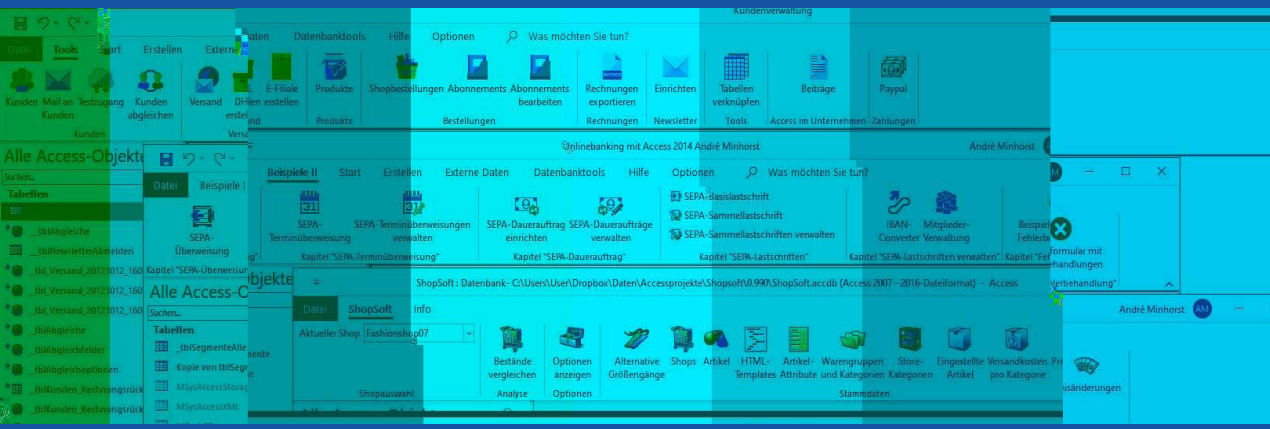
RIBBON PROGRAMMIERUNG



PROFESSIONELLE MENÜS FÜR ACCESS-ANWENDUNGEN PROGRAMMIEREN

ANDRÉ MINHORST

André Minhorst Verlag

André Minhorst

Ribbon Programmierung

PROFESSIONELLE MENÜS FÜR ACCESS-ANWENDUNGEN PROGRAMMIEREN

André Minhorst – Ribbon Programmierung

ISBN 978-3-944216-11-9

1. Auflage 2022

Lektorat André Minhorst
Cover/Titelbild André Minhorst
Typographie, Layout und Satz André Minhorst
Herstellung André Minhorst
Druck und Bindung Kösel, Krugzell (www.koeselbuch.de)

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.

Zeit ohne Zeit.

Inhalt

Inhalt

Vorwort

Anders als in den übrigen Office-Anwendungen wie Word, Excel oder Outlook konsumieren wir als Access-Entwickler nicht nur das Ribbon, sondern wir wollen es auch anpassen, um unsere eigene Arbeit zu optimieren und auch um Anwendungen für Kunden ergonomischer zu gestalten. Dazu gibt es unter Access sehr viele Möglichkeiten: Sie können das Ribbon mit Bordmitteln anpassen, Sie können eigene Ribbondefinitionen für individuelle Anwendungen anlegen oder Sie fügen Elemente über COM-Add-Ins hinzu, die Access um selbst entwickelte Funktionen erweitern.

Dieses Buch zeigt, wie Sie die verschiedenen Möglichkeiten zur Anpassung nutzen können. Dabei schauen wir uns als Erstes an, wie Sie die Bordmitteln von Access nutzen, die Sie übrigens auch in allen anderen Office-Anwendungen gleichermaßen nutzen können. Mit Bordmitteln meinen wir den Bereich Menüband anpassen des Optionen-Dialogs von Access. Hier können Sie bestehende Ribbons und auch die Schnellzugriffsleiste manipulieren. Und Sie können auch Änderungen vornehmen, die mit der aktuell geöffneten Access-Anwendung gespeichert werden.

Während diese Möglichkeiten beschränkt sind, gibt es noch eine weitere Methode zur Anpassung des Ribbons. Diese greift nicht nur für die Ribbonleiste und die Schnellzugriffsleiste, sondern Sie können damit auch den Backstagebereich anpassen – das ist der Bereich, der erscheint, wenn Sie auf den Registerreiter Datei des Ribbons klicken. Um diese Anpassungen vorzunehmen, müssen Sie eine spezielle Tabelle anlegen und die Ribbondefinition im XML-Format formulieren. Hierzu finden Sie alle notwendigen Informationen – sowohl, was den Aufbau der Struktur angeht als auch die Beschreibung der verschiedenen Steuerelemente.

Da Microsoft keinen eigenen Editor zum Bearbeiten des Ribbons zur Verfügung stellt, haben wir selbst einen programmiert – den Ribbon-Admin. Diesen finden Sie als Testversion in den Downloads zu diesem Buch. Sie können die Testversion bereits zum Erstellen einfacher Ribbons nutzen.

Schließlich stellen wir noch ein weiteres Tool vor – nämlich twinBASIC, mit dem Sie auf einfachste Art und Weise COM-Add-Ins erstellen können. COM-Add-Ins sind Add-Ins, mit denen Sie beispielsweise die Funktionen von Access erweitern können und deren Funktionen üblicherweise über Ribbonbefehle gesteuert werden. Wir zeigen im Buch, wie Sie Teile des Dialogs mit den Access-Optionen ins Ribbon verlagern und die Optionen von dort aus steuern können.

Duisburg, 28. Februar 2022

André Minhorst

1 Grundlagen zum Ribbon

Das Ribbon ist der Oberbegriff für die Elemente der Benutzeroberfläche, die wir in diesem Buch beschreiben. Später werden Sie sehen, wie Sie das Ribbon per XML anpassen können. Dort lernen Sie, dass mit dem dortigen *ribbon*-Element nur die tab-Element und deren untergeordnete Elemente bezeichnet werden. Schon der Bereich, den Sie mit einem Klick auf den Registerreiter Datei öffnen, hat beispielsweise eine eigene Bezeichnung, nämlich *backstage*.

Die in diesem Buch beschriebenen Elemente der Benutzeroberfläche sind also eigentlich die folgenden:

» das Ribbon, also die Zeile mit den Registerreitern *Start*, *Erstellen* und so weiter und den darunter befindlichen Gruppen und den darin enthaltenen Steuerelementen.

» der Backstage-Bereich (das ist der Bereich, der erscheint, wenn Sie in Access ab Version 2010 auf den Ribbonreiter *Datei* klicken)

» die Schnellzugriffsleiste (Quick Access Toolbar, QAT), welche immer sichtbare Schaltfläche anzeigt und in der Regel über dem eigentlichen Ribbon positioniert ist

» die Kontextmenüs (*contextMenus*), die zum Ausführen von Befehlen dienen, die per Klick mit der rechten Maustaste erscheinen und

» die Commands. Das sind nicht sichtbare Elemente, mit denen Sie die Funktion vorhandener, eingebauter Elemente durch benutzerdefinierte Funktionen ersetzen oder erweitern können.

Die Kontextmenü-Funktion des Ribbons können Sie unter Access nicht nutzen, um eigene Kontextmenüs zu programmieren. Daher zeigen wir in einem eigenen Kapitel, wie Sie dies per VBA erledigen.

1.1 Aufbau des Ribbons

Wir schauen uns zunächst die Elemente des Ribbons an, die wir auf den ersten Blick sehen (siehe Abbildung 1.1). Ganz oben sehen wir die Quick Access Toolbar beispielsweise mit den Schaltflächen zum Speichern, Wiederholen oder Rückgängigmachen von Schritten.

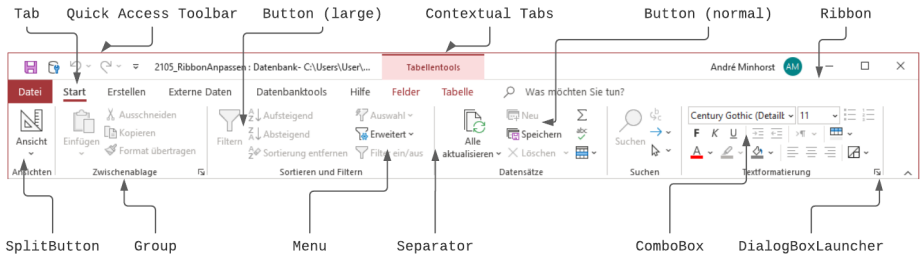


Abbildung 1.1: Elemente des Ribbons

Der Bereich darunter bis zum Arbeitsbereich von Access ist das Ribbon beziehungsweise das Menüband. Die Hauptelemente im Ribbon sind die *tab*-Elemente.

Neben diesen gibt es noch sogenannte Contextual Tabs. Diese werden nur im Kontext mit bestimmten Elementen der Benutzeroberfläche eingeblendet, beispielsweise mit der Datenblattansicht von Tabellen, Abfragen oder Formularen.

In der Hierarchie unterhalb der *tab*-Elemente sind die *group*-Elemente angeordnet. Jedes *tab*-Element kann ein oder mehrere *group*-Elemente enthalten, die jeweils nur mit dem übergeordneten Tab angezeigt werden. Die *group*-Elemente wiederum enthalten die eigentlichen Steuerelemente – siehe weiter unten. Außerdem finden wir im Ribbon rechts neben den Beschriftungen einiger *group*-Elemente noch sogenannte *dialogBoxLauncher*, mit denen Sie Dialoge mit weiteren Optionen zu den Gruppen öffnen können.

1.1.1 Der Bereich Datei/Backstage

Wenn Sie auf den Tab-Reiter *Datei* klicken, erscheint kein entsprechendes *Tab*-Element, sondern ein komplett anderer Bereich. Dieser heißt *Backstage*-Bereich und enthält einige Schaltflächen und Steuerelemente für allgemeine Funktionen von Access, zum Beispiel zum Öffnen und Schließen von Datenbanken, Drucken oder Anzeigen der Access-Optionen. Abbildung 1.2 zeigt diesen Bereich.

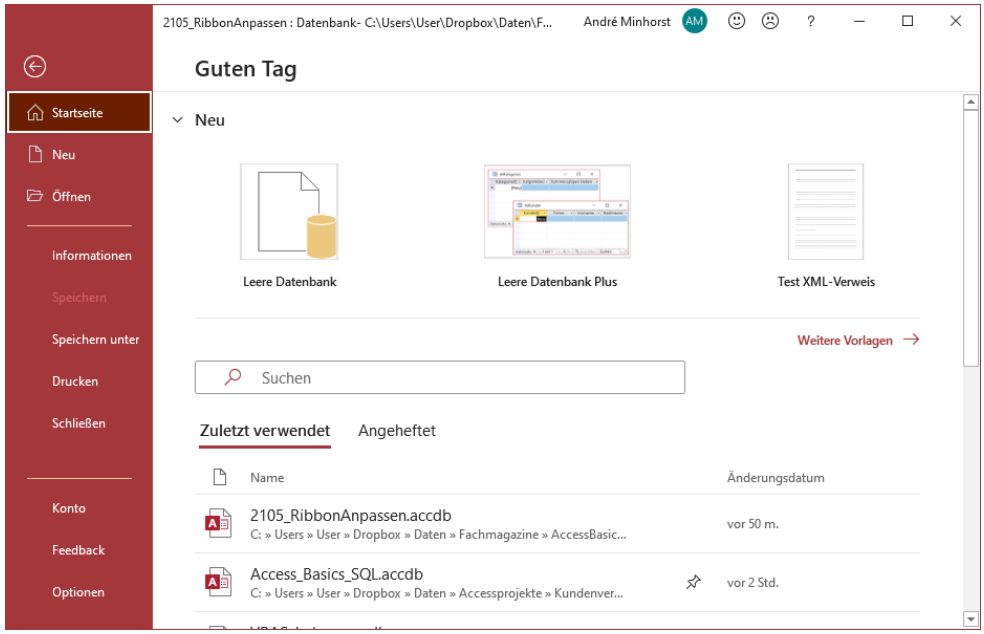


Abbildung 1.2: Der Backstage-Bereich

Die Schaltflächen hier können Sie zum Öffnen der dahintersteckenden Bereiche nutzen. Manche rufen auch einfach Funktionen auf, ohne einen eigenen Bereich anzuzeigen.

1.1.2 Steuern des Ribbons

Das Ribbon steuert man gewöhnlich durch Anklicken von Tabs oder Steuerelementen mit der Maus. Sie können aber auch Tastenkombinationen nutzen. Betätigen Sie die *Alt*-Taste, erscheinen die in Abbildung 1.3 sichtbaren Buchstaben im Ribbon, mit denen Sie die einzelnen Tabs oder Befehle der Schnellzugriffsleiste aufrufen können.

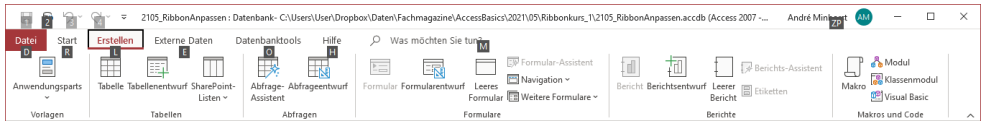


Abbildung 1.3: Aktivieren von Ribbonbefehlen per *Alt*-Taste

Betätigen Sie dann die Taste mit dem jeweiligen Buchstaben, um einen Befehl direkt auszuführen oder das entsprechende *tab*-Element zu aktivieren und die Tastenkombinationen für die dortigen Befehle anzuzeigen.

1.2 Mögliche Anpassungen des Ribbons

Es gibt zwei Möglichkeiten, wie man das Ribbon anpassen kann:

» Anpassung mit Bordmitteln (siehe »Ribbon mit Bordmitteln anpassen«, Seite 19)

» Anpassung mit XML (siehe »Ribbon per XML-Definition anpassen«, Seite 49)

Die Anpassung mit Bordmitteln bezieht sich immer auf die aktuelle Access-Instanz. Diese Anpassungen erfolgen über die Benutzeroberfläche in einem dafür vorgesehenen Dialog. Sie können diese Anpassungen in einer Datei speichern und diese gegebenenfalls auf einem anderen Rechner wiederherstellen. Anpassungen mit XML beziehen sich immer auf die jeweils geladene Datenbankanwendung. Damit können Sie zum Beispiel einer Kundendatenbank die zum Öffnen der darin enthaltenen Formulare zum Verwalten von Kunden benötigten Schaltflächen hinzufügen.

André Minhorst

Ribbon Programmierung

LESEPROBE

Hier finden Sie in der Vollversion noch weiteres Know-how rund um die Ribbon-Programmierung!

André Minhorst

Ribbon Programmierung

LESEPROBE

Hier finden Sie in der Vollversion noch weiteres Know-how rund um die Ribbon-Programmierung!

2 Ribbon mit Bordmitteln anpassen

Die Anpassung mit Bordmitteln bezieht sich immer auf die aktuelle Access-Installation, also nicht etwa auf die aktuelle Datenbankdatei.

Dafür ist eine XML-Anpassung nötig. Die Anpassung mit Bordmitteln erfolgt über zwei Bereiche des Dialogs *Access-Optionen*, und zwar:

» *Menüband anpassen*

» *Symbolleiste für den Schnellzugriff*

Unter *Menüband anpassen* finden Sie zwei Listen (siehe Abbildung 2.1). Die linke zeigt alle Befehle an, die überhaupt für das Ribbon zur Verfügung stehen. Diese können Sie nach verschiedenen Kriterien über das Kombinationsfeld *Befehle auswählen* filtern.

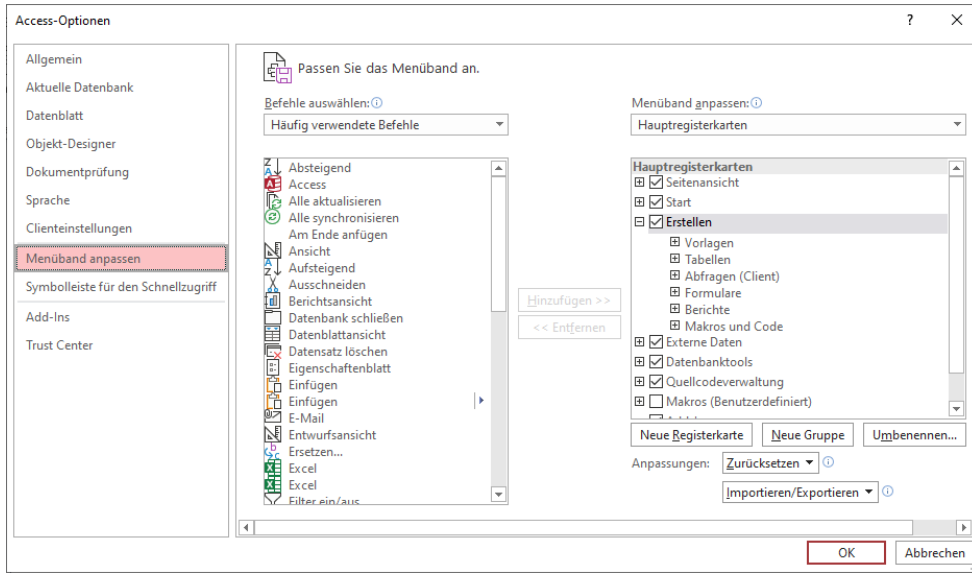


Abbildung 2.1: Bereich des Optionen-Dialogs zum Anpassen des Menübands

Rechts sehen Sie den Bereich *Menüband anpassen*, der die nach einem Eintrag des oberen Kombinationsfeldes gefilterten Registerkarten anzeigt.

Sie können jeden Befehl zu jedem Bereich des Ribbons hinzufügen, allerdings nur in einer eigenen, benutzerdefinierten Gruppe. Sie nutzen also zuerst die Schaltfläche *Neue Gruppe*, um eine neue Gruppe hinzuzufügen, und kopieren dann die Befehle per Drag-and-Drop dort hinein.

Sie können auch eine komplett neue Registerkarte hinzufügen (Schaltfläche *Neue Registerkarte*) oder benutzerdefinierte Elemente über die Schaltfläche *Umbenennen...* umbenennen. Diese benutzerdefinierten Einträge können Sie mit einem Klick auf die Schaltfläche *Zurücksetzen* wieder verwerfen. Oder Sie exportieren diese mit einem Klick auf die Schaltfläche *Importieren/Exportieren*. So können Sie die Änderungen in eine Datei exportieren und diese darüber wieder in eine andere Access-Installation importieren.

Wie das im Detail gelingt, zeigen wir weiter unten im Abschnitt »Ribbon über die Benutzeroberfläche anpassen« ab Seite 22.

Hier lernen Sie auch, wie Sie die Anpassungen sichern und auf dem gleichen oder auch auf einem anderen Rechner wiederherstellen.

Anpassen der Schnellzugriffsleiste

Die Schnellzugriffsleiste passen Sie auf ähnliche Weise an, und zwar im Bereich *Symbolleiste für den Schnellzugriff* in den Access-Optionen (siehe Abbildung 2.2). Der linke Bereich sieht genauso aus wie im zuvor beschriebenen Dialog, der rechte ist etwas einfacher gehalten. Hier können Sie nur einfach Einträge zur Schnellzugriffsleiste hinzufügen, diese hat keine weiteren Hierarchien wie Tabs oder Gruppen.

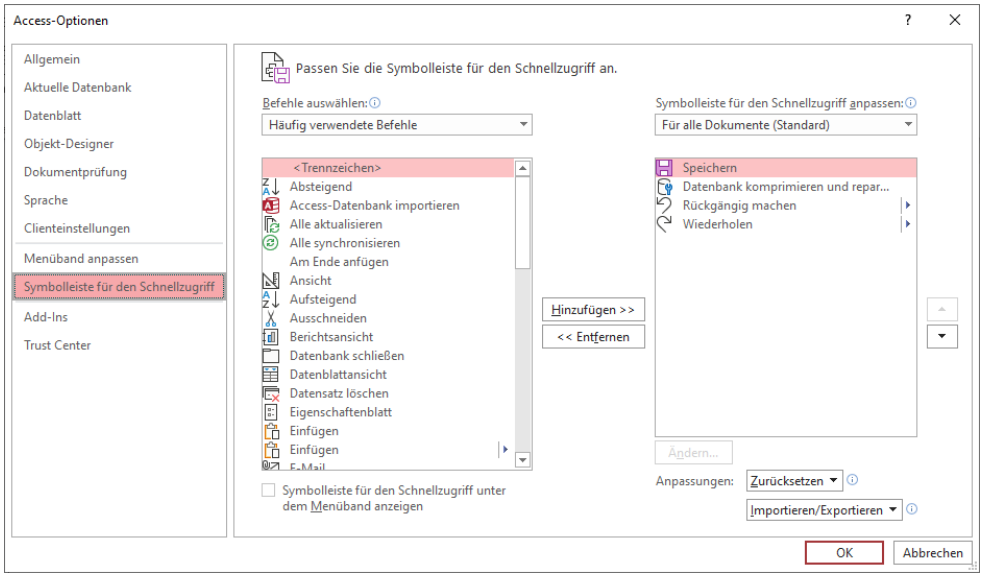


Abbildung 2.2: Bereich des Optionen-Dialogs zum Anpassen der Schnellzugriffsleiste

Wie Sie die Schnellzugriffsleiste über die Benutzeroberfläche anpassen, erfahren Sie unter dem Abschnitt »Schnellzugriffsleiste über die Benutzeroberfläche anpassen« ab Seite 38.

Anpassungen des Ribbons für die aktuelle Datenbankdatei

Die zweite Möglichkeit der Anpassung des Ribbons bezieht sich nicht auf die Access-Installation, sondern auf die aktuell geladene Access-Datenbank. Auf diese Weise können Sie einer von Ihnen programmierten Datenbank eine Art Menüleiste hinzufügen, mit der Sie oder andere Benutzer mit Schaltflächen die Funktionen der Datenbank starten oder Elemente wie Formulare oder Berichte aufrufen können. Mit weiteren Steuerelementen wie Menüs, Splitbuttons oder dem Gallery-Steuerelement können Sie die Schaltflächen weiter verschachteln.

Oder Sie nutzen Steuerelemente wie *comboBox* oder *dropDown* zur Auswahl von Werten, *editBox* zur Anzeige oder Eingabe von Texten oder auch Umschaltflächen oder Kontrollkästchen für das Anzeigen oder Einstellen von Optionen. Die Anpassungen lassen sich im Wesentlichen für zwei Bereiche definieren:

» die Hauptanwendung

» einzelne Formulare und Berichte

Sie können also eine Ribbondefinition festlegen, die gleich beim Start als Anwendungsribbon angezeigt wird. Diese Einstellung nehmen Sie in den Access-Optionen im Bereich *Aktuelle Datenbank* vor (siehe Abbildung 2.3).

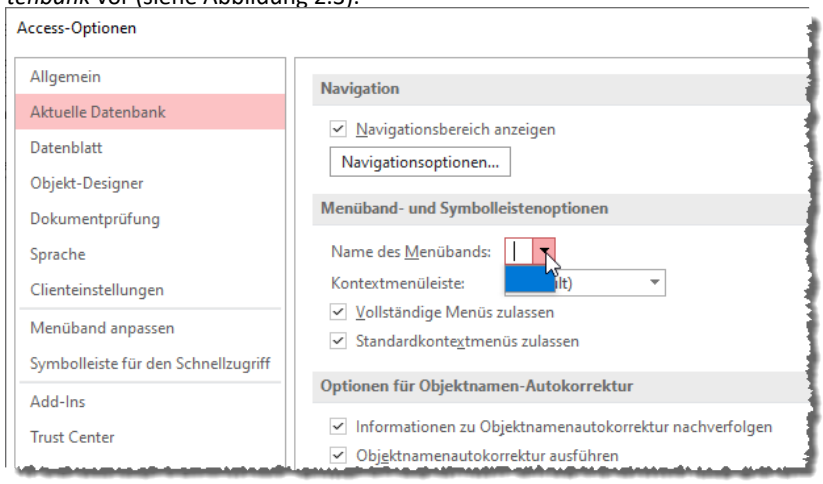


Abbildung 2.3: Option zum Auswählen des Menübands für eine Anwendung

Außerdem können Sie weitere Ribbondefinitionen hinterlegen, die Sie einzelnen Formularen oder Berichten zuweisen. Die entsprechenden Ribbons erscheinen dann, sobald das entsprechende Formular oder der Bericht geöffnet werden oder anderweitig den Fokus erhalten.

Diese Einstellung nehmen Sie für die Eigenschaft *Name des Menübands* im Bereich *Andere* des Formularentwurfs vor (siehe Abbildung 2.4). Auch hier sehen wir noch keinen Wert, da wir noch keine Ribbondefinition zur Anwendung hinzugefügt haben.

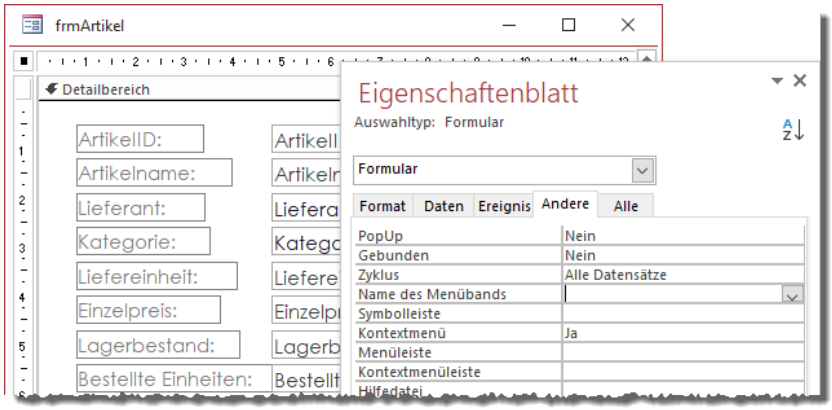


Abbildung 2.4: Option zum Auswählen des Menübands für ein Formular

Die gleiche Eigenschaft finden Sie auch für Berichte vor.

2.1 Ribbon über die Benutzeroberfläche anpassen

Den für die Anpassungen notwendigen Bereich in den Access-Optionen können Sie auf zwei Arten öffnen:

» Anzeigen der Access-Optionen durch Anklicken von *Datei* und anschließendes Betätigen des Eintrags *Optionen* im Backstage-Bereich, danach wechseln Sie zum Bereich *Menüband anpassen*.

» Klicken auf die Schaltfläche mit dem Pfeil nach unten in der Schnellzugriffsleiste und Betätigen des Eintrags *Weitere Befehle...* (siehe Abbildung 2.5).

Dies öffnet die Access-Optionen und zeigt direkt den Bereich *Symbolleiste für den Schnellzugriff* an. Hier wollen wir allerdings jetzt nicht hin, sondern zum Bereich *Menüband anpassen*. Mit einem Klick auf diesen Eintrag öffnen Sie diesen Bereich.

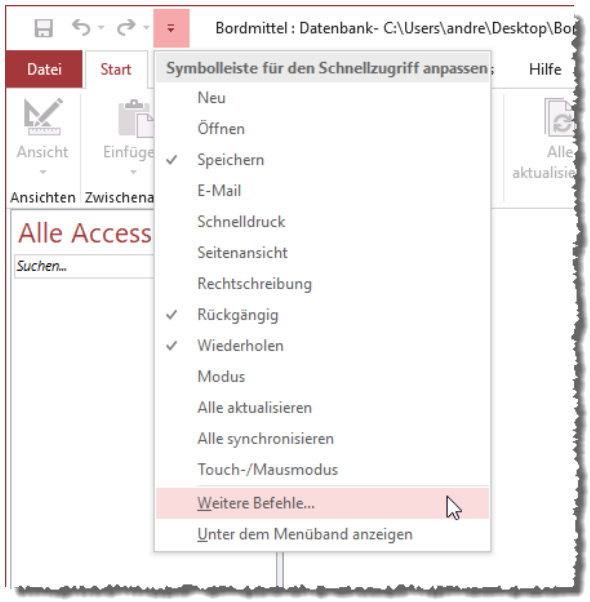


Abbildung 2.5: Öffnen der Access-Optionen

2.1.1 »Menüband anpassen« in den Access-Optionen

Der Dialog »Menüband anpassen« ist die Schaltzentrale, wenn es um Anpassungen des Ribbons geht, die nicht nur für die aktuell geöffnete Datenbank gelten, sondern die immer wirksam sind, unabhängig von der aktuell geöffneten Datenbank. Es gibt natürlich Ausnahmen, denn Sie können durch die Ribbondefinition für eine bestimmte Datenbank die allgemein gültigen Ribbondefinitionen überschreiben.

Der Dialog enthält die folgenden Bereiche:

» Kombinationsfeld *Befehle auswählen*: Hier filtern Sie die in dem darunter befindlichen Listenfeld angezeigten Befehle.

» Liste der Befehle: Die Liste unter dem Feld *Befehle auswählen* zeigt alle Befehle des Ribbons an, die zu der ausgewählten Kategorie gehören.

» Kombinationsfeld *Menüband anpassen*: Hier stellen Sie ein, welche Registerkarten (*tab*-Elemente) des Ribbons in der Liste darunter angezeigt werden sollen.

» Liste unterhalb vom Kombinationsfeld *Menüband anpassen*: Hier werden die Registerkarten (*tab*-Elemente) und die untergeordneten Gruppen und Befehle gefiltert nach der Auswahl aus dem darüber befindlichen Kombinationsfeld angezeigt.

» Die Schaltflächen *Neue Registerkarte*, *Neue Gruppe* und *Umbenennen...* erlauben das Anpassen des Ribbons.

» Unten rechts finden Sie dann noch zwei Schaltflächen namens *Zurücksetzen* und *Importieren/Exportieren*, deren Funktionen wir weiter unten vorstellen.

In Abbildung 2.6 sehen Sie den Bereich *Menüband anpassen* in der Übersicht. In den folgenden Abschnitten gehen wir auf die einzelnen Elemente ein.

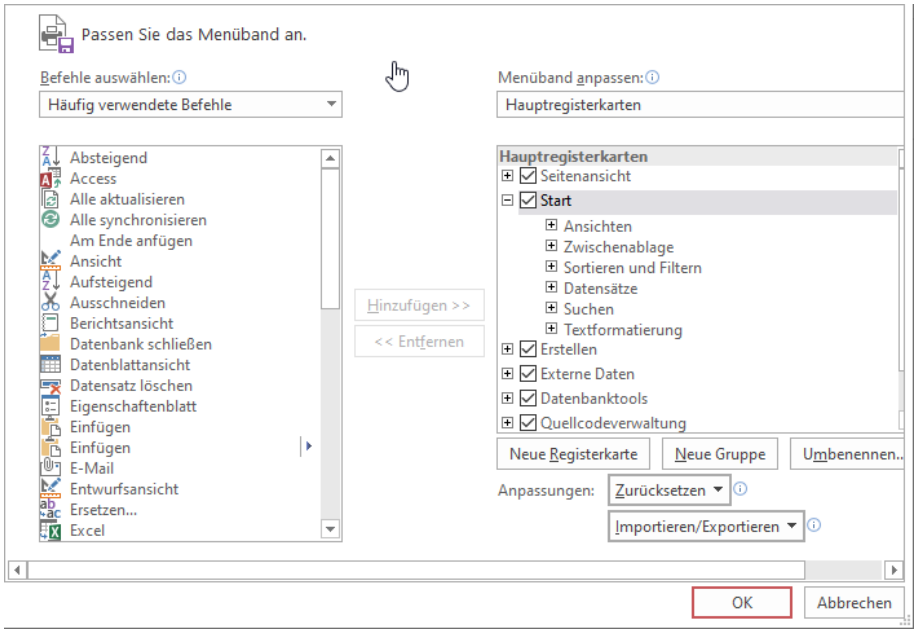


Abbildung 2.6: Der Bereich »Menüband anpassen« in den Access-Optionen

2.1.2 Befehlskategorien und Befehle

Im linken Teil des Bereichs *Menüband anpassen* sehen wir ein Kombinationsfeld namens Befehle auswählen, mit dem Sie einen der Filter wie *Häufig verwendete Befehle*, *Alle Befehle*, *Registerkarte »Datei«* oder andere vorfinden (siehe Abbildung 2.7). Sie werden in der Regel nach einem bestimmten Befehl suchen, den es bereits irgendwo im Ribbon gibt, und wollen diesen in der Liste aller Befehle ausfindig machen. Dazu können Sie die Befehle nach den Einträgen unter *Befehle auswählen* filtern. Wenn Sie beispielsweise einen Befehl aus dem Backstagebereich an anderer Stelle im Ribbon verfügbar machen wollen, wählen Sie einfach den Eintrag »Registerkarte Datei« aus.

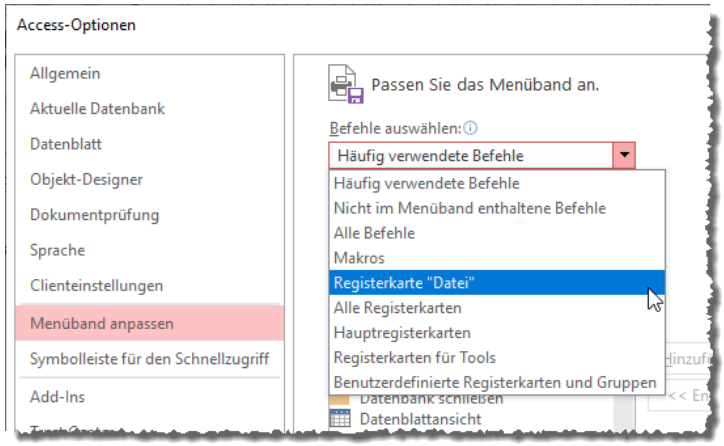


Abbildung 2.7: Auswahl der Befehlskategorien

Danach finden Sie in der Liste unter der Auswahl alle darin enthaltenen Befehle. Diese sind alphabetisch sortiert, sodass Sie den gewünschten Befehl schnell auffinden können sollten (siehe Abbildung 2.8). Wir nehmen einfach an, dass Sie den Befehl zum Komprimieren und Reparieren einer Datenbank an prominenterer Stelle unterbringen wollen – beispielsweise auf der Registerseite *Start*.

Den dafür benötigten Eintrag namens *Datenbank komprimieren und reparieren* finden Sie dann auch gleich an vorletzter Position des sichtbaren Bereichs in der Liste der Befehle.

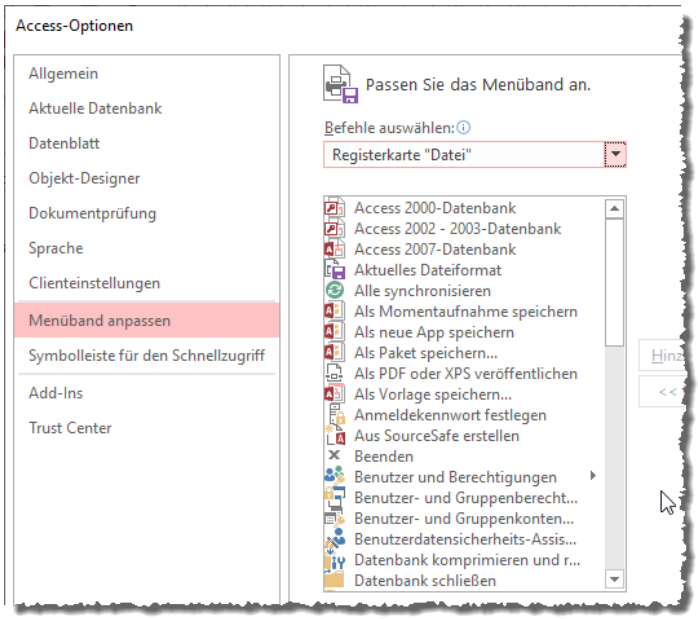


Abbildung 2.8: Befehle der Kategorie *Datei*

idMso ermitteln

An dieser Stelle auch gleich der Hinweis, dass jeder eingebaute Ribbonbefehl eine eindeutige ID aufweist, die sogenannte *idMso*. Diesen Wert benötigen Sie beispielsweise, wenn Sie einen der eingebauten Befehle in einer benutzerdefinierten Ribbondefinition unterbringen möchten. Um diese *idMso* zu ermitteln, suchen Sie den gewünschten Befehl in der linken Liste und positionieren dann den Mauszeiger auf diesem Befehl. Daraufhin erscheint ein Tooltip-Text, der ganz hinten die englischsprachige *idMso* präsentiert (siehe Abbildung 2.9).

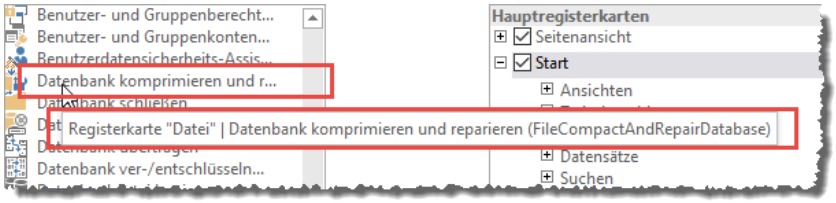


Abbildung 2.9: Ermitteln der *idMso* eines Steuerelements

2.1.3 Bereich zum Anpassen des Menübands

Im rechten Bereich des Dialogs finden Sie wieder zwei Steuerelemente. Das obere Kombinationsfeld dient wieder als Filter für die Registerkarten beziehungsweise *tab*-Elemente, die in der unteren Liste angezeigt werden.

Hier wählen Sie einen der drei Einträge *Alle Registerkarten*, *Hauptregisterkarten* oder *Registerkarten für Tools* aus (siehe Abbildung 2.10).

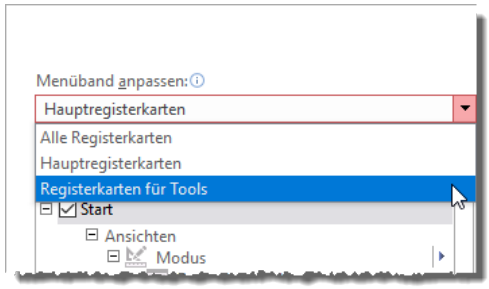


Abbildung 2.10: Auswahl der anzuzeigenden Registerkarten

Wir wollen, wie oben angekündigt, zunächst einmal den Befehl *Datenbank komprimieren und reparieren* zur Registerkarte *Start* hinzufügen. Diese Registerkarte gehört zur Gruppe *Hauptregisterkarten*, weshalb wir diese als Filterkriterium auswählen.

Danach sehen Sie in der Liste unten die Hauptregisterkarten (siehe Abbildung 2.11). Hier finden wir auch gleich die Registerkarte *Start*. Diese haben wir im Screenshot aufgeklappt, sodass direkt einige Untereinträge erscheinen. Diese entsprechen den Gruppen, die auf dieser Registerkarte angezeigt werden.

Klappen Sie eine dieser Gruppe auf, erscheinen sogar noch die Befehle, die in dieser Gruppe enthalten sind.

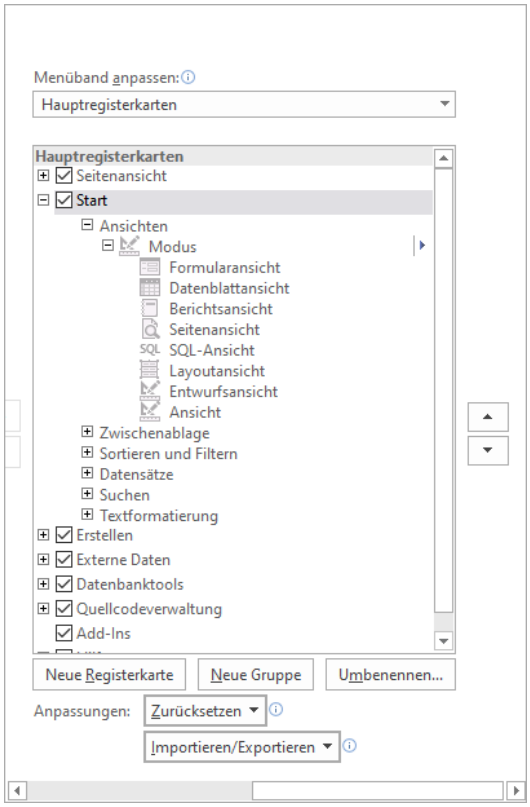


Abbildung 2.11: Bereich zum Bearbeiten der Registerkarten und Gruppen

2.1.4 Eingebautes Steuerelement zu einer Registerkarte hinzufügen

Damit kommen wir zum eigentlichen Hinzufügen des Befehls *Datenbank komprimieren und reparieren* zur Registerkarte *Start*. Dazu markieren wir den Befehl *Datenbank komprimieren und reparieren* in der linken Liste und in der rechten Liste markieren wir als Ziel die Gruppe *Textformatierung* (das macht keinen Sinn, aber es geht nur um ein Beispiel). Danach betätigen Sie die Schaltfläche *Hinzufügen >>* zwischen den beiden Listen (siehe Abbildung 2.12).

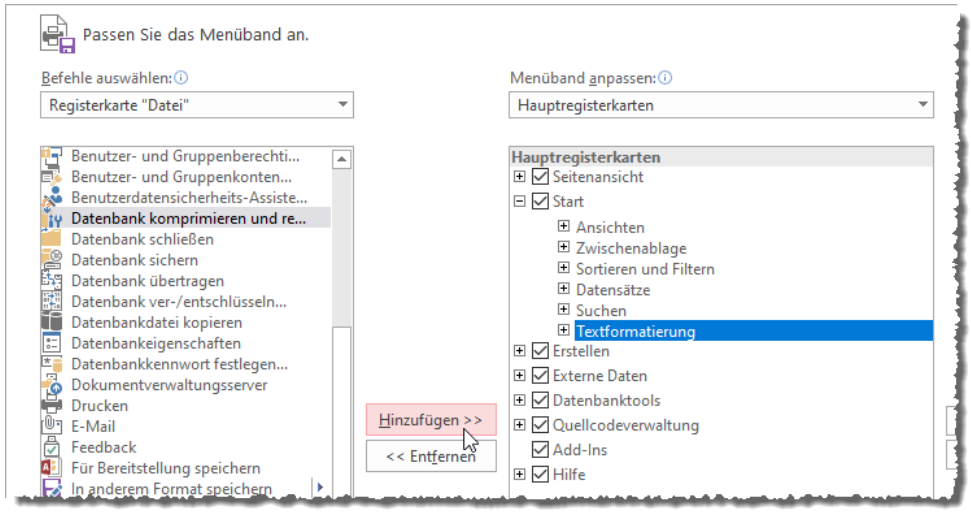


Abbildung 2.12: Versuch, einen Befehl zu einer bestehenden Gruppe hinzuzufügen

Dies führt allerdings zu der Fehlermeldung aus Abbildung 2.13. Wir können also keine Befehle zu eingebauten Gruppen hinzufügen.

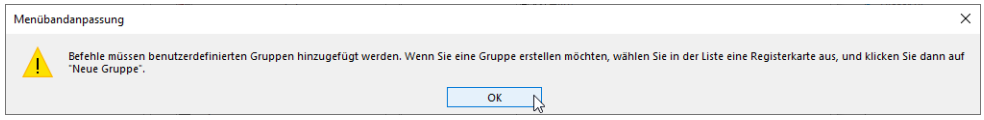


Abbildung 2.13: Meldung beim Versuch, einen Befehl zu einer eingebauten Gruppe hinzuzufügen

2.1.5 Neue Gruppe erstellen

Also müssen wir eine neue Gruppe zu der gewünschten Registerkarte hinzufügen. Dazu gibt es zwei Möglichkeiten:

» Sie klicken mit der rechten Maustaste auf den Eintrag für die Hauptregisterkarte, der Sie die Gruppe hinzufügen möchten, und wählen den Kontextmenüeintrag *Neue Gruppe hinzufügen* aus (siehe Abbildung 2.14).

» Oder Sie markieren den Eintrag für die Hauptregisterkarte, der Sie die Gruppe hinzufügen möchten, und klicken unten auf die Schaltlfäche *Neue Gruppe*.

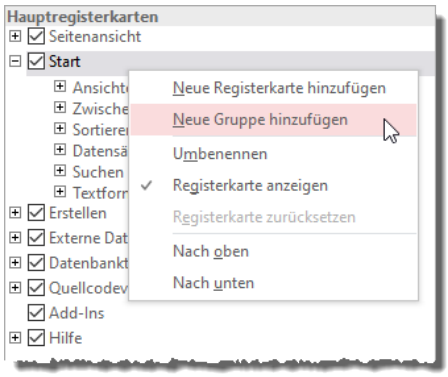


Abbildung 2.14: Hinzufügen einer neuen Gruppe

Die neue Gruppe erscheint anschließend wie in Abbildung 2.15 in der Übersicht der Registerkarten.

