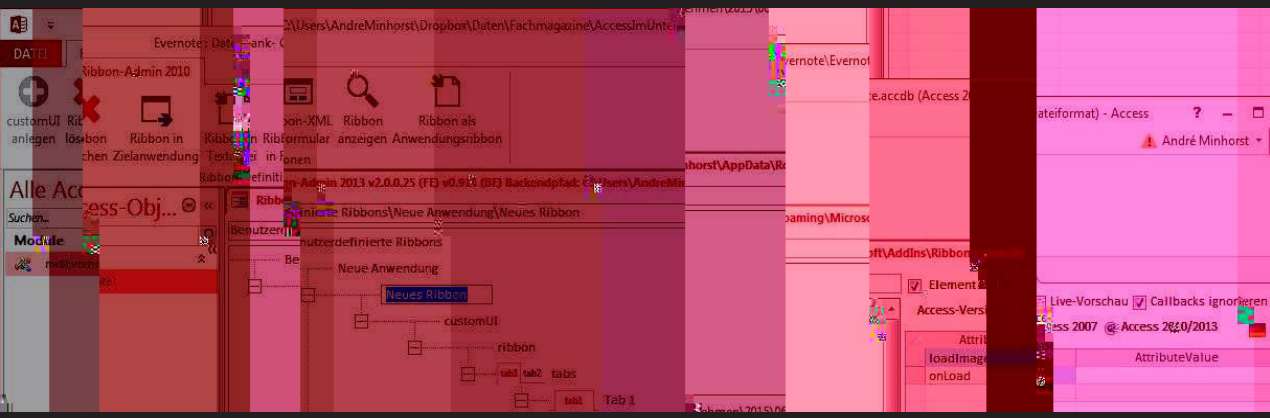
RIBBON-ADMIN



RIBBONS ENTWICKELN MIT ACCESS 2007-2016

André Minhorst Verlag

André Minhorst

Ribbon-Admin

Ribbons entwickeln
mit Access 2007 bis 2016

André Minhorst – Ribbon-Admin

ISBN 978-3-944216-05-8

Borkhofer Straße 17, 47137 Duisburg/Deutschland

1. Auflage 2015

Lektorat André Minhorst
Korrektur Rita Klingenstein
Cover/Titelbild André Minhorst
Typographie, Layout und Satz André Minhorst
Herstellung André Minhorst

Bibliografische Information der Deutschen Nationalbibliothek

Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliographie. Detaillierte bibliografische Daten finden Sie im Internet unter http://dnb.d-nb.de.

Inhalt

1 Ribbon-Admin installieren

Dieses Kapitel beschreibt die Voraussetzungen, die Installation und die Inbetriebnahme des Ribbon-Admin in den Versionen 2010, 2013 und 2016.

1.1 Voraussetzungen

Voraussetzung für die Verwendung des Ribbon-Admin ist die jeweilige Access-Version in der 32bit-Version (also etwa Access 2016 für den Ribbon-Admin 2016).

1.2 Installation

Der Ribbon-Admin kommt als einzelne Datei etwa namens *RibbonAdmin2010.accdb*, *RibbonAdmin2013.accda* oder *RibbonAdmin2016.accda*. Diese platzieren Sie in einem beliebigen Verzeichnis – merken Sie sich nur, um welches Verzeichnis es sich dabei handelt.

Starten Sie nun Access und öffnen Sie eine beliebige Datenbankdatei. Natürlich können Sie zu diesem Zweck auch eine neue Datenbankdatei anlegen – umso besser, wenn Sie gleich danach ein wenig mit dem Ribbon-Admin experimentieren möchten.

Hier finden Sie nun im Ribbon den Eintrag *Datenbanktools|Add-Ins|Add-Ins|Add-In-Manager* vor (siehe Abbildung 1.1).

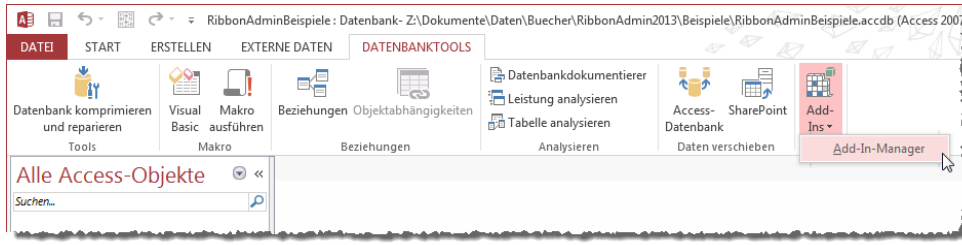


Abbildung 1.1: Starten des Add-In-Managers

Betätigen Sie diesen Befehl, erscheint der Add-In-Manager von Access (siehe Abbildung 1.2). Mit diesem fügen Sie Add-Ins hinzu oder verwalten diese.

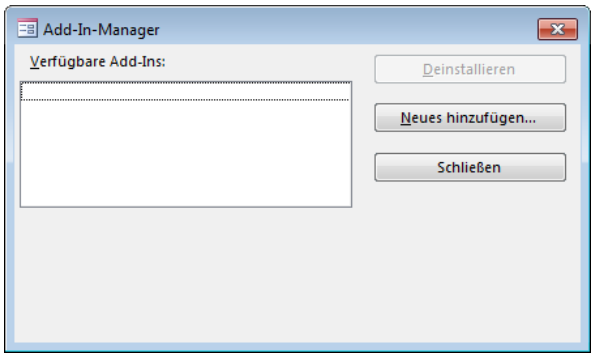


Abbildung 1.2: Der Add-In-Manager

Klicken Sie hier auf die Schaltfläche *Neues hinzufügen...*, um den *Öffnen*-Dialog aus Abbildung 1.3 anzuzeigen. Hiermit navigieren Sie nun zu dem Verzeichnis, in dem Sie den Ribbon-Admin gespeichert haben – in diesem Fall etwa auf dem Desktop.

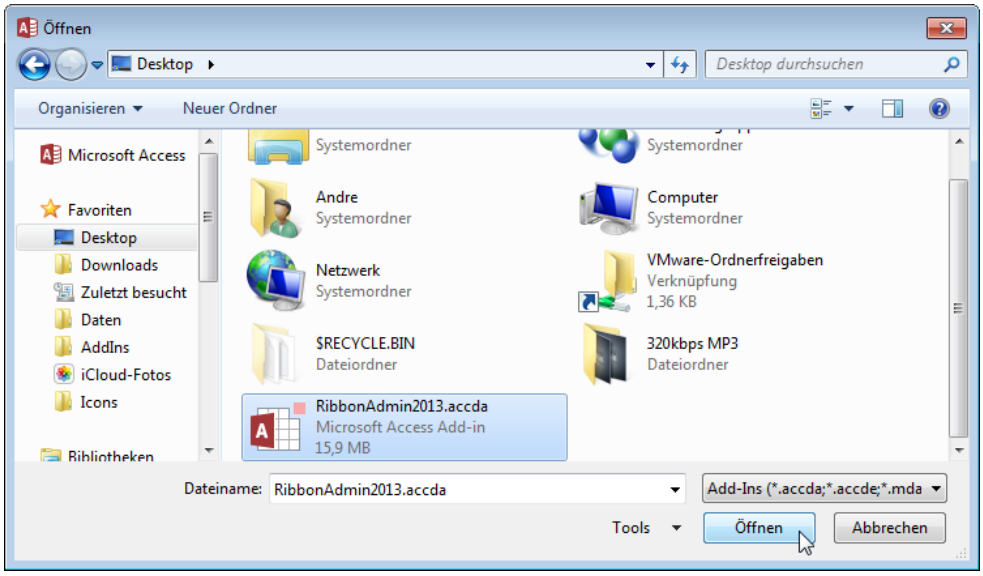


Abbildung 1.3: Auswahl des Ribbon-Admin, hier in der Version für Access 2013

Gleich im Anschluss kehren Sie zum Add-In-Manager zurück, der nun den Ribbon-Admin in der Liste der verfügbaren Add-Ins anzeigt. Damit sind Sie an dieser Stelle fertig und können den Dialog mit einem Klick auf Schließen beenden (siehe Abbildung 1.4).

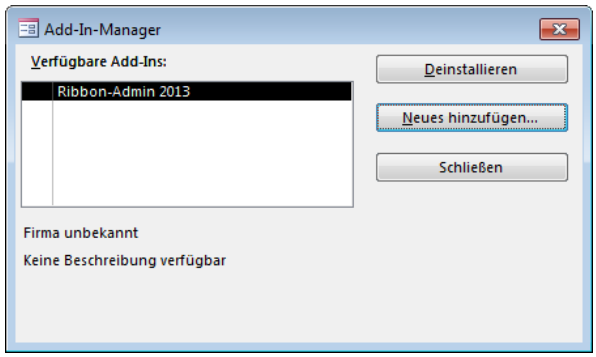


Abbildung 1.4: Add-In-Manager mit dem frisch hinzugefügten Add-In

Öffnen Sie nun erneut das Add-Ins-Menü im Ribbon, erscheint auch dort das neu installierte Add-In (siehe Abbildung 1.5).

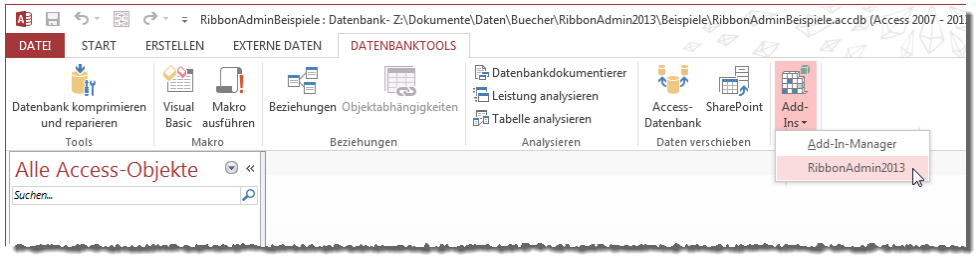


Abbildung 1.5: Der Ribbon-Admin im Add-Ins-Menü

1.2.1 Auswirkungen der Installation

Was ist in der Zwischenzeit geschehen? Erstens hat der Add-In-Manager die Datei *RibbonAdminXXXX.accda* (XXXX für die jeweilige Version) in das Add-In-Verzeichnis kopiert, das etwa unter Windows 7 typischerweise in folgendem Ordner zu finden ist:

```
C:\Users\<Benutzername>\AppData\Roaming\Microsoft\AddIns
```

Zweitens hat der Add-In-Manager einige Einträge zur Registry hinzugefügt. Diese werden beim Start von Access abgefragt und sorgen dafür, dass die dort vermerkten Add-Ins bei der Verwendung von Access an entsprechender Stelle auftauchen – beispielsweise im Add-In-Menü. Die Registry-Einträge können Sie einsehen, wenn Sie zunächst den Registry-Editor öffnen.

Dazu geben Sie im Suchen-Fenster von Windows den Ausdruck *RegEdit* ein und klicken Sie doppelt den nun erscheinenden Eintrag. Navigieren Sie hier zum folgenden Zweig (hier für die Version 2013):

```
HKEY_CURRENT_USER\Software\Microsoft\Office\15.0\Access\Menu Add-Ins\RibbonAdmin2013
```

Dort finden Sie einige Einträge etwa wie in Abbildung 1.6 abgebildet vor.

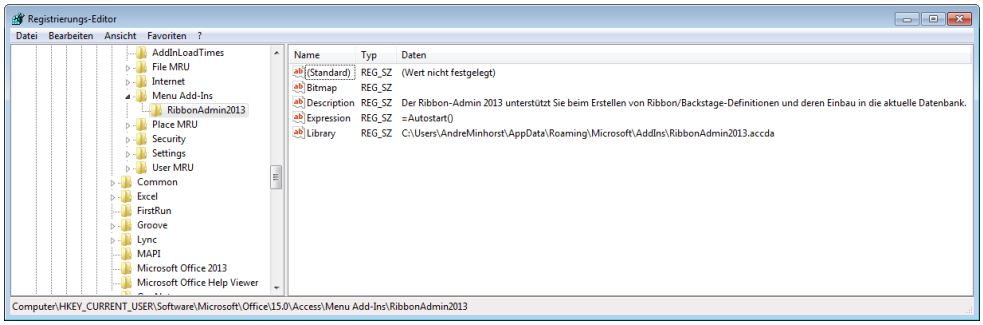


Abbildung 1.6: Der Ribbon-Admin in der Registry

1.2.2 Backend-Datenbank anlegen

Keine Sorge, es sind keine weiteren manuellen Schritte erforderlich, denn das Folgende erledigt sich beim ersten Aufruf des Ribbon-Admin: Dieser legt nämlich im Add-In-Verzeichnis eine weitere Datei an, die alle von Ihnen definierten Ribbon-Einstellungen erfasst.

Dazu rufen Sie nun den Ribbon-Admin erstmalig über das Add-In-Menü von Access auf. Dies sieht im Access-Fenster nun so wie in Abbildung 1.7 aus.

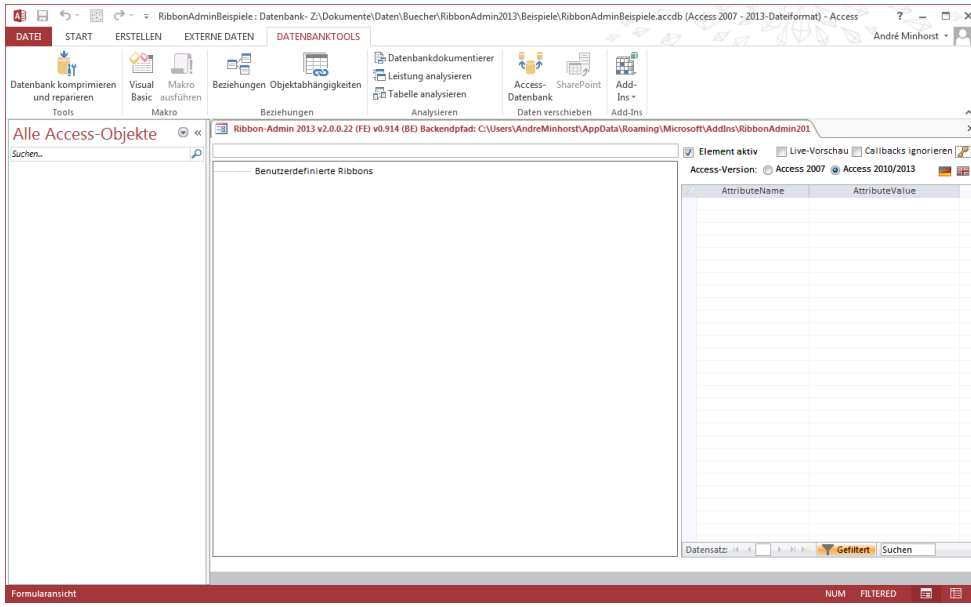


Abbildung 1.7: Erster Start des Ribbon-Admin

Wenn Sie zuvor keine andere Version des Ribbon-Admin verwendet haben, als Sie jetzt nutzen, legt der Ribbon-Admin nun die Datei *RibbonAdmin2010_Data.accdb* im Add-In-Verzeichnis an. Warum taucht hier die Versionsnummer 2010 auf? Weil der Ribbon-Admin 2013 und der Ribbon-Admin 2016 auf eventuell bereits mit der Vorgängerversion erstellten Ribbon-Definitionen zugreifen können soll, die Vorgängerversion aber gegebenenfalls auch noch verwendet wird.

Sollten Sie zuvor bereits *Ribbon-Admin 2010* verwendet haben, binden Ribbon-Admin 2013 und Ribbon-Admin 2016 eine vorhandene Datei *RibbonAdmin2010_Data.accdb* automatisch ein und zeigt die damit erzeugten Ribbon-Definitionen an.

1.3 Registrierung

Direkt nach der Installation läuft der Ribbon-Admin im Demo-Modus. Das heißt, dass Sie alles damit erledigen können, mit einer Einschränkung: Beim Exportieren der Ribbon-Definitionen wird von jedem Element nur ein einziges ausgeliefert beziehungsweise angezeigt. Sie können damit zwar schon umfangreiche Ribbon-Definitionen anlegen, aber diese noch nicht zu einer Access-Anwendung hinzufügen.

Nach der Bestellung erhalten Sie allerdings direkt per E-Mail einen Registrierungsschlüssel. Um diesen einzugeben, klicken Sie rechts oben im Ribbon-Admin auf das Schlüssel-Symbol (siehe Abbildung 1.8).

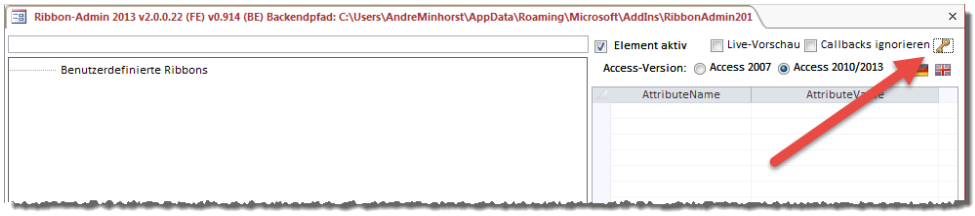


Abbildung 1.8: Anzeige des Dialogs für die Registrierung

Im nun erscheinenden Dialog tragen Sie die per E-Mail erhaltenen Informationen ein (siehe Abbildung 1.9).

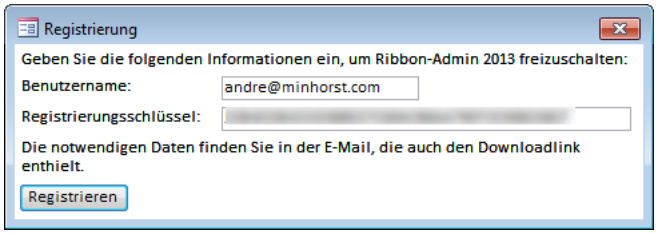


Abbildung 1.9: Eingabe der Registrierungsdaten

Fertig – damit ist der Ribbon-Admin einsatzbereit. Wie der folgende Dialog aus Abbildung 1.10 zeigt, haben wir den Nutzungszeitraum sehr großzügig ausgelegt.

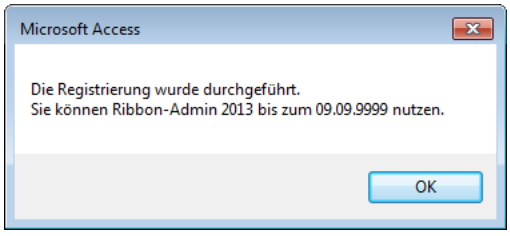


Abbildung 1.10: Bestätigung der Registrierung

2 Ribbon-Admin verwenden

Dieses Kapitel erläutert die einzelnen Elemente der Benutzeroberfläche und wie Sie diese einsetzen.

2.1 Systemvoraussetzungen

Der Ribbon-Admin mit der jeweiligen Access-Version. Der *Ribbon-Admin 2016* ist also für Access 2016 gedacht, der *Ribbon-Admin 2013* für Access 2013 und so weiter. Die verschiedenen Versionen können Sie unter *www.amvshop.de* erwerben.

2.2 Den Ribbon-Admin starten

Den Ribbon-Admin verwenden Sie in Zusammenhang mit der Datenbank, für die Sie Ribbons erstellen möchten. Starten Sie Access 2013, öffnen Sie die Datenbank und starten Sie dann den Ribbon-Admin über den Ribbon-Eintrag **Datenbanktools|Add-Ins|Ribbon-Admin**.

Der Ribbon-Admin empfängt Sie mit zwei Bereichen: Links ist der Ribbon-Baum mit der Struktur des Ribbons, rechts die Liste der Attribute des jeweiligen Ribbons.

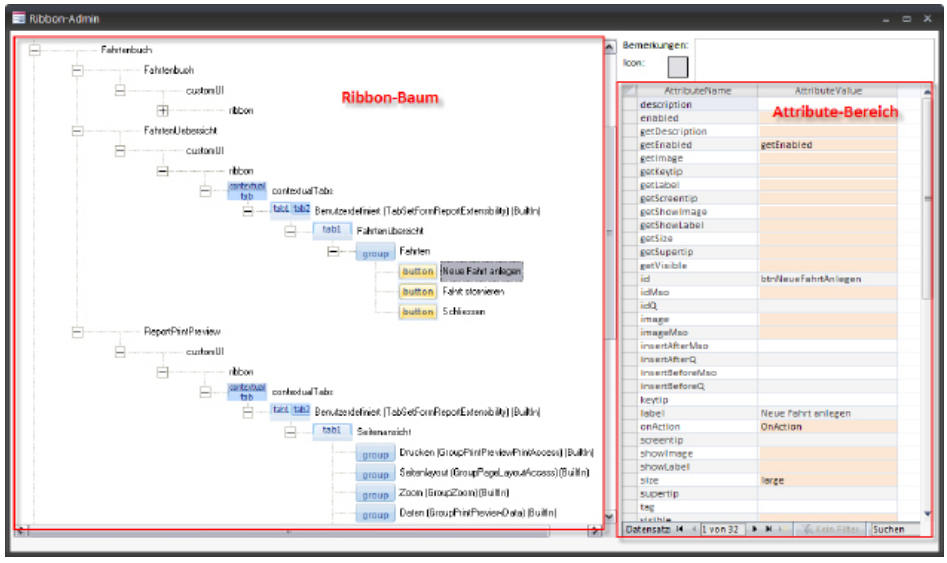


Abbildung 1.1: Elemente des Ribbon-Admins

2.3 Ein einfaches Ribbon anlegen

Der Ribbon-Admin bietet die Möglichkeit, Ribbons für mehrere Anwendungen anzulegen. Das oberste Element eines Ribbons ist daher immer die Anwendung. Diese legen Sie über den Kontextmenüeintrag **Neue Anwendung** des Hauptelements **Benutzerdefinierte Ribbons** an. Tragen Sie direkt einen Namen für die Anwendung ein.

Diesem Element fügen Sie nun mit dem Eintrag **Ribbon hinzufügen** des Kontextmenüs eine neue Ribbon-Definition hinzu und stellen auch dessen Beschriftung ein, beispielsweise auf **Fahrtenbuch**.

Nun folgenden die Elemente des Ribbons. Das Hauptelement heißt **customUI**. Legen Sie dieses über den entsprechenden Kontextmenüeintrag des neuen Elements hinzu. Auf die gleiche Weise legen Sie unterhalb des **customUI**-Elements ein **ribbon**-Element und darunter ein **tabs**-Element an. Danach folgt das erste Element, für das Sie eine Eigenschaft im Attribute-Bereich einstellen: Das Attribut **label** enthält die Beschriftung des **tab**-Elements. Tragen Sie hier die gewünschte Bezeichnung ein. Danach folgt die Gruppe, und zwar in Form eines wiederum untergeordneten **group**-Elements – auch hier ist wieder eine sinnvolle Beschriftung gefragt, die Sie für das Attribut **label** eintragen.

Das Kontextmenü des **group**-Elements offenbart im Anschluss die ganze Pracht der Ribbon-Steuerelemente: Hier können Sie auf alle möglichen Elemente des Ribbons zugreifen. Für den Anfang legen Sie einfach ein **button**-Element an. Stellen Sie dafür folgende Attribute ein:

» **label**: Beschriftung des Steuerelements

» **size**: Größe des Steuerelements. Mit dem Kontextmenü des Textfelds für die Eingabe des Attributwertes können Sie zwischen den beiden verfügbaren Werten **normal** und **large** wählen.

Damit haben Sie das Grundgerüst erstellt. Nun soll Access das Ribbon anzeigen. Dazu wählen Sie aus dem Kontextmenü des oben angelegten Elements mit der Bezeichnung Fahrtenbuch den Eintrag **Ribbon anzeigen** aus. Im Ribbon erscheint ganz rechts eine neue Registerlasche mit dem für das **tab**-Element angegebenen Namen. Wenn Sie dieses anklicken, zeigt Access das neue Ribbon an.

2.4 Ribbon direkt sichtbar machen

Wenn Sie möchten, dass Access nur die von Ihnen festgelegten Elemente des Ribbons anzeigt, klicken Sie das Element **ribbon** an und stellen Sie das Attribut **startFromScratch** auf **true**. Beim erneuten Betätigen des **Ribbon anzeigen**-Befehls erscheint das neu hinzugefügte Tab allein im Ribbon.

2.5 Ribbon in die Anwendung integrieren

Auch wenn Sie wie oben beschrieben schon einen Blick auf den aktuellen Stand des Ribbons werfen konnten, wird dieses wieder verschwunden sein, wenn Sie den Ribbon-Admin beenden. Damit das Ribbon dauerhaft in Ihrer Anwendung enthalten ist, müssen Sie es hinzufügen, und zwar über den Eintrag **Ribbon in Zielanwendung schreiben** des Kontextmenüs des Ribbon-Definitions-Elements. Der Ribbon-Admin legt daraufhin eine spezielle Tabelle namens **USysRibbons** in der Zieldatenbank an. Damit dies funktioniert, muss der aktuelle Benutzer entsprechende Berechtigungen für die Datenbank besitzen. Danach müssen Sie nur noch festlegen, in welchem Zusammenhang die Anwendung das Ribbon anzeigen soll. Dies kann direkt beim Start erfolgen oder auch in Zusammenhang mit dem Öffnen eines Formulars oder Berichts. Um diese Einstellung wie nachfolgend beschrieben vorzunehmen, müssen Sie die Anwendung einmal schließen und erneut öffnen.

Wenn das Ribbon direkt beim Start angezeigt werden soll, verwenden Sie den Eintrag **Ribbon als Anwendungsribbon festlegen** des Kontextmenüs des Ribbon-Definitions-Elements. Soll dies in Zusammenhang mit einem Formular oder Bericht geschehen, ist der Eintrag **Ribbon zu Formularen und Berichten zuweisen** die richtige Wahl. Dies öffnet einen Dialog, mit dem Sie zunächst die Formulare und Berichte einlesen und diese den entsprechenden Formularen und Berichten zuweisen.

2.6 Bilder, Callbackfunktionen und mehr

Mit dem Ribbon-Admin können Sie auch benutzerdefinierte Bilder für die Anzeige in den Ribbon-Steuerelementen verwenden oder Callbackfunktionen anlegen. All dies würde den Rahmen einer kurzen Einführung sprengen – daher studieren Sie einfach den Rest dieser Dokumentation. Wenn Sie bereits selbst Ribbons definiert haben, können Sie an vielen Stellen springen, und wenn nicht, lernen Sie gleichzeitig eine Menge über den Aufbau des Ribbons und seiner Elemente.

3 Bedienung des Ribbon-Admin im Überblick

Die Bedienung des Ribbon-Admin erfordert grundlegende Kenntnisse der Struktur des Ribbons. Sind diese vorhanden, ist der Rest allerdings ganz leicht: Der Ribbon-Admin wird Ihnen dann die beste Unterstützung beim Anlegen und Verwalten Ihrer Ribbons für verschiedene Anwendungen liefern.

3.1 Aufbau des Ribbon-Admins

Der Ribbon-Admin besteht nur aus zwei Elementen: Dem Ribbon-Baum und der Attributliste. Der Ribbon-Baum zeigt den Aufbau der gespeicherten Ribbons an, während die Attributliste die Attribute zum jeweils im Ribbon-Baum aktivierten Element liefert.

Alle Funktionen des Ribbon-Admins sind über Kontextmenüs verfügbar – sowohl im Ribbon-Baum als auch in der Attributliste.

Im Ribbon-Baum liefern die Kontextmenüs im Wesentlichen die Funktionen zum Hinzufügen und zum Entfernen von Elementen. Es gibt noch einige weitere, die den Import- und Export von Ribbons von und aus Anwendungen und ähnliche Aufgaben erledigen.

3.2 Elemente anlegen

Zum Anlegen von Elementen verwenden Sie den Kontextmenüeintrag **<Elementname> anlegen** des jeweils übergeordneten Elements. Das Element wird dann, wenn dies erforderlich ist, mit einer oder mehreren Standardwerten angelegt, damit es zu Testzwecken schnell angezeigt werden kann. Die Kontextmenüs zeigen jeweils nur die möglichen Unterelemente an.

Mit weiteren Kontextmenüeinträgen können Sie eingebaute Elemente anlegen, also beispielsweise ein **group**-Element, dass alle Steuerelemente einer eingebauten Gruppe des Ribbons mitbringt.

3.3 Sortieren, Verschieben und Kopieren

Wer Ribbons für verschiedene Anwendungen entwickelt, kann unter Umständen bereits vorhandene Ribbons oder zumindest deren Elemente wiederverwenden. Dazu können Sie diese Elemente per Drag and Drop von einem zum anderen Zweig kopieren oder verschieben. Auf die

gleiche Weise ändern Sie die Reihenfolge von Elementen unterhalb des gleichen übergeordneten Elements.

Die folgenden Abschnitte erläutern die Drag-and-Drop-Möglichkeiten des Ribbon-Admin.

3.3.1 Elemente verschieben

Das Verschieben eines Elements erfolgt am einfachsten per Drag and Drop im Ribbon-Baum, wobei nur solche Zielelemente beim Daraufziehen eines anderen Elements markiert werden, die sich auch als Ziel für ein Element eignen. Sie können also beispielsweise kein **button**-Element auf ein **tab**-Element ziehen, sondern nur auf ein **group**-Element und auf einige andere, speziell dafür geeignete Steuerelemente wie das **buttonGroup**-Element.

Beachten Sie, dass beim Verschieben eines Steuerelements alle untergeordneten Steuerelemente mit verschoben werden.

Die zweite Möglichkeit zum Verschieben von Steuerelementen finden Sie im Kontextmenü. Wählen Sie aus dem Kontextmenü des zu verschiebenden Elements den Eintrag **Ausschneiden** aus und für das neue übergeordnete Steuerelement den Eintrag **Einfügen**. Diese Methode bietet sich an, wenn das Ziel nicht zusammen mit dem zu verschiebenden Element angezeigt werden kann.

3.3.2 Elemente kopieren

Sie haben zwei Möglichkeiten zum Kopieren von Elementen:

» Drag and Drop mit der Maus

» Verwenden der Kontextmenüeinträge Kopieren und Einfügen

Beim Kopieren eines Elements per Drag and Drop müssen Sie die **Strg**-Taste gedrückt halten – also genau wie Sie es vom Windows Explorer gewohnt sind. Ansonsten gelten die gleichen Regeln wir für das Verschieben von Elementen.

Beim Kopieren und Einfügen über die entsprechenden Einträge des Kontextmenüs ist zu beachten, dass der Ribbon-Admin den Einfügen-Eintrag nach dem Kopieren eines Elements nur für solche Elemente anzeigt, die das zu kopierende Element auch aufnehmen können.

3.3.3 Reihenfolge von Elementen ändern

Die Reihenfolge für einige Elemente ist vorgegeben:

» Die Anwendungs- und Ribbon-Definitions-Elemente werden nach dem Alphabet sortiert.

» Einige Elemente werden entsprechend der Vorgabe der Schemadatei für Ribbons sortiert. So müssen zum Beispiel das **commands**- und das **ribbon**-Element des **customUI**-Elements immer in dieser Reihenfolge vorliegen, da sonst ein Fehler ausgelöst wird.

Bei den übrigen Elementen bleibt Ihnen die Reihenfolge überlassen. Sie können diese einstellen, indem Sie das Element, dessen Reihenfolge Sie ändern möchten, markieren und dann mit den Tastenkombinationen **Strg + Nach oben** sowie **Strg + Nach unten** nach oben oder unten verschieben.

3.4 Anwendungen anlegen

Der Ribbon-Admin erlaubt die Verwaltung von Ribbon-Definition von mehr als einem Ribbon gleichzeitig. Da Access-Entwickler gelegentlich mehr als eine Anwendung entwickeln, legt man im Ribbon-Admin zunächst den Namen der Anwendung fest, in der das Ribbon platziert werden soll (dies ist nur eine Einteilung, damit Sie später sehen, welches Ribbon Sie für welche Anwendung erstellt haben – natürlich können Sie jedes Ribbon zu beliebigen Anwendungen hinzufügen).

Eine Anwendung fügen Sie über den Kontextmenüeintrag **Neue Anwendung** des Hauptelements im Ribbon-Baum hinzu.

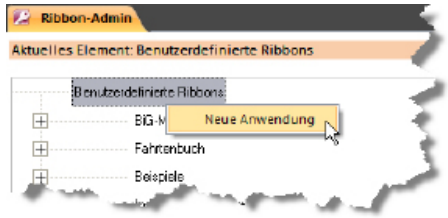


Abbildung 3.1: Anlegen einer neuen Anwendung im Ribbon-Baum

3.5 Ribbon-Definitionen anlegen

Jede Anwendung kann beliebig viele Ribbon-Definitionen enthalten. Eine solche Definition legen Sie mit einem der drei Kontextmenüeinträge des Anwendungs-Elements im Ribbon-Baum an:

» **Ribbon neu anlegen**: Legt eine leere Definition an, die Sie anschließend mit weiteren Elementen füllen können.

» **Ribbon aus Datei laden**: Lädt eine XML-Definition aus einer XML-Datei. Die Elemente dieser XML-Definition werden automatisch im Ribbon-Admin gespeichert und im Ribbon-Baum angezeigt. Die Ribbon-Definition muss valide sein.

» **Ribbon aus USysRibbons einlesen**: Ermöglicht das Einlesen aller in der Tabelle **USysRibbons** der aktuell geöffneten Anwendung gespeicherten Ribbon-Definitionen.

Weitere Informationen zu diesen Themen finden Sie hier in den folgenden Abschnitten.

