Sie hier hinzufügen, zeigt Access zusätzlich zu den für alle Datenbanken ausgewählten Befehlen an.

Wenn Sie also nur datenbankspezifische Einträge anzeigen wollten, müssten Sie alle allgemein sichtbaren Einträge der Schnellzugriffsleiste entfernen. Damit wäre allerdings nicht sichergestellt, dass auch der Anwender nur diese Einträge sieht – das hängt von der auf seinem Rechner vorliegenden Konfiguration der Schnellzugriffsleiste ab. Um dies zu gewährleisten, kommen Sie um die Definition eines eigenen Ribbons nicht herum.

12.3 Eigene Ribbon-Tabs erstellen

Bevor Sie loslegen, sollten Sie im Dialog *Verweise* des VBA-Editors (*Extras | Verweise*) einen Verweis auf die Bibliothek *Microsoft Office 12.0 Object Library* anlegen. Anderenfalls kann es bei den folgenden Beispielen zu Fehlermeldungen kommen.

12.3.1 Ein einfaches Ribbon

Bei der Anpassung des Ribbons ist aller Anfang schwer, vor allem für VBA-verwöhnte Entwickler: Immerhin haben Sie es hier mit XML zu tun, und dies ist bezüglich der korrekten Groß- und Kleinschreibung mindestens so pingelig wie die Korrektorin dieses Buches. Fangen Sie also mit einem ganz einfachen Beispiel an, das im Ribbon nur ein Tab mit einer einzigen Schaltfläche anlegen soll.

Der einfachste Weg ist, das notwendige XML-Dokument in einer speziell für diesen Zweck vorgesehenen Tabelle zu speichern, die einen festen Aufbau und den Namen *USysRibbons* haben muss. Access liest beim Starten einer Anwendung alle in dieser Tabelle enthaltenen Datensätze ein und stellt diese zur Auswahl zur Verfügung (wo, erfahren Sie weiter unten).

Wenn Sie diese Tabelle unter diesem Namen anlegen und speichern, scheint sie plötzlich verschwunden zu sein. Der Grund ist, dass Access alle Tabellen mit bestimmten Anfangsbuchstaben (etwa *MSys...* oder *USys...*) standardmäßig für Systemtabellen hält und ausblendet. Zu sehen bekommen Sie diese Tabellen, wenn Sie in den Navigationsoptionen (Kontextmenü des Navigationsfensters, Eintrag *Navigationsoptionen...*) die Option *Systemobjekte anzeigen* aktivieren.

Zum Speichern von Ribbon-Definitionen in der Tabelle *USysRibbons* gehen Sie wie folgt vor: Legen Sie eine neue Tabelle an und fügen Sie die in Abbildung 12.5 dargestellten Felder hinzu. Das Feld *RibbonName* (Text, 255 Zeichen) soll die später in einer Auswahlliste angezeigte Bezeichnung enthalten, das Feld *RibbonXML* (Memofeld) den für die Erstellung notwendigen XML-Code.

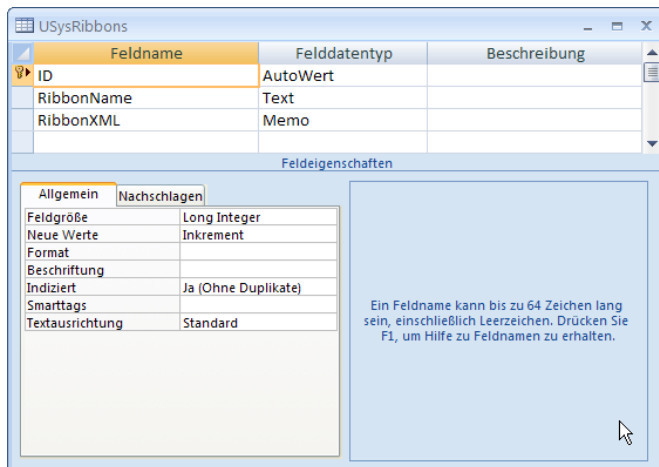


Abbildung 12.5: Die Tabelle *USysRibbons* speichert die Informationen zu benutzerdefinierten Anpassungen des Ribbons

Wechseln Sie dann in die Datenblattansicht der Tabelle und legen Sie einen neuen Datensatz an. Für das Feld *RibbonName* tragen Sie die Bezeichnung *RibbonMitEinerSchaltflaeche* ein, für das Feld *RibbonXML* den folgenden XML-Code:

```
<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui">
  <ribbon startFromScratch="false">
    <tabs>
      <tab id="tabRibbonMitEinerSchaltflaeche"
        label="RibbonMitEinerSchaltflaeche"
        visible="true">
        <group id="grpBeispielgruppe" label="Beispielgruppe">
          <button id="btnBeispielschaltflaeche"
            label="Beispielschaltfläche" />
        </group>
      </tab>
    </tabs>
  </ribbon>
</customUI>
```

Listing 12.1: XML-Code für das Anlegen eines Ribbons mit einer einzigen Schaltfläche

Das Attribut *startFromScratch* des Elements *ribbon* legt fest, ob die eingebauten Ribbons angezeigt (*false*) oder ausgeblendet werden sollen (*true*). Jedes Ribbon-XML-Dokument enthält ein *tabs*-Element, das wiederum eines oder mehrere *tab*-Elemente enthält. Ein *tab*-Element entspricht einer eigenen Registerseite des Ribbons. Im vorliegenden Fall hat das *tab*-Element die Attribute *id*, *label* und *visible*. Das Attribut *id* muss für jedes Element vorhanden sein. Das *group*-Element leitet eine Gruppe eines Ribbons ein – das ist ein eigener Bereich, der ein oder mehrere Steuerelemente enthalten kann. Es ist das einzig mögliche Unterelement des *tab*-Elements, jedes Tab muss also eine Gruppe enthalten. Das *button*-Element schließlich beschreibt die eigentliche Schaltfläche, die in diesem Fall einfach nur angezeigt wird und keinerlei Funktion hat. Das Ribbon soll schließlich wie in Abbildung 12.6 aussehen.

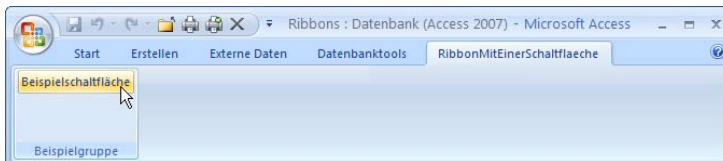


Abbildung 12.6: Dieses Ribbon entspricht der Definition aus Listing 12.1

Vor der Anzeige des Ribbons müssen Sie jedoch noch einige weitere Schritte durchführen, denn irgendwie muss Access ja noch erfahren, dass es dieses Ribbon anzeigen soll. Nach dem Einfügen des Ribbon-Datensatzes in die Tabelle *USysRibbons* schließen Sie die Datenbank und öffnen diese erneut, damit Access die in der Tabelle enthaltenen

Ribbons einliest – und zwar in die Liste der verfügbaren benutzerdefinierten Ribbons. Am schnellsten schließen Sie die Datenbank und starten diese neu, wenn Sie den Eintrag *Verwalten | Datenbank komprimieren und reparieren* aus dem Office-Menü auswählen.

Die vorhandenen Ribbons finden Sie in den Access-Optionen im Bereich *Aktuelle Datenbank | Multifunktionsleisten- und Symbolleisteoptionen* unter *Name der Multifunktionsleiste* (siehe Abbildung 12.7). Wenn Sie hier das soeben angelegte Ribbon-XML-Dokument auswählen, fordert Access Sie zum Schließen und erneuten Öffnen der Anwendung auf und zeigt dann endlich das modifizierte Ribbon an.

Die Tabelle *USysRibbons* der Beispieldatenbank enthält übrigens einige Dutzend Ribbon-Definitionen, deren Code in diesem Kapitel ausschnittsweise besprochen wird.

12.3.2 Schaltfläche mit Funktion versehen

Bisher löst ein Klick auf die Beispielschaltfläche noch keine Funktion aus, aber das holen Sie nun nach. Erweitern Sie die Definition des *button*-Elements aus dem obigen Ribbon-XML-Dokument wie folgt:

```
<button id="btnBeispielschaltflaeche" label="Beispielschaltfläche" onAction="Beispielfunktion" />
```

Das *onAction*-Attribut enthält den Aufruf einer öffentlichen VBA-Routine – eine so genannte Callback-Funktion –, die Sie in einem Standardmodul anlegen. Erstellen Sie also ein neues Standardmodul namens *mdlRibbon* und fügen Sie die folgende Routine hinzu:

```
Public Function Beispielfunktion(ctl As IRibbonControl)
    MsgBox "Sie haben eine Ribbon-Schaltfläche angeklickt."
End Function
```

Listing 12.2: Diese Routine wird beim Klick auf die passende Schaltfläche ausgelöst

Die Funktion soll schlicht und einfach ein Meldungsfenster anzeigen, um die einwandfreie Funktion des Aufrufs zu bestätigen. Um dies zu testen, müssen Sie die Datenbankanwendung erneut schließen und wieder öffnen und dann auf die neue Schaltfläche klicken.

Möglicherweise erscheint nun eine Meldung wie »Der von Ihnen eingegebene Ausdruck enthält den Namen einer Funktion, die von Microsoft Office Access nicht gefunden werden kann«. Das bedeutet nicht, dass Sie etwas falsch gemacht haben, allerdings liefert diese Meldung auch nicht unbedingt hilfreiche Informationen.

Vermutlich verbieten die aktuellen Sicherheitseinstellungen für diese Datenbankanwendung das Ausführen von VBA-Code, was Sie ganz einfach durch einen Aufruf der soeben angelegten Funktion im Direktfenster testen können. Die Meldung »Die Makros in diesem Projekt sind deaktiviert. Informationen zum Aktivieren der Makros finden Sie in

der Onlinehilfe oder der Dokumentation der Host-Anwendung.« würde diese Annahme bestätigen; Sie sollten daher zunächst die Sicherheitseinstellungen anpassen. Weitere Informationen hierzu finden Sie in Kapitel 18, Abschnitt 18.4, »Vertrauensstellungscenter«.

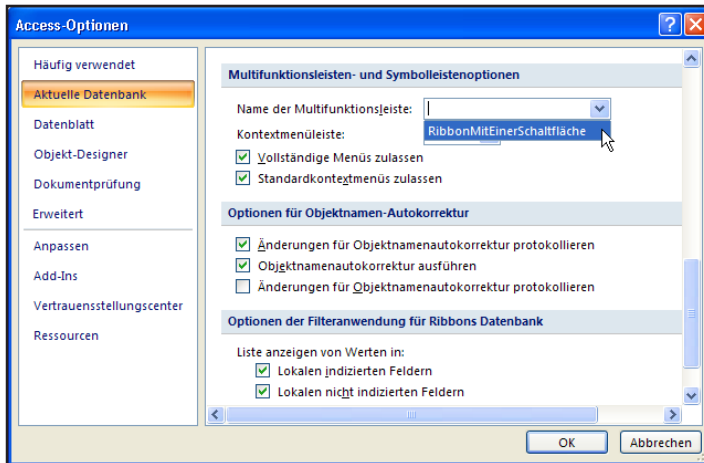


Abbildung 12.7: Auswahl eines benutzerdefinierten Ribbons im Dialog Access-Optionen

Wenn die Sicherheitseinstellungen für die aktuelle Datenbank entsprechend eingestellt sind, sollte nun das in der Funktion angegebene Meldungsfenster erscheinen und die Funktionstüchtigkeit Ihrer ersten selbst angelegten Ribbon-Schaltfläche bestätigen.

Angeklickte Schaltfläche ermitteln

Vielleicht möchten Sie ja alle Funktionen für Schaltflächen in einer einzigen Routine unterbringen. Eine Ribbon-Schaltfläche liefert alles Notwendige dazu – nämlich die Möglichkeit, einen Verweis auf die betätigte Schaltfläche an die aufgerufene Routine zu übergeben. Dazu müssen Sie im Ribbon-XML-Dokument gar nichts ändern, sondern nur die Zielfunktion. Diese sieht im einfachsten Fall so aus:

```
Public Function Beispielfunktion(ctl As IRibbonControl)
    MsgBox "Sie haben die Schaltfläche '" & ctl.Id & "' angeklickt."
End Function
```

Listing 12.3: Ausgeben des Namens der angeklickten Schaltfläche

Etwas strukturierter darf es dann doch sein, also fügen Sie ein *Select Case*-Konstrukt in die Routine ein:

```
Public Function Beispielfunktion(ctl As IRibbonControl)
    Select Case ctl.Id
```

```

Case "btnBeispielschaltflaeche1"
    'tu was
Case "btnBeispielschaltflaeche2"
    'tu was anderes
Case Else
    MsgBox "Unbekannte Schaltfläche!"
End Select
MsgBox "Sie haben die Schaltfläche '" & ctl.Id & "' angeklickt."
End Function

```

Listing 12.4: Auswerten des aufrufenden Steuerelements in einer Select Case-Anweisung

12.4 Fehler in Ribbon-XML-Dokumenten erkennen

Beim Experimentieren mit XML-Dokumenten zur Definition von Ribbons passiert es zwangsläufig, dass Access das gewünschte Ribbon einfach nicht anzeigt. Meist ist dies auf einen Fehler im XML-Dokument zurückzuführen, den Access stillschweigend übergeht. Das lässt sich durch Einstellen einer Option ändern: Öffnen Sie per *Office-Button | Access-Optionen* den Optionen-Dialog, wechseln Sie dort zur Registerseite *Erweitert* und aktivieren Sie die Option *Fehler in Benutzeroberflächen in Add-Ins anzeigen* im Bereich *Allgemein* (siehe Abbildung 12.8).

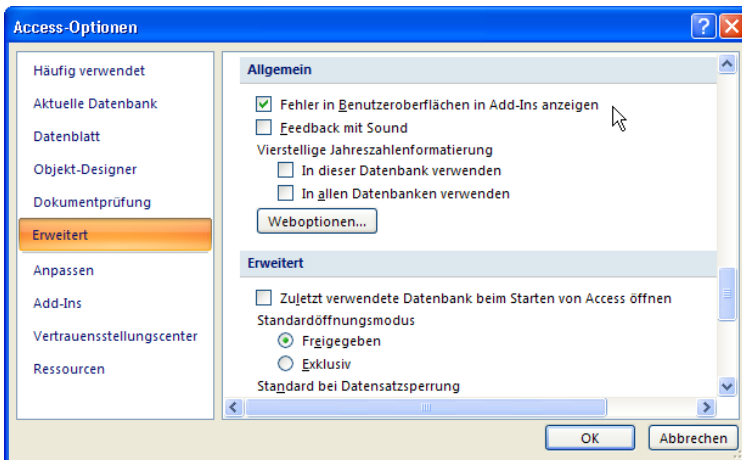


Abbildung 12.8: Aktivieren der Anzeige von Fehlern in Ribbon-XML-Dokumenten

Wenn Access dann einen Fehler im Ribbon-XML-Dokument findet, zeigt es diesen mit einer Meldung wie in Abbildung 12.9 an. In diesem Fall enthält das Element *ribbon* das nicht dafür definierte Attribut *loadImage*. Anhand der Zeilen- und Spaltennummer können Sie den Fehler im XML-Dokument dann schnell lokalisieren.