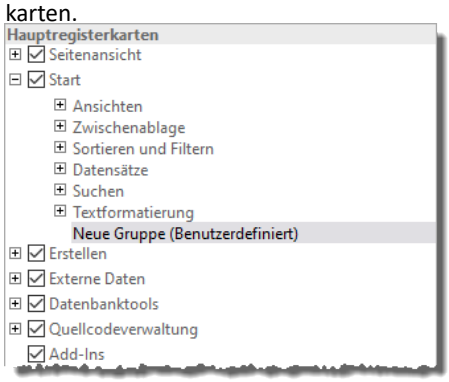


Abbildung 2.15: Die neu hinzugefügte Gruppe

Allerdings entspricht der Name noch nicht unseren Wünschen. Also ändern wir diesen, was wiederum auf zwei Arten möglich ist:

» Sie wählen den Eintrag *Umbenennen* aus dem Kontextmenü aus.

» Markieren den zu ändernden Eintrag und klicken auf die Schaltfläche *Umbenennen...* im unteren Bereich des Dialogs.

Beides öffnet den Dialog *Umbenennen* aus Abbildung 2.16. HIer können Sie nicht nur einen neuen Namen für die Gruppe eingeben, sondern auch noch ein Symbol festlegen. Wir verwenden die Bezeichnung *Tools* und wählen eines der Symbole aus.

André Minhorst

Ribbon Programmierung

LESEPROBE

Hier finden Sie in der Vollversion noch weiteres Know-how rund um die Ribbon-Programmierung!

André Minhorst

Ribbon Programmierung

LESEPROBE

Hier finden Sie in der Vollversion noch weiteres Know-how rund um die Ribbon-Programmierung!

3 Ribbon per XML-Definition anpassen

Die zweite Möglichkeit zum Anpassen des Ribbons ist die über die Definition der Anpassung per XML. Die Struktur des Ribbons lässt sich mit XML abbilden und somit können Sie bestehende, eingebaute Elemente referenzieren und manipulieren oder auch ausblenden, aber auch eigene Tabs, Gruppen oder Steuerelemente zu eingebauten oder neuen Elementen hinzufügen. Wie das gelingt, zeigen wir in diesem Kapitel.

Die Beispiele zu diesem Kapitel finden Sie in der folgenden Beispieldatenbank:

```
RibbonPerXMLDefinitionAnpassen.accdb
```

3.1 Einfache Ribbondefinition erstellen

Damit wir uns in der Praxis ansehen können, wie wir ein Ribbon erstellen und dieses auch als Anwendungsribbon oder als Formular- oder Berichtsribbon anzeigen können, müssen wir eine entsprechende Definition erstellen. Diese formulieren wir im XML-Format. Genau wie wir oben festgestellt haben, dass das Ribbon nur ein Teil der möglichen Anpassungen der Benutzeroberfläche ist, sieht es auch in der XML-Definition aus. Das Basiselement lautet nämlich *customUI* und wird wie folgt angelegt:

```
<?xml version="1.0"?>
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui"/>
```

Diesem fügen wir als Erstes ein *ribbon*-Element hinzu, weil wir das eigentliche Ribbon anpassen möchten:

```
<?xml version="1.0"?>
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui">
  <ribbon/>
</customUI>
```

Der nächste Schritt ist ein *tab*-Element, das allerdings – im Ribbon nicht sichtbar – ein umschließendes *tabs*-Element benötigt. Dieses fasst alle *tab*-Elemente ein.

Ein *tab*-Element muss zumindest das Attribut *id* aufweisen und sollte auch eine Beschriftung enthalten, hier durch das *label*-Attribut realisiert:

```
<?xml version="1.0"?>
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui">
  <ribbon>
    <tabs>
```

```
        <tab id="tab" label="Beispieltab"/>
      </tabs>
    </ribbon>
  </customUI>
```

Das reicht als Grundstein aus, um die Anpassung vorzunehmen. Dazu sind jedoch noch weitere Schritte nötig.

3.1.1 Tabelle zum Speichern von Ribbondefinitionen

Wir benötigen nämlich eine Tabelle, in der wir alle Ribbondefinitionen für die Anwendung speichern. Diese enthält die folgenden Felder:

» *ID*: Primärschlüsselfeld der Tabelle

» *RibbonName*: Bezeichnung, die in den Auswahlfeldern zum Festlegen der Menübänder in den Access-Optionen und den Eigenschaften von Formularen und Berichten angezeigt werden soll

» *RibbonXML*: Definition des Ribbons im XML-Format

Der Entwurf der Tabelle wird in Abbildung 3.1 abgebildet.

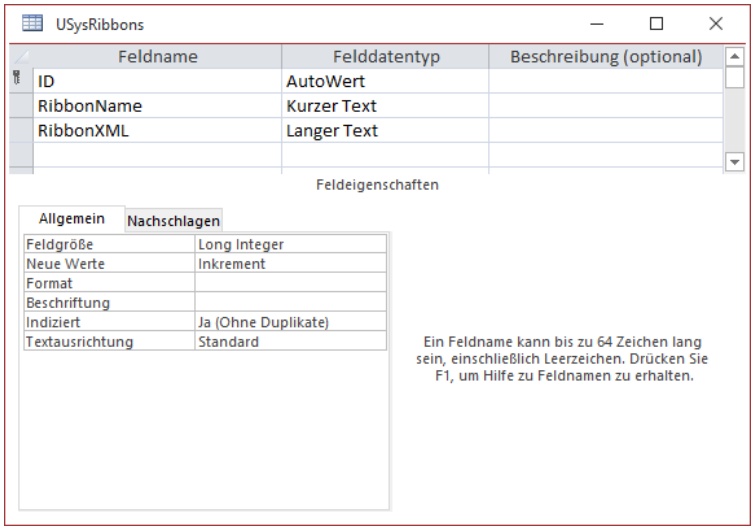


Abbildung 3.1: Entwurf der Tabelle zum Speichern der Ribbondefinitionen

Diese Tabelle muss zwingend *USysRibbons* heißen. Wenn wir in die Datenblattansicht wechseln, können wir den soeben entwickelten XML-Code für die Ribbondefinition wie in Abbildung 3.2 als ersten Datensatz in die Tabelle einfügen.

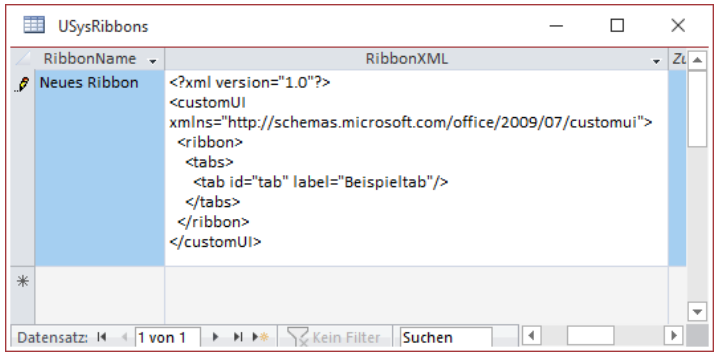


Abbildung 3.2: Eine erste Ribbondefinition in der Tabelle *USysRibbons*

Wenn wir nun allerdings versuchen, dieses Ribbon in den Access-Optionen oder als Eigenschaft eines Formulars oder Berichts auszuwählen, gelingt dies noch nicht auf Anhieb.

Dazu ist noch ein wichtiger Schritt notwendig: Wir müssen die Anwendung neu starten und somit die Inhalte der Tabelle *USysRibbons* einlesen, damit diese in den Auswahlfeldern mit den verfügbaren Ribbondefinitionen erscheinen.

3.1.2 USysRibbons ist eine Systemtabelle

Unter Umständen haben Sie die Tabelle *USysRibbons* nun gespeichert und geschlossen und wundern sich, dass diese gar nicht im Navigationsbereich erscheint. Das ist der Fall, weil *USysRibbons* mit *USys...* beginnt, was benutzerdefinierten Systemtabellen vorbehalten ist. Diese werden von Access standardmäßig nicht angezeigt. Wenn Sie diese bearbeiten wollen, können Sie die Systemtabellen über eine Option einblenden. Dazu klicken Sie mit der rechten Maustaste auf den Titel des Navigationsbereichs und wählen den Kontextmenü-Eintrag *Navigationsoptionen...* aus.

Es erscheint der Dialog aus Abbildung 3.3, in dem Sie die Option *Systemobjekte anzeigen* aktivieren.

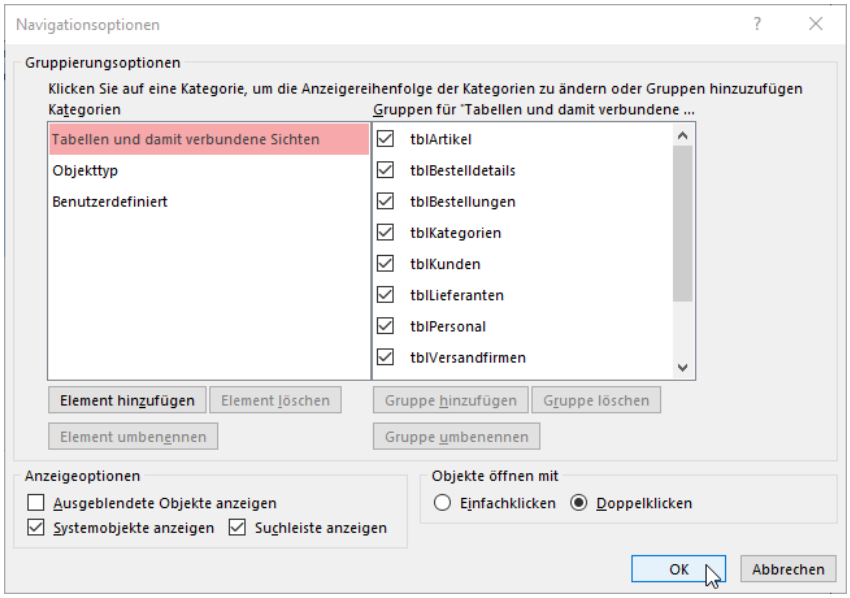


Abbildung 3.3: Einblenden der Systemtabellen

Direkt danach erscheint auch die Tabelle *USysRibbons* im Navigationsbereich, wo Sie diese per Doppelklick zur weiteren Bearbeitung öffnen können.

Wenn Sie die Systemtabellen nicht einblenden möchten, können Sie die Tabelle *USysRibbons* übrigens auch mit dem folgenden Befehl über den Direktbereich des VBA-Editors öffnen (anzuzeigen mit *Strg + G*):

```
DoCmd.OpenTable "USysRibbons"
```

3.1.3 Anwendungsribbon einstellen

Nun wollen wir allerdings erstmals unser Ribbon als Anwendungsribbon einstellen. Also schließen wir die Datenbank und öffnen diese erneut, was am schnellsten über den Befehl *Datei|Informationen|Datenbank komprimieren und reparieren* gelingt.

Danach öffnen wir die Access-Optionen und wechseln dort zum Bereich *Aktuelle Datenbank*. Unter *Menüband- und Symbolleistenoptionen* finden wir die Option *Name des Menübands*, wo wir nun die Definition mit dem Namen aus dem Feld *RibbonName* der Tabelle *USysRibbons* auswählen können (siehe Abbildung 3.4).

André Minhorst

Ribbon Programmierung

LESEPROBE

Hier finden Sie in der Vollversion noch weiteres Know-how rund um die Ribbon-Programmierung!

André Minhorst

Ribbon Programmierung

LESEPROBE

Hier finden Sie in der Vollversion noch weiteres Know-how rund um die Ribbon-Programmierung!

4 Eingabehilfe für Ribbondefinitionen

Bevor wir uns in den nächsten Abschnitten die übrigen Steuerelemente ansehen, bauen wir uns noch ein Eingabeformular für die Ribbondefinitionen. Dieses erlaubt den Zugriff auf die Tabelle *USysRibbons*, in der die benutzerdefinierten Ribbondefinitionen als Erweiterung von Datenbanken gespeichert werden. Sie können hier den Code ansehen, bearbeiten, Ribbondefinitionen erstellen oder löschen und die aktuell gewählte Ribbondefinition anzeigen oder als Anwendungsribbon definieren.

Wenn Sie ein professionelles Tool zum Anlegen von Ribbondefinitionen verwenden wollen, finden Sie in »Ribbon-Admin« ab Seite 219 eine Dokumentation eines solchen Tools sowie weitere Informationen.

Die Beispiele zu diesem Kapitel finden Sie in der folgenden Beispieldatenbank:

```
EingabehilfeFuerRibbons.accdb
```

4.1 Formular erstellen

Bei dem Eingabeformular handelt es sich um ein Formular, das die Tabelle *USysRibbons* als Datensatzquelle verwendet. Dazu legen wir zuerst ein neues Formular an uns speichern dieses unter dem Namen *frmRibbonXML*. Danach stellen Sie die Eigenschaft *Datensatzquelle* auf den Wert *USysRibbons* ein. Das setzt allerdings zunächst einmal voraus, dass diese Tabelle überhaupt schon in der aktuellen Datenbank vorhanden ist. Diese Tabelle müssen Sie nämlich, wenn Sie Ribbons in Ihrer Anwendung anzeigen wollen, zunächst erstellen. Wie das gelingt, zeigen wir unter »Tabelle zum Speichern von Ribbondefinitionen« ab Seite 50.

Damit diese Systemtabelle überhaupt in der Liste der möglichen Werte für die Eigenschaft *Datensatzquelle* erscheint, müssen Sie die Anzeige der Systemobjekte für diese Access-Datenbank aktivieren. Mehr dazu erfahren Sie unter »USysRibbons ist eine Systemtabelle« ab Seite 51. Sie können allerdings auch einfach den Namen der Tabelle als Wert der Eigenschaft *Datensatzquelle* eintragen.

Die Eigenschaft sieht anschließend so aus wie in »Einstellen der Eigenschaft Datensatzquelle« ab Seite 58 beschrieben.

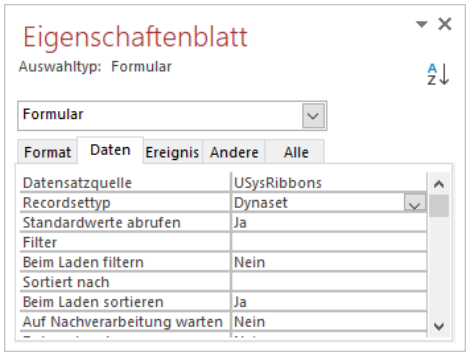


Abbildung 4.1: Einstellen der Eigenschaft *Datensatzquelle*

Nach dem gegebenenfalls notwendigen Erstellen der Tabelle und dem Einstellen der Datensatzquelle zeigen Sie die Feldliste an und ziehen die enthaltenen drei Felder in den Detailbereich der Entwurfsansicht des neuen Formulars (siehe Abbildung 4.2).

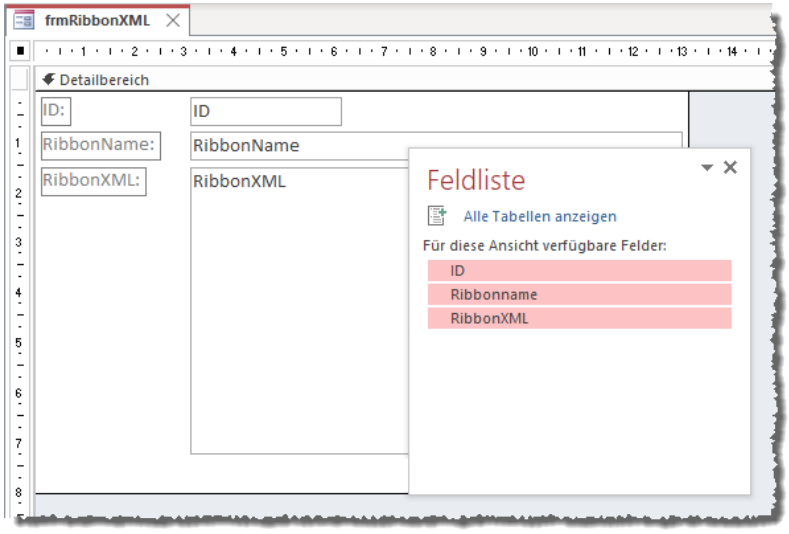


Abbildung 4.2: Einfügen der benötigten Felder in das Formular

Für das Feld *RibbonXML* stellen wir die Eigenschaften *Horizontaler Anker* und *Vertikaler Anker* jeweils auf den Werte *Beide* ein. Auf diese Weise wird es beim Vergrößern des Formulars ebenfalls vergrößert. Damit das Beschriftungsfeld dieses Textfeldes weiterhin an der vorgesehenen Position verbleibt, legen Sie den Wert für die Eigenschaft *Horizontaler Anker* auf *Links* und *Vertikaler Anker* auf *Oben* fest.

Damit Sie bequem Ribbondefinitionen in das Formular einfügen können, stellen wir noch die Eigenschaft *Eingabetastenverhalten* auf *Neue Zeile im Feld* ein. Auf diese Weise wird das Feld beim Betätigen der Eingabetaste nicht verlassen, sondern Sie fügen damit einen Zeilenumbruch an der aktuellen Position der Einfügemarke ein.

4.2 Aktuelles Ribbon als Anwendungsribben einstellen

Außerdem fügen wir eine Schaltfläche hinzu, mit der Sie das aktuelle Ribbon schnell als Anwendungsribbon festlegen können. Dabei führen wir den Schritt, der normalerweise das Öffnen der Access-Optionen und das Einstellen der Option *Name des Menübands* auf das gewünschte Ribbon erfordert, per Code aus.

Die notwendige Ereignisprozedur sieht wie folgt aus:

```
Private Sub cmdAlsAnwendungsribbonFestlegen_Click()
    Dim db As DAO.Database
    Dim prp As DAO.Property
    Dim strRibbonname As String
    Set db = CurrentDb
    strRibbonname = Me!RibbonName
    On Error Resume Next
    Set prp = db.Properties("CustomRibbonID")
    On Error GoTo 0
    If prp Is Nothing Then
        Set prp = db.CreateProperty("CustomRibbonID", dbText, strRibbonname)
        db.Properties.Append prp
    Else
        prp.Value = strRibbonname
    End If
End Sub
```

Die Prozedur liest als Erstes den Wert des Feldes *Ribbonname* des Formulars in die Variable *strRibbonname* ein. Dann versucht sie, bei deaktivierter Fehlermeldung auf eine Eigenschaft namens *CustomRibbonID* zuzugreifen. Das ist die Repräsentation der Eigenschaft *Name des Menübands* der Access-Optionen. Hat die Variable *prp* anschließend nicht den Wert *Nothing*, ist die Eigenschaft bereits vorhanden (sie wird erst beim erstmaligen Zuweisen erstellt). Sollte *prp* den Wert *Nothing* haben, die Eigenschaft also noch nicht vorhanden sein, erstellt die Prozedur diese mit der *CreateProperty*-Methode neu und weist ihr als Wert den Inhalt von *strRibbonname* zu. Anschließend hängt sie das neu erstellte *Property*-Element an die Auflistung *Properties* der Datenbank an.

Ist die Property namens *CustomRibbonUI* bereits vorhanden, stellt die Prozedur diese einfach auf den Wert aus strRibbonname ein.

Anschließend muss die Datenbank wie üblich nach dem Ändern der Eigenschaft *Name des Menübands* erneut geöffnet werden, damit die Ribbondefinition angewendet wird.

4.3 Aktuelles Anwendungsribbon direkt im Formular anzeigen

Eine weitere praktische Funktion fügen wir über die Ereignisprozedur *Form_Load* hinzu. Diese sorgt dafür, dass das Formular *frmRibbonXML* immer die Ribbondefinition anzeigt, die gerade als Anwendungsribbon eingestellt ist:

```
Private Sub Form_Load()
    Dim strRibbonname As String
    On Error Resume Next
    strRibbonname = CurrentDb.Properties("CustomRibbonID")
    If Not Len(strRibbonname) = 0 Then
        Me.Recordset.FindFirst "Ribbonname = '" & strRibbonname & "'"
    End If
End Sub
```

4.4 Angezeigte Ribbondefinition anwenden

Sie möchten sicher nicht jedes Mal die Anwendung neu starten, um eine an einer Ribbondefinition durchgeführte Änderung testen zu können. Es geht auch etwas einfacher, und zwar indem Sie einfach die Eigenschaft *Name des Menübands* des aktuellen Formulars auf den Namen des anzuzeigenden Ribbons einstellen. Das ist allerdings nur möglich, wenn das Ribbon in der Tabelle *USysRibbons* gespeichert und die Anwendung anschließend einmal neu geöffnet wurde. Dann können Sie eine Schaltfläche namens *cmdFormularribbonHinzufuegen* mit dem folgenden Code versehen, damit die aktuell angezeigte Ribbondefinition auch angewendet wird:

```
Private Sub cmdAlsFormularribbonFestlegen_Click()
    Me.RibbonName = Me!RibbonName
End Sub
```

Damit können Sie allerdings auch nur die Version der gewählten Ribbondefinition anzeigen, wie Sie vor dem letzten Öffnen der Datenbank aussah. Eine spontante Änderung und erneutes Anklicken der Schaltfläche liefert nicht direkt die geänderte Version.

André Minhorst

Ribbon Programmierung

LESEPROBE

Hier finden Sie in der Vollversion noch weiteres Know-how rund um die Ribbon-Programmierung!

André Minhorst

Ribbon Programmierung

LESEPROBE

Hier finden Sie in der Vollversion noch weiteres Know-how rund um die Ribbon-Programmierung!

5 Programmieren von Buttons mit VBA

Im vorherigen Kapitel haben wir uns angesehen, wie Sie eine einfache Ribbon-Erweiterung samt Schaltfläche implementieren. Diese hat allerdings noch keine Funktion. Diese fügen wir per VBA hinzu. In diesem Kapitel beschränken wir uns auf die Techniken zum programmieren des *onAction*-Ereignisses von *button*-Elementen. Wie Sie die übrigen Steuerelemente programmieren, erfahren Sie später in der Beschreibung der einzelen Steuerelemente.

Die Beispiele zu diesem Kapitel finden Sie in der folgenden Beispieldatenbank:

```
ProgrammierenVonButtonsMitVBA.accdb
```

5.1 Ereignis beim Anklicken der Schaltfläche

Auch wenn die Schaltfläche noch ein Icon vertragen könnte, wollen wir zunächst eine Funktion dafür hinterlegen. Dazu sind einige weitere Schritte nötig. Als Erstes müssen wir in der Ribbondefinition hinterlegen, dass überhaupt eine Aktion beim Anklicken des *button*-Elements ausgelöst werden soll. Dazu fügen wir das Attribut *onAction* hinzu und legen als Wert den Namen einer VBA-Routine fest, die beim Anklicken der Schaltfläche ausgelöst werden soll, zum Beispiel ebenfalls *onAction*. Der Code nur für das *button*-Element innerhalb unserer Definition lautet nun:

```
<?xml version="1.0"?>
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui">
  <ribbon>
    <tabs>
      <tab id="tab" label="Beispieltab">
        <group id="grp" label="Beispielgruppe">
          <button label="Beispielbutton" id="btn" onAction="onAction"/>
        </group>
      </tab>
    </tabs>
  </ribbon>
</customUI>
```

5.1.1 VBA-Prozedur für das Callback-Attribut

Nun benötigen wir eine entsprechende Prozedur in einem Modul des VBA-Projekts der Datenbankdatei, die beim Anklicken des *button*-Elements ausgelöst wird. Diese hat eine spezielle Signatur, die wir zwingend verwenden müssen – anderenfalls wird die Routine nicht gefunden.

Die Routinen für das Ribbon wollen wir in einem eigenen Modul namens *mdlRibbon* ablegen. Die Prozedur sieht ohne weitere Anweisungen zunächst wie folgt aus:

```
Sub onAction(control As IRibbonControl)


End Sub
```

5.1.2 Fehlender Office-Verweis für Ribbon-Funktionen

Wenn Sie die Anwendung nun kompilieren, erhalten Sie allerdings einen Fehler. Die entsprechende Fehlermeldung sehen Sie in Abbildung 5.1. Der einzige benutzerdefinierte Typ, der hier infrage kommt, heißt *IRibbonControl*. Warum ist dieser nicht definiert? Weil die Elemente zur Programmierung des Ribbons nicht in den Bibliotheken enthalten sind, die standardmäßig in einem VBA-Projekt referenziert werden, sondern in einer Bibliothek, die allen Office-Anwendungen zur Verfügung steht. Das ist logisch, denn auch das Ribbon basiert ja in allen Office-Anwendungen auf der gleichen Technik.

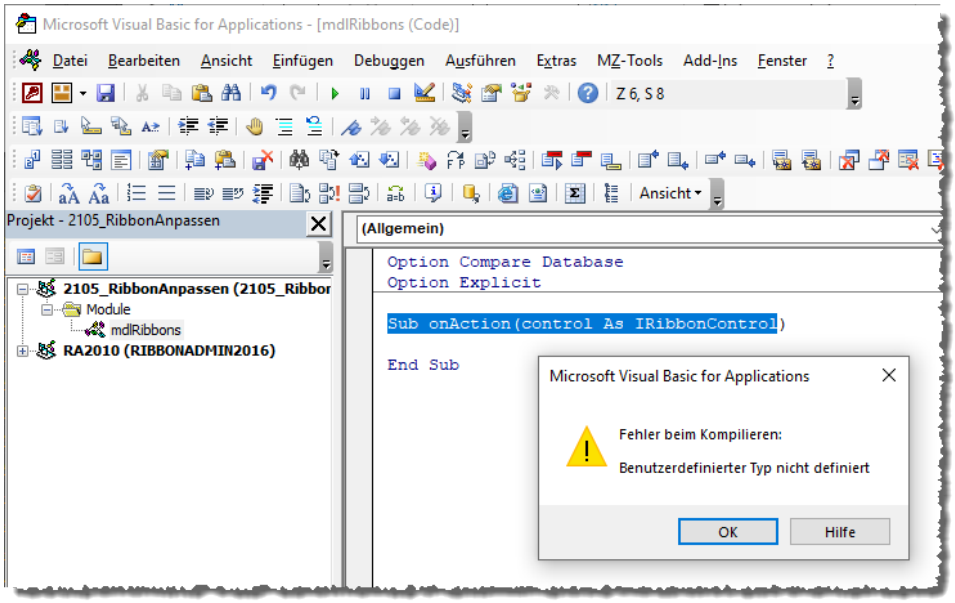


Abbildung 5.1: Fehler beim Kompilieren der Anwendung

Der fehlende Verweis heißt *Microsoft Office x.0 Object Library* und Sie können ihn über den *Verweise*-Dialog hinzufügen. Dazu wählen Sie im VBA-Editor den Menüeintrag *Extras|Verweise* aus. Hier fügen Sie den fehlenden Verweis wie in Abbildung 5.2 hinzu. Danach lässt sich die Funktion erfolgreich kompilieren.

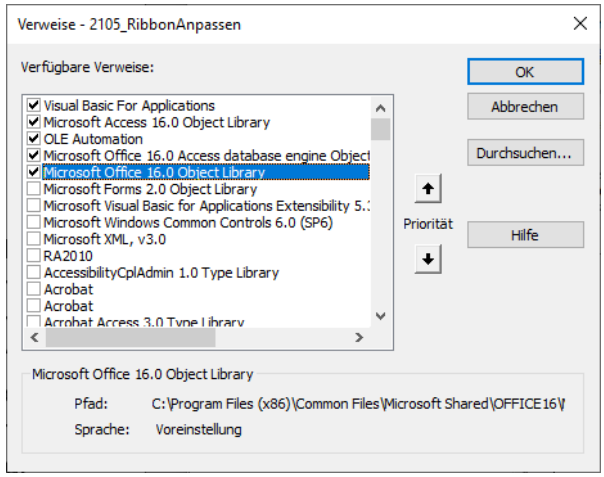


Abbildung 5.2: Hinzufügen des fehlenden *Office*-Verweises

5.1.3 Anzeige von Ribbonfehlern aktivieren

Bevor wir die Schaltfläche ausprobieren, wollen wir uns noch um eine weitere Sache kümmern. Die Office-Anwendungen können eine Meldung ausgeben, wenn eine benutzerdefinierte Ribbondefinition Fehler erzeugt. Dies müssen Sie jedoch zunächst noch aktivieren. Dazu bemühen wir wiederum die Access-Optionen, wo Sie im Bereich *Clienteinstellungen* unter *Allgemein* die Option *Fehler von Benutzeroberflächen-Add-Ins anzeigen* finden und aktivieren. Die entsprechende Stelle im Optionen-Dialog sehen Sie in Abbildung 5.3.

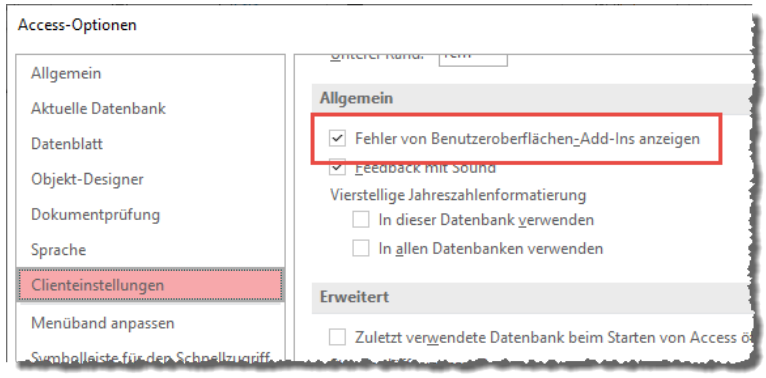


Abbildung 5.3: Aktivieren der Fehleranzeige für Ribbons

Danach zeigt Access beispielsweise eine Fehlermeldung an, wenn die Ribbondefinition fehlerhaft ist – also wenn beispielsweise das schließende Element zu einem Element fehlt oder ein Pflichtattribut nicht enthalten ist.

5.1.4 Test der Ribbon-Schaltfläche

Bevor wir die Schaltfläche ausprobieren, fügen wir der Prozedur *onAction* noch eine *MsgBox*-Anweisung hinzu:

```
Sub onAction(control As IRibbonControl)
    MsgBox "Aufruf von Steuerelement '" & control.Id & "'"
End Sub
```

Hier setzen wir eine Meldung zusammen, welche die Eigenschaft *id* des übergebenen Parameters *control* verwendet. Wie Abbildung 5.4 zeigt, liefert dies genau den Namen, den wir für das Attribut *id* des *button*-Elements festgelegt haben.

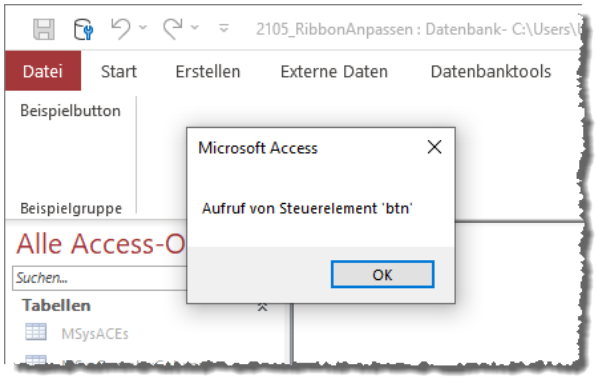


Abbildung 5.4: Meldung per *button*-Steuerelement

Das liefert uns die Möglichkeit, alle Ereignisse, die durch *button*-Elemente ausgelöst werden, in einer einzigen *onAction*-Prozedur zu verarbeiten. Wir müssen in dieser lediglich den Wert des Parameters *control.id* prüfen und beispielsweise in einer *Select Case*-Bedingung die entsprechenden Anweisungen aufrufen.

5.2 Ansätze zur Programmierung mehrerer button-Elemente

Die Wahrscheinlichkeit, dass Sie nur ein *button*-Element in Ihren Ribbondefinitionen unterbringen, ist sehr gering. Oben haben wir zunächst gezeigt, wie Sie die onAction-Prozedur zur Pro-

André Minhorst

Ribbon Programmierung

LESEPROBE

Hier finden Sie in der Vollversion noch weiteres Know-how rund um die Ribbon-Programmierung!

André Minhorst

Ribbon Programmierung

LESEPROBE

Hier finden Sie in der Vollversion noch weiteres Know-how rund um die Ribbon-Programmierung!

6 Bilder im Ribbon anzeigen

Das Ribbon bietet viele Möglichkeiten für das Definieren einer optisch ansprechenden und dennoch funktionalen Benutzeroberfläche. Auch wenn Sie ein rein textbasiertes Ribbon definieren können, werden Sie vermutlich eingebaute oder benutzerdefinierte Icons verwenden wollen – beispielsweise, um diese auf Schaltflächen anzuzeigen.

Die Beispiele zu diesem Kapitel finden Sie in der folgenden Beispieldatenbank:

`BilderImRibbonAnzeigen.accdb`

6.1 Bilder für Schaltflächen

Wenn Sie eine optisch ansprechende Benutzeroberfläche verwenden wollen, können Sie beispielsweise für die *button*-Elemente des Ribbons Icons anzeigen. Das ist allerdings nicht trivial, denn wir benötigen eine Menge weiteren Code dafür. Außerdem müssen wir eine weitere Tabelle nutzen, die allerdings normalerweise schon in Ihrer Datenbank vorhanden ist. Um ein Bild anzuzeigen, sind die folgenden Schritte nötig:

» Hinzufügen des Bildes zu der Tabelle *MSysResources* und Merken des Wertes in der Spalte *Name*

» Hinzufügen des Bildnamen als Wert des Attributs *image* des *button*-Elements

» Hinzufügen des Wertes *loadImages* zum Attribut *loadImages* des Elements *customUI*

» Anlegen eines Moduls namens *mdlRibbonImages* mit speziellen Bildfunktionen

» Hinzufügen einer Prozedur namens *LoadImages* zum Modul *mdlRibbons*

6.2 Bild zur Tabelle MSysResources hinzufügen

Die Tabelle *MSysResources* sehen wir erst, wenn wir nicht nur die Option *Systemobjekte anzeigen* aktiviert haben, sondern auch die Option *Ausgeblendete Objekte anzeigen*.

Diese finden wir an der gleichen Stelle wie die zuvor erwähnte Option. Anschließend können wir die Tabelle *MSysResources* über den Navigationsbereich öffnen und dann mit einem Doppelklick auf das @-Symbol eines neuen Datensatzes ein Bild hinzufügen – optimalerweise eines mit der Dateiendung *.png*. Die Auswahl erledigen wir mit dem nun erscheinenden Dialog *Attachments* (siehe Abbildung 6.1). Hier klicken Sie auf die Schaltfläche *Add...* und wählen im nächsten Dialog die einzufügende Bilddatei aus.

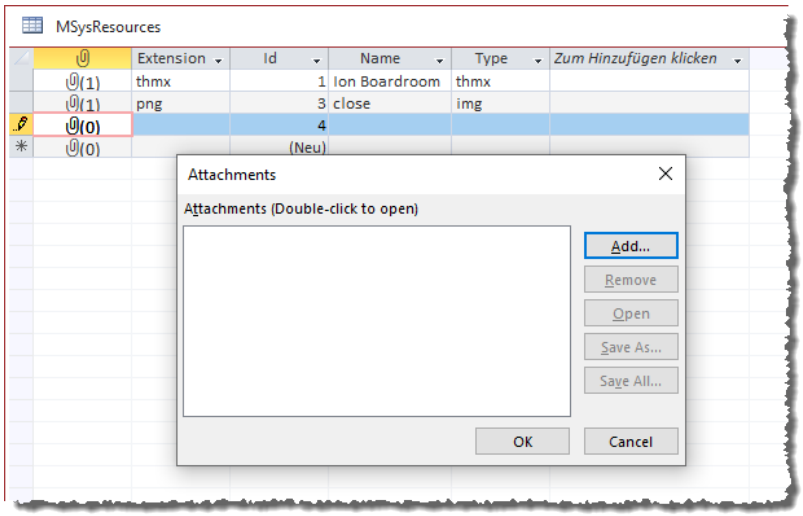


Abbildung 6.1: Hinzufügen einer Bilddatei

Größe des Bildes

Hier taucht nun die Frage auf, welche Größe die als Icon eines Ribbonelements zu verwendenden Bilddateien haben sollen. Es gibt zwei darstellbare Bildgrößen: 16x16 für normale und 32x32 für große Schaltflächen und Steuerelemente.

Am besten fügen Sie immer Bilder mit der Größe 32x32 hinzu, denn die sehen auch bei kleinen Steuerelementen, die Bilder im Format 16x16 anzeigen, gut aus, während Bilder mit der Größe 16x16 bei Verwendung in Steuerelementen mit großen Bildern pixelig aussehen.

Nach der Auswahl des Bildes

Im Dialog *Choose File* wählen Sie wie in Abbildung 6.2 die gewünschte Bilddatei aus.

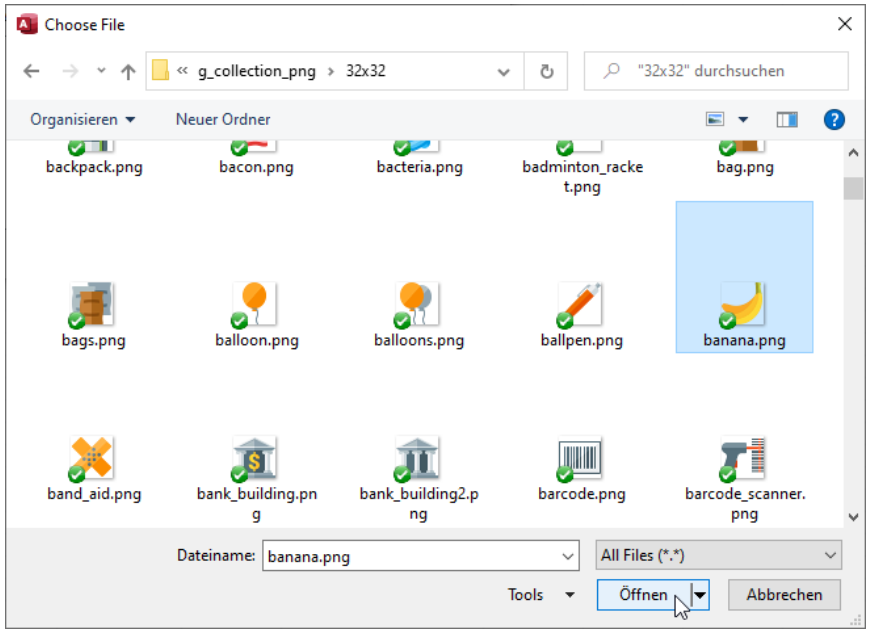


Abbildung 6.2: Auswählen der Bilddatei

Danach erscheint die Bilddatei zunächst im Dialog *Attachments* (siehe Abbildung 6.3).

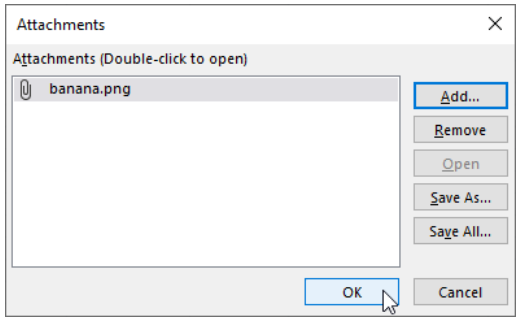


Abbildung 6.3: Die Bilddatei ist im *Attachment*-Feld gelandet.

Schließen wir den Dialog, finden wir für das @-Symbol in Klammern die Zahl *1* vor. Nun können wir noch die übrigen Felder mit den benötigten Werten füllen. Als Namen legen wir in diesem Fall beispielsweise *banana* fest, für das Feld *Extension* den Wert *png* und für *Type* den Wert *img*.

6.2.1 Bilder per Formularansicht hinzufügen

Sie können die Bilder auch noch auf eine etwas einfachere Weise zur Tabelle *MSysResources* hinzufügen. Dazu verwenden Sie die Entwurfsansicht von Formularen. Hier klicken Sie im Ribbon auf den Eintrag *Entwurf|Steuerelemente|Bild einfügen|Durchsuchen...* (siehe Abbildung 6.4).

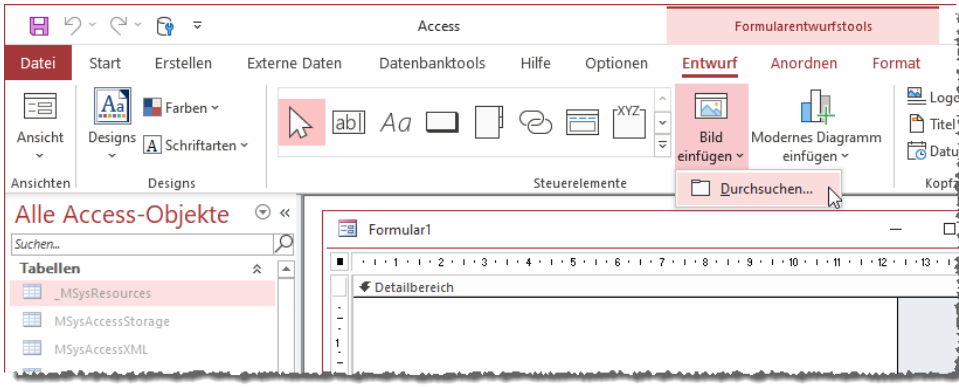


Abbildung 6.4: Öffnen des Dialogs zum Auswählen einer Bilddatei

Im nun erscheinenden *Grafik einfügen*-Dialog wählen Sie die gewünschte Bilddatei aus. Danach verwandelt der Mauszeiger sich in ein Symbol zum Einfügen eines Bildes, wenn Sie damit über den Formularentwurf fahren. Wir wollen das Bild aber nicht in das Formular einfügen, sondern in die Tabelle *MSysResources*. Aber keine Sorge: Das ist längst geschehen! Sie können den Vorgang mit der *Esc*-Taste abbrechen. Klicken Sie danach erneut auf den Ribboneintrag von eben, zeigt dieser das hinzugefügte Icon im Aufklappmenü an (siehe Abbildung 6.5).

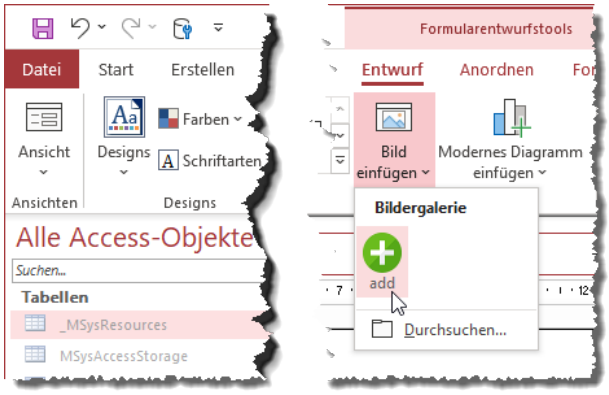


Abbildung 6.5: Das hinzugefügte Icon erscheint in der Auswahlliste für den Formularentwurf.

André Minhorst

Ribbon Programmierung

LESEPROBE

Hier finden Sie in der Vollversion noch weiteres Know-how rund um die Ribbon-Programmierung!

André Minhorst

Ribbon Programmierung

LESEPROBE

Hier finden Sie in der Vollversion noch weiteres Know-how rund um die Ribbon-Programmierung!

7 Callback-Funktionen nutzen

Weiter oben haben wir erläutert, dass das Attribut *getImage* genau wie die übrigen Attribute, deren Namen mit *get...* beginnen, das Aktualisieren von Ribbonattributen zur Laufzeit ermöglichen. Bisher haben wir gesehen, dass *getImage* direkt beim ersten Laden und Umsetzen der Ribbondefinition aufgerufen wird, um das Bildobjekt zu laden, das in dem jeweiligen Steuerelement angezeigt werden soll.

Sie können diese Attribute jedoch auch zur Angabe von Callback-Funktionen nutzen, die nach dem ersten Erstellen des Ribbons erneut aufgerufen werden. Was ist der Sinn, diese erneut aufzurufen? Damit ermöglichen wir ein sehr wichtiges Feature des Ribbons: Das Einstellen bestimmter Attribute zur Laufzeit.

