Ribbon-TV

Einen schönen guten Abend, liebe Access-Freund:innen! Wir beginnen unseren Themenabend „Ribbons und Datenbanken“ mit einer investigativen Dokumentation über die Vor- und Nachteile. In der TerraXpress-Reihe zum heutigen Thema „Schnell geklickt, lang gereut?“ folgen wir dem erschütternden Schicksal eines lieblos zusammengeklickten Ribbons.

Danach gibt es die beliebte Quizshow für die ganze Familie und schließlich folgen zwei erfolgreiche Wissenssendungen für Kinder und Erwachsene. Wir runden den Abend ab mit der spannenden Liveshow direkt aus dem Access-Dschungel. Und als Nachtlektüre empfiehlt unser Literarisches Duett noch ein Buch (für die jüngeren unter uns: Das ist Retro-Hardware aus Papier und zum Umblättern statt Wischen …).

Heute im Programm:

Wir wünschen Euch gute Unterhaltung bei unserem Themenabend rund um die Ribbons in Access!

19:30 Gute Ribbons – schlechte Ribbons

Diese beliebte Vorabendserie zeichnet das Schicksal einer der bekanntesten Bedienungs-Oberflächen nach. Menüs und Symbolleisten waren über Jahrzehnte hinweg so selbstverständlich, dass ihr Ende undenkbar war.

2007 kam Microsoft plötzlich mit einem ganz neuen Design, dessen wesentlichstes Kennzeichen ein Ribbon („Menüband“[1]) war. Die ursprüngliche Idee war, dass alle Buttons in einem waagerechten unendlichen Band angeordnet sind (das funktioniert übrigens heute noch, denn mit dem Mausrollrad innerhalb des Ribbons bewegen sich die Buttons hin und her).

Weil das aber zu umständlich war, wurden die Buttons bereits in der ersten veröffentlichten Version in Tabs (Registern) zusammengefasst. Dadurch ist das ursprünglich durchgängig geplante Band gruppiert und einzelne Buttons sind schneller zu erreichen.

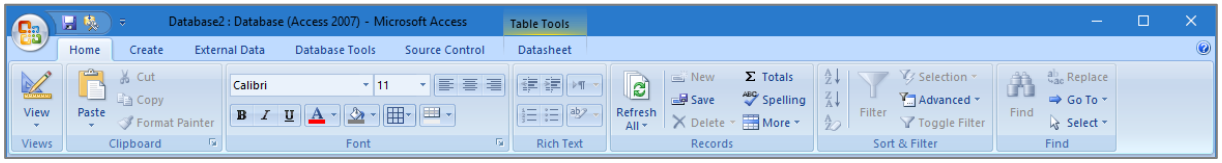

Abbildung 1: Start-Register in Access 2007 (englische Version, daher "Home")

Die verspielte Optik der Anfangszeit hat Microsoft später flacher, mit größeren Elementen und mit mehr Abständen gestaltet, um auch eine Bedienung auf kleinen Bildschirmen und mit Touchscreens zu ermöglichen.

Abbildung 2: Start-Register in Access 2016

Außerdem wurden die Tabs zunehmend unsichtbarer. Inzwischen ist es statt einer wirklichen Registerlasche nur noch eine schmale Unterstreichung des Registernamens.

Abbildung 3: Start-Tab in Access 2019

Das ist aber reine Optik. Die Struktur des Ribbons ist seit 2007 fast unverändert geblieben. Deswegen ist es ziemlich egal, mit welcher Access-Version ich es hier zeige.

Das erstaunlichste bei diesem scheinbar gravierenden Wechsel von Menüs/Symbolleisten zu Ribbons ist: Eigentlich sind Ribbons nur eine Mischung aus Menüs und Symbolleisten und es hat sich viel weniger verändert, als die meisten glauben. Mit <Strg>+<F1> lässt sich das Ribbon beispielsweise minimieren und verhält sich dann genau wie eine Menüleiste.

[1] Ich werde weitestgehend die englischen Bezeichnungen benutzen, weil diese auch in XML und der VBA-Programmierung stehen.

19:55 Terra Xpress

Eigene Ribbons sind angeblich kompliziert zu erstellen? Nein, überhaupt nicht! Es braucht nur wenige Klicks, um sich so ein eigenes Ribbon zusammenzustellen.

Mit einem Rechtsklick auf das vorhandene Ribbon wählst Du im PopUp-Menü den Eintrag „Menüband anpassen“. Daraufhin erscheint der Dialog, um die Elemente des Ribbons nach eigenen Wünschen zu verändern.

Hier lassen sich sowohl vorhandene Elemente (einzelne Buttons ebenso wie ganze Groups oder Tabs) einfach ein- und ausklicken als auch beliebige neue Elemente hinzufügen.

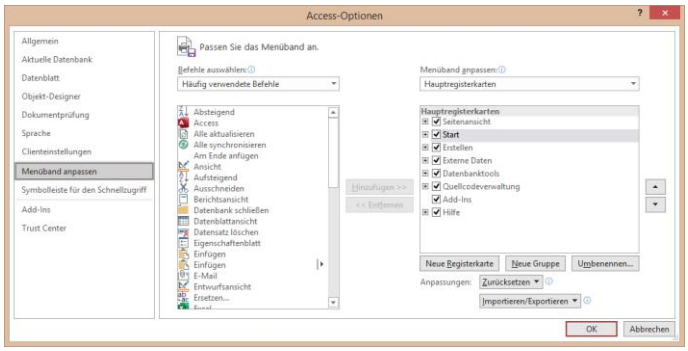

Abbildung 4: Ursprüngliche Ansicht des MenübandAnpassen-Dialogs

Um ein neues Tab hinzuzufügen, klickst Du auf [Neue Registerkarte]. Im rechten Treeview erscheint daraufhin nicht nur „Neue Registerkarte“, sondern darunter auch sofort eine „Neue Gruppe“. Zum Umbenennen machst Du einen Rechtsklick auf den Namen der Registerkarte und gibst anschließend den gewünschten Namen ein:

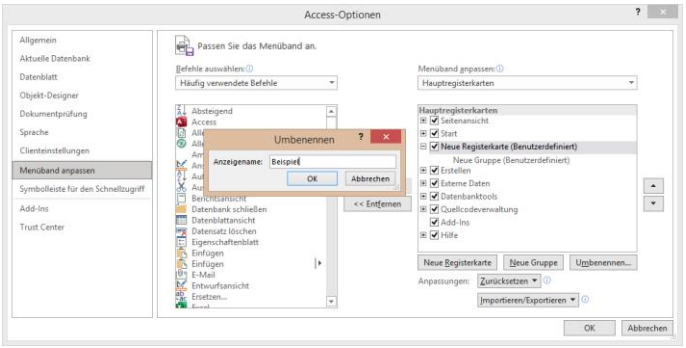

Abbildung 5: Die eingefügte neue Registerkarte wird direkt umbenannt

Auch die Umbennung der Gruppe kannst Du per Rechtsklick vornehmen. Hier fragt der Dialog nicht nur nach dem neuen Namen, sondern bietet auch die Auswahl eines Icons an. Dieses Icon ist erst dann zu sehen, wenn das Access-Fenster so schmal ist, dass Groups zusammengefaltet und auf ihr Icon reduziert werden.