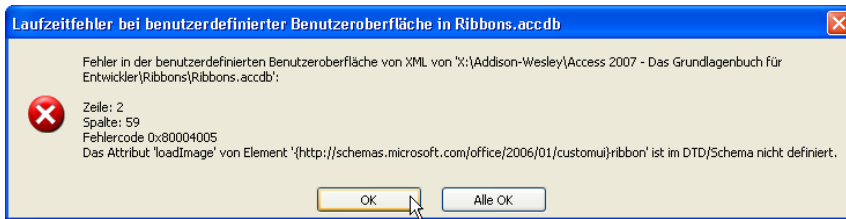


Abbildung 12.9: Fehler in Ribbon-XML-Dokumenten werden bei aktivierter Option »Fehler in Benutzeroberflächen in Add-Ins anzeigen« gemeldet

12.5 Callback-Funktionen

Callback-Funktionen werden, soweit angegeben, vom Ribbon beim Anlegen oder bei bestimmten Aktionen aufgerufen. Es gibt zwei Arten von Callback-Funktionen: solche, die Werte für Attribute zurückliefern, und jene, die als Reaktion auf Benutzeraktionen aufgerufen werden.

12.5.1 Die get...-Attribute

Jedes Element hat mehrere Attribute, die mit *get...* beginnen. Beim *button*-Element sind dies beispielsweise *getDescription*, *getEnabled*, *getImage*, *getKeytip*, *getLabel*, *getScreenTip*, *getShowImage*, *getShowLabel*, *getSize*, *getSupertip* und *getVisible*. All diesen Attributen können Sie Namen von Callback-Funktionen zuweisen, die beim Erzeugen des Ribbons die passenden Attributwerte ermitteln und zurückgeben. Ihr Einsatz ist dann sinnvoll, wenn Sie beim Erstellen des Ribbon-XML-Dokuments noch nicht wissen, wie der Inhalt eines der zu füllenden Attribute aussieht – sonst könnten Sie ja den passenden Wert auch einfach als Wert des Attributs eintragen.

Wenn Sie etwa dem Attribut *label* erst beim Anlegen des Ribbons (beim Öffnen der Anwendung oder, wenn das Ribbon einem Formular oder Bericht zugeordnet ist, beim Öffnen dieser Objekte) einen bestimmten Wert zuordnen möchten, legen Sie mit *getLabel* eine für diesen Zweck angelegte VBA-Routine namens *GetLabel* fest. Das Beispiel ist nicht so abwegig und könnte beispielsweise beim Erstellen mehrsprachiger Anwendungen interessant sein.

Der Kopf dieser Routine muss einer bestimmten Syntax folgen; eine Zusammenstellung der Syntax aller möglichen Funktionen finden Sie weiter hinten in diesem Kapitel in Tabelle 12.3. Die Definition einer Schaltfläche im XML-Dokument für das Ribbon sieht etwa wie folgt aus (*get...*-Attribut hervorgehoben):

```
<button id="btnBeispielschaltflaeche" onAction="Beispielfunktion"
  image="ribbon.png" getLabel="GetLabel"/>
```

Damit sich das Attribut beim Laden des Ribbons überhaupt bemerkbar macht, müssen Sie eine passende Routine in einem Standardmodul anlegen. So stellen Sie zumindest schon einmal sicher, dass Access entweder einen Fehler meldet oder, wenn Sie alles richtig gemacht haben, das Attribut mit dem entsprechenden Wert füllt. Eine Routine für das Zurückgeben eines Wertes für das im XML-Element angegebene Attribut sieht so aus:

```
Public Sub GetLabel(ct1 As IRibbonControl, ByRef label)
    label = "Beispielbeschriftung"
End Sub
```

Listing 12.5: Diese Callback-Routine liefert den String »Beispielbeschriftung« an das aufrufende Ribbon zurück

Vielleicht wundert es Sie, dass hier von einer Funktion die Rede, die aufgerufene Routine aber als Sub-Prozedur ausgeführt ist. Tatsächlich handelt es sich bei den meisten Callback-Routinen um solche Sub-Prozeduren, die eine von der Prozedur zu füllende *ByRef*-Platzhaltervariable bereitstellen.

Sonderfall LoadImage

Etwas anders funktioniert die Routine *LoadImage*. Sie wird wie die in den *get...*-Attributen angegebenen Funktionen beim Anlegen eines Ribbons aufgerufen, ersetzt aber unter Umständen eine ganze Reihe notwendiger *getImage* und *getItemImage*-Attribute: Sie liest alle in Steuerelementen und deren Unterelementen wie etwa *item*-Elementen enthaltenen *image*-Attribute aus und ruft die unter *loadImages* (*customUI*-Tag) angegebene Callback-Funktion für jedes Image einmal auf. Diese Routine gibt dann passende Objekte mit Verweisen auf die entsprechenden Bilder zurück.

12.5.2 Ereignisseigenschaften

Ribbons bieten Ereignisseigenschaften, wie sie von der VBA-Programmierung von Formularen und Berichten bekannt sind – zumindest, was den Auslöser angeht. Dabei handelt es sich nämlich entweder um einen Klick auf ein Steuerelement, das Drücken einer mit dem Attribut *keytip* angegebenen Tastenkombination oder die Änderung seines Inhalts. Außerdem gibt es noch eine Ereignisseigenschaft, die beim Laden des Ribbons ausgelöst wird:

- ▶ *onAction*: Verfügbar für *button*-, *checkBox*-, *dropDown*-, *gallery*- und *toggleButton*-Elemente, wird bei Aktionen wie etwa einem Mausklick oder der Auswahl eines der Einträge ausgelöst.
- ▶ *onChange*: Verfügbar für *comboBox*- und *editBox*, wird beim Ändern des Inhalts beziehungsweise der Auswahl eines neuen Eintrags ausgewählt.

- ▶ *onLoad*: Wird beim Laden des Ribbons ausgelöst.

Auch für die Ereignisseigenschaften gibt es jeweils eine vorgegebene Syntax, die immer einen Verweis auf das aktuelle Steuerelement und auf den aktuellen Wert übergibt. Auch diese Syntax-Beschreibungen finden Sie weiter hinten in Tabelle 12.3.

12.5.3 Umgang mit Callback-Funktionen

Wenn Sie einmal ein Ribbon mit mehr als nur einem *tab*-Element mit wenigen Unterelementen und passenden Callback-Funktionen bestücken, bekommen Sie möglicherweise Probleme bei der Vergabe der Namen für die Callback-Funktionen. Die Beispiele verwenden ja meist Routinennamen, die dem Namen der Ereignisseigenschaft bis auf den großgeschriebenen Anfangsbuchstaben gleichen.

Wenn Sie auch so vorgehen möchten, müssen Sie berücksichtigen, dass es auch einmal mehr als ein Steuerelement geben kann, für das Sie etwa eine Routine namens *OnAction* anlegen möchten.

In diesem Fall haben Sie zwei Möglichkeiten:

- ▶ Sie verwenden für jede Callback-Routine jedes Elements einen eindeutigen Namen.
- ▶ Sie verwenden für jede Ereignisart eine Routine, die jeweils das auslösende Steuerelement auswertet – etwa mit einem *Select Case* über den per Parameter übergebenen Steuerelementnamen.

Callback-Routinen mit eindeutigem Namen

Im ersten Fall weisen Sie den zu erstellenden Routinen Namen zu, die eindeutig sind und dennoch dem Element zugeordnet werden können (sonst entsteht schnell ein sehr unübersichtlicher Berg von Callback-Funktionen).

So können Sie etwa das *label*- oder *id*-Attribut des Steuerelements integrieren und die Bezeichnung *btnBeispiel_onChange* verwenden. Und wenn verschiedene *group*- oder *tab*-Elemente Steuerelemente gleichen Namens enthalten, bauen Sie eben auch noch das *label*- oder *id*-Attribut der übergeordneten Elemente mit ein.

Im Extremfall würde eine Callback-Routine dann beispielsweise *tabMain_grpDateien_btnOeffnen_onAction* heißen.

Eine Callback-Routine für alle Elemente

Bei der zweiten empfehlenswerten Variante legen Sie tatsächlich nur eine Callback-Routine für jedes Ereignis wie etwa *onAction* oder *getDescription* an, deren Name beispielsweise der Attributbezeichnung mit großem Anfangsbuchstaben entspricht (also

OnAction oder *GetDescription*). Beim Aufruf einer solchen Funktion wertet diese dann den Parameter *control* aus, der übrigens in allen Callback-Funktionen enthalten ist. Das sieht dann beispielsweise wie im folgenden Listing aus:

```
Public Sub OnAction(ctl As IRibbonControl)
    Select Case ctl.id
        Case "btnBeispielschaltflaeche1"
            'tu was
        Case "btnBeispielschaltflaeche2"
            'tu was anderes
        Case Else
            MsgBox "Unbekannte Schaltfläche!"
    End Select
End Sub
```

Listing 12.6: Diese Routine wertet aus, von welchem Steuerelement sie aufgerufen wurde, und reagiert mit der für dieses Element vorgesehenen Aktion

Leider kommen Sie damit nicht allzu weit: Wie Sie Tabelle 12.3 entnehmen können, besitzen Callback-Funktionen für gleichnamige Ereignisseigenschaften durchaus nicht immer die gleiche Syntax.

Die obige Routine entspricht etwa der Syntax für ein *button*-Element, die Syntax für die *onAction*-Callback-Routine eines *dropDown*-Elements hingegen sieht so aus:

```
Sub OnAction(control As IRibbonControl, selectedId As String, selectedIndex As Integer)
```

Und da Access einen Fehler meldet, wenn eine Callback-Funktion die falschen Parameter enthält (vorausgesetzt, Sie haben wie oben beschrieben die Fehlerbehandlung für Ribbons aktiviert), müssen Sie zumindest für jede Steuerelementart eine Callback-Funktion mit einem eigenen Namen anlegen. Beim *onAction*-Attribut könnte diese etwa *OnButtonAction*, *OnCheckBoxAction*, *OnDropDownAction*, *OnGalleryAction* oder *OnToggleActionButton* heißen. Die Klasse (genauer: das Interface) *IRibbonControl* ist übrigens in der *Microsoft Office 12.0 Library* definiert, die Sie deshalb den Verweisen des VBA-Projekts hinzufügen müssen. Sie hat die folgenden Eigenschaften:

- ▶ *ID*: Gibt die in der XML-Definition für das Steuerelement festgelegte und obligatorische ID zurück.
- ▶ *Context*: Gibt einen Objektverweis auf die Anwendung des Ribbons zurück. Bei Access handelt es sich hierbei um das *Application*-Objekt.
- ▶ *Tag*: Gibt eine Zeichenfolge zurück, die Sie optional in der XML-Definition für das Steuerelement angegeben haben. Beispiel:

```
<button id="btn1" label="Beispielschaltfläche" tag="Zusatzinfo"/>
```

Fehler in Callback-Routinen

Falls die Deklaration der Callback-Funktion nicht genau den Vorgaben entspricht, weil Sie fälschlicherweise etwa eine Deklaration wie in Listing 12.6 für ein *dropDown*-Element angegeben haben, dann meldet Access nicht etwa einen Syntaxfehler, sondern bemerkt lapidar: »Access kann die Makro- oder Rückrufaktion 'OnAction' nicht ausführen«. Die gleiche Meldung erscheint aber auch, wenn die Prozedur gar nicht existiert. Sollten Sie also feststellen, dass die betroffene Prozedur nicht fehlt, können Sie von einer fehlerhaften Deklaration der Parameter ausgehen. Access setzt das automatische Debugging von VBA bei allen Callback-Funktionen, die Eigenschaften zurückliefern, außer Kraft. Das bedeutet, dass bei Fehlern im VBA-Code der Callback-Routine nicht die übliche VBA-Fehlermeldung erscheint, sondern die Routine stillschweigend verlassen wird – so, als enthielte die Prozedur zu Beginn ein *On Error Resume Next*. Das macht das Debuggen dieser Routinen schwieriger. Wenn Sie vermuten, dass mit Ihrer Callback-Routine etwas nicht stimmt oder sie nicht das erwartete Ergebnis liefert, fügen Sie dem Code ganz vorne die Anweisung *Stop* hinzu. VBA unterbricht dann an dieser Stelle die Ausführung des Codes und lässt Sie im Einzelschritt (F8) die nächsten Code-Zeilen durchlaufen, den Ablauf untersuchen und Variableninhalte einsehen.

12.6 Weitere Ribbon-Steuerelemente

Natürlich gibt es noch mehr Steuerelemente als nur Schaltflächen – genau genommen sogar noch mehr, als es sie für herkömmliche Menü- und Symbolleisten gab – und vor allem noch erheblich mehr Eigenschaften, als das erste Beispiel gezeigt hat.

Die folgenden Abschnitte stellen die Steuerelemente und ihre wichtigsten Eigenschaften vor, außerdem lernen Sie hier Einsatzmöglichkeiten für die unterschiedlichen Callback-Routinen kennen. Zu Beginn wird jedoch noch einmal die Schaltfläche aufgegriffen, da dieses wohl das meistbenutzte Steuerelement im Ribbon sein dürfte.

12.6.1 Schaltflächen

Grundlegendes zum Hinzufügen einer Schaltfläche haben Sie ja bereits weiter oben in Zusammenhang mit dem ersten benutzerdefinierten Beispiel-Ribbon gelesen. Nun möchten Sie der Schaltfläche vielleicht ein Symbol hinzufügen, da diese sonst ein wenig einsam aussieht (keine Sorge, wir fügen später auch noch weitere Steuerelemente hinzu). Dazu gibt es verschiedene Möglichkeiten:

- ▶ Verwenden eines Symbols der Microsoft-Office-Bibliothek
- ▶ Hinzufügen einer benutzerdefinierten Bilddatei mit der Callback-Funktion *LoadImage* für das *customUI*-Objekt

- Hinzufügen einer benutzerdefinierten Bilddatei mit der Callback-Funktion *getImage* des *button*-Elements oder sonstiger Steuerelemente

In den folgenden Abschnitten erfahren Sie mehr über die ersten beiden Methoden und darüber, was es mit der Callback-Funktion *getImage* auf sich hat.

Hinzufügen eines eingebauten Symbols

Die größte Hürde beim Hinzufügen eines bereits in einem der Ribbon-Steuerelemente vorhandenen Bildes ist, den Namen des jeweiligen Steuerelements herauszufinden. Dabei hilft die Access-Tabelle aus [3]. Aus dieser Tabelle suchen Sie die Bezeichnung des gesuchten Steuerelements heraus und entnehmen dann der Spalte *ControlName* die englische Bezeichnung. Wenn Sie beispielsweise eine Schaltfläche mit dem *Speichern*-Symbol versehen möchten, passen Sie die Definition des *button*-Elements an, indem Sie das Attribut *imageMso* wie folgt hinzufügen:

```
<button id="btnBeispielschaltflaeche" label="Beispielschaltfläche" onAction="Beispielfunktion" imageMso="FileSave"/>
```

Die Schaltfläche sieht dann wie in Abbildung 12.10 aus.