Die Beispiele zu diesem Kapitel finden Sie in der folgenden Beispieldatenbank:

```
CallbackfunktionenNutzen.accdb
```

7.1 Beispiel getEnabled oder getVisible

getImage ist kein gutes Beispiel für das Anpassen von Elementen zur Laufzeit – es mag zwar vorkommen, dass man in bestimmten Situationen einmal ein Bild eines Steuerelements ersetzen möchte. Aber in der Regel behält ein Steuerelement sein Bild während einer Sitzung. Viel nützlicher sind hingegen Attribute wie *getEnabled* oder *getVisible*. Damit können Sie zur Laufzeit festlegen, ob ein Steuerelement aktiviert oder deaktiviert ist (*getEnabled*) oder ob es überhaupt angezeigt wird oder ausgeblendet (*getVisible*). Wir schauen uns am Beispiel von *getEnabled* an, wie die *get...*-Attribute funktionieren. Also fügen wir zu einer neuen Gruppe in der Ribbondefinition ein *button*-Element hinzu, für das wir das Attribut *getEnabled* mit dem Wert *getEnabled* füllen:

```
<button id="btnAktiviertDeaktiviert" label="Aktiviert/Deaktiviert"
getEnabled="getEnabled" image="barrel" size="large"/>
```

Wenn wir das Ribbon nun durch Schließen und erneutes Öffnen der Datenbank anzeigen, erscheint zunächst die Fehlermeldung aus Abbildung 7.1 – immerhin haben wir noch keine Prozedur namens *getEnabled* angelegt.

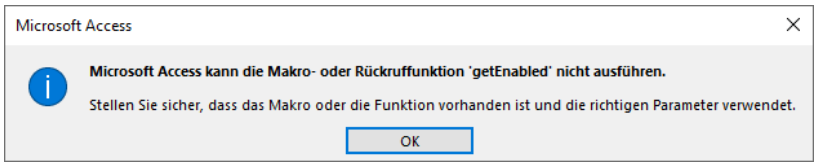


Abbildung 7.1: Die Funktion *getEnabled* konnte nicht gefunden werden.

Das holen wir nun mit der Prozedur mit der folgenden Signatur nach:

```
Sub getEnabled(control As IRibbonControl, ByRef enabled)

End Sub
```

Beim erneuten Anzeigen erscheint die betroffene Schaltfläche im nicht aktivierten Zustand (siehe Abbildung 7.2).

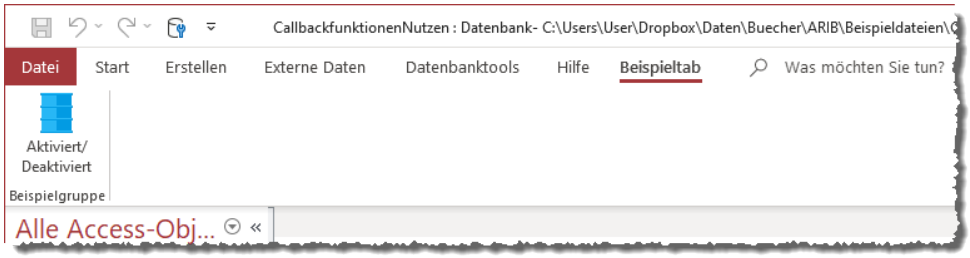


Abbildung 7.2: Eine nicht aktivierte Schaltfläche

Der Grund ist, dass der Standardwert des zweiten Parameters der Prozedur *getEnabled False* lautet. Und da wir in der Prozedur keinen Wert setzen, gibt diese für den Parameter *enabled* den Wert *False* zurück. Stellen wir *enabled* einmal auf *True* ein, wird die Schaltfläche beim nächsten Anzeigen aktiviert dargestellt:

```
Sub getEnabled(control As IRibbonControl, ByRef enabled)
    enabled = True
End Sub
```

Damit bleibt diese aber auch dauerhaft aktiviert.

7.1.1 enabled-Attribut zur Laufzeit einstellen

Welche Schritte sind nun nötig, um die Schaltfläche abhängig beispielsweise vom Wert einer Variablen zu aktivieren oder deaktivieren? Dazu benötigen wir erst einmal die Variable, die wir dann auch in der Prozedur *getEnabled* verwenden:

André Minhorst

Ribbon Programmierung

LESEPROBE

Hier finden Sie in der Vollversion noch weiteres Know-how rund um die Ribbon-Programmierung!

André Minhorst

Ribbon Programmierung

LESEPROBE

Hier finden Sie in der Vollversion noch weiteres Know-how rund um die Ribbon-Programmierung!

8 Ribbon-Steuerelemente

In den folgenden Abschnitten schauen wir uns die verfügbaren Ribbonelemente an. Dabei gehen wir davon aus, dass Sie die grundlegende Struktur einer Ribbondefinition bereits kennengelernt haben – also den hierarchischen Aufbau aus den Elementen *customUI*, *ribbon*, *tabs*, *tab*, *group* und den darin enthaltenen Elementen wie *button* und weiteren Steuerelementen, die wir in diesem Kapitel vorstellen. Davon ausgehend starten wir mit dem *tab*-Element und dem *group*-Element und schauen uns dann die eigentlichen Steuerelemente an.

Die Beispiele zu diesem Kapitel finden Sie in der folgenden Beispieldatenbank:

```
RibbonSteuerelemente.accdb
```

Die Beispiele können Sie bequem über das integrierte Formular *frmRibbonXML* betrachten. Die Abschnitte enthalten jeweils einen Hinweis, unter welchem Wert für das Feld *Ribbonname* das Beispiel in der Tabelle *USysRibbons* abgespeichert ist. Sie können mit dem Formular *frmRibbonXML* zum gewünschten Beispiel blättern und dieses dann durch einen Klick auf die Schaltfläche *Als Formularribbon festlegen* anzeigen.

8.1 Das customUI-Element

Das *customUI*-Element ist das Hauptelement, dass alle weiteren Elemente enthält. Es enthält nur zwei Attribute:

» *loadImage*: Für dieses Attribut können Sie den Namen einer Prozedur angeben, die beim Laden der Ribbondefinition aufgerufen wird. Diese Prozedur nutzt man in der Regel, um das mit dem Parameter übergebene *IRibbonUI*-Element in einer Variablen des gleichen Typs zu speichern. Über das so referenzierte Objekt können Sie beispielsweise einstellen, welches *tab*-Element angezeigt werden soll oder dass die Werte von Eigenschaften, die mit *get*...-Attributen ermittelten werden, beim Anzeigen erneut eingelesen werden. Mehr dazu erfahren Sie unter »Callback-Funktionen nutzen« ab Seite 87.

» *onLoad*: Für dieses Attribut geben Sie den Namen einer Prozedur an, die zum Einlesen von Bildern für untergeordnete Elemente verwendet werden soll. Diese Prozedur wird für jedes untergeordnete Element, das ein *image*-Attribut enthält, einmal aufgerufen. Mehr dazu lesen Sie unter »Bilder im Ribbon anzeigen« ab Seite 77.

Das *customUI*-Element kann die folgenden Unterelemente enthalten:

» *ribbon*: Enthält Elemente, die in der Ribbonleiste angezeigt werden.

» *backstage*: Hiermit können Sie den Bereich anpassen, der beim Klick auf den Registerreiter Datei angezeigt wird (weitere Informationen unter »Backstage verwenden« ab Seite 189).

» *commands*: Hiermit können Sie eingebaute Befehle umprogrammieren (weitere Informationen unter »Funktion von Ribbonelementen überschreiben« ab Seite 175).

» *contextMenus*: Wird unter Access nicht unterstützt.

8.2 Das ribbon-Element

Das *ribbon*-Element ist das übergeordnete Element für die Elemente, die als Registerkarten oder in der Schnellzugriffsleiste angezeigt werden.

Attribute des ribbon-Elements

Das *ribbon*-Element hat nur ein Attribut:

» *startFromScratch*: Legt fest, ob das Ribbon komplett neu aufgebaut werden soll oder ob die eingebauten Elemente beibehalten werden sollen.

Stellen Sie das Attribut auf den Wert *true* ein, werden alle eingebauten Elemente des Ribbons ausgeblendet (siehe Beispieldefinition *StartFromScratchLeer*):

```
<?xml version="1.0"?>
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui">
    <ribbon startFromScratch="true"/>
</customUI>
```

Das Ergebnis sieht, wenn Sie keine benutzerdefinierten Elemente hinzufügen, wie in Abbildung 8.1 aus.

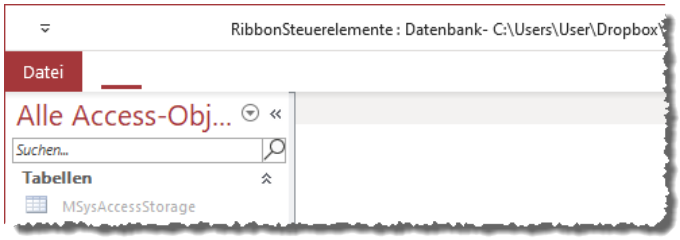


Abbildung 8.1: Ribbon ohne Elemente

Fügen Sie der Definition ein *tab*-, ein *group*- und ein *button*-Element hinzu, sieht es schon besser aus (siehe Abbildung 8.2).

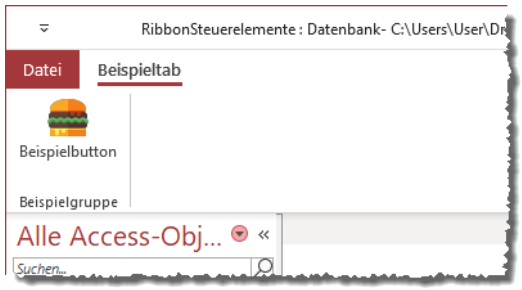


Abbildung 8.2: Ein neu aufgesetztes Ribbon

In der Beispieldatenbank finden Sie diese Definition unter dem Namen *StartFromScratchMit-Button*.

8.2.1 Unterelemente des ribbon-Elements

Das *ribbon*-Element kann die folgenden Unterelemente enthalten. Diese müssen in dieser Reihenfolge angegeben werden:

» *tabs*: Übergeordnetes Element für *tab*-Elemente

» *contextualTabs*: Übergeordnetes Element für kontextabhängige *tab*-Elemente

» *qat*: Übergeordnetes Element für die Schnellzugriffsleiste

8.3 Das tabs-Element

Das *tabs*-Element ist ein Container für die anzuzeigenden *tab*-Elemente. Das *tabs*-Element muss mindetsens ein *tab*-Element enthalten. Es enthält keine Attribute.

8.4 Das tab-Element

Wenn Sie ein *tabs*-Element definiert haben, müssen Sie darunter mindestens ein *tab*-Element hinzufügen – Sie können aber auch mehrere *tab*-Elemente darin platzieren. Dabei dient das *tab*-Element als Container für ein oder mehrere *group*-Elemente, die dann die eigentlichen Steuerelemente enthalten. Für das *tab*-Element gibt es jedoch auch einige eigene Attribute, die Sie kennen sollten.

8.4.1 Attribute des tab-Steuerelements

Das *tab*-Steuerelement hält unter anderem die folgenden Attribute bereit:

» *id*: Eindeutiger Bezeichner für benutzerdefinierte Elemente

» *idMso*: Eindeutiger Bezeichner, wenn Sie etwas mit einem eingebauten *tab*-Element erledigen wollen, zum Beispiel um dieses auszublenden

» *insertAfterMso*: Angabe des Names eines eingebauten *tab*-Elements, hinter dem Sie ein benutzerdefiniertes *tab*-Element platzieren wollen

» *insertBeforeMso*: Angabe des Names eines eingebauten *tab*-Elements, vor dem Sie ein benutzerdefiniertes *tab*-Element platzieren wollen

» *keytip*: Angabe des Buchstabens, über das dieses *tab*-Element bei Betätigen der *Alt*-Taste erreichbar sein soll

» *label*: Beschriftung für ein benutzerdefiniertes *tab*-Element

» *tag*: Möglichkeit, einen Text anzugeben, der für dieses Element per Code abgefragt werden kann – zum Beispiel in einer Callbackfunktion

» *visible*: Gibt an, ob das mit *id* oder *idMso* referenzierte *tab*-Element sichtbar sein soll. Damit können Sie zum Beispiel eingebaute *tab*-Elemente ausblenden.

» *getKeytip*: Callbackfunktion, um den Buchstaben für das Attribut *keytip* per Code zu ermitteln

» *getLabel*: Callbackfunktion, um die Beschriftung für das *tab*-Element per Code zu ermitteln

» *getVisible*: Callbackfunktion, um per Code zu ermitteln, ob das *tab*-Element sichtbar sein soll

8.4.2 Das Attribut keytip nutzen

Mit dem *keytip*-Attribut oder mit *getKeytip* können Sie festlegen, welcher Buchstabe für ein *tab*-Element (und auch für die meisten anderen Steuerelemente) angezeigt werden soll, wenn der Benutzer die *Alt*-Taste betätigt. Wenn Sie für das eingebaute Ribbon die *Alt*-Taste betätigen, blendet das Ribbon die Buchstaben wie in Abbildung 8.3 ein.

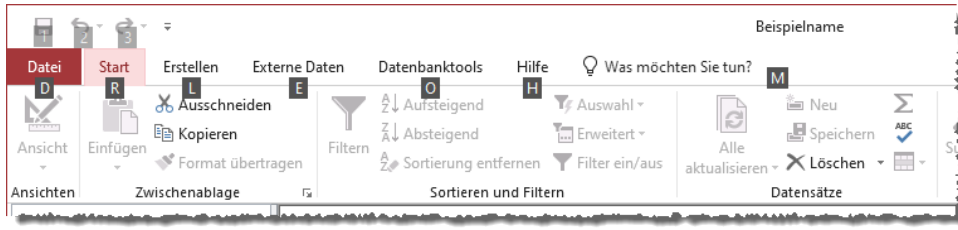


Abbildung 8.3: Diese Buchstaben erscheinen, wenn Sie die *Alt*-Taste betätigen

Klicken Sie dann beispielsweise auf die Taste *L*, wird das *Erstellen*-Tab aktiviert. Hier tauchen dann entsprechende Buchstaben für die enthaltenen Steuerelemente auf (siehe Abbildung 8.4).

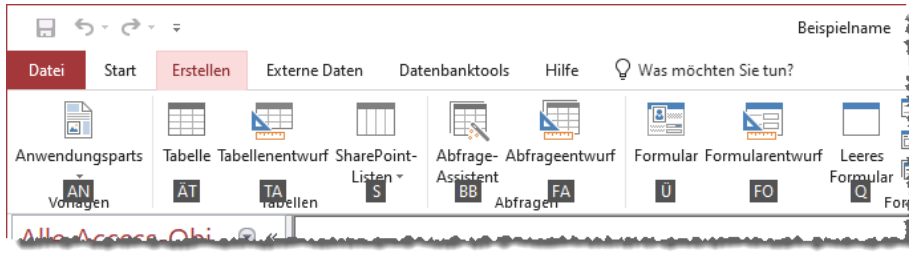


Abbildung 8.4: *keytip*-Buchstaben für die Steuerelemente eines *tab*-Elements

Um nun ein eigenes Element mit *keytip* darzustellen, müssen Sie erst einmal einen Buchstaben finden, der für die jeweilige Ebene noch nicht von eingebauten Elementen verwendet wird. Das ist für die *tab*-Elemente beispielsweise beim Buchstaben *W* der Fall. Mit der folgenden Definition betätigen Sie das *button*-Element durch die Tastenkombination *Alt + W+ W*:

```
<?xml version="1.0"?>
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui"
loadImage="loadImage">
  <ribbon startFromScratch="true">
    <tabs>
      <tab id="tab" label="Beispieltab" keytip="W">
        <group id="grp" label="Beispielgruppe">
          <button image="hamburger" label="Beispielbutton" id="btn" size="large"
                                                                    keytip="W" />
        </group>
      </tab>
    </tabs>
  </ribbon>
</customUI>
```

Das Beispiel finden Sie unter dem Namen *KeyTip* im Formular *frmRibbonXML*.

8.4.3 tab-Element positionieren mit insertAfterMso und insertBeforeMso

Vorausgesetzt, Sie wollen ein benutzerdefiniertes *tab*-Element anzeigen, ohne die eingebauten Elemente auszublenden, wollen Sie dieses vielleicht an einer bestimmten Position darstellen und nicht, wie standardmäßig vorgesehen, als letztes *tab*-Element hinter den eingebauten Elementen. Dazu können Sie die beiden Attribute *insertAfterMso* und *insertBeforeMso* verwenden.

Um beispielsweise ein benutzerdefiniertes *tab*-Element als erstes *tab*-Element zu platzieren, benötigen Sie den Namen beziehungsweise die *idMso* des ersten *tab*-Elements. Diesen finden Sie in einer Excel-Tabelle mit einer Übersicht aller Ribbonelemente. Die Exceltabelle ist im Download zu diesem Buch enthalten und heißt *AccessControls.xlsx*. Dort sind allerdings lediglich die englischen Bezeichnungen enthalten, die Sie als *idMso* verwenden können.

Daher haben wir den Inhalt dieser Exceltabelle in eine Tabelle einer Access-Datenbank geladen und überall dort, wo es möglich war, die deutschen Bezeichnungen ergänzt. Mehr Informationen erhalten Sie unter 16.1.

Wenn Sie beispielsweise ein neues *tab*-Element als erstes Element im Ribbon anzeigen wollen, dann verwenden Sie die folgende Definition für das *tab*-Element:

```
<tab id="tab" label="Beispieltab" insertBeforeMso="TabHomeAccess">
```

Das Beispiel finden Sie im Formular *frmRibbonXML* unter dem Namen *TabGanzVorn*.

8.4.4 Das Attribut visible nutzen

Das *visible*-Attribut können Sie verwenden, um eingebaute oder auch benutzerdefinierte *tab*-Elemente (oder auch alle anderen Steuerelementtypen) ein- oder auszublenden. Um ein Element auszublenden, referenzieren sie es über die Ribbondefinition und stellen sein Attribut *visible* auf den Wert *false* ein.

Wenn Sie beispielsweise das *tab*-Element *Datenbanktools* ausblenden wollen, referenzieren Sie das Element wie folgt und stellen sein Attribut *visible* auf *false* ein:

```
<?xml version="1.0"?>
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui">
  <ribbon>
    <tabs>
      <tab idMso="TabDatabaseTools" visible="false" />
    </tabs>
  </ribbon>
```

```
</customUI>
```

Diese Ribbondefinition finden Sie im Formular *frmRibbonXML* unter dem Namen *TabAusblenden*.

8.4.5 Das Attribut label zum Umbenennen nutzen

Das Sie das *label*-Attribut bei benutzerdefinierten Elementen nutzen, um diesen eine Beschriftung hinzuzufügen, ist bekannt. Sie können damit aber auch die Beschriftungen eingebauter Ribbonelemente anpassen. Dazu referenzieren Sie das Element, dessen Beschriftung Sie ändern möchten, über die *idMso* und stellen das Attribut *label* für dieses auf den gewünschten Wert ein:

```
<?xml version="1.0"?>
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui">
  <ribbon>
    <tabs>
      <tab idMso="TabHomeAccess" label="Home" />
    </tabs>
  </ribbon>
</customUI>
```

Das Beispiel finden Sie im Formular *frmRibbonXML* unter dem Namen *TabUmbenennen*.

8.4.6 Unterelemente des tab-Elements

Das *tab*-Element kann lediglich *group*-Elemente enthalten.

8.5 Das group-Element

Das *group*-Steuerelement dient dazu, die Steuerelemente eines *tab*-Elements zu gruppieren. Sie können einem *tab*-Element nicht direkt Steuerelemente zuweisen, sondern diese müssen sich immer innerhalb eines *group*-Elements befinden.

8.5.1 Attribute des group-Elements

Das *group*-Element verwendet die gleichen Attribute wie das *tab*-Element und zusätzlich die folgenden Attribute:

» *autoScale*: Gibt mit dem Wert *true* an, dass die Elemente innerhalb einer Gruppe verkleinert dargestellt werden sollen, bevor die Gruppe selbst minimiert wird.

» *centerVertically*: Gibt an, ob die enthaltenen Steuerelemente vertikal zentriert werden sollen.

» *getImage*: Gibt die Callbackfunktion an, die das für ein minimiertes *group*-Element anzuzeigende Bild ermitteln soll.

» *getScreentip*: Hat beim *group*-Element keine Auswirkung.

» *getSupertip*: Hat beim *group*-Element keine Auswirkung.

» *image*: Gibt den Namen des Bildes an, das für ein minimiertes *group*-Element angezeigt werden soll.

» *imageMso*: Gibt den *imageMso*-Wert für das Bild eines eingebauten Steuerelements an, das für ein miniminertes *group*-Element angezeigt werden soll.

» *screentip*: Hat beim *group*-Element keine Auswirkung.

» *supertip*: Hat beim *group*-Element keine Auswirkung.

8.5.2 Unterelemente des group-Elements

Dem *group*-Element können Sie die folgenden Steuerelemente unterordnen:

» *box*: siehe »Steuerelemente zum Strukturieren des Ribbons« ab Seite 109

» *button*: siehe »Das button-Steuerelement« ab Seite 106

» *buttonGroup*: siehe »Steuerelemente zum Strukturieren des Ribbons« ab Seite 109

» *checkBox*: siehe »Das checkBox-Element« ab Seite 112

» *comboBox*: siehe »Das comboBox-Element« ab Seite 119

» *control*: siehe »Das control-Element« ab Seite 148

» *dialogBoxLauncher*: siehe »Das dialogBoxLauncher-Element« ab Seite 150

» *dropDown*: siehe »Das dropDown-Element« ab Seite 125

» *dynamicMenu*: siehe »Das dynamicMenu-Element« ab Seite 135

» *editBox*: siehe »Das editBox-Steuerelement« ab Seite 108

» *gallery*: siehe »Das gallery-Element« ab Seite 141

» *labelControl*: siehe »Das labelControl-Element« ab Seite 115

» *menu*: siehe »Das menu-Element« ab Seite 133

» *separator*: siehe »Steuerelemente zum Strukturieren des Ribbons« ab Seite 109

» *splitButton*: siehe »Das splitButton-Element« ab Seite 146

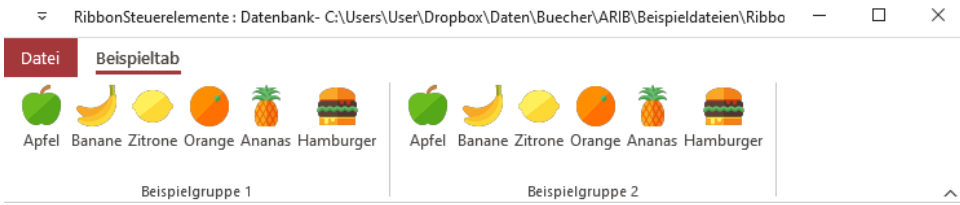
» *toggleButton*: siehe »Das toggleButton-Element« ab Seite 116

8.5.3 Image für minimierte Gruppen definieren

Wenn Ihr Ribbon recht viele Elemente enthält und deshalb sehr breit ist, kann es sein, dass Access nicht alle Elemente vollständig anzeigen kann. Dann werden von rechts nach links die Gruppen minimiert. Wir haben dazu zwei Gruppen mit jeweils sechs Schaltflächen wie folgt angelegt:

```
<?xml version="1.0"?>
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui"
loadImage="loadImage">
  <ribbon startFromScratch="true">
    <tabs>
      <tab id="tab" label="Beispieltab">
        <group id="grp1" label="Beispielgruppe 1">
          <button id="btn1" label="Apfel" image="apple" size="large"/>
          <button id="btn2" label="Banane" image="banana" size="large"/>
          <button id="btn3" label="Zitrone" image="lemon" size="large"/>
          <button id="btn4" label="Orange" image="orange" size="large"/>
          <button id="btn5" label="Ananas" image="pineapple" size="large"/>
          <button id="btn6" label="Hamburger" image="hamburger" size="large"/>
        </group>
        <group id="grp2" label="Beispielgruppe 2">
          <button id="btn7" label="Apfel" image="apple" size="large"/>
          <button id="btn8" label="Banane" image="banana" size="large"/>
          <button id="btn9" label="Zitrone" image="lemon" size="large"/>
          <button id="btn10" label="Orange" image="orange" size="large"/>
          <button id="btn11" label="Ananas" image="pineapple" size="large"/>
          <button id="btn12" label="Hamburger" image="hamburger" size="large"/>
        </group>
      </tab>
    </tabs>
  </ribbon>
</customUI>
```

Ohne weiteres Zutun sieht das wie in aus.



Zwei Gruppen mit ausreichend Platz

Wenn wir das Ribbon nun über die Breite des Access-Fensters schmaler machen, sieht das wie in Abbildung 8.5 aus. Access fügt hier ein Platzhalter-Icon ein.

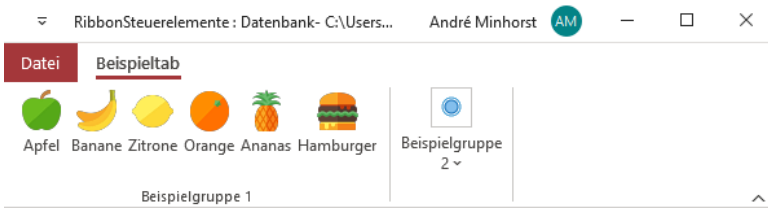


Abbildung 8.5: Eine minimierte Gruppe auf der rechten Seite

Reduzieren wir die Breite weiter, werden beide Gruppen nur noch als Icon angezeigt.

Icon für die minimierte Gruppe festlegen

Sie können nun als Erstes ein Icon festlegen, das statt dieses Platzhalters erscheinen soll. Dazu nutzen Sie die Eigenschaft *image* des *group*-Elements. Für diese hinterlegen Sie wie üblich den Namen des Icons aus der Tabelle *MSysResources*, das statt des Platzhalters angezeigt werden soll:

```
<group id="grpButtonGroup" label="Beispiel ButtonGroup"
      image="lemon">
    ...
</group>
```

Das Ergebnis sieht wie in Abbildung 8.6 aus.

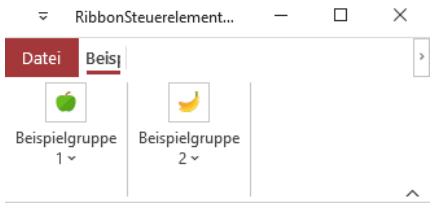


Abbildung 8.6: Ersetzen des Platzhalter-Icons durch ein benutzerdefiniertes Icon

Die Ribbondefinitionen für dieses Beispiel finden Sie im Formular *frmRibbonXML* unter den Namen *GroupOhneImage* und *GroupMitImage*.

8.5.4 Einstellen, ob Elemente einer Gruppe minimiert werden dürfen

Wenn Sie keine weiteren Schritte unternehmen, werden die Gruppen im Ribbon automatisch von rechts nach links minimiert, wenn Sie die Breite fortlaufend verkleinern. Sie können allerdings noch eine Einstellung vornehmen, welche dies zunächst verhindert und erst einmal die Steuerelemente verkleinert, bevor die Gruppe nur noch als Icon angezeigt wird. Diese Einstellung gehört ebenfalls zum *group*-Element und lautet *autoScale*. Diese haben wir im folgenden Beispiel für die linke Gruppe auf den Wert *true* eingestellt und für die rechte Gruppe auf *false* (wobei dies der Standardwert ist und Sie die Eigenschaft auch hätten weglassen können):

```
<group id="grp1" label="Beispielgruppe 1 mit autoScale" image="apple" autoScale="true">
  ...
</group>
<group id="grp2" label="Beispielgruppe 2" image="banana" autoScale="false">
  ...
</group>
```

Dies führt dazu, dass die Schaltflächen zunächst mit großen Icons dargestellt werden (siehe Abbildung 8.7).

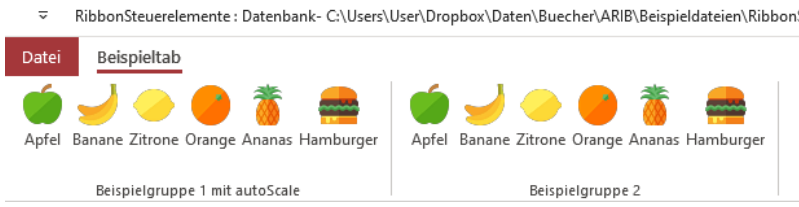


Abbildung 8.7: Icons in voller Größe

Verringern Sie nun die Breite des Ribbons, werden die Elemente der linken Gruppe zunächst verkleinert, aber noch mit Beschriftung angezeigt (siehe Abbildung 8.8).

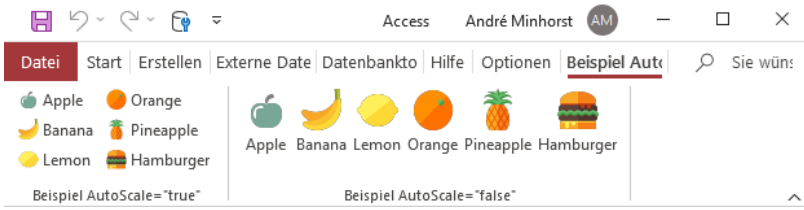


Abbildung 8.8: Schaltflächen mit kleinen und großen Icons

Verringern wir die Breite des Ribbons, werden die Elemente in der Gruppe mit *autoScale="true"* in kleine Symbole umgewandelt (siehe Abbildung 8.9).

Abbildung 8.9: Die Icons der linken Gruppe werden mit dem Ribbon verkleinert.

Erst danach wird die rechte Gruppe, für die *autoScale* den Wert *false* aufweist, minimiert – dies allerdings direkt in Form des Gruppen-Icons wie zuvor beschrieben (siehe Abbildung 8.10). Es macht also Sinn, für alle *group*-Elemente die Einstellung *autoScale* auf *true* einzustellen.

Abbildung 8.10: Auch die Beschriftungen der Schaltflächen verschwinden.

Das Beispiel hierzu finden Sie im Formular *frmRibbonXML* unter dem Namen *GroupMitImage-Skalierung*.

8.5.5 Gruppenelemente vertikal zentrieren

Wenn eine Gruppe nur ein Element in einem Abschnitt enthält, dann wir dieser in der Regel oben angeordnet. Wir haben im folgenden Beispiel jeweils sechs *button*-Elemente in je einem von zwei *group*-Elementen angeordnet und diesen *separator*-Elemente hinzugefügt (dieses stellen wir weiter unten vor). Die erste Gruppe erhält für die Eigenschaft *centerVertically* den Wert *false*, die zweite den Wert *true*:

```
<group id="grp1" label="Beispielgruppe ohne centerVertically" image="apple" 7
                                          autoScale="true" centerVertically="false">
  <button id="btn1" label="Apfel" image="apple"/>
  <separator id="sep1"/>
  <button id="btn2" label="Banane" image="banana"/>
  <button id="btn3" label="Zitrone" image="lemon" />
  <separator id="sep2"/>
  <button id="btn4" label="Orange" image="orange" />
  <button id="btn5" label="Ananas" image="pineapple" />
  <button id="btn6" label="Hamburger" image="hamburger" />
</group>
<group id="grp2" label="Beispielgruppe mit centerVertically" image="apple" 7
                                           autoScale="true" centerVertically="true">
  <button id="btn7" label="Apfel" image="apple"/>
  <separator id="sep3"/>
  <button id="btn8" label="Banane" image="banana"/>
  <button id="btn9" label="Zitrone" image="lemon" />
  <separator id="sep4"/>
  <button id="btn10" label="Orange" image="orange" />
  <button id="btn11" label="Ananas" image="pineapple" />
  <button id="btn12" label="Hamburger" image="hamburger" />
</group>
```

Das Ergebnis sieht wie in Abbildung 8.11 aus.

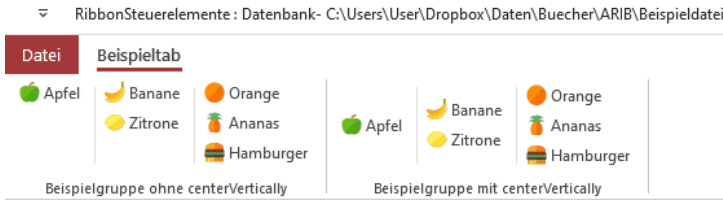


Abbildung 8.11: *group*-Elemente mit verschiedenen Werten für das *centerVertically*-Attribut

8.6 Das button-Steuerelement

Das *button*-Element haben wir bereits in einigen vorherigen Abschnitten ausführlich beschrieben (siehe »button-Element hinzufügen« ab Seite 54 und »Programmieren von Buttons mit VBA« ab Seite 69). Daher schauen wir uns hier nur die verfügbaren Attribute für das *button*-Element und ihre Funktion an und liefern Beispiele für den Einsatz der noch nicht beschriebenen Attribute *screentip*, *supertip*, *showImage* und *showLabel*.

8.6.1 Attribute des button-Elements

Das *button*-Element verwendet einige der zuvor bereits vorgestellten Attribute. Nachfolgend finden Sie die wichtigsten dieser bereits vorgestellten Elemente und einige, die wir noch nicht vorgestellt haben:

» *description*: Wird im *button*-Element nicht angezeigt.

» *enabled*: Gibt an, ob das *button*-Element aktiviert ist.

» *onAction*: Legt eine Prozedur fest, die beim Anklicken des *button*-Elements ausgeführt wird.

» *screentip*: Erlaubt die Angabe eines Textes, der beim Verharren mit der Maus über dem *button*-Element erscheint.

» *showImage*: Gibt an, ob ein Image für das *button*-Element angezeigt werden soll.

» *showLabel*: Gibt an, ob eine Beschriftung für das button-Element angezeigt werden soll.

» *size*: Gibt die Größe des *button*-Elements an. Es gibt die Werte *normal* (Standardwert) und *large*.

» *supertip*: Erlaubt die Angabe eines umfangreicheren Textes, der beim Verharren mit der Maus über dem *button*-Element erscheint.

» *tag*: Erlaubt die Angabe eines Textes, der beim Aufrufen der *onAction*- und der *get...*-Prozeduren als Eigenschaft des als Parameter übergebenen *IRibbonControl*-Elements ausgewertet werden kann.

» *getDescription*: Wird im *button*-Element nicht angezeigt.

» *getEnabled*: Callbackfunktion, um den Wert für das Attribut *enabled* einzulesen.

» *getImage*: Callbackfunktion, um den Wert für das Attribut *image* einzulesen.

» *getScreentip*: Callbackfunktion, um den Wert für das Attribut *screentip* einzulesen.

» *getShowImage*: Callbackfunktion, um den Wert für das Attribut *showImage* einzulesen.

» *getShowLabel*: Callbackfunktion, um den Wert für das Attribut *showLabel* einzulesen.

» *getSize*: Callbackfunktion, um den Wert für das Attribut *size* einzulesen.

» *getSupertip*: Callbackfunktion, um den Wert für das Attribut *supertip* einzulesen.

8.6.2 button-Informationen mit screentip und supertip

Mit den beiden Attributen *screentip* und *supertip* können Sie den Text festlegen, der beim Verharren mit der Maus über einem *button*-Element eingeblendet wird. Ein Beispiel finden Sie im folgenden Element, das gleichzeitig zeigt, dass das Attribut description bei normalen button-Elementen nicht berücksichtigt wird:

```
<button id="btn1" label="Apfel" image="apple" size="large" description="Eine descripti-
on." screentip="Ein screentip." supertip="Ein supertip&#13; mit Zeilenumbruch."/>
```

Mit der Zeichenfolge *
* können Sie für den Text des Attributes *supertip* Zeilenumbrüche einfügen. Das Ergebnis sehen Sie in Abbildung 8.12.

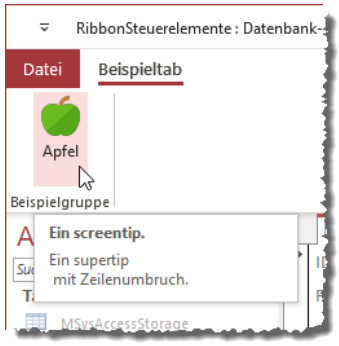


Abbildung 8.12: Der Screentip und der Supertip eines *button*-Elements

Im Formular *frmRibbonXML* der Beispieldatenbank finden Sie dieses Beispiel unter dem Namen *buttonScreentipSupertip*.

8.6.3 Anzeige von Bild und Beschriftung beeinflussen

Die Attribute zum Anzeigen von Bildern und Beschriften namens *image* und *label* beziehungsweise die entsprechenden Callbackfunktionen *getImage* und *getLabel* haben Sie bereits kennengelernt. Es gibt jedoch noch zwei Attribute für die *button*-Schaltfläche, mit der Sie einstellen können, ob Bild oder Beschriftung überhaupt angezeigt werden sollen. Diese heißen *showLabel* und *showImage*.

Diese beiden Attributen funktionieren nur, wenn das Attribut *size* nicht auf den Wert *large* eingestellt ist, also wenn es entweder den Wert *normal* aufweist oder nicht angegeben wird:

```
<button id="btn1" label="Apfel" image="apple" size="normal" showLabel="false"/>
<button id="btn2" label="Zitrone" image="lemon" size="normal" showImage="false"/>
```

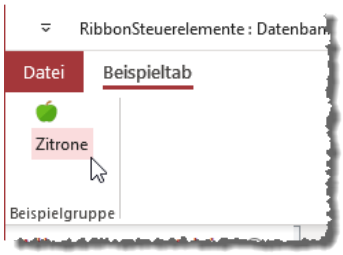


Abbildung 8.13: Beispiele für die Attribute *showImage* und *showLabel* im *button*-Element

Das Beispiel finden Sie in der Beispieldatenbank im Formular *frmRibbonXML* unter dem Namen *buttonShowImageShowLabel*.

8.7 Das editBox-Steuerelement

Nachdem wir uns mit dem Einstellen von Werten für Ribbonelemente beschäftigt haben, wird es Zeit, dass wir uns die Steuerelemente ansehen, mit denen wir Informationen vom Benutzer abfragen können, die über das Anklicken einer Schaltfläche hinausgehen. Das erste Beispiel dabei ist das *editBox*-Steuerelement. Dieses entspricht einem Textfeld, bietet allerdings längst nicht so viele Möglichkeiten. So kann der Benutzer einen Text eingeben und die Eingabe abschließen, um die Callbackprozedur für das Attribut *onChange* auszulösen. Ein *editBox*-Steuerelement mit den wichtigsten Attributen definieren wir wie folgt:

```
<editBox id="txtBeispiel" onChange="onChange" getText="getText" label="Beispieltext:"  />
```

Wenn das *editBox*-Steuerelement angezeigt wird, können Sie mit dem Callbackattribut *getText* den Text ermitteln, der initial angezeigt werden soll. Dazu hinterlegen wir die folgende Callbackprozedur, die einen Verweis auf das aufrufende Steuerelement übergibt und mit dem Parameter *text* den anzuzeigenden Text entgegennimmt. Wir stellen diesen hier auf den Wert *Beispieltext* ein:

```
Sub getText(control As IRibbonControl, ByRef text)
    text = "Beispieltext"
End Sub
```

Dies sieht nach dem erneuten Laden der Beispieldatenbank wie in Abbildung 8.14 aus.

André Minhorst

Ribbon Programmierung

LESEPROBE

Hier finden Sie in der Vollversion noch weiteres Know-how rund um die Ribbon-Programmierung!

André Minhorst

Ribbon Programmierung

LESEPROBE

Hier finden Sie in der Vollversion noch weiteres Know-how rund um die Ribbon-Programmierung!

9 Ribbons für Formulare und Berichte

Neben einem Anwendungsribbon können Sie Ribbondefinitionen festlegen, die Access in Zusammenhang mit Formularen und Berichten anzeigt. Bei Formularen macht dies nur Sinn, wenn Sie dieses nicht als modalen Dialog öffnen, also nicht etwa mit *DoCmd.OpenForm "Formularname", WindowMode:=acDialog*. In dem Fall kann man bekanntlich nicht auf andere Elemente der Benutzeroberfläche der Anwendung außer auf das Formular selbst zugreifen; ein Ribbon würde hier also keinen Sinn ergeben.

Schauen wir uns zunächst an, welche Möglichkeiten sich beim Arbeiten mit Formular-Ribbons auftun; die Ribbons von Berichten sind anschließend schnell abgehandelt.

9.1 Ribbons für Formulare

Um ein Ribbon in Zusammenhang mit einem Formular anzuzeigen, gibt es verschiedene Möglichkeiten. Davon ausgehend, dass die Anwendung ein eigenes Ribbon besitzt, das angezeigt wird, wenn kein Objekt geöffnet ist, können Sie

» eine Ribbondefinition erstellen, die mit dem Attribut *startFromScratch="true"* dafür sorgt, dass Access nur die Elemente des Formular-Ribbons anzeigt, oder

» die vorhandenen Elemente beibehalten und eines oder mehrere weitere Tabs einblenden, welche die für die Arbeit mit dem Formular wichtigen Steuerelemente enthalten. Hier müssen Sie im Gegensatz zur ersten Variante dafür Sorge tragen, dass Access die formularspezifischen Elemente auch einblendet, also zum entsprechenden *tab*-Element wechselt.

Möglicherweise fragen Sie sich, warum man überhaupt Funktionen im Ribbon verfügbar machen sollte, die sich auf das aktuell angezeigte Formular beziehen. Nun, das Ribbon weist eine gewisse Höhe auf und nimmt gerade auf den modernen Breitbildschirmen eine beträchtliche Menge Platz in Anspruch. Viele Formulare enthalten im oberen oder unteren Bereich eine Reihe Steuerelemente wie die *OK*- oder *Schließen*-Schaltfläche. Die könnte man leicht in das Ribbon verschieben und den im Formular eingesparten Platz anderweitig verwenden. Wenn Sie bereits fertige Access-Anwendungen haben, entdecken Sie möglicherweise noch andere Gelegenheiten, Steuerelemente vom Formular in das Ribbon zu verpflanzen – und vielleicht gelingt es auf diese Weise sogar, Formulare mit vielen Steuerelementen ein wenig aufzuräumen und den Bedienkomfort zu erhöhen.

9.1.1 Nur Formularfunktionen im Ribbon

Wenn Sie ein Formular zur Bearbeitung von Daten öffnen, ist es sinnvoll, nicht gleichzeitig auch noch andere Formulare anzuzeigen, erst recht nicht, wenn zwei oder mehr Formulare die gleichen oder voneinander abhängige Daten anzeigen. Sie müssen dann dafür Sorge tragen, dass beim Ändern von Daten auch die in den anderen Formularen angezeigten Daten geändert werden. Das ist aus mehreren Gründen keine gute Idee: Erstens könnten Sie den Benutzer damit verwirren, und zweitens erhöht dies nicht unbedingt die Wartungsfreundlichkeit einer Anwendung. Bei dieser Taktik ersparen Sie sich auch unnötige Komplikationen, wenn Sie das Ribbon für ein Formular konzipieren: Sie bringen darin einfach die für die Arbeit mit dem Formular nötigen Funktionen unter und blenden alle vorhandenen Elemente einfach aus.

9.1.2 Allgemeine und Formularfunktionen im Ribbon

Wenn Sie eine Anwendung betreiben, die das gleichzeitige Öffnen von Fenstern erlaubt, werden Sie eine ganz andere Strategie beim Aufbau der Ribbons verfolgen: Zunächst wird wohl das Anwendungsribbon, über das Sie Elemente wie Formulare und Berichte öffnen, auch beim Einblenden von Tabs aus formularbezogenen Ribbondefinitionen erhalten bleiben.

Damit das Formularribbon nicht nur angezeigt, sondern auch noch direkt beim Öffnen des Formulars eingeblendet wird, können Sie das *contextualTabs*-Element mit einem speziellen Parameter verwenden. Der Hintergrund hiervon ist, dass ein über ein Formularribbon definiertes *tab*-Element zwar zum Ribbon hinzugefügt, aber nicht sofort aktiviert wird:

```
<tab id="tab" label="Beispieltab" insertBeforeMso="TabHomeAccess">
```

Das heißt, dieses *tab*-Element steht zwar ganz links bei den *tab*-Reitern, aber beim Öffnen wird dennoch das eingebaute *tab*-Element *TabHomeAccess* aktiviert. Abhilfe schaffen das *contextualTabs*- und das *tabSet*-Element in Kombination mit dem Parameter *TabSetFormReportExtensibility*:

```
<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui">
  <ribbon>
    <contextualTabs>
      <tabSet idMso="TabSetFormReportExtensibility">
        <tab id="tab" label="Beispieltab">
          <group id="grp" label="Beispielgruppe">
            <button id="cmdBeispiel" ... />
          </group>
        </tab>
      </tabSet>
    </contextualTabs>
```

André Minhorst

Ribbon Programmierung

LESEPROBE

Hier finden Sie in der Vollversion noch weiteres Know-how rund um die Ribbon-Programmierung!