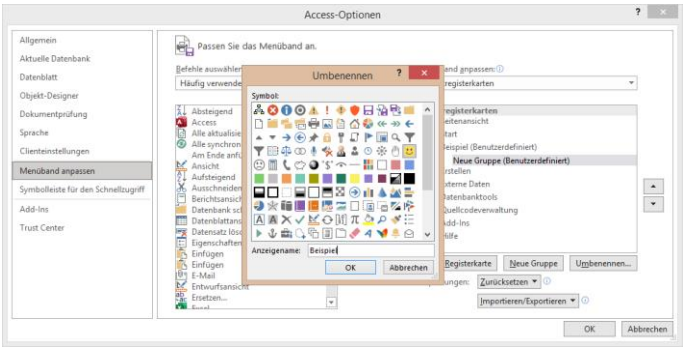

Abbildung 6: Für die Gruppe kann ein Icon ausgewählt werden

Jetzt ist zwar schon die äußere Struktur von neuem Tab und neuer Group vorbereitet, aber noch kein einzelner Button enthalten. Diese Buttons sind alle in der linken Liste angeboten, allerdings steht anfangs darüber noch bei „Befehle auswählen“ die Standardoption „Häufig verwendete Befehle“. Hier würde ich direkt auf „Alle Befehle“ wechseln.

Die Liste ist nun zwar sehr lang und leider nicht filterbar, aber dafür sind alle Möglichkeiten da. Beispielhaft werde ich die Befehle „Kopieren“, „Einfügen“ (der erste von den dreien), „Aufsteigend“ und „Absteigend“ jeweils markieren und mit dem [Hinzufügen]-Button in die „Beispiel“-Gruppe einfügen.

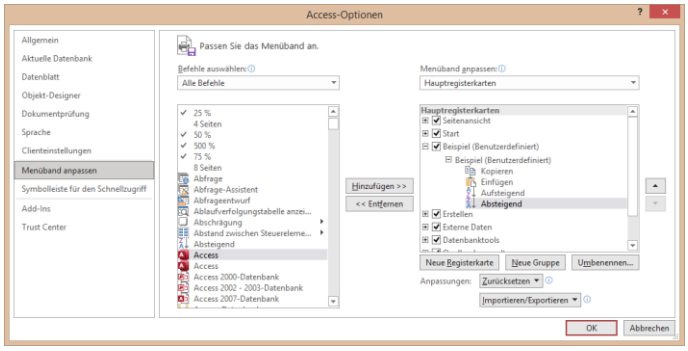

Abbildung 7: Die vier Buttons sind eingefügt

Danach beschließe ich, dass mein neues Ribbon fertig ist, und bestätige mit [OK]. Jetzt sind alle Änderungen direkt zu sehen.

Abbildung 8: Das angepasste Ribbon mit dem eigenen Tab ist sofort sichtbar

Wer sich jetzt fragt, ob die offensichtlich inaktiven Buttons noch irgendwie freigeschaltet werden müssen: Nein, sie sind sofort funktionsfähig. Aber alle Ribbon-Buttons erkennen selbstständig, in welchem Zustand sie sich befinden müssen. Sobald also sortierbare oder kopierbare Daten vorhanden sind, ändern sich die Buttons automatisch.

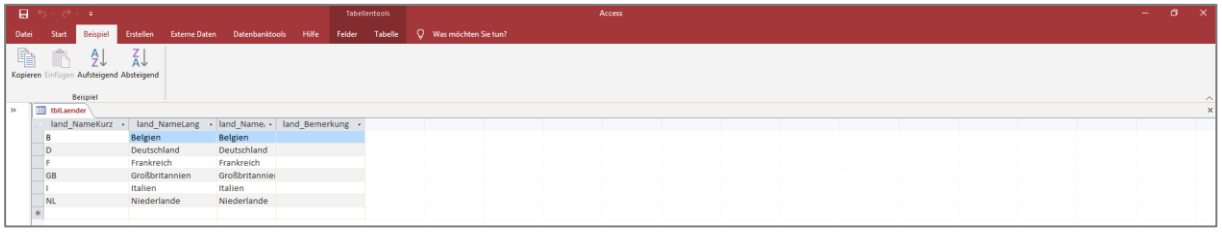

Abbildung 9: Mit Daten sind die passenden Buttons auch aktiv

Mit diesem MenübandAnpassen-Dialog ist es also wirklich leicht, sich das Ribbon anzupassen. Es ist sofort sichtbar und sofort funktionsfähig.

Ja, aber …

… es gibt eine Reihe von Einschränkungen bei dieser Methode.

Du kannst (fast) keine wirklich neuen Buttons einfügen, sondern nur aus der Liste aller vorhandenen auswählen. Diese Liste ist sehr umfangreich, daher ist das nur eine kleine Einschränkung. Vor allem gewährleisten nur integrierte Buttons, dass sie selbstständig organisieren, wann sie aktiv werden dürfen und was sie tun sollen.

Tatsächlich lassen sich doch ein paar wirklich neue Buttons einfügen, nämlich eigene Makros. Diese erscheinen mit der Option „Makros“ in „Befehle auswählen“ und können auch mit passenden Icons bestückt werden (nächste Einschränkung übrigens: es sind keine eigenen Icons möglich).

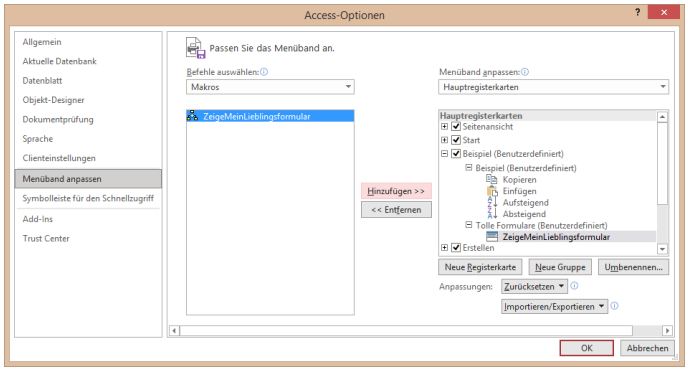

Abbildung 10: Mein Makro kann ebenfalls übernommen werden

Dieses Makro habe ich direkt in einer eigenen Group organisiert, daher sieht das Ergebnis jetzt so aus und führt bei Klick das Makro auch direkt aus.

Abbildung 11: Der Makro-Button steht in einer eigenen Gruppe mit eigenem Icon

Wenn dieses Ribbon so gelungen ist, dass auch meine Kolleg:innen das haben wollen, kann ich das im MenübandAnpassen-Dialog mit dem [Importieren/Exportieren]-Button in eine *.exportedUI-Datei schreiben und auf einem anderen PC einfach mit dem gleichen Button wieder importieren.

Das Hauptproblem mit dieser Art der Ribbon-Anpassung ist, dass diese Änderungen **in allen Access-Datenbanken auf diesem PC** zu sehen sind. Das ist also keineswegs auf die Datenbank beschränkt, in welcher ich die Änderungen vorgenommen habe. Für allgemeine Aktionen wie „Kopieren“ oder „Aufsteigend (sortieren)“ ist das unproblematisch, aber ein konkreter Makro-Aufruf wie „ZeigeMeinLieblingsformular“ scheitert woanders krachend mit einer Fehlermeldung, weil ja das Makro fehlt.