3.5.1 Ribbon neu anlegen

Der Kontextmenübefehl **Ribbon neu anlegen** eines Anwendungs-Elements im Ribbon-Baum legt eine neue Ribbon-Definition für die aktuell markierte Anwendung an. Unterhalt des Elements für die Ribbon-Definition beginnt der Teil dieses Zweiges im Ribbon-Baum, der die eigentlichen für die XML-Definition notwendigen Elemente enthält. Dazu müssen Sie in jedem Fall ein **customUI**-Element anlegen.

→ **»Das customUI-Element«, Seite 23**

3.5.2 Ribbon aus Datei laden

Der Kontextmenübefehl **Ribbon aus Datei laden** öffnet einen **Datei öffnen**-Dialog, mit dem Sie die gewünschte XML-Datei auswählen können. Nach Bestätigen per **Öffnen**-Schaltfläche liest der Ribbon-Admin die enthaltene Ribbon-Definition ein.

3.5.3 Ribbon aus USysRibbons einlesen

Möglicherweise enthält die aktuell geöffnete Datenbank bereits Ribbon-Definitionen in dem dafür vorgesehenen Ort. Der Ribbon-Admin kann diese verlustfrei einlesen, das heißt, dass alle darin definierten Elemente auch im Ribbon-Admin gespeichert und wieder zurückgeschrieben werden können. Natürlich können Sie auf diese Weise auch die Ribbon-Definitionen aus einer Anwendung importieren und einer anderen hinzufügen.

Mit dem Ribbon-Admin können Sie bestehende Ribbons weiterbearbeiten, sie zurück in die Anwendung schreiben oder auch in andere Anwendungen einfügen.

Um ein Ribbon zu importieren, gehen Sie folgendermaßen vor:

» Wenn Sie im Ribbon-Admin noch keine Anwendung angelegt haben, tun Sie dies nun mit dem Kontextmenüeintrag **Neue Anwendung** des Elements **Benutzerdefinierte Anwendungen** im Ribbon-Baum.

» Wählen Sie aus dem Kontextmenü des neuen Eintrags den Befehl **Ribbon aus USysRibbons einlesen**. Es erscheint nun der Dialog **Ribbon aus USysRibbons importieren** aus der folgenden Abbildung.

» Wählen Sie hier das zu importierende Ribbon aus und klicken Sie auf **OK**.

» Der Ribbon-Admin liest die Ribbon-Definition ein, sodass Sie diese direkt im Ribbon-Baum betrachten und bearbeiten können.

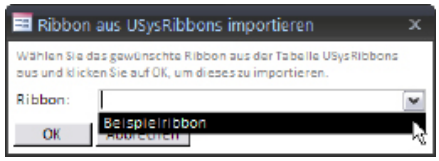


Abbildung 3.2: Auswählen eines Ribbons aus einer Anwendung für den Import in den Ribbon-Admin.

Weitere Schritte:

» Sie können das Ribbon nun bearbeiten und zurück in die ursprüngliche Anwendung schreiben oder

» Sie können das Ribbon nach dem Bearbeiten in eine andere Anwendung übertragen.

4 Ribbon-Elemente anlegen

Die folgenden Abschnitte erläutern die Vorgehensweise zum Anlegen der verschiedenen Elemente des Ribbons. Per Hyperlink können Sie zu den jeweils unter- beziehungsweise übergeordneten Elementen springen.

4.1 Das customUI-Element

Das **customUI**-Element ist das oberste Element in einer Ribbon-Definition. Sie legen es mit dem Kontextmenüeintrag **customUI** eines der Ribbon-Definitions-Elemente im Ribbon-Baum an.

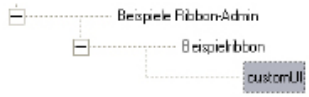


Abbildung 4.1: Beispiel für ein **customUI**-Element

Das Element besitzt zwei Callback-Attribute, für die Sie die Namen entsprechender VBA-Funktionen festlegen können. Die VBA-Funktionen fügt der Ribbon-Admin nach Wunsch ebenfalls zur aktuell geöffneten Anwendung hinzu.

→ **»Callback-Funktionen anlegen«, Seite 41**

Das **customUI**-Element kann zwei untergeordnete Elemente enthalten:

» Das **ribbon**-Element enthält die sichtbaren Elemente des Ribbons, also beispielsweise die Tabs der Ribbon-Leiste, das Office-Menü und die Schnellstartleiste.

» Das **commands**-Element kann lediglich **command**-Elemente enthaltenen, mit denen Sie die Befehle eingebauter Ribbon-Elemente erweitern oder ersetzen können.

Informationen über diese beiden Elemente finden Sie hier:

→ **»Das ribbon-Element«, Seite 23**

→ **»Das commands-Element«, Seite 39**

4.2 Das ribbon-Element

Das **ribbon**-Element enthält die sichtbaren Elemente des Ribbons: die **tab**-Elemente, die noch unterhalb eines **tabs**-Elements zusammengefasst werden, das Office-Menü (**officeMenu**-Ele-

ment), die Schnellzugriffsleiste (**qat**-Element) und die kontextabhängigen **tab**-Elemente, die sich unterhalb des **contextualTabs**-Elements und des **tabSet**-Elements befinden.

Abbildung 4.2: Beispiel für ein ribbon-Element im Ribbon-Baum

Das **ribbon**-Element legen Sie mit dem Kontextmenüeintrag **ribbon** eines Ribbon-Definitions-Elements an. Es enthält ein Attribut namens **startFromScratch**, mit dem Sie festlegen, ob die eingebauten Tabs des Ribbons und weitere Elemente der Benutzeroberfläche ausgeblendet werden.

Informationen über die untergeordneten Elemente finden Sie hier:

→ **»Das tabs-Element«, Seite 24**

→ **»Das qat-Element«, Seite 35**

→ **»Das contextualTabs-Element«, Seite 35**

4.3 Das tabs-Element

Das **tabs**-Element dient lediglich dazu, **tab**-Elemente zusammenzufassen. Ein **tabs**-Element legen Sie an, indem Sie im **ribbon**-Element den Kontextmenüeintrag **tabs** betätigen. Das **tabs**-Element besitzt keine Attribute.

Abbildung 4.3: Beispiel für ein tabs-Element

Die unterhalb des **tabs**-Elements gespeicherten **tab**-Elemente sind die ersten sichtbaren Elemente der Hierarchie des Ribbon-Baums.

→ **»Das tab-Element«, Seite 25**

4.4 Das tab-Element

Das **tab**-Element entspricht einer der Registerkarten der Ribbon-Leiste. Der Ribbon-Admin bietet drei Möglichkeiten, um **tab**-Elemente zu einer Ribbon-Definition hinzuzufügen. Alle drei sind über das Kontextmenü des übergeordneten **tabs**-Elements verfügbar:

» **tab**: Fügt ein benutzerdefiniertes **tab**-Element zum aktuellen **tabs**-Element hinzu.

» **Eingebautes tab**: Fügt eines der eingebauten **tab**-Elemente zum **tabs**-Element hinzu.

Detaillierte Informationen zu diesen drei Möglichkeiten finden Sie in den folgenden Abschnitten:

→ **»Benutzerdefinierte tab-Elemente hinzufügen«, Seite 25**

→ **»Eingebaute tab-Elemente hinzufügen«, Seite 38**

4.4.1 Benutzerdefinierte tab-Elemente hinzufügen

Wenn Sie einer Ribbon-Definition mit dem Kontextmenübefehl **tab** des **tabs**-Elements ein **tab**-Element hinzugefügt haben, können Sie dieses bereits anzeigen lassen. Dazu wählen Sie aus dem Kontextmenü des übergeordneten Ribbon-Elements den Eintrag **Ribbon anzeigen** aus.

Achtung: Auch, wenn Sie das Ribbon schon anzeigen können, ist es zu diesem Zeitpunkt noch nicht in der Zielanwendung gespeichert. Wie Sie das erledigen, erfahren Sie hier:

→ **»Ribbon in der Zielanwendung speichern«, Seite 53**

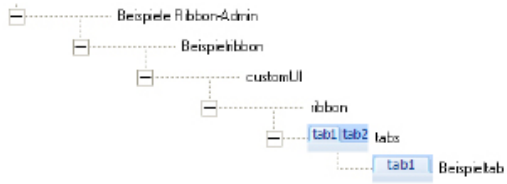


Abbildung 4.4: Beispiel für ein benutzerdefiniertes **tab**-Element im Ribbon-Baum

Abbildung 4.5: Im Ribbon sieht das einsame **tab**-Element aus obiger Abbildung so aus – vorausgesetzt, das Attribut **startFromScratch** des **ribbon**-Elements hat den Wert **true**.

Das **tab**-Element besitzt einige Attribute, von denen die folgenden für die Definition benutzerdefinierter Elemente zu Beginn am wichtigsten sind:

» **id**: Kennzeichnung, die innerhalb der Ribbon-Definition eindeutig sein mit einem Buchstaben beginnen muss.

» **label**: Beschriftung der Registerlasche

Wenn Sie gar kein benutzerdefiniertes **tab**-Element erstellen, sondern eines der eingebauten Tabs in das Ribbon Ihrer Anwendung einblenden möchten oder ein vorhandenes Tab anpassen möchten, um es beispielsweise um eingebaute Gruppen erweitern möchten, müssen Sie nicht das Attribut **id** verwenden, um ein individuelles **tab**-Element zu erzeugen, sondern mit dem Attribut **idMso** festlegen, welches der eingebauten Tabs Sie referenzieren möchten.

Der Ribbon-Admin vereinfacht diese Aufgabe, indem er Ihnen alle vorhandenen Tabs anbietet, auf die Sie über eine eindeutige **idMso** Bezug nehmen können.

Dazu brauchen Sie einfach nur mit der rechten Maustaste auf das Eingabefeld für das Attribut **idMso** zu klicken und im Kontextmenü den Eintrag **Eingebaute idMsos hinzufügen** auswählen. Es erscheint ein Dialog, mit dem Sie den gewünschten Eintrag auswählen können.

→ **»Eingebaute idMso auswählen«, Seite 37**

4.5 Das group-Element

Das **group**-Element entspricht einem einzelnen Bereich eines **tab**-Elements. Es ist der einzige Elementtyp, den ein **tab**-Element enthalten kann. Ein **group**-Element können Sie auf zwei Arten anlegen, die wiederum den Kontextmenüeinträgen des **tab**-Elements entsprechen:

» **group**: Fügt eine benutzerdefinierte Gruppe zum **tab**-Element hinzu.

» **Eingebaute group**: Fügt eine eingebaute Gruppe zum **tab**-Element hinzu.

Detaillierte Informationen gibt es hier:

→ **»Benutzerdefinierte group-Elemente hinzufügen«, Seite 26**

→ **»Eingebaute group-Elemente hinzufügen«, Seite 38**

4.5.1 Benutzerdefinierte group-Elemente hinzufügen

group-Elemente können Sie direkt nach dem Anlegen testweise anzeigen lassen. Dazu wählen Sie im Kontextmenü des übergeordneten Ribbon-Definitions-Element den Eintrag **Ribbon anzeigen** aus.

Achtung: Auch, wenn Sie das Ribbon schon anzeigen können, ist es zu diesem Zeitpunkt noch nicht in der Zielanwendung gespeichert. Wie Sie das erledigen, erfahren Sie hier:

→ **»Ribbon in der Zielanwendung speichern«, Seite 53**

Ein **group**-Element hat die folgenden wichtigen Attribute:

» **id**: Kennzeichnung, die innerhalb der Ribbon-Definition eindeutig und mit einem Buchstaben beginnen muss.

» **label**: Beschriftung des **group**-Elements am unteren Rand der Ribbon-Leiste

Das **group**-Element enthält die eigentlichen Steuerelemente des Ribbons, also zum Beispiel Schaltflächen, Auswahlfelder oder Kontrollkästchen.

→ **»Die Steuerelemente des Ribbons«, Seite 27**

Abbildung 4.6: Ein benutzerdefiniertes group-Element im Ribbon-Baum

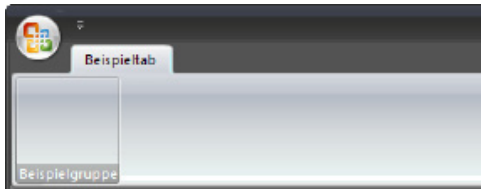


Abbildung 4.7: Das **group**-Element aus der obigen Definition in der Ribbon-Leiste

4.6 Die Steuerelemente des Ribbons

Das Ribbon bietet viel mehr Steuerelemente als Menü- und Symbolleisten. Wir verwenden im folgenden die im Ribbon-Sprachgebrauch üblichen Bezeichnungen, also beispielsweise Button oder Checkbox. Daher hier eine Übersicht der möglichen Steuerelemente:

» **control**: Generisches Element für eingebaute Objekte, erlaubt etwa das Einbauen des Schriftfarbe-Steuerelements in benutzerdefinierte **tab**-Elemente

» **dialogBoxLauncher**: Fügt einer Gruppe eine kleine Schaltfläche zum Öffnen eines Dialogs hinzu. Mögliche Unterelemente: **button** (maximal ein Element erlaubt)

- **dropDown**: Kombinationsfeld, das beim Ändern die angezeigte ID und den Index liefert. Mögliche Unterelemente: **item**
- **dynamicMenu**: Zur Laufzeit zu erzeugendes Menü. Mögliche Unterelemente: **button**, **toggleButton**, **checkBox**, **menu**, **gallery**, **splitButton**, **menuSeparator**, **dynamicMenu**, **control**
- **editBox**: Normales Textfeld.
- **gallery**: Galerie — wie Kombinationsfeld, jedoch mehrspaltig und komfortabler **button**, **item**
- **item**: Element eines **comboBox**-, **dropDown**- oder **gallery**-Steuerelements, wird beim dynamischen Zuweisen überschrieben
- **labelControl**: Steuerelement mit einer Beschriftung
- **menu**: Menü-Steuerelement. Mögliche Unterelemente: **button**, **toggleButton**, **checkBox**, **menu**, **gallery**, **splitButton**, **menuSeparator**, **dynamicMenu**, **control**
- **menuSeparator**: Trennstrich zwischen Menüpunkten
- **separator**: Trennstrich zwischen Steuerelementen
- **splitButton**: Kombination aus Schaltfläche und Menü wie etwa das Steuerelement zum Einstellen der Objektansicht. Mögliche Unterelemente: **button**, **toggleButton**
- **toggleButton:** Umschaltfläche

Weitere Informationen zu den Steuerelementen:

→ **»Das button-Element«, Seite 28**

→ **»Das box-Element«, Seite 30**

→ **»Das checkBox-Element«, Seite 32**

→ **»Das toggleButton-Element«, Seite 30**

Informationen zu den weiteren Elementen finden Sie in den angehängten Beispiel-Kapiteln aus *Das Access-Entwicklerbuch*:

→ **»Ribbon«, Seite 59**

→ **»Backstage«, Seite 123**

4.6.1 Das button-Element

Das **button**-Element liefert die gemeine Schaltfläche des Ribbons. Es kann beim Anklicken eine Callback-Funktion aufrufen. Es kann eine Beschriftung, ein Symbol oder beides enthalten und in zwei Größen angezeigt werden, wobei die größere nur in Verbindung mit einem Symbol sinnvoll

ist. Benutzerdefinierte Symbole sollten eine Größe von 16 x 16 Pixel für normale und 32 x 32 Pixel für große **button**-Elemente und einen transparenten Hintergrund aufweisen, weshalb als Quelle in erster Linie .**png**-Dateien verwendet werden.

Benutzerdefinierte Schaltflächen werden häufig verwendet, um eigene VBA-Prozeduren aufzurufen.

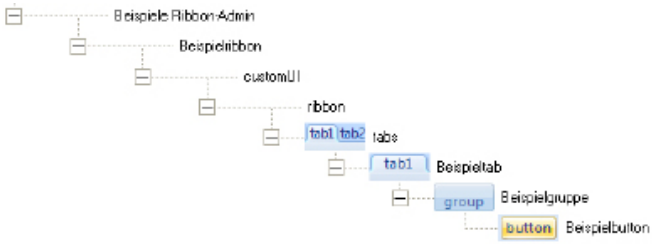


Abbildung 4.8: Eine einfache Schaltfläche im Ribbon-Baum ...

Abbildung 4.9: ... und im Ribbon.

Viel besser kommt eine Schaltfläche natürlich zur Geltung, wenn Sie ein Bild hinzufügen. Dafür gibt es zwei Methoden: Entweder Sie legen eines der vielen eingebauten Symbole fest oder Sie verwenden ein benutzerdefiniertes Bild. Mehr zu diesen Varianten erfahren Sie hier:

→ **»Benutzerdefinierte Bilder zu Steuerelementen hinzufügen«, Seite 49**

→ **»Eingebaute Bilder zu Steuerelementen hinzufügen«, Seite 50**

Größe von Schaltflächen

Die Größe bestimmter Schaltflächen legen Sie über das **size**-Attribut fest. Es stehen die Werte **normal** (Standard) und **large** zur Verfügung, die Sie in der Attribut-Liste des button-Elements finden.

Das resultierende Ribbon sieht so aus:

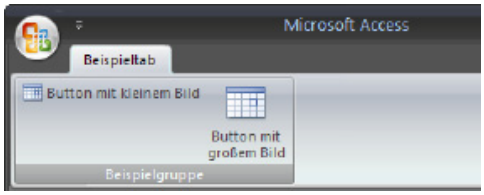


Abbildung 4.10: Ribbon-Button in kleiner und großer Ausführung

Schaltflächen mit Funktion versehen

Der Ribbon-Admin unterstützt Sie beim Anlegen von Callback-Funktionen, die beim Anklicken von **button**-Elementen ausgeführt werden sollen. Weitere Informationen finden Sie hier:

→ **»Callback-Funktionen anlegen«, Seite 41**

4.6.2 Das toggleButton-Element

Das **toggleButton**-Element entspricht einer Umschaltfläche und bildet, wenn man ein Beispiel aus dem richtigen Leben sucht, etwa einen Lichtschalter ab, der die zwei Zustände **Gedrückt** und **Nicht gedrückt** zur Wahl stellt. Das Ribbon stellt das **toggleButton**-Element im nicht gedrückten Zustand wie eine normale Schaltfläche und im gedrückten Zustand in oranger Farbe dar. Damit es die Farbe wechselt, brauchen Sie überhaupt nichts zu tun – außer das Steuerelement anzuklicken. Wenn Sie das **toggleButton**-Element ernsthaft einsetzen möchten, können Sie eine ganze Menge mehr damit anstellen, müssen aber auch einen nicht unwesentlichen Aufwand betreiben.

Legen Sie zunächst ein **toggleButton**-Element an, indem Sie den Eintrag **toggleButton** des Kontextmenüs des **group**-Elements anklicken. Die restlichen Schritte finden Sie im folgenden Abschnitt:

→ **»get...-Callbacks anlegen«, Seite 44**

4.6.3 Das box-Element

Das **box**-Element dient der Gruppierung von Steuerelementen. Es fasst Steuerelemente zu horizontalen oder vertikalen Gruppen zusammen.

Im Ribbon-Baum sieht das beispielsweise so aus:

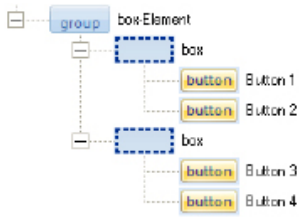


Abbildung 4.11: Anordnung von **box**- und **button**-Elementen im Ribbon-Baum

Das führt zu folgendem Ribbon:

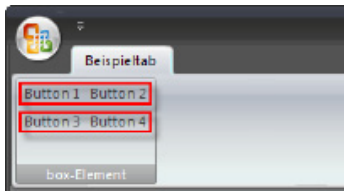


Abbildung 4.12: Zweimal horizontale Boxen mit je zwei Steuerelementen

4.6.4 Das buttonGroup-Element

Das **buttonGroup**-Element fasst die enthaltenen Steuerelemente optisch zu einem einzigen Steuerelement zusammen, wobei ein Rahmen für den Zusammenhalt sorgt und Trennlinien kenntlich machen, dass die Gruppe dennoch verschiedene Elemente enthält. Im Ribbon-Baum sieht das so aus:

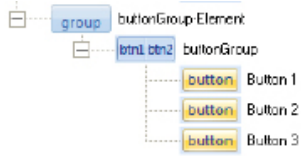


Abbildung 4.13: Ein **buttonGroup**-Element im Ribbon-Baum ...

Die Ribbon-Leiste zeigt eine **buttonGroup** beispielsweise so an:

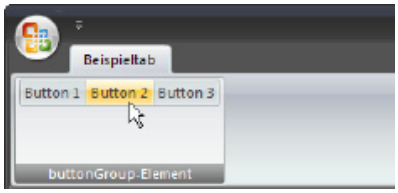


Abbildung 4.14: ... und im Ribbon selbst.

4.6.5 Das checkBox-Element

Kontrollkästchen verwendet man häufig zum Darstellen von Daten, die den Wert **Ja** oder **Nein** annehmen können (beziehungsweise **true** und **false**, **-1** und **0** und so weiter).

Sie bestehen aus einer Beschriftung und dem eigentlichen Kontrollkästchen. Das Kontrollkästchen stellt den Wert **true** beziehungsweise **-1** durch ein gesetztes Häkchen dar.

Neben den **Get...**-Attributen besitzt das **checkBox**-Steuerelement mit **onAction** ein Ereignis-Callback. Die Definition der entsprechenden VBA-Callback-Funktion unterscheidet sich von der des **button**-Elements: Sie liefert mit dem Parameter **pressed** nämlich noch einen Boolean-Wert mit, der angibt, ob das auslösende Kontrollkästchen markiert ist oder nicht.

onAction-Callback

Im folgenden Beispiel soll ein **checkBox**-Steuerelement beim Anklicken seinen Namen anzeigen und außerdem mitteilen, ob es gerade markiert ist oder nicht. Dazu legen Sie ein solches Konstrukt im Ribbon-Baum an:

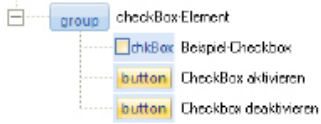


Abbildung 4.15: Ein **checkBox**-Element mit zwei Schaltflächen zum Aktivieren und Deaktivieren

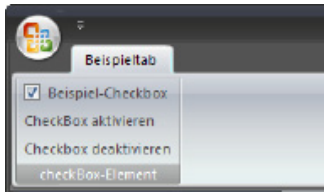


Abbildung 4.16: Die Checkbox im Einsatz

Sie können den Zustand eines **checkBox**-Elements und auch der anderen Elemente nicht direkt abfragen, wie das etwa mit den Steuerelementen in einem Access-Formular möglich ist. Stattdessen müssen Sie diesen beim Anklicken in einer Variablen speichern und später auf diesen gespeicherten Wert zugreifen. Dazu legen Sie eine entsprechende Variable an:

```
Dim bolCheckBox As Boolean
```

Den Wert setzen Sie in einer Callback-Funktion, die durch das Attribut **onAction** ausgelöst wird und die im Parameter **pressed** einen Boolean-Wert mit dem aktuellen Zustand liefert:

```
Sub onActionCheckbox(control As IRibbonControl, pressed As Boolean)
    bolCheckBox = Not pressed
End Sub
```

Die Variable **bolCheckBox** müssen Sie im Kopf des Moduls noch selbst deklarieren:

```
Dim bolToggleButton As Boolean
```

Wenn Sie das **checkBox**-Element von außen einstellen möchten, also beispielsweise mit den beiden Schaltflächen aus der obigen Konstellation, brauchen Sie ein **IRibbonUI**-Objekt und dessen **Invalidate**- oder **InvalidateControl**-Methode. Diese stellen Sie wie hier beschrieben bereit:

→ **»Invalidate und InvalidateControl nutzen«, Seite 45**

In unserem Beispiel sehen die onAction-Callback-Funktionen der beiden Schaltflächen so aus:

```
Sub onActionCheckboxAktivieren(control As IRibbonControl)
    bolCheckBox = True
    objRibbon.InvalidateControl "chk1"
End Sub
```

```
Sub onActionCheckboxDeaktivieren(control As IRibbonControl)
    bolCheckBox = False
    objRibbon.InvalidateControl "chk1"
End Sub
```

Beide Funktionen sorgen dafür, dass die weiter unten beschriebene **getPressed**-Callback-Funktion ausgelöst wird.

Übrigens können Sie den Wert der Checkbox auch direkt beim Anzeigen festlegen. Dazu stellen Sie die Variable **bolCheckBox** beim Laden des Ribbons, also beispielsweise in der Callback-Funktion **onLoad**, auf den gewünschten Wert ein:

```
bolCheckBox = True
```

Anschließend verwenden Sie die getPressed-Callback-Funktion, um diesen Wert zuzuweisen:

```
Sub getPressedCheckbox(control As IRibbonControl, ByRef returnValue)
    returnValue = bolCheckBox
End Sub
```

4.6.6 Weitere Steuerelemente

Das Ribbon bietet noch viele weitere Steuerelemente, auf die wir hier im Detail nicht eingehen. Die Handhabung mit dem Ribbon-Admin erfolgt allerdings wie mit den bereits beschriebenen Elementen. Informationen zu fast allen weiteren Steuerelementen finden Sie hier:

→ **»Ribbon«, Seite 59**

→ **»Backstage«, Seite 123**

4.7 Die Schnellzugriffsleiste anpassen

Wenn Sie die Schnellzugriffsleiste anpassen möchten, legen Sie Elemente für die Anwendung und für das Ribbon-Definition an, fügen ein **customUI**-Element und ein **ribbon**-Element hinzu und erstellen dann ein **qat**-Element. Für eine Access-Anwendung legen Sie darunter ein **documentControls**-Element und in diesem schließlich die eigentlichen Steuerelemente an, wobei nur die drei Elemente **button**, **control** und **separator** möglich sind.

4.7.1 Das qat-Element

Das **qat**-Element können Sie nur unterhalb des **ribbon**-Elements anlegen. Es enthält wiederum eines der beiden Elemente **documentControls** (unter Access üblicherweise verwendet) oder **sharedControls**. Erst darin folgen die eigentlichen Steuerelemente, bei denen es sich um **button**-, **control**- oder **separator**-Elemente handeln kann. Hier ist zu beachten, dass nur eingebaute Elemente ohne Probleme arbeiten. Benutzerdefinierte Elemente werden oft nicht direkt angezeigt, sondern erst nach dem Wechseln des Fokus zu einem anderen Element (wie einem Formular) und zurück.

Das **qat**-Element können Sie nur verwenden, wenn das Attribut **startFromScratch** des **ribbon**-Elements den Wert **true** besitzt.

4.8 Das contextualTabs-Element

Das **contextualTabs**-Element benötigen Sie, wenn ein Formular oder ein Bericht zusätzliche Ribbon-Anpassungen zum bestehenden Ribbon vornehmen soll, also wenn Sie beispielsweise ein formularspezifisches **tab**-Element rechts neben den anderen Tabs einfügen wollen und dieses direkt angezeigt werden soll – was bei normalen **tab**-Elementen nicht der Fall ist.

Die folgende Konfiguration im Ribbon-Baum bewirkt die Anzeige eines Ribbons wie in der darauffolgenden Abbildung. Wichtig sind drei Dinge:

» Das Attribut **idMso** des **tabSet**-Elements muss den speziellen Wert **TabSetFormReportExtensibility** enthalten.

» Das Ribbon muss einem Formular oder Bericht zugewiesen sein, um damit eingeblendet zu werden.

» Das **tab**-Element wird nicht beim Testen über den Kontextmenüeintrag **Ribbon anzeigen** des Ribbon-Definitions-Elements eingeblendet, sondern nur angelegt.

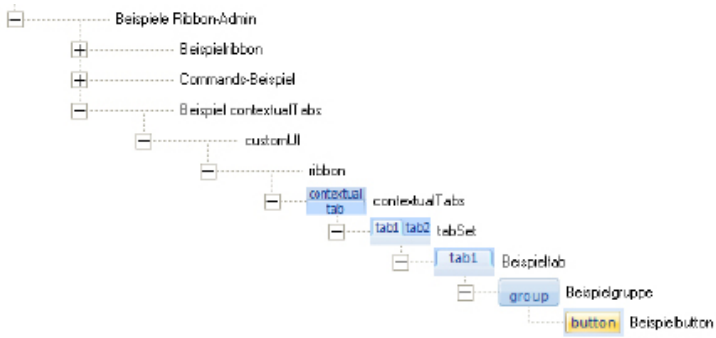


Abbildung 4.17: Der Aufbau eines **contextualTabs**. Das Attribut **idMso** des **tabSet**-Elements muss den Wert **TabSetFormReportExtensibility** aufweisen.

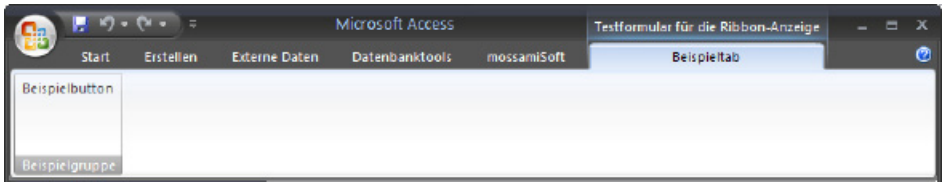


Abbildung 4.18: Ein kontextsensitives **tab**-Element

5 Eingebaute Ribbon-Elemente

Möglicherweise möchten Sie eingebaute Elemente des Ribbons in eigene Ribbon-Definitionen einbauen oder diese im Original-Ribbon anpassen. Dabei helfen die folgenden Abschnitte.

5.1 Eingebaute idMso auswählen

Für Elemente wie **tabSet**, **tab**, **group** oder die verschiedenen Steuerelemente wie **button**, **editbox** oder **comboBox** können Sie das Aussehen und die Funktionalität eingebauter Elemente festlegen. Dazu stellen Sie das Attribut **idMso** des jeweiligen Elements auf eine bestimmte Zeichenkette ein, die beispielsweise für die **Speichern**-Schaltfläche **FileSave** heißt. Auf diese Weise können Sie alle eingebauten Elemente referenzieren.

Der Ribbon-Admin vereinfacht dies, indem er Ihnen den Kontextmenüeintrag **Eingebaute idMso hinzufügen** für das Attribut **idMso** der genannten Elemente anbietet. Dieser Befehl öffnet einen Dialog, mit dem Sie die gewünschte **idMso** auswählen können.

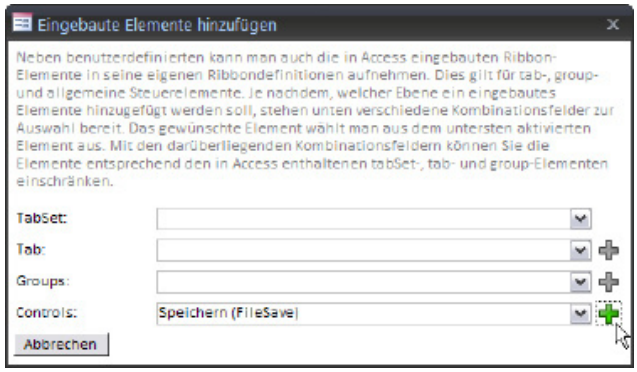


Abbildung 5.1: Dialog zur Auswahl der idMso eingebauter Elemente

Dieser Dialog dient gleichzeitig zur Auswahl der Elemente **tabSet**, **tab**, **group** und **control**. Je nachdem, welche Steuerelementart gerade markiert ist, blendet der Dialog die darunterliegenden Elemente aus – wenn Sie also etwa die **idMso** für ein **tab**-Element auswählen möchten, dann werden die beiden Auswahlfelder für die Groups und die Controls deaktiviert. Andersherum wird das Plus-Zeichen rechts neben dem Auswahlfeld nur für das aktuell zu bestückende Steuerelement grün markiert und somit aktiviert. Ein Klick auf dieses Plus-Steuerelement schließt den Dialog und fügt die **idMso** in das entsprechende Attribut ein.

Es gibt zwei Strategien für die Auswahl der **idMso**: Wenn Sie die Bezeichnung des betreffenden Steuerelements kennen, können Sie dieses einfach im Auswahlfeld für die entsprechende

Steuerelementart eingeben oder auswählen, ohne die darüber liegenden Auswahlfelder zu verwenden.

Sie können diese aber auch zum Filtern der Einträge des jeweils darunter befindlichen Auswahlfelds verwenden.

5.2 Eingebaute tab-Elemente hinzufügen

Wenn Sie aus dem Kontextmenü des **tabs**-Elements den Eintrag **Eingebautes tab** auswählen, erscheint der weiter oben bereits erwähnte Dialog zur Auswahl der idMso eingebauter Elemente.

→ **»Eingebaute idMso auswählen«, Seite 37**

Der Dialog arbeitet in diesem Zusammenhang zwar genauso, wie oben beschrieben, jedoch sieht das Resultat anders aus. Nach dem Schließen des Dialogs legt der Ribbon-Admin nämlich ein neues **tab**-Element an, aber er versieht es nicht mit dem entsprechenden Wert für das **idMso**-Attribut: Ein **tab**-Element mit der **idMso** eines eingebauten Tabs würde Access nämlich einfach nicht anzeigen – über das Warum gibt es keine Informationen. Der Ribbon-Admin erzeugt stattdessen ein neues **tab**-Element mit einem eindeutigen Wert für das Attribut **id** und legt darunter die eingebauten **group**-Elemente an, die das eingebaute tab-Element normalerweise liefern sollte.

5.3 Eingebaute group-Elemente hinzufügen

Das Hinzufügen eingebauter **group**-Elemente erfolgt mit dem gleichen Dialog wie das Anlegen von **tab**-Elementen.