André Minhorst

Ribbon Programmierung

LESEPROBE

Hier finden Sie in der Vollversion noch weiteres Know-how rund um die Ribbon-Programmierung!

10 Weitere Programmiertechniken

In diesem Kapitel stellen wir weitere Programmiertechniken rund um das Ribbon vor.

10.1 Ribbonobjekt fehlerresistent speichern

Wenn während der Benutzung einer Anwendung ein unbehandelter Laufzeitfehler auftritt (einfach gesagt: ein Fehler, der eine Fehlermeldung von Access hervorruft), werden alle Objektvariablen geleert, also auch die im obigen Beispiel oft verwendete Variable *objRibbon* zum Speichern des aktuellen Ribbons. Wir schauen uns dieses Verhalten einmal an. Dazu legen wir ein Ribbon mit der folgenden Definition an (siehe Tabelle *USysRibbons* unter *RibbonNichtPersistent* und Modul *mdlRibbons*):

```
<?xml version="1.0"?>
<customUI ... onLoad="onLoad_NichtPersistentesRibbon">
  <ribbon>
    <tabs>
      <tab id="tab" label="Beispiel Nicht Persistentes Ribbon" 7
                                                    insertBeforeMso="TabHomeAccess">
        <group id="grp0" label="Beispiel Nicht Persistentes Ribbon">
          <labelControl id="lbl" getLabel="lbl_getLabel" />
        </group>
      </tab>
    </tabs>
  </ribbon>
</customUI>
```

Für das *getLabel*-Attribut haben wir die folgende Prozedur hinterlegt, die nach dem Invalidieren jeweils die aktuelle Uhrzeit liefert:

```
Sub lbl_getLabel(control As IRibbonControl, label)
    label = Now
End Sub
```

Außerdem ist das Callbackattribut *onLoad* des *customUI*-Elements wichtig. Dieses löst die folgende Prozedur aus. Darin speichert sie die aktuelle Ribbondefinition in der Variablen *objRibbon_NichtPersistent* und öffnet ein Formular namens *frmNichtPersistentesRibbon*:

```
Public objRibbon_NichtPersistent As IRibbonUI


Sub onLoad_NichtPersistentesRibbon(ribbon As IRibbonUI)
```

```
    Set objRibbon_NichtPersistent = ribbon
    DoCmd.OpenForm "frmNichtPersistentesRibbon"
End Sub
```

Das Formular enthält zwei Schaltflächen. Wir betätigen zunächst einmal die zweite, welche die folgende Prozedur auslöst:

```
Private Sub cmdRibbonInvalidieren_Click()
    objRibbon_NichtPersistent.Invalidate
End Sub
```

Damit werden alle Elemente des Ribbons, die *get...*-Attribute enthalten, ungültig gemacht, sodass die *get...*-Prozeduren beim nächsten Anzeigen erneut aufgerufen werden. Das ist hier beim Element *labelControl* der Fall, dessen Attribut getLabel wir definiert haben. Dieses zeigt immer die aktuelle Zeit an, sodass wir erkennen können, ob das Objekt *objRibbon_NichtPersistent* gefüllt ist. Dann klicken wir die erste Schaltfläche an, welche die folgende Prozedur auslöst:

```
Private Sub cmdUnbehandeltenFehlerAusloesen_Click()
    Debug.Print 1 / 0
End Sub
```

Damit lösen wir einen Fehler aus, dessen Fehlermeldung wir mit einem Klick auf die Schaltfläche *Beenden* quittieren (siehe Abbildung 10.1).

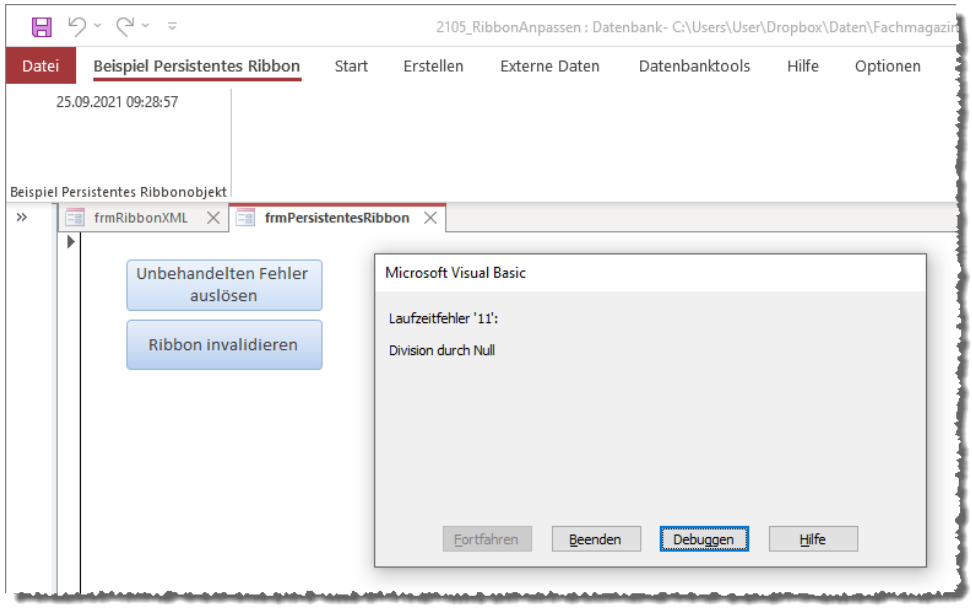


Abbildung 10.1: Unbehandelten Fehler auslösen

So erhalten wir einen unbehandelten Laufzeitfehler, der dafür sorgt, dass unter anderem die Variable *objRibbon_Persistent* geleert wird. Dass dies tatsächlich der Fall ist, prüfen wir durch einen erneuten Klick auf die zweite Schaltfläche. Wir erhalten dann den Fehler aus Abbildung 10.2.

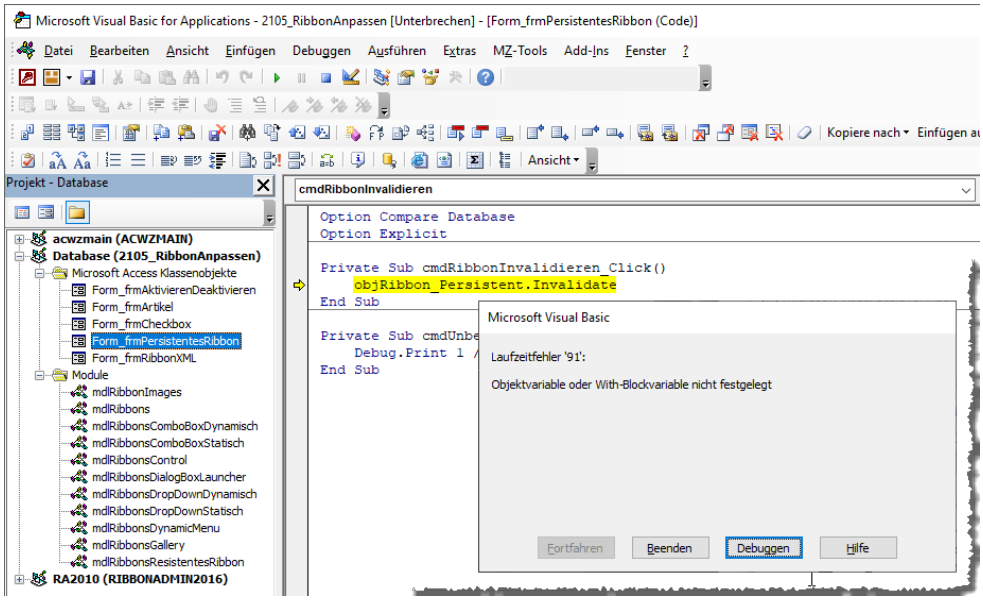


Abbildung 10.2: Fehler beim Versuch, ein geleertes IRibbonUI-Objekt zu nutzen

Den Code zu diesem Beispiel finden Sie in der Beispieldatenbank *WeiterеProgrammiertechniken_NichtFehlerresistent.accdb*.

Diesem Fehler können Sie auf zwei Arten vorbeugen:

» Sie gestalten Ihren Code so, dass keine unbehandelten Fehler auftreten können.

» Sie verwenden die nachfolgende Methode, um das Ribbon persistent und fehlerresistent zu machen.

Optimalerweise verwenden Sie eine Kombination aus beiden Ansätzen.

10.1.1 Ribbon als TempVar speichern

Access liefert ab der Version 2007 mit der *TempVars*-Auflistung ein Objekt, in dem Sie Objekte so speichern können, dass ihr Inhalt bei unbehandelten Laufzeitfehlern nicht verloren geht. Die *TempVars*-Auflistung ist immer einsatzbereit, Sie müssen sie nicht wie ein *Collection*-Objekt oder ähnliche erst deklarieren und instanziieren. In der im *onLoad*-Attribut angegebenen Callback-Funktion speichern Sie den Inhalt des übergebenen *ribbon*-Parameters zunächst wie gewohnt in *objRibbon_Persistent*. Anschließend fügen Sie seinen Inhalt einem neuen *TempVars*-Element hinzu. Genau genommen ist es nicht der Inhalt der Objektvariablen selbst, sondern nur der Pointer darauf, den Sie über die Funktion *ObjPtr* (ein verborgenes Element der VBA-Bibliothek)

ermitteln. Außerdem öffnet auch diese Prozedur wieder ein Formular zum Testen, diesmal namens *frmPersistentesRibbon*:

```
Sub onLoad_PersistentesRibbon(ribbon As IRibbonUI)
    Set objRibbon_Persistent = ribbon
    TempVars.Add "objRibbonPersistentPtr", ObjPtr(ribbon)
    DoCmd.OpenForm "frmPersistentesRibbon"
End Sub
```

In diesem Formular bieten wir wieder zwei Schaltflächen an. Die erste löst wieder den unbehandelten Laufzeitfehler aus:

```
Private Sub cmdUnbehandeltenFehlerAusloesen_Click()
    Debug.Print 1 / 0
End Sub
```

Dies leert auch zuverlässig die Objektvariable, wenn Sie die Fehlermeldung mit einem Klick auf *Beenden* statt auf *Debuggen* quittieren (siehe Abbildung 10.3).

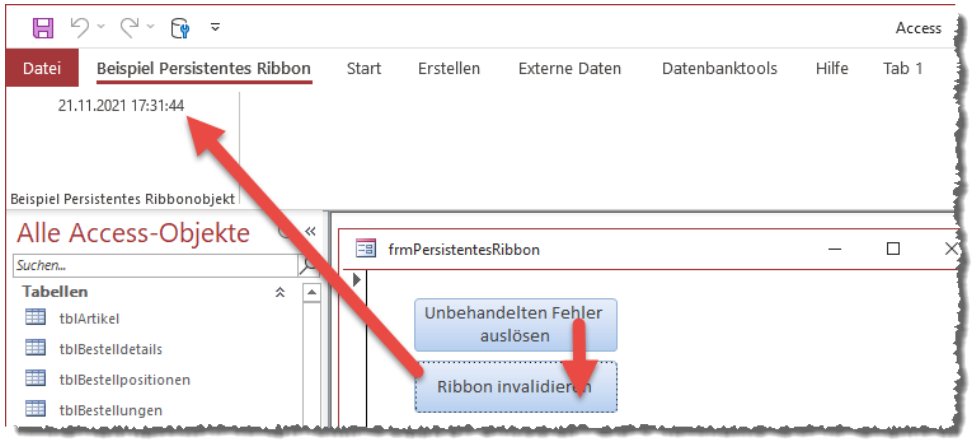


Abbildung 10.3: Bei unbehandeltem Fehler wird die Ribbonvariable direkt wieder hergestellt.

Das können Sie im Direktbereich des VBA-Editors (*Strg + G*) mit der folgenden Anweisung prüfen:

```
? objRibbon_Persistent Is Nothing
Wahr
```

Das Objekt ist also leer. Wie kommen wir nun weiter? Wir stellen die Verknüpfung von dem noch im Speicher befindlichen und in *TempVars* gespeicherten Objektzeiger und der Variablen wieder her. Das gelingt über die zweite Schaltfläche des Formulars. Diese prüft, ob *objRibbon_*

Persistent leer ist. Falls ja, ruft die Prozedur *RebuildRibbonObj* auf und übergibt zwei Parameter – die Objektvariable *objRibbon_Persistent* und den Namen der Variablen, unter dem wir den Zeiger in der *TempVars*-Auflistung gespeichert haben:

```
Private Sub cmdRibbonInvalidieren_Click()
    If objRibbon_Persistent Is Nothing Then
        RebuildRibbonObj objRibbon_Persistent, "objRibbon_Persistent"
    End If
    objRibbon_Persistent.Invalidate
End Sub
```

Die Routine *RebuildRibbonObj* sieht so aus (siehe Modul *mdlRibbonTools*):

```
Public Sub RebuildRibbonObj(objRibbon As IRibbonUI, strRibbon As String)
    Dim lObj As Long
    lObj = TempVars(strRibbon)
    If lObj <> 0 Then
        Dim obj As Object
        CopyMemory obj, lObj, 4&
        Set objRibbon = obj
        Debug.Print TypeName(obj)
        CopyMemory obj, 0&, 4&
    End If
End Sub
```

Fehlt noch die hier verwendete API-Funktion *CopyMemory*, die wie folgt deklariert werden muss:

```
Private Declare Sub CopyMemory Lib "kernel32.dll" Alias "RtlMoveMemory" ( ↵
                    ByRef Destination As Any, ByRef Source As Any, ByVal Length As Long)
```

Wenn wir nun im Formular *frmPersistentesRibbon* auf die Schaltfläche *cmdRibbonInvalidieren* klicken, funktioniert dies reibungslos – das *labelControl*-Element zeigt jeweils die aktuelle Uhrzeit an. Diese Lösung funktioniert auf diese einfache Weise leider nur mit Access. Unter Word, Excel und Co. benötigt man einen alternativen Weg zum Speichern des Zeigers auf das *IRibbonUI*-Objekt, da hier das *TempVars*-Objekt nicht zur Verfügung steht.

Die Beispielobjekte zu diesem Abschnitt finden Sie in der Beispieldatenbank *WeitereProgrammiertechniken_Fehlerresistent.accdb*.

10.2 Wohlgeformtheit der Ribbondefinitionen in USysRibbons

Bei den Arbeiten zu diesem Text ist plötzlich immer wieder der gleiche Ribbonfehler aufgetaucht, obwohl das aktuell angezeigt Ribbon aus der Tabelle *USysRibbons* einwandfrei definiert war. Für den Fehler war jedoch ein ganz anderer Datensatz in der Tabelle *USysRibbons* verantwortlich: Dieser hatte schlicht keinen Inhalt im Feld *RibbonXML*. Access prüft aber beim Öffnen alle Einträge der Tabelle *USysRibbons* und hat hierfür einen Fehler angezeigt.

10.3 Alle Elemente im Backstage ausblenden

Wir wollen hier nicht genauer auf die Elemente des Backstage-Bereichs eingehen, aber dennoch zeigen, wie Sie die dort enthaltenen Befehle ausblenden können. Um unter Access 2016 alle Elemente auszublenden, nutzen Sie den Code aus folgendem Listing:

```
<?xml version="1.0"?>
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui">
  <ribbon />
  <backstage>
    <tab idMso="PlaceTabHome" visible="false" />
    <tab idMso ="TabOfficeStart" visible="false" />
    <tab idMso="TabRecent" visible="false" />
    <tab idMso ="TabInfo" visible="false"/>
    <button idMso="FileSave" visible="false"/>
    <tab idMso ="TabSave" visible="false" />
    <tab idMso ="TabPrint" visible="false"/>
    <button idMso="FileCloseDatabase" visible="false"/>
    <tab idMso="TabHelp" visible="false" />
    <tab idMso="TabOfficeFeedback" visible="false" />
    <button idMso="ApplicationOptionsDialog" visible="false"/>
  </backstage>
</customUI>
```

Die Elemente sind in der gleichen Reihenfolge aufgeführt, wie Sie auch im Backstage-Bereich aus Abbildung 10.4 vorkommen.

Abbildung 10.4: Die üblichen Backstage-Befehle unter Access 2016

Wenn wir die obige Ribbondefinition zum Ausblenden der Backstage-Elemente anwenden, ist der Backstage-Bereich völlig geleert (siehe Abbildung 10.5).

André Minhorst

Ribbon Programmierung

LESEPROBE

Hier finden Sie in der Vollversion noch weiteres Know-how rund um die Ribbon-Programmierung!

André Minhorst

Ribbon Programmierung

LESEPROBE

Hier finden Sie in der Vollversion noch weiteres Know-how rund um die Ribbon-Programmierung!

11 Backstage verwenden

Der Backstagebereich von Access und auch der übrigen Office-Anwendungen enthält im Wesentlichen die Funktionen, die einst im *Datei*-Menü (bis Office 2003) beziehungsweise im Office-Menü (Office 2007) enthalten waren. Im Fall von Access finden Sie hier außerdem einige Elemente des früheren *Extras*-Menüs wie beispielsweise die Funktionen zum Komprimieren und Reparieren, Verschlüsseln oder zum Erstellen einer *.mde*/*.accde*-Datenbank.

Den Backstagebereich öffnen Sie durch einen Klick auf den *Datei*-Tab des Ribbons.

11.1 Datenbanken verwalten

Unter dem Eintrag *Startseite* finden Sie verschiedene Möglichkeiten, eine neue Datenbank zu erstellen oder eine vorhandene Datenbank zu öffnen.

Neue Datenbank über den Bereich »Startseite«

Zum Anlegen einer neuen Datenbank gibt es verschiedene Möglichkeiten. Die wohl am meisten genutzte finden Sie direkt im Bereich *Startseite* des Backstagebereichs. Ein Klick auf den Eintrag *Leere Datenbank* startet das Anlegen einer neuen, leeren Datenbank (siehe Abbildung 11.1).

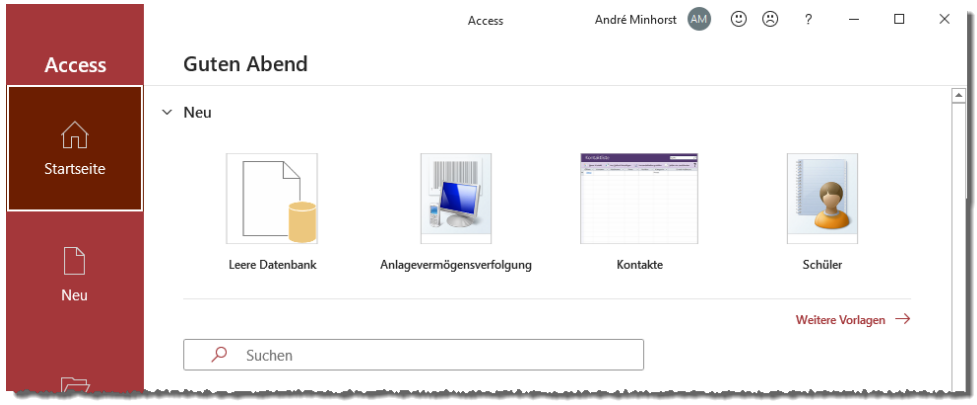


Abbildung 11.1: Möglichkeiten zum Anlegen einer neuen, leeren Datenbank

Dies öffnet den Dialog zum Festlegen des Dateinamens und des Verzeichnisses, in dem die Datenbankdatei gespeichert werden soll (siehe Abbildung 11.2). Mit einem Klick auf die Schaltfläche rechts neben dem Dateinamen öffnen Sie noch den Dialog zum Auswählen von Verzeichnis und Datei.

Abbildung 11.2: Festlegen von Datenbankname und Verzeichnis

Neben der Schaltfläche *Leere Datenbank* finden wir einige Vorschläge für Datenbankvorlagen. Ein Doppelklick auf eine dieser Schaltflächen lädt die entsprechende Vorlage herunter und legt eine neue Datenbank auf Basis dieser Vorlage an. Access speichert die neue Datenbankdatei dabei im Dokumente-Verzeichnis des aktuellen Benutzers.

Mit einem Klick auf die Schaltfläche *Weitere Vorlagen* wechseln Sie zum Bereich *Neu* des Backstagebereichs (siehe Abbildung 11.3). Hier finden Sie weitere Vorlagen oder auch eine Suchfunktion für Onlinevorlagen. Die Menge der verfügbaren Vorlagen hält sich jedoch in Grenzen.

Sie können auch eigene Vorlagen erstellen, dies ist jedoch nicht Thema dieses Buchs.

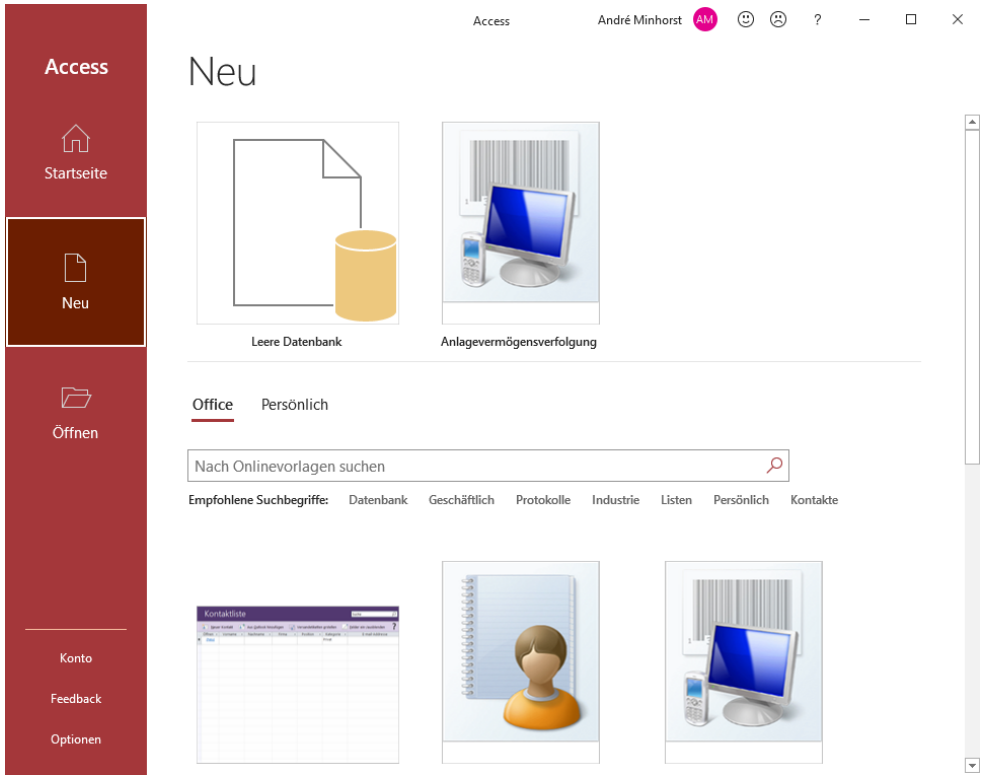


Abbildung 11.3: Weitere Vorlagen im Bereich *Neu*

Wenn Sie ohnehin nur neue, leere Datenbanken anlegen und keine der verfügbaren Vorlagen nutzen wollen, können Sie den entsprechenden Bereich auf der Startseite auch verkleinern. Dazu klicken Sie oben auf den nach unten zeigenden Pfeil links vom Text *Neu*. Anschließend haben Sie mehr Platz für die Anzeige der zuletzt geöffneten Datenbankdateien (siehe Abbildung 11.4).

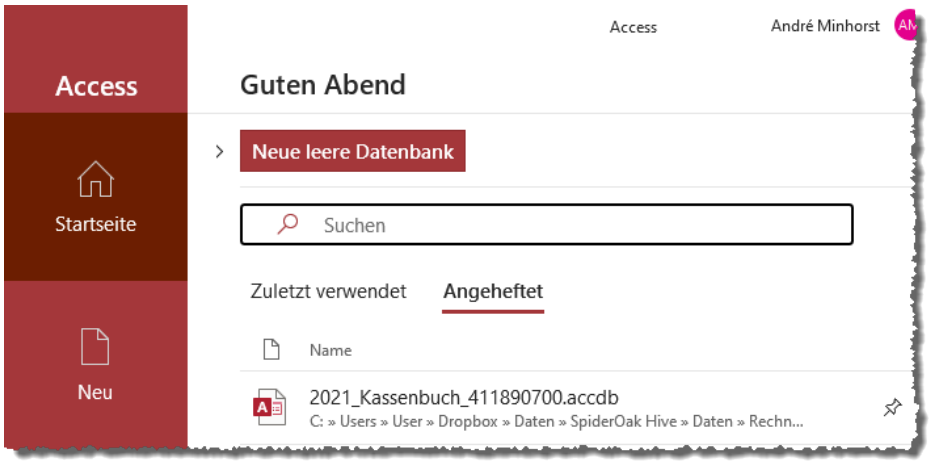


Abbildung 11.4: Die Datenbankvorlagen lassen sich leicht ausblenden.

11.1.1 Neue Datenbank über den Bereich »Neu«

Den zweiten Unterbereich des Backstagebereichs haben Sie soeben schon kennengelernt, als Sie im Bereich *Startseite* auf *Weitere Vorlagen* geklickt haben. Hier finden Sie neben der Schaltfläche *Leere Datenbank* auch noch eine umfangreiche Übersicht über die verfügbaren Vorlagen.

11.1.2 Vorhandene Datenbankdatei öffnen

Eine Neuerung ist eine Erweiterung der Liste der zuletzt geöffneten Datenbankdateien auf der Startseite: Bereits auf den ersten Blick finden Sie eine Liste mit Dateinamen im rechten, größeren Teil des Backstage. Unter *Zuletzt verwendet* finden Sie die zuletzt verwendeten Datenbanken (siehe Abbildung 11.5).

André Minhorst

Ribbon Programmierung

LESEPROBE

Hier finden Sie in der Vollversion noch weiteres Know-how rund um die Ribbon-Programmierung!

André Minhorst

Ribbon Programmierung

LESEPROBE

Hier finden Sie in der Vollversion noch weiteres Know-how rund um die Ribbon-Programmierung!

12 Backstage anpassen

Beim Blick auf die Benutzeroberfläche könnte man den Eindruck gewinnen, dass der Backstagebereich ein Teil des Ribbons ist – immerhin öffnen Sie ihn über eines der Ribbon-Tabs. In der XML-Definition des Ribbons sehen wir jedoch, dass dies nicht die ganze Wahrheit ist. Tatsächlich gibt es dort ein Hauptelement namens *customUI*, dem die Elemente *ribbon* und *backstage* genau wie *commands* und *contextMenus* untergeordnet sind.

Die Beispiele finden Sie in der folgenden Datenbankdatei:

```
Backstage.accdb
```

12.1 Anpassungen des Backstagebereichs

Für den Access-Entwickler ergeben sich einige Möglichkeiten: Im *Backstage*bereich lassen sich wunderbar eigene Elemente unterbringen, ja sogar komplette Anwendungsbereiche. Grundsätzlich haben Sie dort fast die gleichen Möglichkeiten wie in richtigen Access-Formularen. Gleichwohl sei direkt zu Beginn angemerkt, dass der Aufbau und die Pflege von Anwendungsteilen, die statt in Formulare in den Backstagebereich ausgelagert werden, wesentlich aufwendiger sein dürfte.

Im Gegensatz zu den Elementen im Ribbon können Sie jedoch den Backstagebereich nicht über die Benutzeroberfläche anpassen, sondern nur durch das Anwenden einer entsprechenden XML-Definition. Diese hinterlegen Sie, wie bereits weiter oben beschrieben, üblicherweise in einer speziellen Tabelle namens *USysRibbons* – mehr dazu später.

Unter »Eingabehilfe für Ribbondefinitionen« ab Seite 57 haben wir ein Formular namens *frmRibbonXML* vorgestellt, mit dem Sie auch die Beispiele dieses Kapitels ansehen können. Sie finden dieses in der Beispieldatenbank.

In den meisten vorherigen Kapiteln konnten Sie die Beispiele einfach anwenden, indem Sie die gewünschte Ribbondefinition im Formular *frmRibbonXML* ausgewählt und dann die Schaltfläche *Als Formularribbon festlegen* betätigt haben. Das gelingt bei der Anpassung des Backstagebereichs nicht, denn sobald Sie zum Backstagebereich wechseln, verliert das Formular seinen Fokus und die Ribbondefinition ist nicht mehr wirksam.

Also gehen Sie zum Testen wie folgt vor:

» Wählen Sie die Ribbondefinition, die Sie testen möchten, im Formular *frmRibbonXML* aus.

» Klicken Sie auf die Schaltfläche *Als Anwendungsribbon festlegen* (siehe Abbildung 12.1). Das stellt die Access-Option *Name des Menübands* auf diese Ribbondefinition ein.

» Schließen Sie die Datenbank und öffnen Sie diese erneut.

Danach zeigt der Backstagebereich die gewünschten Änderungen an.

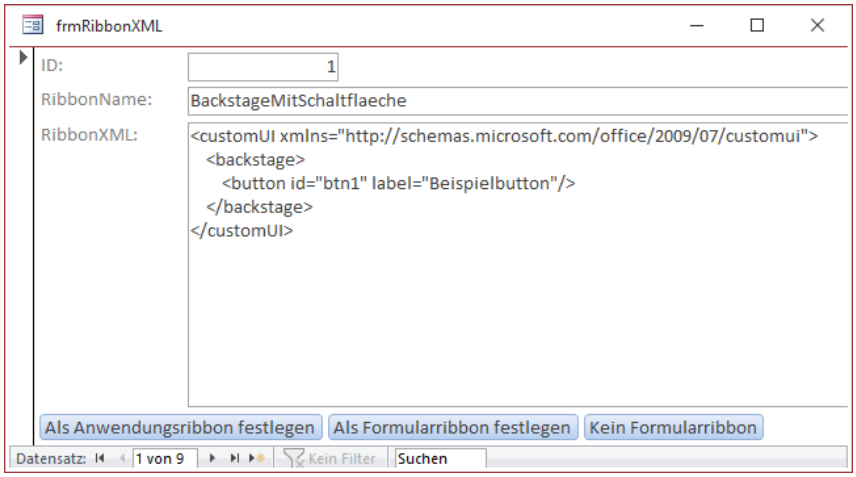


Abbildung 12.1: Das Formular *frmRibbonXML* mit einer Backstage-Anpassung

12.2 Backstage anpassen per XML

Das Anpassen des Backstagebereichs erfolgt also genau wie beim Ribbon über die Erstellung eines XML-Dokuments, das dann in der Tabelle *USysRibbons* in der Datenbank gespeichert wird. Damit eine der in dieser Tabelle gespeicherten Ribbon-Definitionen gleich beim Start der Anwendung zum Einsatz kommt, legen Sie den Namen dieser Definition in den Access-Optionen dieser Datenbankdatei fest.

Im Einzelnen sieht das so aus, dass Sie zunächst eine Tabelle namens *USysRibbons* erstellen, die wie in Abbildung 12.2 aufgebaut ist.

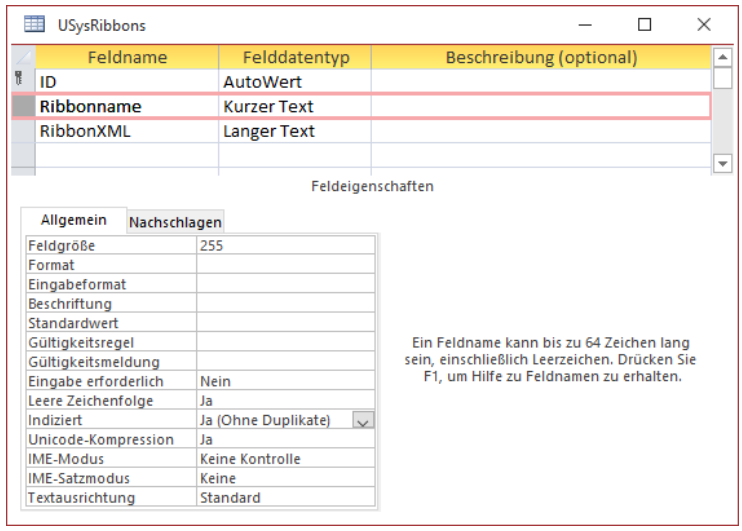


Abbildung 12.2: Entwurf der Tabelle *USysRibbons* zum Speichern von Anpassungen der Benutzeroberfläche

In dieser Tabelle speichern Sie je Datensatz den Namen des Ribbons sowie die XML-Definition dieses Ribbons. Der Name ist wichtig, weil Sie das Ribbon darüber in den Access-Optionen als Anwendungsribbon einstellen können.

Ab Access 2010 verwenden Sie das folgende Schema, das unter anderem die Definition des Backstagebereichs festlegt (unter Access 2007 wurde noch ein anderes Schema verwendet). Die erste Zeile einer jeden XML-CustomUI-Definition sieht unter Access 2010 und jünger daher wie folgt aus:

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

Das den Backstagebereich zusammenfassende Element folgt direkt unterhalb des *customUI*-Elements und heißt folgerichtig *backstage*.

```
<backstage>
    ...
</backstage>
```

Es enthält alle weiteren Elemente des Backstagebereichs. Für die folgende Beschreibung der Implementierung einer *customUI*-Definition fügen wir noch ein weiteres Element hinzu, das eine Schaltfläche zum Backstagebereich hinzufügt. Die Definition sieht so aus:

```
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui">
    <backstage>
        <button id="btn1" label="Beispielbutton"/>
```

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

Damit die Anwendung den Backstagebereich um diese Schaltfläche erweitert, fügen Sie die obige XML-Definition gemeinsam mit dem Namen *BackstageMitSchaltflaeche* zur Tabelle *USysRibbons* hinzu (s. Abbildung 12.3).

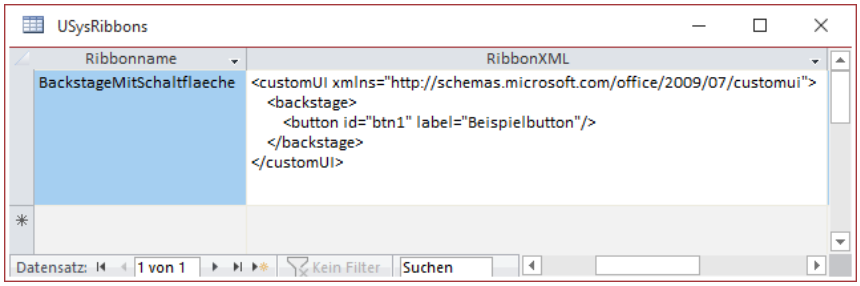


Abbildung 12.3: Definition einer einfachen Backstage-Anpassung

Danach schließen Sie die Anwendung und öffnen sie erneut. Wechseln Sie zum Backstagebereich und klicken Sie unten auf *Optionen*. Im nun erscheinenden Dialog wechseln Sie zum Bereich *Aktuelle Datenbank* und wählen dort unter *Name des Menübands* den Wert *BackstageMitSchaltfläche* aus (s. Abbildung 12.4).

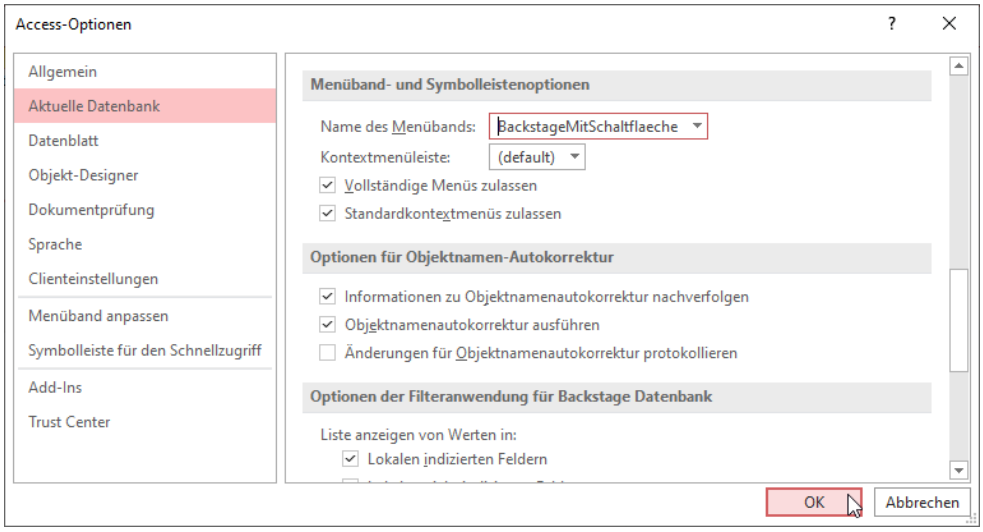


Abbildung 12.4: Einstellen der Anpassung der Benutzeroberfläche

Schließen Sie den Dialog und die Datenbank und öffnen Sie sie nochmals neu. Wenn Sie nun zum Backstagebereich wechseln, finden Sie ganz unten die soeben definierte Schaltfläche (s. Abbildung 12.5).

Abbildung 12.5: Die neue Schaltfläche im Backstagebereich

Sie finden die Ribbondefinition zu diesem Beispiel im Formular *frmRibbonXML* unter dem Namen *BackstageMitSchaltflaeche*.

12.2.1 Eingebaute Elemente ausblenden

Im Backstagebereich lassen sich alle eingebauten Elemente ausblenden. Bevor wir uns also um das Erstellen eigener Elemente im Backstagebereich kümmern, entsorgen wir erst einmal die vorhandenen Elemente.

Genau wie für die Elemente im Ribbon gibt es auch für jedes eingebaute Element im Backstagebereich einen eindeutigen Bezeichner, die sogenannte *idMso*. Diese *idMso* können Sie

für viele Elemente ermitteln, indem Sie in den Access-Optionen zur Registerseite *Symbolleiste für den Schnellzugriff* wechseln, dort im linken Listenfeld den gewünschten Befehl ausfindig machen und mit der Maus darüberfahren. Es erscheint dann ein Tooltip-Text, der die englische *idMso* enthält (s. Abbildung 12.6). Dort finden Sie allerdings nur die *idMsos* für Schaltflächen, nicht für Tabs, Gruppen und andere Struktur-Elemente des CustomUI. Eine komplette Liste aller *idMsos* stellt Microsoft daher in Form einer Excel-Tabelle zum Download bereit. Die für Access relevante Tabelle finden Sie im Download zu diesem Beitrag. Die Tabelle heißt *AccessControls.xlsx*.

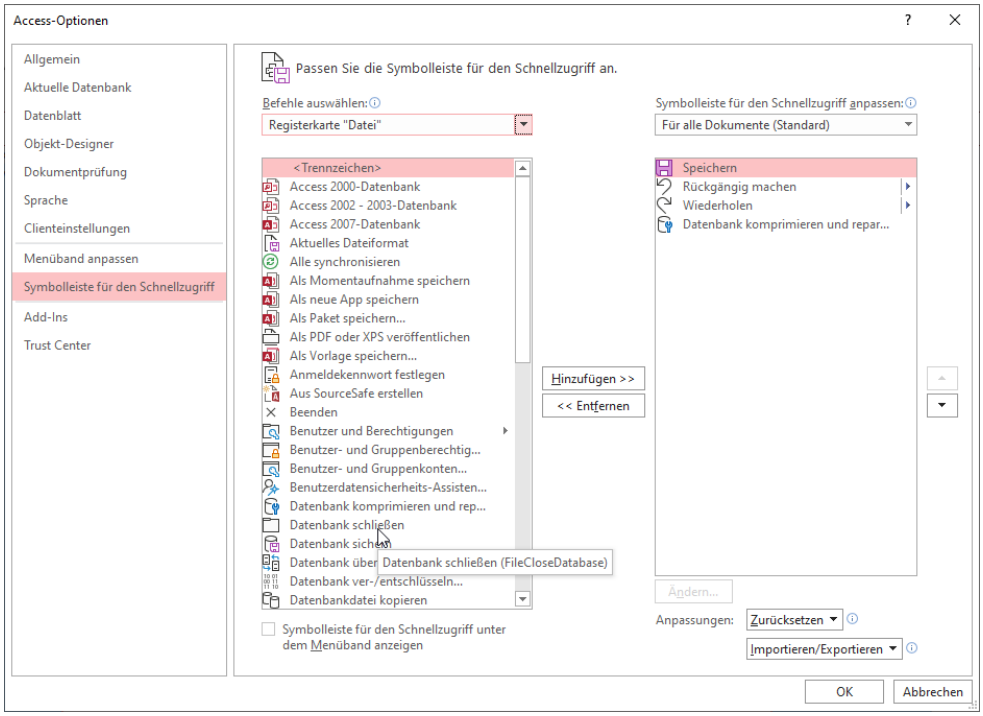


Abbildung 12.6: *idMso* einer Schaltfläche im Backstagebereich ermitteln

Wenn Sie die *idMso* etwa einer Schaltfläche des Backstagebereichs kennen, den Sie beispielsweise ausblenden möchten, können Sie diese Schaltfläche wie das oben beschriebene *button*-Element definieren.

Sie lassen allerdings das *id*-Attribut weg und ersetzen dieses durch das *idMso*-Attribut. Außerdem brauchen Sie natürlich kein *label*-Attribut anzugeben, denn eingebaute Steuerelemente besitzen bereits eine voreingestellte Beschriftung.

Da Sie das Steuerelement verbergen möchten, fügen Sie das Attribut *visible* hinzu und legen den Wert *false* fest. Für die Speichern-Schaltfläche sieht das etwa so aus:

```
<button idMso="FileSave" visible="false"/>
```

Auf die gleiche Weise verfahren Sie mit den übrigen Schaltflächen. Anschließend kümmern Sie sich um die Schaltflächen, welche die Anzeige verschiedener Bereiche rechts im Backstage bewirken – also beispielsweise *Informationen* oder *Zuletzt verwendet*.

Dies sind keine herkömmlichen *button*-Elemente, sondern *tab*-Elemente des Backstagebereichs.

Davon abgesehen verhalten sich diese allerdings genau so wie die *button*-Elemente, sodass Sie etwa den Eintrag *Informationen* mit der folgenden Zeile verschwinden lassen:

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

Definieren Sie ein XML-Dokument wie folgt und fügen Sie dieses in die Tabelle USysRibbons ein. Fügen Sie außerdem für das Feld *RibbonName* die Bezeichnung *BackstageElementeAusblenden* hinzu:

```
<?xml version="1.0"?>
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui">
  <ribbon />
  <backstage>
    <tab idMso="PlaceTabHome" visible="false" />
    <tab idMso ="TabOfficeStart" visible="false" />
    <tab idMso="TabRecent" visible="false" />
    <tab idMso ="TabInfo" visible="false"/>
    <button idMso="FileSave" visible="false"/>
    <tab idMso ="TabSave" visible="false" />
    <tab idMso ="TabPrint" visible="false"/>
    <button idMso="FileCloseDatabase" visible="false"/>
    <tab idMso="TabHelp" visible="false" />
    <tab idMso="TabOfficeFeedback" visible="false" />
    <button idMso="ApplicationOptionsDialog" visible="false"/>
  </backstage>
</customUI>
```

Schließen und öffnen Sie die Datenbankdatei erneut und öffnen Sie die Access-Optionen. Wählen Sie dort im Bereich *Aktuelle Datenbank* unter *Menüband- und Symbolleistenoptionen* für die Eigenschaft *Name des Menübands* den Wert *BackstageElementeAusblenden* aus.

Schließen und öffnen Sie die Datenbank erneut und wechseln Sie dann über den Registerreiter Datei zum Backstagebereich. Dieser sieht nun wie in Abbildung 12.7 aus.