Das heutige Hauptthema werden also die „richtigen“ Ribbons sein, welche mit XML, Callback-Buttons und VBA-Programmierung erstellt werden müssen.

20:15 Der große Preis

In der großen Familien-Abend-Show beschäftigen wir uns mit den Gründen für und gegen das Ribbon. Einige davon sind berechtigt, manche gar unbekannt und viele davon vielleicht nur Vorurteile. Auf der großen Ratewand können hier nun alle mitmachen und ihre Lieblingsvorurteile bestätigen oder sogar ganz neue Gegenargumente finden.

Daher braucht es wenigstens einen Kurzüberblick darüber, wie Ribbons überhaupt funktionieren. Wum und Wendelin hatten leider keine Zeit, also übernehme ich das selbst.

Problem: XML

XML (eXtensible Markup Language) ist die Sprache, in welcher die Ribbons beschrieben werden. Es ist keine Programmiersprache, sondern eine sogenannte Auszeichnungssprache wie zum Beispiel auch HTML. XML beschreibt das Konzept des Ribbons. Das ist eine Mischung aus beschreibenden Inhalten (`label="Mein Titel"`) und Callback-Prozedurnamen (`getLabel="GetLabel"`).

Keine grafische Oberfläche

Nur das Zusammenklicken der oben beschriebenen einfachen Ribbons ist leicht möglich, aber die „richtigen“ Ribbons müssen komplett außerhalb von Access erstellt werden. Es gibt in Access nicht den Hauch einer Unterstützung für die Erstellung der benötigten XML-Anweisungen, weder einen Assistenten-Dialog noch überhaupt irgendeine helfende Oberfläche. Nirgends.

Nicht mal ein Editor mit Syntaxcheck

Angesichts der durchaus vielen syntaktischen Fehlermöglichkeiten wäre wenigstens ein einfacher Texteditor zu erwarten. Aber schon der Editor für den SQL-Abfrageentwurf ist ja so armselig, dass es vielleicht zu viel verlangt ist, einen banalen XML-Editor in Access anzubieten.

Groß-/Kleinschreibung muss exakt sein

XML ist besonders pingelig. Während bei HTML ja die Groß-/Kleinschreibung egal ist, muss diese bei XML unbedingt korrekt sein. Das ist keine Besonderheit des Ribbon-XML-Codes, aber eine sehr beliebte zusätzliche Fehlerquelle.

Lösung: XML

Da Access das Erstellen des Ribbon-XML-Codes seit Jahren nicht hinbekommt, gibt es inzwischen externe Programme, die das leisten. Hier möchte ich insbesondere die beiden deutschen Programme nennen, welche jeweils sehr hilfreiche Oberflächen zur Erstellung (und anschließenden Übernahme in die Datenbank!) anbieten:

Der „RibbonCreator“ (https://www.ribboncreator2019.de) von Gunter Avenius und der „Ribbon-Admin“ (https://shop.minhorst.com/access-tools/309/ribbon-admin-2016) von André Minhorst. Beide Programme existieren in verschiedenen Varianten für die jeweiligen Access-Versionen.

Problem: Transparenz

Nur integrierte Icons transparent

Es ist richtig, dass Access kein Grafik-Programm ist, sondern für den Umgang mit Daten entworfen wurde. Aber dass über 30 Jahre nach Einführung von teiltransparenten GIFs diese zwar beim Import in Formularen/Berichten funktionieren, jedoch nicht in den modernen Ribbons, ist schon peinlich.

Lösung: Transparenz

Sascha Trowitzsch hat ein VBA-Modul („basGDIPlus“ für 32-Bit-Access bzw. „basGDIPlus64“ für 64-Bit-Access) entwickelt, welches diese Transparenz ermöglicht. Dabei muss nur an einer einzigen Stelle der originale `LoadPicture`-Befehl durch den `LoadPictureGDI`-Befehl ersetzt werden. Aus Copyright-Gründen kann ich es hier nicht einfach so zur Verfügung stellen.

Problem: Aktualisierung

Sichtbare Buttons sofort aktualisieren

Da alle Buttons potentiell immer sichtbar sind, müssen sie jederzeit ihren Status korrekt halten. Das betrifft insbesondere aktiv/inaktiv, aber je nach Gestaltung der Ribbons auch sichtbar/unsichtbar. Während die integrierten Buttons das selbstständig übernehmen, muss ich das für die selber programmierten Buttons auch selber organisieren.

Es gibt zwei Methoden, um diese Aktualisierung auszulösen: Entweder `Ribbon.Invalidate` (dabei wird das gesamte Ribbon aktualisiert, das ist bequem, aber bei vielen Buttons eventuell langsam) oder `Ribbon.InvalidateControl` (dabei wird gezielt nur ein einzelner Button aktualisiert). In jedem Fall muss mein Code jedoch immer berücksichtigen, dass jetzt irgendwas aktualisiert werden muss.

Manchmal funktioniert es nicht

Vor allem bei der Anzeige von `contextualTabs` (also der Ribbon-Tabs, die nur in bestimmten Formularen oder Berichten angezeigt werden) habe ich die Erfahrung gemacht, dass die Aktualisierung trotz der `Invalidate`-Befehle nicht immer zuverlässig funktioniert. Nach dem Neuladen der Datenbank klappt es wieder, es scheint also eine Überlastung der internen Ribbon-Verwaltung zu sein.

Lösung: Aktualisierung

Da das wesentliche Problem darin liegt, dass die Verwaltung der sichtbaren Elemente schwierig ist, besteht die Lösung darin, diese unsichtbar zu machen. Der einfachste Weg ist ein `dynamicMenu`-Steuerelement. Anders als das normale Menü lädt es seine Menüelemente im Moment des Auslösens nach und muss erst zu diesem Zeitpunkt entscheiden, welche davon aktiv oder sichtbar sind.

Problem: Anordnung

Rechtsbündige Buttons unmöglich

Ribbon-Inhalte werden automatisch nebeneinander angeordnet, aber immer von links. Wenn ich also beispielsweise einen „Info“- oder „Fenster schließen“-Button am rechten Bildschirmrand zeigen möchte, ist das nicht möglich.

Gemischte Tabs in falscher Reihenfolge

Wenn ein Formular automatisch beim Öffnen der Datenbank angezeigt wird, sind die allgemeinen Ribbons offenbar noch nicht fertig erstellt. Das führt dazu, dass für dieses Formular hinzugefügte Ribbon-Groups, die korrekterweise sonst immer rechts dahinter erscheinen, plötzlich links davor stehen.

Keine Lösung: Anordnung

Für die gewünschte Anordnung von Ribbon-Buttons am rechten Bildschirmrand gibt es keine Lösung.

Damit die Ribbon-Elemente bei allen Formularen in der erwartbaren Reihenfolge stehen, dürfen diese nicht im AutoStart aufgerufen werden. Weder mit dem „AutoExec“-Makro noch über die Access-Option „Formular anzeigen“ funktioniert es.