→ **»Eingebaute tab-Elemente hinzufügen«, Seite 38**

Allerdings wählen Sie dort ein Element aus der dritten Auswahlliste aus. Technisch gibt es weitere Unterschiede, weil **group**-Elemente auf Basis einer **idMso** ihre Steuerelemente automatisch mitliefern und darstellen – die weiter oben beschriebenen **tab**-Elemente tun das nicht, sondern müssen um die enthaltenen **group**-Elemente ergänzt werden.

5.4 Eingebaute Steuerelemente hinzufügen

Eingebaute Steuerelemente fügen Sie auf etwas andere Weise zum Ribbon hinzu. Dazu erstellen Sie zunächst ein Steuerelement des Typs **control** und wählen dann aus dem Kontextmenü des Attributs **idMso** den Eintrag **Eingebaute idMso hinzufügen** aus.

Es erscheint der gleiche Dialog wie oben, nur dass Sie damit nur Elemente aus dem unteren Auswahlfeld auswählen können.

5.5 Neue Funktionen für eingebaute Steuerelemente

Manchmal brauchen Sie gar keine neuen Elemente im Ribbon, sondern passen die Funktionen der vorhandenen Elemente an. Dies erledigen Sie, indem Sie je ein Element für die Anwendung und für die Ribbon-Definition erstellen, ein **customUI**-Element und darunter ein **commands**-Element mit einem oder mehreren **command**-Elementen anlegen.

5.5.1 Das commands-Element

Das **commands**-Element kann eines oder mehrere **command**-Elemente enthalten, denen Sie eine der eingebauten Ribbon-Funktionen zuweisen und dafür eine neue Funktion schreiben können, welche die eingebaute Funktion ersetzt oder ergänzt.

5.5.2 Das command-Element

Das **command**-Element enthält die eigentlichen Informationen zu dem anzupassenden Steuerelement.

Das Beispiel aus der folgenden Abbildung zeigt den grundsätzlichen Aufbau:

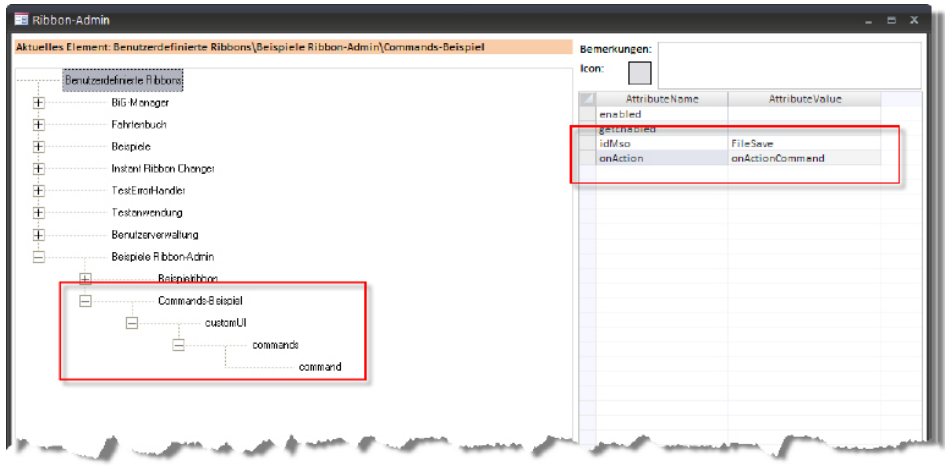


Abbildung 5.2: Diese Ribbon-Definition fügt eine zusätzliche Funktion zur Speichern-Schaltfläche hinzu.

Die **idMso** können Sie mit dem passenden Dialog auswählen, der erscheint, wenn Sie den Kontextmenüeintrag dieses Attributs betätigen.

Die Callback-Funktion fügen Sie hinzu, indem Sie einfach die Bezeichnung **onActionCommand** für das Attribut einfügen und dann den Kontextmenüeintrag **Callback hinzufügen** auswählen. Diese wird im Modul **mdlRibbons** angelegt und sieht so aus:

```
Sub onActionCommand(control As IRibbonControl, ByRef CancelDefault)
    If MsgBox("Wirklich speichern?", vbYesNo) = vbYes Then
        CancelDefault = True
    Else
        CancelDefault = False
    End If
End Sub
```

Beim Klicken auf die **Speichern**-Schaltfläche erscheint nun eine Meldung, die fragt, ob man tatsächlich speichern möchte. Klicken Sie dort auf Nein, geschieht nichts.

6 Callback-Funktionen anlegen

Callback-Funktionen sind das Salz in der Ribbon-Suppe. Wenn Sie ein Ribbon-Steuerelement mit benutzerdefinierten Funktionen ausstatten möchten, brauchen Sie erstens eine entsprechende VBA-Routine in einem bestimmten Format und müssen zweitens ein Callback-Attribut mit dem Namen dieser Routine füllen. Den Großteil dieser Arbeit nimmt Ihnen der Ribbon-Admin ab.

Sie brauchen nur die folgenden Schritte durchzuführen:

Suchen Sie aus den Attributen des Steuerelements das gewünschte Callback-Attribut aus, zum Beispiel das Attribut **onAction** eines **button**-Elements.

Klicken Sie mit der rechten Maustaste darauf und wählen Sie den Eintrag **Callback hinzufügen** aus. Der Ribbon-Admin legt daraufhin eine neue Callback-Funktion mit dem Namen des Callback-Attributs an (hier **onAction**) und trägt diese Bezeichnung für das Attribut ein. Alternativ tragen Sie zuerst einen selbstgewählten Namen für die Callback-Funktion ein und legen diese dann mit dem bereits erwähnten Eintrag des Kontextmenüs an.

Beim Arbeiten mit Callback-Funktionen brauchen Sie einige Objekte der Bibliothek Microsoft Office 12.0 Object Library. Der Ribbon-Admin prüft, ob diese bereits vorhanden ist und legt diese gegebenenfalls an. Die folgende Meldung berichtet von dieser Aktion:

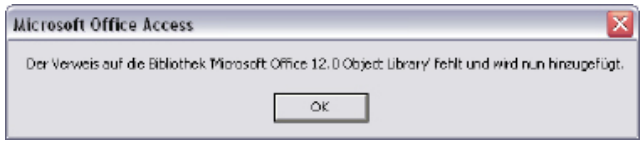


Abbildung 6.1: Hinzufügen eines Verweises auf die Office-Bibliothek

Wenn Sie noch keine Callback-Funktionen mit dem Ribbon-Admin angelegt haben und auch noch kein Modul namens **mdlRibbons** von Hand erstellt haben, erledigt der Ribbon-Admin dies. Zuvor weist er allerdings noch auf die anstehende Aktion hin:

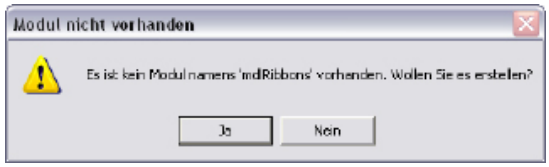


Abbildung 6.2: Der Ribbon-Admin legt ein Modul für die Callback-Funktionen an.

In diesem Fall wird ebenfalls noch keine entsprechende Callback-Funktion in diesem Modul vorhanden sein. Der Ribbon-Admin fügt auch dieses hinzu:

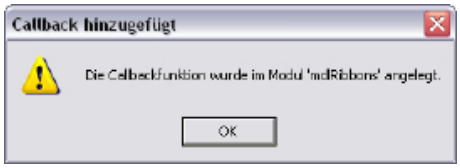


Abbildung 6.3: Auch auf das Anlegen von Callback-Funktionen weist der Ribbon-Admin hin.

Ihre Anwendung enthält nun ein Modul **mdlRibbons** mit einer Callback-Funktion, die beim Klick auf das **button**-Element ausgelöst wird. Wenn Sie hier noch eine Zeile wie die folgende einfügen, können Sie die Schaltfläche schon testen:

```
Sub onAction(control As IRibbonControl)
    MsgBox "Dies war ein Klick auf die Schaltfläche '" _
        & control.id & "'."
End Sub
```

6.1 Ribbon-Objekt speichern

Auch wenn der Ribbon-Admin versucht, das Erstellen von Ribbons so unkompliziert wie möglich zu machen, kommen Sie in vielen Fällen nicht um den Einsatz von mehr VBA-Routinen als den reinen Callback-Funktionen herum. Das ist beispielsweise so, wenn Sie die **get...**-Callbacks verwenden möchten. Dabei handelt es sich um solche Callbacks, die durch eine der Methoden **Invalidate** oder **InvalidateControl** des Ribbon-Objekts ausgelöst werden. Diese Methoden ermöglichen das Aktualisieren der Attribute des Ribbons zur Laufzeit, etwa wenn der Benutzer eine Aktion tätigt, die den Status eines Ribbon-Steuerelements ändern soll.

Um die **get...**-Callbacks aufzurufen, müssen Sie folgendes tun:

» Speichern eines Verweises auf das Ribbon und seine Steuerelemente

» Anlegen einer Callbackfunktion für das Attribut **onLoad** des **customUI**-Elements, welche die aktuelle Ribbon-Definition in einem VBA-Objekt speichert

» Anlegen von **get...**-Attributen für die betroffenen Steuerelemente, also etwa das Attribut **getLabel** eines **toggleButton**-Elements, dessen Beschriftung zur Laufzeit geändert werden soll

» Schreiben von Anweisungen, welche die **Invalidate**- oder die **InvalidateControl**-Methoden des Ribbon-Objekts auslösen und in der Folge die **get...**-Callbacks aufrufen.

Weitere Informationen finden Sie hier:

→ **»IRibbonUI-Objekt speichern«, Seite 43**

→ »onLoad-Callback anlegen«, Seite 43

→ »get...-Callbacks anlegen«, Seite 44

→ »Invalidate und InvalidateControl nutzen«, Seite 45

6.2 IRibbonUI-Objekt speichern

Ein **IRibbonUI**-Objekt ist das Objekt, über dessen **Invalidate**- und **InvalidateControl**-Methode Sie alle oder auch nur ein bestimmtes Ribbon-Element aktualisieren können, was bedeutet, dass all seine **get...**-Callbacks ausgeführt werden.

Für das Speichern dieses Objekts brauchen Sie zunächst die folgende Variable (die Sie aber nicht selbst anlegen müssen, weil sie im nächsten Schritt automatisch mit erzeugt wird):

```
Dim objRibbon As IRibbonUI
```

Es gibt nur eine Gelegenheit, diese Variable zu füllen, und zwar in der Callback-Funktion zum Attribut **onLoad** des **customUI**-Elements. Legen Sie eine entsprechende Callback-Funktion mit dem einzigen Kontextmenüeintrag dieses Elements an, die wie folgt aussieht:

```
Sub onLoad(ribbon As IRibbonUI)
    Set objRibbon = ribbon
End Sub
```

Über dieses Objekt können Sie nun die Methoden **Invalidate** und **InvalidateControl** aufrufen.

→ »Invalidate und InvalidateControl nutzen«, Seite 45

6.3 onLoad-Callback anlegen

Um eine Callback-Funktion für das Attribut **onLoad** anzulegen, brauchen Sie nur den Eintrag **Callback hinzufügen** des Kontextmenüs des **customUI**-Elements einer Ribbon-Definition auszuwählen. Der Ribbon-Admin legt dann eine Callback-Funktion namens **onLoad** an und stellt auch das Attribut auf den Namen der Funktion ein. Sie können auch zunächst einen alternativen Namen für das Attribut eintragen und dann den Kontextmenübefehl ausführen, um die Callback-Funktion mit diesem Namen anzulegen. Das Ergebnis sieht etwa wie in der folgenden Abbildung aus.

Abbildung 6.4: Eine vom Ribbon-Admin angelegte Callback-Funktion

Zu beachten ist hier, dass der Ribbon-Admin auch gleich eine Variable zum Speichern des **IRibbonUI**-Objekts anlegt, die Sie etwa für das Aufrufen der **Invalidate**- und der **InvalidateControl**-Methoden benötigen.

6.4 get...-Callbacks anlegen

get...-Callback-Funktionen benötigen Sie, wenn Sie den Zustand von Steuerelementen im Ribbon zur Laufzeit anpassen möchten. Ein **toggleButton**-Steuerelement beispielsweise könnte im gedrückten Zustand die Beschriftung **Aktiv** und im nicht gedrückten Zustand die Beschriftung **Nicht aktiv** anzeigen.

Für ein Beispiel legen Sie in einem **group**-Element ein **toggleButton**-Steuerelement an, bei dem Sie nicht das **label**-, sondern das **getLabel**-Attribut füllen, und zwar mit dem Namen der Funktion, die den jeweils aktuellen Wert für die Beschriftung liefern soll. Diese soll in diesem Fall **GetLabelToggleButton** heißen. Klicken Sie dann mit der rechten Maustaste auf das Attribut und wählen Sie aus dem Kontextmenü den Eintrag **Callback anlegen** aus. Der Ribbon-Admin sollte dies mit der folgenden Meldung quittieren – vorausgesetzt, das Modul **mdlRibbons** existiert bereits, sonst erscheint zuvor noch mindestens eine weitere Meldung:

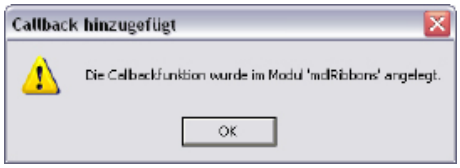


Abbildung 6.5: Die Callback-Funktion wurde erfolgreich angelegt.

Die neu hinzugefügte Funktion sieht so aus:

```
Sub GetLabelToggleButton(control As IRibbonControl, ByRef label)

End Sub
```

Diese Routine wird nun ausgelöst, wenn man entweder die **Invalidate**- oder die **InvalidateControl**-Methode mit dem Namen des Steuerelements (hier **tgl1**) als Parameter aufruft. Das ist hier erstmal weniger wichtig und wird außerdem hier erläutert:

→ **»Invalidate und InvalidateControl nutzen«, Seite 45**

Viel wichtiger ist die Funktion der Callback-Funktion **GetLabelToggleButton**. Diese liefert mit **control** einen Verweis auf das aufrufende Steuerelement (seinen Namen können Sie beispielsweise über **control.id** auslesen). Der zweite Parameter namens **label** ist hier interessanter: Er erwartet nämlich den Wert, den das Steuerelement als Beschriftung anzeigen soll. Und diesen füllen Sie mit einer einfachen Anweisung wie etwa der folgenden:

```
label = "Ich bin eine toggleButton-Beschriftung!"
```

Das hilft noch nicht weiter, denn auf diese Weise ändert sich die Beschriftung nie. Das soll sie aber, in Abhängigkeit vom Zustand des **toggleButton**-Elements, tun. Dazu muss aber erstmal jemand auf den **toggleButton** drücken und damit sein Ereignis **onAction** auslösen, das hier **OnActionToggleButton** heißen soll. Die Callback-Funktion legen Sie wie gewohnt über das Kontextmenü an. Die Funktion sieht so aus:

```
Sub onAction(control As IRibbonControl, pressed As Boolean)

End Sub
```

Im Gegensatz zur Callback-Funktion des **button**-Elements liefert es noch einen Parameter mehr, der den Zustand des **toggleButton**-Elements angibt. Den Wert dieses Zustands speichern Sie einfach in einer Variablen des Typs **Boolean**, die Sie wie folgt im Kopf des Moduls **mdlRibbons** deklarieren:

```
Dim bolToggleButton As Boolean
```

6.5 Invalidate und InvalidateControl nutzen

Die **Invalidate**-Methode sorgt dafür, dass die **get...**-Callbacks aller Steuerelemente des Ribbons aufgerufen werden, die **InvalidateControl**-Methode erledigt dies nur für das Steuerelement mit der im Parameter angegebenen **id**.

Diese Methoden können Sie überall aufrufen: Zum Testen im Direktfenster, und im Code überall dort, wo es nötig ist. Das sind normalerweise Stellen in Prozeduren, die durch Aktionen des Benutzers ausgelöst werden, wie das weiter oben beschriebene Anklicken eines **toggleButton**-Elements. Wichtig ist, dass Sie zuvor einen Verweis auf das **IRibbonUI**-Element erstellt und in ei-

ner Objektvariablen gespeichert haben, welches das aktuelle Ribbon und seine Steuerelemente repräsentiert und die beiden genannten Methoden zur Verfügung stellt.

Ein Klick auf das **toggleButton**-Element löst nun die folgende Routine aus, die erstens den **Gedrückt**-Zustand in der Variablen **bolToggleButton** speichert und zweitens die **InvalidateControl**-Methode mit dem Namen des **toggleButton**-Elements aufruft, um dieses zu aktualisieren:

```
Sub OnActionToggleButton(control As IRibbonControl, pressed As Boolean)
    bolToggleButton = pressed
    objRibbon.InvalidateControl "tgl1"
End Sub
```

Die für das Attribut **getLabel** angelegte Callback-Funktion bestücken Sie hingegen wie folgt, damit diese einfach den **Gedrückt**-Zustand aus der Variablen **bolToggleButton** ausliest und den entsprechenden Text als Beschriftung des **toggleButton**-Elements verwendet:

```
Sub GetLabelToggleButton(control As IRibbonControl, ByRef label)
    If bolToggleButton = True Then
        label = "Aktiv"
    Else
        label = "Nicht aktiv"
    End If
End Sub
```

Wer diesen Ablauf verstanden und verinnerlicht hat, der muss bei der Programmierung des Ribbons kaum noch mit Überraschungen rechnen.

6.6 loadImage-Callback anlegen

Die für das **loadImage**-Attribut hinterlegte Callbackfunktion wird beim Laden des Ribbons für jedes Steuerelement, dessen Attribut **image** einen Wert enthält, aufgerufen. Wenn Sie die Callback-Funktion über das Kontextmenü des **customUI**-Elements angelegt haben, finden Sie im Modul **mdlRibbons** eine Routine vor, die diese Aufgabe übernimmt und die in der Tabelle **USysImages** gespeicherten Bilder im Ribbon anzeigt.

```
Public Sub loadImage(control, ByRef image)
    ...
End Sub
```

Der Parameter **imageID** liefert die Zeichenkette, die im Steuerelement für das Attribut **image** hinterlegt ist. Dies ist üblicherweise der Name einer Bilddatei wie beispielsweise **add.png**.

Der zweite Parameter namens **image** erwartet ein Objekt des Typs **StdPicture** mit einem Verweis auf die anzuzeigende Bilddatei.

6.7 getImage-Callback anlegen

Das **getImage**-Callback wird bei jedem Aufruf von **Invalidate** (für alle Steuerelemente) oder **InvalidateControl** (nur für das angegebene Steuerelement) ausgelöst.

Die Callback-Funktion ist genauso wie die für das **loadImage**-Callback aufgebaut, weist aber einen entscheidenden Unterschied auf. Sie erhält mit dem Parameter **control** nicht den Namen des für das Attribut **image** gespeicherten Werts, sondern ein **IRibbonControl**-Objekt, über das man mit der Eigenschaft **id** den Namen des Steuerelements oder mit der Eigenschaft **tag** den Wert des Attributs **tag** ermitteln kann.

Das **image**-Attribut können Sie genau genommen gar nicht belegen, wenn Sie das **getImage**-Attribut verwenden – dies führt zu einem Fehler. Also müssen Sie der **getImage**-Callback-Funktion auf andere Weise den Namen des zu ladenden Images mitteilen – und das ist die **tag**-Eigenschaft des übergebenen **IRibbonControl**-Objekts.

Sie brauchen bei Verwendung des **getImage**-Callback-Attributs also einfach nur das **tag**-Attribut des Ribbon-Elements mit dem Namen des Images, also beispielsweise **car.png**, zu füllen. Die Routine wertet dies wie im folgenden Code-Auszug aus::

```
Sub getImage(control As IRibbonControl, ByRef image)
    ...
    lngID = Nz(DLookup("ID", "USysImages", "ImageName = '" & control.Tag & "'"), 0)
    ...
    Set image = PicFromField(CurrentDb.OpenRecordset("SELECT ImageObject " _
        & "FROM USysImages WHERE ID = " & lngID).Fields(0))
    ...
End Sub
```

Dazu gehen Sie am besten so vor:

» Legen Sie das Steuerelement, beispielsweise ein **button**-element, an und geben Sie **id** und **label** ein.

» Wählen Sie mit dem Kontextmenüeintrag des Attributs **image** ein Bild aus und speichern Sie es in der Zieldatenbank.

» Schneiden Sie den Inhalt des Attributs **image** aus und tragen Sie ihn für das Attribut **tag** ein.

» Legen Sie mit dem Kontextmenüeintrag des Attributs **getImage** die entsprechende Callback-Funktion an – fertig!

7 Bilder im Ribbon

Die Optik des Ribbons steht und fällt mit den darin angezeigten Bildern. Es gibt zwei Möglichkeiten, ein Bild für ein Steuerelement festzulegen – entweder über das Attribut **image** (benutzerdefinierte Bilder) oder das Attribut **imageMso** (eingebaute Bilder). Bei benutzerdefinierten Bildern brauchen Sie zusätzlich brauchen noch eine VBA-Routine, welche die Bilder einliest.

7.1 Benutzerdefinierte Bilder zu Steuerelementen hinzufügen

Das Hinzufügen eines benutzerdefinierten Bilds mit dem Ribbon-Admin geschieht in den folgenden Schritten:

» Wählen Sie das Steuerelement aus, dem Sie ein Bild hinzufügen möchten.

» Betätigen Sie den Befehl **Benutzerdefiniertes Image hinzufügen** aus dem Kontextmenü des Attributs **Image**.

» Es erscheint der Dialog **Benutzerdefiniertes Image für Ribbon-Steuerelement auswählen**. Dieser ist beim ersten Aufruf leer.

» Klicken Sie auf das Plus-Zeichen neben dem Feld Bilder im Ribbon-Admin und wählen Sie mit dem **Datei öffnen**-Dialog das gewünschte Bild aus (vornehmlich 32 x 32, PNG mit Transparenz).

» Das Image erscheint nun im oberen Bereich. Sie müssen es nun zur Zielanwendung hinzufügen, damit das dortige Ribbon darauf zugreifen kann. Ziehen Sie das Image per Drag and Drop in den Bereich **Bilder in Ihrer Datenbank**.

» Wenn die Zieldatenbank noch keine Images enthält, fragt der Ribbon-Admin, ob dafür eine Tabelle angelegt werden soll. Nachdem Sie zugestimmt haben, legt der Ribbon-Admin die Tabelle an, speichert das Image darin und zeigt es im Bereich **Bilder in Ihrer Datenbank** an.

» Das neue Image wird gleichzeitig unter **Aktuell markiertes Image** angezeigt. Wenn Sie diese Auswahl beibehalten, trägt der Ribbon-Admin den Namen dieses Images für das Attribut **image** des aktuellen Steuerelements ein.

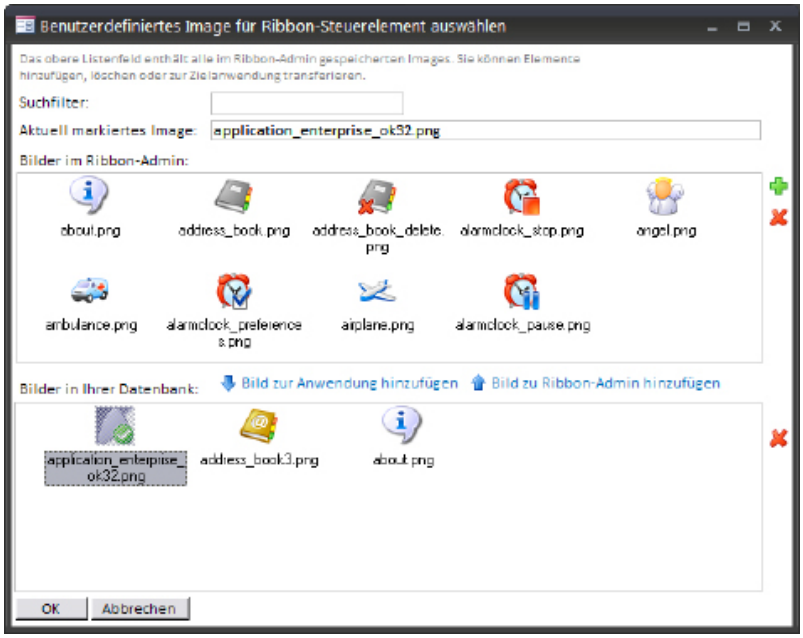


Abbildung 7.1: Der Dialog zum Verwalten und Auswählen von Images

Das Festlegen eines Images für das Attribut **image** reicht noch nicht aus, um dieses auch im Ribbon anzuzeigen. Sie benötigten weiterhin eine Callback-Funktion für mindestens eines der beiden Callback-Attribute **loadImage** im **customUI**-Element oder **getImage** im jeweiligen Steuerelement. Sollten Sie noch keines angelegt haben, weist der Ribbon-Admin Sie nach dem Füllen des image-Attributs darauf hin.

Informationen über das Anlegen dieser Callback-Attribute und der dazu gehörenden VBA-Funktionen finden Sie hier:

→ **»get...-Callbacks anlegen«, Seite 44**

→ **»loadImage-Callback anlegen«, Seite 46**

7.2 Eingebaute Bilder zu Steuerelementen hinzufügen

Das Hinzufügen eingebauter Bilder ist die einfachere Variante. Sie wählen dazu aus dem Kontextmenü des Attributs **image** den Befehl **Eingebautes Image hinzufügen** aus und öffnen damit den Dialog **Eingebautes Image für Ribbon-Steuerelement auswählen**.

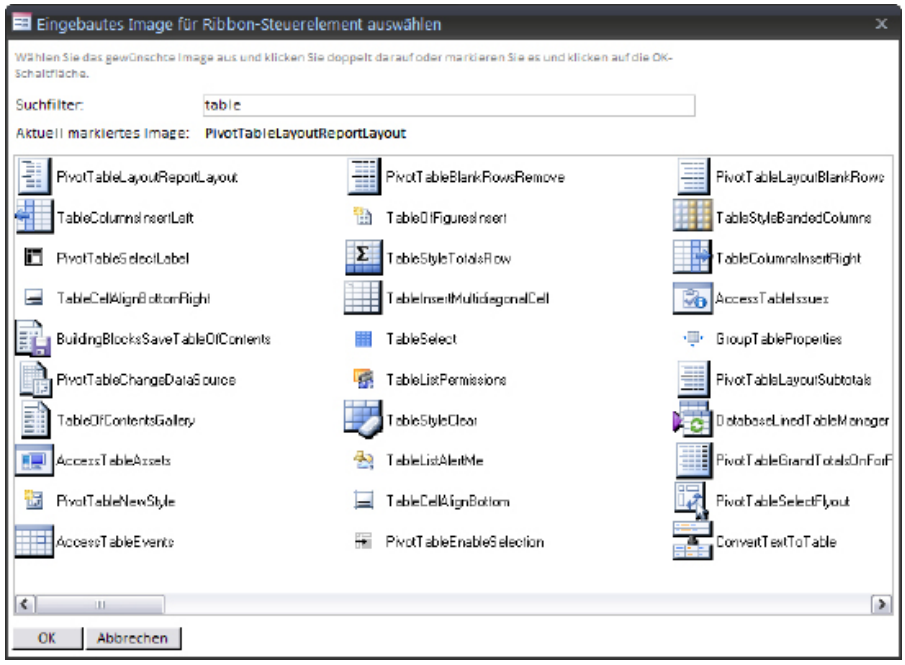


Abbildung 7.2: Dieser Dialog hilft beim Auswählen eingebauter Bilder.

Hier können Sie die eingebauten Bilder anhand ihrer **imageMso** filtern. Die **imageMso** entspricht der **idMso** der eingebauten Befehle. Wenn Sie im Suchfilter beispielsweise den Ausdruck **table** eingeben, zeigt der Dialog nur noch die Bilder an, deren **imageMso** den Text **table** enthält.

Nach der Auswahl des gewünschten Bilds klicken Sie auf **OK**, um die **imageMso** in das entsprechende Attribut des Steuerelements zu übernehmen.

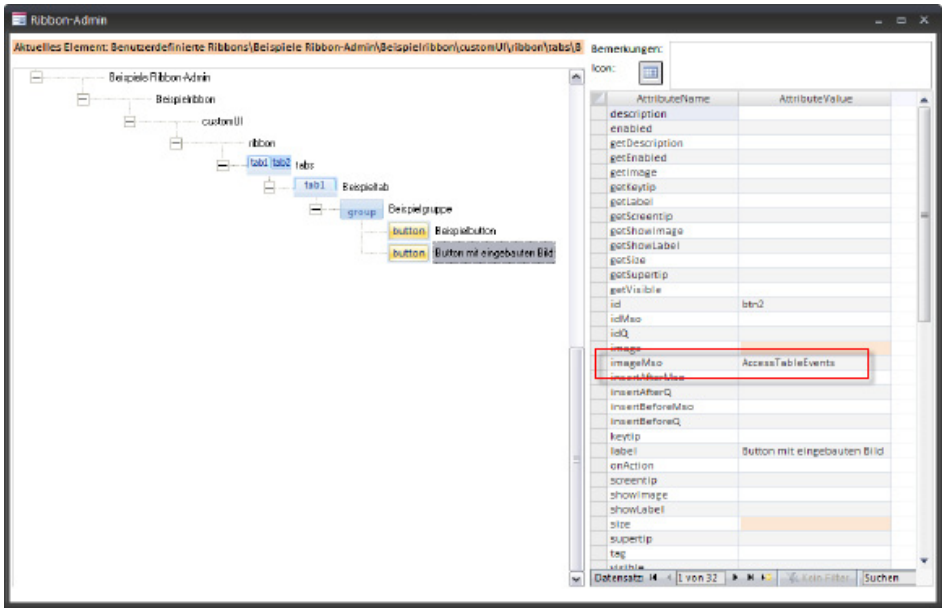


Abbildung 7.3: Die ausgewählte **idMso** wird in das entsprechende Attribut übernommen.

8 Weitere Funktionen des Ribbon-Admin

Dieses Kapitel liefert weitere Funktionen des Ribbon-Admin.

8.1 Das Ribbon anzeigen

Neben der Funktion eines selbst gebauten Ribbons ist natürlich sein Äußeres wichtig, und das möchten Sie vermutlich regelmäßig während des Zusammenstellens kontrollieren. Dazu brauchen Sie einfach nur den Kontextmenübefehl **Ribbon anzeigen** des Ribbon-Definitions-Elements zu betätigen. Sofern die Ribbon-Definition keine Fehler enthält, zeigt die aktuelle Anwendung die von Ihnen definierte Ribbon-Erweiterung an.

Wenn Sie nicht die Eigenschaft **startFromScratch** des **ribbon**-Elements auf den Wert **true** gesetzt haben, geschieht die Anzeige mit unter recht unspektakulär: Neue Ribbon-Tabs reihen sich dann, wenn Sie dies nicht anders vorgesehen haben, einfach rechts neben den vorhandenen Tabs ein.

Ein in Bearbeitung befindliches Ribbon zeigen Sie an, indem Sie aus dem Kontextmenü seines Ribbon-Definitions-Elements den Eintrag **Ribbon anzeigen** auswählen.

8.2 Ribbon in der Zielanwendung speichern

Nach dem Zusammenstellen eines Ribbons fehlt noch der letzte Schritt: Die Ribbon-Definition muss in der Zielanwendung gespeichert werden. Dazu gibt es zwei Möglichkeiten:

» Sie speichern alle für die Anwendung angelegten Ribbons in der Zielanwendung oder

» Sie speichern nur eine Ribbon-Definition in der Zielanwendung.

Im ersten Fall verwenden Sie den Eintrag **Alle Ribbons in Zielanwendung schreiben** des Anwendungs-Elements. Wollen Sie nur eine Ribbon-Definition speichern, verwenden Sie den Eintrag **Ribbon in Zielanwendung schreiben** des übergeordneten Ribbon-Definitions-Elements.

Wenn dies das erste Ribbon der Zielanwendung ist, muss der Ribbon-Admin dort zuvor noch eine spezielle Tabelle namens **USysRibbons** anlegen. Der entsprechenden Meldung sollten Sie daher zustimmen:

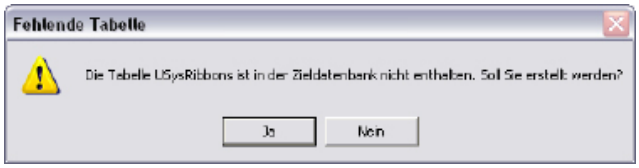


Abbildung 8.1: Wenn die Anwendung noch kein Ribbon enthält, muss der Ribbon-Admin zunächst eine Tabelle namens **USysRibbons** anlegen.

Die Tabelle taucht anschließend direkt im Navigationsbereich auf – vorausgesetzt, die Einstellungen von Access erlauben die Anzeige von Systemtabellen. Falls nicht, können Sie dies durch einen Klick mit der rechten Maustaste auf die Titelleiste des Navigationsbereichs, auswählen von **Navigationsoptionen...** und anschließendes Aktivieren der Option **Systemobjekte anzeigen** erreichen.

Die neue Tabelle **USysRibbons** sieht wie in folgender Abbildung aus und wird von nun an alle neuen Ribbon-Definitionen im XML-Format aufnehmen:

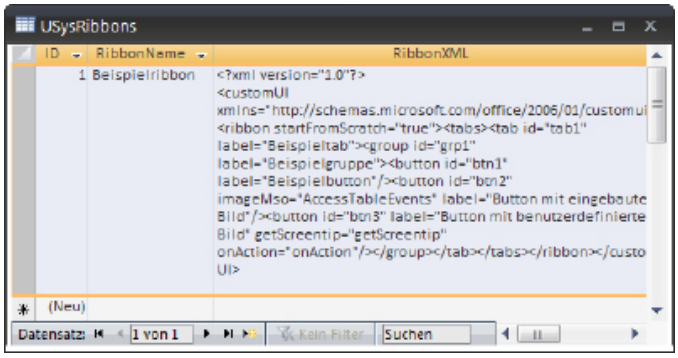


Abbildung 8.2: Die Tabelle **USysRibbons** speichert die Ribbon-Definitionen einer Datenbank.