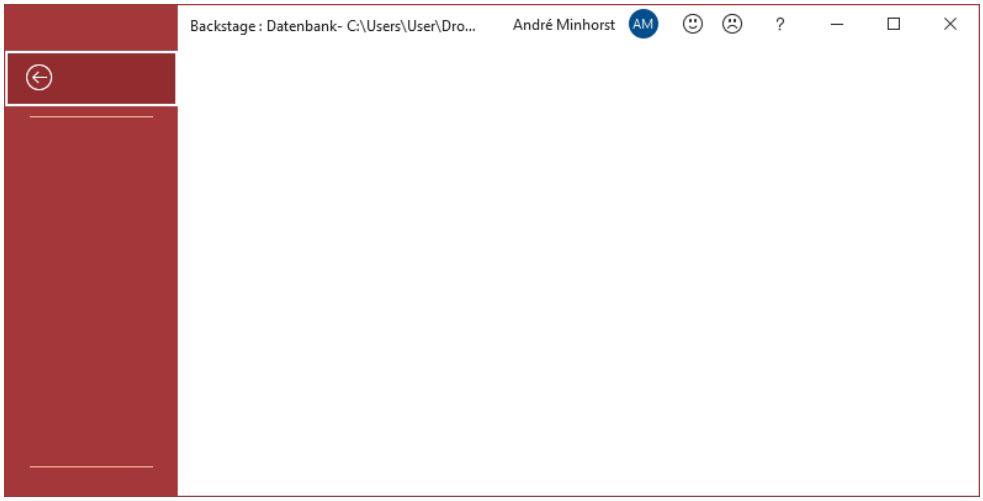


Abbildung 12.7: Der komplett geleerte Backstagebereich

Sie finden die Ribbondefinition zu diesem Beispiel im Formular *frmRibbonXML* unter dem Namen *BackstageElementeAusblenden*.

12.2.2 Einfache Schaltflächen

Am schnellsten fügen Sie dem Backstagebereich einfache Schaltflächen im linken Bereich zu. Dazu fügen Sie unterhalb des *backstage*-Elements wie bereits weiter oben gezeigt einfach ein *button*-Element ein.

Wie wollen an dieser Stelle auch gleich die bereits vorgestellten Techniken zur Anzeige von Bildern für Schaltflächen und andere Steuerelemente nutzen, um den Backstagebereich optisch ansprechend und funktional zu gestalten. Dazu benötigen Sie, wie in »Bilder im Ribbon anzeigen« ab Seite 77 erläutert, das Modul *mdlRibbonImages* sowie die ebenfalls bereits vorgestellte Callbackprozedur für das Attribut *loadImage*.

Der Code für die Definition des Ribbons mit den beiden Schaltflächen sieht wie folgt aus::

```
<?xml version="1.0"?>
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui"
loadImage="loadImage">
  <backstage>
    <tab idMso="PlaceTabHome" visible="false" />
    <tab idMso ="TabOfficeStart" visible="false" />
    <!--weitere ausgeblendete eingebaute Elemente-->
    <button id="btnOhneBild" onAction="onAction" label="Button ohne Bild"/>
```

```
    <button id="btnMitBild" onAction="onAction" label="Button mit Bild" image="close"/>
  </backstage>
</customUI>
```

Das Ergebnis sehen Sie in Abbildung 12.8. Die Schaltflächen werden allerdings noch ganz unten angezeigt – wir hätten diese gern weiter oben dargestellt, wo sich auch die üblichen Steuerelemente normalerweise befinden.

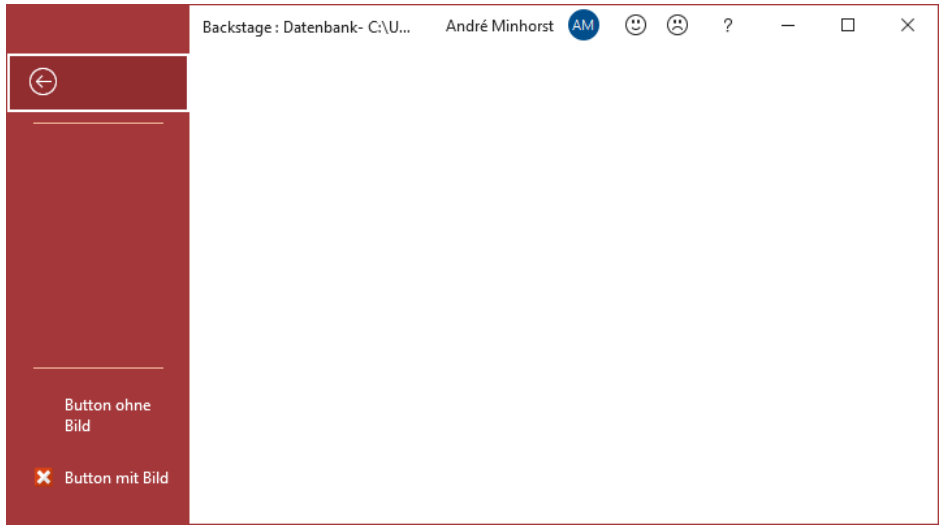


Abbildung 12.8: Einfache Schaltflächen mit und ohne Bild

Der Grund, warum die Elemente ganz unten erscheinen, ist einfach: Elemente, für welche nicht explizit eine Position angegeben wird, erscheinen immer am Ende, in diesem Fall hinter dem letzten eingebauten Element. In unserem Beispiel wird zwar kein eingebautes Element angezeigt, aber es wird die Position dieses Elements herangezogen, um die benutzerdefinierten Steuerelemente darunter zu platzieren. Und das letzte Element ist das mit der Beschriftung Optionen und der *idMso* namens *ApplicationOptionsDialog*. Diese Schaltfläche erscheint, sofern sichtbar, ganz unten links. Dementsprechend werden auch neue benutzerdefinierte Elemente dort angezeigt. Das können Sie allerdings ändern, indem Sie die Position der benutzerdefinierten Schaltflächen mit dem Attribut *insertBeforeMso* anpassen – beispielsweise unter Angabe der *idMso* für das oberste Element, nämlich *PlaceTabHome*. Also ändern wir die beiden benutzerdefinierten Elemente wie folgt:

```
<button id="btnOhneBild" onAction="onAction" label="Button ohne Bild"
insertBeforeMso="PlaceTabHome" />
<button id="btnMitBild" onAction="onAction" label="Button mit Bild" image="close" insert
BeforeMso="PlaceTabHome"/>
```

Das Ergebnis sieht nun wie in Abbildung 12.9 aus.

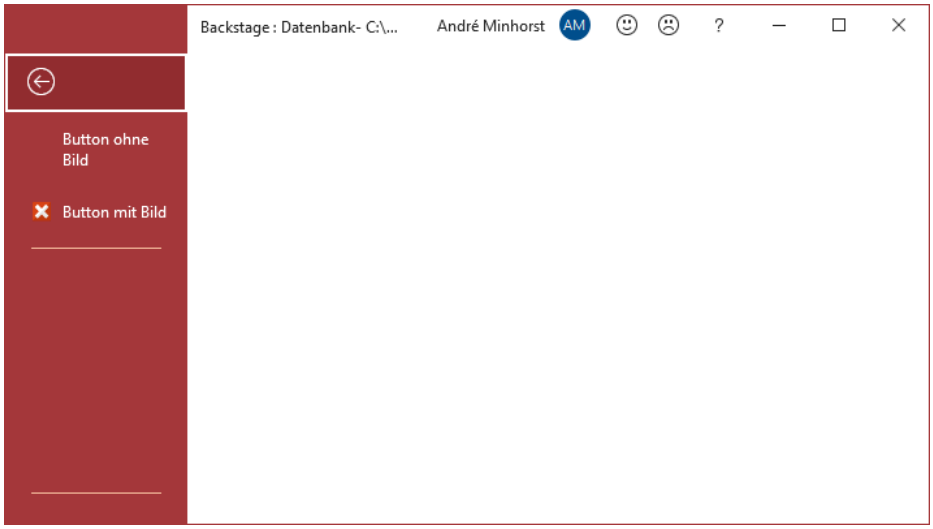


Abbildung 12.9: Benutzerdefinierte Schaltflächen oben im Backstagebereich

Für das Attribut *onAction* hinterlegen wir noch wie folgt eine passende Prozedur im Modul *mdlRibbons*:

```
Sub onAction(control As IRibbonControl)
    Select Case control.ID
        Case "btnOhneBild"
            MsgBox "Klick auf Button mit Bild"
        Case "btnMitBild"
            MsgBox "Klick auf Button ohne Bild"
        Case Else
            Debug.Print control.ID
    End Select
End Sub
```

Damit können Sie nun auch noch ausprobieren, ob ein Klick auf die beiden Schaltflächen jeweils die Prozedur *onAction* aufruft, was für das nachfolgend vorgestellte Attribut *isDefinitive* interessant ist.

Das Beispiel finden Sie im Formular *frmRibbonXML* unter dem Namen *BackstageSchaltflaechen*.

André Minhorst

Ribbon Programmierung

LESEPROBE

Hier finden Sie in der Vollversion noch weiteres Know-how rund um die Ribbon-Programmierung!

André Minhorst

Ribbon Programmierung

LESEPROBE

Hier finden Sie in der Vollversion noch weiteres Know-how rund um die Ribbon-Programmierung!

13 Ribbon-Admin

Der Ribbon-Admin ist ein Tool aus dem gleichen Verlag wie das vorliegende Buch. Sie finden eine Testversion sowie die Möglichkeit zum Kauf unter folgendem Link:

```
https://shop.minhorst.com/access-tools/309/ribbon-admin-2016?c=78
```

Das Tool erlaubt in der Testversion das Anlegen von Ribbons mit maximal einem gleichartigen Unterelement pro Ribbon-Definition. Sie können damit also beispielsweise ein *customUI*-Element mit einem *tabs*-, einem *tab*-, einem *group*- und einem *button*-Element anlegen. Das reicht für die Grundstruktur und zum Experimentieren aus.

Dieses Kapitel beschreibt die wichtigsten Möglichkeiten des Ribbon-Admin bei der Entwicklung von Ribbon-Definitionen.

13.1 Installation des Ribbon-Admin

Der Ribbon-Admin kommt als Access-Add-In. Die Installation erfolgt daher in folgenden Schritten:

» Entpacken Sie die Datei aus dem Zip-Archiv an einem beliebigen Ort im Dateisystem.

» Starten Sie Access und öffnen Sie eine beliebige Datenbankdatei.

» Betätigen Sie den Ribbon-Befehls *Datenbanktools|Add-Ins|Add-Ins|Add-In-Manager* (siehe Abbildung 13.1).

» Klicken Sie im nun erscheinenden Dialog *Add-In-Manager* auf die Schaltfläche *Neues hinzufügen...* (siehe Abbildung 13.2).

» Wählen Sie im nun erscheinenden Dateiauswahl-Dialog die soeben gespeicherte *.accda*-Datenbank aus und schließen Sie den Dialog.

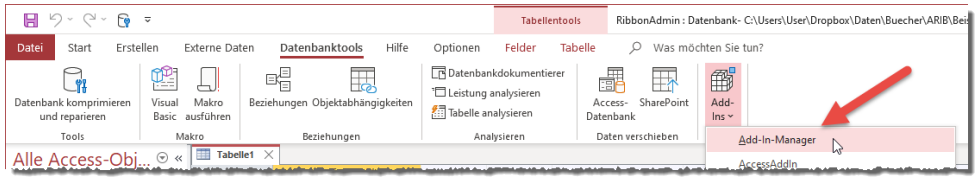


Abbildung 13.1: Starten des Add-In-Managers

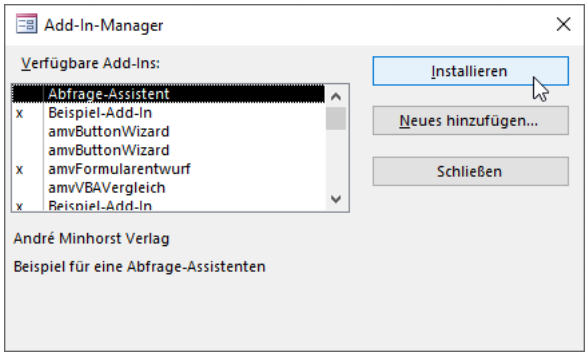


Abbildung 13.2: Installieren eines neuen Access-Add-Ins

Nachdem Sie den die Add-In-Datenbank ausgewählt und den Dialog geschlossen haben, sollte der Eintrag *Ribbon-Admin 2016* in der Liste *Verfügbare Add-Ins* im *Add-In-Manager* erscheinen (siehe Abbildung 13.3). Im unteren Bereich werden allgemeine Informationen über das aktuell markierte Add-In angezeigt.

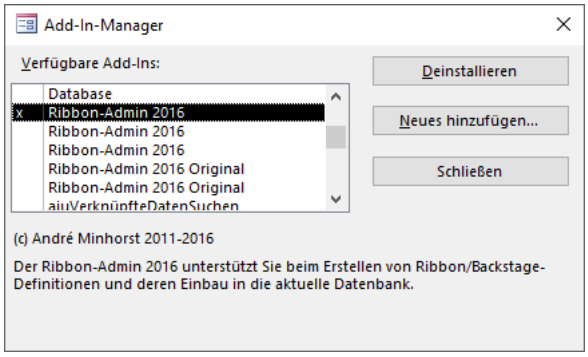


Abbildung 13.3: Das neue Add-In im Add-In-Manager

Schließen Sie nun den Add-In-Manager mit der Schaltfläche *Schließen*. Klappen Sie danach erneut die Liste der Add-Ins auf, indem Sie den Ribboneintrag *Datenbanktools|Add-Ins|Add-Ins* betätigen. In dieser Liste sollte ebenfalls ein entsprechender Eintrag namens RibbonAdmin2016 sichtbar sein (siehe Abbildung 13.4).

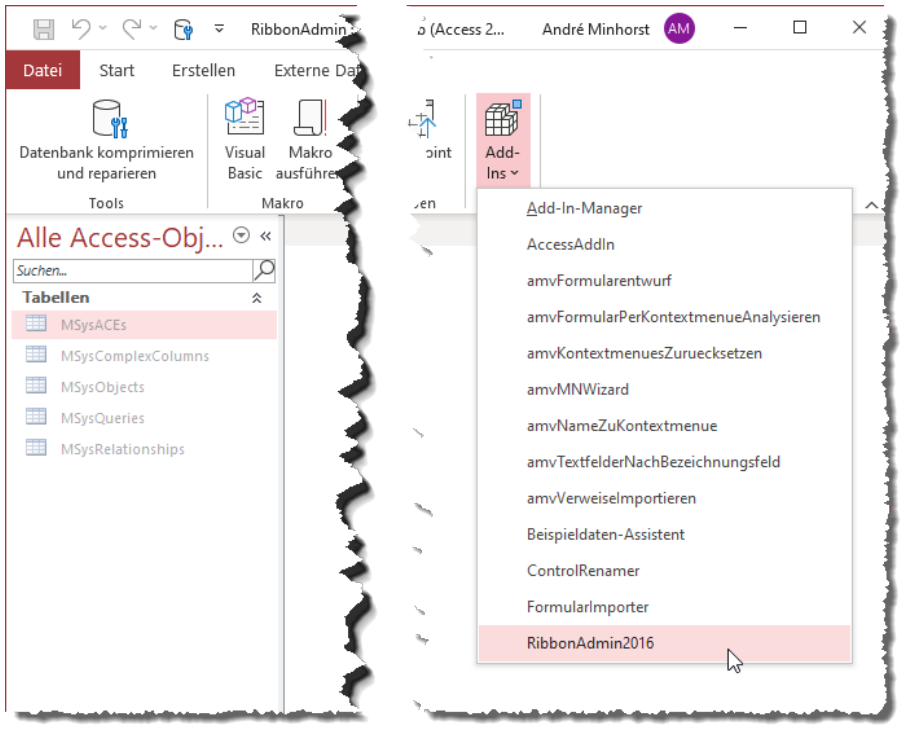


Abbildung 13.4: Anzeige des Ribbon-Admin im Add-In-Menü

13.2 Starten des Ribbon-Admins

Diese können Sie nun durch einen Klick auf den Eintrag *RibbonAdmin2016* in der Liste der verfügbaren Add-Ins starten. Wenn Sie den Ribbon-Admin frisch installiert haben, ist dieser noch nicht registriert. Um das zu erledigen, klicken Sie nach dem Start und dem Erscheinen des Ribbon-Admin oben rechts auf das Schlüsselsymbol (gegebenenfalls müssen Sie das Fenster soweit vergrößern, dass das Schlüsselsymbol sichtbar wird).

Daraufhin erscheint der Dialog *Registrierung*. Wenn Sie eine Lizenz erworben haben, geben Sie die Informationen aus der Registrierungs-E-Mail in die beiden Felder *Benutzername* und *Registrierungsschlüssel* ein und betätigen Sie wie in Abbildung 13.5 die *Registrieren*-Schaltfläche.

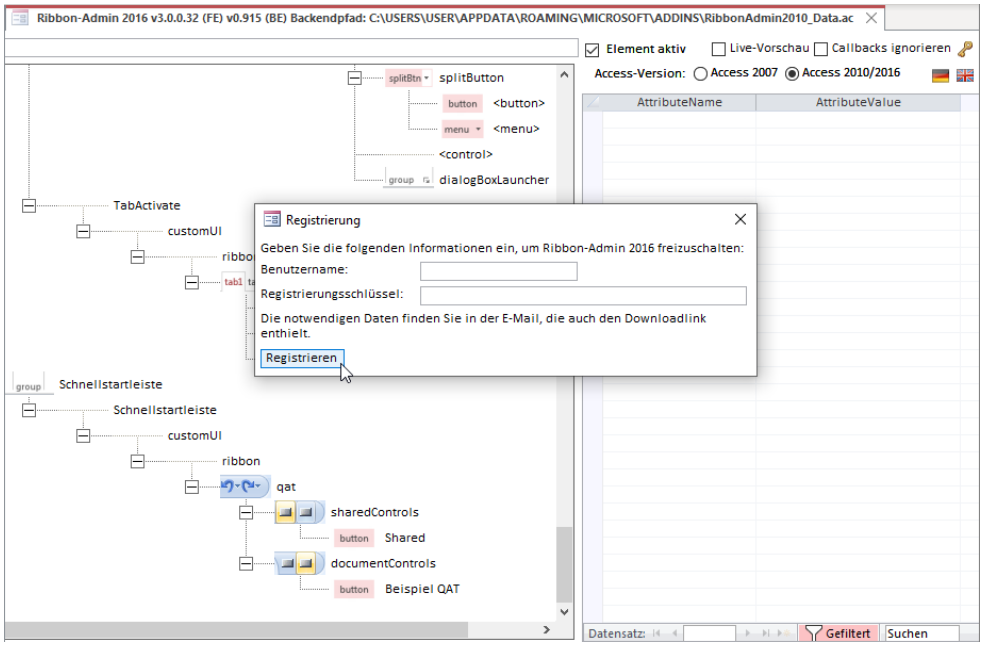


Abbildung 13.5: Eingabe der Registrierungsdaten

Nachdem Sie die Daten eingegeben haben, zeigt eine Meldung wie aus Abbildung 13.6 an, dass Sie ausreichend Zeit haben, den Ribbon-Admin anzuwenden.

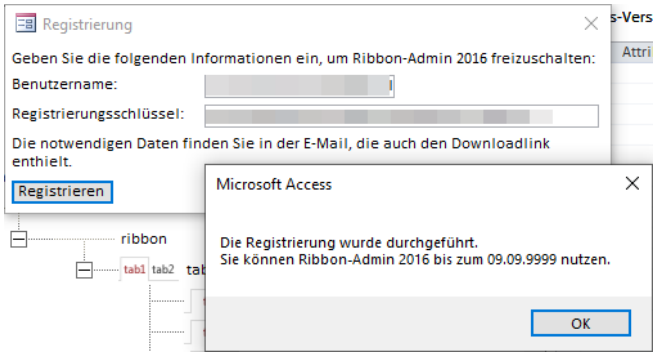


Abbildung 13.6: Bestätigung der Registrierung

Start aus der Zielanwendung heraus

Grundsätzlich ist an dieser Stelle anzumerken, dass Sie den Ribbon-Admin öffnen, wenn die Anwendung, der Sie eine Ribbondefinition hinzufügen, in Access geöffnet ist. Dann können Sie die Ribbon-Definitionen direkt in die Zielanwendung übertragen.

13.3 Neue Anwendung anlegen

Wenn Sie den Ribbon-Admin neu installiert haben, enthält das TreeView-Steuerelement auf der linken Seite lediglich einen Eintrag mit dem Text *Benutzerdefinierte Ribbons* (siehe Abbildung 13.7).

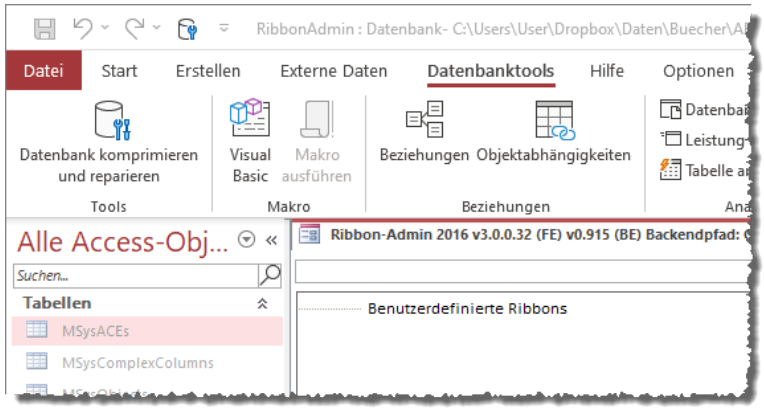


Abbildung 13.7: Dieser Ribbon-Admin enthält noch keine Ribbon-Definitionen

Bevor Sie eine Ribbondefinition anlegen können, legen Sie zunächst ein Element an, dass alle Ribbondefinitionen für eine Anwendung erfasst. Dazu klicken Sie mit der rechten Maustaste auf das Element *Benutzerdefinierte Ribbons* und wählen aus dem nun erscheinenden Kontextmenü den Eintrag *Neue Anwendung* aus (siehe Abbildung 13.8).

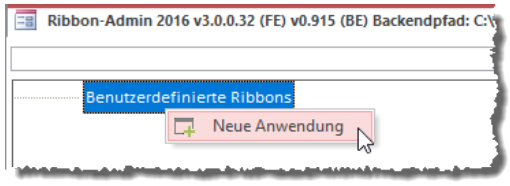


Abbildung 13.8: Hinzufügen einer neuen Anwendung

Für den neuen Eintrag legen Sie nun noch einen neuen Namen fest, zum Beispiel *Beispielanwendung* (siehe Abbildung 13.9).

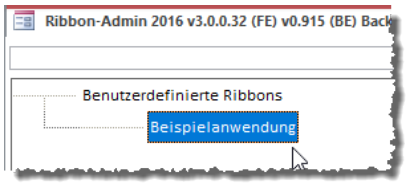


Abbildung 13.9: Eine neue Beispielanwendung

13.4 Neues Ribbon anlegen

Wenn Sie mit der rechten Maustaste auf das Element für die neue Anwendung klicken, erscheinen weitere Optionen im Kontextmenü (siehe Abbildung 13.10).

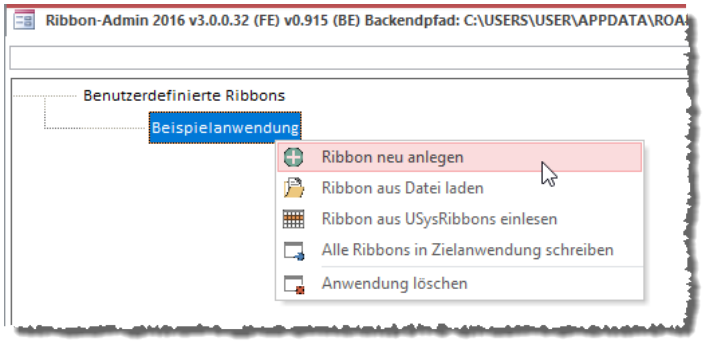


Abbildung 13.10: Option zum Anlegen eines neuen Ribbons

Die wichtigste Option für uns ist hier die zum Anlegen eines neuen Ribbons. Nachdem Sie diese betätigt haben, erscheint ein weiterer Eintrag unterhalb von Bespielanwendung, für den Sie einen Namen vergeben können. Hier ist wichtig, welche Bezeichnung Sie verwenden, denn diese wird beim Übertragen der Ribbondefinition in die Zielanwendung in das Feld *RibbonName* der Tabelle *USysRibbons* übermittelt und wird somit auch in den Access-Optionen und in den Eigenschaften von Formularen und Berichten unter *Name des Menübands* angezeigt. Sie sollten hier also durchaus einen sprechenden Namen verwenden.

Für das Hauptribbon einer Anwendung hat sich die Verwendung der Bezeichnung *Main* bewährt. Diesen Namen verwenden wir daher auch für unser Beispielribbon, das nach dem Anlegen wie in Abbildung 13.11 erscheint.

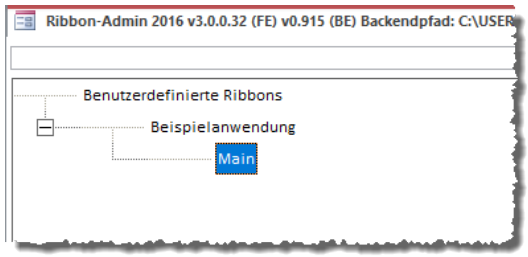


Abbildung 13.11: Neues Hauptelement der Ribbondefinition

13.5 Ribbon-Elemente anlegen

Damit haben wir die grundlegenden Elemente hinzugefügt. Ab jetzt geht es weiter mit den eigentlichen Ribbonelementen, die Sie auch in der XML-Ribbondefinition festlegen.

13.5.1 customUI-Element anlegen

Das nächste Element, das wir anlegen, ist das erste eigentliche Ribbonelement. Dabei handelt es sich um das *customUI*-Element. Dieses legen Sie über den Eintrag *customUI anlegen* des Kontextmenüs des Ribbonelements an (siehe Abbildung 13.12).

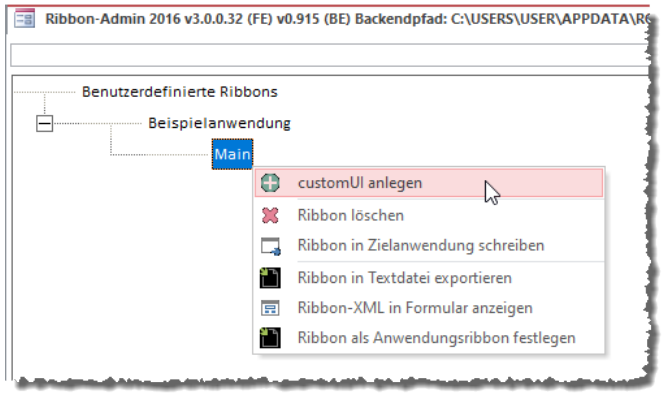


Abbildung 13.12: Hinzufügen des *customUI*-Elements

Hier finden Sie noch weitere Kontextmenüeinträge, die wir weiter unten erläutern.

Das so hinzugefügte *customUI*-Element ist das erste, für das Sie im rechten Bereich Attribute definieren können. Hier finden Sie wie in Abbildung 13.13 die beiden Attribute *loadImage* und *onLoad* vor.

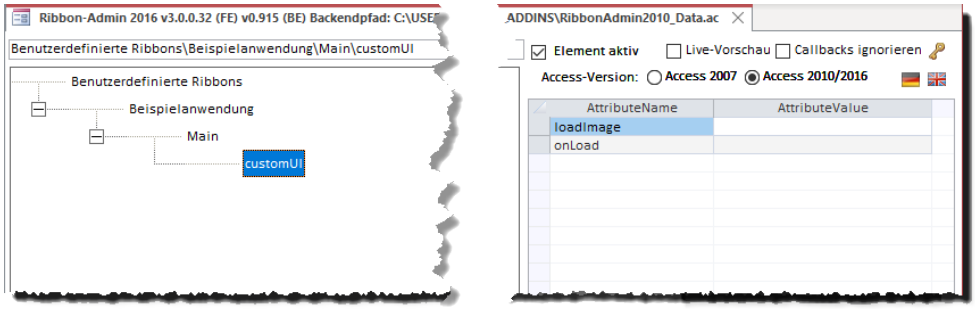


Abbildung 13.13: Das *customUI*-Element mit seinen Attributen

Wie Sie diese nutzen, erfahren Sie weiter unten unter »Callback-Attribute nutzen« ab Seite 254.

13.5.2 ribbon-Element anlegen

Ein Rechtsklick auf das customUI-Element liefert gleich vier Möglichkeiten zum Hinzufügen von Unterelementen – für das *backstage*-, das *commands*-, das *contextMenus*- und das *ribbon*-Element (siehe Abbildung 13.14). Wir wählen hier das *ribbon*-Element aus.

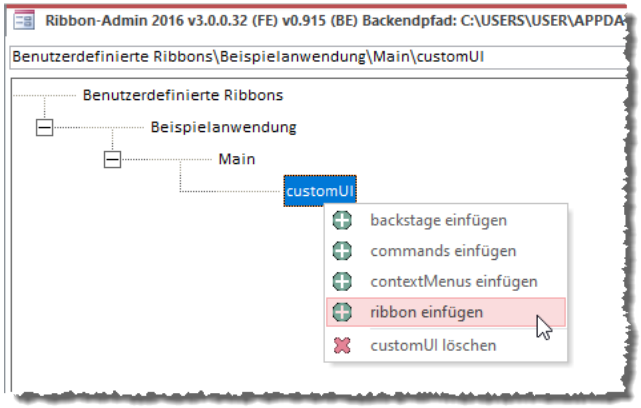


Abbildung 13.14: Hinzufügen eines *ribbon*-Elements

Das *ribbon*-Element bietet im Attribute-Bereich das Attribut *startFromScratch* an, mit dem Sie einstellen können, dass die eingebauten Elemente ausgeblendet werden.

13.5.3 tabs-Element anlegen

Damit können wir gleich weitermachen und über das Kontextmenü des Eintrags *ribbon* ein *tabs*-Element hinzufügen. Auch hier gibt es wieder weitere Elemente, die Sie anlegen können – nämlich das *contextualTabs*- und das *qat*-Element (siehe Abbildung 13.15). Wir wählen jedoch zunächst das oft verwendete *tabs*-Element aus.

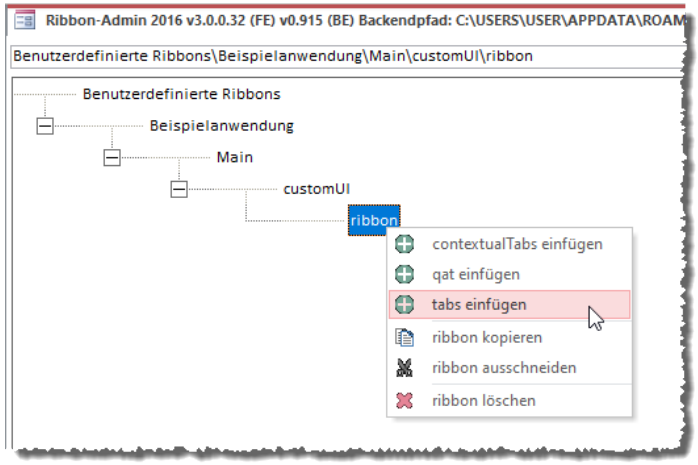


Abbildung 13.15: Hinzufügen des *tabs*-Elements

13.5.4 tab-Element hinzufügen

Wir wollen jedoch zunächst einmal ein einfaches Ribbon definieren und fügen deshalb ein *tab*-Element hinzu (siehe Abbildung 13.16).

Nachdem dies erledigt ist, sehen wir, dass das *tab*-Element gleich eine ganze Reihe *Attribute* präsentiert (siehe Abbildung 13.17). Einige dieser Attribute sind sogar für die Anzeige des Ribbons sichtig, denn jedes *tab*-Element enthält eine Beschriftung. Diese legen Sie mit dem Attribut *label* fest. Außerdem möchten Sie das *tab*-Element gegebenfalls referenzieren. Dazu hinterlegen wir einen Wert für das Attribut *id*, in diesem Fall *tabBeispiel*.

Sie können auch eingebaute tab-Elemente einfügen. Mehr dazu lesen Sie weiter unten unter »Eingebaute tab-Elemente hinzufügen« ab Seite 243.

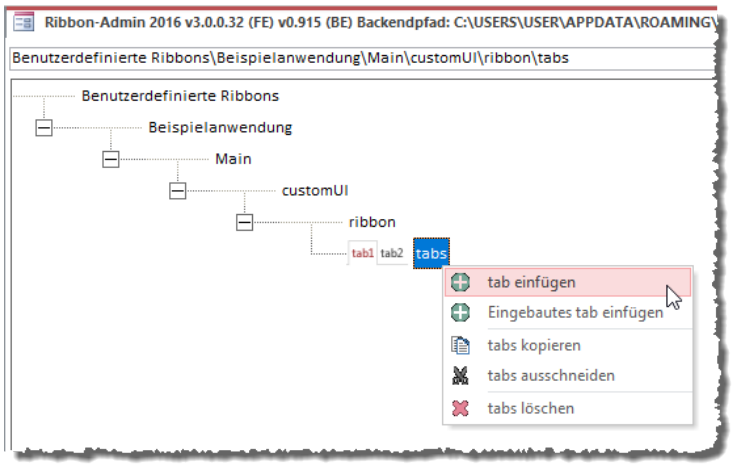


Abbildung 13.16: Hinzufügen eines *tab*-Elements

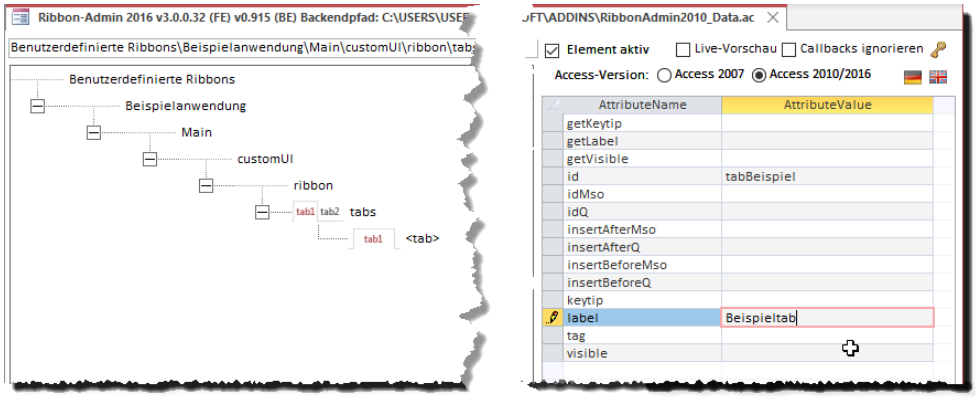


Abbildung 13.17: Einstellen der Attribute eines *tab*-Elements

Der Wert, den Sie für das Attribut *label* eingetragen haben, wird anschließend direkt im TreeView für das *tab*-Element angezeigt (siehe Abbildung 13.18).

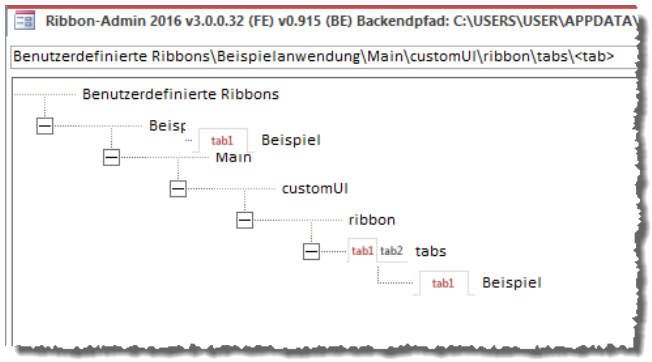


Abbildung 13.18: *tab*-Element mit Beschriftung

13.5.5 group-Element hinzufügen

Als Nächstes folgt ein *group*-Element. Dieses fügen Sie mit dem Kontextmenüeintrag *group* des *tab*-Elements hinzu (siehe Abbildung 13.19).

Alternativ können Sie hier auch noch eine eingebaute Gruppe hinzufügen. Mehr dazu weiter unten unter »Eingebaute group-Elemente hinzufügen« ab Seite 247.

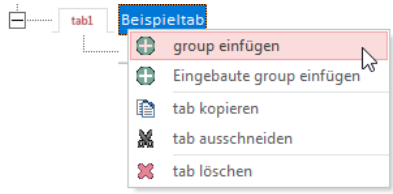


Abbildung 13.19: Einfügen eines *group*-Elements

Auch das *group*-Element bietet einige Attribute zum Einstellen an. Die wichtigsten sind für den Moment das Attribut *id*, mit dem Sie die eindeutige Bezeichnung des Elements festlegen (zum Beispiel *grpBeispiel*) und das Attribut *label*. Der Text, den Sie für dieses Attribut eingeben, wird unter der Gruppe angezeigt.

13.5.6 button-Element hinzufügen

Bevor wir erstmalig die Ribbondefinition zur Zielanwendung übertragen, wollen wir noch ein funktionsfähiges *button*-Element zu dem soeben erstellten *group*-Element hinzufügen. Dazu wählen wir den Eintrag *button* aus dem Kontextmenü des *group*-Elements aus (siehe Abbildung 13.20). Neben dem *button*-Element finden Sie eine ganze Reihe weiterer Einträge, denn Sie können hier alle verfügbaren Steuerelemente hinzufügen.

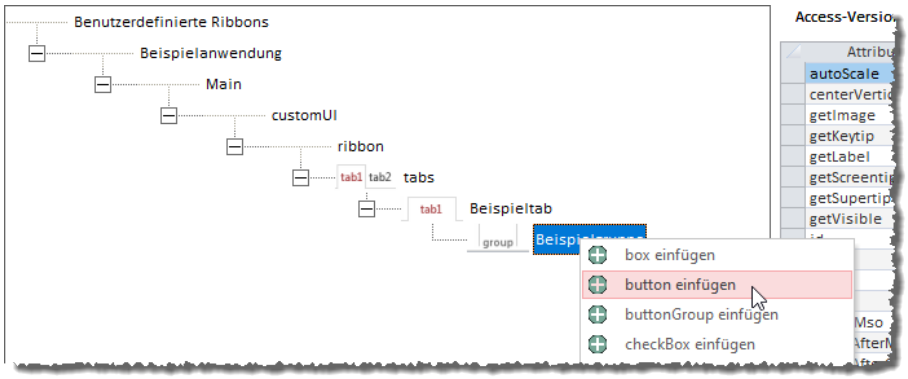


Abbildung 13.20: Hinzufügen eines *button*-Elements

Wie für das *tab*- und das *group*-Element legen wir auch hier wieder Werte für die Attribute *id* und *label* fest, in diesem Fall *btnBeispiel* und *Beispielbutton* (siehe Abbildung 13.21).

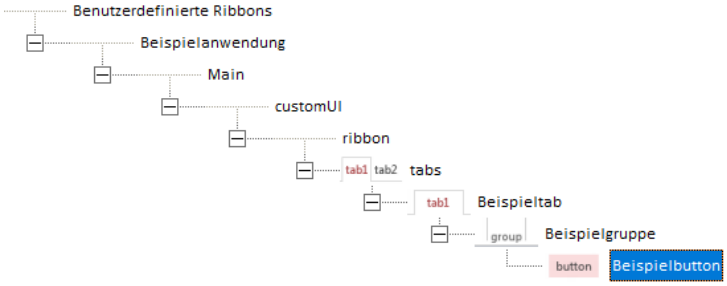


Abbildung 13.21: Beispiel für ein *button*-Element

Auch für das *button*-Element stellen wir einige Attribute ein. Diesmal sind es allerdings nicht nur die beiden Attribute *id* (*btnBeispiel*) und *label* (*Beispielbutton*). Wir legen auch noch einen Wert für das Attribut *onAction* fest. Damit stellen wir den Namen der VBA-Prozedur ein, die durch einen Klick auf die Schaltfläche ausgelöst werden soll. Allerdings tragen wir den Namen hier nicht einfach ein, sondern nutzen das Kontextmenü. Klicken Sie mit der rechten Maustaste auf das Feld für das Attribut *onAction*, erscheint ein Kontextmenü mit einem einzigen Eintrag (siehe Abbildung 13.22).

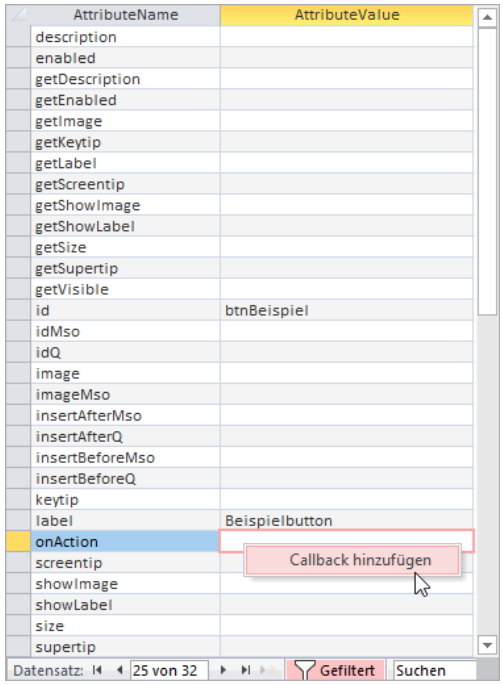


Abbildung 13.22: Hinzufügen eines Callbacks

Nach dem Betätigen dieses Eintrags erscheint als Erstes eine Meldung, die darauf hinweist, dass dem aktuellen VBA-Projekt ein Verweis auf die Bibliothek *Microsoft Office x.0 Object Library* hinzugefügt wird. Diese Meldung erscheint übrigens nur beim erstmaligen Hinzufügen einer Callback-Prozedur (siehe Abbildung 13.23).

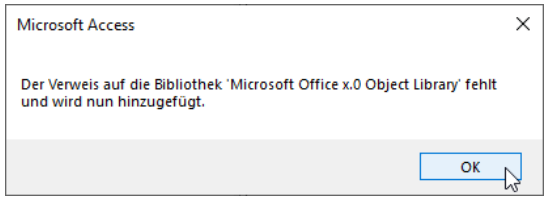


Abbildung 13.23: Hinweis auf das Hinzufügen der Office-Bibliothek zum VBA-Projekt

Direkt im Anschluss weist eine weitere Meldung darauf hin, dass ein Modul namens *mdlRibbons* zum VBA-Projekt hinzugefügt wird (siehe Abbildung 13.24). Auch diese Meldung erscheint nur, wenn noch kein gleichnamiges Modul in der Zielanwendung vorhanden ist. Dieses Modul nimmt alle vom Ribbon-Admin erstellten Callback-Prozeduren auf.

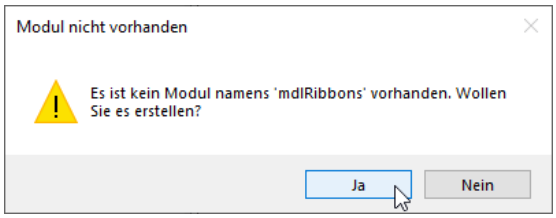


Abbildung 13.24: Hinzufügen eines Moduls namens *mdlRibbons*

Schließlich folgt noch die Bestätigung, dass die Callbackprozedur im Modul *mdlRibbons* angelegt wurde (siehe Abbildung 13.25).

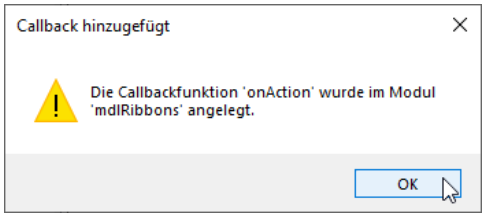


Abbildung 13.25: Hinweis auf das Anlegen der Callbackprozedur *onAction*

Damit werfen wir einen Blick auf das Modul *mdlRibbons*, das wir beispielsweise durch einen Doppelklick auf den entsprechenden Eintrag im Navigationsbereich öffnen. Das Modul sieht im VBA-Editor wie in Abbildung 13.26 aus.