Problem: Ribbon-Variable

Bei Laufzeitfehlern/Stop Variable leer

Um auf das Ribbon per VBA (beispielsweise für den `Invalidate`-Befehl) zugreifen zu können, muss eine `public`-Variable darauf verweisen. Nach jedem Laufzeitfehler und Stop der VBA-Ausführung ist diese Variable leer.

Nicht aktiv wieder ermittelbar

Besonders ärgerlich aber ist, dass diese Variable ausschließlich beim Öffnen der Datenbank mit der dann automatisch aufgerufenen `OnRibbonLoad`-Prozedur ermittelt werden kann. Es gibt keine Möglichkeit, aktiv deren Wert zu ermitteln.

Lösung: Ribbon-Variable

Die naheliegendste Lösung ist, nach jedem Verlust des Variablen-Inhalts die Datenbank neu zu starten. Das lässt sich mit dem Komprimieren verbinden und ist oft ein guter Anlass, schnell noch mal eine Kopie der Datenbank zu sichern. Aber Spaß macht es nicht. Vor allem greift so ein FrontEnd ja vielleicht per Netzwerk auf einen SQL-Server zu oder ist einfach sehr umfangreich und mit Passwörtern gesichert, was bei jedem Neuaufruf viel Wartezeit erzeugt.

Als bessere Lösung lässt sich der Verweis auf das Ribbon nicht nur in einer VBA-Variablen, sondern außerdem noch in einer Dokument-Variablen (`DocVar`-Auflistung) speichern. Diese bleiben unabhängig von VBA-Laufzeitfehlern immer erhalten, so dass ein VBA-Code den gewünschten Wert aus einer Dokument- wieder in die VBA-Variable zurückschreiben kann. Das klappt einigermaßen, wenn auch nicht besonders zuverlässig, ist aber einen Versuch wert.

Problem: Verknüpfung

„USysRibbons“ nicht verknüpfen

Der XML-Code für die Ribbons steht in der (meistens ausgeblendeten) Tabelle „USysRibbons“. Beim Entwickeln einer Datenbank stehen aus Bequemlichkeit alle Tabellen noch unverknüpft in der gleichen Datei.

Dann werden, z.B. mit dem Access-Assistenten, alle Tabellen ins BackEnd verschoben und mit dem FrontEnd verknüpft. Da ist automatisch auch „USysRibbons“ dabei und das scheint sinnvoll zu sein. Aber beim nächsten Start des FrontEnds kommt dann die unschöne Überraschung: Access steht dann komplett! Keine Fehlermeldung, kein Abbruch möglich, keine Reaktion.

Lösung: Verknüpfung

Da das Problem lediglich darin besteht, dass offenbar die Verknüpfung zur Tabelle „USysRibbons“ nicht rechtzeitig genug offen ist und das Ribbon mit diesem Fehler nicht umgehen kann, muss „USysRibbons“ einfach unverknüpft im FrontEnd bleiben.

Problem: Gewohnheit

Ribbons sind einfach doof

Wer seit Jahren umfangreiche und bestens durchdachte Lösungen für die perfekte Bedienungs-Oberfläche in Access erstellt hat, möchte das natürlich nicht einfach so wegschmeißen. Da kann ich eine ablehnende Haltung durchaus verstehen. Als die erste deutsche Eisenbahn von Nürnberg(!) nach Fürth gebaut wurde, gab es auch nicht gleich überall Begeisterung.

Früher war alles besser

Früher war vielleicht mehr Lametta, aber es war auf jeden Fall nicht alles besser. Es war bestenfalls anders. Was die Ribbons betrifft, können die sowieso schon mal alles, was Menüs und Symbolleisten früher auch konnten. Und zusätzlich können sie erheblich mehr.

Lösung: Gewohnheit

Gegen Gewohnheiten ist schwer anzukommen. Aber wenn es doch eine Chance gegen alte Gewohnheiten gibt, dann vielleicht mit diesem Überblick über die tollen Möglichkeiten der Ribbons. Außerdem liefere ich die Beispieldatenbanken mit. Anstatt also mühsam eigenen XML-Code schreiben zu müssen, könnt Ihr diesen als fertige und vor allem funktionierende Grundlage benutzen und erweitern.

Lösung: Integriert

Nutzung bereits vorhandener Buttons

Viele Datenbankoberflächen nutzen Buttons auf Formularen, weil die so schön einfach zu erstellen sind. Der Assistent bietet sogar funktionsfähige Buttons zum Einfügen an, welche integrierte Makros enthalten. Das ist nett, aber eine doch arg begrenzte Auswahl (mal abgesehen von den eher altmodischen Icons darin, die deutlich an die 1990er Jahre erinnern).

Ribbons bieten schon in der zusammenklickenden MenübandAnpassen-Version schlicht *alle* existierenden Access-Buttons. Dabei sind nicht nur die jeweiligen Icons zugehörig korrekt, sondern passen sich optisch selbstständig an die aktuelle Access-Version an. Die gleiche alte Access2016-Datenbank präsentiert sich also automatisch im Access2019-Look, wenn sie in der neuen Access-Version geöffnet wird.

Auch bei der richtigen Ribbon-Erstellung anhand eigener XML-Anweisungen stehen natürlich alle integrierten Buttons und Icons zur Verfügung. Hier lassen sich sogar integrierte Icons für eigene Buttons nutzen oder integrierte Buttons mit eigenen Icons versehen.

Die gleichen Buttons (z.B. integrierte Buttons wie „Schließen“ oder „Nächster/Voriger Datensatz“, aber genauso eigene Befehle) lassen sich parallel für mehrere Formulare nutzen, ohne sie zu kopieren. Mit Schaltflächen auf dem Formularentwurf geht das nicht, sie müssen in jedes neue Formular immer wieder kopiert werden[2].

Buttons mit idMso verwalten sich selber

Alle integrierten Buttons (also solche, die in XML mit `idMso` statt nur `id` beschrieben werden), organisieren ihren Status selbstständig. Es reicht, sie an der gewünschten Stelle im eigenen Ribbon einmalig per XML-Code einzufügen. Keine einzige VBA-Zeile ist notwendig, damit sie funktionieren oder aktiv/inaktiv bzw. sichtbar/unsichtbar sind.

Lösung: Notwendig

Commands nur so deaktivierbar

Die Ribbons bestehen nicht nur aus der sichtbaren Oberfläche mit Tabs und Icons, sondern es zählt auch die Deaktivierungen integrierter Buttons dazu. Wer also beispielsweise verhindern will, dass der FettSchrift-Button ausgeführt werden darf, braucht die `<commands>`-Struktur in XML, mit der integrierte Buttons auf eigenen VBA-Code umgeleitet oder ganz deaktiviert werden können.

Problem: Notwendig

Leider funktioniert das Deaktivieren integrierter Buttons nicht komplett. Beispielsweise lassen sich die FettSchrift-Buttons überall im Ribbon deaktivieren, was dann sogar im „Mini-Ribbon“ (beim Markieren von Textteilen) übernommen wird, soweit ist es perfekt. In der Entwurfsansicht eines Formulars kann allerdings immer noch die Eigenschaft „Schriftbreite“ auf „Fett“ gestellt werden. Das halte ich jedoch für ein unwesentliches Restproblem.