Gegebenenfalls enthält die Tabelle **USysRibbons** der Zielanwendung bereits ein Ribbon mit dem Namen des zu exportierenden Ribbons. In diesem Fall fragt der Ribbon-Admin vor dem Speichern, ob das vorhandene Ribbon überschrieben werden soll.

Wenn Sie möchten, dass die Anwendung das entsprechende Ribbon anzeigt, müssen Sie dieses entweder als Anwendungsribbon festlegen oder einem Formular oder Bericht zuweisen.

→ **Ribbon als Anwendungsribbon festlegen**

→ **Ribbon zu Formular oder Bericht zuweisen**

8.2.1 Ribbon als Anwendungsribbon festlegen

Wenn die Anwendung ein Ribbon als Anwendungsribbon verwenden und dieses gleich beim Start anzeigen soll, müssen Sie es der Anwendung zunächst wie oben beschrieben zuweisen und die Anwendung einmal schließen und wieder öffnen.

Danach gehen Sie folgendermaßen vor:

» Wählen Sie den Eintrag **Ribbon als Anwendungsribbon festlegen** des Kontextmenüs des Ribbon-Definitions-Elements aus.

» Schließen Sie die Datenbank und öffnen Sie diese erneut – die Anwendung zeigt nun das angegebene Ribbon an.

Beim nächsten Start sieht die Anwendung beispielsweise wie in der folgenden Abbildung aus.

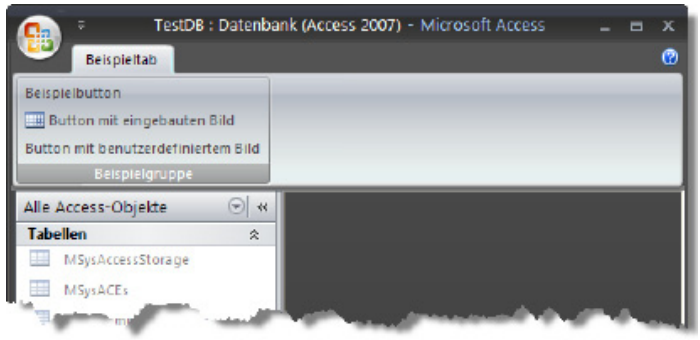


Abbildung 8.3: Anwendung mit einem direkt beim Start erscheinenden Ribbon

8.2.2 Ribbon zu Formular oder Bericht zuweisen [Achtung: Diese Funktion ist in Planung, eine Umsetzung scheitert möglicherweise aber an den technischen Gegebenheiten!]

Wenn der Benutzer Formulare oder Berichte öffnet, soll er unter Umständen zusätzliche oder andere Ribbon-Befehle präsentiert bekommen. Nachdem Sie eine entsprechende Ribbon-Definition zusammengestellt haben, können Sie diese mit dem Ribbon-Admin dem Zielformular oder -bericht zuweisen.

Dazu gehen Sie so vor:

» Wählen Sie den Eintrag **Ribbon zu Formularen und Berichten zuweisen** aus dem Kontextmenü des Ribbon-Definitions-Elements aus.

» Klicken Sie auf die Schaltfläche **Formulare/Berichte und zugewiesene Ribbons einlesen/aktualisieren**. Wenn die Datenbank viele Formulare und/oder Berichte enthält, können Sie das Laden durch Ausblenden einer der beiden Gruppen beschleunigen.

» Anschließend finden Sie im rechten Fenster alle in der Datenbank enthaltenen Elemente vor. Klicken Sie doppelt auf ein Element, um es von einer Seite zur anderen zu bewegen, oder verwenden Sie die Schaltflächen zwischen den beiden Listen, um einem Formular oder einem Bericht ein Ribbon hinzuzufügen.

» Verwenden Sie das Kombinationsfeld, um ein anderes als das zu Beginn angezeigte Ribbon anzuzeigen. Beachten Sie, dass Sie jedem Formular/Bericht nur ein Ribbon zuweisen können. Das rechte Kombinationsfeld zeigt immer alle Elemente an, die nicht dem aktuell ausgewählten Ribbon zugeordnet sind – unabhängig davon, ob Sie ihnen schon ein anderes Ribbon zugewiesen haben. Somit kann es passieren, dass Sie ein zugeordnetes Ribbon durch ein anderes ersetzen.

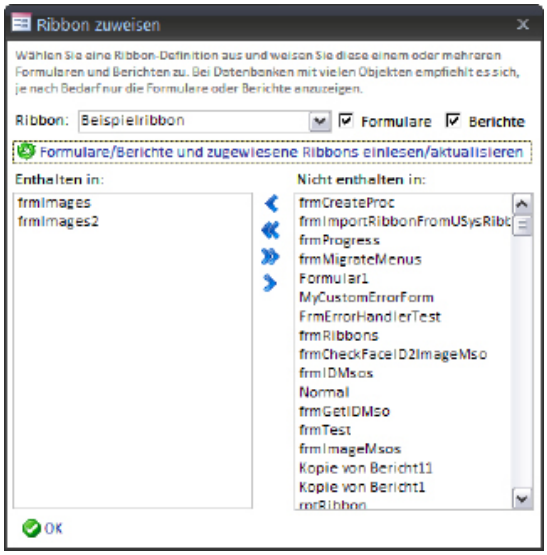


Abbildung 8.4: Mit diesem Dialog weisen Sie Formularen und Berichten Ribbons zu.

Durch das Zuweisen eines Ribbons über diesen Dialog schreibt der Ribbon-Admin den Namen des Ribbons in die Eigenschaft **Name der Multifunktionsleiste** des Formulars/Berichts.

8.3 Ribbon-XML anzeigen

Während der Ribbon-Admin alles tut, um Ihnen den Umgang mit XML zu ersparen, möchte man vielleicht doch einmal einen Blick auf den XML-Code des aktuellen Ribbons werfen.

Dazu wählen Sie aus dem Kontextmenü des Ribbon-Definitions-Elements den Eintrag **Ribbon-XML im Formular anzeigen** aus. Es erscheint ein Formular mit der Definition des Ribbons:

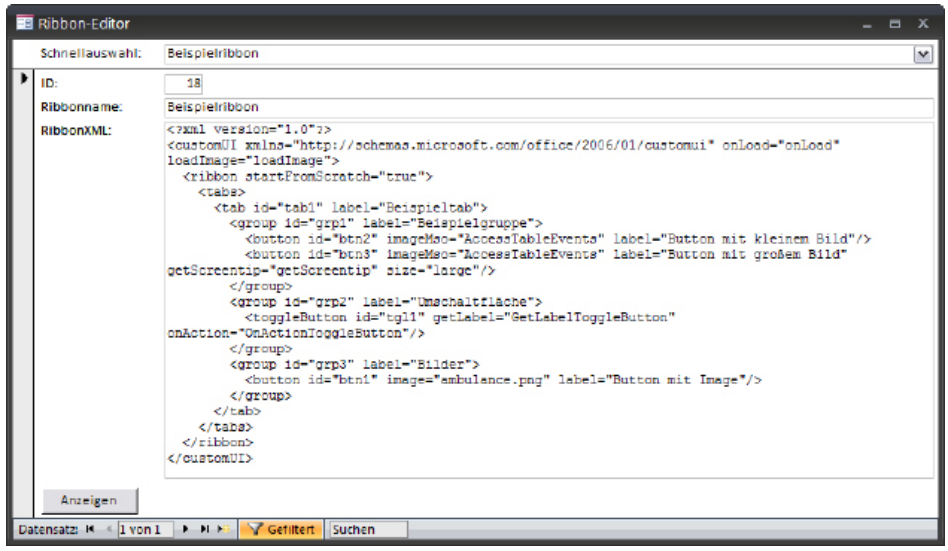


Abbildung 1.2: Die Ribbon-Definition im XML-Format

Änderungen an dieser Ansicht wirken sich nicht auf die im Ribbon-Admin gespeicherten Ribbons aus.

Ribbon-Kapitel aus: Das Access-Entwicklerbuch www.amvshop.de

12

Ribbon

Wenn Sie von Access 2007 auf Access 2010 gewechselt sind, haben Sie den größten Kulturschock bereits hinter sich. Seit Access 2007 heißt die neue Menüleiste der Microsoft Office-Anwendungen nämlich Ribbon. Die deutsche Bezeichnung für Ribbon (wörtlich übersetzt: »Band«) lautete in Office 2007 Multifunktionsleiste, in Office 2010 hat der Übersetzer sich für den Ausdruck »Menüband« entschieden. In diesem Buch wird daher ausschließlich von Ribbon die Rede sein – das ist erstens kürzer und zweitens wesentlich cooler. Das Ribbon ist Teil der runderneuerten Benutzeroberfläche und übernimmt die Rolle von Menü- und Symbolleisten gleichzeitig. Während es unter Office 2007 noch einen Office-Button gab, mit dem Sie das Office-Menü anzeigen konnten, bietet uns Office 2010 etwas, von dem die Fans von Musikgruppen träumen: den Backstage-Bereich. Leider laufen dort keine Promis und Groupies herum, sondern es handelt sich

dabei um eine Erweiterung des Ribbons. 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. Und wenn Sie professionelle Anwendungen entwickeln, freuen Sie sich möglicherweise darauf, den Backstage-Bereich etwa für die Unterbringung von Anwendungsoptionen zu verwenden.

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. Kommen Sie von Access 2007, brauchen Sie ebenfalls keinen allzu großen Respekt zu haben: Auch der Backstage-Bereich wird schlicht und einfach per XML definiert und später per VBA automatisiert.

Während die Entwicklergemeinde seit der Version von Access 2007 auf Unterstützung beim Bau von Ribbons durch ein Tool von Microsoft hofft, geschieht dies auch mit Access 2010 noch nicht. Außerdem lernen Sie das vom Autor dieses Buchs entwickelte Tool zum Programmieren des Ribbons kennen. Wir würden es gern als kostenlose Dreingabe zum Buch liefern, aber die Entwicklungszeit beträgt bis jetzt bereits mehrere Monate. Daher finden Sie im Buch-Download eine Testversion, mit der Sie Ribbons mit einer begrenzten Anzahl von Elementen einfach programmieren können – so viel, dass es zumindest für alle in diesem Kapitel beschriebenen Beispiele reicht. Wenn Sie auf der Webseite zum Buch unter *http://www.access-entwicklerbuch.de/2010/ribbon* den Registrierungsschlüssel eingeben, erhalten Sie außerdem einen Rabatt von 50% auf den Listenpreis.

BEISPIELDATENBANKEN

Den Download mit dem Beispielen finden Sie unter *www.acciu.de/aeb2010*. Die Datenbank zu diesem Kapitel heißt *Ribbons.accdb*.

12.1 Anpassen des Ribbons/CustomUI

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. Genau genommen brauchen

Sie sich noch nicht einmal überhaupt mit XML zu beschäftigen, wenn Sie die eingeschränkte Version des *Ribbon-Admin 2010* aus dem Buch-Download verwenden.

Begriffsklärung

Die Gesamtheit der Elemente, die Sie per XML-Dokument anpassen können, nennen wir im Folgenden *CustomUI*. Dazu gehören diese Bereiche:

» Backstage-Bereich: der Bereich, den Sie durch einen Klick auf den Registerreiter *Datei* des Ribbons aktivieren

» Schnellstart-Leiste: Schaltflächen ganz oben neben dem Access-Symbol

» Ribbon: Die eigentliche Leiste mit den Tabs und Menübefehlen

Eine eigene *CustomUI*-Anpassung legen Sie in einem XML-Dokument fest, das Sie auf verschiedene Arten anwenden können. Dieses XML-Dokument speichern Sie am einfachsten in einer speziellen Tabelle und vergeben dabei auch gleich einen Namen für diese Anpassung. Mit bestimmten Eigenschaften legen Sie dann fest, welche Ihrer Anpassungen direkt beim Start der Anwendung durchgeführt werden sollen. Außerdem können Sie dafür sorgen, dass beim Öffnen von Formularen und Berichten andere oder weitere Anpassungen durchgeführt werden sollen – zum Beispiel, um einen weiteren Registerreiter hinzuzufügen, der spezielle Steuerelemente zum Aufrufen der Funktionen von Formularen oder Berichten enthält.

Aufbau des Ribbons

Das Ribbon besteht aus einem oder mehreren Registerreitern, den sogenannten Tabs. Jedes *tab*-Element enthält eine oder mehrere Gruppen (*group*) mit Steuerelementen (*button*, *comboBox*, ...). Einige Steuerelemente können wiederum andere Steuerelemente enthalten. Die Schnellstartleiste besteht aus einfachen Schaltflächen zum Aufrufen eingebauter oder benutzerdefinierter Funktionen. Der Backstage-Bereich ist der Bereich, den Sie durch einen Klick auf den Registerreiter *Datei* des Ribbons öffnen. Ihm ist ein eigenes Kapitel gewidmet (siehe »Backstage« ab Seite 777).

12.2 Schnellstart

Die grundsätzlichen Schritte für die Anpassung des Ribbons, das beim Start der Anwendung erscheinen soll, sehen so aus:

» Erstellen einer Tabelle zum Speichern einer oder mehrerer Ribbon-Anpassungen beziehungsweise XML-Dokumente

» Definieren der Anpassung durch ein entsprechendes XML-Dokument

» Eintragen dieser Anpassung in einen neuen Datensatz der dafür vorgesehenen Tabelle

» Schließen und öffnen der aktuellen Access-Datei

» Auswählen des beim Start anzuzeigenden Ribbons in den Access-Optionen

» Erneutes Schließen und Öffnen der Access-Datei – die Ribbon-Anpassung ist da!

Sie können auch Ribbon-Anpassungen definieren, die beim Öffnen von Formularen oder Berichten erscheinen sollen. In diesem Fall gehen Sie fast genauso wie oben vor – mit dem Unterschied, dass Sie den Namen der Ribbon-Anpassung in der Eigenschaft *Name des Menübands* des Formulars oder Berichts auswählen.

12.2.1 Tabelle USysRibbons erstellen

Schauen wir uns an, wie das im Detail abläuft. Erstellen Sie zunächst eine Tabelle namens *USysRibbons* mit den drei Feldern *ID* (Primärschlüsselfeld), *RibbonName* (Textfeld), *RibbonXML* (Memofeld). Die Tabelle sieht im Entwurf wie in Abbildung 12.1 aus. Nach dem Speichern erscheint die Tabelle normalerweise nicht im Navigationsbereich. Das liegt daran, dass Access Tabellen mit Namen wie *MSys...* oder *USys...* als Systemtabellen einstuft und ausblendet. Sie können diese Tabellen einblenden, indem Sie mit der rechten Maustaste auf den Titel des Navigationsbereichs klicken und aus dem Kontextmenü den Eintrag *Navigationsoptionen* auswählen. Im nun erscheinenden Dialog aktivieren Sie die Option *Systemobjekte anzeigen*.

12.2.2 customUI-Definition erstellen

Erstellen Sie dann ein XML-Dokument mit der Beschreibung der Ribbon-Anpassung. Das folgende Beispiel fügt schlicht ein *tab*-Element mit einem *group*-Element und einer Schaltfläche (*button*) hinzu. Einige *id*- und *label*-Attribute sorgen dafür, dass die Elemente eindeutig benannt und mit einer Beschriftung versehen werden. Die umschließenden *customUI*-, *ribbon*- und *tabs*-Elemente sind für Anpassungen des Ribbons obligatorisch:

```
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui">
  <ribbon>
    <tabs>
      <tab id="tab1" label="Beispieltab">
        <group id="grp1" label="Beispielgruppe">
          <button id="btn1" label="Beispielbutton"/>
        </group>
      </tab>
    </tabs>
  </ribbon>
</customUI>
```

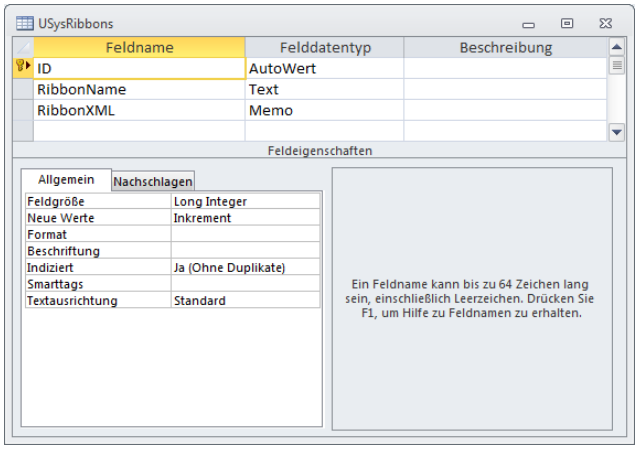


Abbildung 12.1: Diese Tabelle speichert später Ihre Ribbon-Anpassungen.

Legen Sie einen neuen Datensatz in der Tabelle *USysRibbons* an. Tragen Sie als Namen den Wert *Beispielribbon* ein und fügen Sie dieses XML-Dokument zum Feld *RibbonXML* hinzu. Die Tabelle sieht nun wie in Abbildung 12.2 aus.

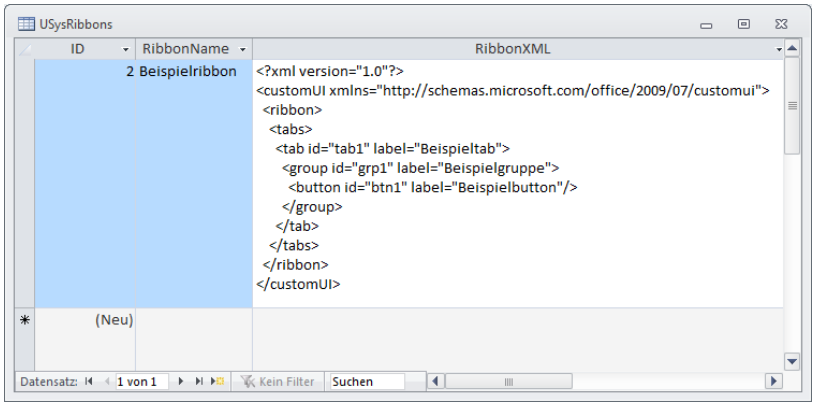


Abbildung 12.2: Die Tabelle *USysRibbons* mit einem Beispiel-CustomUI

EINFACHE RIBBON-ERSTELLUNG

Mit dem Tool *Ribbon-Admin 2010* erstellen Sie ganz einfach Ribbon- und Backstage-Anpassungen. Mehr dazu erfahren Sie am Ende des Kapitels. Im Download finden Sie eine Version dieses Kapitels, dass um die Beschreibung der Funktionen des Ribbon-Admin 2010 angereichert ist.

12.2.3 customUI-Anpassungen anwenden

Nun schließen Sie die aktuelle Access-Anwendung und öffnen diese erneut. Aktivieren Sie mit *Datei|Optionen* den Optionen-Dialog von Access und wählen Sie im Bereich *Aktuelle Datenbank* unter *Menüleisten- und Symbolleistenoptionen* den Wert *Neues Ribbon* für *Name des Menübands* aus (siehe Abbildung 12.3).

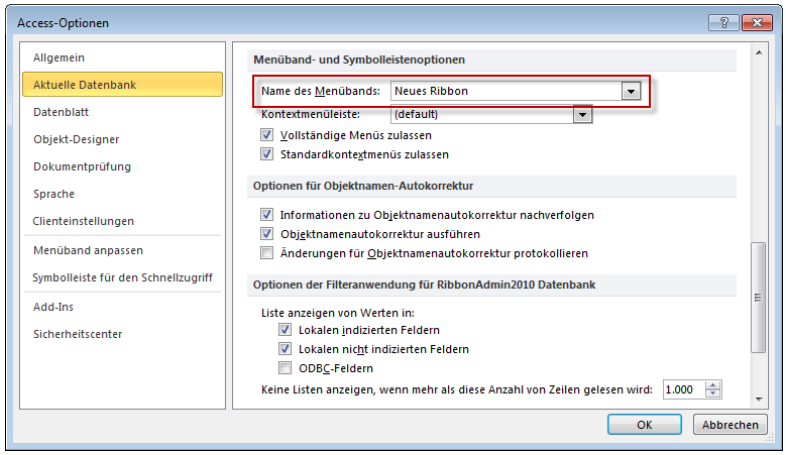


Abbildung 12.3: Auswählen eines *customUI* für die aktuelle Anwendung

Schließen Sie die Anwendung nochmals und öffnen Sie diese wieder. Nun ist das *tab*-Element mit der Beschriftung *Beispieltab* sichtbar. Ein Klick darauf zeigt auch die gewünschte Gruppe mit der Schaltfläche (siehe Abbildung 12.4).

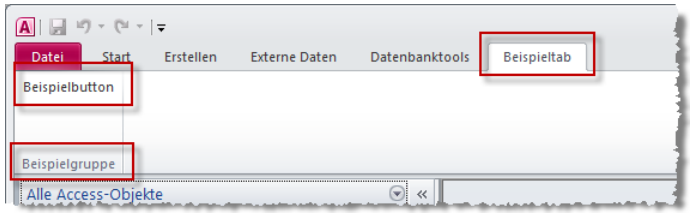


Abbildung 12.4: Access-Anwendung mit Beispielribbon

12.3 Manuelles Anpassen des customUI

Bevor Sie mit dem Erstellen von XML-Dokumenten beginnen, schauen wir uns noch schnell an, wie Sie die Benutzeroberfläche von Hand anpassen können. Auch hierzu liefert Access mittlerweile einige Möglichkeiten mehr als in der Version 2007.

Damit können Sie zumindest die eingebauten Elemente des *customUI* beeinflussen. Ausgangspunkt hierfür ist der Bereich *Menüband anpassen* des Dialogs *Access-Optionen*. Voraussetzung für die Anpassung ist außerdem eine geöffnete Access-Datenbank.

Damit Sie gleich ein praxisnahes Beispiel erhalten, schauen wir uns das *tab*-Element *Entwurf* der Entwurfsansicht von Formularen an. Hier hat Microsoft eine Änderung gegenüber Access 2007 vorgenommen, die viele Entwickler nicht wahrhaben wollen. Schauen Sie selbst: Abbildung 12.5 zeigt, dass in der Formularansicht nicht alle Steuerelement-Schaltflächen gleichzeitig sichtbar sind. Wenn Sie beispielsweise ein Kontrollkästchen, ein Listenfeld oder ein Unterformularsteuerelement zum Formular hinzufügen möchten, müssen Sie die Galerie erst wie in Abbildung 12.6 ausklappen.

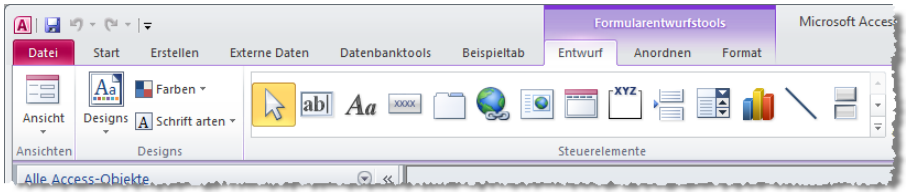


Abbildung 12.5: Steuerelemente im Ribbon vor ...

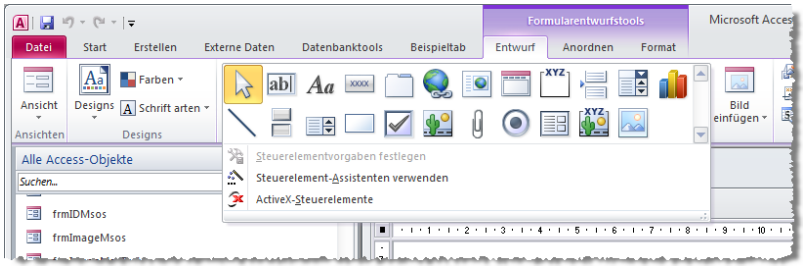


Abbildung 12.6: ... und nach dem Aufklappen der Galerie.

In Access 2007 waren die Steuerelemente über kleinere Schaltflächen erreichbar, die dafür aber zumindest alle gleichzeitig sichtbar waren. Was sich Microsoft auch immer bei dieser Änderung gedacht hat: Es ist ein guter Anlass, die Möglichkeiten der *customUI*-Anpassung mit Access-Bordmitteln zu demonstrieren.

Dazu öffnen Sie die Access-Optionen und wechseln zum Bereich *Menüband anpassen*. Es erwartet Sie dort ein Anblick wie in Abbildung 12.7. Das linke Listenfeld zeigt alle für Anpassungen verfügbaren Befehle an. Sie können diese mit dem darüber befindlichen Kombinationsfeld nach verschiedenen Kriterien filtern. Aber Achtung: Der Eintrag *Alle Befehle* etwa liefert längst nicht alle Befehle – so findet sich der Befehl *Textfeld (Formularsteuerelement)* beispielsweise nur unter *Nicht im Menüband enthaltene Befehle*.

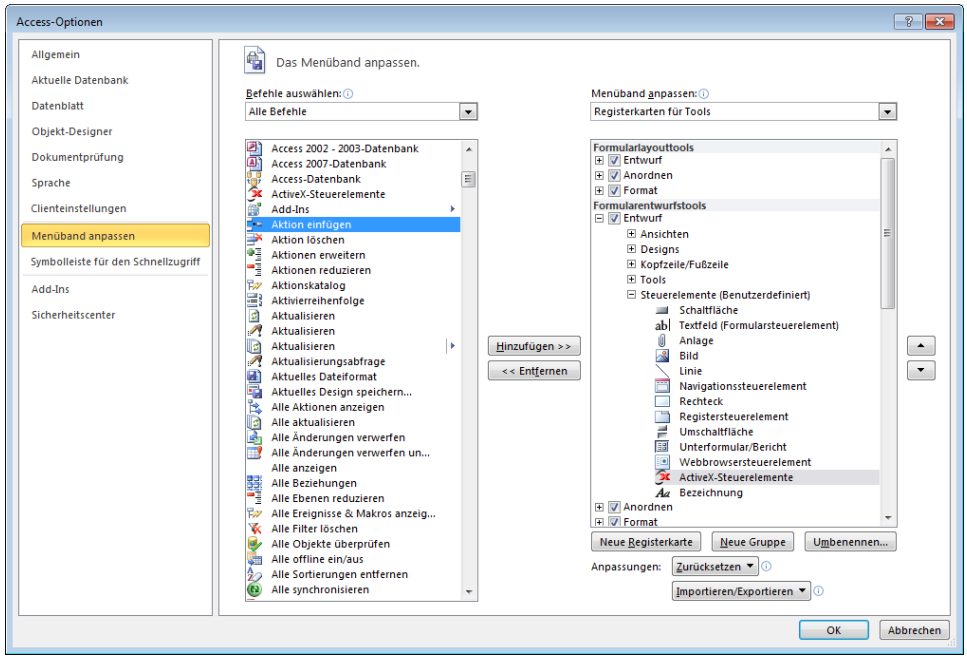


Abbildung 12.7: Anpassen des customUI mit Bordmitteln

Um die Schaltflächen für alle Steuerelemente zusammenzusuchen, müssen Sie sich also ein wenig durch die verschiedenen Listen wühlen.

Wo aber landen die Befehle? Das rechte Listenfeld zeigt die Struktur des Ribbons an, auch hier gibt es verschiedene Filterkriterien. Wählen Sie oben *Registerkarten für Tools* aus und klicken sich dann wie in Abbildung 12.8 zu den Steuerelementen durch. Mit einem Rechtsklick auf *Entfernen* entsorgen Sie die unpraktische Gruppe. Experimentieren Sie ruhig, mit der Schaltfläche *Zurücksetzen* bringen Sie das *customUI* wieder in den Anfangszustand (dies gilt nur für Änderungen, die über diesen Dialog herbeigeführt wurden).

Danach fügen Sie eine benutzerdefinierte Gruppe zum *tab*-Element *Entwurf* hinzu. Aktivieren Sie dazu den Eintrag *Entwurf* und klicken Sie auf *Neue Gruppe*. Benennen Sie die neue Gruppe gleich im Anschluss in *Steuerelemente* um. Nun beginnt die Sucherei: Fügen Sie alle benötigten Steuerelement-Schaltflächen aus dem linken Listenfeld zum rechten Listenfeld hinzu (siehe Abbildung 12.9).

Mit den *Nach oben*- und *Nach unten*-Schaltflächen können Sie die Reihenfolge der Steuerelemente nach Ihren Wünschen ändern. Schließen Sie den Dialog und öffnen Sie ein Formular in der Entwurfsansicht. Und das Ergebnis ist ... fantastisch! Sie erreichen nun alle Steuerelemente mit einem einzigen Mausklick, wie Abbildung 12.10 zeigt.

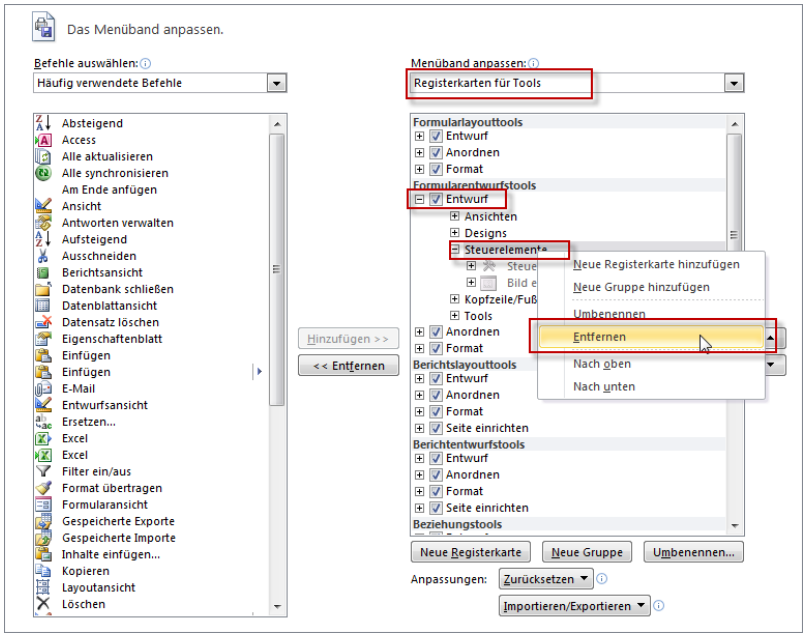


Abbildung 12.8: Löschen der Steuerelemente im Formularentwurf

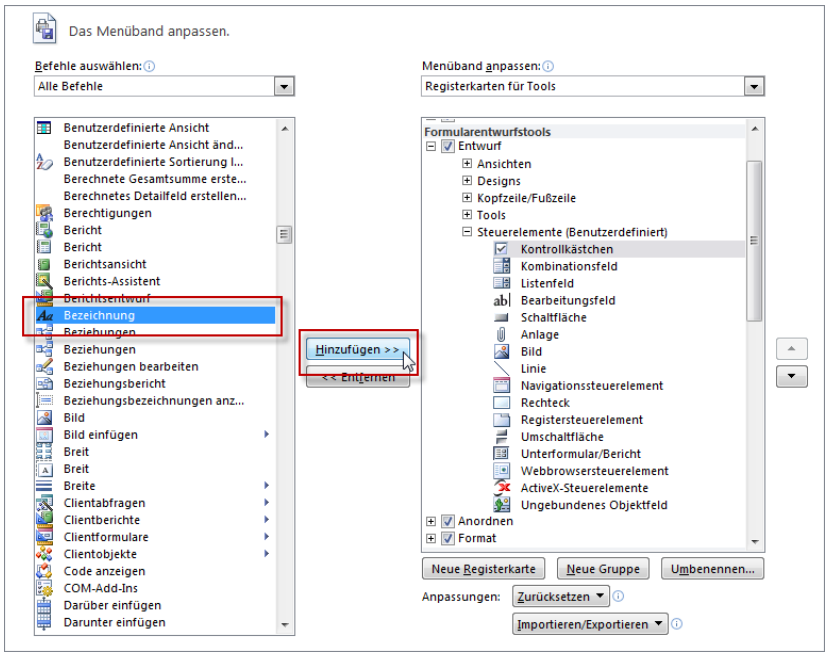


Abbildung 12.9: Hinzufügen von Steuerelementen zu einer benutzerdefinierten Gruppe

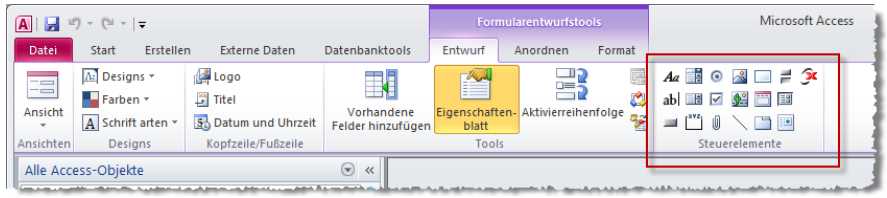


Abbildung 12.10: Die Steuerelemente mit kleinen Icons und auf einen Blick sichtbar

Übrigens müssen Sie die Datenbank nicht erneut öffnen, damit die Änderungen sichtbar werden (außer bei Anpassungen der Schnellzugriffsleiste).