Die hier angelegte VBA-Prozedur ergänzen wir um eine Zeile, damit wir testen können, ob sie beim Betätigen der Ribbon-Schaltfläche ausgelöst wird:

```
Sub onAction(control As IRibbonControl)
    MsgBox "Klick auf: " & control.Id
End Sub
```

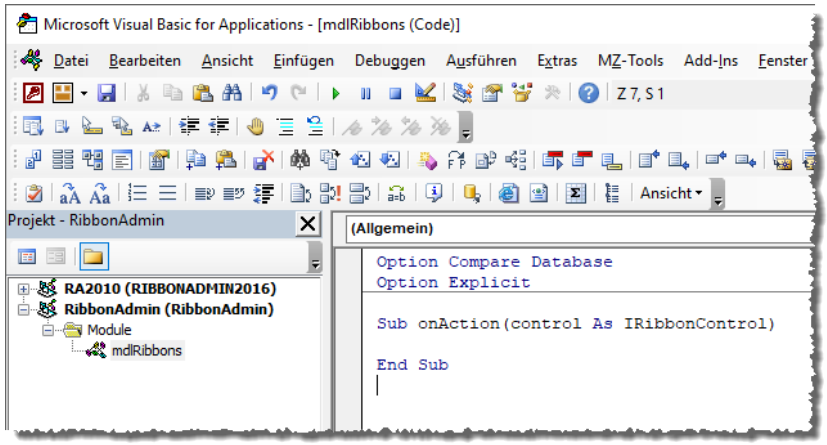


Abbildung 13.26: Die frisch angelegte Callbackprozedur

13.6 Ribbon in Anwendung übertragen

Danach wollen wir die Ribbon-Definition erstmalig in die Anwendung übertragen, in der wir nun bereits das Modul *mdlRibbons* mit der ersten *onAction*-Callback-Funktion angelegt haben.

Dazu klicken Sie mit der rechten Maustaste auf das Element mit dem Namen des Ribbons, hier *Main*, und wählen wie in Abbildung 13.27 den Kontextmenüeintrag *Ribbon in Zielanwendung schreiben* aus.

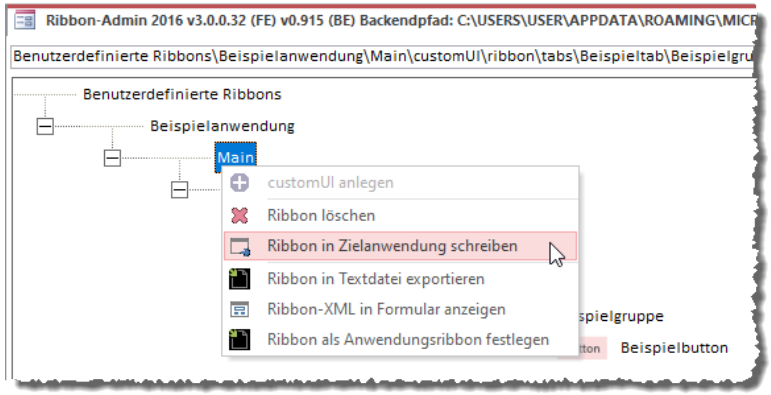


Abbildung 13.27: Schreiben des Ribbons in die Zielanwendung

Nun folgt, sofern die Datenbank noch keine Tabelle namens *USysRibbons* enthält, die Frage, ob eine solche Tabelle erstellt werden soll. Diese Aufgabe lassen wir uns gern abnehmen und betätigen die Schaltfläche *Ja* (siehe Abbildung 13.28).

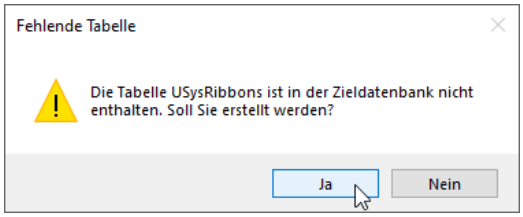


Abbildung 13.28: Frage, ob die Tabelle *USysRibbons* erstellt werden soll

Danach finden wir in der Datenbank den Eintrag *USysRibbons* im Navigationsbereich vor – vorausgesetzt, die Anzeige von Systemobjekten ist aktiviert. Ein Doppelklick auf diesen Eintrag zeigt die Tabelle mit der neu erstellten Ribbondefinition an (siehe Abbildung 13.29).

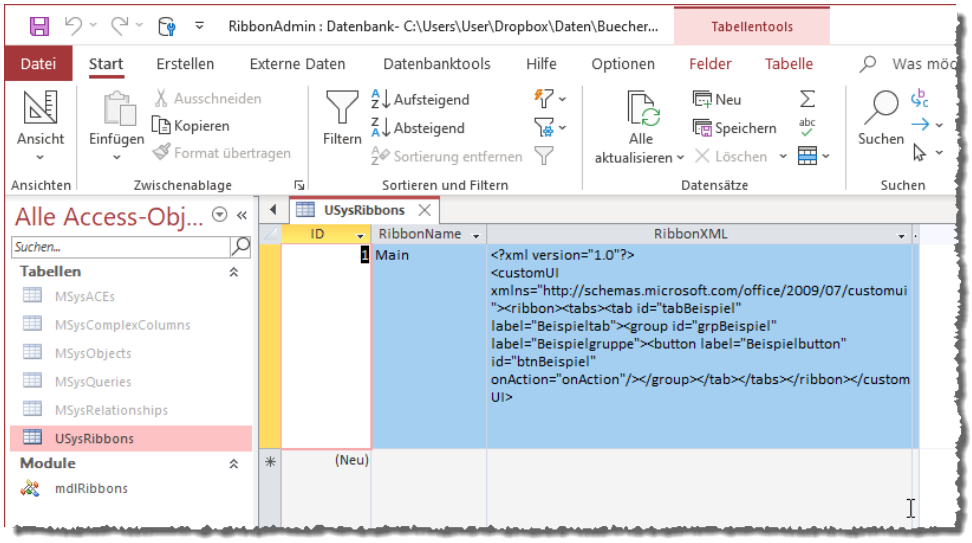


Abbildung 13.29: Die Tabelle *USysRibbons* mit der neu angelegten Ribbon-Definition

13.7 Ribbon als Anwendungsribbon festlegen

Bevor wir das Ribbon nun anzeigen können, müssen wir dieses noch als Anwendungsribbon einstellen. Auch das erledigen wir mit dem Ribbon-Admin, denn so sparen wir einen Schritt – wir

müssen so nicht erst die Datenbank schließen und wieder öffnen, um das Ribbon in den Access-Optionen vorzufinden und einstellen zu können.

Dazu kehren wir nochmals zum Ribbon-Admin zurück und wählen nun den Eintrag *Ribbon als Anwendungsribbon festlegen* für das Element mit der Bezeichnung *Main* aus (siehe Abbildung 13.30).

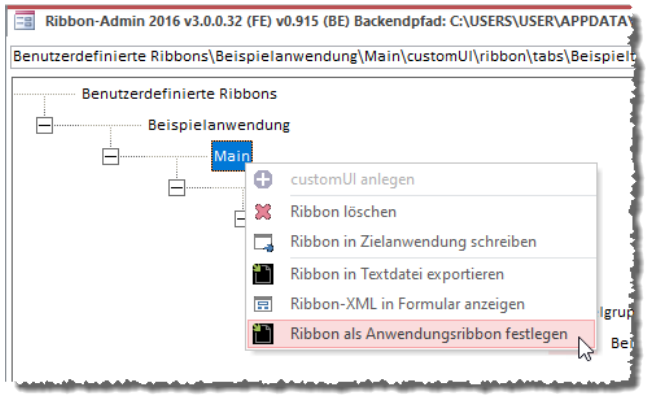


Abbildung 13.30: Festlegen des Ribbons als Anwendungsribbon

Nun brauchen wir nur noch die Anwendung zu schließen und erneut zu öffnen und schon erscheint unser neues Ribbon-Tab. Dieses wird allerdings zunächst noch nicht aktiviert (siehe Abbildung 13.32).

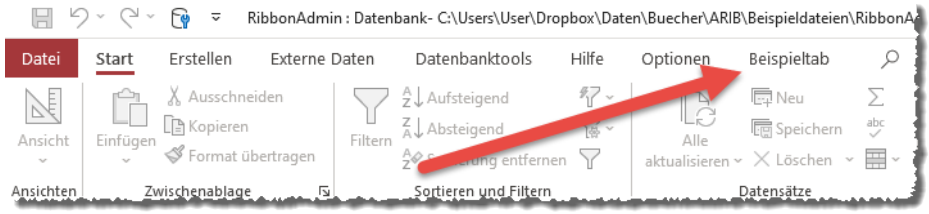


Abbildung 13.31: Unser neues Ribbon-Tab ist bereits sichtbar, aber noch nicht aktiviert.

Ein Klick auf Beispieltab zeigt jedoch unserer Gruppe mit der Beispielschaltfläche an. Ein Klick auf diese liefert schließlich die dadurch angezeigte Meldung (siehe Abbildung 13.32).

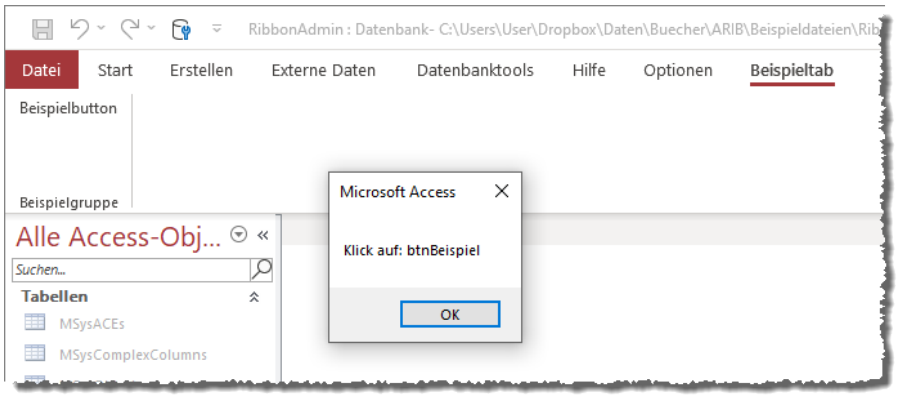


Abbildung 13.32: Das erste Beispielribbon ist vorhanden und die Schaltfläche funktioniert.

13.8 Bilder hinzufügen

Als Nächstes wollen wir der Schaltfläche ein Bild hinzufügen und ihre Größe anpassen. Bislang wird diese nämlich noch mit der voreingestellten Größe angezeigt, was dem Wert *normal* für das Attribut *size* entspricht.

Für die notwendigen Änderungen müssen wir also das Attribut *size* auf den Wert *large* einstellen und ein Bild zur Datenbank hinzufügen und das Attribut *image* auf den entsprechenden Namen einstellen. Die erste Aufgabe ist schnell erledigt, denn das Attribut *size* können wir über das Kontextmenü auf den Wert *large* festlegen.

Zum Hinterlegen und Einstellen des Bildes sind nur wenige Schritte mehr erforderlich. Dazu klicken sie mit der rechten Maustaste auf das Attribut *image*. Hier finden Sie nur einen einzigen Kontextmenüeintrag vor, nämlich *Benutzerdefiniertes Image hinzufügen*.

Diesen führen Sie aus und zeigen damit den Dialog aus Abbildung 13.33 an.

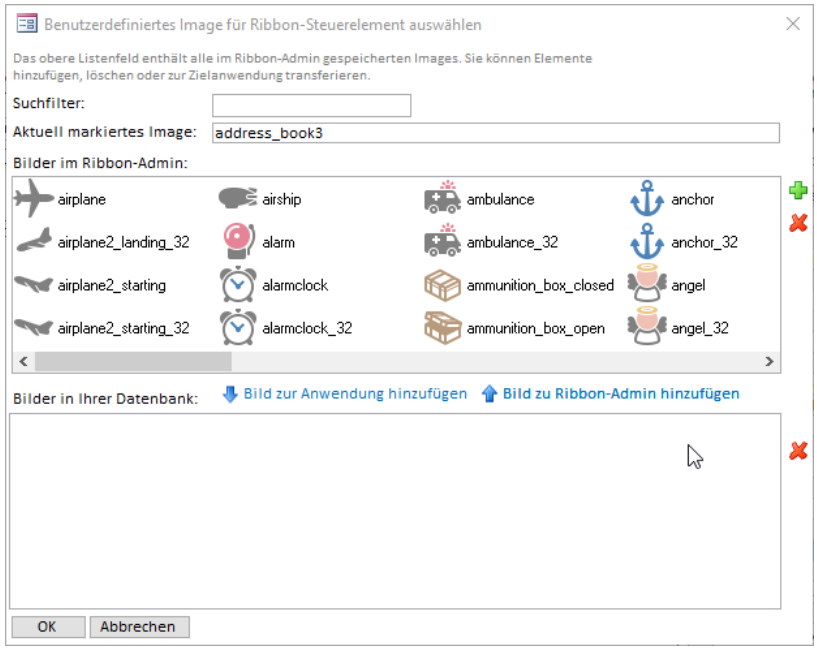


Abbildung 13.33: Dialog zum Auswählen des anzuzeigenden Bildes

Dieser Dialog zeigt normalerweise auch im oberen Bereich noch keine Icons an – diese kommen erst im Laufe der Zeit hinzu. Zum Verständnis: Die Bilder im oberen Bereich sind diejenigen, die Sie bereits einmal über den Ribbon-Admin zu einer Ihrer Anwendungen hinzugefügt haben. Der Ribbon-Admin speichert diese, damit Sie diese schnell wiederverwenden können.

Um einen Eintrag zu der oberen Liste hinzuzufügen, klicken Sie auf die *Plus*-Schaltfläche rechts von der Liste. Dies öffnet einen Dateiauswahldialog, mit dem Sie die *.png*-Datei auswählen können, die Sie im Ribbon anzeigen möchten.

Die Bilder sollten für kleine Icons im Format 16x16 und für große Icons im Format 32x32 Pixel vorliegen. Ein neu hinzugefügtes Icon wird zunächst in der oberen Liste angezeigt. Um diese für die Anzeige im Ribbon der aktuell geöffneten Anwendung verfügbar zu machen, markieren Sie das hinzuzufügende Icon im oberen Listenfeld und klicken dann auf die Schaltfläche Bild zur Anwendung hinzufügen. Das Icon erscheint dann zusätzlich in der unteren Liste (siehe Abbildung 13.34).

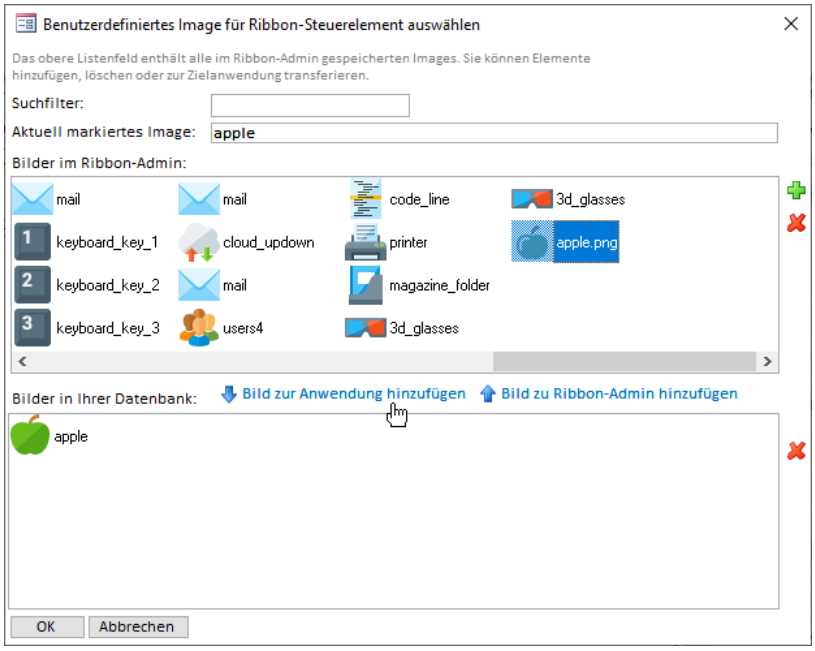


Abbildung 13.34: Image-Dialog mit einem neu hinzugefügten Icon

Außerdem wird das neu hinzugefügte Icon oben im Textfeld *Aktuell markiertes Image* eingetragen. Das bedeutet, dass Sie dieses nun durch einen Klick auf die *OK*-Schaltfläche für das *button*-Element übernehmen können.

Sobald Sie dies erledigt haben und die Eingabe in das Attribut *image* bestätigen, erscheint die Meldung aus Abbildung 13.35.

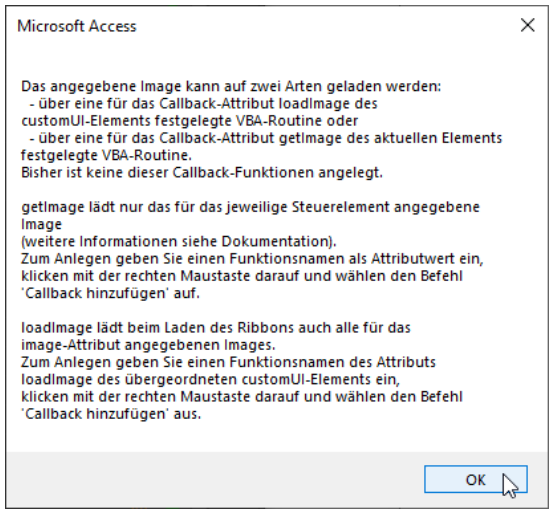


Abbildung 13.35: Hinweis auf weitere notwendige Aktionen

Diese weist darauf hin, dass Sie für das Anzeigen des Bildes noch einen weiteren Schritt erledigen müssen: Nämlich entweder eine Callbackprozedur für das Attribut *loadImage* des Elements *customUI* hinzuzufügen oder die *getImage*-Callbackprozedur für das *button*-Element hinzufügen.

Die praktischere Lösung ist meist, die *loadImage*-Callbackprozedur zu nutzen. Der Grund ist einfach: Sie brauchen diese dann nur einmal pro Ribbondefinition anzugeben. Und egal, wieviele Ribbondefinitionen Ihre Anwendung in der Tabelle *USysRibbons* enthält – Sie benötigen auch immer nur einmal die entsprechende Prozedur.

Gehen Sie also wie folgt vor:

» Klicken Sie auf das Element *customUI* der Ribbon-Definition, für die Sie das *loadImage*-Attribut füllen möchten.

» Klicken Sie dann mit der rechten Maustaste auf das Attribut *loadImage* und wähle Sie aus dem Kontextmenü den Wert *Callback hinzufügen* aus (siehe Abbildung 13.36).

» Die folgende Meldung bestätigt das Hinzufügen der Callbackroutine (siehe Abbildung 13.37).

» Wenn dies die erste Ribbondefinition ist, für die Sie die loadImages-Prozedur hinzufügen, erscheint auch noch die Rückfrage, ob Sie das Modul *mdlRibbonImages* zum VBA-Projekt der aktuellen Datenbank hinzufügen möchten. Dieses Modul enthält für die Anzeige von Icons essenziellen Code, sodass Sie diesem zustimmen sollten (siehe Abbildung 13.38).

» Das Hinzufügen dieses Moduls wird dann gegebenenfalls noch per Meldung bestätigt (siehe Abbildung 13.39).

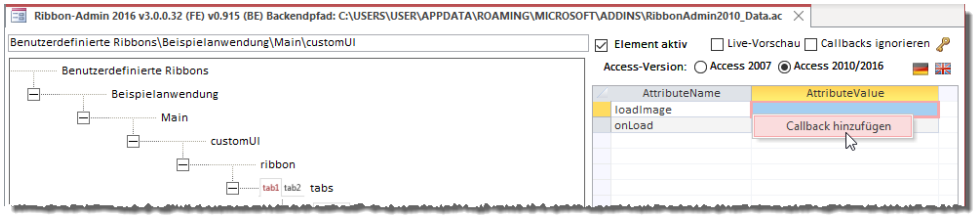


Abbildung 13.36: Hinzufügen der *loadPicture*-Prozedur

Abbildung 13.37: Die Callback-Funktion wurde hinzugefügt.

Abbildung 13.38: Rückfrage, ob das Modul *mdlRibbonImages* hinzugefügt werden soll

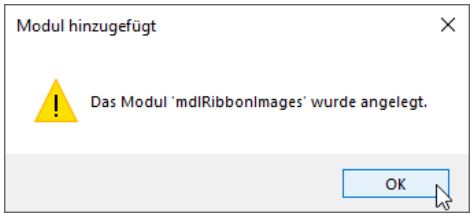


Abbildung 13.39: Bestätigung des neuen Moduls

Nachdem dies erledigt ist, finden Sie im Navigationsbereich von Access bereits die beiden Module *mdlRibbons* und *mdlRibbonImages* vor. Den Inhalt des Moduls *mdlRibbonImages* können Sie ignorieren, wichtig ist nur, dass das Modul vorhanden ist und die Prozedur *loadImages* auf diese zugreifen kann.

13.8.1 Testen des Ribbons mit Bild

Um das Ribbon mit dem neu hinzugefügten Bild auszuprobieren, übertragen Sie die Definition, indem Sie mit der rechten Maustaste auf den Eintrag *Main* (oder den jeweils für die Ribbondefinition vergebenen Namen) klicken und aus dem nun erscheinenden Kontextmenü den Befehl Ribbon in Zielanwendung schreiben wählen. Falls noch nicht geschehen, können Sie das Ribbon noch als Anwendungsribbon festlegen. Dazu nutzen Sie den Eintrag *Ribbon als Anwendungsribbon festlegen* des gleichen Kontextmenüs.

Wenn Sie nun die Datenbankdatei schließen und erneut öffnen, wechseln Sie zum Ribbontab *Beispieltab* und sehen dann das soeben hinzugefügte Icon (siehe Abbildung 13.40).

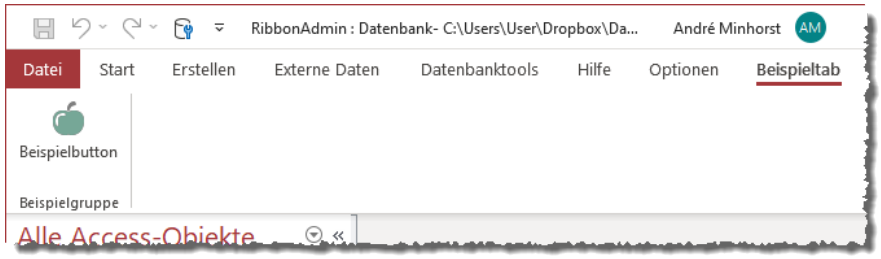


Abbildung 13.40: Das Ribbon mit einem Button, der ein Icon anzeigt

13.8.2 Bilder für weitere Steuerelemente

Wenn Sie weiteren *button*-Elementen Bilder hinzufügen wollen, wiederholen Sie einfach nur die folgenden Schritte:

» Fügen Sie über den Kontextmenüeintrag *Benutzerdefiniertes Image hinzufügen* des Attributs *image* des Steuerelements das gewünschte Bild hinzu. Dieses wird dann automatisch in der Datenbank gespeichert.

» Übertragen Sie die geänderte Ribbondefinition mit dem Kontextmenüeintrag *Ribbon in Zielanwendung schreiben* des zu übertragenden Ribbons.

» Schließen und öffnen Sie die Datenbankdatei, um das Ribbon zu aktualisieren.

Wenn Sie anderen Steuerelementen Bilder hinzufügen wollen, können Sie das ebenfalls jeweils über das *image*-Attribut erledigen.

13.9 Eingebaute Elemente hinzufügen

Mit dem Ribbon-Admin können Sie auch die eingebauten Ribbon-Elemente in ihre eigenen Elemente einbauen – also alle Elemente, deren *idMso*-Wert Sie kennen. Diesen finden Sie bei den Steuerelementen heraus, indem Sie in den Access-Optionen zum Bereich *Menüband anpassen* wechseln und dort den gewünschten Befehl in der linken Liste finden. Verharren Sie dann mit dem Mauszeiger über dem passenden Eintrag, blendet Access einige Informationen inklusive der englischen *idMso*-Bezeichnung ein (siehe Abbildung 13.41).

Diese Bezeichnung können Sie sich merken und gleich einsetzen.

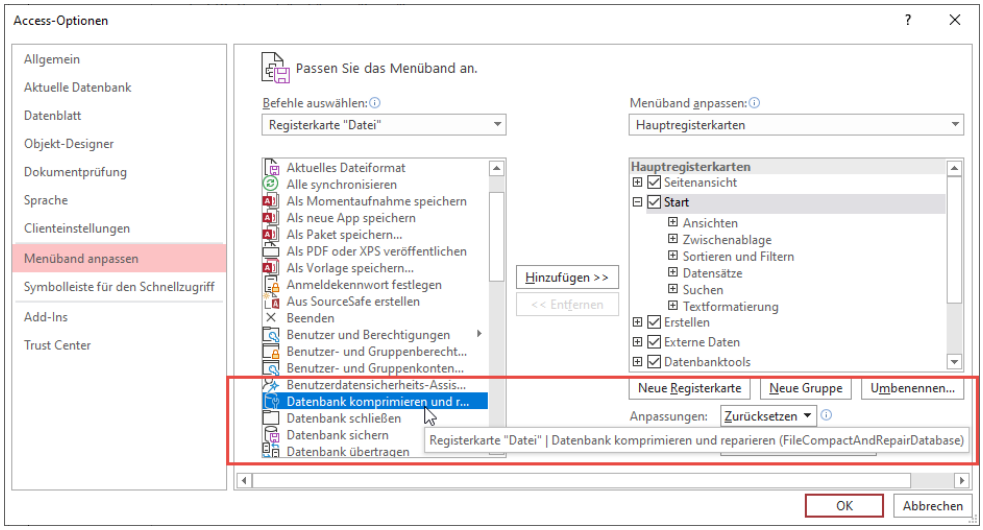


Abbildung 13.41: Ermitteln der *idMso* für ein Element

Wenn Sie etwa die Schaltfläche zum Komprimieren und Reparieren einer Datenbank in einem benutzerdefinierten Ribbon einfügen wollen, erledigen Sie das, indem Sie ein neues *control*-Element hinzufügen und dann sein Attribut *idMso* auf *FileCompactAndRepairDatabase* einstellen. Das Attribut *id* müssen Sie hingegen leeren, da nur eines der drei Attribute *id*, *idMso* oder *idQ* verwendet werden darf. Außerdem sollten Sie das Attribut *label* leeren, damit die eigentlich vorgesehene Bezeichnung angezeigt wird. Wenn Sie nun noch das Attribut *size* auf *large* einstellen, erhalten Sie eine Schaltfläche wie in Abbildung 13.42.

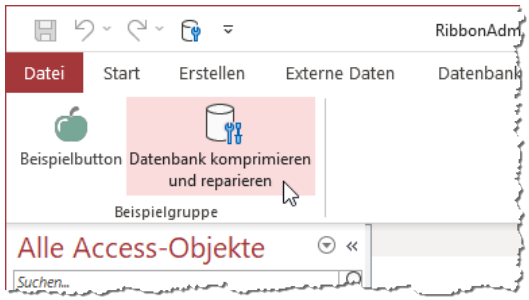


Abbildung 13.42: *control*-Element mit *idMso*

Dies war die grundlegende Vorgehensweise, wenn Sie die *idMso* kennen – falls nicht, können Sie für *tab*-, *group*- und *control*-Elemente wie nachfolgend beschrieben vorgehen.

13.9.1 Eingebaute tab-Elemente hinzufügen

tab-Elemente können Sie nicht über die *idMso*-Bezeichnung hinzufügen. Sie können lediglich ein neues *tab*-Element hinzufügen und diesem dann die *group*-Elemente anhand der entsprechenden *idMso*-Werte unterordnen. Das heißt, wenn Sie beispielsweise das *tab*-Element *Erstellen* nachbilden wollen, legen Sie ein *tab*-Element mit der Beschriftung *Erstellen* an und fügen dann *group*-Elemente hinzu, die den eingebauten Elementen entsprechen – also zum Beispiel *GroupAccessTemplates* oder *GroupCreateTables*.

Diese Schritte können Sie manuell erledigen oder Sie verwenden den Kontextmenüeintrag *Eingebautes tab einfügen* des *tabs*-Elements im Ribbon-Admin (siehe Abbildung 13.43).

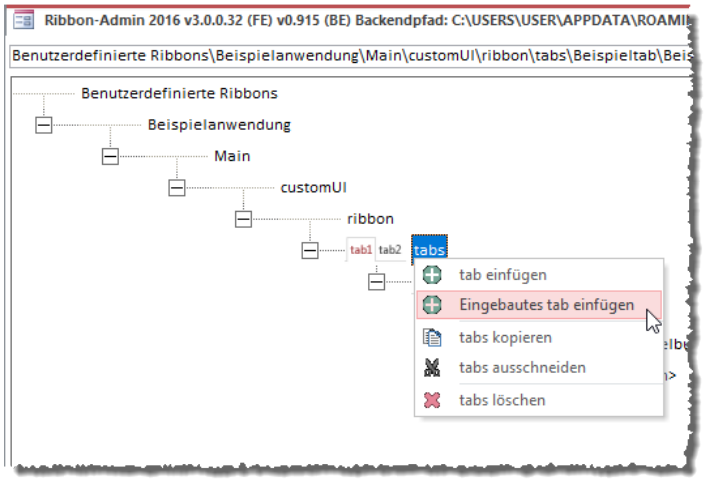


Abbildung 13.43: Hinzufügen eines eingebauten *tab*-Elements

Dies zeigt den Dialog *Eingebaute Elemente hinzufügen* an. Dieser erlaubt zunächst das Auswählen eines Eintrags für die Option *TabSet*. Hier finden Sie verschiedene Elemente. Diejenigen, die mit *TabSet...* beginnen, sind Zusatztabs, die im Zusammenhang mit verschiedenen Objekten wie Tabellen oder Formularen in der Datenblatt- oder Entwurfsansicht erscheinen. Die Ribbons, die immer sichtbar sind wie *Erstellen*, *Externe Daten* oder *Datenbanktools*, finden Sie unter *None (Core Tab)* – siehe Abbildung 13.44.

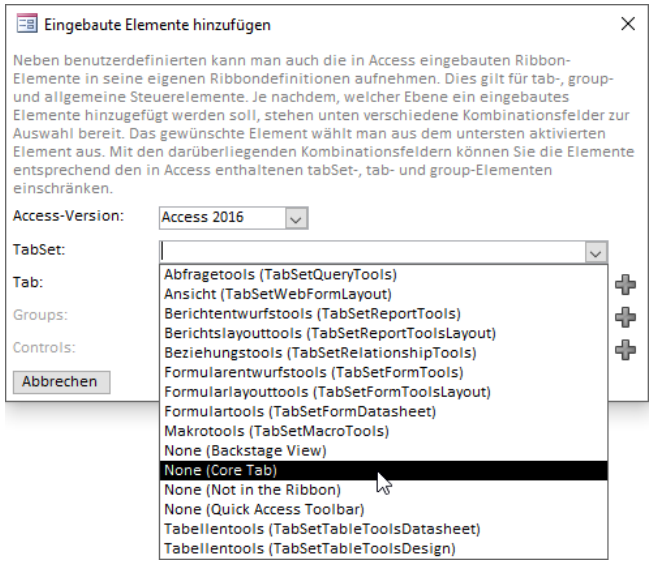


Abbildung 13.44: Auswählen eines *TabSet*-Eintrags

Wählen Sie diesen Eintrag aus, zeigt das nächste Kombinationsfeld *Tab* die *tab*-Elemente für dieses eingebaute *TabSet* an. Hier selektieren wir beispielsweise den Eintrag *Erstellen (TabCreate)* – siehe Abbildung 13.45.

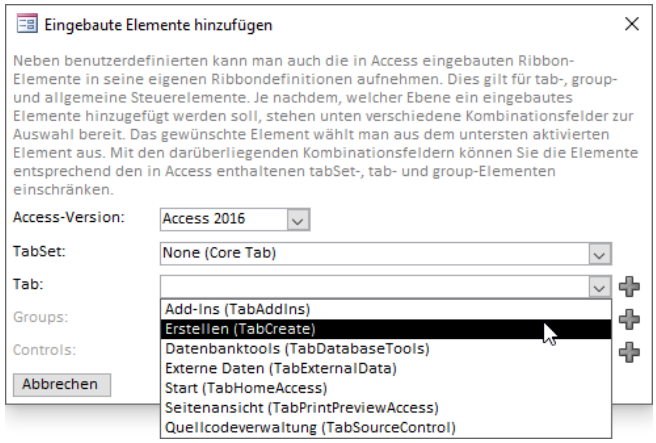


Abbildung 13.45: Auswahl eines *tab*-Elements

Danach aktiviert der Dialog die *Plus*-Schaltfläche rechts neben dem ausgewählten *Tab*-Eintrag (siehe Abbildung 13.46).

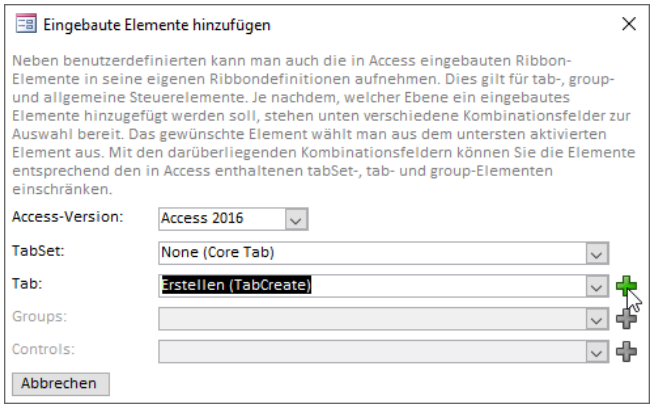


Abbildung 13.46: Das *Erstellen*-Tab kann nun hinzugefügt werden.

Danach sehen wir, was damit gemeint ist, dass die eingebauten *tab*-Elemente nicht direkt nachgebildet werden können. Das würde bedeuten, dass hier nur ein *tab*-Element mit einem bestimmten Wert im Attribut *idMso* angelegt worden wäre. Das ist aber nicht der Fall, wie Abbildung 13.47 zeigt.

Hier sehen Sie stattdessen zwar ein neues *tab*-Element mit der Beschriftung *Erstellen*. Dieses enthält aber eine ganze Reihe untergeordneter *group*-Elemente. *group*-Elemente können Sie im Gegensatz zu *tab*-Elemente nachbilden, indem wir ein *group*-Element hinzufügen und für dieses den entsprechenden *idMso*-Wert einstellen – in diesem Fall beispielsweise *GroupAccessTemplates*, *GroupCreateTablesWeb* (wird in Desktopdatenbanken nicht angezeigt) oder *GroupCreateTables*.

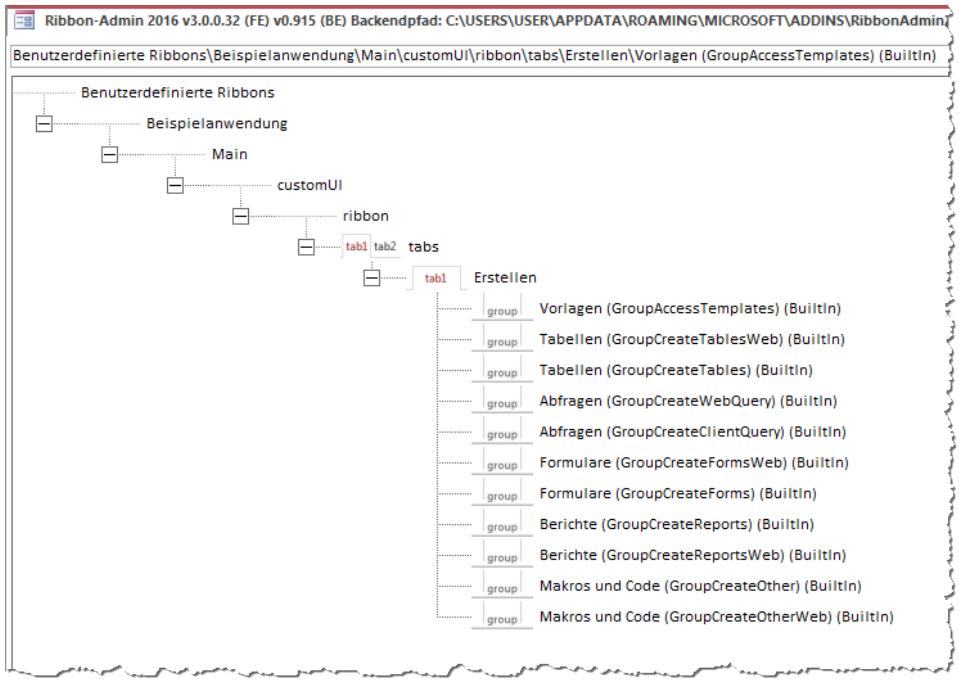


Abbildung 13.47: Das nachgebaute *tab*-Element *Erstellen*

Wenn wir das Ribbon nun in die Datenbank übertragen und diese erneut öffnen, finden wir als rechtes Tab eine Kopie des *Erstellen*-Tabs vor (siehe Abbildung 13.48).

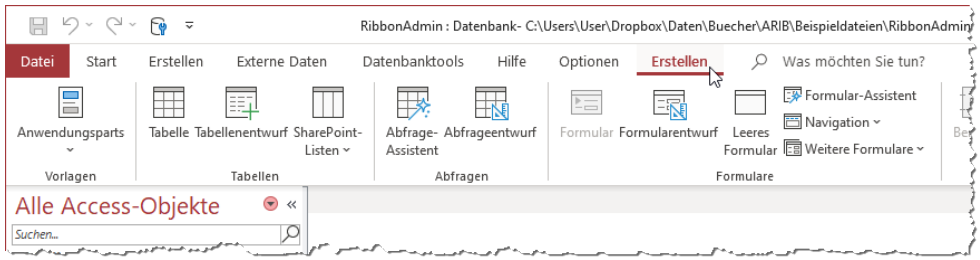


Abbildung 13.48: Das nachgebaute *Erstellen*-Tab

13.9.2 Eingebaute group-Elemente hinzufügen

Mit der gleichen Vorgehensweise können Sie eingebaute *group*-Elemente zu einem *tab*-Element hinzufügen. Wir wollen ein *tab*-Element erstellen, dass nur die Steuerelemente zum Erstellen

von Tabellen aus dem *Erstellen*-Tab enthält. Dazu fügen wir zunächst ein *tab*-Element namens *tabTabellenErstellen* hinzu und versehen es mit der Beschriftung *Tabellen erstellen*.

Anschließend wählen wir für dieses den Kontextmenüeintrag *Eingebaute group einfügen* aus (siehe Abbildung 13.49).

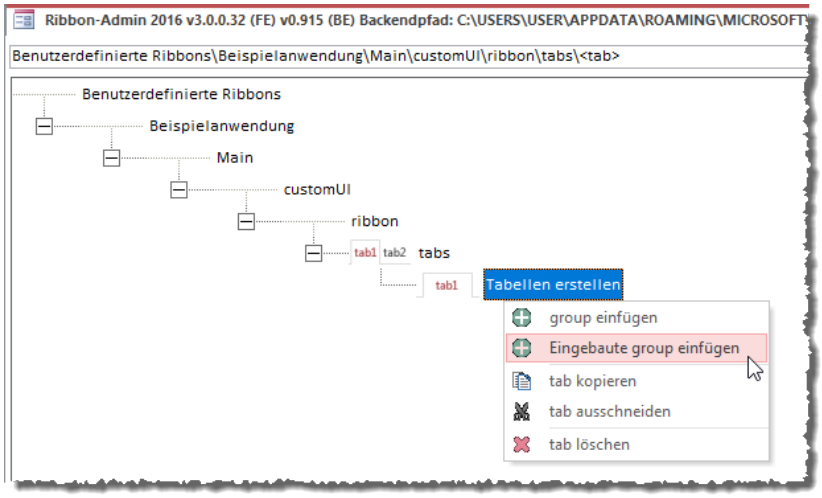


Abbildung 13.49: Hinzufügen eines eingebauten *group*-Elements

Im Dialog *Eingebaute Elemente hinzufügen* ist nun neben den Kombinationsfeldern *TabSet* und *Tab* auch noch das Kombinationsfeld *Groups* aktiviert. Hier können Sie entweder direkt einen Eintrag aus allen verfügbaren Gruppen auswählen oder Sie nutzen die beiden übergeordneten Kombinationsfelder, um die Gruppen zu filtern. Wenn Sie beispielsweise nur die Gruppen des *tab*-Elements *Erstellen (TabCreate)* sehen wollen, selektieren Sie unter *TabSet* den Eintrag *None (Core Tab)* und unter *Tab* den Eintrag *Erstellen (TabCreate)*. Danach zeigt das Kombinationsfeld *Groups* nur noch die Gruppen des *Erstellen*-Tabs an (siehe Abbildung 13.50).

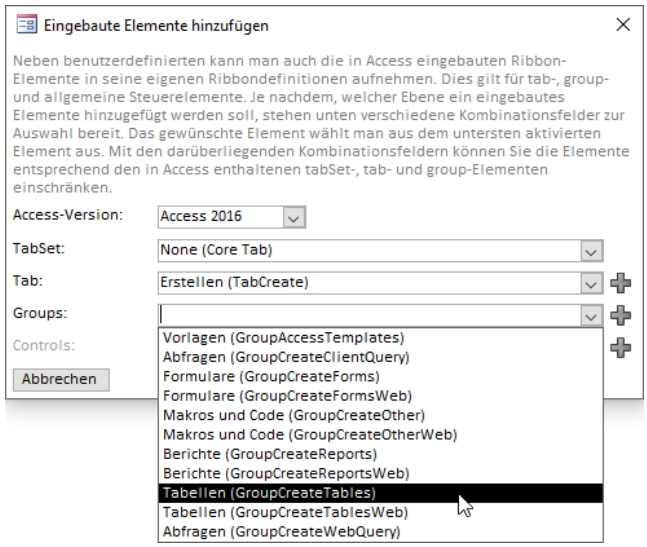


Abbildung 13.50: Auswahl eines eingebauten *group*-Elements

Wählen Sie hier den Eintrag *Tabellen (GroupCreateTables)* aus und klicken anschließend auf die rechts daneben befindliche *Plus*-Schaltfläche, fügt dies ein *group*-Element zum *tab*-Element hinzu und stellt den Wert des Attributs auf *GroupCreateTables* ein (siehe Abbildung 13.51).

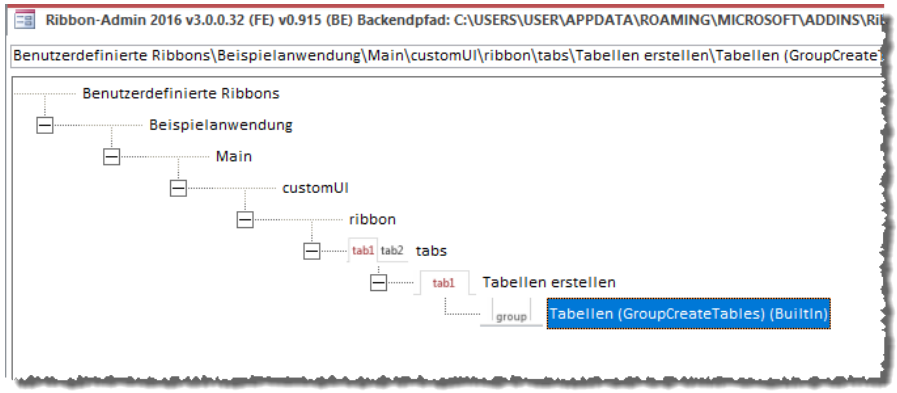


Abbildung 13.51: Ein eingebautes *group*-Element im Ribbon-Admin

Übertragen Sie die Ribbondefinition nun in die Accessdatenbank und starten diese erneut, finden Sie das *tab*-Element *Tabellen erstellen* mit der Gruppe aus Abbildung 13.52 vor.

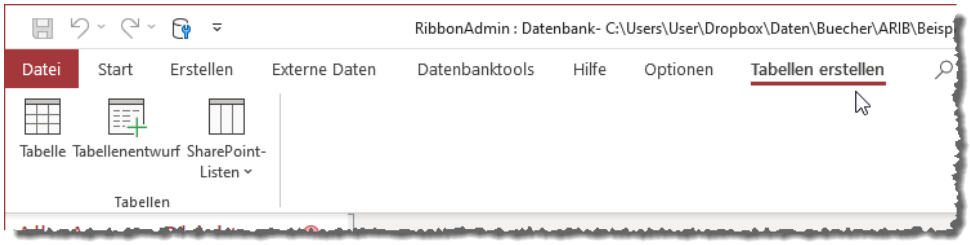


Abbildung 13.52: Ribbon mit der Gruppe zum Erstellen neuer Tabellen

13.10 Eingebaute Steuerelemente hinzufügen

Wenn Sie nicht wie oben beschrieben erst die *idMso* eines Steuerelements ermitteln wollen, um dieses dann manuell hinzuzufügen, können Sie auch für Steuerelemente den Dialog *Eingebaute Elemente hinzufügen* nutzen. Dazu wählen Sie aus dem Kontextmenü eines *group*-Elements den Eintrag *Eingebautes Control einfügen* aus (siehe Abbildung 13.53).