Auch das Tastenkürzel <Strg>+<B> funktioniert weiterhin, aber das ist kein Fehler, sondern muss mit dem VBA-Befehl `Application.OnKey` definitiv anders gelöst werden.

[2] Ja, es gibt auch für Buttons auf Formularen eine Lösung, bei der sich diese Buttons auf einem eingebetteten Unterformular befinden. Das mache ich selber für einheitliche Formularköpfe mit solchen Bedienungselementen. Das kann aber durchaus aufwändig werden, damit es vernünftig funktioniert.

Lösung: Vielfältig

Viel mehr Control-Typen als im Menü

In den damaligen Menüs konnten nur Menüeinträge mit relativ wenigen Typen eingefügt werden. Es gab einen „normalen“ Menüeintrag (=Button), ein Untermenü und ein sehr selten genutztes Eingabefeld. Die Symbolleisten kannten außerdem noch ToggleButton, List- und ComboBoxen.

Die Ribbons bieten nicht nur alle diese Buttonstypen, sondern viele weitere neue wie splitButton, gallery, imageList und andere. Und selbst die bisherigen Buttontypen sind deutlich umfangreicher. Beispielsweise kann ein Menüeintrag klein und groß angezeigt werden und zusätzliche Erläuterungstexte bieten.

Lösung: Organisiert

Buttons werden automatisch angeordnet

Anders als manuell ausgerichtete Buttons auf Formularen werden die Buttons im Ribbon automatisch angeordnet. Das ist dann wichtig, wenn einzelne Buttons oder ganze Groups unsichtbar gemacht werden. Auf einem Formular bleiben dann Lücken, im Ribbon rutschen die nachfolgenden Elemente einfach auf.

Außerdem muss ich mir keine Gedanken um Raster und einheitliche Ausrichtungen machen. Das Ribbon selber sorgt für die korrekte Ausrichtung und faltet Groups sogar selbstständig zusammen, wenn das Programmfenster zu klein wird.

Größe umschaltbar

Während Buttons auf Formularen eine feste Größe haben und sich bei manueller Größenänderung das Icon nur im Zoomen-Modus (dann allerdings oft in unscharfer Qualität) ändert, lassen sich die Ribbon-Buttons standardmäßig in genau zwei Größen darstellen. Mit `size="normal"` ist die Darstellung so klein, dass drei Buttons übereinander passen. Mit `size="large"` gibt es die große Darstellung in voller Ribbon-Höhe.

Lösung: Platzsparend

Konzentriert am oberen Rand

Alle Ribbon-Buttons liegen immer gemeinsam im Ribbon. Das kann ein Vorteil sein, weil dann die Benutzer:innen immer wissen, wo Befehle zu suchen sind. Für Buttons, die sich auf ein konkretes Bedienungselement (z.B. eine alternative Auswahl neben einer ComboBox) beziehen, ist es manchmal aber doch benutzungsfreundlicher, einen einzelnen Button direkt daneben zu haben.

Jederzeit minimierbar

Da das Ribbon sich mit <Strg>+<F1> jederzeit minimieren lässt und das Formular dadurch mehr Fläche bekommt, habt Ihr letzten Endes mehr Platz für die Daten. Das ist vor allem auf kleinen Bildschirmen wie z.B. Laptops ein spürbarer Vorteil.

Das vorübergehende Ausblenden von Buttons auf dem Formular selber bedeutet demgegenüber einen erheblichen Programmieraufwand.

Lösung: Gruppiert

Tabs/Gruppen/Trennstriche

Die Ribbon-Buttons sind automatisch in Tabs und Groups hierarchisch zusammengefasst. Zusätzlich lassen sich noch Separator (Trennstriche) innerhalb einer Group einfügen. Dadurch wird das Ribbon sehr übersichtlich strukturiert.

Keine Umrahmungskästchen nötig

Die auf Formularen sehr beliebte Umrahmung von zusammenhängenden Buttons (mit Rechtecken oder dem Frame-Element) ist vor allem nötig, um den Überblick zu verbessern. In Ribbons ist das durch die offensichtliche Strukturierung überflüssig.

21:45 Ribbon macht Ah!

Jetzt wird es aber langsam mal Zeit für Butter bei die Fische. Wie erzeuge ich denn nun so ein Ribbon?

Zuallererst braucht es einen Text in XML mit vorgegebenen Strukturen. Wenn dieser XML-Code nicht in einem der Ribbon-Generatoren wie „RibbonCreator“ oder „Ribbon-Admin“ erzeugt wurde, könnt Ihr ihn ganz banal in einem beliebigen Text-Editor schreiben. Wenn man ein bisschen Übung hat und ein altes Beispiel weiterentwickelt, geht das tatsächlich ziemlich schnell.

Der XML-Code muss nicht eingerückt werden, aber das macht die Struktur besser erkennbar. Eine typische und sehr kleine Ribbon-Beschreibung sieht so aus:

```
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui">
  <ribbon startFromScratch="false">
    <tabs>
      <tab id="tabGRSR" label="GRSR" >
        <group id="grpDatenbank" label="Datenbank">
          <button id="btnBeenden" size="large" label="Diese minimale GRSR-Datenbank beenden"
               imageMso="MasterViewClose"/>
          <separator id="sep01" />
          <button idMso="FileCompactAndRepairDatabase" size="large"
               label="Datenbank komprimieren"/>
          <button idMso="DatabaseLinedTableManager" size="large" label="Tabellen verknüpfen"/>
        </group>
      </tab>
    </tabs>
  </ribbon>
</customUI>
```

Dabei ist `<customUI>` bzw. dessen Ende-Tag[3] `</customUI>` praktisch immer gleich, lediglich das scheinbare Datum ändert sich. `/2006/01/` steht für die ältere Access-Version 2007 und funktioniert immer noch, ab Access 2010 erlaubt der Wert `/2009/07/` aber mehr Fähigkeiten.

Darin geschachtelt liegt die `<ribbon>`-Struktur. Alternativ könnte hier auch die Schnellzugriffsleiste (= „quick access toolbar“) `<qat>` oder die Deaktivierung integrierter Buttons mit `<commands>` beschrieben werden.

Dann folgt eine eher fiktive Angabe, nämlich `<tabs>` im Plural für die Gesamtheit aller Register. Das einzelne, benutzerdefinierte Register steht als `<tab>` mit Angabe einer eindeutigen ID und einer Beschriftung darin.

Innerhalb des Tabs werden dann eine oder mehrere `<group>`-Elemente angelegt. Diese nehmen die eigentlichen Buttons auf, hier vom Typ `<button>`.

Dabei kannst Du auf integrierte Buttons zurückgreifen, die anhand einer `idMso` eindeutig benannt werden. Diese `idMso` findet sich im QuickInfo der Buttons im MenübandAnpassen-Dialog.

[3] Dies ist nicht der deutsche (Wochen-)„Tag“, sondern ein englisches „tag“, also eine Marke.