Diese Änderung ist nun immer verfügbar, wenn Sie mit Access arbeiten. Fehlt nur noch eine Möglichkeit, diese Anpassung auf einen anderen Rechner übertragen zu können. Kein Problem: Auch das gelingt mit Access 2010. Klicken Sie unten auf die Schaltfläche *Importieren/Exportieren* und wählen Sie zunächst den Eintrag *Exportieren* aus. Geben Sie einen Dateinamen an und speichern Sie die Änderungen. Anschließend können Sie diese auf anderen Rechnern wieder importieren. Die exportierte Datei sieht wie folgt aus:

```
<mso:cmd app="Access" dt="0" />
  <mso:customUI xmlns:mso="http://schemas.microsoft.com/office/2009/07/customui">
    <mso:ribbon>
      <mso:qat/>
      <mso:contextualTabs>
        <mso:tabSet idMso="TabSetFormTools">
          <mso:tab idQ="mso:TabFormToolsDesign">
            <mso:group id="mso_c1.1A400E2" label="Steuerelemente" autoScale="true">
              <mso:control idQ="mso:FormControlLabel" visible="true"/>
              <mso:control idQ="mso:FormControlEditBox" visible="true"/>
              <mso:control idQ="mso:FormControlButton" visible="true"/>
              <mso:control idQ="mso:FormControlComboBox" visible="true"/>
              ... weitere Steuerelemente
            </mso:group>
            <mso:group idQ="mso:GroupControlsAccess" visible="false"/>
          </mso:tab>
        </mso:tabSet>
      </mso:contextualTabs>
    </mso:ribbon>
  </mso:customUI>
```

Der Aufbau sieht genauso aus wie der einer benutzerdefinierten Anpassung der Benutzeroberfläche – es gibt lediglich ein paar kleine Unterschiede wie etwa das *mso:* vor jedem Elementnamen.

12.4 Symbolleiste für den Schnellzugriff

Auf ähnliche Art passen Sie die Schnellzugriffsleiste an, also die Leiste mit den kleinen Symbolen oben links neben dem Access-Symbol. Die Schnellzugriffsleiste 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:

» Über das mit der nebenan liegenden Schaltfläche zu öffnende Menü: Damit können Sie die vorhandenen Einträge abwählen und andere hinzufügen (siehe Abbildung 12.11).

» Beliebige Elemente aus den vorhandenen Ribbons fügen Sie der Schnellzugriffsleiste über den Eintrag *Zu Symbolleiste für den Schnellzugriff hinzufügen* des Kontextmenüs des jeweiligen Elements hinzu (siehe Abbildung 12.12).

» Im Bereich *Symbolleiste für den Schnellzugriff* 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 12.13) öffnen Sie entweder über den herkömmlichen Weg (*Datei|Access-Optionen*), über den Eintrag *Weitere Befehle...* des Schnellzugriffsleisten-Menüs oder den Eintrag *Symbolleiste für den Schnellzugriff anpassen...* des Kontextmenüs eines Ribbon-Steuerelements.

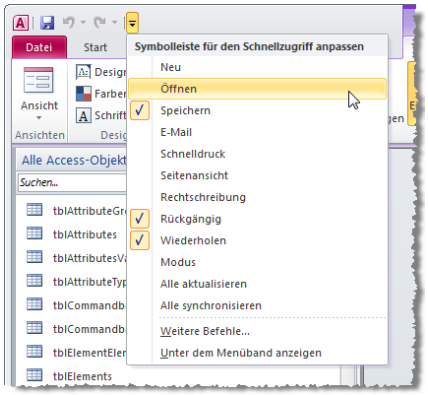


Abbildung 12.11: Die Symbolleiste für den Schnellzugriff und das Menü für ihre Anpassung

Schnellzugriffsleiste positionieren

Die standardmäßig neben dem Access-Symbol befindliche Schnellzugriffsleiste können Sie auch unterhalb des Ribbons platzieren. Dazu wählen Sie etwa den Eintrag *Unter dem*

Menüband anzeigen des Menüs der Schnellzugriffsleiste. Mit dieser Einstellung vergrößert sich der Ribbon-Bereich allerdings um einen zusätzlichen Balken.

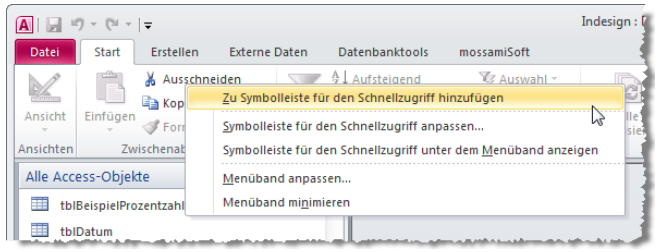


Abbildung 12.12: Per Kontextmenü können Sie beliebige Befehle zur Schnellzugriffsleiste hinzufügen

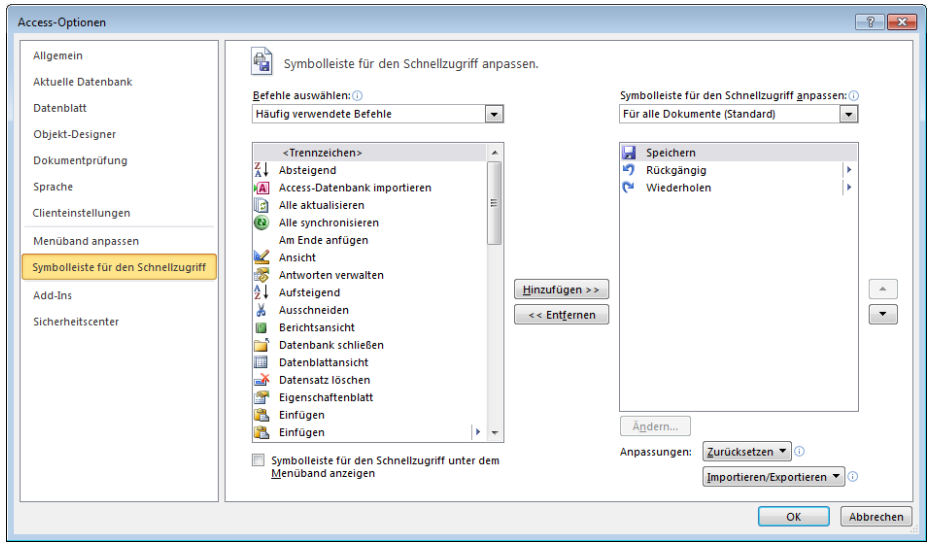


Abbildung 12.13: Dialog zum Anpassen der Schnellzugriffsleiste

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 (die Konfiguration ist in der Registry gespeichert und kann per VBA über die *SetOption*-Methode eingestellt werden).

Um dies zu gewährleisten, kommen Sie um die Definition eines eigenen Ribbons nicht herum.

12.5 Eigene Ribbon-Anpassung erstellen

Bevor Sie loslegen, sollten Sie im Dialog *Verweise* des VBA-Editors (*Extras/Verweise*) einen Verweis auf die Bibliothek *Microsoft Office 14.0 Object Library* anlegen. Anderenfalls kann es bei den folgenden Beispielen zu Fehlermeldungen kommen.

Außerdem beugen wir schon jetzt einem weit verbreiteten Problem vor: Das mühselig programmierte Ribbon erscheint nicht, und eine Fehlermeldung gibt es auch nicht. Dies ist der Fall, wenn Sie die Option *Fehler des Benutzeroberflächen-Add-Ins anzeigen* in den Access-Optionen nicht aktiviert haben.

Prüfen Sie diese Einstellung, die standardmäßig nicht eingeschaltet ist, und aktivieren Sie diese gegebenenfalls (siehe Abbildung 12.14).

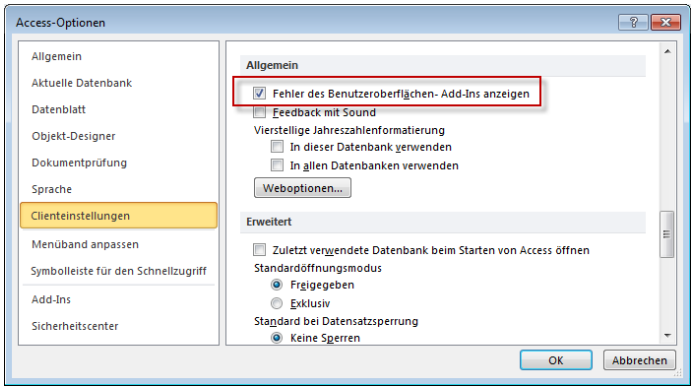


Abbildung 12.14: Diese Einstellung entscheidet, ob Fehler in der *customUI*-Definition angezeigt werden oder stillschweigend übergangen werden.

Sollten Sie diese Option aktiviert haben, liefert Access entsprechende Fehlermeldungen, wenn die XML-Datei nicht der in der Schemadatei vorgegebenen Form entspricht (siehe Abbildung 12.15).

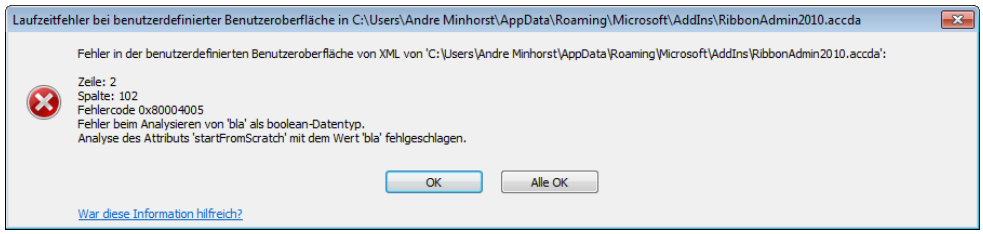


Abbildung 12.15: Fehlermeldung bei ungültigem XML-Dokument

12.5.1 Elemente einer customUI-Anpassung

Im Beispiel von oben haben Sie bereits gesehen, dass die *customUI*-Anpassung durch hierarchisch angeordnete Elemente wie *ribbon*, *tabs*, *tab*, *group* oder *button* realisiert wird. Jedes dieser Elemente besitzt wiederum Attribute, die sich entweder auf das Aussehen oder auf das Verhalten auswirken.

Oben haben Sie bereits das Attribut *label* kennen gelernt: Es legt fest, welche Bezeichnung zum jeweiligen Element angezeigt wird. Es gibt viele weitere Attribute für das Aussehen, mit denen Sie die Größe, das Icon und weitere Einstellungen vornehmen.

Außerdem besitzen die meisten *customUI*-Elemente Attribute, die den Ereigniseigenschaften von Formularen und den enthaltenen Steuerelementen entsprechen. Die Eigenschaft *onAction* eines *button*-Elements etwa erwartet als Wert die Angabe des Namens der VBA-Funktion, die beim Auslösen des *onAction*-Ereignisses aufgerufen werden soll.

ÜBERSICHTSTABELLEN

Übersichten über alle Ribbon-Elemente finden Sie im *.pdf*-Format im Download zu diesem Buch unter *www.access-entwicklerbuch.de/2010/download*.

12.5.2 Die Datei customUI14.xsd

Damit Access mit dem von Ihnen zusammengestellten XML-Dokument auch etwas anfangen kann, prüft es dieses zunächst gegen eine sogenannte Schemadatei namens *customUI14.xsd*. Diese können Sie im Internet herunterladen, aber Sie finden sie auch im Download zu diesem Kapitel. Access braucht sie ohnehin nicht, da das Schema zusätzlich fest in Access verdrahtet ist.

Sie könnten es aber gebrauchen, wenn Sie die XML-Dokumente mit einer Anwendung wie Visual Studio oder XML Notepad 2007 bearbeiten möchten. Wenn Sie das Schema

dann beispielsweise unter einem Pfad wie *C:\Program Files (x86)\Microsoft Visual Studio 10.0\xml\Schemas\1033* ablegen, kann Visual Studio das Schema beim Erstellen eines XML-Dokuments, das auf dieses Schema verweist, für die Bereitstellung der *IntelliSense*-Funktion verwenden.

Die Schemadatei legt in unserem Fall beispielsweise fest, dass alle weiteren Elemente in einem *customUI*-Element enthalten sein müssen.

Der öffnende Teil des Elements enthält einen Hinweis auf die Schema-Datei, die zur Validierung herangezogen werden soll:

```
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui">
    ...
</customUI>
```

Sie können die im *customUI*-Element angegebene Internetadresse schlicht und einfach als Zeichenkette betrachten, die mit dem in der Schema-Datei enthaltenen Ausdruck verglichen wird.

Die *NameSpace*-Bezeichnung hat sich in der finalen Version gegenüber der Beta-Version geändert – nur für den Fall, dass Sie bereits mit der Beta gearbeitet haben und sich wundern, dass die *customUI*-Anpassung nicht mehr funktioniert.

Der *NameSpace* des *customUI* von Access 2007 funktioniert unter Access 2010 immer noch. Andersherum ist dies allerdings nicht der Fall.

customUI

Unterhalb des *customUI*-Elements können Sie eines oder mehrere der folgenden vier Elemente platzieren – je nachdem, welche Bereiche Sie anpassen möchten:

» *backstage*: Einstellen des Backstage-Bereichs

» *ribbon*: Anpassen der eigentlichen Menüleiste, also des Ribbons

» *commands*: Anpassen der durch die Steuerelemente des *customUI* ausgelösten Befehle. Sie können hiermit beispielsweise festlegen, dass die *Speichern*-Schaltfläche eine benutzerdefinierte Funktion auslöst.

» *contextMenus*: Anpassen eingebauter Kontextmenüs etwa durch Hinzufügen eingebauter oder benutzerdefinierter Steuerelemente

backstage

Das *backstage*-Element enthält einen eigenen Bereich der Benutzeroberfläche, nämlich den Backstage-Bereich. Das Buch widmet diesem Bereich ein eigenes Kapitel unter »Backstage« ab Seite 777.

ribbon

Unterhalb des *ribbon*-Elements können Sie drei weitere Bereiche anlegen, um diese anzupassen. Es handelt sich um die folgenden:

» *qat* (Schnellzugriffsleiste): Kleine Leiste rechts neben dem Access-Symbol 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*: Übergeordnetes Element der einzelnen *tab*-Elemente. Diese machen das »eigentliche« Ribbon aus, können über die »Registerreiter« ausgewählt werden und enthalten in Gruppen aufgeteilte Steuerelemente

» *contexualTabs*: Übergeordnetes Element solcher *tab*-Elemente, die in Zusammenhang mit bestimmten Objekten eingeblendet werden – etwa das Entwurf-Tab beim Anzeigen eines Formulars in der Entwurfsansicht. Der Vorteil gegenüber herkömmlichen Tabs ist, dass *contextualTabs* farblich hervorgehoben werden und beim Öffnen des Formulars oder Berichts aktiviert werden.

Gegenüber Access 2007 fällt unter Access 2010 das Element *officeMenu* weg. Es wird komplett ersetzt durch das *backstage*-Element, das jedoch in der XML-Struktur eine Ebene höher angesiedelt ist.

commands

Das dritte Element, das Sie unterhalb von *customUI* anlegen können, heißt *commands*. Dieser Bereich enthält ein oder mehrere *command*-Objekte.

Die *command*-Objekte dienen nicht dazu, neue Objekt hinzuzufügen, sondern Sie können damit das *enabled*-Attribut eingebauter Steuerelemente einstellen oder sogar neue Funktionen für eingebaute Elemente definieren. Dazu suchen Sie zunächst die *idMso* des Steuerelements des Ribbons oder Backstage-Bereichs aus

contextMenus

Dieses Element ist in der Version 2010 noch nicht unter Access verfügbar, sondern nur etwa unter Word oder Excel. Dort können Sie auch Kontextmenüs per CustomUI-Anpassung verändern.

12.6 Struktur und Steuerelemente des Ribbons

Nachfolgend lernen Sie die grundlegende Struktur des Ribbons kennen, danach kümmern wir uns um die einzelnen Steuerelemente.

12.6.1 Das ribbon-Element

Dieses Element ist das Hauptelement einer Ribbon-Anpassung. Neben dem Aufnehmen der untergeordneten *tab*-Elemente hat es eine weitere wichtige Aufgabe: Es stellt mit der Eigenschaft *startFromScratch* eine Möglichkeit bereit, die eingebauten Ribbon-Elemente auszublenden. Dazu stellen Sie diese Eigenschaft auf den Wert *true* ein:

```
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui">
  <ribbon startFromScratch="true"/>
</customUI>
```

Diese *customUI*-Definition würde übrigens schlicht alle Ribbons ausblenden. Backstage, Schnellstartleiste und Kontextmenüs bleiben erhalten.

12.6.2 Das tabs-Element

Das *tabs*-Element hat keine Attribute. Es dient lediglich dazu, die enthaltenen *tab*-Elemente zusammenzufassen.

12.6.3 Das tab-Element

Wenn Sie Registerkartenreiter zum Ribbon hinzufügen möchten, brauchen Sie jeweils ein *tab*-Element. Die *tab*-Elemente landen innerhalb eines öffnenden und eines schließenden *tabs*-Elements. Sie müssen für jedes *tab*-Element die Attribute *id* und *label* festlegen. Die folgenden beiden *tab*-Elemente sorgen für ein Ribbon wie in Abbildung 12.16.

```
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui">
  <ribbon>
    <tabs>
      <tab id="tab1" label="Erstes Beispieltab"/>
      <tab id="tab2" label="Zweites Beispieltab"/>
    </tabs>
  </ribbon>
</customUI>
```

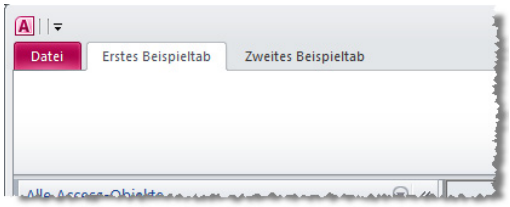


Abbildung 12.16: Zwei *tab*-Elemente

12.6.4 Das group-Element

Innerhalb der *tab*-Elemente nehmen Sie mit *group*-Elementen weitere Unterteilungen vor. Genau genommen brauchen Sie mindestens ein *group*-Element, um weitere Steuerelemente auf dem Ribbon platzieren zu können. Das folgende Beispiel fügt zwei Gruppen zum ersten Tab hinzu (siehe Abbildung 12.17):

```
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui">
  <ribbon>
    <tabs>
      <tab id="tab1" label="Erstes Beispieltab">
        <group id="grp1" label="Erste Gruppe"/>
        <group id="grp2" label="Zweite Gruppe"/>
      </tab>
      <tab id="tab2" label="Zweites Beispieltab"/>
    </tabs>
  </ribbon>
</customUI>
```

Abbildung 12.17: Ein *tab*-Element mit zwei *group*-Elementen

Das *group*-Element besitzt in Access 2010 gegenüber Access 2007 ein wichtiges neues Attribut: Mit *autoScale* legen Sie fest, ob die enthaltenen Elemente an den verfügbaren Platz angepasst werden.

Außerdem können Sie nun mit dem Attribut *centerVertically* festlegen, dass die in einer Spalte der Gruppe enthaltenen Steuerelemente vertikal zentriert werden. Dies sieht dann beispielsweise wie in Abbildung 12.18 aus.

Abbildung 12.18: Vertikal zentrierte Steuerelemente in einer Gruppe

Flexible Gruppen

Unter Access 2007 wurden einzelne Elemente von Gruppen bereits schrittweise verkleinert, wenn der Platz im Ribbon eng wurde. Dies ist nun in Access 2010 auch in benutzerdefinierten Gruppen möglich. Dazu arbeiten Sie das Attribut *autoScale* mit dem Wert *True* in das *group*-Element ein. Die folgenden Abbildungen zeigen, wie sich die Gruppe mit ihren Elementen bei schwindendem Platzbedarf verändert. Schließlich offenbart sich auch, wofür das bereits seit Access 2007 vorhandene *image*-Attribut des *group*-Elements gut ist: Es wird angezeigt, wenn der Platz nur noch zur Anzeige einer einzigen Schaltfläche zum Aufklappen des Menüs ausreicht.

```
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui"
loadImage="loadImage">
  <ribbon>
    <tabs>
      <tab idMso="TabHomeAccess" label="Start [mit flexibler Gruppe]">
        <group id="grp1" image="beer_glass" insertAfterMso="GroupSortAndFilter"
            label="Flexible Gruppe" autoScale="true">
          <button id="btn1" image="beer_bottle" label="Schaltfläche 1" size="large"/>
          <button id="btn2" image="beer_glass" label="Schaltfläche 2" size="large"/>
          <button id="btn3" image="cocktail" label="Schaltfläche 3" size="large"/>
        </group>
      </tab>
    </tabs>
  </ribbon>
</customUI>
```

Listing 12.1: Beispiel für eine Gruppe mit flexibler Größe

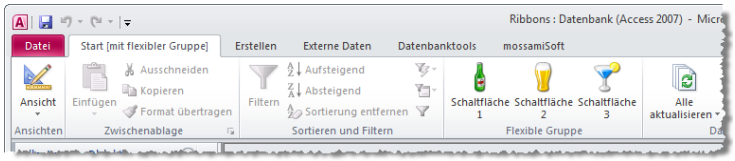


Abbildung 12.19: Schaltflächen mit großen Bildern bei voller Ribbon-Breite, ...

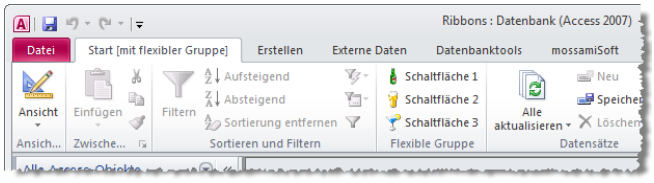


Abbildung 12.20: ... bei etwas knapper werdendem Platz, ...

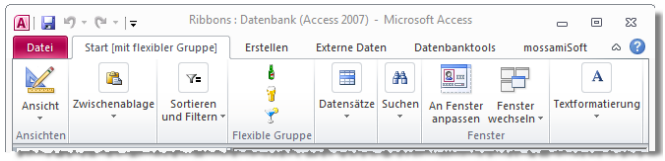


Abbildung 12.21: ... nur noch als Icon und ...

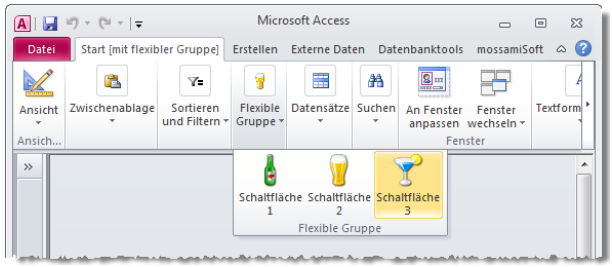


Abbildung 12.22: ... als Menü, das beim Anklicken wieder die komplette Gruppe anzeigt

12.6.5 Das button-Element

Das meistverwendete Element dürfte das *button*-Element sein. Es kann mit oder ohne Bild (*image*, *imageMso*) und klein oder groß (*size*) angezeigt werden.

button-Elemente werden normalerweise innerhalb eines *group*-Elements angelegt (es gibt noch andere Möglichkeiten als Bestandteil weiterer Steuerelemente – dazu später mehr). Im einfachsten Fall fügen Sie einfach nur ein *button*-Element zu einer Gruppe hinzu und legen seine Attribute *id* und *label* fest:

```
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui">
  <ribbon>
    <tabs>
      <tab id="tab" label="Schaltflächenbeispiele">
        <group id="grp1" label="Einfache Schaltflächen">
          <button id="cmd1" label="Einfache Schaltfläche"/>
        </group>
      </tab>
    </tabs>
  </ribbon>
</customUI>
```

Dieses Beispiel zeigt, wie Sie das *button*-Element innerhalb der übergeordneten Elemente einfügen. Die folgenden Beispiele verzichten auf diese Elemente und zeigen nur noch die *button*-Elemente mit ihren Attributen.

Kleine und große Schaltflächen

Schaltflächen gibt es in der kleinen (normalen) und in einer großen Variante. Drei kleine Schaltflächen nehmen übereinander den Platz einer großen Schaltfläche ein. Damit dies in Abbildung 12.23 deutlich wird, haben wir der Schaltfläche mit dem Attribut *size=large* über *imageMso* ein eingebautes Icon hinzugefügt:

```
<button id="btn1" label="Schaltfläche 1"/>
<button id="btn2" label="Schaltfläche 2"/>
<button id="btn3" label="Schaltfläche 3"/>
<button id="btnGrosseSchaltflaeche" imageMso="CreateForm" label="Große Schaltfläche"
size="large"/>
```

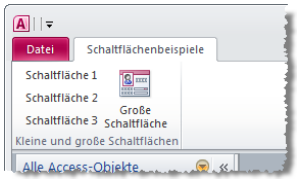


Abbildung 12.23: Kleine und große Schaltflächen im Ribbon

An die *imageMso* für das Bild eines eingebauten Steuerelements gelangen Sie auf die gleiche Weise wie an die *idMso* (siehe »Eingebaute Elemente in benutzerdefinierten Ribbons« ab Seite 108).

12.6.6 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 CustomUI-XML-Dokument beispielsweise für die große Schaltfläche *btnGrosseSchaltflaeche* wie folgt:

```
<button id="btn4" imageMso="CreateForm" label="Große Schaltfläche"
onAction="OnActionCommand" size="large"/>
```

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

```
Sub OnActionCommand(control As IRibbonControl)
    MsgBox "Sie haben die Schaltfläche '" & control.id & "' angeklickt."
End Sub
```

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 – und gleichzeitig noch den Namen des Steuerelements anzuzeigen. Diesen liefert die Eigenschaft *id* des Parameters *control*. 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.

Eine oder mehrere OnAction-Prozeduren

Da Access beim Aufruf einer Callback-Funktion meist einen Verweis auf das aufrufende Steuerelement in Form eines Parameters bereitstellt, brauchen Sie nicht für jedes Steuerelement eine eigene Callback-Funktion anzulegen.

Sie können auch nur eine einzige Callback-Funktion etwa mit dem Namen *OnAction* anlegen und dann innerhalb einer *Select Case*-Anweisung auf das entsprechende Steuerelement reagieren:

```
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 12.3: Auswerten des aufrufenden Steuerelements in einer Select Case-Anweisung

12.7 customUI und VBA

Neben dem *button*-Element gibt es noch viele weitere Steuerelemente im *customUI*. Diese werden später detailliert beschrieben.

Die folgenden Abschnitte zeigen zunächst, wie Sie auf Ereignisse im *customUI* reagieren können, wie Sie die Steuerelemente mit Bildern ausstatten und wie Sie die Eigenschaften von Steuerelementen zur Laufzeit dynamisch anpassen können.

12.7.1 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.

12.7.2 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 im PDF-Format im Download zu diesem Buch (*Ribbon-Referenz.pdf*). 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 zumindst 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(ctl As IRibbonControl, ByRef label)
    label = "Beispielbeschriftung"
End Sub
```

Listing 12.4: Diese Callback-Routine liefert den String *Beispielbeschriftung* an das aufrufende Ribbon zurück

Vielleicht wundert es Sie, dass hier von einer Funktion die Rede ist, 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.

12.7.3 Ereigniseigenschaften

Ribbons bieten Ereigniseigenschaften, 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 Ereigniseigenschaft, 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 Ereigniseigenschaften 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 in den Übersichtstabellen im Download zu diesem Buch.

12.7.4 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 Ereigniseigenschaft 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 12.5: 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 der Tabelle *Ereigniseigenschaften der customUI-Elemente* aus dem Download entnehmen können, besitzen Callback-Funktionen für gleichnamige Ereigniseigenschaften 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 *OnToggleButtonAction* heißen. Die Klasse (genauer: das Interface) *IRibbonControl* ist übrigens in der *Microsoft Office 14.0 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 12.5 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, kön-

nen 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.

12.7.5 Ribbon-Tab per VBA einstellen

In der Access-Community wurde oft nach einer Möglichkeit gefragt, ein bestimmtes *tab*-Element zu aktivieren. Eine Möglichkeit ist das Öffnen eines Formulars mit einem kontextsensitiven *tab*-Element – mehr dazu später unter »Kontextsensitive Ribbons« ab Seite 116.

Microsoft hat die Hilferufe erhört und mit den Methoden *ActivateTab, ActiveTabMso* und *ActiveTabQ* Mittel präsentiert, mit denen Sie ein bestimmtes Tab im aktuellen Ribbon aktivieren können. Die folgende XML-Definition fügt dem Ribbon drei Tabs hinzu:

```
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui" onLoad="OnLoad_
Tabwechsel">
  <ribbon>
    <tabs>
      <tab id="tab1" label="Tab 1"/>
      <tab id="tab2" label="Tab 2"/>
      <tab id="tab3" label="Tab 3"/>
    </tabs>
  </ribbon>
</customUI>
```

Das *customUI*-Element löst beim Laden die Funktion *OnLoad_Tabwechsel* aus, welche die Objektvariable *objRibbon_Tabwechsel* mit einem Verweis auf das *IRibbonUI*-Objekt füllt:

```
Public objRibbon_Tabwechsel As IRibbonUI

Sub onLoad_Tabwechsel(ribbon As IRibbonUI)
    Set objRibbon_Tabwechsel = ribbon
End Sub
```

Das Formular *frmTabwechsel* der Beispieldatenbank enthält einige Schaltflächen, welche die eingebauten und die benutzerdefinierten *tab*-Elemente aktivieren sollen (siehe Abbildung 12.24).

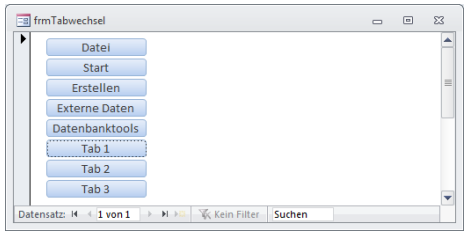


Abbildung 12.24: Formular zum Auswählen des Ribbon-Tabs

Diese Schaltflächen lösen Ereignisprozeduren aus. Im Fall eines eingebauten *tab*-Elements verwenden Sie die *ActivateTabMso*-Methode des *ICustomUI*-Objekts *objRibbon_Tabwechsel* und geben die *idMso* des zu aktivierenden *tab*-Elements als Parameter an:

```
Private Sub cmdStart_Click()
    objRibbon_Tabwechsel.ActivateTabMso "TabHomeAccess"
End Sub
```

Beim benutzerdefinierten *tab*-Element verwenden Sie einfach die Methode *ActivateTab*:

```
Private Sub cmd1_Click()
    objRibbon_Tabwechsel.ActivateTab "tab1"
End Sub
```

Damit das Beispiel zuverlässig funktioniert, weisen Sie den Namen der *customUI*-Definition der Eigenschaft *Name des Menübands* des Formulars zu. Dieses wird dann beim Öffnen des Formulars auf jeden Fall angezeigt.

12.8 Bilder im customUI

Schaltflächen, Galerien und Co. laden dazu ein, benutzerdefinierte oder eingebaute Bilder anzuzeigen. Am einfachsten ist dies für eingebaute Bilder. Benutzerdefinierte Bilder fügen Sie per Code hinzu. Wenn Sie diesen jedoch einmal in Ihre Anwendung integriert haben, bedeutet dies jedoch auch kaum noch Aufwand.

12.8.1 Eingebaute Bilder anzeigen

Wenn Sie einem benutzerdefinierten Steuerelement wie etwa einer Schaltfläche ein eingebautes Bild zuweisen möchten, brauchen Sie lediglich die *idMso* des Steuerelements,

das dieses Bild anzeigt. Wie Sie diese ermitteln, erfahren Sie in »Eingebaute Elemente in benutzerdefinierten Ribbons« ab Seite 108.