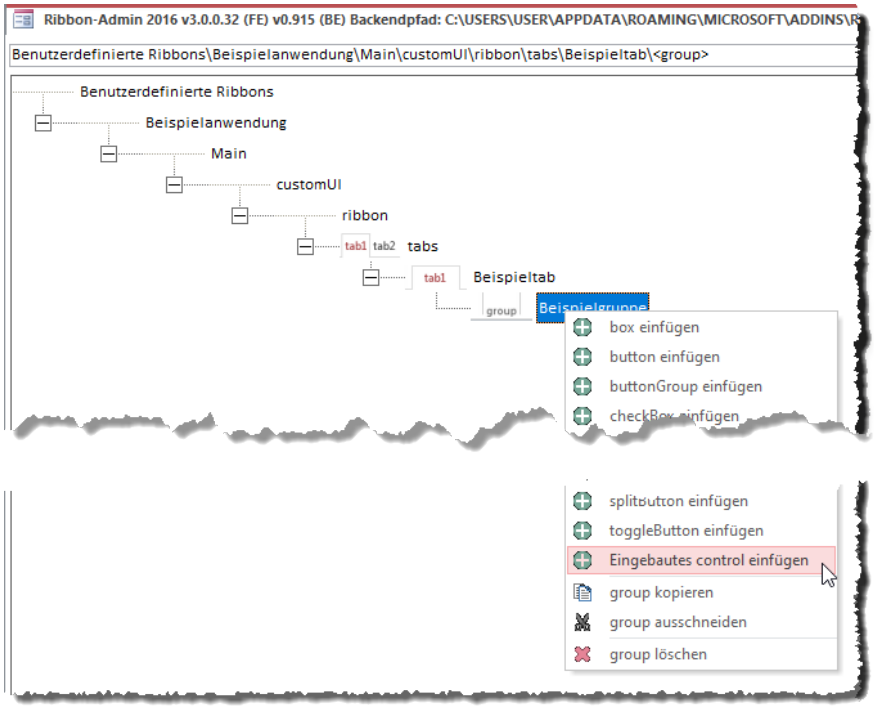


Abbildung 13.53: Hinzufügen von eingebauten Steuerelementen

Hier haben Sie wieder die Möglichkeit, zu filtern – diesmal nach *TabSet*, *Tab* und *Groups* (siehe Abbildung 13.54). Danach oder auch ohne vorheriges Filtern wählen Sie einen der Einträge des Kombinationsfeldes *Controls* aus.

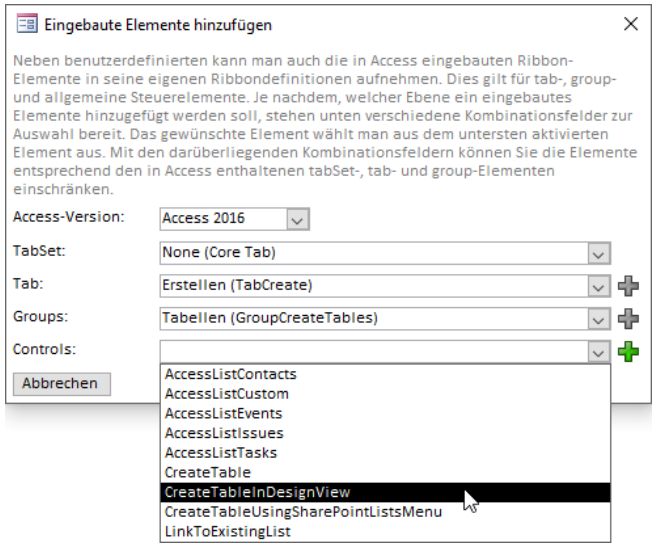


Abbildung 13.54: Hinzufügen eines eingebauten Steuerelements

Betätigen Sie anschließend die *Plus*-Schaltfläche rechts neben dem gewählten Steuerelement, fügt dies dem *group*-Element ein neues *control*-Element hinzu, für dessen Attribut *idMso* der entsprechende Wert angelegt wird. Hier können Sie gegebenenfalls noch die Größe über das Attribut *size* auf den Wert *large* einstellen, falls Sie das Steuerelement groß darstellen wollen.

13.11 Weitere Steuerelemente anlegen

Sie können alle weiteren Steuerelemente genauso anlegen, wie wir es zuvor für Elemente wie das *tab*-, *group*- oder *button*-Element beschrieben haben. Sie rufen einfach den entsprechenden Eintrag im Kontextmenü des Elements auf, unter dem Sie das neue Elemente hinzufügen wollen.

Es gibt einige Elemente, die aus mehr als einem Element bestehen, wie zum Beispiel das *splitButton*-Element. Dieses enthält ein *button*- oder ein *toggleButton*-Element sowie darunter ein *menu*-Element zur Auswahl weiterer Schaltlfächen. Wenn Sie zuerst ein *splitButton*-Element angelegt haben, können Sie anschließend die untergeordneten Elemente wie in Abbildung 13.55 hinzufügen. In manchen Fällen können Sie nur ein Exemplare eines untergeordneten Elements anlegen. Das ist auch beim *splitButton*-Element der Fall. Wenn Sie hier bereits ein *button*- oder

toggleButton-Element angelegt haben, können Sie danach nur noch ein *menu*-Element hinzufügen und umgekehrt. Der Ribbon-Admin erlaubt es aber nicht, beispielsweise zwei *menu*-Elemente unterhalb eines *splitButton*-Elements anzulegen.

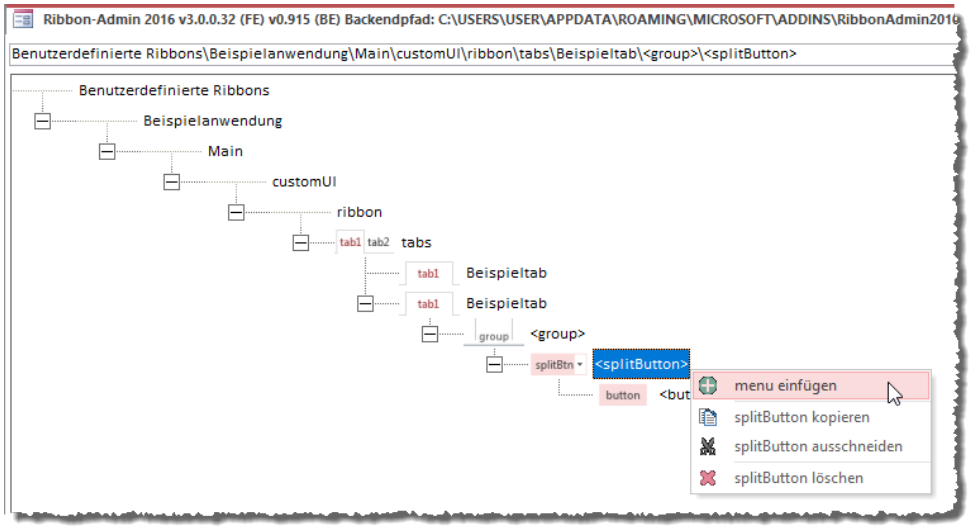


Abbildung 13.55: Hinzufügen der Unterelemente eines *splitButton*-Elements

13.12 Elemente löschen

Das Löschen von Elementen, die Sie im Ribbon-Admin hinzugefügt haben, erledigen Sie über den jeweiligen Kontextmenüeintrag des Elements. Dieser ist beispielsweise bezeichnet mit *tab löschen*, *group löschen* oder *button löschen*. Wenn Sie beispielsweise ein *tab*-Element löschen wollen, klicken Sie dieses mit der rechten Maustaste an und wählen aus dem Kontextmenü den Eintrag *tab löschen* aus (siehe Abbildung 13.56).

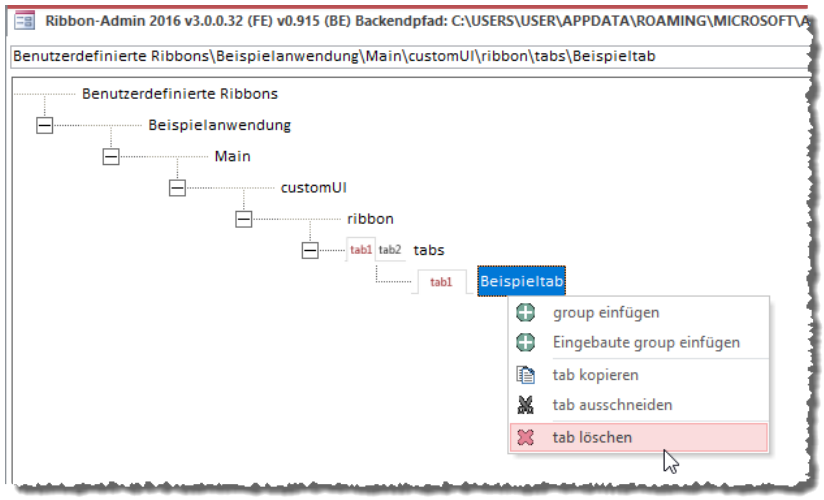


Abbildung 13.56: Löschen eines *tab*-Elements

Wenn das zu löschende Elemente noch untergeordnete Elemente enthält, müssen Sie noch eine Meldung bestätigen, bevor das Element samt untergeordneten Elementen gelöscht wird.

13.13 Elemente kopieren, ausschneiden und einfügen

Sie können die hinzugefügten Elemente auch kopieren oder ausschneiden und an anderer Stelle innerhalb Ihrer benutzerdefinierten Ribbondefinitionen einfügen. So lassen sich beispielsweise einzelne Steuerelemente, aber auch komplette *ribbon*-Elemente von einer Ribbondefinition zur nächsten kopieren oder verschieben. Dazu sind folgende Schritte nötig:

» Betätigen Sie das Kontextmenü des zu kopierenden oder auszuschneidenden Elements und wählen Sie den Befehl *<Elementbezeichnung> kopieren* oder *<Elementbezeichnung> ausschneiden* aus.

» Klicken Sie mit der rechten Maustaste auf das Element, dem sie das kopierte oder ausgeschnittene Element unterordnen möchten und wählen Sie aus dem Kontextmenü den Eintrag *<Elementbezeichnung> '<Elementname>' einfügen* aus.

Wenn Sie beispielsweise ein *tab*-Element namens *Beispieltab* kopiert oder ausgeschnitten haben und das Kontextmenü des *tabs*-Elements in einer beliebigen Ribbondefinition aufrufen, finden Sie dort den Eintrag *tab 'Beispieltab' einfügen* vor (siehe Abbildung 13.57).

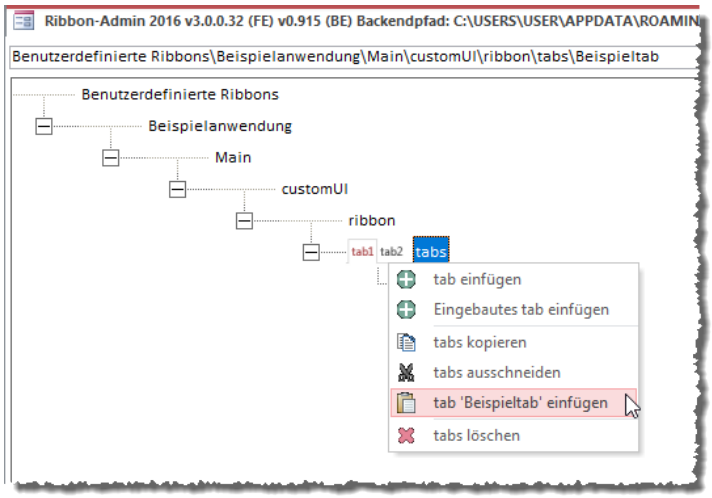


Abbildung 13.57: Einfügen eines zuvor kopierten oder ausgeschnittenen Elements

Die meisten Attribute werden beim Kopieren übernommen. Die Ausnahme ist das Attribut *id*. Dieses wird nicht übernommen, um Konflikte mit gleichnamigen Elementen in der gleichen Ribbondefinition zu verhindern.

13.14 Callback-Attribute nutzen

Wenn Sie Callback-Attribute, also solche Attribute, die mit *get...* beginnen, nutzen wollen, finden Sie im Ribbon-Admin eine Erleichterung. Angenommen, Sie möchten, dass ein *button*-Steuerelement abhängig vom Wert einer *Boolean*-Variablen gesteuert wird. Dann fügen Sie zunächst ein *button*-Element zu einer Gruppe hinzu. Außerdem stellen wir einige Attribute ein wie *image*, *size*, *id* und *label* sowie *onAction*. Danach folgt das für uns interessante Attribut, nämlich *getEnabled*. Für dieses wählen wir den Kontextmenüeintrag *Callback hinzufügen* aus (siehe Abbildung 13.58).

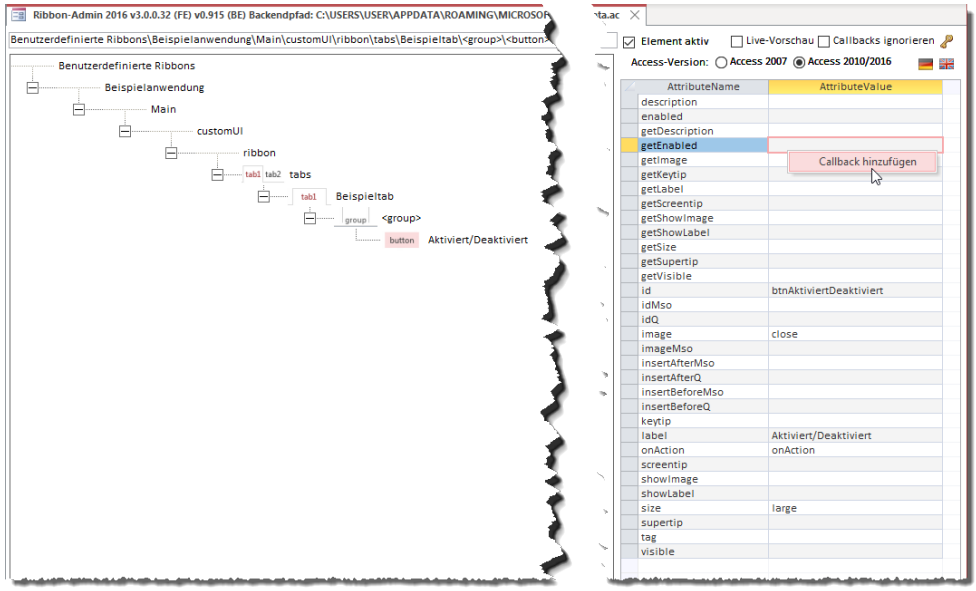


Abbildung 13.58: Vorbereitung der Schaltfläche und Hinzufügen des *getEnabled*-Attributs

Dies quittiert der Ribbon-Admin mit dem Hinweis, dass eine entsprechende Callback-Funktion angelegt wurde (siehe Abbildung 13.59).

Abbildung 13.59: Hinweis auf das Anlegen einer Callback-Funktion

Das allein reicht jedoch noch nicht. Wir müssen außerdem noch eine Prozedur für das Attribut *onLoad* des *customUI*-Elements hinterlegen. Dazu klicken Sie zuerst auf das *customUI*-Element und wählen für sein Attribut *onLoad* den Kontextmenüeintrag *Callback hinzufügen* auf.

Dies bringt zunächst die Meldung aus Abbildung 13.60 hervor. Diese gibt an, dass für jede Rib bondefinition eine eigene Objektvariable zum Referenzieren dieser Definition angelegt wird.

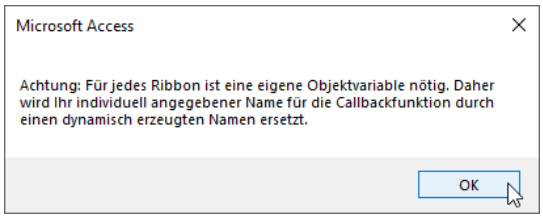


Abbildung 13.60: Hinweis, dass eine eigene Objektvariable für jede Ribbondefinition angelegt wird

Danach erscheint noch eine weitere Meldung, die über das Anlegen der neuen Callback-Funktion im Modul *mdlRibbons* informiert (siehe Abbildung 13.61).

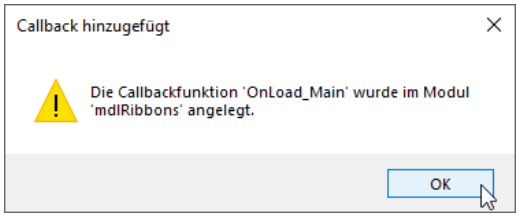


Abbildung 13.61: Hinweis auf die neu angelegte Callback-Funktion

Schauen wir nach dem Festlegen des Attributs *onLoad* in das Modul *mdlRibbons*, finden Sie die folgenden neuen Elemente vor:

```
Public objRibbon_Main As IRibbonUI

Sub onLoad_Main(ribbon As IRibbonUI)
    Set objRibbon_Main = ribbon
End Sub
```

Das erste ist eine Objektvariable, mit der die Ribbondefinition referenziert werden kann. Diese Referenzierung nimmt die Prozedur *onLoad_Main* vor, die direkt beim Laden des Ribbons aufgerufen wird. Wir können also davon ausgehen, dass diese Variable beim ersten Anzeigen des Ribbons mit einem Verweis darauf gefüllt ist.

Wir können nun also beispielsweise eine der Methoden *invalidate* oder *invalidateControl* der Variablen *objRibbon_Main* aufrufen, um dafür zu sorgen, dass die *get...*-Callback-Funktionen beim nächsten Erscheinen der betroffenen Steuerelemente erneut aufgerufen werden, um die entsprechenden Attributwerte zu aktualisieren.

Desweiteren haben wir weiter oben bereits die Callback-Funktion *getEnabled* angelegt. Diese sieht zunächst wie folgt aus:

```
Sub getEnabled(control As IRibbonControl, ByRef enabled)
```

```
End Sub
```

Wenn wir die *getEnabled*-Funktion auf einfache Weise testen wollen, legen wir im oberen Teil des Moduls die folgende *Boolean*-Variable an:

```
Public bolEnabled As Boolean
```

Den Wert dieser Variablen weisen wir in der *getEnabled*-Funktion wie folgt dem Rückgabeparameter *enabled* zu:

```
Sub getEnabled(control As IRibbonControl, ByRef enabled)
    enabled = bolEnabled
End Sub
```

Wenn wir die Ribbondefinition nun in die Datenbank übertragen und diese erneut öffnen, wird die Schaltfläche zunächst einmal deaktiviert dargestellt. Der Grund ist, dass die Variable *bolEnabled* standardmäßig den Wert *False* enthält. Wird das Ribbon nun geladen, ruft das Ribbon für das *button*-Element den Wert der Funktion *getEnabled* ab und erhält somit den Wert *False* für das Attribut *enabled* (siehe Abbildung 13.62).

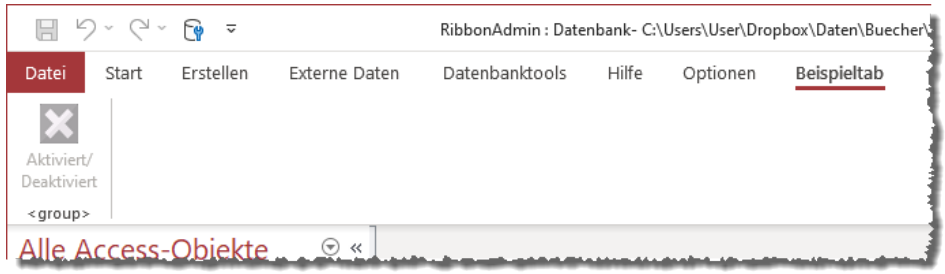


Abbildung 13.62: Per *getEnabled* deaktivierte Schaltfläche

Nun fügen wir dem Modul *mdlRibbons* im VBA-Projekt zwei kleine Prozeduren hinzu:

```
Public Sub Aktivieren()
    bolEnabled = True
    objRibbon_Main.Invalidate
End Sub

Public Sub Deaktivieren()
    bolEnabled = False
    objRibbon_Main.Invalidate
End Sub
```

Die erste stellt die Variable *bolEnabled* auf den Wert *True* ein und ruft dann die Methode *Invalidate* des *CustomUI*-Elements auf. Dadurch wird, wenn die Schaltfläche mit dem *getEnabled*-Attribut gerade sichtbar ist, die *getEnabled*-Funktion erneut aufgerufen und der Wert von *bolEnabled* erneut abgerufen. Da dieser nun *True* lautet, wird die Schaltfläche nun als aktiviert dargestellt.

Den gegenteiligen Effekt sehen Sie, wenn Sie die zweite Prozedur *Deaktivieren* aufrufen.

IRibbonUI-Variablen bei mehreren Ribbon-Definitionen

Es kann sein, dass Sie mehr als eine Ribbon-Definition gleichzeitig nutzen – beispielsweise, wenn Sie eine als Anwendungsribbon verwenden und eine als Ribbon, das mit einem Formular oder einem Bericht eingeblendet wird. Damit Sie immer noch über die *IRibbonUI*-Variable, die in der Prozedur *onLoad* gefüllt wird, die Methoden *Invalidate* oder *InvalidateControl* aufrufen können, wird für jede Ribbondefinition sowohl eine eigene *IRibbonUI*-Variable als auch eine eigene *onLoad*-Prozedur angelegt.

13.15 Ribbondefinition im Formular anzeigen

Manchmal möchten Sie prüfen, ob die vom Ribbon-Admin erstellte Ribbondefinition Ihren Vorstellungen entspricht. Dann können Sie den Kontextmenüeintrag *Ribbon-XML in Formular anzeigen* des Elements für das aktuelle Ribbon aufrufen (siehe Abbildung 13.63).

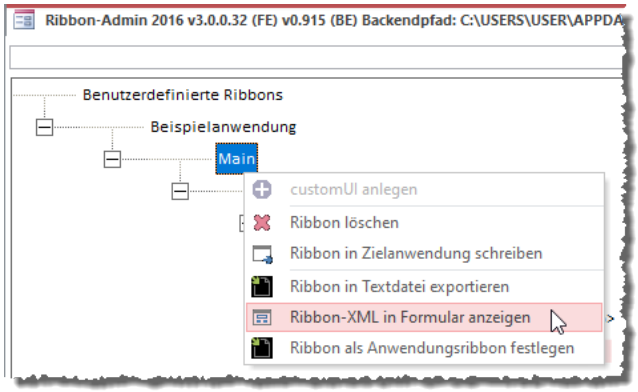


Abbildung 13.63: Aufruf der Anzeige der XML-Ribbondefinition in einem Formular

Dies zeigt die Ribbondefinition im XML-Format in einem Textfeld in einem Accessformular an (siehe Abbildung 13.64).

Abbildung 13.64: Die Ribbondefinition im Formular

Mit einem Klick auf die Schaltfläche *Anzeigen* sorgen Sie dafür, dass die Ribbondefinition angewendet wird und im Ribbon der aktuellen Anwendung erscheint.

André Minhorst

Ribbon Programmierung

LESEPROBE

Hier finden Sie in der Vollversion noch weiteres Know-how rund um die Ribbon-Programmierung!

André Minhorst

Ribbon Programmierung

LESEPROBE

Hier finden Sie in der Vollversion noch weiteres Know-how rund um die Ribbon-Programmierung!

14 Ribbon-Lösungen

In diesem Kapitel stellen wir kleine Lösungen vor, mit denen Sie das Ribbon für Ihre eigenen Anwendungsfälle erweitern können oder die zeigen, was mit dem Ribbon alles möglich ist.

14.1 Beim Wechsel zu einem Tab ein Formular anzeigen

Microsoft verwöhnt den Office-Entwickler nicht gerade mit Ereignissen im Ribbon. Es gibt ein bis zwei Ereignisattribute für einige Steuerelemente, eines, das beim Anzeigen einer Ribbondefinition ausgeführt wird – und das war es schon fast. Was aber, wenn Sie andere Ereignisse benötigen? Zum Beispiel, um beim Wechsel von *tab*-Elementen ein spezielles Formular für das jeweilige *tab*-Element einzublenden? Hierfür gibt es kein eingebautes Ereignis. Also müssen wir ein wenig improvisieren.

Das Beispiel zu diesem Abschnitt finden Sie in der folgenden Beispieldatenbank:

```
NeueFormBeiTabwechsel.accdb
```

14.1.1 Aufgabenstellung

Die Lösung aus diesem Abschnitt soll zeigen, wie Sie durch einen Klick auf eines der *tab*-Elemente des Ribbons direkt ein entsprechendes Formular öffnen können.

Unser Ribbon hat drei Tabs namens *tabKunden*, *tabArtikel* und *tabPersonal*. Dazu enthält die Lösung drei Formulare namens *frmKunden*, *frmArtikel* und *frmPersonal*. Direkt beim Öffnen der Anwendung wird das *tab*-Element *tabKunden* angezeigt und damit soll auch das Formular *frmKunden* erscheinen (siehe Abbildung 14.1).

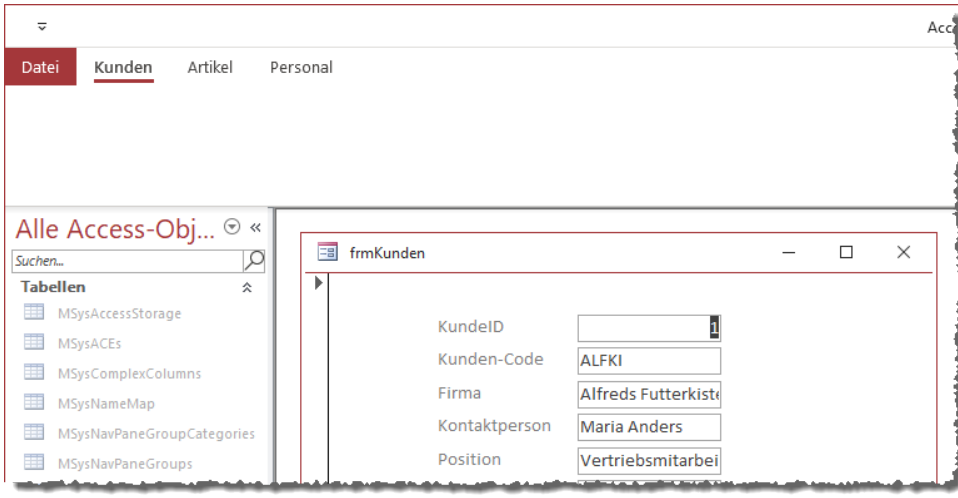


Abbildung 14.1: Mit dem *tab*-Element *tabKunden* soll auch das Formular *frmKunden* erscheinen.

Wenn der Benutzer auf das *tab*-Element *tabArtikel* klickt, soll sich das Formular *frmArtikel* öffnen (siehe Abbildung 14.2), klickt er auf *tabPersonal*, soll das Formular *frmPersonal* erscheinen.

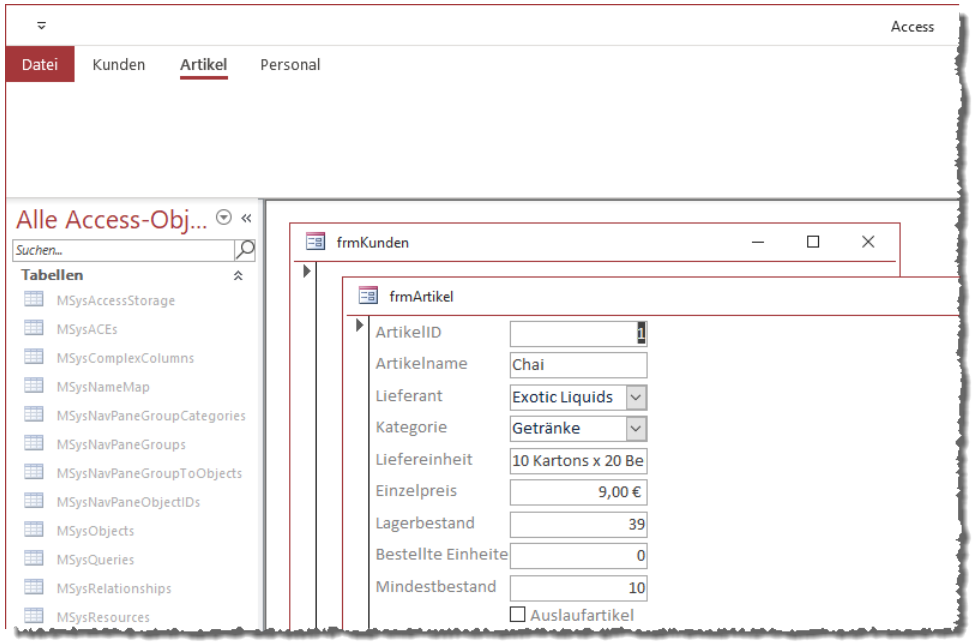


Abbildung 14.2: Anzeige des Formulars *frmArtikel* beim Klick auf das Tab *tabArtikel*

Ist eines der Formulare bereits geöffnet und der Benutzer klickt auf ein *tab*-Element, das gerade nicht aktiviert ist, dann soll das entsprechende Formular aktiviert werden.

14.1.2 Passendes Ereignisattribut finden

Wenn wir uns nun die Attribute des *tab*-Elements ansehen, stellen wir fest, dass dieses kein Attribut anbietet, für das wir beispielsweise beim Anzeigen oder Aktivieren eines *tab*-Elements ein Ereignis auslösen können. Generell bietet das Ribbon quasi keine Ereignisse, die durch das Anklicken oder Wechseln eines *tab*-Elements ausgelöst werden. Das *tab*-Element bietet nur die folgenden *get…*-Callbackattribute:

» *getKeytip*

» *getLabel*

» *getVisible*

Diese Attribute erwarten Prozeduren, welche die Werte für die Attribute *Keytip*, *Label* und *Visible* des *tab*-Elements liefern.

Diese Prozeduren werden aufgerufen, wenn die *tab*-Elemente erstmalig angezeigt werden oder nachdem die *Invalidate*-Methode für die Ribbondefinition oder die *InvalidateControl*-Methode für ein einzelnes Ribbonelement aufgerufen wurde.

Das *customUI*-Element bietet ein Ereignisattribut namens *onLoad* an, aber das wird nur einmalig beim Laden der Ribbon-Definition ausgelöst.

Das hilft alles nur wenig weiter – wir benötigen ja schließlich ein Ereignis, dass immer beim Aktivieren eines *tab*-Elements ausgelöst wird, damit wir dann das jeweilige Formular anzeigen können.

14.1.3 Ereignis auf Umwegen

Da die *get…*-Callbackattribute immer beim ersten Anzeigen ausgelöst werden (oder beim erneuten Anzeigen, wenn zwischendurch die *Invalidate*– oder die *InvalidateControl*-Methode aufgerufen wurde), könnten wir sie eigentlich auch nutzen! Wir müssen hier nur den Befehl zum Öffnen des jeweiligen Formulars unterbringen und dafür sorgen, dass die *tab*-Elemente immer nach dem Auswählen eines anderen *tab*-Elements wieder »invalidiert« werden.

Allerdings kann es sein, dass Sie die drei *get…*-Attribute des *tab*-Elements für andere Zwecke benötigen.

Also fügen wir einfach zu jedem *tab*-Element ein *button*-Element ein, wobei das *button*-Element einzig und allein der Anzeige des Formulars zum jeweiligen *tab*-Element dient.

Die Ribbon-Definition gestalten wir wie folgt:

```
<?xml version="1.0"?>
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui"
    onLoad="OnLoad_Main">
  <ribbon startFromScratch="true">
    <tabs>
      <tab id="tabAKunden" tag="Kunden" label="Kunden" getKeytip="getKeytip">
        <group id="grpKunden" label="Kundendaten" getKeytip="getKeytip">
          <button label="Beispielschaltfläche 1" getVisible="getVisible"
            id="btnKunden" tag="frmKunden"/>
        </group>
      </tab>
      <tab id="tabArtikel" tag="Artikel" label="Artikel" getKeytip="getKeytip">
        <group id="grp2" label="Artikeldaten" getKeytip="getKeytip">
          <button tag="frmArtikel" label="Beispielschaltfläche 2"
            getVisible="getVisible" id="btnArtikel"/>
        </group>
      </tab>
```

André Minhorst

Ribbon Programmierung

LESEPROBE

Hier finden Sie in der Vollversion noch weiteres Know-how rund um die Ribbon-Programmierung!

André Minhorst

Ribbon Programmierung

LESEPROBE

Hier finden Sie in der Vollversion noch weiteres Know-how rund um die Ribbon-Programmierung!

15 Ribbons in COM-Add-Ins

Wer mit Access programmiert, hat sicher bereits einmal ein Access-Add-In verwendet. Diese erstellen Sie mit den Bordmitteln von Access. Auch für ein Access-Add-In können Sie ein Ribbon definieren – dieses müssten Sie dann mit einem Formular des Access-Add-Ins anzeigen. Es gibt jedoch noch eine andere Art außer Access-Add-Ins, um Access mit selbst programmieren Funktionen zu erweitern. Dabei handelt es sich um die sogenannten COM-Add-Ins. Diese können Sie für Access selbst als auch für den VBA-Editor programmieren. Da es in diesem Buch um das Ribbon geht und nur Access das Ribbon anbietet und nicht der VBA-Editor, schauen wir uns in diesem Kapitel die Anpassung des Ribbons per COM-Add-In an.

Das Beispielprojekt finden Sie im Verzeichnis *AccessOptionen* des Downloads zu diesem Buch.

15.1 COM-Add-Ins erstellen – aber wie?

Um ein COM-Add-In zu erstellen, gibt es unter anderem die folgenden drei Möglichkeiten:

» mit Visual Studio 6 und Visual Basic 6

» mit Visual Studio .NET und einer der dort verfügbaren Programmiersprachen wie Visual Basic oder C#

» mit Visual Studio Code und twinBASIC

Die erste Variante ist deshalb wenig sinnvoll, weil Sie damit ausschließlich COM-Add-Ins auf Basis von 32-Bit erstellen können. Diese sind nicht mehr mit Microsoft Office (und somit auch Microsoft Access) in der 64-Bit-Version kompatibel. Seit der Version 2019 wird bei der Installation von Office oder Access jedoch standardmäßig die 64-Bit-Version installiert, weshalb Sie mit der Programmierung von reinen 32-Bit-Erweiterungen nicht mehr alle Benutzer erreichen können.

Natürlich können Sie auch mit Visual Studio und einer der Programmiersprachen wie Visual Basic oder C# arbeiten, wenn Sie Erweiterungen wie COM-Add-Ins für Microsoft Access erstellen wollen. Allerdings gibt es dafür keine vorgefertigten Vorlagen mehr, wie es für die übrigen Office-Anwendungen wie beispielsweise Word, Excel oder Outlook der Fall ist. Davon abgesehen geht es in diesem Buch vorrangig um die Ribbonprogrammierung auf Basis der für Office zur Verfügung stehenden Möglichkeiten und .NET verwendet einen ganz anderen Ansatz.

Bleibt noch die Programmierung von COM-Add-Ins mit Visual Studio Code und der neuen Programmiersprache twinBASIC. Diese wurde von Wayne Philips ins Leben gerufen und entwickelt. Damit können Sie mit den mit Visual Studio 6 und Visual Basic 6 entwickelten COM-Add-Ins kompatible Erweiterungen programmieren. Hier kommt genau der gleiche Ansatz zur Definition des Ribbons zum Einsatz wie auch unter Access.

Da wir mit twinBASIC darüberhinaus extrem schlanke Erweiterungen programmieren können, schauen wir uns diesen Ansatz im vorliegenden Kapitel an.

COM-Add-Ins für ständig verfügbare Erweiterungen

Wenn Sie dauerhafte Ergänzungen oder Änderungen am Ribbon vornehmen wollen, benötigen Sie ein COM-Add-In. Eine andere Möglichkeit gibt es nur, wenn Sie mit den eingebauten Funktionen zum Anpassen der Benutzeroberfläche arbeiten wollen – und die sind bei Weitem nicht ausreichend.

15.2 Visual Studio Code und twinBASIC installieren

Bevor Sie Visual Studio Code und twinBASIC zur Erstellung von COM-Add-Ins nutzen können, müssen Sie beides herunterladen und installieren.

15.2.1 Visual Studio Code installieren

Den Download von Visual Studio Code finden Sie unter der folgenden Adresse:

```
https://code.visualstudio.com
```

Hier laden Sie die Setup-Datei herunter, die zum Zeitpunkt der Erstellung dieses Buchs *VSCode-UserSetup-x64-1.64.2.exe* heißt. Die Installation starten Sie nach dem Download mit einem Doppelklick auf diese Datei.

15.2.2 twinBASIC zu Visual Studio Code hinzufügen

twinBASIC müssen Sie nicht separat herunterladen, sondern Sie können es direkt von Visual Studio Code aus installieren.

Dazu starten Sie Visual Studio Code. Falls noch nicht geschehen, akzeptieren Sie die gewünschten Optionen, die beim ersten Start angezeigt werden.

Anschließend finden Sie das Hauptfenster von Visual Studio Code vor. Auf der linken Seite gibt es einige Schaltflächen, mit denen Sie verschiedene Bereiche einblenden können. Uns interessiert aktuell der Bereich *Extensions*, denn über diesen können Sie Erweiterungen zu Visual Studio Code hinzufügen.

Aktivieren Sie diesen Bereich und geben Sie im Suchfeld den Begriff *twinBASIC* ein. Visual Studio Code zeigt nun die aktuelle Version von twinBASIC an. Klicken Sie auf die *Install*-Schaltfläche, um die twinBASIC-Erweiterung unter Visual Studio Code zu installieren (siehe Abbildung 15.1).

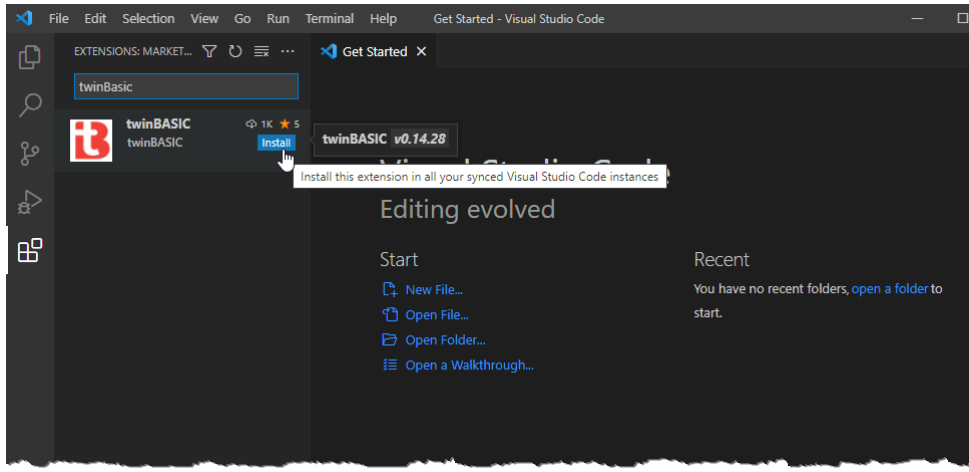


Abbildung 15.1: Hinzufügen von twinBASIC zu Visual Studio Code

Damit ist die Installation bereits abgeschlossen und Sie können mit dem Programmieren beginnen.

15.3 COM-Add-In für Access programmieren

In den folgenden Abschnitten zeigen wir, wie Sie mit Visual Studio Code und twinBASIC ein COM-Add-In inklusive Aufruf beziehungsweise Steuerung über das Ribbon erstellen können.

15.3.1 Neues COM-Add-In anlegen

Um ein neues COM-Add-In anzulegen, klicken Sie in Visual Studio Code links auf das *twinBASIC*-Symbol. Es erscheint eine Liste mit Projektvorlagen, wo Sie die Vorlage *Sample 5 MyCOMAddin* auswählen (siehe Abbildung 15.2).

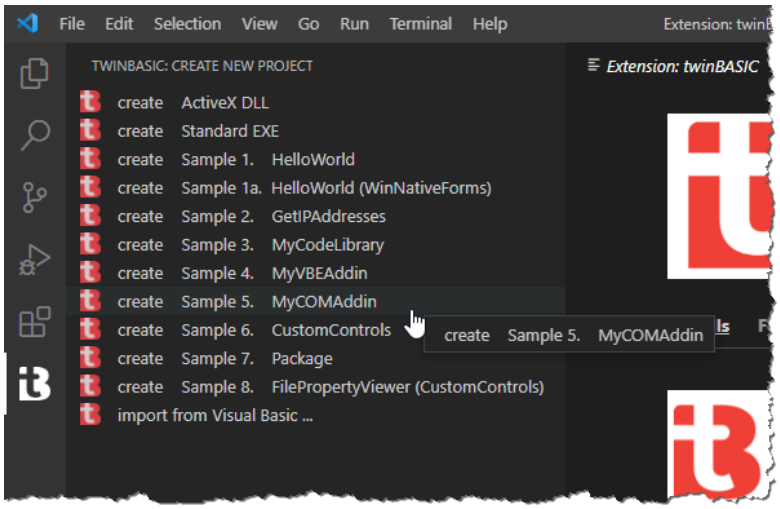


Abbildung 15.2: Erstellen eines neuen COM-Add-Ins

Anschließend erscheint ein Dialog, in dem Sie die anzulegende Datei samt Verzeichnis auswählen. Hier geben wir *COMAddInMitRibbon.twinproj* an (siehe Abbildung 15.3).

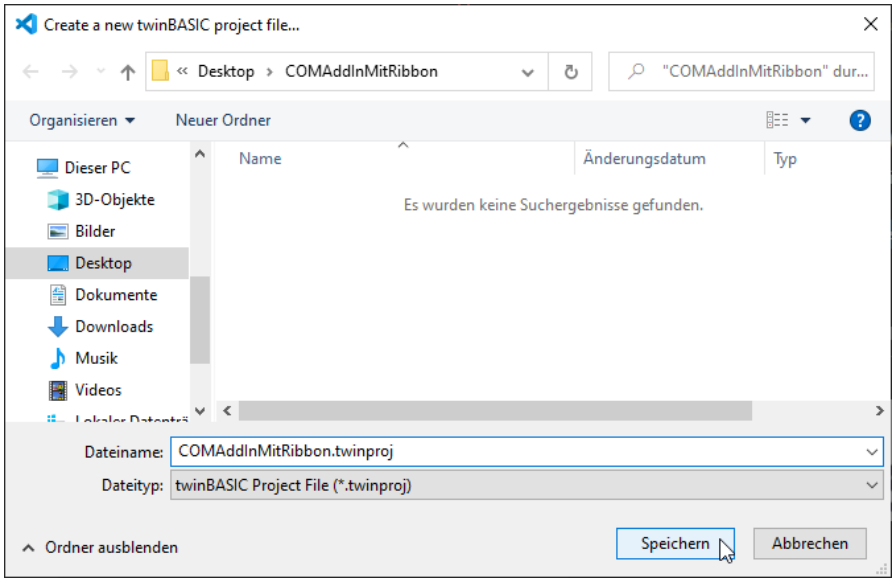


Abbildung 15.3: Angeben des Projektnamens

Dies fügt einige Dateien zum gewählten Verzeichnis hinzu, von denen die folgenden für uns interessant sind:

» *COMAddInMitRibbon_myCOMAddin.code-workspace*: Arbeitsumgebung

» *COMAddInMitRibbon_myCOMAddin.twinproj*: Projektdatei

Abbildung 15.4 zeigt die beiden Dateien im Windows Explorer.