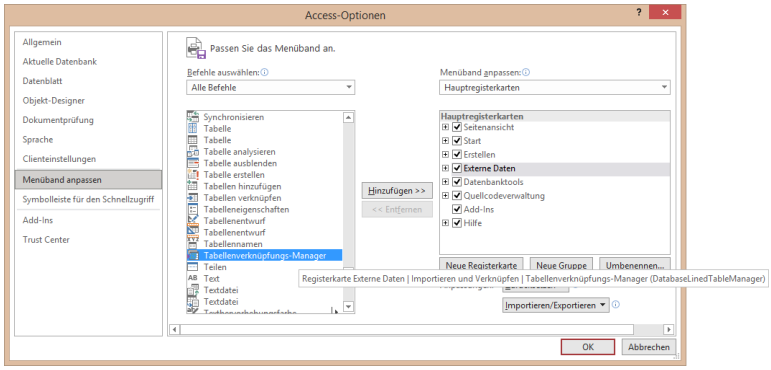

Abbildung 12: Das QuickInfo zeigt die idMso an

Diese `idMso` muss exakt in dieser Schreibweise übernommen werden, auch wenn sie wie hier Schreibfehler (`DatabaseLinedTableManager` statt eigentlich `DatabaseLinkedTableManager`) enthält. Für eigene Buttons ist es ein beliebiger Text hinter `id`, der nur eindeutig sein muss.

Das mit diesen XML-Buttons entstehende Ribbon sieht so aus:

Abbildung 13: So sieht das Ribbon aus, welches durch das obige XML beschrieben wird

Die originalen Ribbon-Elemente sind hier weiterhin vorhanden, weil `startFromScratch="false"` eingestellt ist. Steht der Wert auf `"true"`, werden praktisch alle Access-eigenen Ribbon-Elemente entfernt.

Und wo steht der XML-Code nun?

In der Datenbank musst Du eine neue Tabelle mit reservierter Struktur anlegen. Sie heißt immer „USysRibbons“ und hat definierte Felder. Als „RibbonName“ erfindest Du irgendeine Bezeichnung und kopierst den Code in das „RibbonXml“-Feld.

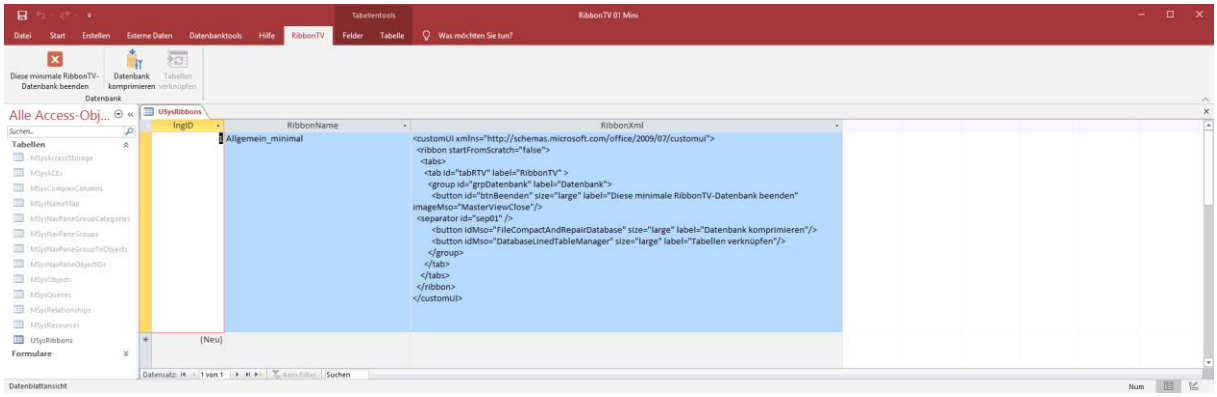

Abbildung 14: Der XML-Code steht in der Tabelle „USysRibbons“

Diese Tabelle kann mehrere XML-Definitionen enthalten, wie das für Tabellen ja zu erwarten ist. Damit die Datenbank aber weiß, welche davon sozusagen die Hauptbeschreibung des Ribbons ist, muss das noch in den Access-Optionen für diese Datenbank als „Name des Menübands“ eingetragen werden.

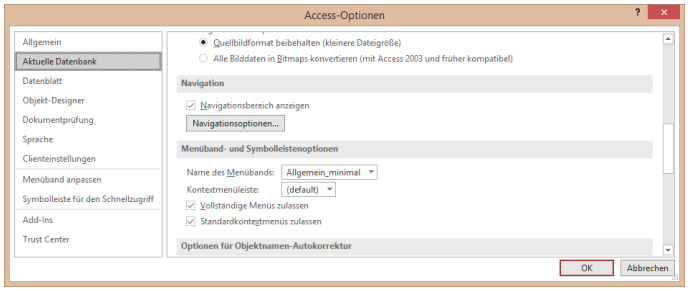

Abbildung 15: Haupt-Menüband in den Optionen eintragen

Jetzt ist alles fertig, aber noch nichts aktiv. Du musst die Datenbank jetzt schließen und neu öffnen. Wenn es geklappt hat, ist jetzt das erweiterte Ribbon mit dem eigenen Tab „RibbonTV“ und den Buttons wie in Abbildung 13 zu sehen.

Wenn es nicht klappt, passiert – nichts. Das ist sehr frustrierend, denn es gibt auch keine Fehlermeldung. Das ist in allen MS-Office-Programmen so eingestellt, dass Ribbon-Fehler erst einmal nicht gemeldet werden. Um diese Meldung zu sehen, musst Du in den Access-Optionen bei „Clienteinstellungen“ die Checkbox „Fehler von Benutzeroberflächen-Add-in anzeigen“ aktivieren:

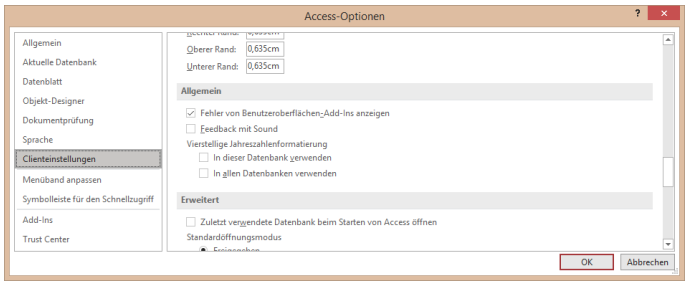

Abbildung 16: Ribbons melden erst mit dieser Option eventuelle Fehler

Damit gratuliere ich Dir zu Deinem ersten eigenen Ribbon!

22:30 Die Sendung mit dem Ribbon

Die Sendung mit dem Ribbon

„Lach- und Sachgeschichten. Heute mit Callback-Prozeduren, KontextRegistern, gemischten Registern und natürlich mit Lorenz, aber ohne die Maus und den Elefanten.

Laughing and factual stories. Today with callback procedures, contextual tabs, mixed registers and of course with Lorenz, but without the mouse and the elephant.

Das war: Englisch.”