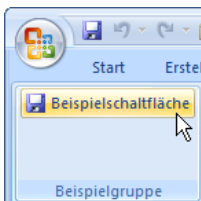


Abbildung 12.10: Eine Schaltfläche mit einem eingebauten Symbol

Wenn Sie wissen möchten, welches Image wie heißt, finden Sie auf der Buch-CD eine passende Beispieldatenbank (*\Kap_12\imagemsos.accdb*). Die Datenbank zeigt alle über 4000 Symbole in mehreren Ribbon-Tabs an, wodurch das Laden der Datenbank einige Momente dauern kann. Nach dem Klick auf eines der Symbole zeigt ein Formular direkt das passende Attribut wie etwa *imageMso="CreateFormBlankForm"* zum Kopieren und Einfügen in die XML-Datei an.

Hinzufügen einer benutzerdefinierten Bilddatei per loadImage

Zum Einfügen einer benutzerdefinierten Bilddatei sind zwei Anpassungen am XML-Dokument und zusätzlicher VBA-Code notwendig. Fügen Sie zunächst der Definition der Schaltfläche, die mit einem Symbol ausgestattet werden soll, das Attribut *Image* hinzu und versehen Sie dieses mit dem Namen der zu verwendenden Bilddatei:

```
<button id="btnBeispielschaltflaeche" label="Beispielschaltfläche" onAction="Beispielfunktion" image="ribbon.png" />
```

Zusätzlich müssen Sie Access mitteilen, dass die im Ribbon enthaltenen benutzerdefinierten Symbole geladen werden sollen. Dazu fügen Sie dem Root-Element *customUI* das Attribut *loadImage* mit dem Namen einer Callback-Funktion hinzu, die einen Objektverweis auf das jeweils zu verwendende Image zurückliefern soll:

```
<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui" loadImage="LoadImage">
```

Das *loadImage*-Attribut sorgt dafür, dass beim Initialisieren des Ribbons alle in Steuerelementen angegebenen *image*-Attribute ausgewertet und die passenden Symbole hinzugefügt werden. Für die angegebene Callback-Funktion müssen Sie natürlich eine passende VBA-Routine anlegen – die hier interessanterweise gar nicht als Funktion, sondern als *Sub*-Prozedur ausgeführt werden muss und statt eines Funktionswertes den Rückgabewert per Parameter liefert. Die Funktion sieht wie folgt aus:

```
Public Sub LoadImage(control, ByRef image)
    Set image = LoadPictureGDIP(CurrentProject.Path & "\" & control)
End Sub
```

Listing 12.7: Die *LoadImage*-Routine liefert das per Parameter angegebene Bild zurück

Der erste Parameter *control* enthält die mit dem Attribut *image* angegebene Zeichenkette und damit in der Regel den Dateinamen der zu verwendenden Datei. Sie können diese auch mit komplettem Pfad angeben, aber wenn Sie die passenden Bilddateien in einem Verzeichnis unterhalb des Datenbankverzeichnisses oder im Datenbankverzeichnis selbst (wie in diesem Fall) speichern, reicht der pure Dateiname des Bildes.

Die Routine weist dem Parameter *image* einen Objektverweis auf ein mit der Funktion *LoadPictureGDIP* geladenes Bild zu. Die Funktion *LoadPictureGDIP* stammt aus dem in Kapitel 11, »Bilder und binäre Dateien« vorgestellten Standardmodul *mdlOGL2007*. In dem Kapitel haben Sie auch erfahren, wie Sie Bilder in einem OLE-Feld speichern und daraus ein *StdPicture*-Objekt erzeugen, das Sie direkt ohne Umweg über das Dateisystem für die Anzeige in einem Steuerelement eines Ribbons verwenden können. Das Ergebnis sieht schließlich wie in Abbildung 12.11 aus – eine Schaltfläche mit einem benutzerdefinierten Symbol.

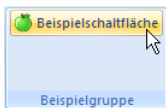


Abbildung 12.11: Eine Schaltfläche mit einem benutzerdefinierten und auf der Festplatte befindlichen Symbol

Ribbon-Schaltfläche anpassen

Anhand der soeben erzeugten Schaltfläche lassen sich leicht die weiteren Attribute von Schaltflächen und sonstigen Steuerelementen erklären.

Größe der Schaltfläche

Die Größe der Schaltfläche passen Sie über das Attribut *size* des *button*-Elements an. Es stehen die Werte *normal* (Standard) und *large* zur Verfügung. Die obige Schaltfläche sieht größenmäßig wie in Abbildung 12.12 aus.

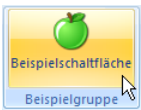


Abbildung 12.12: Eine Schaltfläche in der Ausführung »large«

Hilfetexte

Die Attribute *screentip* und *supertip* sind das (erweiterte) Pendant zum Attribut *ControlTipText*. Erweitert deshalb, weil Sie mit den beiden Attributen eine Überschrift und einen zusätzlichen Text angeben können. Das mit folgender Zeile definierte Schaltflächen-Element sieht wie in Abbildung 12.13 aus:

```
<button id="btnBeispielschaltflaeche" label="Beispielschaltfläche" onAction="
  Beispielfunktion" screentip="Dies ist die Überschrift des Hilfetextes (max.
  1024 Zeichen) ..." supertip="... und das ist der eigentliche Hilfetext
  (auch nur 1024 Zeichen)." image="ribbon.png" size="large"/>
```

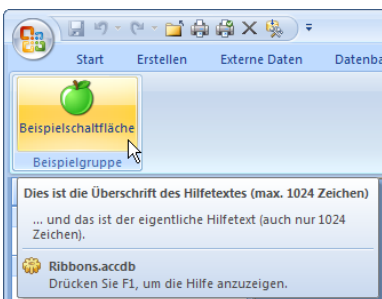


Abbildung 12.13: Eine Schaltfläche mit *screentip*- (Überschrift) und *supertip*-Attribut (Text)

Zeilenumbrüche innerhalb des Textes für das Attribut *supertip* markiert die Zeichenfolge: ``

12.6.2 Kontrollkästchen (checkBox)

Kontrollkästchen bestehen aus einer Beschriftung und dem eigentlichen Kontrollkästchen. Zusätzliche Symbole sind eher selten. Optisch wirkt es am besten, wenn man Kontrollkästchen zwischen Schaltflächen mit Symbolen in der Größe »normal« einreihet (siehe Abbildung 12.14).

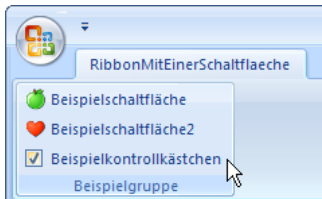


Abbildung 12.14: Beispiel für ein Kontrollkästchen

Wert des Kontrollkästchens abfragen

Was hilft ein Kontrollkästchen in einer Menüleiste, wenn man dessen Wert nicht programmatisch abfragen kann? Also bauen Sie eine passende Funktion ein. Die Definition des Kontrollkästchens erweitern Sie um das *onAction*-Attribut:

```
<checkBox id="ctlBeispielkontrollkaestchen" label="Beispielkontrollkästchen"
onAction="fctKontrollkaestchen" />
```

Die aufgerufene Funktion heißt *fctKontrollkaestchen*. Wenn Sie nun eine Funktion wie weiter oben für eine Schaltfläche verwenden, erleiden Sie Schiffbruch: Die für ein Kontrollkästchen notwendige Funktion hat eine etwas andere Syntax, die einen zusätzlichen Parameter für den Wert des Kontrollkästchens enthält. Ein Beispiel sieht wie folgt aus:

```
Public Function fctKontrollkaestchen(ctl As IRibbonControl, _
    pressed As Boolean)
    MsgBox "Das Kontrollkästchen '" & ctl.Id & "' hat den Wert '" _
        & pressed & "'."
End Function
```

Listing 12.8: Auswertung des Wertes eines Kontrollkästchens

12.6.3 Textfelder

Textfelder heißen im Ribbon-Slang *editBox*-Steuerelement. Eine Besonderheit einer solchen Steuerelements ist wie bei den Kontrollkästchen wiederum die Syntax der VBA-Funktion, die Sie durch Eingeben des Textes und Bestätigung per Eingabetaste oder

Verschieben des Fokus auf ein anderes Element der Benutzeroberfläche auslösen. Die Definition einer einfachen EditText im XML-Dokument erfolgt wie beim Kontrollkästchen:

```
<editBox id="txtBeispielEditBox" label="Beispieltextfeld"
onChange="fctTextfeld" />
```

Lediglich die Routine zum Auswerten der Eingabe verwendet einen anderen, zweiten Parameter:

```
Public Sub fctTextfeld(control As IRibbonControl, text As String)
    MsgBox "Das Textfeld '" & control.Id & "' hat den Wert '" & text & "'"
End Sub
```

Listing 12.9: Diese Routine wird durch das Ereignis onChange einer EditText ausgelöst und zeigt den Namen des Steuerelements sowie den enthaltenen Text an

Das Aussehen des oben definierten Textfeldes sowie die Meldung, die Listing 12.9 verursacht, zeigt Abbildung 12.15.

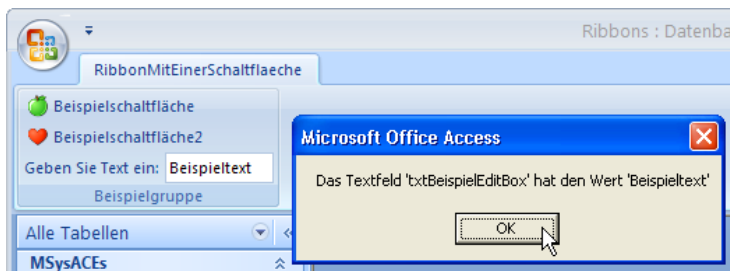


Abbildung 12.15: Ein Ribbon mit Textfeld und einer Meldung, die nach dem Aktualisieren des Textfelds angezeigt wird

Weitere wichtige Attribute:

- ▶ *maxLength*: Anzahl Zeichen, die der Benutzer in das Textfeld eingeben kann, der maximale Wert für dieses Attribut beträgt 1024.
- ▶ *sizeString*: Eine Zeichenkette zum Festlegen der Breite des Steuerelements.

Die folgende Beispielformatierung setzt die beiden genannten Attribute ein und sorgt somit dafür, dass unter den gegebenen Bedingungen die Zeichenfolge »André Minhorst« genau in das Textfeld passt.

```
<editBox maxLength="50" sizeString="André Minhorst" id="txtBeispielEditBox"
label="Beispieltextfeld" onChange="fctTextfeld" />
```

Das Resultat zeigt Abbildung 12.16.



Abbildung 12.16: Die EditText meldet das Überschreiten der zulässigen Zeichenanzahl

12.6.4 Kombinationsfelder I: Das comboBox-Element

Es gibt in Ribbons zwei Typen von Kombinationsfeldern: *comboBox* und *dropDown*. Beide haben Eigenschaften, die Sie vermutlich gerne in einem vereint sehen würden – warum, erfahren Sie in den folgenden Abschnitten. Zunächst lernen Sie dabei das *comboBox*-Element kennen. *comboBox*-Kombinationsfelder in Ribbons bieten wie ihre Formular-Pendants einen oder mehrere Werte zur Auswahl an. Das schreit natürlich nach dem dynamischen Füllen per VBA, zunächst aber soll ein einfaches Beispiel den grundlegenden Aufbau veranschaulichen.

```
<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui"
loadImage="LoadImage" onLoad="OnLoad">
  <ribbon startFromScratch="true">
    <tabs>
      <tab id="tabRibbonMitKombinationsfeld"
        ...
        <group id="grpBeispielgruppe2" label="Noch eine Beispielgruppe.">
          <comboBox id="cboBeispielkombinationsfeld"
            label="Kombinationsfeld:">
            <item id="i1" label="Eintrag 1"/>
            <item id="i2" label="Eintrag 2"/>
            <item id="i3" label="Eintrag 3"/>
          </comboBox>
        </group>
      </tab>
    </tabs>
  </ribbon>
</customUI>
```

Listing 12.10: Diese Ribbon-Definition erzeugt ein tab-Element mit einem Kombinationsfeld

Der Aufbau ähnelt dem eines Kombinationsfeldes in HTML. Das *comboBox*-Element schließt die enthaltenen *item*-Elemente ein, die Informationen über die zur Auswahl stehenden Einträge enthalten. Das obige XML-Dokument führt zum Ribbon in Abbildung 12.17.



Abbildung 12.17: Ein Ribbon mit einer zweiten Gruppe und einem einfachen Kombinationsfeld

Kombinationsfeld dynamisch füllen

Das dynamische Füllen eines einfachen *comboBox*-Elements erfolgt durch drei Callback-Funktionen. Die erste ermittelt die Anzahl der einzufügenden Einträge, die zweite die IDs und die dritte die Beschriftungen der Einträge. Der XML-Code für das Ribbon sieht wie im folgenden Listing aus, die für den Aufruf der Callback-Funktionen verantwortlichen Attribute sind fett markiert:

```
<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui">
  <ribbon startFromScratch="false">
    <tabs>
      <tab id="tabKombinationsfeldbeispiel"
        label="Kombinationsfeldbeispiel" visible="true">
        <group id="grpBeispielgruppe2" label="Noch eine Beispielgruppe.">
          <comboBox
            id="cboBeispielkombinationsfeld"
            label="Kombinationsfeld:"
            getItemCount="GetItemCount"
            getItemID="GetItemID"
            getItemLabel="GetItemLabel">
          </comboBox>
        </group>
      </tab>
    </tabs>
  </ribbon>
</customUI>
```

Listing 12.11: Ribbon-XML-Dokument für das dynamische Füllen eines Kombinationsfeldes

Die drei VBA-Callback-Funktionen sehen wie im folgenden Listing aus. Die Routine *GetItemCount* liefert die Anzahl der Einträge zurück, in diesem kleinen Beispiel sind dies drei. *GetItemID* ermittelt die IDs der anzuzeigenden Einträge, die hier aus dem Text »Testindex« und dem Wert des vom Ribbon gelieferten Index zusammengestellt werden. Fehlt noch die Beschriftung, die in *GetItemLabel* auf ähnliche Weise wie in der Routine *GetItemID* entsteht. Das resultierende Kombinationsfeld sieht im ausgeklappten Zustand schließlich wie in Abbildung 12.18 aus.

```
Public Sub GetItemCount(control As IRibbonControl, ByRef count)
    count = 3
End Sub

Public Sub GetItemID(control As IRibbonControl, index As Integer, ByRef id)
    id = "Testindex" & index
End Sub

Public Sub GetItemLabel(control As IRibbonControl, index As Integer, _
    ByRef label)
    label = "Testlabel" & index
End Sub
```

Listing 12.12: Callback-Funktionen zum Füllen eines einfachen Kombinationsfeldes

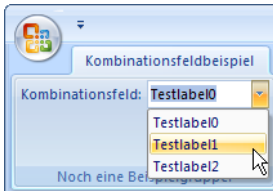


Abbildung 12.18: Kombinationsfeld mit dynamisch zusammengestelltem Inhalt

Im wirklichen Leben füllen Sie Steuerelemente wie Kombinationsfelder natürlich meist mit Daten aus Tabellen. Dies ist am einfachsten, wenn Sie den Inhalt der Tabelle direkt beim Ermitteln der Anzahl der einzufügenden Datensätze in einem Array zwischenspeichern. Auf dieses können Sie dann einfach über dessen Index zugreifen. Die VBA-Routinen für eine solche Konstellation sehen so aus:

```
Public Sub GetItemCount(control As IRibbonControl, ByRef count)
    Dim db As DAO.Database
    Dim rst As DAO.Recordset
    Dim i As Integer
    Set db = CurrentDb
    Set rst = db.OpenRecordset("SELECT KontaktID, Nachname & ', ' & " _
        & "Vorname AS Kontakt FROM tblKontakte", dbOpenDynaset)
    Do While Not rst.EOF
        ReDim Preserve strKontakte(2, i + 1) As String
        strKontakte(0, i) = rst!KontaktID
        strKontakte(1, i) = rst!Kontakt
        rst.MoveNext
        i = i + 1
    Loop
    count = i
    rst.Close
    Set rst = Nothing
```

```

Set db = Nothing
End Sub

Public Sub GetItemID(control As IRibbonControl, index As Integer, ByRef id)
    id = strKontakte(0, index)
End Sub

Public Sub GetItemLabel(control As IRibbonControl, index As Integer, _
    ByRef label)
    label = strKontakte(1, index)
End Sub

```

Listing 12.13: Diese Callback-Routinen füllen ein Kombinationsfeld mit den Daten einer Tabelle

Weitere Features von Ribbon-Kombinationsfeldern

Gegenüber herkömmlichen Access-Kombinationsfeldern hat das *comboBox*-Element in Ribbons einen sehr interessanten Vorteil: Sie können damit jedem Eintrag ein Symbol zuweisen (siehe Abbildung 12.19).