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="btn1" imageMso="FileSave" label="Datei speichern" size="large"/>
```

Die Schaltfläche sieht dann wie in Abbildung 12.25 aus.

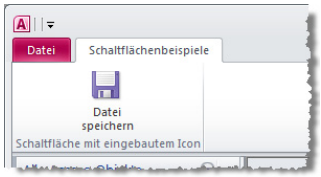


Abbildung 12.25: Eine Schaltfläche mit einem eingebauten Symbol

12.8.2 Benutzerdefinierte Bilder anzeigen

Womöglich möchten Sie ja auch eigene Icons im *customUI* anzeigen. Bevor Sie die folgenden Schritte durchführen, müssen Sie sich überlegen, wo Sie die zu verwendenden Icons speichern wollen. Sie können diese im Dateisystem speichern oder auch direkt in der Datenbank. Letzteres scheint sinnvoller, da der Benutzer die Bilddateien nicht versehentlich löschen kann. Beim Speichern von Bilddateien in einer Tabelle gibt es ebenfalls zwei Methoden: Entweder Sie verwenden ein OLE-Objekt-Feld oder ein Anlagefeld.

Im Folgenden werden wir die Vorgehensweise für das Speichern der Bilder in der Tabelle *MSysResources*-Tabelle beschreiben. Dort gespeicherte Ribbonbilder können Sie dann gegebenenfalls auch in Formularen und Berichten verwenden und umgekehrt.

Benutzerdefinierte Bilder in MSysResources speichern

Benutzerdefinierte Bilder gelangen per Einzeiler in die Systemtabelle *MSysResources*:

```
CurrentProject.AddSharedImage "<Bildname>", "<Dateiname>"
```

Als *<Bildname>* geben Sie den Namen an, unter dem Sie später auf das Bild zugreifen wollen, *<Dateiname>* enthält Pfad und Name der zu importierenden Bilddatei. Unter Access 2007 konnte das Ribbon nur Bilder mit 16 x 16 und 32 x 32 Pixeln anzeigen, mittlerweile gibt es im Backstage-Bereich aber auch die Möglichkeit, größere Bilder mit 64 x 64 Pixeln darzustellen. Im neuen *imageControl*, das allerdings nur im Backstage-Element bereitsteht, können Sie sogar noch größere Bilddateien anzeigen.

Bilder in Steuerelemente laden

Es gibt zwei Möglichkeiten, benutzerdefinierte Bilder in ein Steuerelement zu laden. Die beiden Varianten unterschieden sich in den Callback-Attributen, welche die VBA-Funktionen zum Laden der Bilder aufrufen.

» Die Callback-Funktion für das Attribut *loadImage* des Elements *customUI* wird beim Laden der Ribbon-Definition für jedes *image*-Attribut einmal aufgerufen. Wenn das Ribbon also vier *button*-Elemente besitzt, für die das *image*-Attribut angegeben wurde, wird *loadImage* vier Mal aufgerufen. Der Rumpf der entsprechenden Callback-Funktion sieht wie folgt aus. *control* liefert dabei den im *image*-Attribute gespeicherten Wert und *image* erwartet einen Objektverweis auf ein *StdPicture*-Objekt:

```
Public Sub loadImage(control, ByRef image)
    ...
End Sub
```

» Die Callback-Funktion für das Attribut *getImage* eines jeden Steuerelements wird beim Anzeigen des entsprechenden Steuerelements genau einmal aufgerufen. Der erste Parameter des nachfolgend vorgestellten Prozedurrumpfes liefert im Gegensatz zur *loadImage*-Funktion nicht den Wert des Attributs *image*, sondern einen Verweis auf das *IRibbonControl*-Objekt des betroffenen Steuerelements. Dieses liefert über die Eigenschaften *id* den Namen oder über *tag* den Wert des entsprechenden Attributs:

```
Sub getImage(control As IRibbonControl, ByRef image)
    ...
End Sub
```

Wann setzt man nun die erste und wann die zweite Variante ein? Der technisch wichtige Unterschied tritt in Erscheinung, wenn Sie die Bilder eines Steuerelements einmal nach dem ersten Anzeigen austauschen möchten. Die Grundlagen dazu werden weiter unten unter »Dynamisches Aktualisieren des Ribbons« ab Seite 118 erläutert.

Verwenden Sie die *loadImage*-Variante, müssen Sie mit der *Validate*-Methode des *IRibbonUI*-Objekts das komplette Ribbon aktualisieren. Bei der *getImage*-Variante können Sie gezielt die Bilder eines einzigen Steuerelements austauschen, indem Sie die *ValidateControl*-Methode von *IRibbonUI* verwenden und den Namen des Steuerelements als Paramter angeben.

Schritt für Schritt

Das Anzeigen eines Bildes für ein Ribbon-Steuerelement geschieht wie folgt – hier zunächst die Variante mit *loadImage*:

» Sie speichern ein Bild in der Tabelle *MSysResources*. Dieses sollte den Datentyp *.gif*, *.png* oder *.ico* haben – Transparenz ist hier für ein ordentliches Erscheinungsbild unbedingt erwünscht!

» Sie weisen dem Steuerelement das Attribut *image* mit dem Namen des Bildes in der Tabelle *MSysResources* zu. Außerdem legen Sie für das Attribut *loadImage* des *customUI*-Elements den Wert *loadImage* als Name der aufzurufenden Callback-Funktion fest:

```
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui"
    loadImage="loadImage">
  <ribbon>
    <tabs>
      <tab id="tab1" label="Schaltflächenbeispiele">
        <group id="grp1" label="Benutzerdefiniertes Icon">
          <button id="btn1" image="about_32" label="Informationen" size="large"/>
        </group>
      </tab>
    </tabs>
  </ribbon>
</customUI>
```

» Dann legen Sie in einem Standardmodul etwa namens *mdlRibbons* die Callback-Funktion *loadImage* an und ergänzen nur eine einzige Zeile:

```
Public Sub loadImage(control, ByRef image)
    Set image = PicFromSharedResource_Ribbon(CStr(control))
End Sub
```

» Nun brauchen Sie nur noch die hier aufgerufene Funktion *PicFromSharedResource_Ribbon*, die allerdings noch einige weitere Funktionen aufruft. Der komplette benötigte Code befindet sich im Modul *mdlRibbonImages* der Beispieldatenbank zu diesem Kapitel. Kopieren Sie also einfach den Inhalt dieses Moduls in eine neues Modul und speichern Sie es unter dem Namen *mdlRibbonImages* oder importieren Sie dieses Modul einfach in die Zieldatenbank.

Fertig! Die zweite Variante ist sieht nicht viel anders aus. Sie nehmen hier alle Anpassungen direkt am *button*-Element vor. Das *loadImage*-Attribut im *customUI*-Element ist hier nicht mehr nötig:

```
<button id="btn1" getImage="getImage" label="Button mit Bild"/>
```

Für die Callback-Funktion gibt es hier mehrere Varianten. Die erste wertet den Wert des *id*-Attributs des *button*-Elements aus und übergibt der Funktion *PicFromSharedRecource_Ribbon* einen entsprechenden Bildnamen, hier *Beispielbild*:

```
Sub getImage(control As IRibbonControl, ByRef image)
    Dim strImage As String
    Select Case control.ID
        Case "btn1"
            strImage = "Beispielbild"
    End Select
    Set image = PicFromSharedResource_Ribbon(strImage)
End Sub
```

Die zweite Variante nutzt die Möglichkeit, dass Sie jedem Element über das *tag*-Attribut einen benutzerdefinierten Wert zuweisen können. In diesem Falle soll *tag* den Namen des Bildes in der Tabelle *MSysResources* enthalten:

```
<button id="btn2" tag="Beispielbild" getImage="getImage" label="Button mit Bild"/>
```

Die Callback-Funktion erhält den angegebenen Wert über die *Tag*-Eigenschaft der Variablen *control*:

```
Sub getImage(control As IRibbonControl, ByRef image)
    Set image = PicFromSharedResource_Ribbon(CStr(control.Tag))
End Sub
```

Die dritte Variante geht schlicht davon aus, dass Sie in der Tabelle *MSysResources* ein Bild vorhalten, dessen Namen mit dem *id*-Wert des Steuerelements übereinstimmt:

```
Sub getImage(control As IRibbonControl, ByRef image)
    Set image = PicFromSharedResource_Ribbon(CStr(control.ID))
End Sub
```

Die Prozedur *PicFromSharedResource_Ribbon* prüft selbständig, ob die Tabelle *MSysResources* vorhanden ist und die benötigten Bilder enthält. Falls nicht, liefert sie eine entsprechende Hinweismeldung.

12.9 Die Ribbon-Steuerelemente

Die folgenden Abschnitte stellen die Steuerelemente und ihre wichtigsten Eigenschaften vor, außerdem lernen Sie hier Einsatzmöglichkeiten der Callback-Routinen kennen.

12.9.1 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« einreiht (siehe Abbildung 12.26).

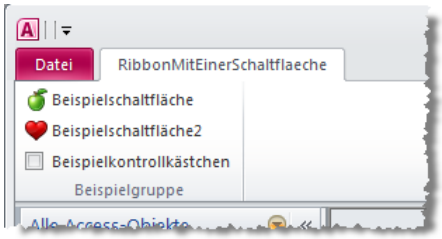


Abbildung 12.26: Beispiel für ein Kontrollkästchen

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="chk1_onAction" />
```

Die aufgerufene Funktion heißt *onActionKontrollkaestchen*. 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 chk1_onAction(ctl As IRibbonControl, pressed As Boolean)
    MsgBox "Das Kontrollkästchen '" & ctl.ID & "' hat den Wert '" & pressed & "'."
End Function
```

Listing 12.6: Auswertung des Wertes eines Kontrollkästchens

12.9.2 Textfelder

Textfelder heißen im Ribbon-Slang *editBox*-Steuerelement. Eine Besonderheit eines 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 EditBox im XML-Dokument erfolgt wie beim Kontrollkästchen:

```
<editBox id="txtEditbox" label="Beispieltextfeld" onChange="onChange"/>
```

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

```
Sub onChange(control As IRibbonControl, text As String)
    MsgBox "Das Steuerelement '" & control.ID & "' enthält nun den Wert '" & text & "'"
End Sub
```

Listing 12.7: Diese Routine wird durch das Ereignis *onChange* einer *EditBox* ausgelöst und zeigt den Namen des Steuerelements sowie den enthaltenen Text an

Das Aussehen des oben definierten Textfeldes sowie die Meldung, die Listing 12.7 verursacht, zeigt Abbildung 12.27.

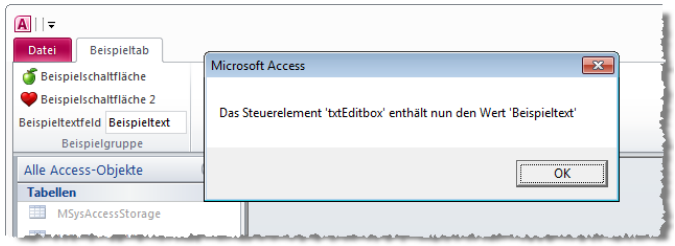


Abbildung 12.27: Ein Ribbon mit Textfeld und einer Meldung, die nach dem Aktualisieren des Textfelds angezeigt wird

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 Beispieldefinition 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 id="txtEditbox" label="Beispieltextfeld" sizeString="André Minhorst"
maxLength="50" onChange="onChange"/>
```

Das Resultat zeigt Abbildung 12.28.

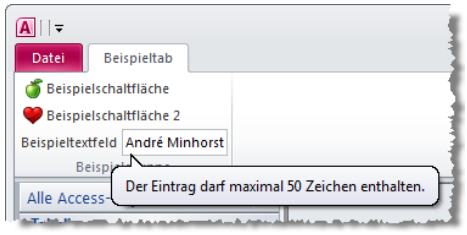


Abbildung 12.28: Die EditBox meldet das Überschreiten der zulässigen Zeichenanzahl

12.9.3 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/2009/07/customui" onLoad="OnLoad"
loadImage="LoadImage">
  <ribbon startFromScratch="true">
    <tabs>
      <tab id="tabRibbonMitEinerSchaltflaeche" visible="true"
          label="RibbonMitEinerSchaltflaeche">
        ...
        <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 12.8: 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 12.29.

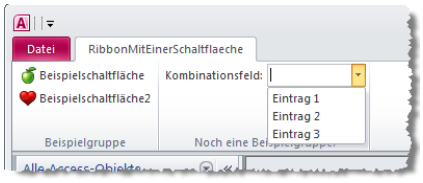


Abbildung 12.29: Ein Ribbon mit einer zweiten Gruppe und einem einfachen Kombinationsfeld

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/2009/07/customui" onLoad="OnLoad"
loadImage="LoadImage">
  <ribbon startFromScratch="true">
    <tabs>
      <tab id="tabKombinationsfeldbeispiel" visible="true"
          label="Kombinationsfeldbeispiel">
        <group id="grpBeispielgruppe2" label="Noch eine Beispielgruppe.">
          <comboBox id="cboBeispielkombinationsfeld" label="Kombinationsfeld:"
              getItemCount="GetItemCount" getItemLabel="GetItemLabel"
              getItemID="GetItemID"/>
        </group>
      </tab>
    </tabs>
  </ribbon>
</customUI>
```

Listing 12.9: 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 12.30 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 12.10: Callback-Funktionen zum Füllen eines einfachen Kombinationsfeldes

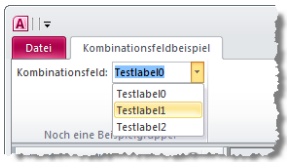


Abbildung 12.30: Kombinationsfeld mit dynamisch zusammengestelltem Inhalt

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:

```
Dim strKontakte() As String
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 12.11: 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 12.31).

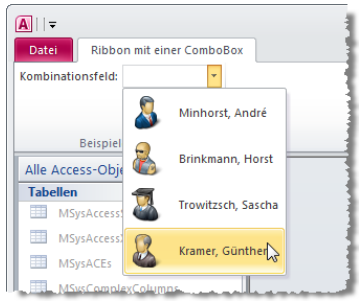


Abbildung 12.31: Ein comboBox-Steuerelement mit Symbolen

Und dies funktioniert ganz einfach – in diesem Fall für vier Bilddateien, die unter den Namen *1*, *2*, *3* und *4* in der Tabelle *MSysResources* gespeichert wurden. 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 aus der Ressourcen-Tabelle ist wiederum die Routine *PicFromSharedResource* verantwortlich.

```
Public Sub GetItemImage_DB(control As IRibbonControl, index As Integer, ByRef image)
    Set image = PicFromSharedResource_Ribbon(index + 1)
End Sub
```

Listing 12.12: 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 12.13: 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.

12.9.4 Kombinationsfelder II: Das dropDown-Element

Warum gibt es zwei Typen von Kombinationsfeldern und welche Unterschiede weisen diese auf? Einen genauen Einblick gibt die Ribbon-Referenz, die Sie im Download zu diesem Buch unter *Ribbon-Referenz.pdf* finden. Diese enthält 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 Ereigniseigenschaften *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 ebenfalls *Ribbon-Referenz.pdf* im Download zu diesem Buch): 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 Indexes des Eintrags.

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

```
<dropDown id="cboBeispielkombinationsfeld"
    label="Kombinationsfeld:" getItemCount="GetItemCount_DB"
    getItemID="GetItemID_DB" getItemLabel="GetItemLabel_DB"
    getItemImage="GetItemImage_DB" onAction="OnAction_DB"
    getSelectedItemIndex="GetSelectedItemIndex_DB">
```

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

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

Listing 12.14: Anzeigen der ID und des Indexes eines neu ausgewählten Eintrags eines *dropDown*-Steuerelements

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

Listing 12.15: 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 »Dynamisches Aktualisieren des Ribbons« ab Seite 118).

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.

Gegebenenfalls fällt noch das Attribut *invalidateContentOnDrop* des *comboBox*-Steuerelements ins Gewicht, das dafür sorgt, dass dieses beim Aufklappen neu eingelesen wird.

12.9.5 Umschaltflächen

Mit Umschaltflächen lassen sich Status 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="onAction_Toggle" getPressed="GetPressed_Toggle"/>
```

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

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

Listing 12.16: 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 12.17: Ob ein *toggleButton*-Element beim Anlegen gedrückt oder nicht gedrückt ist, legt die durch *getPressed* ausgelöste Routine fest

Sie können die Prozedur *GetPressed* also mit Anweisungen füllen, die beim Anzeigen des *toggleButton*-Elements einen entsprechenden Wert übergibt und somit den Status der Umschaltfläche festlegt.

12.9.6 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 12.32 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 12.33.

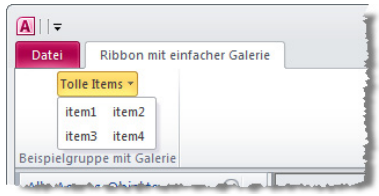


Abbildung 12.32: Eine Galerie mit vier einfachen Einträgen

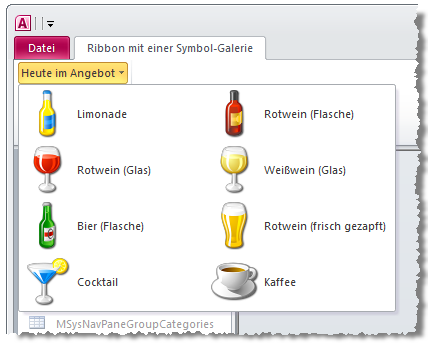


Abbildung 12.33: Eine Galerie mit Symbolen

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. Das Ribbon verwendet die gleiche *LoadImage*-Funktion wie oben.

```
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/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"/>
    <item id="id2" label="Rotwein (Flasche)" image="wine_red_bottle"/>
    <item id="id3" label="Rotwein (Glas)" image="wine_red_glass"/>
    <item id="id4" label="Weißwein (Glas)" image="wine_white_glass"/>
    <item id="id5" label="Bier (Flasche)" image="beer_bottle"/>
    <item id="id6" label="Bier (frisch gezapft)"
    image="beer_glass"/>
    <item id="id7" label="Cocktail" image="cocktail"/>
    <item id="id8" label="Kaffee" image="cup"/>
  </gallery>
</customUI>
```

Listing 12.18: Dieses *gallery*-Element zeigt acht Items mit passenden Symbolen an

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 ist, dass Sie mit dem Attribut *invalidateContentOnDrop* festlegen können, ob das *gallery*-Element beim Öffnen neu gefüllt werden soll (*true*/*false*).

Weitere interessante Attribute sind folgende:

» *itemHeight*: Gibt die Höhe des *item*-Elements an.

» *itemWidth*: Gibt die Breite des *item*-Elements an.

Die Ereigniseigenschaft 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)
```

12.9.7 Menüs (menu)

Menüs ähneln den von älteren Office-Versionen bekannten Untermenüs der Menüleiste. Dabei kann man einige unterschiedliche Steuerelemente anlegen, angezeigt werden al-

lerdings stets nur normale Menüeinträge. Das folgende Ribbon-XML-Dokument erzeugt beispielsweise das Menü aus Abbildung 12.34.

```
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/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 12.19: Beispiel für ein Menü, dessen verschiedenartige Steuerelemente jedoch alle in Standardmenüeinträgen enden

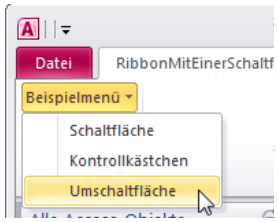


Abbildung 12.34: Das menu-Steuerelement mit einigen Untersteuerelementen

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 12.35.

Der passende XML-Code sieht wie folgt aus:

```
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui" onLoad="OnLoad"
loadImage="LoadImage">
  <ribbon startFromScratch="false">
    <tabs>
      <tab id="tabSport" visible="true" label="Sport">
        <group id="grpSportarten" label="Sportarten">
          <menu id="mnuSportartAuswaehlen" label="Sportart auswählen" itemSize="large">
            <button id="btnTennis" image="tennis_ball" label="Tennis-Abteilung"
              description="Tennis ist eine schöne Sportart."/>
            <button id="btnBaseball" image="baseball" label="Baseball-Abteilung"
              description="Baseball auch."/>
            <button id="btnBasketball" image="basketball" label="Basketball-Abteilung"
              description="Und Basketball ist erstmal toll!"/>
            <button id="btnFussball" image="soccer_ball" label="Fussball-Abteilung"
              description="&quot;Fussballspielen kann ja jeder.&quot;"/>
            <button id="btnFootball" image="football" label="Football-Abteilung"
              description="Football hat komplizierte Regeln..."/>
          </menu>
        </group>
      </tab>
    </tabs>
  </ribbon>
</customUI>
```

Listing 12.20: Diese XML-Definition zeigt beispielhaft, wie man Menüelemente mit Beschreibungen versieht

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

Abbildung 12.35: Im *menu*-Steuerelement lassen sich neben Symbolen auch Beschreibungstexte zu einem Steuerelement anzeigen

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 12.36:

```
<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>
```

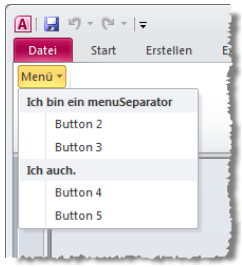


Abbildung 12.36: Ein Menü mit zwei textbehafteten Trennlinien

12.9.8 Dynamische Menüs (dynamicMenu)

Das *dynamicMenu*-Element ist prinzipiell mit dem *menu*-Element identisch; für die grundlegenden Eigenschaften lesen Sie daher den vorherigen Abschnitt. Es gibt allerdings einen entscheidenden Unterschied, der sich im Attribut *getContent* manifestiert. Damit können Sie nämlich den Inhalt des *dynamicMenu*-Steuerelements zur Laufzeit im XML-Format zusammensetzen. Für das *getContent*-Attribut müssen Sie auf jeden Fall einen Wert angeben (also den Namen der Callback-Funktion), da das Anzeigen des Ribbons sonst einen Fehler auslöst.

Das folgende Beispiel eines *dynamicMenu*-Elements sieht sehr überschaubar aus:

```
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui">
  <ribbon>
    <tabs>
      <tab id="tab1" label="Beispiel für ein dynamicMenu-Element">
        <group id="grp1" label="Beispielgruppe">
          <dynamicMenu id="dyn1" label="Dynamisches Menü"
```

```
            getContent="getContent"/>
        </group>
      </tab>
    </tabs>
  </ribbon>
</customUI>
```

Die *getContent*-Callback-Funktion muss nun den XML-Code hinzufügen, der normalerweise innerhalb eines handelsüblichen *menu*-Elements steht – im einfachsten Fall also etwa so:

```
<button id="btn1" label="Button 1"/>
<button id="btn2" label="Button 2"/>
<button id="btn3" label="Button 3"/>
```

In diesem Fall enthalten die button-Elemente allerdings noch nicht einmal onAction-Callback-Attribute, was diese relativ nutzlos macht. Aus Gründen der Übersicht lassen wir diese aber auch in folgendem Quellcode weg:

```
Sub getContent(control As IRibbonControl, ByRef content)
    Dim strXML As String
    Dim i As Integer
        strXML = "<menu xmlns=""http://schemas.microsoft.com/office/2009/07/customui"">"
    For i = 1 To 3
        strXML = strXML & "<button id=""btn" & i & """ label=""Button " _
            & i & """/>"
    Next i
    strXML = strXML & "</menu> "
    content = strXML
End Sub
```

12.9.9 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 12.37 notwendigen Elemente:

```
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/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">
        <splitButton id="spbSplitButton" size="large">
          <button id="cmdButton" label="Rot"
          image="trafficlight_red"/>
          <menu id="mnuMenu" label="Beispielmenü" itemSize="large">
            <button id="btnButton1" label="Rot"
            image="trafficlight_red"/>
            <button id="btnButton2" label="Rot-Gelb"
            image="trafficlight_red_yellow"/>
            <button id="btnButton3" label="Grün"
            image="trafficlight_green"/>
          </menu>
        </splitButton>
      </group>
    </tab>
  </tabs>
</ribbon>
</customUI>
```

Listing 12.21: Dieses XML-Dokument erzeugt das *splitButton*-Element aus Abbildung 12.37

Abbildung 12.37: Das *splitButton*-Steuerelement zeigt ständig eine Schaltfläche und bietet andere zur Auswahl an

12.9.10 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 12.38. 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*-Callback-Funktion von Schaltflächen hat.

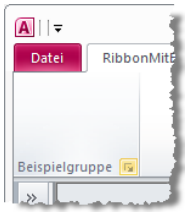


Abbildung 12.38: Eine Gruppe mit einem *dialogBoxLauncher*-Element

12.9.11 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 12.39:

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

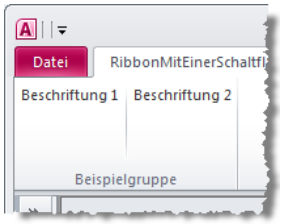


Abbildung 12.39: Beispiel für einen Trennstrich (*separator*)

12.10 Weitere Anpassungen des Ribbons

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

12.10.1 Eingebaute Elemente in benutzerdefinierten Ribbons

Sie können auch eingebaute Elemente in Ihren eigenen Ribbons anzeigen. Sie legen dazu genau wie bei benutzerdefinierten Elementen jeweils ein *group*- oder *control*-Element an. Sie müssen jedoch nur eine einzige Eigenschaft namens *idMso* einstellen: Diese enthält den Namen des jeweiligen eingebauten Elements der Benutzeroberfläche.

Für die Steuerelemente erhalten Sie diese Bezeichnung in den Access-Optionen, genauer gesagt im Bereich *Symbolleiste für den Schnellzugriff*. Wählen Sie im linken Listenfeld den gewünschten Eintrag aus und verharren Sie mit der Maus darüber – Access zeigt dann einen Hilfetext an, der auch den englischen Namen preisgibt (siehe 12.10.1).

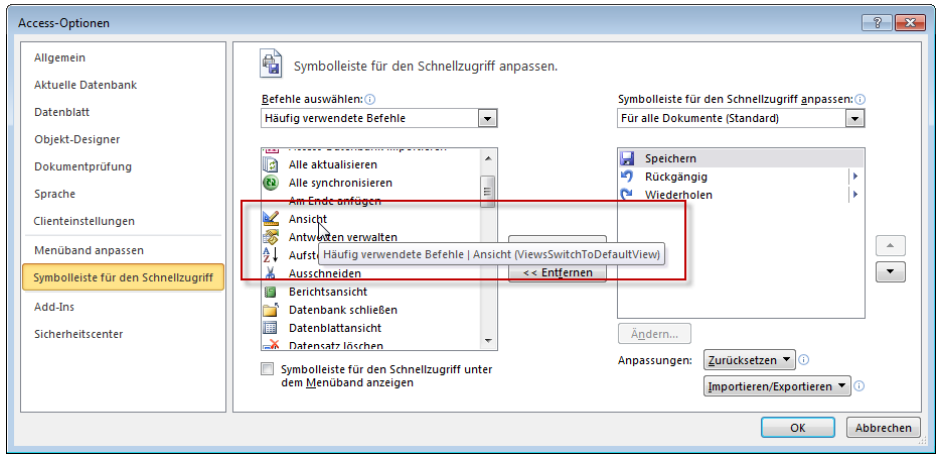


Abbildung 12.40: Ermitteln der *idMso* eines Steuerelements

Diesen Wert tragen Sie dann für das Attribut *idMso* des *group*- oder *control*-Elements ein. Für ein *group*-Element sieht das etwa so aus:

```
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui">
  <ribbon>
    <tabs>
      <tab id="tab1" label="Erstes Tab">
        <group idMso="GroupClipboard"/>
      </tab>
```

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

Komplette *tab*-Elemente lassen sich so leider nicht zu einem benutzerdefinierten Ribbon hinzufügen. Diese Funktion ergänzt der *Ribbon-Admin 2010*: Er erstellt ein neues *tab*-Element und fügt die in diesem *tab* enthaltenen *groups* hinzu, sodass das Tab wie das Original erscheint.

Alternativ ermitteln Sie die *idMso* aus einer Excel-Tabelle, die Microsoft für alle Office-Anwendungen zum Download bereitstellt. Die Access-Version finden Sie im Download zu diesem Kapitel. Die Datei *AccessControls.xlsx* enthält eine Liste aller *tab*-, *group*- und sonstiger eingebauter Steuerelemente.

12.10.2 Tastenkombinationen

Genau wie bei älteren Access-Versionen können Sie auch in Access 2010 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.

Natürlich können Sie auch für benutzerdefinierte Elemente diese sogenannten 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/2009/07/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 12.22: 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 12.41). 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 12.42).

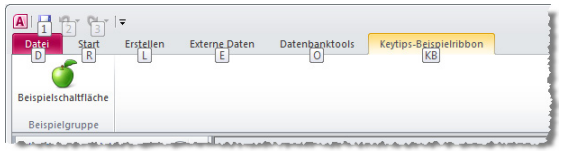


Abbildung 12.41: Nach dem Betätigen der Alt-Taste erscheinen zunächst die Tastenkombinationen für die Auswahl eines Ribbons ...

Abbildung 12.42: ... 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.

12.10.3 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 12.43 aus:

```
<button id="btnBeispielschaltflaeche" image="about_32" label="Beispielschaltfläche"
screentip="Dies ist die Überschrift des Hilfetextes (max. 1024 Zeichen) ..."
supertip="... und das ist der eigentliche Hilfetext (auch nur 1024 Zeichen)."
onAction="Beispielfunktion" size="large"/>
```

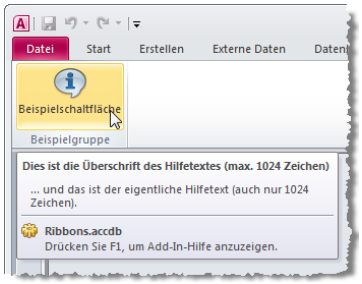


Abbildung 12.43: Eine Schaltfläche mit *screentip*- (Überschrift) und *supertip*-Attribut (Text)

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

12.10.4 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|Menüband- und Symbolleistenoptionen|Name des Menübands* angegebene Ribbon an. Im Gegensatz zu Access 2007, wo das Office-Menü ebenfalls erheblich ausgedünnt wurde, erscheint der Backstage-Bereich von Access in voller Pracht.

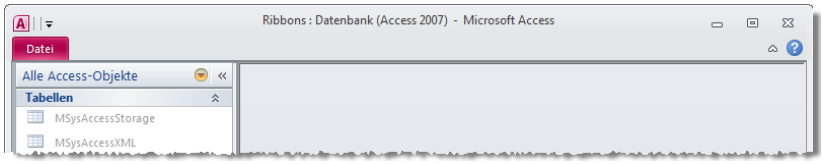


Abbildung 12.44: Das Setzen von *startFromScratch* auf *true* entfernt alle Registerkartenreiter außer Datei

12.10.5 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 *Menüband 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.

12.10.6 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 die *idMso* des betroffenen Elements und den Sichtbarkeitsstatus enthält und fügen das Attribut *visible* mit dem Wert *false* hinzu..

Das folgende Element blendet das *tab*-Element *Erstellen* 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 – mehr dazu unter »Eingebaute Elemente in benutzerdefinierten Ribbons« ab Seite 108.

12.10.7 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/2009/07/customui"
loadImage="LoadImageSplitMenu" onLoad="OnLoad">
  <ribbon>
    <tabs>
      <tab idMso="TabHomeAccess">
        <group idMso="GroupViews" visible="false" />
      </tab>
    </tabs>
  </ribbon>
</customUI>
```

Listing 12.23: 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="getVisible" />
```

Die Sichtbarkeit wird dann in der Callback-Funktion gesteuert, deren Abfrage Sie mit der Methode *InvalidateControl* (siehe »Dynamisches Aktualisieren des Ribbons« ab Seite 118) jederzeit erzwingen können.

12.10.8 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*.

12.10.9 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/2009/07/customui">
  <commands>
    <command idMso="FiltersMenu" enabled="false"/>
  </commands>
</customUI>
```

Die *idMso* entnehmen Sie der im Download zu diesem Kapitel 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"/>
```

12.10.10 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 *Ausschneiden*-Schaltfläche die VBA-Routine *OnActionCut* auslöst.