Callback-Prozeduren

Unsere erste Sachgeschichte beschäftigt sich damit, wie eigene Buttons im Ribbon funktionsfähig gemacht werden. Dazu erweitern wir den ursprünglichen XML-Code um eigene Buttons, hier auch in der eigenen Group:

```
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui">
  <ribbon startFromScratch="false">
    <tabs>
      <tab id="tabRTV" label="RibbonTV" >
        <group id="grpDatenbank" label="Datenbank">
<!-- […  hier weiterhin wie vorher …] -->
        </group>
        <group id="grpFormUeberblick" label="Überblick-Formulare">
          <button id="btnFUNiederlassungen1" label="Ü1: nebeneinander"
  onAction="OnActionButton" imageMso="CreateStoredProcedure" getEnabled="GetEnabled" />
          <button id="btnFUNiederlassungen2" label="Ü2: Register manuell"
  onAction="OnActionButton" getEnabled="GetEnabled" />
          <button id="btnFUNiederlassungen3" label="Ü3: Register, Formular per VBA"
  onAction="OnActionButton" getEnabled="GetEnabled" />
          <button id="btnFUNiederlassungen4" label="Ü4: Register, Datenquellen per VBA"
  onAction="OnActionButton" getEnabled="GetEnabled" />
        </group>
      </tab>
    </tabs>
  </ribbon>
</customUI>
```

Das zugehörige Ribbon sieht nun so aus:

Abbildung 17: Das Ribbon enthält eigene Buttons mit integrierten Icons

Die `id` für die Buttons ist beliebig, aber eindeutig innerhalb des Codes. Ich setze wie immer ein Präfix `btn` davor, damit ich sehe, dass es die `id` für einen Button ist und weniger Verwechslungsgefahr besteht. Mit `imageMso` kann das Bild eines integrierten Buttons genutzt werden.

Das Spannende bei eigenen Buttons ist aber die Verbindung zu VBA. Da es sich bei XML-Code um einen reinen Text handelt, der nicht kompiliert wird, hat er sozusagen mit VBA nichts zu tun. Der einzige Zusammenhang besteht darin, dass in XML die Namen der bei Bedarf aufzurufenden VBA-Prozeduren stehen. Es ist kein direkter aktiver Aufruf, sondern gibt sozusagen nur die „Rückruf“-Adresse bekannt, daher „Callback“.

Die folgende (etwas vereinfachte) XML-Zeile nennt darin zwei VBA-Prozeduren:

```
<button id="btnTest" label="Test-Titel" onAction="OnActionButton" getEnabled="GetEnabled" />
```

Falls der Button angeklickt wird, ruft das Ribbon mit dem XML-Attribut `OnAction` die VBA-Prozedur `OnActionButton()` auf. Um den aktiv/inaktiv-Status zu ermitteln, ruft das Ribbon die VBA-Prozedur `GetEnabled()` auf. Die Namen beider VBA-Prozeduren sind beliebig. Sie müssen keineswegs so heißen wie die jeweiligen XML-Attribute, aber das ist üblich und übersichtlicher.

Hier ist allerdings überhaupt nicht zu erkennen, dass diese VBA-Prozeduren zwingend vordefinierte Parameter mitübergeben bekommen. Die VBA-Prozedur hat also einen freien Namen, aber trotzdem eine fest vorgegebene Signatur. Für die VBA-Prozedur `OnActionButton()` zum XML-Attribut `OnAction`, hier schon mit einer Meldung darin, sieht das so aus:

```
Sub OnActionButton(control As IRibbonControl)
    MsgBox "Klick auf """ & control.Id & """ im Ribbon."
End Sub
```

Damit das wiederum funktioniert, weil darin neue Objektdatentypen (hier z.B. `IRibbonControl`) enthalten sind, muss ein Verweis auf die „Microsoft Office 16.0[4] Object Library“ eingerichtet sein.

Ab jetzt wird ein Klick auf einen der vier `btnFUNiederlassung…`-Buttons immer nur diese Meldung zeigen. Das ist natürlich nicht perfekt. Manche erfinden jetzt für jeden Button den Namen einer eigenen VBA-Prozedur, also `KlickAufButton1()`, `KlickAufButton2()` und so weiter. Das ist nicht nur eine Riesenarbeit, sondern führt zu vielfach kopiertem Code, z.B. bei der Fehlerbehandlung.

Viel schlauer ist es, weiterhin für jedes `OnAction`-Attribut die gleiche `OnActionButton`[5]-Prozedur aufzurufen und in dieser zu unterscheiden, wer sie aufgerufen hat. Das kann ich aus der `id`[6]-Eigenschaft des `control`-Objekts erfahren:

```
Sub OnActionButton(control As IRibbonControl)
    'Reaktion auf Mausklick auf Element (Button und Menüs)
    Select Case control.Id
    Case "btnBeenden":              Application.Quit

    Case "btnFUNiederlassungen1":   OeffneFormular "frmNiederlassungenUeberblick1"
    Case "btnFUNiederlassungen2":   OeffneFormular "frmNiederlassungenUeberblick2"
    Case "btnFUNiederlassungen3":   OeffneFormular "frmNiederlassungenUeberblick3"
    Case "btnFUNiederlassungen4":   OeffneFormular "frmNiederlassungenUeberblick4"

    Case Else:
        MsgBox "Sie haben auf """ & control.Id & """ im Ribbon geklickt."
    End Select
End Sub
```

Alle Buttons rufen also einheitlich die gleiche `OnActionButton()`-Prozedur auf und falls ich mich bei deren Namen vertippt haben sollte, erscheint hier eine Meldung. Das erleichtert die Fehlersuche später beträchtlich.

Genau das Gleiche gilt auch für die anderen Callback-Prozeduren, hier also für das XML-Attribut `getEnabled`:

```
Sub GetEnabled(control As IRibbonControl, ByRef enabled)
    'gibt zurück, ob dieses Element aktiviert sein soll

    enabled = True
    Select Case control.Id
    Case "btnFUNiederlassungen1": enabled = BenutzerIstAdmin()
    Case "btnFUNiederlassungen2": enabled = BenutzerIstAdmin()
    Case "btnFUNiederlassungen3": enabled = BenutzerIstAdmin()
    Case "btnFUNiederlassungen4": enabled = BenutzerIstAdmin()
    Case Else: enabled = True
    End Select
End Sub
```

Wie zu sehen ist, unterscheidet sich die Signatur (das ist die Liste der Parameter) von derjenigen zu `OnActionButton`. Außerdem ist `enabled` ein sogenannter Rückgabeparameter, der also ungewohnterweise keinen Wert entgegennimmt, sondern an den aufrufenden Ribbon-Button zurückgibt.

In beiden Fällen müssen natürlich intern aufgerufene Prozeduren VBA-seitig vorhanden sein. Sowohl die Sub-Prozedur `OeffneFormular()` als auch die `Boolean`-Function `BenutzerIstAdmin()` sind hier zwar beispielhaft benutzt, Ihr müsst sie aber selber noch irgendwo definieren.

Davon abgesehen enthält dieses Ribbon nun fünf eigene und hoffentlich funktionsfähige Buttons.