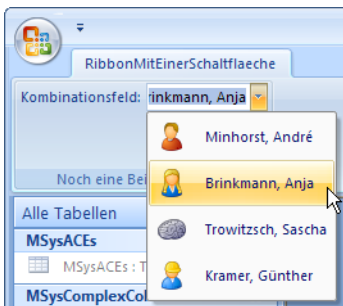


Abbildung 12.19: Ein comboBox-Steuerelement mit Symbolen

Und dies funktioniert ganz einfach – in diesem Fall für vier Bilddateien, die unter den Namen *1.png*, *2.png*, *3.png* und *4.png* im gleichen Verzeichnis wie die Datenbank gespeichert sind. Um obige Ansicht zu erhalten, müssen Sie dem *comboBox*-Element lediglich das Attribut *getItemImage* mit dem Namen der Callback-Funktion *getItemImage* zuweisen. Im gleichen Zug stellen Sie dann auch noch das Attribut *onChange* auf den Wert *OnChange* ein – gleich mehr dazu:

```

<comboBox
  id="cboBeispielkombinationsfeld"
  label="Kombinationsfeld:"
  getItemCount="GetItemCount"
  getItemID="GetItemID"

```

```
getItemLabel="GetItemLabel"  
getItemImage="GetItemImage"  
onChange="OnChange"  
>
```

Die folgende Routine gibt schließlich für jedes Element des Kombinationsfeldes eine Objektvariable mit einem Verweis auf eine Bilddatei zurück. Für das Einlesen der Bilddatei ist wiederum die Routine *LoadPicturePlus* verantwortlich – siehe auch Kapitel 11, »Bilder und binäre Dateien«.

```
Public Sub GetItemImage(control As IRibbonControl, index As Integer, _  
    ByRef image)  
    Set image = LoadPicturePlus(CurrentProject.Path & "\" & index + 1 _  
        & ".png")  
End Sub
```

Listing 12.14: Einlesen von Bilddateien für die Einträge eines *comboBox*-Elements per Callback-Routine

Das Attribut *OnChange* sorgt für den Aufruf der folgenden Routine. Diese wird nach jeder Neuauswahl eines Eintrags des *comboBox*-Elements aufgerufen und gibt lediglich den Inhalt des ausgewählten Eintrags aus, den das Ribbon mit dem Parameter *text* übergibt.

```
Public Sub OnChange(control As IRibbonControl, text As String)  
    MsgBox "Sie haben den Eintrag '" & text & "' ausgewählt."  
End Sub
```

Listing 12.15: Ausgabe des aktuellen Eintrags des *comboBox*-Steuerelements

Genau wie bei den Menüs von Access 2003 und älter kann man hier den Nachteil ausmachen, dass man nicht die ID des ausgewählten Eintrags auswerten kann; diese muss man in Abhängigkeit des angezeigten Wertes selbst ermitteln, denn die Eigenschaft *control.id* der *OnChange*-Prozedur liefert immer die ID des *comboBox*-Steuerelements.

12.6.5 Kombinationsfelder II: Das *dropDown*-Element

Warum gibt es zwei Typen von Kombinationsfeldern und welche Unterschiede weisen diese auf? Einen genauen Einblick gibt Tabelle 12.4 – dort finden Sie einen Vergleich der Attribute der beiden Kandidaten. Die wesentlichen Unterschiede sind, dass das *comboBox*-Element ein *onChange*-Attribut und das *dropDown*-Element dafür die Attribute *onAction*, *getSelectedItemID* und *getSelectedItemIndex* enthält.

Welche Vor- und Nachteile bringt das nun? Zwischen den beiden Ereignisseigenschaften *onChange* und *onAction* an sich fällt jedenfalls kein Unterschied auf; beide werden beim Auswählen eines anderen Eintrags ausgelöst. Den Unterschied entdeckt man

erst, wenn man sich die Syntax der passenden Callback-Routinen anschaut (siehe Tabelle 12.3): Während die *onChange*-Variante nur das Attribut *label* des ausgewählten Eintrags zurückliefert, wartet *onAction* mit den Attributen *index* und *id* auf.

Das spart natürlich Arbeit, wenn Sie ein Kombinationsfeld dynamisch mit Daten aus einer Tabelle bestücken und auf Basis der getroffenen Auswahl etwas mit den dahinter stehenden Datensätzen anfangen möchten.

Die weiteren Unterschiede beschränken sich auf zwei weitere zusätzliche Attribute des *dropDown*-Elements namens *getSelectedItemID* und *getSelectedItemIndex*. Beide dienen dazu, beim Anlegen des Steuerelements einen Eintrag vorauszuwählen – entweder auf Basis der ID oder des Index des Eintrags.

Das folgende Beispielelement enthält die Attribute *onAction* und *getSelectedItemIndex*:

```
<dropDown id="cboBeispielkombinationsfeld"
  label="Kombinationsfeld:" getItemCount="GetItemCount"
  getItemID="GetItemID" getItemLabel="GetItemLabel"
  getItemImage="GetItemImage" onAction="OnAction"
  getSelectedItemIndex="GetSelectedItemIndex">
```

Die beiden Routinen, die bei *onAction* beziehungsweise *getSelectedItemIndex* ausgelöst werden, sehen wie folgt aus:

```
Public Sub OnAction(control As IRibbonControl, selectedId As String, _
  selectedIndex As Integer)
  MsgBox "Der ausgewählte Eintrag hat die ID '" & selectedId _
    & "' und den Index '" & selectedIndex & "'."
End Sub
```

Listing 12.16: Anzeigen der ID und des Index eines neu ausgewählten Eintrags eines *dropDown*-Steuerelements

```
Public Sub GetSelectedItemIndex(control As IRibbonControl, ByRef index)
  index = 3
End Sub
```

Listing 12.17: Festlegen eines Eintrags eines *dropDown*-Elements per *Index*-Wert (der *Index* ist in diesem Fall immer nullbasiert)

Sie können die Neuabfrage der Vorauswahl mit *GetSelectedItemIndex* übrigens auch zur Laufzeit mit der Methode *invalidateControl* erzwingen (siehe auch Abschnitt 12.9.1).

comboBox oder dropDown?

Diese Entscheidung ist leicht zu fällen: Bezieht das Kombinationsfeld seine Daten aus einer Tabelle und sollen Aktionen auf Basis dieser Einträge in Zusammenhang mit den

zugrunde liegenden Daten ausgeführt werden oder ist eine automatische Voreinstellung notwendig, ist das *dropDown*-Element erste Wahl. Sollen einfach nur Texte zur Auswahl stehen und diese nach der Auswahl weiterverarbeitet werden, setzen Sie das *comboBox*-Steuerelement ein.

12.6.6 Umschaltflächen

Mit Umschaltflächen lassen sich Stati festlegen – etwa, um Funktionen zu aktivieren oder zu deaktivieren (aktivierte Umschaltflächen sind standardmäßig orange).

In einem Ribbon sieht die Definition eines einfachen *toggleButton*-Elements so aus:

```
<toggleButton id="tglUmschaltflaeche" label="Umschaltfläche"
  onAction="OnToggleAction" getPressed="GetPressed"/>
```

Die passende Routine für das Attribut *onAction* stellt sich wie folgt dar:

```
Public Sub OnToggleAction(control As IRibbonControl, pressed As Boolean)
  If pressed = True Then
    MsgBox "Funktion aktiviert"
  Else
    MsgBox "Funktion deaktiviert"
  End If
  cancelDefault = False
End Sub
```

Listing 12.18: Beim Klick auf ein *toggleButton*-Element wertet diese Routine den Parameter *pressed* aus

Möglicherweise fällt Ihnen auf, dass hier nicht einfach die großgeschriebene Variante des Attributs als Prozedurname herhält. Das ist zwingend notwendig, wenn verschiedenartige Steuerelemente die gleiche Callback-Funktion aufrufen, die Syntax sich aber unterscheidet.

In diesem Fall verwendet man einfach unterschiedliche Funktionsnamen, die etwa den Namen oder die Bezeichnung des Steuerelements enthalten. Den Status des *toggleButton*-Steuerelements legt die folgende Routine fest.

Sie weist dem Parameter *returnvalue* entweder den Wert *True* oder *False* zu, wobei *True* dem gedrückten Zustand entspricht.

```
Public Sub GetPressed(control As IRibbonControl, ByRef returnvalue)
  returnvalue = True
End Sub
```

Listing 12.19: Ob ein *toggleButton*-Element beim Anlegen gedrückt oder nicht gedrückt ist, legt die durch *getPressed* ausgelöste Routine fest

12.6.7 Galerien

Noch mehr Komfort als die beiden Kombinationsfeldtypen liefert das *gallery*-Element. Es kann verschiedene Einträge mit Symbolen anzeigen und das auch noch zweidimensional. Ein ganz einfaches Beispiel sieht wie in Abbildung 12.20 aus, den passenden XML-Code für dieses Element finden Sie hier:

```
<gallery id="galBeispiel" columns="2" rows="2" label="Tolle Items">
  <item id="id1" label="item1"/>
  <item id="id2" label="item2"/>
  <item id="id3" label="item3"/>
  <item id="id4" label="item4"/>
</gallery>
```

An diesem Beispiel erkennen Sie schnell, dass Access die Elemente zuerst von links nach rechts und dann von oben nach unten anordnet. Natürlich hat das *gallery*-Element noch einiges mehr zu bieten – etwa das Einfügen von Bilddateien wie in Abbildung 12.21.



Abbildung 12.20: Eine Galerie mit vier einfachen Einträgen



Abbildung 12.21: Eine Galerie mit Symbolen

Dazu ist der XML-Code aus folgendem Listing notwendig, wobei Sie für das Element *customUI* noch das Attribut *loadImage* so einstellen müssen, dass dieses eine passende Routine zum Laden der in den einzelnen Items angegebenen Bilder aufruft.

```
<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui"
loadImage="LoadImage">
...
<gallery id="galBeispiel" columns="2" rows="4"
label="Heute im Angebot"
description="Wählen Sie ein Getränk aus.">
  <item id="id1" label="Limonade" image="lemonade_bottle.png"/>
  <item id="id2" label="Rotwein (Flasche)" image="wine_red_bottle.png"/>
  <item id="id3" label="Rotwein (Glas)" image="wine_red_glass.png"/>
  <item id="id4" label="Weißwein (Glas)" image="wine_white_glass.png"/>
  <item id="id5" label="Bier (Flasche)" image="beer_bottle.png"/>
  <item id="id6" label="Bier (frisch gezapft)"
image="beer_glass.png"/>
  <item id="id7" label="Cocktail" image="cocktail.png"/>
  <item id="id8" label="Kaffee" image="cup.png"/>
</gallery>
</customUI>
```

Listing 12.20: Dieses gallery-Element zeigt acht Items mit passenden Symbolen an

In diesem Fall heißt die Routine *LoadImageGallery* und sieht wie folgt aus:

```
Public Sub LoadImageGallery(control, ByRef image)
    Set image = LoadPicturePlus(CurrentProject.Path & "\\Getraenke\" _
        & control)
End Sub
```

Listing 12.21: Die Callback-Routine *LoadImageGallery* ist für das Laden von Bildern aus einem Unterverzeichnis des Anwendungsverzeichnisses verantwortlich

Besonderheiten des gallery-Elements

Das *gallery*-Element ist insbesondere mit den Kombinationsfeld-Steuerelementen *comboBox* und *dropDown* zu vergleichen. Der erste Vorteil ist die Möglichkeit, Elemente in mehreren Spalten und Zeilen anzuzeigen, der zweite wäre, wenn es denn funktionieren würde, das Attribut *invalidateContentOnDrop*.

Dieses verspricht laut *customUI.xsd* das Ausführen von Callback-Funktionen beim Aufklappen der Galerie, was aber zum Zeitpunkt der Erstellung dieses Kapitels nicht möglich war. Weitere interessante Attribute sind folgende:

- ▶ *itemHeight*: Gibt die Höhe des *item*-Elements an.
- ▶ *itemWidth*: Gibt die Breite des *item*-Elements an.