```
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui">
  <commands>
    <command idMso="Cut" onAction="onActionCut"/>
```

```
    </commands>
</customUI>
```

Diese Prozedur wiederum teilt dem Benutzer mit, dass sie aktuell keine Lust hat und bricht die aufgerufene Funktion ab:

```
Sub onActionCut(control As IRibbonControl, ByRef CancelDefault)
    MsgBox "Keine Lust."
    CancelDefault = True
End Sub
```

Sie können also einen eingebauten Befehl durch eigenen Code ersetzen, den eigentlichen Befehl aber anschließend auch noch ausführen lassen. In diesem Fall stellen Sie *CancelDefault* auf *False* ein. Aber Achtung: Im Gegensatz zu ähnlichen Ereignissen in Formularen ist *CancelDefault* nicht standardmäßig mit False vorbelegt – das heißt, dass die eigentlich ausgelöste Aktion ohne Ihr Zutun immer abgebrochen wird. Erst *CancelDefault = False* führt auch den eigentlichen Befehl noch aus.

12.10.11 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 *"*.

12.10.12 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 12.45:

```
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui"
loadImage="LoadImage">
  <ribbon startFromScratch="true">
    <qat>
      <documentControls>
        <button id="btnHerz" image="heart_16" label="Herz"/>
```

```
        <button id="btnApfel" image="apple" label="Apfel"/>
      </documentControls>
    </qat>
  </ribbon>
</customUI>
```

Listing 12.24: Hinzufügen benutzerdefinierter Schaltflächen zur Schnellzugriffsleiste

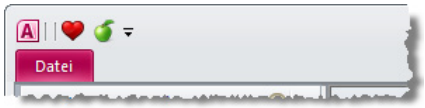


Abbildung 12.45: Die Schnellzugriffsleiste mit benutzerdefinierten Schaltflächen

12.11 Ribbons für Formulare und Berichte

Genau wie in Access 2007 können Sie auch in Access 2010 Formularen und Berichten Ribbon-Tabs zuweisen. Dazu weisen Sie einfach der Eigenschaft *Name des Menübands* eines der verfügbaren benutzerdefinierten Ribbons zu (siehe Abbildung 12.46). 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 des Menübands* den passenden Eintrag zu und modal geöffnete Berichte versagen die Anzeige des angegebenen Menübands. Auch in der Layoutansicht von Berichten zeigt sich ein angegebenes Menüband nicht, wohl aber in der neuen Berichtsansicht.

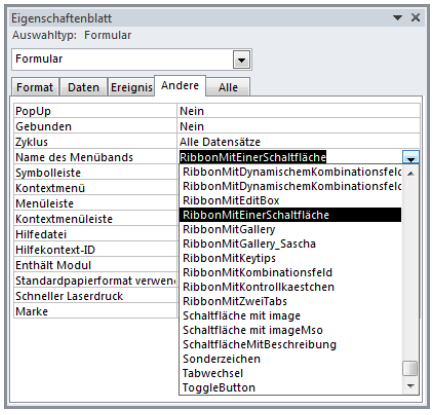


Abbildung 12.46: Auswählen eines benutzerdefinierten Ribbons für ein Formular

Kontextsensitive Ribbons

Vielleicht haben Sie schon einmal die andersfarbigen Tabs auf der rechten Seite des Ribbons bemerkt, die beim Aktivieren etwa der Datenblattansicht oder im Entwurfsmodus erscheinen und automatisch aktiviert werden.

Auch solche Tabs können Sie selbst verwenden, um diese beispielsweise beim Öffnen von Formularen oder Berichten anzuzeigen. Dazu legen Sie unterhalt des *ribbon*-Elements nicht das *tabs*-, sondern das *contextualTabs*-Element an und weisen seiner *idMso* den Wert *TabSetFormReportExtensibility* zu. Darunter legen Sie dann einfach ein oder mehrere *tab*-Elemente mit den gewünschten Inhalten an:

```
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui">
  <ribbon>
    <contextualTabs>
      <tabSet idMso="TabSetFormReportExtensibility">
        <tab id="tab1" label="Kontextsensitives Tab">
          ...
        </tab>
      </tabSet>
    </contextualTabs>
  </ribbon>
</customUI>
```

Nach dem Öffnen des Formulars sieht das (noch leere) Ribbon wie in Abbildung 12.47 aus.

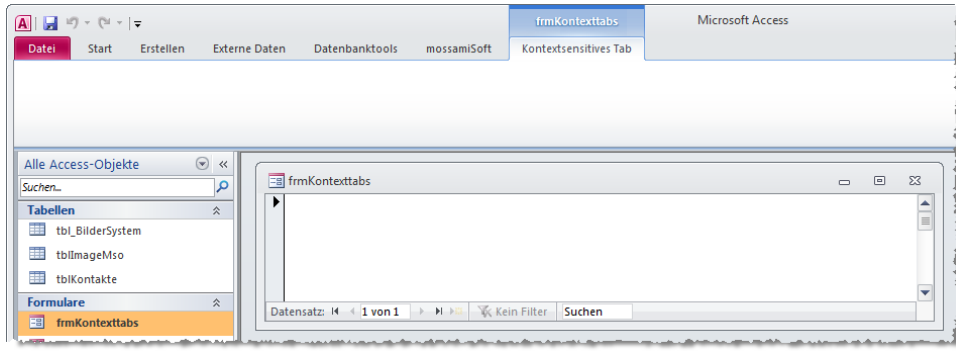


Abbildung 12.47: Ein in Zusammenhang mit einem Formular eingeblendetes *tab*-Element

Callbacks im Klassenmodul des Formulars

Normalerweise legen Sie die Callback-Funktionen des Ribbons in einen Standardmodul ab. Dies ist aber nicht die einzige Möglichkeit: Wenn Sie etwa ein Ribbon für ein Formular definieren, können Sie den durch das Ribbon ausgelösten Code unter bestimmten

Bedingungen auch direkt im Klassenmodul des Formulars unterbringen. Das *onAction*-Attribut legen Sie dabei wie folgt auf den Wert *=onAction_Form()* fest:

```
<button id="btn1" image="form_yellow_32" label="Beispielbutton" onAction="=onAction_
Form()"/>
```

Wenn Sie nun die folgende Funktion im Klassenmodul des Formulars unterbringen und das Ribbon unter *Name des Menübands* in den Eigenschaften des Formulars angeben, lösen Sie mit einem Klick auf die Ribbon-Schaltfläche direkt diese Funktion aus:

```
Private Function OnAction_Form()
    MsgBox "onAction im Formular wurde aufgerufen"
End Function
```

Sie können so sogar noch benutzerdefinierte Parameter übergeben. Allerdings ist es nicht mehr möglich, Werte für *get...*-Callbacks zurückzugeben – dies müssen Sie dann doch in einem Standardmodul erledigen.

12.12 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.

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 12.26) 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|Menüband- und Symbolleistenoptionen|Name des Menübands* auswählen.

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

```
    Application.LoadCustomUI "RibbonAusDatei", strRibbon
End Sub
```

Listing 12.25: 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 12.26: 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.

12.12.1 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.

Stattdessen 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/2009/07/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 12.27: 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. 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.

Beachten Sie: Es kann sein, dass Sie zwei oder mehr CustomUI-Definitionen gleichzeitig verwenden. Dies geschieht, wenn Sie etwa ein Anwendungsribbon festlegen (etwa über die Access-Optionen) und zusätzlich eine CustomUI-Anpassung für ein Formular oder einen Bericht laden. Nicht nur, dass die Anpassungen gleichzeitig sichtbar sein können: Auch auf deren Steuerelemente möchten Sie möglicherweise etwa mit der *InvalidateControl*-Methode zugreifen. Und dazu benötigen Sie einen Verweis auf das *IRibbonUI*-Objekt der jeweiligen Ribbon-Definition. Sie müssen dazu für jedes Ribbon ein eigenes *IRibbonUI*-Objekt mit einem Verweis auf das jeweilige Ribbon füllen! Wenn Sie nur ein Objekt namens *objRibbon* deklarieren und diesem dann nacheinander in den *OnLoad*-Ereignissen mehrerer Ribbons die Verweise auf die *IRibbonUI*-Elemente aller Ribbons zuweisen, wird jeweils nur der letzte Verweis gespeichert!

Deklarieren Sie also etwa wie folgt für jedes Ribbon eine eigene Variable:

```
Dim objRibbonMain As IRibbonUI
Dim objRibbonfrmKunden As IRibbonUI
```

Sie können dann gezielt auf die einzelnen Ribbon-Instanzen zugreifen.

12.12.2 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 (siehe Abbildung 12.48).

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

```
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/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 12.28: Ribbon-XML-Dokument für abhängige Kombinationsfelder

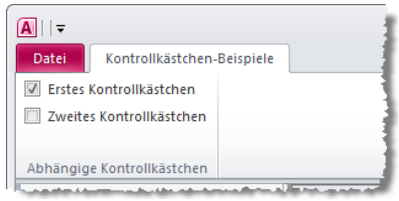


Abbildung 12.48: Das untere Kontrollkästchen ist aktiviert, wenn das obere Kontrollkästchen markiert ist und umgekehrt

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

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

Listing 12.29: 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 12.30: 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 12.31: 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 12.32: 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.

12.13 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. Allerdings gibt es zwei Möglichkeiten, eine benutzerdefinierte Menü- und Symbolleistenstruktur dennoch anzuzeigen. Ob und wie das geschieht, hängt von der Option *Integrierte Symbolleisten zulassen* ab (siehe Abbildung 12.49). Wird diese akti-

viert, erscheint die Menüleiste in einem separaten Ribbon-Tab namens *Add-Ins* (siehe Abbildung 12.50) Falls nicht, wird das Ribbon komplett ausgeblendet und es erscheint stattdessen die Menüleiste (siehe Abbildung 12.51). Diese kann allerdings nicht mehr beliebig platziert werden.

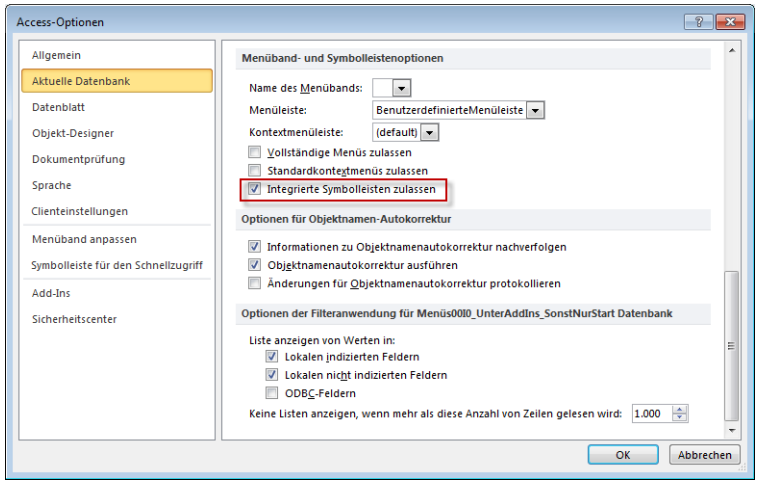


Abbildung 12.49: Diese Option entscheidet, ob eine Menüleiste statt des Ribbons oder im Ribbon angezeigt wird.

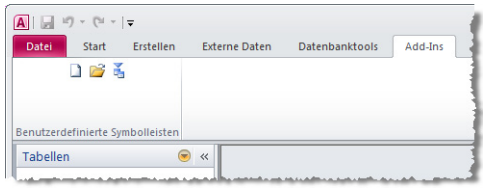


Abbildung 12.50: Menüleiste in einer Gruppe des Ribbon-Tabs *Add-Ins*

Abbildung 12.51: Menü statt Ribbon

Diese Einstellung können Sie sowohl von Access 2003 und älter aus vornehmen als auch in der unter Access 2007/2010 geöffneten *.mdb*-Datenbank. Wichtig ist allein, dass es sich um eine *.mdb*-Datenbank handelt. Für *.accdb*-Datenbanken werden keine Menü- und Symbolleisten mehr angezeigt.

Backstage-Kapitel aus: Das Access-Entwicklerbuch www.amvshop.dej

13

Backstage

Der Backstage-Bereich ersetzt das Office-Menü von Office 2007 und fügt viele weitere mögliche Elemente zu diesem Bereich hinzu. Im Gegensatz zum Office-Menü erstreckt sich der Backstage-Bereich gleich über das komplette Anwendungsfenster. Dem Entwickler bietet der Backstage-Bereich Funktionen an, die in Access 2003 und älter größtenteils im *Datei*-Menü, aber auch im *Extras*-Menü enthalten waren. Auch gegenüber Access 2007 gibt es einige Verbesserungen – so können Sie nun endlich mehr als nur neun Einträge in der Liste der zuletzt verwendeten Datenbanken anzeigen. Und es kommt noch besser: Es gibt sogar die Möglichkeit, oft benutzte Anwendungen ganz oben in der Liste »anzupinnen«. Sie als Leser dieses Buchs interessieren sich aber vermutlich viel mehr dafür, wie Sie den neuen Backstage-Bereich anpassen und wofür sie ihn einsetzen können. Aufbauend auf dem vorherigen Kapitel zum Thema Ribbon lernen Sie

auf den folgenden Seiten die Struktur und die Steuerelemente des Backstage-Bereichs kennen.

BEISPIELDATENBANKEN

Den Download mit dem Beispielen finden Sie unter *www.acciu.de/aeb2010*. Die Datenbank zu diesem Kapitel heißt *Backstage.accdb*.

Vorab einige Ideen für Einsatzbereiche:

» Optionen-Dialog: Während Sie normalerweise vermutlich ein eigenes Formular als Dialog für die Einstellungen Ihrer Datenbank verwendet haben, können Sie dies nun in den Backstage-Bereich auslagern.

» Informationen über die Anwendung: Wenn Sie eine Anwendung für einen Kunden programmieren, kann es sinnvoll sein, wenn dieser Ihnen schnell Informationen wie etwa den Dateipfad von Front- und/oder Backend oder auch die Versionsnummer mitteilen kann. Der Backstage-Bereich ist ein prima Ort, um solche Daten unterzubringen.

» Neben der Versionsnummer können Sie im Backstage-Bereich natürlich auch die Möglichkeit zum Updaten einer Anwendung unterbringen.

13.1 Elemente des Backstage-Bereichs

In den folgenden Abschnitten lernen Sie die Struktur und die verschiedenen Elemente des Backstage-Bereichs kennen.

ÜBERSICHTSTABELLEN

Übersichten über alle Backstage-Elemente finden Sie im *.pdf*-Format im Download zu diesem Buch unter *www.acciu.de/aeb2010*.

13.1.1 Das backstage-Element

Der Backstage-Bereich ist, dies suggeriert zumindest ein Blick auf die Benutzeroberfläche, ein Teil des Ribbons. Zumindest klicken Sie scheinbar auf ein Ribbon-Tab, um diesen Bereich zu öffnen. In der XML-Definition des CustomUI, also der Benutzeroberfläche von Access, stellt sich dies jedoch anders dar: Direkt unterhalb des *customUI*-Elements gibt es gleich vier weitere Bereiche. Einer davon heißt *ribbon* und umfasst die im vorigen

Kapitel beschriebenen Elemente, ein anderer *backstage*. Wenn Sie eine Anpassung des *backstage*-Bereichs vornehmen möchten, fügen Sie daher ein öffnendes und ein schließendes Element unterhalb des *customUI*-Elements hinzu:

```
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui">
  <backstage>
  </backstage>
</customUI>
```

Allein für sich ändert dieses Element nichts. Dies ist erst der Fall, wenn Sie die nachfolgend beschriebenen Elemente hinzufügen. Da das *backstage*-Element sich nicht auf die Benutzeroberfläche auswirkt, gibt es auch keine entsprechenden Attribute. Die einzigen Attribute dieses Elements machen sich vielmehr bemerkbar, wenn Sie den Backstage-Bereich öffnen und schließen: Es handelt sich dabei um die Callback-Attribute *onShow* und *onHide*.

Backstage-Bereich ändern

Wenn Sie Änderungen des Backstage-Bereichs in Form einer entsprechenden XML-Definition zusammengestellt haben, wenden Sie diese genau wie eine Ribbon-Anpassung an. Wie dies funktioniert, erfahren Sie unter »Schnellstart« ab Seite 715.

Ereignisse beim Ein- und Ausblenden

Wie Callback-Funktionen grundsätzlich funktionieren, haben Sie ja bereits in »customUI und VBA« auf Seite 734 erfahren.

Wenn Sie per VBA-Prozedur auf das Ein- und Ausblenden des Backstage-Bereichs reagieren möchten, fügen Sie dem Element je nach Bedarf das Attribut *onShow* oder *onHide* hinzu und geben den Namen der aufzurufenden VBA-Prozedur an:

```
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui">
  <backstage onShow="OnShow" onHide="OnHide"/>
</customUI>
```

Fügen Sie in ein Standardmodul wie etwa *mdlRibbon* die beiden folgenden Prozeduren ein:

```
Sub OnShow(contextObject As Object)
    MsgBox "OnShow " & contextObject.Name
End Sub

Sub OnHide(contextObject As Object)
    MsgBox "OnHide " & contextObject.Name
End Sub
```

Dies führt beim Aktivieren beziehungsweise Deaktivieren zur Anzeige der entsprechenden Meldungen. Das mit dem Parameter *contextObject* referenzierte Objekt liefert einen Verweis auf das *Application*-Objekt der jeweiligen Anwendung, in diesem Fall also *Access.Application*.

13.1.2 button- und tab-Elemente

Ganz links im Backstage-Bereich finden Sie zwei Arten von Steuerelementen: Schaltflächen und Registerlaschen. Letztere sehen zwar nicht so aus wie die vom Registersteuerelement bekannten Laschen, aber sie funktionieren genau so. Wo ist der Unterschied? Mit den Schaltflächen lösen Sie schlicht die angegebene Funktion aus, mit einem Klick auf eine Registerlasche zeigen Sie rechts die dafür hinterlegte Registerseite an. Wenn Sie diese beiden Elemente mit der Maus überfahren, hebt Access bei einer Schaltfläche nur den angezeigten Text und gegebenenfalls das Icon durch einen farbigen Hintergrund hervor (siehe Abbildung 13.1), bei einer Registerlasche den kompletten Bereich des Eintrags (siehe Abbildung 13.2).

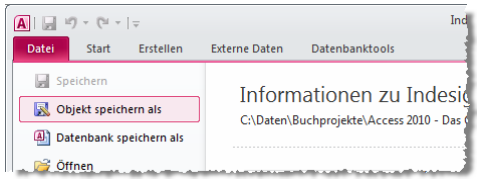


Abbildung 13.1: Schaltfläche für den direkten Aufruf von Funktionen

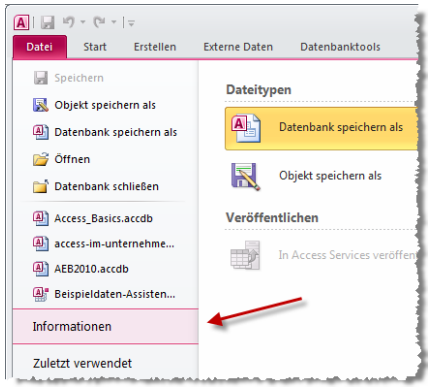


Abbildung 13.2: Registerlasche im Backstage-Bereich

Die XML-Definition des Backstage-Bereichs nehmen Sie grundsätzlich wie bei den übrigen Elementen des *customUI* wie etwa beim Ribbon vor. Sie benötigen eine XML-Datei mit einem *customUI*-Element. Gleich darunter definieren Sie den Backstage-Bereich

mit dem *backstage*-Element und den darin enthaltenen Steuerelementen *button* und *tab*. Die Erweiterung des Backstage-Bereichs um eine Schaltfläche und eine Registerlasche geschieht so:

```
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui">
  <backstage>
    <button id="btn1" label="Button 1"/>
    <tab id="tab1" label="Tab 1" />
  </backstage>
</customUI>
```

Das *button*-Element im linken Backstage-Bereich besitzt einige Attribute weniger als das *button*-Element im Ribbon. Dies liegt beispielsweise daran, dass Sie diese Schaltfläche nicht in verschiedenen Größen anzeigen können. Für ein *tab*-Element im Backstage-Bereich können Sie zwei Beschriftungen festlegen: eine für die Registerlasche (*label*) und eine Überschrift für die Registerseite (*title*).

```
<tab id="tab1" title="Beispieltab (title)" label="Beispieltab (label)">
```

Ein Beispiel finden Sie weiter unten in Abbildung 13.3.

13.1.3 firstColumn und secondColumn: Spalten einer Registerseite

Dem *tab*-Element können Sie wiederum zwei Elemente unterordnen – das *firstColumn*- und das *secondColumn*-Element. Diese Elemente besitzen weder eigene Attribute noch ändern sie das Erscheinungsbild des Backstage-Bereichs, wenn Sie keine untergeordneten Elemente hinzufügen.

Sie können jedoch über die Attribute des übergeordneten *tab*-Elements einige Einstellungen vornehmen – zum Beispiel zum Festlegen der Spaltenbreiten:

columnWidthPercent: Gibt an, welchen Anteil der verfügbaren Breite die linke Spalte einnehmen soll (Zahl von 0 bis 100)

firstColumnMinWidth: Gibt die minimale Breite der linken Spalte in Pixel an.

firstColumnMaxWidth: Gibt die maximale Breite der linken Spalte in Pixel an.

secondColumnMinWidth: Gibt die minimale Breite der rechten Spalte in Pixel an.

secondColumnMaxWidth: Gibt die maximale Breite der rechten Spalte in Pixel an.

Dabei ist Folgendes zu beachten: Wenn die verfügbare Breite größer oder gleich der unter *firstColumnMaxWidth* und *secondColumnMaxWidth* angegebenen Breite ist, werden beide Spalten in der maximalen Breite angezeigt. Ist die gesamte Breite kleiner als die

Summe der beiden Spaltenbreiten, werden diese entsprechend dem unter *columnWidthPercent* angegebenen Verhältnis verkleinert.

Die unter *firstColumnMinWidth* und *secondColumnMinWidth* angegebenen Werte begrenzen die Breite des Access-Fensters nach unten. Ein mit diesen Attributen versehenes *tab*-Element wird beispielsweise so definiert:

```
<tab id="tab1" title="Beispieltab (title)" label="Beispieltab (label)"
columnWidthPercent="40" firstColumnMinWidth="100" firstColumnMaxWidth="300"
secondColumnMinWidth="100" secondColumnMaxWidth="300">
...
</tab>
```

Wichtig ist auch, dass Sie auch ein einspaltiges Layout erzeugen können. Dazu brauchen Sie einfach nur das Element *firstColumn* zu verwenden – das Element *secondColumn* lassen Sie weg.

13.1.4 group

Unterhalb von *firstColumn* und *secondColumn* können Sie weitere Elemente anlegen. Mit einem oder mehreren *group*-Elementen teilen Sie die Spalte in mehrere Bereiche auf.

Die *group*-Elemente werden oben angeordnet und nehmen nur den Platz ein, den die enthaltenen Elemente benötigen.

Mehrere Gruppen in einer Spalte

Im folgenden Beispiel soll die linke Spalte drei Gruppen anzeigen, die rechte eine. Das Attribut *style* legt mit den drei Werten *normal*, *warning* und *error* verschiedene Layouts fest (siehe Abbildung 13.3).

```
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui">
  <backstage>
    <tab id="tab1" title="Beispieltab (title)" label="Beispieltab (label)"
        columnWidthPercent="40" firstColumnMinWidth="100" firstColumnMaxWidth="300"
        secondColumnMinWidth="100" secondColumnMaxWidth="300">
      <firstColumn>
        <group id="grp3" label="Gruppe mit style = warning" style="warning"/>
        <group id="grp2" label="Gruppe mit style = normal" style="normal"/>
        <group id="grp1" label="Gruppe mit style = error" style="error"/>
      </firstColumn>
      <secondColumn>
        <group id="grp4" label="Gruppe ohne style"/>
      </secondColumn>
```

```
      </tab>
    </backstage>
  </customUI>
```

Listing 13.1: Code des Beispiels *Tab mit zwei Spalten und Gruppen*

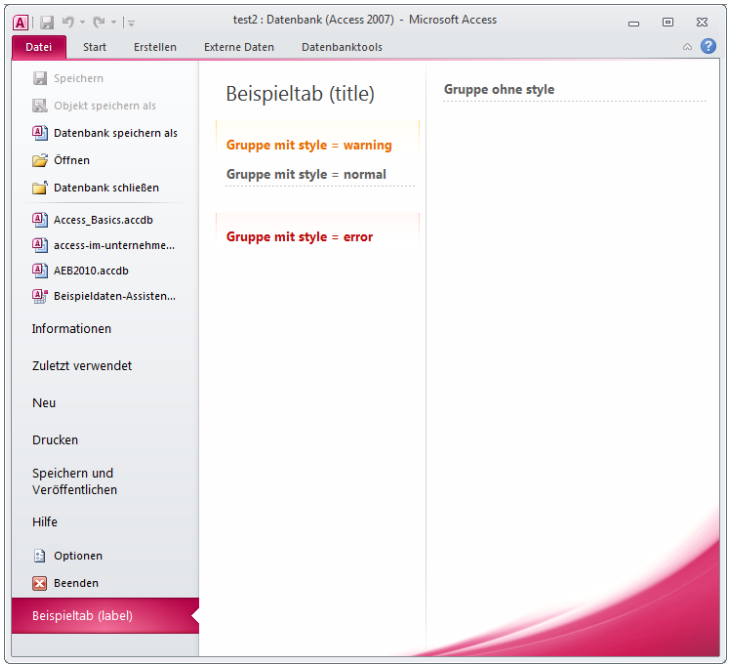


Abbildung 13.3: Backstage-Bereich mit verschiedenen Gruppen

Sie können einer Gruppe außerdem einen weiteren Text mitgeben. Diesen legen Sie mit dem Attribut *helperText* des jeweiligen *group*-Elements fest. Der Text erscheint unterhalb des unter *label* angegebenen Textes (siehe Abbildung 13.4).

Wenn Sie nur einen Text anzeigen möchten, können Sie entweder das Attribut *label* leer lassen oder das Attribut *showLabel* auf *false* einstellen. Den *helperText* geben Sie beispielsweise wie folgt ein:

```
<group id="grp3" label="Gruppe mit style = warning" helperText="Dieser Text steht im
Attribute helperText." style="warning"/>
```

13.1.5 primaryItem, topItems und bottomItems

Unterhalb der *group*-Elemente gibt es wiederum drei Orte, an denen Sie die eigentlichen Steuerelemente unterbringen können.

Abbildung 13.4: *group*-Elemente mit *helperText* (links) und mit *helperText* und ohne *label* (rechts).

primaryItem

Unterhalb von *primaryItem* können Sie nur ein Element unterbringen, und zwar ein *button*- oder ein *menu*-Element:

```
<tab id="tab3" label="Primary-, Top- und Bottom-Items">
  <firstColumn>
    <group id="grp6" label="Gruppe mit primaryItem" style="error">
      <primaryItem>
        <button id="btn8" label="Button im primaryItem"/>
      </primaryItem>
    </group>
  </firstColumn>
</tab>
```

Diese Definition zeigt den Bereich aus Abbildung 13.5 an.

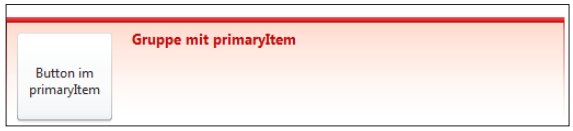


Abbildung 13.5: Ein *button*-Element unterhalb des *primaryItem*-Elements

Statt einer einfachen Schaltfläche können Sie hier auch ein Menü unterbringen, das eines oder mehrere Steuerelemente der Typen *button*, *checkbox*, *toggleButton* oder weitere *menu*-Elemente enthält:

```
<group id="grp10" label="Gruppe mit Menü im primaryItem" style="error">
  <primaryItem>
    <menu id="mnu1" label="Menü">
      <menuGroup id="mnugrp1" label="Menügruppe">
```

```
        <checkBox id="chk1" label="Checkbox im Menü"/>
        <button id="btn11" label="Button im Menü"/>
      </menuGroup>
    </menu>
  </primaryItem>
</group>
```

Der mit dieser Definition festgelegte Bereich sieht wie in Abbildung 13.6 aus. Das *menuGroup*-Element enthält dabei eine Überschrift und kann so mehrere Einträge zusammenfassen.

Abbildung 13.6: Ein Menü unterhalb des *primaryItem*-Elements

Sie können das Label eines *menuGroup*-Elements auch weglassen:

```
<menuGroup id="mnugrp2">
  <button id="btn12" label="Button in Menügruppe ohne Label"/>
</menuGroup>
```

Statt einer farbig hinterlegten Überschrift zeigt der Backstage-Bereich dann eine gestrichelte Trennlinie im Menü an (Abbildung 13.7).

Abbildung 13.7: Menüunterteilung ohne Beschriftung

topItems

Wie die Bezeichnung *topItems* gegenüber *primaryItem* schon sagt, lassen sich hier mehrere Steuerelemente unterbringen und nicht nur eines. Die unterhalb des *topItems*-Elements untergebrachten Steuerelemente erscheinen unter dem *label*- und gegebenenfalls dem *helperText*-Element. Wenn gleichzeitig ein *primaryItem*-Element vorhanden

ist, werden nicht nur *label* und *helperText* der Gruppe, sondern auch die unter *topItems* angegebenen Steuerelemente entsprechend nach rechts verschoben.

Im *topItems*-Bereich können Sie die folgenden Elemente unterbringen: *button*, *checkBox*, *comboBox*, *dropDown*, *editBox*, *groupBox*, *hyperlink*, *imageControl*, *labelControl*, *layoutContainer* und *radioGroup*. Einige dieser Elemente zeigt Abbildung 13.8, einige sind in Access 2010 neu hinzugekommen. Die Beschreibung dieser Elemente und weitere Beispiele finden Sie weiter unten.

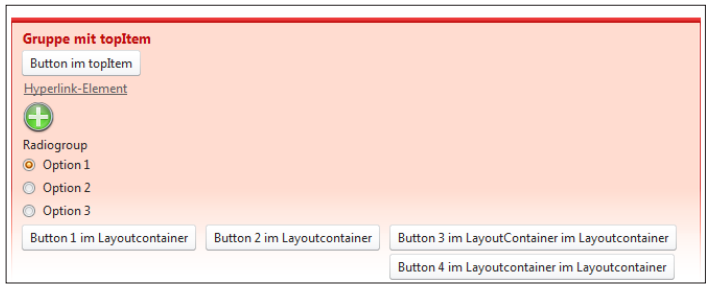


Abbildung 13.8: Steuerelemente im topItems-Bereich

bottomItems

Die unterhalb des Elements *bottomItems* befindlichen Elemente werden linksbündig unter den anderen Elementen angezeigt (siehe Abbildung 13.9):

```
<group id="grp9" label="Gruppe mit Primary- und BottomItem" style="warning">
  <primaryItem>
    <button id="btn21" label="Button im PrimaryItem"/>
  </primaryItem>
  <bottomItems>
    <button id="btn20" label="Button im BottomItem"/>
  </bottomItems>
</group>
```

Abbildung 13.9: *primaryItem*- und *bottomItems*-Elemente mit Schaltflächen

Kommt auch noch ein *topItems*-Bereich hinzu, sieht der Aufbau wie in Abbildung 13.10 aus:

```
<group id="id31544" label="Gruppe mit Primary-, Top- und BottomItem" style="warning">
  <primaryItem>
    <button id="id31548" label="Button im PrimaryItem"/>
  </primaryItem>
  <topItems>
    <button id="btn22" label="Button im TopItem"/>
    <button id="btn23" label="Noch ein Button ..."/>
    <button id="btn25" label="… und noch einer"/>
  </topItems>
  <bottomItems>
    <button id="id31546" label="Button im BottomItem"/>
  </bottomItems>
</group>
```

Abbildung 13.10: *primaryItem*-, *topItems*- und *bottomItems*-Elemente in einer Gruppe

13.1.6 taskGroup

Neben den *group*-Elementen können Sie einer Spalte auch ein *taskGroup*-Element hinzufügen. Im Gegensatz zum *group*-Element enthält dieses Element kein *helperText*-Attribut. Ein gutes Beispiel für ein *taskGroup*-Element ist die linke Spalte der Registerseite *Hilfe*. Es gibt eine Überschrift für mehrere Tasks, hier Support, und die eigentlichen Aufgaben mit Bild, Überschrift und Beschreibung (siehe Abbildung 13.11).

Das *taskGroup*-Element ist wesentlich weniger flexibel als das *group*-Element. Das folgende Beispiel zeigt den Aufbau:

```
<taskGroup id="tg1" label="Task Group" helperText="HelperText dieser Taskgroup">
  <category id="kat1" label="Kategorie 1">
    <task id="tas1" image="add_128" label="Task 1"
      description="Dies ist die Beschreibung einer Aufgabe im Bereich taskGroup."/>
    <task id="tas2" image="bug_red_128" label="Task 2"/>
  </category>
  <category id="kat2" label="Kategorie 2">
    <task id="tas3" image="businesspeople_128" label="Task 3"/>
  </category>
</taskGroup>
```

```
<taskGroup id="tg2" label="Task Group 2">
  <category id="kat3" label="Kategorie 3">
    <task id="tas4" image="yinyang_128" label="Task 4" description="Noch ein
      interessanter Task mit einer noch viel interessanteren Beschreibung."/>
  </category>
</taskGroup>
```

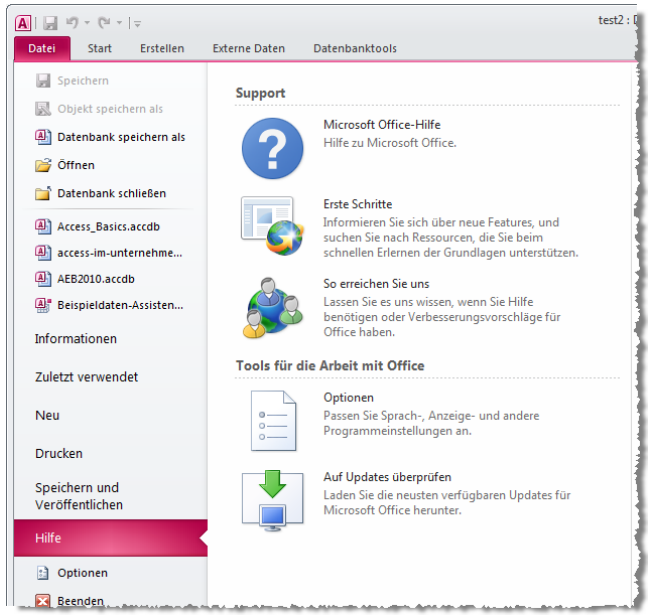


Abbildung 13.11: Beispiel für ein eingebautes *taskGroup*-Element

Das Beispiel besteht aus zwei *taskGroup*-Elementen. Die wesentlichen Attribute sind *label* und *helperText*. Die *taskGroup*-Elemente erhalten unabhängig von ihrer Höhe jeweils den gleichen Platz zugewiesen. Das heißt, dass ein Element mit drei Aufgaben genau die gleiche Höhe einnimmt wie eines mit nur einer Aufgabe (siehe auch Abbildung 13.12).

Jedes *taskGroup*-Element enthält ein oder mehrere *category*-Elemente. Hier ist zunächst nur das Attribut *label* interessant. Innerhalb der *category*-Elemente stecken die eigentlichen *task*-Elemente. Diese bieten ebenfalls zwei Möglichkeiten, Texte unterzubringen – im *label*- und im *description*-Attribut.

Außerdem können Sie für jedes *task*-Element ein eingebautes oder benutzerdefiniertes Bild angeben. Sollten Sie nur wenige *task*-Elemente anzeigen, stellen Sie schnell fest, dass die im Ribbon verwendeten Bilder von 32 x 32 Pixel zu klein sind und hochskaliert werden, was nicht schön aussieht. Wenn Sie die Höhe des Access-Fensters verringern, verkleinert Access auch die Bilder. Und hier kommen wir nochmals auf das Element

taskGroup zurück. Dieses enthält ein Attribut namens *allowedTaskSizes*, das folgende Werte annehmen kann: *large*, *largeMedium*, *largeMediumSmall*, *medium*, *mediumSmall* und *small*. Sie können also alle Kombinationen aus *small*, *medium* und *large* festlegen, standardmäßig verwendet Access hier *largeMediumSmall*.

Sollten Ihre Icons nur 32 x 32 Pixel groß sein, stellen Sie also *allowedTaskSizes* auf *mediumSmall* ein – es werden dann keine großen Icons produziert.