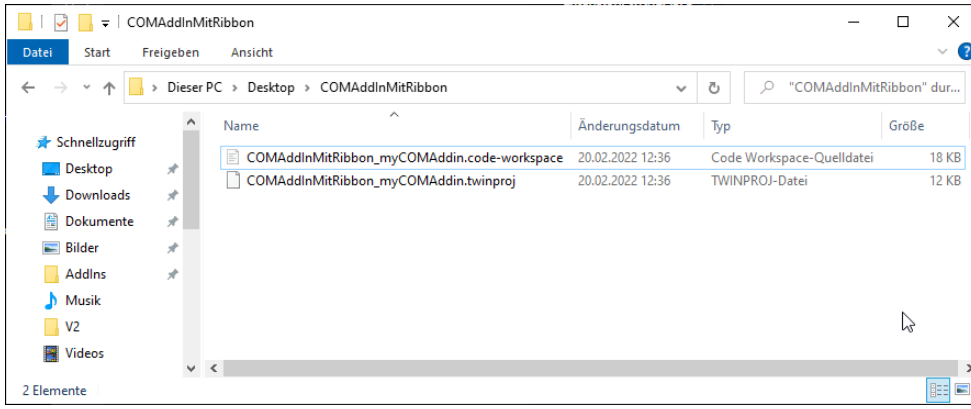


Abbildung 15.4: Durch das *twinBASIC*-Projekt angelegte Dateien

Das Projekt wird allerdings auch direkt in einem neuen Fenster in Visual Studio Code angezeigt (siehe Abbildung 15.5). Hier sehen Sie verschiedene Bereiche:

» Auf der linken Seite sehen Sie den Explorer, über den Sie die verschiedenen Objekte, Verweise, Pakete et cetera ansehen können.

» Auf der rechten, oberen Seite finden Sie oben den Code des aktuell im linken Bereich markierten Moduls, hier *HelloWorld.twin*. Dieser ist im vorliegenden Fall recht umfangreich, denn wir finden hier direkt ein vollständig funktionsfähiges COM-Add-In.

» Auf der rechten, unteren Seite finden Sie zum Beispiel die *Debug Console*.

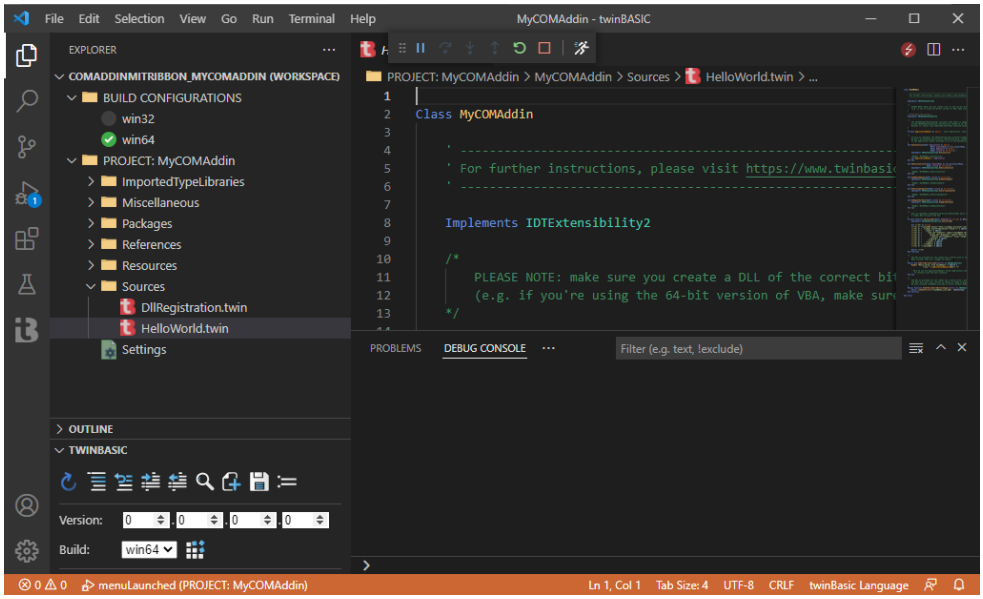


Abbildung 15.5: Das neue Projekt in Visual Studio Code

15.3.2 Was bietet die Vorlage

Mit der Vorlage erhalten Sie bereits fast alles, was Sie benötigen:

» Die Implementierung der Schnittstelle *IDTExtensibility2*, die einige Ereignisprozeduren bereitstellt, die zu verschiedenen Zeitpunkten beim Verwenden des COM-Add-Ins ausgelöst werden – zum Beispiel beim Start (*OnConnection*)

» Die Implementierung der Schnittstelle *IRibbonExtensibility* in Form der Ereignisprozedur *GetCustomUI*. Diese wird beim Start des COM-Add-Ins ausgelöst und stellt das Ribbon zusammen, über das die Benutzeroberfläche beziehungsweise die Funktionen des COM-Add-Ins bereitgestellt werden sollen.

» Eine von dieser Ereignisprozedur aufgerufene Callback-Funktion, die zeigt, wie Sie Funktionen über das Ribbon aufrufen können.

15.3.3 COM-Add-In zum Laufen bringen

Dementsprechend brauchen Sie nur einen Schritt zu erledigen, damit das COM-Add-In läuft:

» Das COM-Add-In kompilieren, indem Sie auf die Schaltfläche *Build* klicken (siehe Abbildung 15.6).

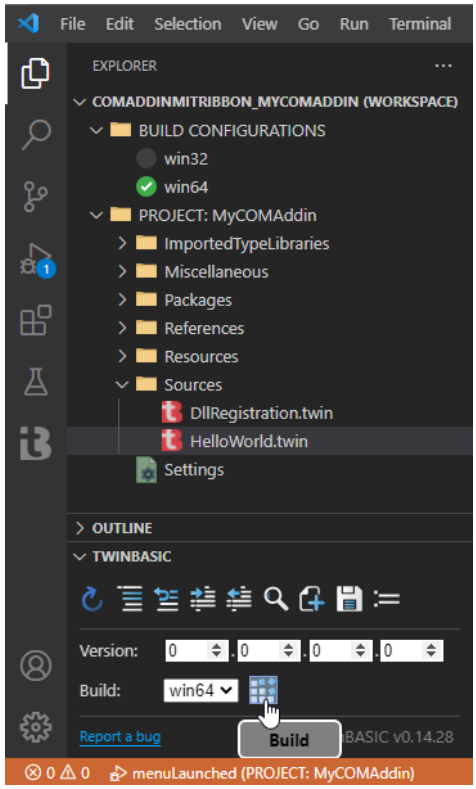


Abbildung 15.6: COM-Add-In kompilieren

Dies liefert die Ausgabe aus Abbildung 15.7 in der Debug Console.

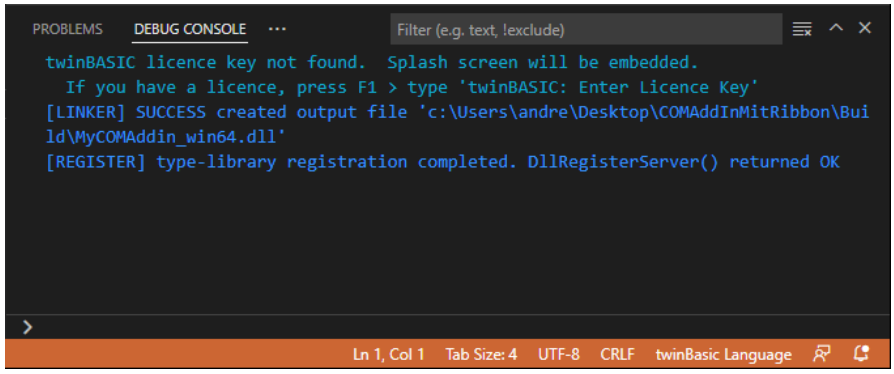


Abbildung 15.7: Meldung über den Fortschritt beim Erstellen der DLL

Hier sehen Sie, dass ein Lizenzschlüssel notwendig ist, da sonst ein Splash Screen von twinBASIC eingeblendet wird. Diese sieht wie in Abbildung 15.8 aus.

Abbildung 15.8: Splash Screen von twinBASIC

Wenn Sie nun eine Access-Datenbank öffnen, erscheint ein neuer Eintrag namens *twinBASIC Test* im Ribbon mit einer eigenen Schaltfläche. Klicken Sie auf diesen Ribboneintrag, erscheint die Meldung aus Abbildung 15.9.

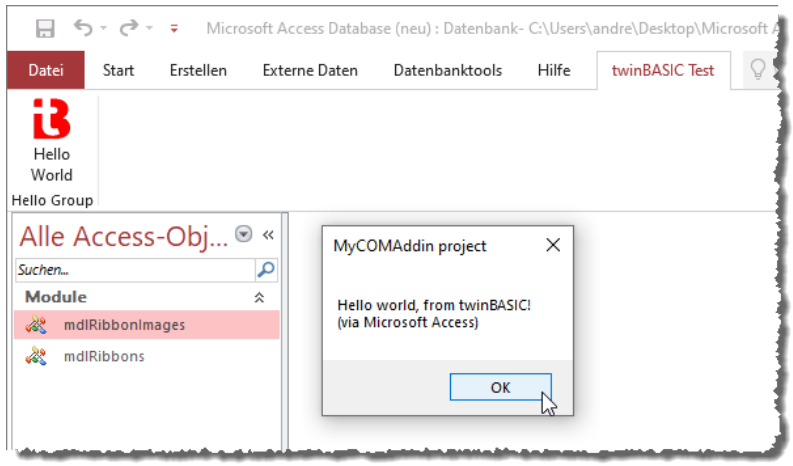


Abbildung 15.9: Das COM-Add-In ist nach wenigen Mausklicks einsatzbereit

15.3.4 Aufbau der einzigen Klasse des COM-Add-Ins

Bevor wir diese anpassen, schauen wir uns den grundlegen den Aufbau der Klasse an, die das COM-Add-In repräsentiert – so, wie Sie es durch das Beispielprojekt erstellt haben.

Die Klasse implementiert die Schnittstellen *IDTExtensibility2* und *IRibbonExtensibility*:

```
Class MyCOMAddin
    Implements IDTExtensibility2
    [ WithDispatchForwarding ]
    Implements IRibbonExtensibility
```

Außerdem deklariert sie eine Variable namens *applicationObjekt*, die gleich in der ersten Ereignisprozedur der Schnittstellt *IDTExtensibility2* namens *OnConnection* mit dem Verweis aus dem Parameter *Application* gefüllt wird. Dieser Parameter enthält, wenn das COM-Add-In beispielsweise von Access aus aufgerufen wird, einen Verweis auf die aktuelle Access-Instanz:

```
    Private applicationObject As Object
    Sub OnConnection(ByVal Application As Object, _
            ByVal ConnectMode As ext_ConnectMode, ByVal AddInInst As Object, _
            ByRef custom As Variant()) Implements IDTExtensibility2.OnConnection
        Set applicationObject = Application
    End Sub
```

Die übrigen Ereignisprozeduren der Schnittstelle *IDTExtensibility2* sehen wir folgt aus – wann diese feuern, können Sie den Namen bereits entnehmen:

```
Sub OnDisconnection(ByVal RemoveMode As ext_DisconnectMode, _
        ByRef custom As Variant()) Implements IDTExtensibility2.OnDisconnection
    'MsgBox "MyCOMAddin.OnDisconnection"
End Sub

Sub OnAddInsUpdate(ByRef custom As Variant()) _
        Implements IDTExtensibility2.OnAddInsUpdate
    'MsgBox "MyCOMAddin.OnAddInsUpdate"
End Sub

Sub OnStartupComplete(ByRef custom As Variant()) _
        Implements IDTExtensibility2.OnStartupComplete
    'MsgBox "MyCOMAddin.OnStartupComplete"
End Sub

Sub OnBeginShutdown(ByRef custom As Variant()) _
        Implements IDTExtensibility2.OnBeginShutdown
    'MsgBox "MyCOMAddin.OnBeginShutdown"
End Sub
```

Wichtig ist noch die folgende Ereignisprozedur, die wir als Teil der Schnittstelle *IRibbonExtensibility* implementieren. Diese erlaubt uns, innerhalb der Prozedur die anzuzeigende Ribbondefinition zusammenzustellen. Hier stellt die Prozedur eine minimale Konfiguration zusammen, die ein *button*-Element im *group*-Element eines *tab*-Elements darstellt.

Hier sehen Sie gleich eine praktische Vereinfachung der Programmiersprache *twinBASIC* gegenüber VBA – Sie können statt *strXML = strXML & "..."* einfach *strXML &= "..."* schreiben:

```
Private Function GetCustomUI(ByVal RibbonID As String) As String _
        Implements IRibbonExtensibility.GetCustomUI
    Dim strXML As String
    strXML &= "<customUI xmlns=""http://schemas.microsoft.com/office/2006/01/" 7
                                                          "customui"">" & vbCrLf
    strXML &= "  <ribbon startFromScratch=""false"">" & vbCrLf
    strXML &= "    <tabs>" & vbCrLf
    strXML &= "      <tab id=""tabTest"" label=""twinBASIC Test"">" & vbCrLf
    strXML &= "        <group id=""grpTwin1"" label=""Hello Group"">" & vbCrLf
    strXML &= "          <button id=""btnHello"" size=""large"" 7
                  label=""Hello World"" getImage=""OnGetHelloWorldButtonImage"" 7
                                      onAction=""OnHelloWorldClicked""/>" & vbCrLf
    strXML &= "        </group>" & vbCrLf
    strXML &= "      </tab>" & vbCrLf
```

```
        strXML &= "    </tabs>" & vbCrLf
        strXML &= "  </ribbon>" & vbCrLf
        strXML &= "</customUI>" & vbCrLf
        Return strXML
    End Function
```

Auch die Callback-Funktionen werden in dieser Klasse untergebracht. Im obigen Beispiel ist für das Attribut *onAction* die Funktion OnHelloWorldClicked angegeben, die wie folgt aussieht und ein Meldungsfenster anzeigen soll:

```
    Public Sub OnHelloWorldClicked(Control As IRibbonControl)
        MsgBox "Hello world, from twinBASIC!" & vbCrLf & "(via " ⤷
                                                & applicationObject.Name & ")"
    End Sub
```

Schließlich enthält die Klasse noch eine Funktion namens *OnGetHelloWorldButtonImage*, die den Verweis auf das aufrufende *IRibbonControl*-Element liefert und welche die Datei *twinBASIC_icon.bmp* aus den Ressourcen des Projekts zurückliefert:

```
    Public Function OnGetHelloWorldButtonImage(Control As IRibbonControl) As IPictureDisp
        Return LoadResPicture("twinBASIC_icon.bmp", vbResBitmap)
    End Function
End Class
```

Wie Sie diese Elemente für eigene Zwecke anpassen können, schauen wir uns in den folgenden Abschnitten an.

15.4 Beispiel: Optionen mit dem Ribbon einstellen

Um einen praktischen Anwendungsfall zu haben, wollen wir mit dem COM-Add-In ein Ribbon bieten, über das Sie Access-Optionen anzeigen und einstellen können. Um solche Optionen mit dem Ribbon einzustellen, benötigen wir mehr als nur ein paar Schaltflächen. Immerhin wollen wir ja vor dem Ändern einer Option auch wissen, wie ihr aktueller Wert lautet! Also benötigen wir Steuerelemente wie Kontrollkästchen, Textfelder et cetera.

Da wir diese dynamisch einlesen und auch anpassen wollen, stellen wir ihre Werte zu Beginn mit entsprechenden Callback-Funktionen ein. Später ändern wir diese dann über das Ribbon und sorgen dafür, dass die Änderungen sich auch in den entsprechenden Access-Optionen niederschlagen.

15.4.1 Projekt erstellen

Als Erstes erstellen wir, wie oben beschrieben, ein neues *twinBASIC*-Projekt des Typs *MyCOM-AddIn* und speichern es in einem Unterverzeichnis namens *AccessOptionen*.

Den Dateinamen ändern wir in *AccessOptionen.twinproj*.

Das neue Projekt zeigt nun gegebenenfalls für die beiden Einträge *BUILD CONFIGURATIONS* und *PROJECT: MyCOMAddin* die Meldung *Unable to resolve workspace folder an* (siehe Abbildung 15.10).

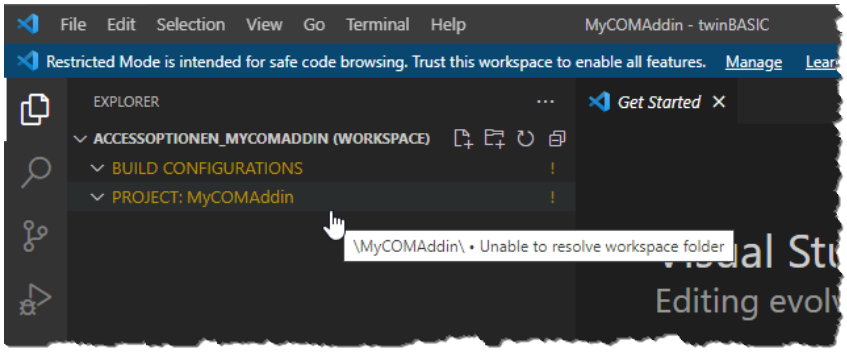


Abbildung 15.10: Problem mit dem Projekt

Das Problem lösen Sie wie folgt:

» Klicken Sie unten links auf die Schaltfläche *Manage*.

» Im nun erscheinenden Menü klicken Sie auf *Manage Workspace Trust* (siehe Abbildung 15.11).

» Dies öffnet einen weiteren Bereich, dem wir entnehmen können, das wir gerade in einer Art gesichertem Zustand arbeiten. Um das zu ändern, klicken Sie wie in Abbildung 15.12 auf die Schaltfläche *Trust*.

Danach können Sie ohne weitere Einschränkungen mit der Anwendung arbeiten.

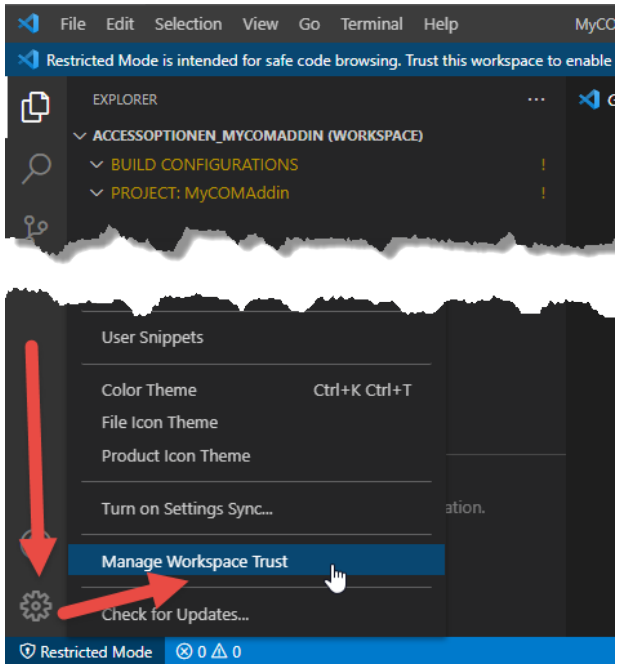


Abbildung 15.11: Aktivieren des Bereichs Workspace Trust

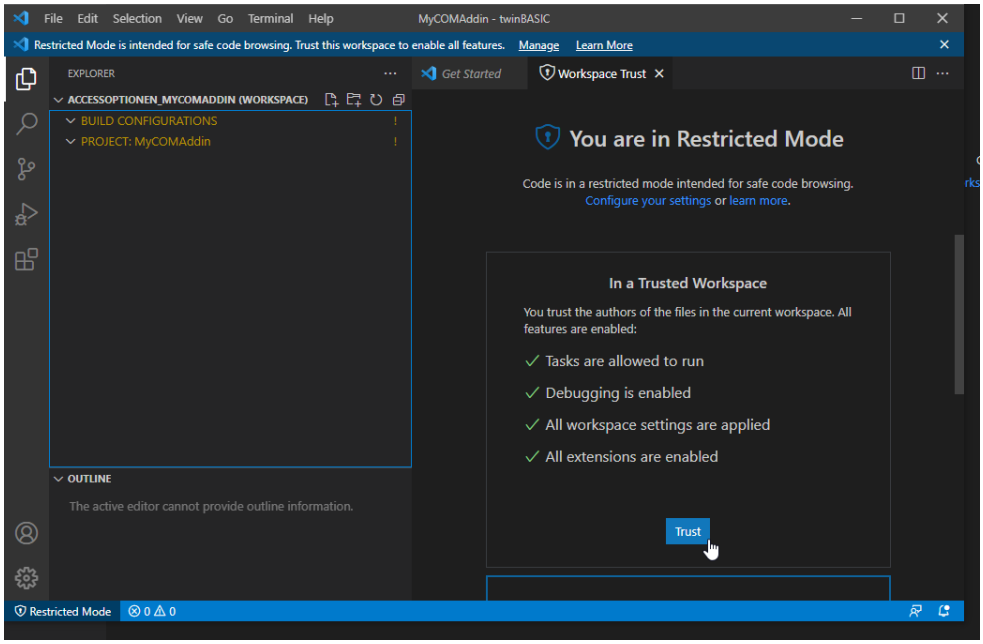


Abbildung 15.12: *Restricted Mode* durch Vertrauen der Lösung verlassen

15.4.2 Objektvariable auf Access.Application speichern

Da wir wissen, dass wir das COM-Add-In mit Access einsetzen wollen, können wir eine Objektvariable speziell zum Aufnehmen der aktuellen Access-Instanz deklarieren:

```
Private objAccess As Access.Application
```

Außerdem weisen wir diese in der Prozedur *OnConnection* wie folgt gezielt zu:

```
Sub OnConnection(ByVal Application As Object, _
        ByVal ConnectMode As ext_ConnectMode, _
        ByVal AddInInst As Object, _
        ByRef custom As Variant()) _
    Implements IDTExtensibility2.OnConnection
    Set objAccess = Application
End Sub
```

Hier tragen wir den mit dem Parameter *Application* übergebenen Verweis auf die aktuelle Access-Instanz in die Variable *objAccess* ein.

Dies wird dazu führen, dass die Angabe des Datentyps *Access.Application* als fehlerhaft markiert wird. Das liegt daran, dass die *Access*-Bibliothek in diesem Projekt noch nicht bekannt ist. Dazu wechseln wir durch einen Klick auf den Eintrag *Settings* am linken Rand zum Bereich *Settings*.

Hier scrollen Sie bis zum Bereich *COM Type Library / ActiveX References* und klicken in der Liste *ALL AVAILABLE COM REFERENCES* auf das Lupe-Symbol, um nach Einträgen mit dem Begriff *Access* zu suchen. Dies filtert die meisten Einträge heraus, sodass wir schnell den gesuchten Eintrag *Microsoft Access 16.0 Object Library* finden. Diesen versehen Sie mit einem Haken, sodass er in die obere Liste *ENABLED REFERENCES* aufgenommen wird (siehe Abbildung 15.13).

Wichtig: Sie müssen nun auf die *Speichern*-Schaltfläche im Bereich *TWINBASIC* am linken Rand klicken und das Speichern der Änderungen bestätigen, damit die Änderungen übernommen werden.

Wechseln Sie danach wieder zur Datei *HelloWorld.twin*, wird die Klasse *Access.Application* nicht mehr als fehlerhaft markiert.

An dieser Stelle können Sie auch direkt einen Verweis auf die Bibliothek *Microsoft Office 16.0 Access Database Engine Object Library* hinzufügen, den wir weiter unten für den Zugriff auf die Access-Optionen benötigen.

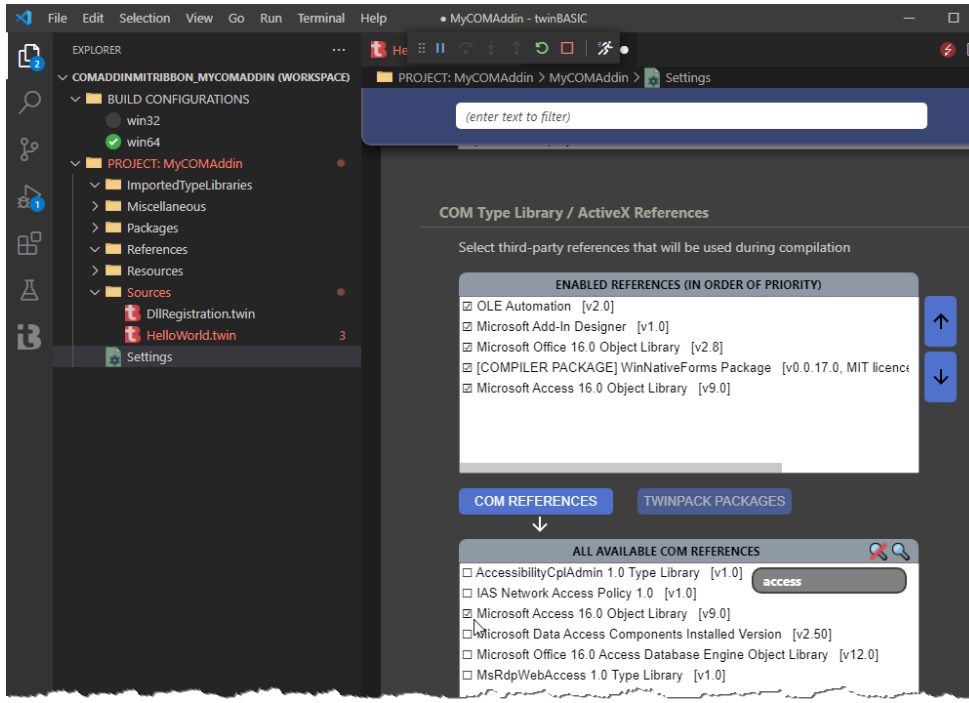


Abbildung 15.13: Hinzufügen der *Access*-Bibliothek

15.4.3 Variable für Ribbon deklarieren und füllen

Es ist absehbar, dass wir eine Variable des Typs *IRibbonUI* benötigen, da wir die Inhalte der Steuerelemente des nachfolgend erstellten Ribbons gelegentlich aktualisieren müssen – und dazu benötigen wir den Aufruf der Methoden *Invalidate* beziehungsweise *InvalidateControl*. Daher deklarieren wir zunächst eine Variable des Typs *IRibbonUI*:

```
Private objRibbon As IRibbonUI
```

Diese füllen wir in einer Callback-Funktion namens *OnLoad*, die wir in der nachfolgend zusammengestellten Ribbondefinition für das Attribut *onLoad* des *customUI*-Elements angeben:

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

15.4.4 Erste Option umsetzen: Anwendungstitel einstellen

Wir beginnen mit einer ersten Option, um uns den Ablauf anzusehen. Dies soll der Titel der Access-Anwendung sein. Der Vorteil gegenüber einigen anderen Optionen ist, dass wir Änderungen direkt in der Titelleiste angezeigt bekommen – im Gegensatz zu vielen anderen Optionen, die sich erst beim erneuten Öffnen der Datenbankanwendung zeigen.

Also fügen wir der Ribbon-Definition ein Textfeld hinzu. Danach sieht Prozedur *GetCustomUI* zur Definition des Ribbons wie folgt aus:

```
Private Function GetCustomUI(ByVal RibbonID As String) As String _
       Implements IRibbonExtensibility.GetCustomUI
    Dim strXML As String
    strXML &= "<xml version=""1.0"">" & vbCrLf
    strXML &= _
      "<customUI xmlns=""http://schemas.microsoft.com/office/2009/07/customui"" _
                                                          onLoad=""onLoad"">" & vbCrLf
    strXML &= "  <ribbon>" & vbCrLf
    strXML &= "    <tabs>" & vbCrLf
    strXML &= "      <tab id=""tabOptions"" label=""Optionen"">" & vbCrLf
    strXML &= "        <group id=""grpThisDatabase"" label=""<Aktuelle Datenbank>"">" _
                                                                        & vbCrLf
    strXML &= "          <editBox label=""Datenbankname:"" id=""txtTitle"" _
                          maxLength=""255"" sizeString=""xxxxxxxxxxxxxxxxxxxxxxxxx"" _
                                  getText=""getText"" onChange=""onChange""/>" & vbCrLf
    strXML &= "        </group>" & vbCrLf
    strXML &= "      </tab>" & vbCrLf
```

André Minhorst

Ribbon Programmierung

LESEPROBE

Hier finden Sie in der Vollversion noch weiteres Know-how rund um die Ribbon-Programmierung!

André Minhorst

Ribbon Programmierung

LESEPROBE

Hier finden Sie in der Vollversion noch weiteres Know-how rund um die Ribbon-Programmierung!

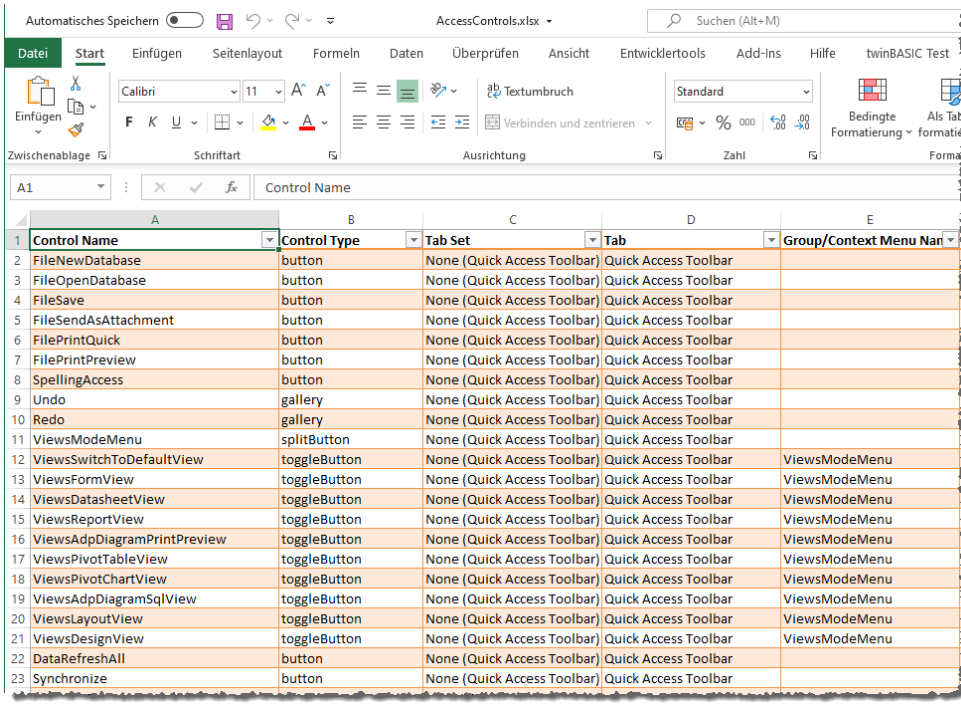
16 Referenz

In diesem Kapitel finden Sie die Referenz einiger wichtiger Elemente des Ribbons.

16.1 idMso-Werte

Die *idMso*-Werte sind die Bezeichnungen der eingebauten Ribbonelemente. Sie benötigen diese immer, wenn Sie eines der eingebauten Elemente referenzieren wollen – sei es, um ein neues, benutzerdefiniertes Element vor oder hinter einem der eingebauten Elemente zu platzieren, die Attribute eines der eingebauten Elemente zu ändern (beispielsweise um dieses auszublenden) oder um eines der eingebauten Elemente mit einem benutzerdefinierten *control*-Element nachzubilden.

Eine Auflistung dieser Werte finden Sie in einer öffentlich verfügbaren Exceltabelle namens *AccessControls.xlsx*. Diese enthält allerdings leider nur die *idMso*-Werte und einige andere Angaben (siehe Abbildung 16.1).



	A	B	C	D	E
1	Control Name	Control Type	Tab Set	Tab	Group/Context Menu Nar
2	FileNewDatabase	button	None (Quick Access Toolbar)	Quick Access Toolbar	
3	FileOpenDatabase	button	None (Quick Access Toolbar)	Quick Access Toolbar	
4	FileSave	button	None (Quick Access Toolbar)	Quick Access Toolbar	
5	FileSendAsAttachment	button	None (Quick Access Toolbar)	Quick Access Toolbar	
6	FilePrintQuick	button	None (Quick Access Toolbar)	Quick Access Toolbar	
7	FilePrintPreview	button	None (Quick Access Toolbar)	Quick Access Toolbar	
8	SpellingAccess	button	None (Quick Access Toolbar)	Quick Access Toolbar	
9	Undo	gallery	None (Quick Access Toolbar)	Quick Access Toolbar	
10	Redo	gallery	None (Quick Access Toolbar)	Quick Access Toolbar	
11	ViewsModeMenu	splitButton	None (Quick Access Toolbar)	Quick Access Toolbar	
12	ViewsSwitchToDefaultView	toggleButton	None (Quick Access Toolbar)	Quick Access Toolbar	ViewsModeMenu
13	ViewsFormView	toggleButton	None (Quick Access Toolbar)	Quick Access Toolbar	ViewsModeMenu
14	ViewsDatasheetView	toggleButton	None (Quick Access Toolbar)	Quick Access Toolbar	ViewsModeMenu
15	ViewsReportView	toggleButton	None (Quick Access Toolbar)	Quick Access Toolbar	ViewsModeMenu
16	ViewsAdpDiagramPrintPreview	toggleButton	None (Quick Access Toolbar)	Quick Access Toolbar	ViewsModeMenu
17	ViewsPivotTableView	toggleButton	None (Quick Access Toolbar)	Quick Access Toolbar	ViewsModeMenu
18	ViewsPivotChartView	toggleButton	None (Quick Access Toolbar)	Quick Access Toolbar	ViewsModeMenu
19	ViewsAdpDiagramSqlView	toggleButton	None (Quick Access Toolbar)	Quick Access Toolbar	ViewsModeMenu
20	ViewsLayoutView	toggleButton	None (Quick Access Toolbar)	Quick Access Toolbar	ViewsModeMenu
21	ViewsDesignView	toggleButton	None (Quick Access Toolbar)	Quick Access Toolbar	ViewsModeMenu
22	DataRefreshAll	button	None (Quick Access Toolbar)	Quick Access Toolbar	
23	Synchronize	button	None (Quick Access Toolbar)	Quick Access Toolbar	

Abbildung 16.1: Die Exceltabelle mit den *idMso*-Werten für die Access-Ribbons

Damit Sie schnell die *idMso* finden (schneller als über den Bereich Menüband anpassen der Access-Optionen), haben wir die Daten aus dieser Exceltabelle in eine Accesstabelle importiert und die deutschen Bezeichnungen der Steuerelemente zu den jeweiligen Zeilen hinzugefügt.

Dazu haben wir der aus der Exceldatei importierten Tabelle *tblAccessControls* ein neues Textfeld namens *Label* hinzugefügt. Dieses haben wir mithilfe der Funktion *GetLabelMso* der *CommandBars*-Auflistung mit der folgenden Prozedur gefüllt:

```
Public Sub WriteIdMsos()
    Dim db As DAO.Database
    Dim rst As DAO.Recordset
    Set db = CurrentDb
    Set rst = db.OpenRecordset("SELECT * FROM AccessControls", dbOpenDynaset)
    Do While Not rst.EOF
        rst.Edit
        On Error Resume Next
        rst!Label = CommandBars.GetLabelMso(rst![Control Name])
        On Error GoTo 0
```

```
            rst.Update
            rst.MoveNext
        Loop
    End Sub
```

Die Accesstabelle *tblControlNames* sieht anschließend auszugsweise wie in Abbildung 16.2 aus.

tblAccessControls

ID	Label	Control Name	Control Ty
1	Neu	FileNewDatabase	button
2	Öffnen	FileOpenDatabase	button
3	Speichern	FileSave	button
4	E-Mail	FileSendAsAttachment	button
5	Schnelldruck	FilePrintQuick	button
6	Seitenansicht	FilePrintPreview	button
7	Rechtschreibung	SpellingAccess	button
8	Rückgängig machen	Undo	gallery
9	Wiederholen	Redo	gallery
10	Modus	ViewsModeMenu	splitButton
11	Ansicht	ViewsSwitchToDefaultView	toggleButt
12	Formularansicht	ViewsFormView	toggleButt
13	Datenblattansicht	ViewsDatasheetView	toggleButt
14	Berichtsansicht	ViewsReportView	toggleButt
15	Seitenansicht	ViewsAdpDiagramPrintPreview	toggleButt
16		ViewsPivotTableView	toggleButt
17		ViewsPivotChartView	toggleButt
18	SQL-Ansicht	ViewsAdpDiagramSqlView	toggleButt
19	Layoutansicht	ViewsLayoutView	toggleButt
20	Entwurfsansicht	ViewsDesignView	toggleButt
21	Alle aktualisieren	DataRefreshAll	button
22	Alle synchronisieren	Synchronize	button

Abbildung 16.2: Tabelle der Bezeichnungen und *idMso*-Werte

Für diese Tabelle haben wir noch ein Formular samt Unterformular angelegt, mit dem Sie verschiedene Suchmöglichkeiten haben.

Mit dem Suchfeld geben Sie den zu suchenden Ausdruck an. Die Eingabe einzelner Zeichen aktualisiert die Ausgabe der gefundenen Einträge jeweils direkt. Mit der ersten Optionsgruppe können Sie festlegen, ob die Treffer mit dem Suchbegriff beginnen müssen, ob sie diesen enthalten müssen oder ob es eine exakte Übereinstimmung geben muss. Mit der zweiten Optionsgruppe geben Sie an, ob in den *idMso*-Bezeichnungen, in den Beschriftungen oder in beiden gesucht werden soll (siehe Abbildung 16.3).

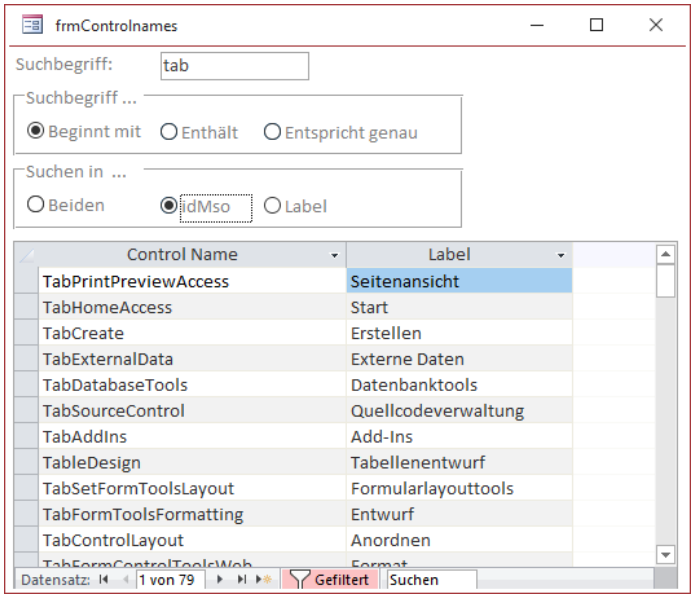


Abbildung 16.3: Formular zum Suchen nach *idMso*-Werten und Bezeichnungen

16.2 Callback-Signaturen für VBA und VB6

In diesem Abschnitt finden Sie die Callback-Signaturen, die Sie für den Einsatz in Access-VBA und für die Programmierung von DLLs mit VB6 oder TwinBasic nutzen können.

16.2.1 Warum verschiedene Callback-Signaturen?

Unter VBA und VB6/twinBASIC kommen unterschiedliche Callback-Signaturen zum Einsatz. Sie müssen genau darauf achten, welche Signaturen Sie verwenden, anderenfalls erhalten Sie Fehler wie den aus Abbildung 16.4.

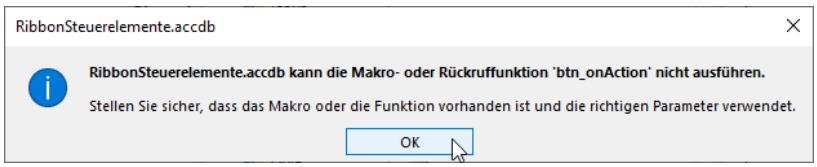


Abbildung 16.4: Fehler durch falsche Callback-Signatur

Der wichtigste Unterschied ist, dass Callback-Funktionen unter VB6/twinBASIC nämlich tatsächlich Funktionen sind – und nicht wie unter VBA Callback-Prozeduren, die einen weiteren Parameter enthalten, der als Rückgabewert verwendet wird.

Schauen wir uns also die unterschiedlichen Signaturen für die einzelnen Steuerelemente des Ribbons für VBA und VB6 an!

Vorab die wichtigste Information: Verwenden Sie alle Callback-Signaturen genau so wie Sie hier abgebildet sind. Manchmal kann bereits das Hinzufügen eines Datentyps zu einem Parameter, der hier keinen Datentyp aufweist, zu Problemen führen. Wichtig ist auch, auf die korrekte Verwendung von Function/Sub zu achten. Wenn Sie twinBASIC verwenden, können Sie das Ergebnis zum Zurückgeben entweder einer Variablen mit dem gleichen Namen wie die Funktion zuweisen, aber auch die Return-Anweisung nutzen:

```
Function GetText(control As IRibbonControl, ByRef text)  As String
    ...
    Return strText
End Function
```

Bei den folgenden Definitionen stellen wir zunächst die je nach Steuerelement individuellen Callback-Funktionen vor. Am Ende finden Sie eine Auflistung all jener Callback-Funktionen, die von einer großen Anzahl an Steuerelementen verwendet wird.

16.2.2 Die Callback-Funktionen des button-Elements

Das *button*-Element bietet die folgenden Callback-Funktionen:

» *getShowImage*: Fragt ab, ob ein Image angezeigt werden soll. Das geschieht unter VBA mit dem Rückgabe-Parameter *showImage*, unter VB6/twinBASIC mit dem Rückgabewert.

```
VBA: Sub GetShowImage (control As IRibbonControl, ByRef showImage)
VB6/twinBASIC: Function GetShowImage (control As IRibbonControl) As Boolean
```

» *getShowLabel*: Fragt ab, ob eine Bezeichnung angezeigt werden soll. Der Wert *True* oder *False* wird mit dem Parameter *showLabel* oder dem Rückgabewert übergeben.

```
VBA: Sub GetShowLabel (control As IRibbonControl, ByRef showLabel)
VB6/twinBASIC: Function GetShowLabel (control As IRibbonControl) As Boolean
```

» *onAction*: Wird beim Anklicken ausgelöst. Diese Signatur gilt für benutzerdefinierte *button*-Elemente.

```
VBA: Sub OnAction(control As IRibbonControl)
VB6/twinBASIC: Sub OnAction(control As IRibbonControl)
```

» *onAction*, Alternative: Wird beim Anklicken eines eingebauten Elements ausgelöst, für das Sie ein *command*-Element definiert haben. Wie Sie diese Variante von *onAction* einsetzen, erfahren Sie unter »Funktion von Ribbonelementen überschreiben« ab Seite 175.

```
VBA: Sub OnAction(control As IRibbonControl, ByRef CancelDefault)
VB6/twinBASIC: Sub OnAction(control As IRibbonControl, ByRef CancelDefault)
```

16.2.3 Die Callback-Funktionen des checkBox-Elements

Das *checkBox*-Element bietet die folgenden Callback-Funktionen:

» *getPressed*: Stellt den Zustand des *checkBox*-Steuerelements ein, also auf *True* oder *False*. Der Rückgabewert heißt unter VBA *returnValue*.

```
VBA: Sub GetPressed(control As IRibbonControl, ByRef returnValue)
VB6/twinBASIC: Function GetPressed(control As IRibbonControl) As Boolean
```

» *onAction*: Wird ausgelöst, wenn der Benutzer auf ein *checkBox*-Steuerelement klickt und seinen Wert damit ändert. Liefert den neuen Wert mit dem Boolean-Parameter *pressed*.

```
VBA: Sub OnAction(control As IRibbonControl, pressed As Boolean)
VB6/twinBASIC: Sub OnAction(control As IRibbonControl, pressed As Boolean)
```

16.2.4 Die Callback-Funktionen des comboBox-Elements

Das *comboBox*-Element bietet die folgenden Callback-Funktionen:

» *getItemCount*: Callback-Funktion zum Abfragen der Anzahl der im *comboBox*-Steuerelement anzuzeigenden Einträge. Zu liefern mit dem Parameter *count* oder dem Rückgabewert der Callback-Funktion.

```
VBA: Sub GetItemCount(control As IRibbonControl, ByRef count)
VB6/twinBASIC: Function GetItemCount(control As IRibbonControl) As Integer
```

» *getItemID*: Ruft den Wert für die *ItemID* eines der Einträge des *comboBox*-Steuerelements ab, dessen Index unter VBA mit dem Parameter *index* abgefragt wird – oder mit dem Rückgabewert unter Visual Basic 6/twinBASIC.

```
VBA: Sub GetItemID(control As IRibbonControl, index As Integer, ByRef id)
VB6/twinBASIC: Function GetItemID(control As IRibbonControl, index As Integer) As
String
```

» *getItemImage*: Ruft den Namen des Bildes eines der Einträge des *comboBox*-Steuerelements ab, dessen Index unter VBA mit dem Parameter *index* oder unter VB6/twinBASIC mit dem Rückgabewert der Callback-Funktion abgefragt wird.