Die Ereignisseigenschaft zum Festlegen einer Callback-Funktion, die beim Klicken auf eines der Elemente ausgelöst wird, heißt beim *gallery*-Element *onAction* und hat folgende Syntax:

```
Sub OnAction(control As IRibbonControl, selectedId As String, selectedIndex As Integer)
```

12.6.8 Menüs (menu)

Menüs ähneln den von älteren Office-Versionen bekannten Untermenüs der Menüleiste. Dabei kann man einige unterschiedliche Steuerelemente anlegen, angezeigt werden allerdings stets nur normale Menüeinträge. Das folgende Ribbon-XML-Dokument erzeugt beispielsweise das Menü aus Abbildung 12.22.

```
<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui"
loadImage="LoadImage" onLoad="OnLoad">
  <ribbon startFromScratch="true">
    <tabs>
      <tab idMso="TabCreate" visible="false" />
      <tab id="tabRibbonMitEinerSchaltflaeche"
label="RibbonMitEinerSchaltflaeche" visible="true">
        <group id="grpBeispielgruppe" label="Beispielgruppe">
          <menu id="mnuBeispielmenue" label="Beispielmenü">
            <button id="btnButton" label="Schaltfläche" image="" />
            <checkbox id="chkCheckbox" label="Kontrollkästchen" />
            <toggleButton id="tglToggleButton" label="Umschaltfläche" />
          </menu>
        </group>
      </tab>
    </tabs>
  </ribbon>
</customUI>
```

Listing 12.22: Beispiel für ein Menü, dessen verschiedenartige Steuerelemente jedoch alle in einem Standardmenüeintrag enden

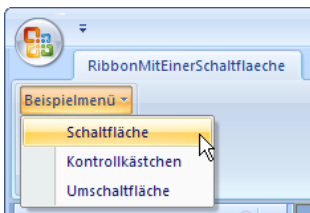


Abbildung 12.22: Das menu-Steuerelement mit einigen Untersteuerelementen

Funktionen beschreiben mit dem description-Attribut

Das *menu*-Element ist eines der wenigen, in denen sich das *description*-Attribut für Steuerelemente bemerkbar macht. In üblichen *button*-Elementen zeigt Access diese einfach

nicht an – in *button*- und weiteren Elementen innerhalb eines *menu*-Elements hingegen schon. Wie das aussehen kann, zeigt Abbildung 12.23. Der passende XML-Code sieht wie folgt aus:

```
<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui" loadI
mage="mnuSportartAuswaehlen_LoadImage" onLoad="OnLoad">
  <ribbon startFromScratch="true">
    <tabs>
      <tab id="tabSport" label="Sport" visible="true">
        <group id="grpSportarten" label="Sportarten">
          <menu id="mnuSportartAuswaehlen"
            label="Sportart auswählen"
            itemSize="large">
            <button id="btnTennis" label="Tennis-Abteilung"
              description="Tennis ist eine schöne Sportart."
              image="tennis_ball.png"/>
            <button id="btnBaseball" label="Baseball-Abteilung"
              description="Baseball auch." image="baseball.png"/>
            <button id="btnBasketball" label="Basketball-Abteilung"
              description="Und Basketball ist erstmal toll!"
              image="basketball.png"/>
            ...
          </menu>
        </group>
      </tab>
    </tabs>
  </ribbon>
</customUI>
```

Listing 12.23: Diese XML-Definition zeigt beispielhaft, wie man Menüelemente mit Beschreibungen versieht

Wichtig ist hier, dass Sie *itemSize* des *menu*-Elements auf *large* einstellen.

Wenn Sie eine optische Trennung zwischen zwei Einträgen vornehmen möchten, setzen Sie das *menuSeparator*-Element ein. Sie können es mit oder ohne Text verwenden, wobei Sie einen Text für das Attribut *title* eintragen. Folgender Codeschnipsel verursacht beispielsweise die beiden Trenner aus Abbildung 12.24:

```
<menu id="mnu" label="Menü">
  <menuSeparator id="mns" title="Ich bin ein menuSeparator"/>
  <button id="btn2" label="Button 2"/>
  <button id="btn3" label="Button 3"/>
  <menuSeparator id="mns1" title="Ich auch."/>
  <button id="btn4" label="Button 4"/>
  <button id="btn5" label="Button 5"/>
</menu>
```



Abbildung 12.23: Im menu-Steuerelement lassen sich neben Symbolen auch Beschreibungstexte zu einem Steuerelement anzeigen



Abbildung 12.24: Ein Menü mit zwei textbepfunden Trennlinien

12.6.9 Splitbuttons (splitButton)

Ein *splitButton*-Steuerelement ist ein kombiniertes Schaltflächen/Menü-Steuerelement, wie es auch etwa im Start-Ribbon für die Auswahl der Ansicht verwendet wird. Dabei kann man auf das Steuerelement selbst oder auf einen der Menüeinträge klicken, um eine bestimmte Aktion auszuführen.

Dementsprechend sieht das XML-Konstrukt für die Realisierung eines solchen Split-Buttons aus: Es besteht aus einem *splitButton*-Element, das ein *button*- oder *toggleButton*-Element und ein *menu*-Element umschließt. Der XML-Code aus dem folgenden Listing veranschaulicht die für das *splitButton*-Steuerelement aus Abbildung 12.25 notwendigen Elemente:

```
<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui"
loadImage="LoadImageSplitMenu" onLoad="OnLoad">
  <ribbon startFromScratch="true">
```

```
<tabs>
  <tab idMso="TabCreate" visible="false" />
  <tab id="tabRibbonMitEinerSchaltflaeche"
    label="RibbonMitEinerSchaltflaeche" visible="true">
    <group id="grpBeispielgruppe" label="Beispielgruppe">
      <splitButton id="spbSplitButton" size="large">
        <button id="cmdButton" label="Rot"
          image="trafficlight_red.png"/>
        <menu id="mnuMenu" label="Beispielmenü" itemSize="large">
          <button id="btnButton1" label="Rot"
            image="trafficlight_red.png"/>
          <button id="btnButton2" label="Rot-Gelb"
            image="trafficlight_red_yellow.png"/>
          <button id="btnButton3" label="Grün"
            image="trafficlight_green.png"/>
        </menu>
      </splitButton>
    </group>
  </tab>
</tabs>
</ribbon>
</customUI>
```

Listing 12.24: Dieses XML-Dokument erzeugt das splitButton-Element aus Abbildung 5.25



Abbildung 12.25: Das splitButton-Steuerelement zeigt ständig eine Schaltfläche und bietet andere zur Auswahl an

12.6.10 Gruppendialog anzeigen

Einige eingebaute Tabs enthalten Gruppen, deren Beschriftungsfeld im rechten Bereich einen kleinen Pfeil anzeigt, mit dem man weiterführende Dialoge öffnen oder beliebigen

anderen Code aufrufen kann. Dies realisieren Sie mit dem *dialogBoxLauncher*-Element. Der folgende Codeschnipsel zeigt, wie es funktioniert:

```
<group id="grpBeispielgruppe" label="Beispielgruppe">
  <dialogBoxLauncher>
    <button id="dbxDialogOeffnen" onAction="DialogOeffnen"/>
  </dialogBoxLauncher>
</group>
```

Heraus kommt dabei die kleine Schaltfläche unten rechts in Abbildung 12.26. Den Code zum Aufrufen des entsprechenden Dialogs – etwa eines Formulars – bringen Sie in einer Routine namens *DialogOeffnen* unter, die die gleiche Prozedurdeklaration wie die *OnAction*-Callbackfunktion von Schaltflächen hat.



Abbildung 12.26: Eine Gruppe mit einem dialogBoxLauncher-Element

12.6.11 Trennstrich (separator)

Das *separator*-Element zwischen zwei weiteren Elementen sorgt für die Anzeige eines Trennstriches. Der folgende Codeschnipsel sorgt für die Anzeige der beiden Beschriftungen und der Trennlinie aus Abbildung 12.27

```
<group id="grpBeispielgruppe" label="Beispielgruppe">
  <labelControl id="Beschriftung1" label="Beschriftung1"/>
  <separator id="Separator"/>
  <labelControl id="Beschriftung2" label="Beschriftung2"/>
</group>
```

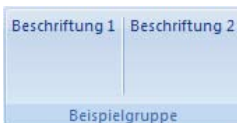


Abbildung 12.27: Beispiel für einen Trennstrich (separator)

12.7 Weitere Anpassungen des Ribbons

Die folgenden Abschnitte zeigen, wie Sie das Ribbon feintunen – etwa durch Hinzufügen passender Tastenkombinationen.

12.7.1 Tastenkombinationen

Genau wie bei älteren Access-Versionen können Sie auch in Access 2007 Tastenkombinationen verwenden, um Menübefehle aufzurufen. Früher fügte man dazu ein Kaufmanns-Und (&) vor dem Buchstaben der Beschriftung ein, der zum Aktivieren des jeweiligen Elements dienen sollte. Heute ist alles anders: Ribbon-Elemente blenden Buchstaben ein, sobald Sie auf die *Alt*-Taste klicken.

Von da an können Sie durch Betätigen der zu den angezeigten Buchstaben passenden Tasten Aktionen wie das Einblenden eines Ribbon, das Auslösen eines Steuerelements und mehr bewirken (mehr dazu in Kapitel 1, »Warum Access 2007?«).

Natürlich können Sie auch für benutzerdefinierte Elemente diese so genannten Keytips festlegen: Dazu verwenden Sie das Attribut *keytip*, das einen Wert bestehend aus ein bis drei Buchstaben oder Zahlen erwartet. Wenn Sie keinen Wert für das Attribut *keytip* angeben, verwendet Access vorgegebene Werte wie *Y* für ein benutzerdefiniertes Ribbon und *Y1*, *Y2*, ... für die darin enthaltenen Steuerelemente.

Fügen Sie einmal wie im folgenden Code eine Tastenkombination für das Ribbon selbst und für die oben erstellte Schaltfläche hinzu (fett markiert):

```
<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui"
loadImage="LoadImage" onLoad="OnLoad">
  <ribbon startFromScratch="false">
    <tabs>
      <tab id="tabKeytipBeispiel" label="Keytips-Beispielribbon"
visible="true" keytip="KB">
        <group id="grpBeispielgruppe" label="Beispielgruppe">
          <button id="btnBeispielschaltflaeche" keytip="BS"
label="Beispielschaltfläche" onAction="Beispielfunktion"
screentip="Klicken Sie doch einmal auf die Beispielschaltfläche!"
image="ribbon.png" size="large"/>
        </group>
      </tab>
    </tabs>
  </ribbon>
</customUI>
```

Listing 12.25: Die fett gedruckten Stellen zeigen Zuweisungen von Tastenkombinationen zu verschiedenen Elementen

Access zeigt dann nach Betätigen der *Alt*-Taste die für das *tab*-Element definierte Tastenkombination an (siehe Abbildung 12.28). Nach Eingabe der Buchstaben *KB* aktiviert Access das gewünschte Ribbon und blendet auch hier die Tastenkombination ein. Gibt man diese ein, führt Access die Funktion der so »gedrückten« Schaltfläche aus (siehe Abbildung 12.29).

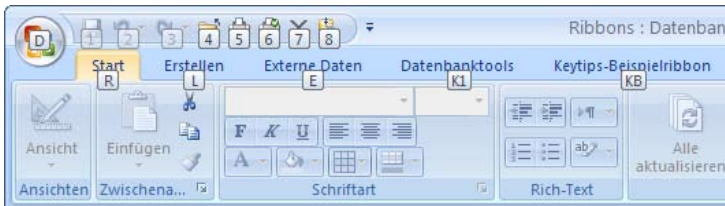


Abbildung 12.28: Nach dem Betätigen der Alt-Taste erscheinen zunächst die Tastenkombinationen für die Auswahl eines Ribbons ...

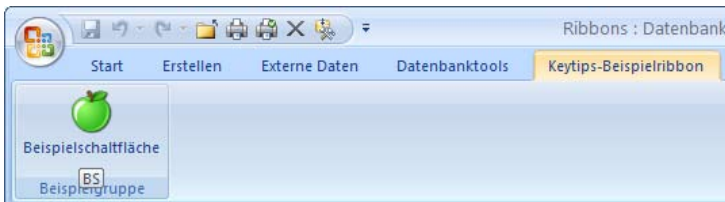


Abbildung 12.29: ... und Access aktiviert das per Tastenkombination ausgewählte Ribbon und zeigt dessen Tastenkombinationen an. Die Eingabe von »BS« hätte nun die gleiche Wirkung wie ein Mausklick auf die Schaltfläche.

12.7.2 Alle Ribbons ausblenden

Wenn Sie vor dem Anlegen eines benutzerdefinierten Ribbons für eine eigene Anwendung alle anderen Ribbons ausblenden möchten, setzen Sie das Attribut *startFromScratch* des *ribbon*-Elements auf den Wert *true*:

```
<ribbon startFromScratch="true ">
```

Access zeigt nun nur noch das in den Access-Optionen unter *Aktuelle Datenbank* | *Multifunktionsleisten- und Symbolleistenoptionen* | *Name der Multifunktionsleiste* angegebene Ribbon an.

Zusätzlich hat Access auch noch das Office-Menü ausgedünnt: Dort finden Sie nur noch die Schaltflächen zum Anlegen einer neuen Datenbank, zum Öffnen einer bestehenden Datenbank, zum Speichern und zum Schließen. Allerdings lassen sich noch die Datenbankoptionen anzeigen (siehe Abbildung 12.30).

12.7.3 Ribbon-Leiste minimieren

Die Leiste mit den Tabs können Sie auch minimieren. Dazu wählen Sie aus dem Kontextmenü des Ribbons oder über den Anpassungspfeil der Schnellstartleiste den Eintrag *Multifunktionsleiste minimieren* aus. Die Tabs blenden sich dann erst nach einem Klick auf

die Registerreiter ein. Sie sparen damit eine Menge Raum, den Sie besser für Formulare und Berichte verwenden können.

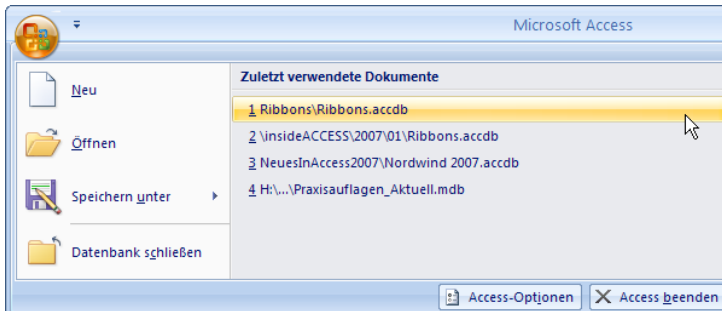


Abbildung 12.30: Das Setzen von `startFromScratch` auf `true` führt zum Eindampfen des Office-Menüs

12.7.4 Ein `tab`-Element ein- und ausblenden

Möglicherweise möchten Sie ein einzelnes `tab`-Element ein- oder ausblenden. In diesem Fall fügen Sie dem XML-Dokument ein einzelnes `tab`-Element hinzu, das den Namen des betroffenen Elements und den Sichtbarkeitsstatus enthält.

Das folgende Element blendet das `tab`-Element `Anpassen` aus, wenn Sie es in die `tabs`-Auflistung einfügen:

```
<tab idMso="TabCreate" visible="false" />
```

Hier wie auch beim Anpassen von Gruppen stellen Sie sich vielleicht die Frage, wie Sie an all die Konstanten für das Attribut `idMso` kommen sollen. Eine Übersicht darüber finden Sie in einer Beispieldatenbank, die eine Tabelle namens `tblAccessRibbonControls` mit allen Ribbon-Elementen von Access enthält [3].

12.7.5 Eine Gruppe ein- und ausblenden

Das funktioniert natürlich auch mit einer Gruppe. Mit folgendem Ribbon-XML-Dokument sorgen Sie etwa dafür, dass die Gruppe zum Wechseln der Ansicht aus dem `Start`-Ribbon ausgeblendet wird:

```
<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui"
loadImage="LoadImageSplitMenu" onLoad="OnLoad">
  <ribbon>
    <tabs>
      <tab idMso="TabHomeAccess">
        <group idMso="GroupViews" visible="false" />
      </tab>
```

```

    </tabs>
  </ribbon>
</customUI>

```

Listing 12.26: Wenn man den Namen einer Gruppe und des übergeordneten Tabs kennt, kann man diese mit dem `visible`-Attribut ausblenden

Die Tabs wie auch andere Steuerelemente oder Gruppen können Sie dynamisch per Code ein- und ausblenden, wenn Sie eine Callback-Funktion und das Attribut `getVisible` einsetzen:

```
<tab idMso="TabCreate" getVisible="fuVisible" />
```

Die Sichtbarkeit wird dann in der Callback-Funktion gesteuert, deren Abfrage Sie mit der Methode `InvalidateControl` (siehe Abschnitt 12.9.1) jederzeit erzwingen können.