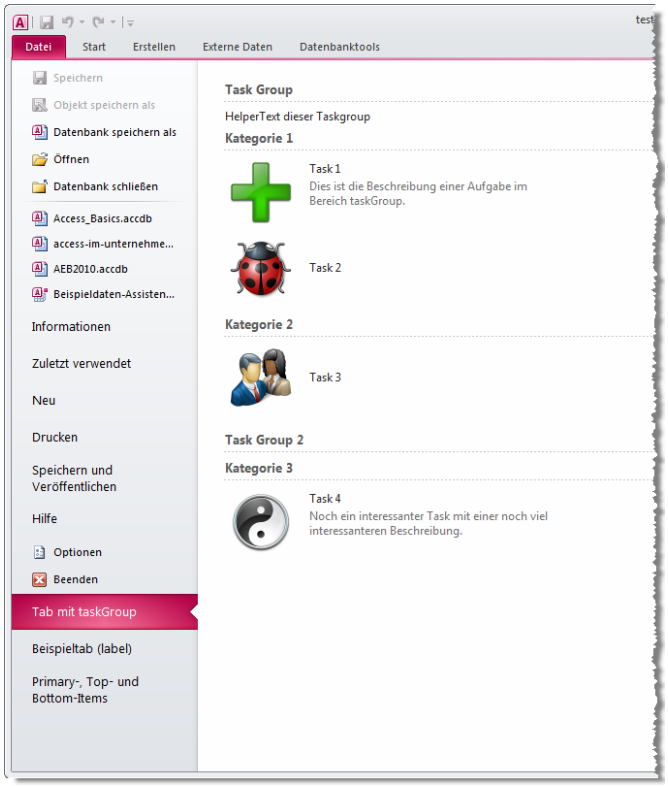


Abbildung 13.12: Benutzerdefiniertes Beispiel für *taskGroup*-Elemente

13.1.7 taskFormGroup

Das *taskFormGroup*-Element ist eine Kombination aus *taskGroup* und *group*. Während das *group*-Element beliebig viele Steuerelemente enthalten kann, zeigt das *taskGroup*-Element *category*- und *task*-Elemente wie oben beschrieben an.

Im *taskFormGroup*-Element werden die *task*-Elemente prinzipiell als weitere Ebene von Registerlaschen ausgeführt, wobei jeder *task* ein eigenes Registerblatt öffnen kann. Abbildung 13.13 zeigt, wie dies aussieht.

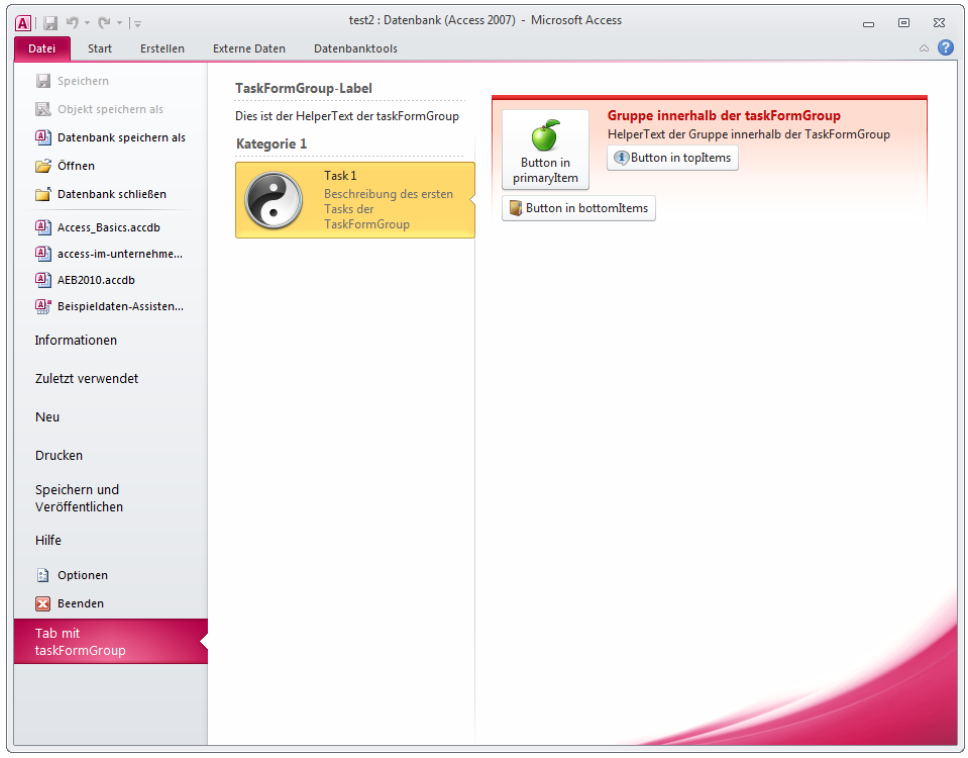


Abbildung 13.13: Die *taskFormGroup* kombiniert *task*-Elemente mit *group*-Elementen

Der Teil der XML-Definition für die linke Spalte sieht wie folgt aus. Das *taskFormGroup*-Element enthält wie einige bereits vorgestellte Elemente ein *label*- und ein *helperText*-Attribut zur Anzeige von Text. Darunter legen Sie eines oder mehrere *category*-Elemente an, die Sie mit dem *label*-Attribut beschreiben können. Jedem *category*-Element fügen Sie ein oder mehrere *task*-Elemente hinzu (diesmal mit *label*- und *description*-Attribut für Beschriftungen). Und jedes *task*-Element enthält ein oder mehrere *group*-Elemente, die genau wie die direkt unterhalb der *firstColumn*- und *secondColumn*-Elemente angeordneten Elemente mit Steuerelementen gefüllt werden können. Ein Klick auf ein task-Element öffnet also quasi ein eigenes Unterformular:

```
<firstColumn>
  <taskFormGroup id="tfg1" label="TaskFormGroup-Label" helperText="Dies ist der
      HelperText der taskFormGroup">
    <category id="kat1" label="Kategorie 1">
      <task id="task1" image="yinyang_128" label="Task 1" description="Beschreibung
          des ersten Tasks der TaskFormGroup">
        <group id="grp1" label="Gruppe innerhalb der taskFormGroup"
```

```
            helperText="HelperText der Gruppe innerhalb der TaskFormGroup" style="error">
            <primaryItem>
              <button id="btn2" image="apple" label="Button in primaryItem"/>
            </primaryItem>
            <topItems>
              <button id="btn1" image="about_32" label="Button in topItems"/>
            </topItems>
            <bottomItems>
              <button id="btn3" image="address_book3" label="Button in bottomItems"/>
            </bottomItems>
          </group>
        </task>
      </category>
    </taskFormGroup>
  </firstColumn>
```

Einschränkung des taskFormGroup-Elements

Dieses Element können Sie nur in der ersten Spalte verwenden, also innerhalb des *firstColumn*-Elements. Wenn Sie das *taskFormGroup*-Element in der ersten Spalte einsetzen, können Sie kein *group*- und kein *taskGroup*-Element auf der gleichen Ebene hinzufügen. Außerdem dürfen Sie nur ein taskFormGroup-Element definieren.

13.2 group-Elemente mit Steuerelementen füllen

Mit den Elementen *firstColumn*, *secondColumn*, *primaryItem, topItems, bottomItems, group*, *taskGroup* und *taskFormGroup*-Element haben Sie die für die grobe Struktur verantwortlichen Elemente im Backstage-Bereich kennen gelernt. Zeit, sich den eigentlichen Steuerelementen zuzuwenden, die Sie darin unterbringen können.

13.2.1 button-Element

Das *button*-Element im Backstage-Bereich unterscheidet sich an manchen Stellen von dem im Ribbon. Den ersten Unterschied macht die Eigenschaft *isDefinitive* aus, mit der Sie festlegen, ob ein button den Backstage-Bereich schließen kann, der zweite Unterschied liegt in der Gestaltung der Schaltfläche.

Backstage-Bereich schließen

Lässt sich der Backstage-Bereich eigentlich schließen, ohne dass Sie auf ein anderes Tab klicken? Ja. Allerdings erledigen Sie dies nicht durch den Aufruf einer Callback-

Funktion und das Ausführen einer entsprechenden Anweisung, sondern durch das Zuweisen des Werts *true* für das Attribut *isDefinitive* eines *button*- oder *task*-Elements.

Für eine Schaltfläche, die wie in Abbildung 13.14 den Backstage-Bereich schließt, benötigen Sie die folgende Definition:

```
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui"
loadImage="loadImage">
  <backstage>
    <button id="btnCloseBackstage" image="exit_32" label="Backstage schließen"
      isDefinitive="true"/>
  </backstage>
</customUI>
```

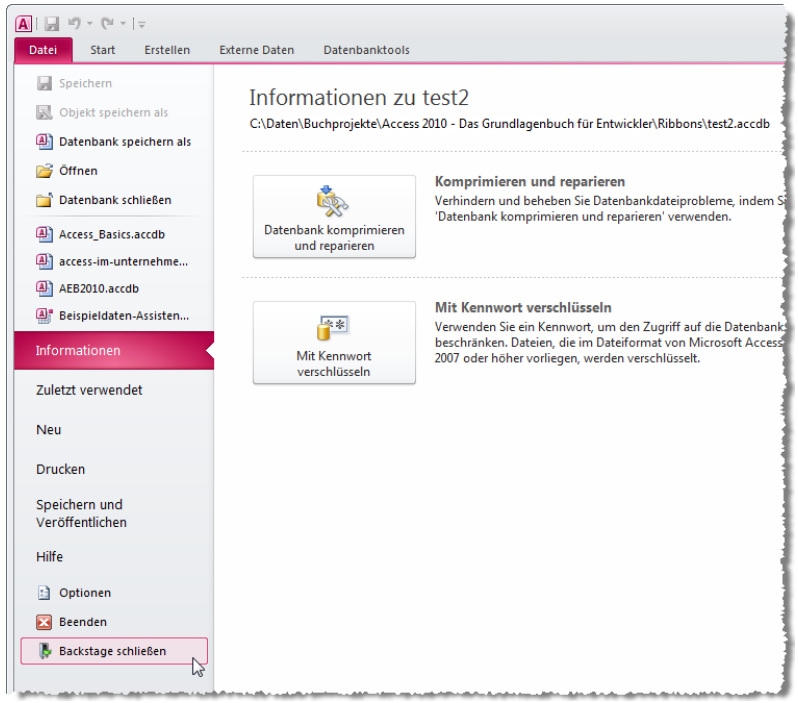


Abbildung 13.14: Diese Schaltfläche schließt den Backstage-Bereich

Aussehen des button-Elements im Backstage-Bereich

Für das *button*-Element gibt es außerdem zusätzlich das Attribut *style*, das die Werte *normal*, *large* und *borderless* annehmen kann. Das folgende Beispiel ordnet mit dem *layoutContainer*-Element einige *button*-Elemente mit verschiedenen Werten für das *style*-Attribut sowie mit und ohne Bilder an:

```
<layoutContainer id="lc1">
  <layoutContainer id="lc2" layoutChildren="vertical">
    <button id="btn1" label="Button normal" style="normal"/>
    <button id="btn2" label="Button large" style="large"/>
    <button id="btn3" label="Button borderless" style="borderless"/>
  </layoutContainer>
  <layoutContainer id="lc3" layoutChildren="vertical">
    <button id="btn4" image="apple" label="Button normal" style="normal"/>
    <button id="btn5" image="banana_32" label="Button large" style="large"/>
    <button id="btn6" image="pineapple_32" label="Button borderless"
style="borderless"/>
  </layoutContainer>
</layoutContainer>
```

Abbildung 13.15 zeigt das Ergebnis dieser XML-Definition.

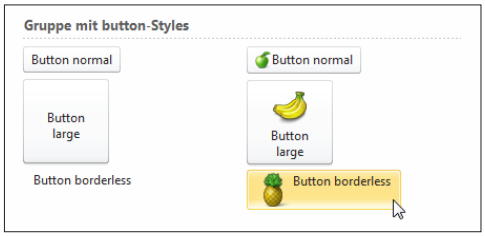


Abbildung 13.15: Schaltflächen mit und ohne Bild und in verschiedenen Layouts

Das expand-Attribut

Mit dem *expand*-Attribut legen Sie fest, ob Steuerelemente wie das *button*-Element über die Höhe oder Breite des jeweiligen Bereichs gestreckt werden. Das *expand*-Attribut kann die Werte *horizontal*, *vertical*, *both* oder *neither* annehmen.

Zum Beispiel aus Abbildung 13.16 gehört der folgende XML-Code, wobei auch hier das nachfolgend erläuterte *layoutContainer*-Element zum Einsatz kam:

```
<topItems>
  <layoutContainer id="lc1" expand="both" layoutChildren="horizontal">
    <layoutContainer id="lc2" expand="both" layoutChildren="vertical">
      <button id="btn5" label="&lt;button&gt;"/>
      <button id="btn6" label="&lt;button&gt;"/>
    </layoutContainer>
    <layoutContainer id="lc3" expand="both" layoutChildren="horizontal">
      <button id="btn2" label="expand = vertical" expand="vertical"/>
      <button id="btn3" label="expand = both" expand="both"/>
```

```
      </layoutContainer>
    </layoutContainer>
    <button id="btn1" label="expand = horizontal" expand="horizontal"/>
    <button id="btn4" label="expand = neither" expand="neither"/>
  </topItems>
```

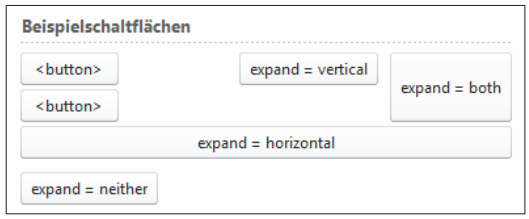


Abbildung 13.16: Steuerelemente mit verschiedenen Einstellungen für das *expand*-Attribut

13.2.2 layoutContainer-Element

Das *layoutContainer*-Element strukturiert die Elemente des Backstage-Bereichs. Sie können *layoutContainer*-Elemente ineinander verschachteln und so beliebige Anordnungen erzeugen. Das Element ist unsichtbar und besitzt die folgenden wichtigen Attribute:

» *align*: Legt fest, wie die enthaltenen Elemente im *layoutContainer* ausgerichtet werden (*center*, *left*, *right*, *top*, *bottom*, *topLeft*, *topRight*, *bottomLeft*, *bottomRight*)

» *expand*: Legt fest, ob sich das Element ausdehnen soll (*horizontal*, *vertical*, *both*, *neither*)

» *layoutChildren*: Legt fest, wie die enthaltenen Elemente im *layoutContainer* angeordnet werden (*vertical*, *horizontal*)

Beispiele zum *layoutContainer* finden Sie in den vorangegangenen und folgenden Abschnitten.

13.2.3 groupBox-Element

Das *groupBox*-Element ist der kleine Bruder des *layoutContainer*-Elements. Es dient der einfachen Zusammenfassung von Steuerelementen. Sie können damit zwar keine Ausrichtung festlegen, dafür aber eine Beschriftung anbringen:

```
<groupBox id="grpbox1" label="Groupbox 1" expand="both">
  <button id="btn2" label="Button 2"/>
  <button id="btn3" image="add_32" label="Button 3"/>
</groupBox>
```

Abbildung 13.17 zeigt ein Beispiel für ein *groupBox*-Element.

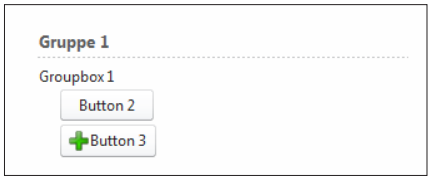


Abbildung 13.17: Zwei Schaltflächen in einem *groupBox*-Element

13.2.4 hyperlink-Element

Das *hyperLink*-Element bietet die Möglichkeit, gleich über einen Klick auf das entsprechende Element die dafür hinterlegte Webseite zu öffnen. Das wichtigste Attribut heißt *target* und nimmt eine Zeichenkette mit der Internetadresse auf, mit dem Callback-Attribut *getTarget* lässt sich diese dynamisch einstellen. Die Beschriftung dieses Elements können Sie wie gewohnt mit der *label*-Eigenschaft angeben.

hyperlink-Element anklicken

Wenn der Benutzer das *hyperlink*-Element anklickt, kann er damit eine von zwei Aktionen auslösen:

» Anzeigen der unter *target* angegebenen Internetadresse

» Ausführen der unter *onAction* angegebenen Callback-Funktion

Sie können nur eines der beiden Attribute *target* und *onAction* verwenden. Das folgende Beispiel verwendet das *target*-Attribut:

```
<hyperlink label="Access-Entwicklerbuch" id="hyp1" target="http://www.access-entwick-
lerbuch.de/2010"/>
```

13.2.5 imageControl-Element

Während Sie im Ribbon nur Schaltflächen, Gallerien und ähnliche Steuerelemente mit Bildern versehen können, ist dies im Backstage-Bereich auch mit einem reinen Bildsteuerelement möglich.

Das Bild wird wie unter »Bilder im customUI« auf Seite 740 beschrieben über das entsprechende Callback-Attribut *image* beziehungsweise *imageMso* zugewiesen (alternativ können Sie das Callback-Attribut *getImage* verwenden – siehe »Bilder im customUI« ab Seite 740):

```
<imageControl image="banana_32" id="img1"/>
```

13.2.6 radioGroup-Element

Das *radioGroup*-Element entspricht der Optionsgruppe in Access-Formularen. Es besitzt ein Unterelement namens *radioButton*, dem Sie mit *label* eine Beschriftung und mit *onAction* eine Ereignisprozedur zuweisen können. Die folgende Definition erzeugt beispielsweise eine Optionsgruppe wie in Abbildung 13.18.

```
<topItems>
  <radioGroup label="Beispielradiogroup" id="rad1">
    <radioButton label="radioButton 1" id="rbtn1"/>
    <radioButton label="radioButton 2" id="rbtn2"/>
    <radioButton label="radioButton 3" id="rbtn3"/>
  </radioGroup>
</topItems>
```

Abbildung 13.18: Beispiel für ein *radioGroup*-Element samt *radioButtons*

Solche Elemente können Sie natürlich auch per Callback-Funktion füllen. Dazu stellen Sie die Attribute *getItemCount*, *getItemLabel* und *getItemID* auf die Namen der entsprechenden Callback-Funktionen ein.

Außerdem legen Sie auch noch eine *onAction*-Ereignisprozedur fest:

```
<radioGroup onAction="onAction" label="Dynamische Radiogroup"
getItemCount="getItemCount" getItemLabel="getItemLabel" getItemID="getItemID"
id="rad2"/>
```

Die Vorgehensweise ist mit der beim Füllen eines *dropDown*-Elements identisch. Zunächst benötigen Sie eine Variable zum Aufnehmen der einzelnen Einträge:

```
Dim strRadiogroup() As String
```

Als Erstes wird die Callback-Funktion *getItemCount* definiert. Sie liefert einen Verweis auf das auslösende Steuerelement und erwartet die Anzahl der anzuzeigenden Elemente als Rückgabewert.

Diese Prozedur übernimmt bereits die Hauptarbeit. Sie durchläuft eine Tabelle namens *tblRadiogroupElements*, die wie in Abbildung 13.19 aussieht.

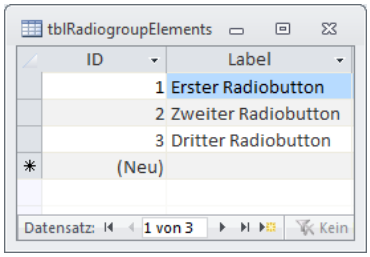


Abbildung 13.19: Tabelle als Datenherkunft für eine Optionsgruppe im Backstage-Bereich

Beim Durchlaufen der Datensätze schreibt die folgende Prozedur jeweils die ID und die Beschriftung in das zweidimensionale Array *strRadiogroup()*. Dieses ist extra global deklariert, damit die nachfolgend aufgerufenen Prozeduren darauf zugreifen können:

```
Sub getItemCount(control As IRibbonControl, ByRef returnedVal)
    Dim db As DAO.Database
    Dim rst As DAO.Recordset
    Dim i As Integer
    Set db = CurrentDb
    Set rst = db.OpenRecordset("tblRadiogroupElements")
    Do While Not rst.EOF
        i = i + 1
        ReDim Preserve strRadiogroup(1, i + 1)
        strRadiogroup(0, i - 1) = rst!id
        strRadiogroup(1, i - 1) = rst!label
        rst.MoveNext
    Loop
    returnedVal = i
End Sub
```

Anschließend ruft Access entsprechend der im Rückgabewert übergebenen Anzahl die Prozeduren *getItemLabel* und *getItemID* auf

Diese lesen jeweils die ID und die Beschriftung aus dem Array *strRadiogroup* ein, und zwar entsprechend dem mit *itemIndex* übergebenen Indexwert:

```
Sub getItemLabel(control As IRibbonControl, itemIndex As Integer, ByRef label)
    label = strRadiogroup(1, itemIndex)
End Sub

Sub getItemID(control As IRibbonControl, itemIndex As Integer, ByRef id)
    id = strRadiogroup(0, itemIndex)
End Sub
```

Wollen Sie das Ereignis abfangen, wenn der Benutzer einen der Einträge anklickt, benötigen Sie das Attribut *onAction* sowie eine entsprechende Callback-Funktion. Diese hat in diesem Fall drei Parameter: *control* liefert einen Verweis auf das auslösende Steuerelement, *id* den als ID gespeicherten Zahlenwert (in unserem Fall den Primärschlüsselwert der Tabelle) und in *index* den Index des ausgewählten Eintrags:

```
Sub onActionRadioGroup(control As IRibbonControl, id As String, index As Integer)
    MsgBox "Sie haben den Eintrag '" & id & "' ausgewählt."
End Sub
```

Wollen Sie im Meldungsfenster wie im Beispiel noch den Text des angeklickten Eintrags ausgeben, müssen Sie ihn über eine entsprechende Datenbankabfrage wie *DLookup("Label", "tblRadiogroupElements", "ID = " & id)* ermitteln. Über die Parameter ist dieser Wert nicht direkt abfragbar.

13.2.7 checkBox-, comboBox-, dropDown-, editBox-, imageControl- und labelControl-Element

Diese Elemente verhalten sich im Backstage-Bereich in der Regel wie ihre Pendants im Ribbon. Sie bieten jedoch meist nicht alle Attribute dieser Elemente und somit weniger Möglichkeiten.

13.3 Eingebaute Backstage-Elemente

Wer eine eigene Anwendung programmiert, möchte gegebenenfalls die eingebauten Backstage-Elemente ausblenden oder anpassen. Die folgenden Abschnitte stellen die notwendigen Techniken vor.

13.3.1 Eingebaute Backstage-Elemente ausblenden

Um die eingebauten Elemente der ersten Ebene des Backstage-Bereichs (also die links sichtbaren Elemente) auszublenden, benötigen Sie den Typ (*button/tab*) und die *idMso* des jeweiligen Steuerelements. Diese liefert die bereits oben erwähnte Excel-Tabelle *AccessControls.xlsx*. Die folgende XML-Definition des Backstage-Bereichs blendet alle Elemente außer den zuletzt geöffneten Elementen aus (siehe Abbildung 13.20):

```
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui">
  <backstage>
    <button idMso="FileSave" visible="false"/>
    <button idMso="ApplicationOptionsDialog" visible="false"/>
    <button idMso="SaveObjectAs" visible="false"/>
```

```
    <button idMso="FileCloseDatabase" visible="false"/>
    <button idMso="FileExit" visible="false"/>
    <button idMso="FileOpen" visible="false"/>
    <button idMso="FileSaveAsCurrentFileFormat" visible="false"/>
    <tab idMso="TabHelp" visible="false"/>
    <tab idMso="TabInfo" visible="false"/>
    <tab idMso="TabRecent" visible="false"/>
    <tab idMso="TabNew" visible="false"/>
    <tab idMso="TabPrint" visible="false"/>
    <tab idMso="TabShare" visible="false"/>
  </backstage>
</customUI>
```

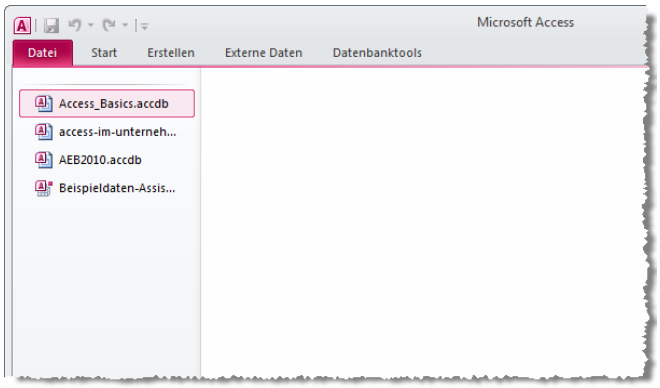


Abbildung 13.20: Ausgedünnter Backstage-Bereich

Fehlen nur noch die zuletzt verwendeten Dateien im Backstage-Bereich. Deren Anzahl stellen Sie über die Eigenschaft aus Abbildung 13.21 ein. Leider gibt es keine Access-Option, die Sie direkt per VBA einstellen könnten. Die Einstellung befindet sich nämlich in der Registry, und hier ist ein wenig mehr Aufwand nötig. Zunächst benötigen Sie die folgenden drei API-Funktionen:

```
Declare Function RegOpenKeyEx Lib "advapi32.dll" Alias "RegOpenKeyExA" ( _
    ByVal hKey As Long, ByVal lpSubKey As String, ByVal ulOptions As Long, _
    ByVal samDesired As Long, phkResult As Long) As Long

Declare Function RegSetValueEx Lib "advapi32.dll" Alias "RegSetValueExA" ( _
    ByVal hKey As Long, ByVal lpValueName As String, ByVal Reserved As Long, _
    ByVal dwType As Long, lpData As Any, ByVal cbData As Long) As Long

Declare Function RegCloseKey Lib "advapi32.dll" (ByVal hKey As Long) As Long
```

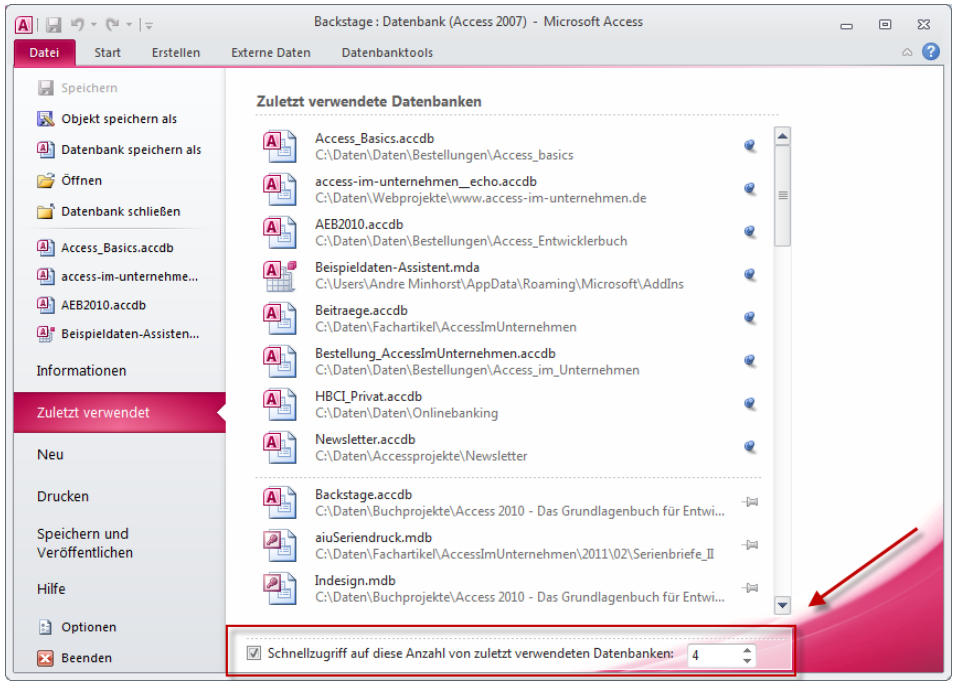


Abbildung 13.21: Einstellen der zuletzt verwendeten Dateien im Schnellzugriff

Außerdem sind diese Konstanten erforderlich:

```
Private Const HKEY_CURRENT_USER = &H80000001
Private Const KEY_SET_VALUE = &H2
Private Const ERROR_SUCCESS = &H0
Private Const REG_DWORD = 4
```

Die eigentliche Arbeit erledigt schließlich die folgende VBA-Funktion, der Sie entweder den Wert *True* (Dateien einblenden) oder *False* (Dateien ausblenden) übergeben. Beachten Sie, dass Sie die Einstellung nach Verwendung einer von Ihnen programmierten Anwendung wieder rückgängig machen sollten, damit der Benutzer seine gewohnte Benutzeroberfläche vorfindet:

```
Public Function QuickAccessDisplay(bol As Boolean)
    Dim kHnd As LongPtr
    Dim lngValue As Long
    Dim lngRtn As Long
    lngValue = Abs(bol)
    Const strSubKey = "Software\Microsoft\Office\14.0\Access\File MRU"
    Const strName = "Quick Access Display"
```

```
    lngRtn = RegOpenKeyEx(HKEY_CURRENT_USER, strSubKey, 0, KEY_SET_VALUE, kHnd)
    If lngRtn = ERROR_SUCCESS Then
        RegSetValueEx kHnd, strName, 0, REG_DWORD, lngValue, Len(lngValue)
    End If
    RegCloseKey kHnd
End Function
```

Wohin aber nun mit dem Aufruf dieser Funktion? Wer sich geistig gerade so richtig in den Backstage-Bereich hineingedacht hat, tendiert vielleicht dazu, dass diese Funktion etwa beim Laden der *customUI*-Definition oder beim Anzeigen des Backstage-Bereichs aufgerufen werden muss. Letzteres ist bereits zu spät, denn damit die mit *QuickAccessDisplay* vorgenommene Einstellung wirkt, muss der Backstage-Bereich neu angezeigt werden. Da Sie den Backstage-Bereich aber vermutlich ohnehin für den ganzen Zeitraum anpassen möchten, in dem der Benutzer mit Ihrer Anwendung arbeitet, können Sie die Anpassung auch gleich beim Öffnen der Anwendung vornehmen und diese beim Schließen wieder rückgängig machen. Den Aufruf von *QuickAccessDisplay* können Sie natürlich auch in der *onLoad*-Prozedur des *customUI*-Elements unterbringen.

13.3.2 Eingebaute Backstage-Elemente erweitern

Wenn Sie dem Backstage-Bereich ein Element hinzufügen möchten, bilden Sie die Struktur bis zu diesem Element nach und referenzieren dabei die eingebauten Elemente über XML-Elemente mit den entsprechenden Werten für das *idMso*-Attribut. Die Werte für das *idMso*-Attribut erhalten Sie dabei aus der im Download verfügbaren und von Microsoft veröffentlichten Excel-Tabelle *AccessControls.xlsx*, die Werte für die Elemente des Backstage-Bereichs befinden sich ganz unten. Wenn Sie beispielsweise unter *Informationen* eine neue Gruppe in der linken Spalte anlegen möchten, benötigen Sie eine XML-Definition wie die folgende:

```
<tab idMso="TabInfo">
  <firstColumn>
    <group id="grp1" label="Anwendungsinformationen">
      <topItems>
        <labelControl id="lbl1" label="Access 2010 - Das Grundlagenbuch für
          Entwickler"/>
        <labelControl id="lbl2" label="Beispielanwendung zum Kapitel
          &quot;Backstage&quot;"/>
        <hyperlink id="hyp1" label="www.access-entwicklerbuch.de"
          target="http://www.access-entwicklerbuch.de/2010"/>
      </topItems>
    </group>
```

```
    </firstColumn>
  </tab>
```

Das Ergebnis können Sie in Abbildung 13.22 betrachten.

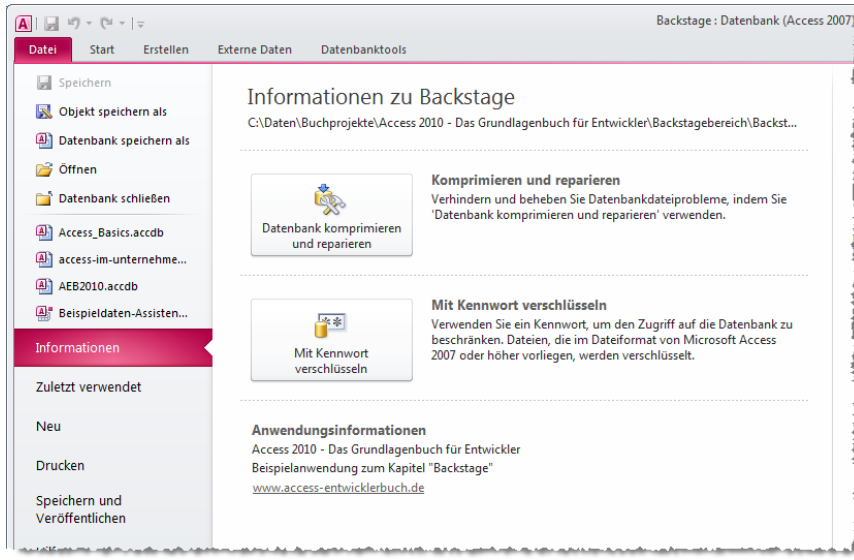


Abbildung 13.22: Ein Hinweistext in einer eingebauten Gruppe im Backstage-Bereich

Während dieses Ergebnis den Erwartungen entspricht, sieht dies im Bereich *Hilfe* ganz anders aus: Dort landet der gleiche Eintrag ganz unten in der linken Spalte des Backstage-Bereichs (siehe Abbildung 13.23). Dies liegt aber schlicht daran, dass *group*-Elemente immer ganz unten angeordnet werden, wenn sich in der gleichen Spalte bereits ein *taskGroup*-Element befindet – was hier der Fall ist. Wenn das *group*-Element oben und das *taskGroup*-Element darunter angelegt wird, werden beide oben ausgerichtet.

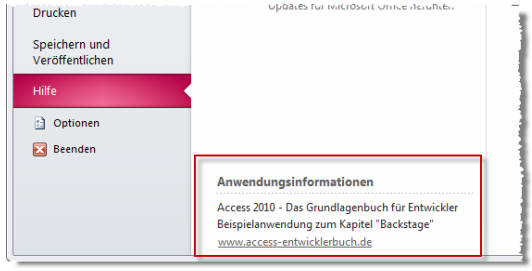


Abbildung 13.23: Eine Gruppe kann auch mal ganz unten im Backstage-Bereich landen