12.7.6 Ein Steuerelement ein- und ausblenden

Schließlich lässt sich das Ein- und Ausblenden bis hin zum einzelnen Steuerelement fortsetzen – aber nur für benutzerdefinierte Elemente. Bei den eingebauten Ribbons ist mit dem Anpassen von `group`-Elementen Schluss; Steuerelemente lassen sich nur in komplett selbst erstellten Ribbons ein- und ausblenden. Dazu fügen Sie einfach das Attribut `visible` hinzu und geben ihm den Wert »`false`«.

12.7.7 Eingebaute Steuerelemente aktivieren und deaktivieren

Das Aktivieren und Deaktivieren eingebauter Steuerelemente erfolgt ebenfalls im Ribbon-XML-Dokument, allerdings nicht unterhalb des `ribbon`-Elements. Hierfür gibt es unterhalb des `customUI`-Elements eine eigene Auflistung namens `commands`. Die Elemente heißen passend `command` und werden anhand des Attributs `idMso` eindeutig vorhandenen Steuerelementen zugeordnet. Der folgende Beispielcode zeigt, wie Sie etwa die Filterschaltfläche deaktivieren:

```

<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui">
  <commands>
    <command idMso="FiltersMenu" enabled="false"/>
  </commands>
</customUI>

```

Die `msoID` entnehmen Sie der in [3] enthaltenen Tabelle der Steuerelemente. Offensichtlich arbeitet diese Eigenschaft aber nicht konsistent, denn etwa die `Suchen`-Schaltfläche ließ sich nicht aktivieren; Access lieferte aber für folgende Zeile auch keinen Fehler:

```
<command idMso="GroupFindAccess" enabled="false"/>
```

12.7.8 Eingebaute Steuerelemente mit neuen Funktionen belegen

Sie können die eingebauten Steuerelemente (allerdings nur *button*-, *toggleButton*- und *checkBox*-Elemente) mit eigenen Funktionen versehen. Die folgende Definition sorgt etwa dafür, dass ein Klick auf die *Filtern*-Schaltfläche die VBA-Routine *OnFilternAction* auslöst.

```
<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui">
  <commands>
    <command idMso="FiltersMenu"
      onAction="OnFilternAction"/>
  </commands>
</customUI>
```

12.7.9 Sonderzeichen in Ribbon-Texten

Wenn Sie in Attribute wie *label*, *description*, *screenTip* oder *superTip* Sonderzeichen einfügen möchten, müssen Sie die in XML-Dokumenten übliche Schreibweise verwenden. Dazu suchen Sie sich den ASCII-Wert des Sonderzeichens heraus und verpacken diesen – etwa für einen Zeilenumbruch (ASCII-Wert 13) – wie folgt: ``. Falls Sie einmal einen Teil des auszugebenden Textes in Anführungszeichen setzen möchten, verwenden Sie keine doppelten Anführungszeichen wie in VBA-Zeichenketten, sondern die Zeichenfolge `"`.

12.7.10 Einen Eintrag zum Office-Menü hinzufügen

Wenn Sie den Benutzern Ihrer Anwendung auch das Office-Menü als Ausgangspunkt für den Aufruf von Funktionen anbieten möchten, können Sie auch dort Einträge hinzufügen. Der passende Bereich ist direkt unterhalb des *ribbon*-Elements, also auf der gleichen Ebene wie die *tabs*-Auflistung zu finden und heißt *officeMenu*. Unterhalb dieses Elements können Sie verschiedene Steuerelemente wie im folgenden Beispiel anlegen. Wenn das Steuerelement an einer bestimmten Position eingefügt werden soll, können Sie mit dem Attribut *insertAfterMso* oder *insertBeforeMso* festlegen, vor oder nach welchem eingebauten Steuerelement das neue Element Platz finden soll. Natürlich können Sie auch hier Symbole hinzufügen (siehe Abbildung 12.31).

```
<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui"
loadImage="LoadImage">
  <ribbon startFromScratch="true">
    <officeMenu>
      <button id="btn" label="Neuer Eintrag"
        image="heart.png" insertAfterMso="FileNewDatabase"/>
    </officeMenu>
  </ribbon>
</customUI>
```

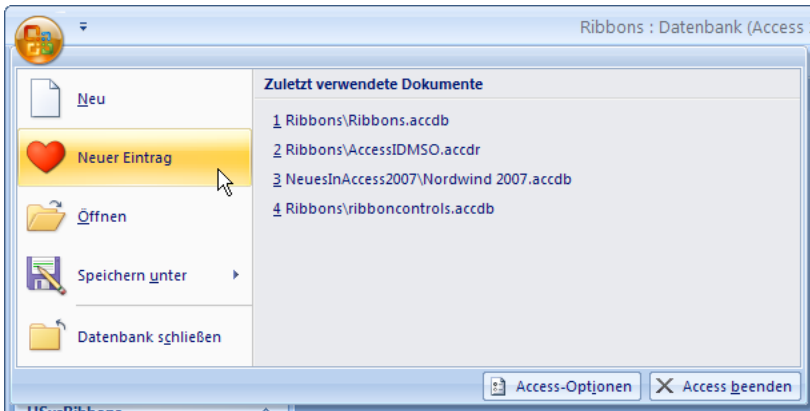


Abbildung 12.31: Ein neuer Eintrag im Office-Menü

12.7.11 Einträge des Office-Menüs ausblenden

Die Einträge des Office-Menüs wie Neu, Öffnen, Speichern und so weiter möchten Sie den Benutzern Ihrer Anwendung unter Umständen nicht zur Verfügung stellen, und ganz sicher nicht den Dialog mit den Access-Optionen. Die einzelnen Einträge des Office-Menüs lassen sich mit XML-Definitionen wie folgt ausblenden:

```
<officeMenu>
  <button idMso="FileNewDatabase" visible="false"/>
  <button idMso="FileOpenDatabase" visible="false"/>
  <button idMso="FileSaveAs" visible="false"/>
  <button idMso="SaveObjectAs" visible="false"/>
  <button idMso="FileSave" visible="false"/>
  <button idMso="FileBackupDatabase" visible="false"/>
  <button idMso="FileCloseDatabase" visible="false"/>
  <splitButton idMso="FileSaveAsMenuAccess" visible="false"/>
</officeMenu>
```

Wenn Sie alle Elemente inklusive der Schaltfläche *Access-Optionen* entfernen möchten, stellen Sie dies in eben jenem Optionen-Dialog ein. Der richtige Bereich heißt logischerweise *Aktuelle Datenbank* und die passende Option lautet *Vollständige Menüs zulassen* (siehe Abbildung 12.32). Wenn Sie diese Option deaktivieren, sieht das Office-Menü wie in Abbildung 12.33 aus.

Der Bereich *Zuletzt verwendete Dokumente* lässt sich übrigens nicht ausblenden.

Wenn Sie die Optionen einer auf diese Weise angepassten Datenbank wieder einblenden möchten, müssen Sie die Datenbank bei gedrückter Umschalt-Taste öffnen – Access setzt die Einstellungen unter *Aktuelle Datenbank* dann außer Kraft.

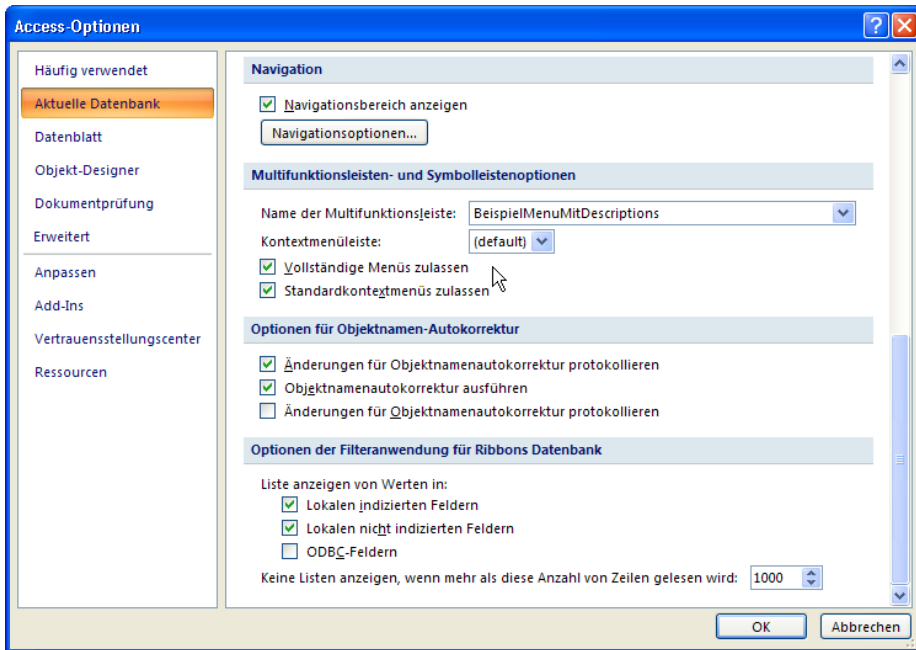


Abbildung 12.32: Der Optionen-Dialog bietet unter anderem die Möglichkeit, die Benutzeroberfläche einer Access-Anwendung anzupassen



Abbildung 12.33: Das Deaktivieren der Option »Vollständige Menüs zulassen« sorgt für ein auf das Notwendigste reduziertes Office-Menü

12.7.12 Einen Eintrag zur Schnellzugriffsleiste hinzufügen

Wenn Sie der Schnellzugriffsleiste (*Quick Access Toolbar, QAT*) eine benutzerdefinierte Schaltfläche hinzufügen möchten, müssen Sie zunächst das Attribut *startFromScratch* auf *true* einstellen und damit alle eingebauten Ribbons ausblenden – anderenfalls funktioniert dies nicht.

Anschließend verwenden Sie den unter dem Element *ribbon* befindlichen Bereich *qat* und fügen dort zunächst die Liste *documentControls* und dann die gewünschten Steuerelemente hinzu. Das folgende Beispiel sorgt etwa für die Anzeige der Schaltflächen aus Abbildung 12.34:

```

<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui"
loadImage="LoadImage">
  <ribbon startFromScratch="true">
    <qat>
      <documentControls>
        <button id="btnHerz" label="Herz" image="heart.png"/>
        <button id="btnApfel" label="Apfel" image="apple.png"/>
      </documentControls>
    </qat>
  </ribbon>
</customUI>

```

Listing 12.27: Hinzufügen benutzerdefinierter Schaltflächen zur Schnellzugriffsleiste

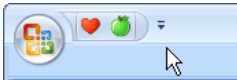


Abbildung 12.34: Die Schnellzugriffsleiste mit benutzerdefinierten Schaltflächen

12.8 Ribbons für Formulare und Berichte

Genau wie in Access 2003 und älter können Sie in Access 2007 Formularen und Berichten Ribbon-Tabs zuweisen. Dazu weisen Sie einfach der Eigenschaft *Name der Multifunktionsleiste* eines der verfügbaren benutzerdefinierten Ribbons zu (siehe Abbildung 12.35). Zu beachten ist hier, dass das Zuweisen eines Ribbon zu Formularen, die als modaler Dialog geöffnet sind, keine Wirkung hat – das Ribbon wird einfach nicht angezeigt. Als modale Dialoge gelten dabei Formulare, deren Eigenschaft *Popup* den Wert *Ja* hat und/oder die mit *DoCmd.OpenForm* mit dem Parameter *WindowMode:=acDialog* geöffnet wurden. Für Berichte gilt das Gleiche, auch hier weisen Sie der Eigenschaft *Name der Multifunktionsleiste* den passenden Eintrag zu und modal geöffnete Berichte versagen die Anzeige der angegebenen Multifunktionsleiste. Auch in der Layoutansicht von Berichten zeigt sich eine angegebene Multifunktionsleiste nicht, wohl aber in der neuen Berichtsansicht.

12.9 XML-Dokument mit Application.LoadCustomUI laden

Neben dem Speichern des Ribbon-XML-Dokuments in der Tabelle *USysRibbons* und dem Festlegen der beim Start einzulesenden Ribbon-Definition gibt es noch eine weitere Möglichkeit, Ribbons anzulegen.

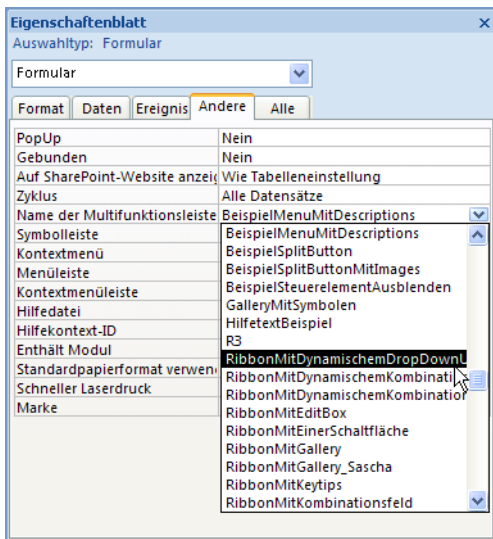


Abbildung 12.35: Auswählen eines benutzerdefinierten Ribbons für ein Formular

Diese erfordert das Vorhandensein des XML-Dokuments in einem beliebigen Format – wie oben beschrieben im Memofeld der Tabelle *USysRibbons* oder einer anderen Tabelle gespeichert oder auch in Form einer externen Textdatei. Sie müssen nur sicherstellen, dass das XML-Dokument in einer String-Variablen gespeichert vorliegt. Anschließend können Sie dann die Methode *LoadCustomUI* des *Application*-Objekts verwenden, um ein Ribbon mit der angegebenen Definition zur Liste der verfügbaren Ribbons hinzuzufügen.

Die folgende Routine liest zunächst mit der Funktion *TextdateiLesen* (siehe Listing 12.29) die in einer Datei gespeicherte Ribbon-Definition ein und weist diese dann per *LoadCustomUI*-Methode der Liste der benutzerdefinierten Ribbons der aktuellen Datenbank zu. Diese können Sie dann in den Access-Optionen unter *Aktuelle Datenbank\Multifunktionsleisten- und Symbolleistenoptionen\Name der Multifunktionsleiste* auswählen.

```
Public Sub DynamischeZuweisungDatei()
    Dim strRibbon As String
    strRibbon = TextdateiLesen(CurrentProject.Path & "\Ribbon.xml")
    Application.LoadCustomUI "RibbonAusDatei", strRibbon
End Sub
```

Listing 12.28: Zuweisen einer Ribbon-Definition per LoadCustomUI

```
Public Function TextdateiLesen(strDateiname As String)
    Dim lngDatei As Long
```

```

Dim strText As String
lngDatei = FreeFile
Open strDateiname For Input As lngDatei
strText = Input$(LOF(1), lngDatei)
Close lngDatei
TextdateiLesen = strText
End Function

```

Listing 12.29: Funktion zum Einlesen von Textdateien

Anschließend können Sie die Ribbon-Definition jederzeit durch eine neu erstellte Textdatei ersetzen – solange der Parameter *CustomUiName* der Methode *LoadCustomUI* gleich bleibt (hier *RibbonAusDatei*), aktualisiert sich das Ribbon dann automatisch.

12.9.1 Dynamisches Aktualisieren des Ribbons

In bestimmten Situationen möchten Sie vielleicht das Aussehen des Ribbons beziehungsweise der enthaltenen Steuerelemente dynamisch verändern.

Normalerweise erhält man keinen Zugriff auf die Ribbons wie es in früheren Versionen von Access bei den Symbolleisten per VBA möglich war. Statt dessen müssen Sie bereits beim Anlegen des Ribbons eine Objektvariable mit einem Verweis auf das Ribbon erstellen, über die Sie dann später auf das Ribbon zugreifen können.

Dazu verwenden Sie wiederum eine Callback-Funktion, die Sie diesmal dem Attribut *onLoad* des *customUI*-Elements der Ribbon-Definition zuweisen. Das *customUI*-Element sieht dann so aus, wobei die beim Laden aufzurufende Funktion *OnLoad* heißt:

```

<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui"
loadImage="LoadImage" onLoad="OnLoad">

```

In einen Standardmodul legen Sie dann zunächst eine globale Objektvariable an, die den Verweis auf das Ribbon speichern soll:

```
Dim objRibbon As IRibbonUI
```

Die folgende Funktion ist schließlich für das Setzen des Objektverweises verantwortlich:

```

Public Function OnLoad(ribbon As IRibbonUI)
Set objRibbon = ribbon
End Function

```

Listing 12.30: Speichern eines per Parameter übergebenen Objektverweises

Ist der Verweis gesetzt, können Sie fortan über die Variable *objRibbon* auf das Ribbon-Objekt zugreifen und eine seiner beiden Methoden aufrufen (siehe Abbildung 12.36). Zu beachten ist, dass keine der Methoden die Ereignisse *onLoad* oder *loadImages* auslöst:

- ▶ *Invalidate*: Initialisiert alle enthaltenen Steuerelemente neu.
- ▶ *InvalidateControl*: Initialisiert das als String-Parameter (*ControlID*) übergebene Steuerelement neu.

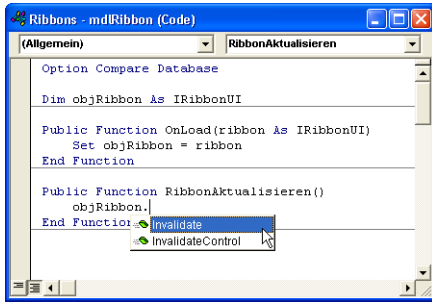


Abbildung 12.36: Der Objektverweis auf ein Ribbon liefert zwei Methoden

12.9.2 Beispiel: Abhängige Kontrollkästchen

Die beiden Kontrollkästchen des folgenden Beispiels für die *InvalidateControl*-Methode sollen eine Abhängigkeit aufweisen, bei der das zweite Kontrollkästchen bei markiertem ersten Kontrollkästchen aktiv ist und umgekehrt (siehe Abbildung 12.37).

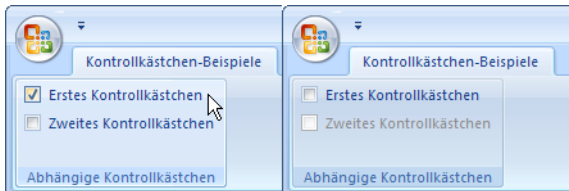


Abbildung 12.37: Das untere Kontrollkästchen ist aktiviert, wenn das obere Kontrollkästchen markiert ist und umgekehrt

Dazu ist zunächst eine passende XML-Definition des Ribbons erforderlich:

```
<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui"
onLoad="OnLoad">
  <ribbon startFromScratch="true">
    <tabs>
      <tab id="tabKontrollkaestchen" label="Kontrollkästchen-Beispiele"
visible="true">
        <group id="grpAbhaengigeKontrollkaestchen"
label="Abhängige Kontrollkästchen">
          <checkBox id="chk1" label="Erstes Kontrollkästchen"
```

```

        onAction="chk1_OnAction" getPressed="chk1_GetPressed"/>
        <checkBox id="chk2" label="Zweites Kontrollkästchen"
        getEnabled="chk2_GetEnabled"/>
    </group>
</tab>
</tabs>
</ribbon>
</customUI>

```

Listing 12.31: Ribbon-XML-Dokument für abhängige Kombinationsfelder

Zunächst muss das *customUI*-Element die Eigenschaft *onLoad* aufweisen und damit die Routine aus folgendem Listing aufrufen.

```

Sub OnLoad(ribbon As IRibbonUI)
    Set objRibbon = ribbon
    boolChk1 = True
End Sub

```

Listing 12.32: Setzen eines Objektverweises auf das Ribbon

Die beiden dort gefüllten Variablen *objRibbon* zum Speichern eines Verweises auf das Ribbon sowie *boolChk1* zum Speichern des Anfangswerts des ersten Kontrollkästchens legen Sie der Einfachheit halber als öffentliche Variablen an:

```

Public objRibbon As IRibbonUI
Public boolChk1 As Boolean

```

Beim Anlegen des Ribbons ruft Access auch sämtliche in *get...*-Attributen angegebenen Routinen auf.

Hier sorgt ein *getPressed*-Attribut dafür, dem ersten Kontrollkästchen den in der Variablen *boolChk1* gespeicherten Wert zuzuweisen und ein *getEnabled*-Attribut aktiviert oder deaktiviert das zweite Kontrollkästchen je nach dem Inhalt von *boolChk1*:

```

Sub chk1_GetPressed(control As IRibbonControl, ByRef pressed)
    pressed = boolChk1
End Sub

```

Listing 12.33: Diese Routine setzt in Abhängigkeit von *boolChk1* einen Haken in das erste Kontrollkästchen

```

Sub chk2_GetEnabled(control As IRibbonControl, ByRef enabled)
    enabled = boolChk1
End Sub

```

Listing 12.34: Aktivieren oder deaktivieren des zweiten Kontrollkästchens in Abhängigkeit von *boolChk1*

Schließlich fehlt noch ein wenig Aktion – und dafür sorgt die Routine, die beim Anklicken des ersten Kontrollkästchens ausgelöst wird. Sie sorgt dafür, dass *boolChk1* den Wert des ersten Kontrollkästchens erhält und das zweite Kontrollkästchen aktualisiert wird – indem es mit der Methode *InvalidateControl* für den erneuten Aufruf der Methode *GetEnabled* sorgt.

```
Sub chk1_OnAction(control As IRibbonControl, pressed As Boolean)
    boolChk1 = pressed
    objRibbon.InvalidateControl "chk2"
End Sub
```

Listing 12.35: Beim Klick auf das erste Kontrollkästchen werden der Inhalt von *boolChk1* und das Steuerelement *chk2* aktualisiert

Wenn Sie diese Technik verwenden, müssen Sie sicherstellen, dass die Objektvariable *objRibbon* ihren Inhalt nie verliert. Das kann aber der Fall sein, sobald es an beliebiger Stelle des VBA-Projekts zu einem unbehandelten Fehler kommt. Folglich müssen Sie den VBA-Code Ihrer Anwendung für einen reibungslosen Betrieb mit *objRibbon* komplett mit Fehlerbehandlungen ausstatten.

12.10 Menü- und Symbolleisten aus bestehenden Access 2003-Anwendungen

Viele Access-Entwickler sorgen sich um die Kompatibilität der Menü- und Symbolleisten von Anwendungen, die sie mit Access 2003 oder älter erstellt haben. Die Sorge ist natürlich berechtigt, denn immerhin hat Microsoft die Menüstruktur komplett umgekrempelt. Für die Verwendung älterer benutzerdefinierter Menüs wurde allerdings ein Hintertürchen offen gelassen: Diese werden standardmäßig einfach in einem eigenen Ribbon namens Add-Ins in einem *tab*-Element mit der Beschriftung *Benutzerdefinierte Symbolleisten* angezeigt (siehe Abbildung 12.38). Wenn Sie eine Menüleiste und eine oder mehrere Symbolleisten gleichzeitig anzeigen, erfolgt dies auch im Ribbon Add-Ins – mit dem feinen Unterschied, dass Sie die Position der Symbolleisten nicht mehr verändern können. Das Verschieben an den linken, rechten oder unteren Rand oder das Platzieren im freien Raum ist also nicht mehr möglich. Davon abgesehen übernimmt Access 2007 alle menürelevanten Einstellungen, die Sie im *Start*-Dialog (*Optionen* | *Start*) vornehmen können – zum Beispiel:

- ▶ *Datenbankfenster*: blendet bei Deaktivierung den Navigationsbereich aus
- ▶ *Statusleiste*: blendet bei Deaktivierung die Statusleiste aus
- ▶ *Unbeschränkte Menüs anzeigen*: blendet bei Deaktivierung alle Einträge im Office-Menü außer *Datenbank schließen* und *Access beenden* aus



Abbildung 12.38: Menü von Anwendungen, die mit Access 2003 und älter erstellt wurden, erhalten ein eigenes Ribbon

Es gibt aber auch die Möglichkeit, in Access 2007 eine Menüleiste im alten Stil anzuzeigen. Dazu sind folgende Voraussetzungen erforderlich:

- ▶ Es handelt sich um eine Datenbank im Format von Access 2000 bis Access 2003.
- ▶ Die Datenbank enthält eine benutzerdefinierte Menüleiste und diese ist in den Startoptionen von Access 2003 und älter als Menüleiste eingetragen.
- ▶ Sie öffnen die Datenbank in Access 2007, deaktivieren dort in den Access-Optionen die Eigenschaft *Aktuelle Datenbank\Navigation\Navigationsbereich anzeigen* und stellen im Bereich *Aktuelle Datenbank\Multifunktionsleisten- und Symbolleistenoptionen* die Eigenschaft *Menüleiste* auf den Namen Ihrer Menüleiste und die Option *Integrierte Symbolleisten zulassen* auf *Nein* ein.

Das Ergebnis sollte nun etwa wie in Abbildung 12.39 aussehen:

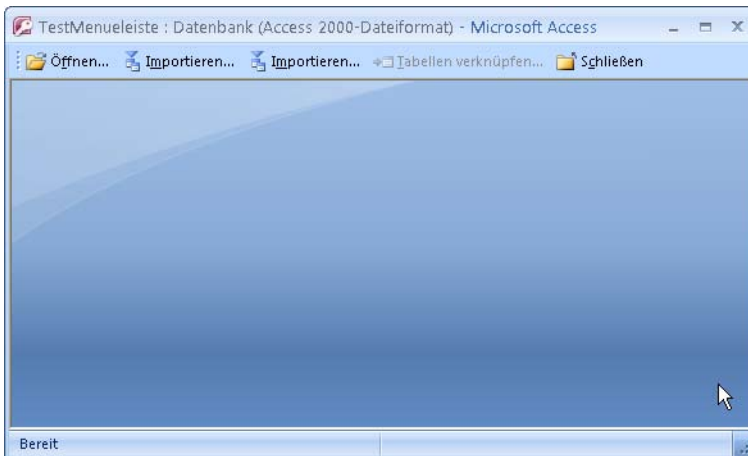


Abbildung 12.39: Eine .mdb-Datei in Access 2007 mit benutzerdefinierter Symbolleiste

Bedenken Sie, dass Sie mit einer solchen .mdb-Datei keine der vielen Neuerungen von Access 2007 nutzen können.

12.11 Übersicht über Ribbon-XML-Elemente und Attribute

Die vorhergehenden Abschnitte haben Ihnen einen Überblick über die Techniken zum Anpassen des Ribbons gegeben. Damit Sie beim Bauen benutzerdefinierter Tabs alle möglichen Elemente im Überblick haben, finden Sie nachfolgend vier Tabellen mit folgendem Inhalt:

- ▶ Tabelle 12.1: Auflistung aller Elemente mit Beschreibung und möglichen Kind-Elementen
- ▶ Tabelle 12.2: Auflistung aller Attribute mit Datentyp, Werten und Beschreibung
- ▶ Tabelle 12.3: Auflistung aller Ereignisseigenschaften und deren VBA-Syntax
- ▶ Tabelle 12.4: Zuordnung der Attribute und Ereignisseigenschaften zu den einzelnen Elementen

12.11.1 Auflistung der Ribbon-Elemente

Die folgende Tabelle enthält alle möglichen Elemente mit Beschreibung möglicher Kind-Elemente.

Element	Beschreibung	Mögliche Kind-Steuerelemente
box	Ein Rahmen, um Elemente zu gruppieren	button, toggleButton, checkBox, editBox, comboBox, dropDown, menu, gallery, box, splitButton, labelControl, buttonGroup, dynamicMenu, control
button	Schaltfläche	
buttonGroup	Kann eine Gruppe von Schaltflächen enthalten	button, toggleButton, menu, gallery, split-Button, dynamicMenu, control
checkBox	Kontrollkästchen	
comboBox	Kombinationsfeld, das beim Ändern den angezeigten Inhalt zurückgibt	item
command	Dient zum Festlegen benutzerdefinierter Funktionen für eingebaute Steuerelemente und aktiviert oder deaktiviert diese	
commands	Auflistung der command-Elemente.	command
contextualTabs	Fasst tabSets mit kontextsensitiven tabs zusammen	tabSet

Tabelle 12.1: Mögliche Elemente von Ribbon-XML-Dokumenten

Element	Beschreibung	Mögliche Kind-Steurelemente
control	Übergeordnetes Element für eingebaute Objekte, erlaubt etwa das Einbauen des Schriftfarbe-Steurelements in benutzerdefinierte tab-Elemente	
customUI	Root-Element eines Ribbon-XML-Dokuments	commands, ribbon
dialogBoxLauncher	Fügt einer Gruppe eine kleine Schaltfläche zum Öffnen eines Dialogs hinzu	button
documentControls	Elemente der Schnellzugriffsleiste (qat), die nur für die aktuelle Anwendung zur Verfügung stehen sollen	button, separator, control
dropDown	Kombinationsfeld, das beim Ändern die angezeigte ID und den Index liefert	item
dynamicMenu	Zur Laufzeit zu erzeugendes Menü	button, toggleButton, checkBox, menu, gallery, splitButton, menuSeparator, dynamicMenu, control
editBox	Textfeld	
gallery	Galerie - wie Kombinationsfeld, jedoch mehrspaltig und komfortabler	button, item
group	Gruppe innerhalb eines tab-Elements	button, toggleButton, checkBox, editBox, comboBox, dropDown, menu, gallery, box, labelControl, buttonGroup, separator, dialogBoxLauncher, control
item	Element eines combo-Box-, dropDown- oder gallery-Steurelements, wird beim dynamischen Zuweisen überschrieben	
labelControl	Steurelement mit einer Beschriftung	
menu	Menü-Steurelement	button, toggleButton, checkBox, menu, gallery, splitButton, menuSeparator, dynamicMenu, control
menuSeparator	Trennstrich zwischen Menüpunkten	

Tabelle 12.1: Mögliche Elemente von Ribbon-XML-Dokumenten (Fortsetzung)

Element	Beschreibung	Mögliche Kind-Steuererelemente
officeMenu	Office-Menü-Element, kann um benutzerdefinierte Steuererelemente ergänzt werden	button, toggleButton, checkBox, menu, gallery, splitButton, menuSeparator, dynamicMenu, control
qat	Schnellzugriffsleiste (Quick Access Toolbar), kann nur bei startFromScratch="true" angepasst werden	sharedControls, documentControls
ribbon	Enthält alle Ribbon-Bestandteile	officeMenu, qat, contextualTabs, tabs
separator	Trennstrich zwischen Steuererelementen	
sharedControls	Elemente, die für alle Office-Anwendungen verfügbar sein sollen	button, separator, control
splitButton	Kombination aus Schaltfläche und Menü wie etwa das Steuererelement zum Einstellen der Objektansicht	button, toggleButton
tab	Eine Registerseite des Ribbon	group
tabs	Enthält die Registerseiten des Ribbon	tab
tabSet	Fasst tab-Elemente zusammen, die in bestimmten Kontexten sichtbar gemacht werden sollen	tab
toggleButton	Umschaltfläche	

Tabelle 12.1: Mögliche Elemente von Ribbon-XML-Dokumenten (Fortsetzung)

12.11.2 Attribute der Ribbon-Elemente

Die folgende Tabelle enthält eine Auflistung aller Attribute der Ribbon-Elemente.

Eigenschaft	Datentyp	Werte	Beschreibung
boxStyle	String	horizontal vertical	Legt fest, wie die Elemente innerhalb eines box-Elements angeordnet werden
columns	Long		Anzahl der Spalten in einem gallery-Objekt
description	String		Beschreibungstext, der in den Einträgen eines menu-Elements angezeigt wird, wenn die Eigenschaft itemSize auf large eingestellt ist

Tabelle 12.2: Eigenschaften von Ribbon-Elementen

Eigenschaft	Datentyp	Werte	Beschreibung
boxStyle	String	horizontal vertical	Legt fest, wie die Elemente innerhalb eines box-Elements angeordnet werden
columns	Long		Anzahl der Spalten in einem gallery-Objekt
description	String		Beschreibungstext, der in den Einträgen eines menu-Elements angezeigt wird, wenn die Eigenschaft itemSize auf large eingestellt ist
enabled	Boolean	false true 0 1	Legt fest, ob ein Steuerelement aktiviert oder deaktiviert ist
id	String		Eindeutige ID eines benutzerdefinierten Steuerelements; nicht zu benutzen in Kombination mit idMso oder idQ
idMso	Long		Eindeutige ID eines eingebauten Steuerelements; nicht zu benutzen in Kombination mit idMso oder idQ
idQ	Long		Steuerelement-ID, enthält Namespace-Bezeichnung; nicht in Kombination mit id oder idMso
image	String		String zum Angeben einer Image-Datei
imageMso	String		Verweist auf ein eingebautes Symbol
insertAfterMso	String		Gibt an, hinter welches eingebaute Steuerelement mit idMso ein neues Element angelegt werden soll
insertAfterQ	String		Gibt an, hinter welches eingebaute Steuerelement mit idQ ein neues Element angelegt werden soll
insertBeforeMso	String		Gibt an, vor welches eingebaute Steuerelement mit idMso ein neues Element angelegt werden soll
insertBeforeQ	String		Gibt an, vor welches eingebaute Steuerelement mit idQ ein neues Element angelegt werden soll
itemHeight	Long		Höhe eines Elements des gallery-Steuerelements in Pixeln
itemSize	String	normal large	Größe eines Menü-Elements
itemWidth	Long		Breite eines Elements des gallery-Steuerelements in Pixeln
keytip	String		Gibt das Tastenkürzel an, dass beim Betätigen der Alt-Taste erscheint

Tabelle 12.2: Eigenschaften von Ribbon-Elementen (Fortsetzung)

Eigenschaft	Datentyp	Werte	Beschreibung
label	String		Bezeichnungsfeld eines Steuerelements
maxLength	Long		Maximale Länge einer einzugebenden Zeichenkette
rows	Long		Anzahl der Zeilen in einem gallery-Steuerelement
screenTip	String		Überschrift des beim Überfahren des Elements angezeigten Textes
showImage	String	false true 0 1	Soll ein Symbol angezeigt werden?
showItemImage	String	false true 0 1	Soll ein Symbol für ein Listenelement des comboBox-, dropDown- oder gallery-Steuerelements angezeigt werden?
showItemLabel	String	false true 0 1	Soll ein Text für ein Listenelement des comboBox-, dropDown- oder gallery-Steuerelements angezeigt werden?
showLabel	String	false true 0 1	Soll die Bezeichnung eines Steuerelements angezeigt werden?
size	String	normal large	Legt die Größe für die Anzeige eines Steuerelements fest.
sizeString	String		Das Steuerelement wird so breit gestaltet, dass der für diese Eigenschaft angegebene Text komplett angezeigt wird.
startFromScratch	String	false true 0 1	Legt fest, ob eingebaute Ribbons und einige Einträge des Office-Menüs ausgeblendet werden.
supertip	String		Unter dem screenTip beim Überfahren eines Elements angezeigter Text
tag	String		?
title	String		Fügt einem Trennstrich eines menu-Elements eine Beschriftung hinzu.
visible	String	false true 0 1	Gibt an, ob ein Element sichtbar ist.

Tabelle 12.2: Eigenschaften von Ribbon-Elementen (Fortsetzung)

12.11.3 Ereignisseigenschaften der Ribbon-Elemente

Die folgende Tabelle enthält alle Ereignisseigenschaften der Ribbon-Elemente und die VBA-Syntax der passenden Callback-Funktionen.

Eigenschaft	Steuerelement	VBA-Syntax
getDescription	(several controls)	Sub GetDescription(control As IRibbonControl, ByRef description)
getEnabled	(several controls)	Sub GetEnabled(control As IRibbonControl, ByRef enabled)
getImage	(several controls)	Sub GetImage(control As IRibbonControl, ByRef image)
getImageMso	(several controls)	Sub GetImageMso(control As IRibbonControl, ByRef imageMso)
getItemCount	comboBox	Sub GetItemCount(control As IRibbonControl, ByRef count)
	dropDown	Sub GetItemCount(control As IRibbonControl, ByRef count)
	gallery	Sub GetItemCount(control As IRibbonControl, ByRef count)
getItemHeight	gallery	Sub getItemHeight(control As IRibbonControl, ByRef height)
getItemID	comboBox	Sub GetItemID(control As IRibbonControl, index As Integer, ByRef id)
	dropDown	Sub GetItemID(control As IRibbonControl, index As Integer, ByRef id)
	gallery	Sub GetItemID(control As IRibbonControl, index As Integer, ByRef id)
getItemImage	comboBox	Sub GetItemImage(control As IRibbonControl, index As Integer, ByRef image)
	dropDown	Sub GetItemImage(control As IRibbonControl, index As Integer, ByRef image)
	gallery	Sub GetItemImage(control As IRibbonControl, index As Integer, ByRef image)
getItemLabel	comboBox	Sub GetItemLabel(control As IRibbonControl, index As Integer, ByRef label)
	dropDown	Sub GetItemLabel(control As IRibbonControl, index As Integer, ByRef label)

Tabelle 12.3: Methoden von Steuerelementen und ihre Syntax

Eigenschaft	Steuerelement	VBA-Syntax
	gallery	Sub GetItemLabel(control As IRibbonControl, index As Integer, ByRef label)
getItemScreenTip	comboBox	Sub GetItemScreenTip(control As IRibbonControl, index As Integer, ByRef screentip)
	dropDown	Sub GetItemScreenTip(control As IRibbonControl, index As Integer, ByRef screentip)
	gallery	Sub GetItemScreenTip(control As IRibbonControl, index as Integer, ByRef screen)
getItemSuperTip	comboBox	Sub GetItemSuperTip (control As IRibbonControl, index As Integer, ByRef supertip)
	dropDown	Sub GetItemSuperTip (control As IRibbonControl, index As Integer, ByRef superTip)
	gallery	Sub GetItemSuperTip (control As IRibbonControl, index as Integer, ByRef screen)
getItemWidth	gallery	Sub getItemWidth(control As IRibbonControl, ByRef width)
getKeytip	(several controls)	Sub GetKeytip (control As IRibbonControl, ByRef label)
getLabel	(several controls)	Sub GetLabel(control As IRibbonControl, ByRef label)
getPressed	checkBox	Sub GetPressed(control As IRibbonControl, ByRef return)
	toggleButton	Sub GetPressed(control As IRibbonControl, ByRef return)
getScreentip	(several controls)	Sub GetScreentip(control As IRibbonControl, ByRef screentip)
getSelectedItemID	dropDown	Sub GetSelectedItemID(control As IRibbonControl, ByRef index)
	gallery	Sub GetSelectedItemID(control As IRibbonControl, ByRef index)
getSelectedItemIndex	dropDown	Sub GetSelectedItemIndex(control As IRibbonControl, ByRef index)
	gallery	Sub GetSelectedItemIndex(control As IRibbonControl, ByRef index)

Tabelle 12.3: Methoden von Steuerelementen und ihre Syntax (Fortsetzung)

Eigenschaft	Steuerelement	VBA-Syntax
getShowImage	button	Sub GetShowImage (control As IRibbonControl, ByRef showImage)
getShowLabel	button	Sub GetShowLabel (control As IRibbonControl, ByRef showLabel)
getSize	(several controls)	Sub GetSize(control As IRibbonControl, ByRef size)
getSupertip	(several controls)	Sub GetSupertip(control As IRibbonControl, ByRef screentip)
getText	comboBox	Sub GetText(control As IRibbonControl, ByRef text)
	editBox	Sub GetText(control As IRibbonControl, ByRef text)
getTitle	menuSeparator	Sub GetTitle (control As IRibbonControl, ByRef title)
getVisible	(several controls)	Sub GetVisible(control As IRibbonControl, ByRef visible)
loadImage	customUI	Sub LoadImage(imageId As string, ByRef image)
onAction	button	Sub OnAction(control As IRibbonControl)
	checkBox	Sub OnAction(control As IRibbonControl, pressed As Boolean)
	dropDown	Sub OnAction(control As IRibbonControl, selectedId As String, selectedIndex As Integer)
	gallery	Sub OnAction(control As IRibbonControl, selectedId As String, selectedIndex As Integer)
	toggleButton	Sub OnAction(control As IRibbonControl, pressed As Boolean)
onChange	comboBox	Sub OnChange(control As IRibbonControl, text As String)
	editBox	Sub OnChange(control As IRibbonControl, text As String)
onLoad	customUI	Sub OnLoad(ribbon As IRibbonUI)

Tabelle 12.3: Methoden von Steuerelementen und ihre Syntax (Fortsetzung)

12.11.4 Steuerelemente und ihre Eigenschaften

Die folgende Tabelle zeigt, welche Steuerelemente aus Tabelle 12.1 mit welchen Eigenschaften aus Tabelle 12.2 und Ereigniseigenschaften aus Tabelle 12.3 ausgestattet sind.

Eigenschaften/ Steuerelemente	box	button	buttonGroup	checkBox	comboBox	command	customUI	dropDown	dynamicMenu	editBox	gallery	item	labelControl	menu	menuSeparator	ribbon	separator	splitButton	tab	tabSet	textBox	toggleButton	
boxStyle	•																					•	
columns											•												
description		•		•					•		•		•	•									•
enabled		•		•	•	•		•	•	•	•		•	•				•					•
getContent									•														
getDescription		•		•					•		•			•									•
getEnabled		•		•	•	•		•	•	•	•		•	•				•					•
getImage		•			•			•	•	•	•			•									•
getImageMso		•			•			•	•	•	•			•									•
getItemCount					•			•			•												
getItemHeight											•												
getItemID					•			•			•												
getItemImage					•			•			•												
getItemLabel					•			•			•												
getItemScreentip					•			•			•												
getItemSupertip					•			•			•												
getItemWidth											•												
getKeytip		•		•	•			•	•	•	•			•				•	•				•
getLabel		•		•	•			•	•	•	•		•	•					•				•
getPressed				•																			•
getScreentip		•		•	•			•	•	•	•		•	•									•
getSelectedItemID								•			•												
getSelectedItem- Index								•			•												
getShowImage		•			•			•	•	•	•			•									•
getShowItemImage					•			•			•												
getShowLabel		•		•	•			•	•	•	•		•	•				•					•
getSize		•												•				•					•
getSupertip		•		•	•			•	•	•	•		•	•				•					•
getText					•			•		•	•												
getTitle															•								
getVisible		•	•	•	•			•	•	•			•	•			•	•	•	•	•	•	•

Tabelle 12.4: Eigenschaften von Steuerelementen

Eigenschaften/ Steuerelemente	box	button	buttonGroup	checkBox	comboBox	command	customUI	dropDown	dynamicMenu	editBox	gallery	item	labelControl	menu	menuSeparator	ribbon	separator	splitButton	tab	tabSet	textBox	toggleButton
id		•	•	•	•			•	•	•	•	•	•	•	•		•	•	•		•	•
idMso		•		•	•	•		•	•	•	•		•	•				•	•	•	•	•
idQ		•	•	•	•			•	•	•	•		•	•	•		•	•	•		•	•
image		•			•			•	•	•	•	•		•								•
imageMso		•			•			•	•	•	•	•		•								•
insertAfterMso		•	•	•	•			•	•	•	•		•	•	•		•	•	•		•	•
insertAfterQ		•	•	•	•			•	•	•	•		•	•	•		•	•	•		•	•
insertBeforeMso		•	•	•	•			•	•	•	•		•	•	•		•	•	•		•	•
insertBeforeQ		•	•	•	•			•	•	•	•		•	•	•		•	•	•		•	•
ItemHeight											•											
itemSize													•									
ItemWidth											•											
keytip		•		•	•			•	•	•			•					•	•			•
label		•		•	•			•	•	•	•	•	•	•				•	•			•
loadImage							•															
maxLength					•			•		•	•											
onAction		•		•		•		•			•											•
onChange					•					•												
onLoad							•															
rows											•											
screenTip		•		•	•			•	•	•	•	•	•	•								•
showImage		•			•			•	•					•								•
showItemImage					•			•		•												
showItemLabel					•			•		•												
showLabel				•	•			•	•	•			•	•				•				•
size		•																	•			•
sizeString					•			•		•	•											
startFromScratch																•						
supertip		•		•	•			•	•	•	•	•	•	•				•				•
tag		•		•	•			•	•	•	•		•	•				•	•			•
visible		•	•	•	•			•	•	•	•		•	•			•	•	•	•	•	•

Tabelle 12.4: Eigenschaften von Steuerelementen (Fortsetzung)

Quellen zu diesem Kapitel

- [1] XML-Notepad Download: <http://www.microsoft.com/downloads/details.aspx?familyid=72d6aa49-787d-4118-ba5f-4f30fe913628&displaylang=en>
- [2] Schemadefinitionsdatei für Ribbons (*customui.xsd*): <http://officeblogs.net/UI/customUI.zip>
- [3] Tabelle mit Informationen zu allen Ribbon-Elementen von Access: Buch-CD \Kap_12\AccessIDMSO.accdb oder <http://www.access-entwicklerbuch.de/2007/grundlagen/AccessIDMSO.accdb>