---

[4] Dies gilt für Access2016, bei anderen Versionen steht hier natürlich eine andere Nummer.
[5] Warum heißt die VBA-Prozedur bei mir eigentlich nicht auch nur `OnAction` wie der Attributname? Weil es irgendwann mal auch `OnActionCheckbox`- oder `OnActionToggleButton`-Prozeduren geben wird mit abweichenden Signaturen.
[6] Achtung, der Name ist irreführend! Die `id` ist keineswegs eine Zahl, sondern der eindeutige Namenstext aus dem XML-Code.

23:15 Ich bin ein Ribbon – holt mich hier rein!

Jetzt haben wir zwar funktionsfähige Ribbons, aber das ist nur die Pflicht. Nun kommt die Kür. Unter den ganzen Promis in der Tabelle „USysRibbons“ küren wir in verschiedenen Dschungel-Prüfungen den Ribbon-König oder die Ribbon-Königin.

Bis dahin war es nett, zeigt aber überhaupt nicht, welches Potential darin steckt. An dieser Stelle enden viele Ribbons und das Publikum senkt den Daumen nach unten.

Ihr habt Euch wahrscheinlich die ganze Zeit schon gewundert, warum das einzige Ribbon überhaupt in einer Tabelle gespeichert wurde. Weil Access es nicht besser kann? Nein, weil es mehrere Ribbons geben sollte.

Und selbst dieses eine Ribbon sollte schon mehr können. Bisher sind darin ausschließlich die allgemeinen Buttons beschrieben, die immer und überall sichtbar sind. Aber ohne eine einzige Zeile VBA lassen sich in XML auch diejenigen Tabs/Groups/Buttons definieren, die nur dann erscheinen, wenn z.B. ein Formular geöffnet ist.

Anstatt also die normalen Tabs mit Buttons vollzupflastern, die etwa für erster/voriger/nächster/letzter Datensatz zuständig sind, gibt es diese nur genau dann, wenn ein Formular[7] offen ist. Ansonsten sind sie ohne Daten schlicht überflüssig und stören.

Diese besonderen Register nennen sich `contextualTabs`, also welche, die im Zusammenhang mit bestimmten Objekten auftauchen. Früher waren die mal dick farbig hervorgehoben, mit den neueren Access-Versionen ist das zunehmend unauffälliger. Diese Kontext-Register gibt es auch ohne selbstdefinierte Ribbons. Sie erscheinen immer dann, wenn bestimmte Objekttypen aktiv sind, hier beispielsweise für eine geöffnete Tabelle (Register „Felder“ und „Tabelle“ unterhalb von „Tabellentools“):

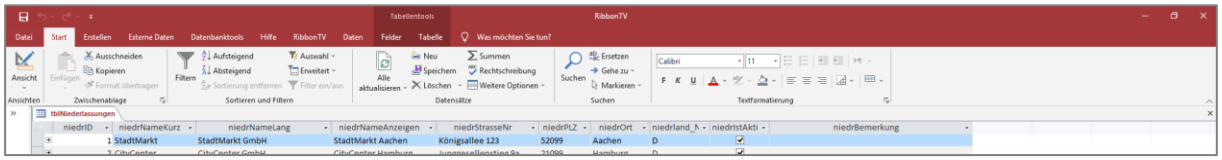

Abbildung 18: Integriertes contextualTab „Tabellentools“

Dabei ist die dunkelrote Farbe über diesen contextualTab-Registern etwas dunkler und enthält bei Formularen die Beschriftung des Formulars (siehe Abbildung 19)

Es ist also sinnvoll, dass ich für spezielle eigene Objekte (Formulare, Berichte, etc.) auch spezielle contextualTabs vorbereite. Bevor ich die Details dazu erläutere, noch schnell ein Überblick über die Kandidaten und Kandidatinnen.

- Das bisher einzige Ribbon namens „Allgemein“ enthält den Abschnitt `<ribbon>` und beschreibt damit die immer sichtbaren Tabs/Groups/Buttons. Damit das funktioniert, muss sein Name ja in der Access-Option „Name des Menübands“ eingetragen sein.
- Ab jetzt gibt es dort einen weiteren Abschnitt `<contextualTabs>` für diese nur manchmal sichtbaren Elemente. Wenn ein Formular in der Eigenschaft „Name des Menübands“ dann „Allgemein“ angibt, erscheinen die Inhalt von `<contextualTabs>` automatisch mit dem Formular.

Das bedeutet, dass in diesen `<contextualTabs>` von „Allgemein“ alle Buttons stehen, die in allen Formularen identisch sind. Typische Buttons dafür sind „Formular schließen“ oder die Navigations-Buttons. Das ist der wesentliche Unterschied zu den alten Buttons, welche einzeln auf jedes Formular kopiert werden mussten: Es reicht, den Namen des allgemeinen Ribbons in der Formular-Eigenschaft

[7] Das lässt sich ebenso mit Tabellen/Abfragen/Berichten synchronisieren. Das Formular steht hier nur als häufigstes Beispiel.

„Name des Menübands“ einzutragen und alles läuft! Wenn ich dort nichts eintrage, sind nur die allgemeinen `<tabs>` des Haupt-Ribbons zu sehen, nicht aber die `<contextualTabs>`.

Der erweiterte XML-Code für das „Allgemein“-Ribbon sieht so aus:

```
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui">
  <ribbon startFromScratch="false">
    <tabs>
       <!— ... wie bisher ... -->
    </tabs>

    <contextualTabs>
      <tabSet idMso="TabSetFormReportExtensibility">
        <tab id="tabFenster" label="Fenster">
          <group id="grpFormular" label="Allgemein">
            <button id="btnSchliessen" size="large" label="Fenster schließen"
             imageMso="FileExit" onAction="OnActionButton"/>
            <button idMso="RecordsSaveRecord" size="large" label="Daten speichern"
             imageMso="FileSave"/>
            <button idMso="RecordsRefreshRecords" size="large" label="Daten aktualisieren"/>
          </group>
          <group id="grpDatensaetze" label="Datensätze">
            <button idMso="MailMergeGoToFirstRecord" size="large"/>
            <button idMso="MailMergeGoToPreviousRecord" size="large"/>
            <button idMso="MailMergeGoToNextRecord" size="large"/>
            <button idMso="MailMergeGotToLastRecord" size="large"/>
            <button idMso="GoToNewRecord" size="large"/>
          </group>
        </tab>
      </tabSet>
    </contextualTabs>
  </ribbon>
</customUI>
```

Die spezielle `idMso="TabSetFormReportExtensibility"` gibt vor, dass diese Elemente automatisch erscheinen, sobald ein Formular oder Bericht aufgerufen wird, in welchem die „Name des Menübands“-Eigenschaft auf „Allgemein“ steht, weil mein Haupt-Ribbon genau so heißt.

Das erzeugt das folgende Ergebnis, wenn dieser Ribbon-Name im Formularentwurf eingetragen ist:

Abbildung 19: Eigenes contextualTab für das Formular "Überblick 1: Niederlassungen"

Nur `btnSchliessen` braucht in `OnActionButton()` eigenen VBA-Code. Alle anderen sind integrierte Buttons, die hier lediglich nach eigenen Wünschen angeordnet sind. Ihre Funktionalität ist ja schon gegeben.

Erste Dschungelprüfung bestanden.

- Jetzt kommen die nächsten Kandidaten ins Dschungel-Camp „USysRibbons“, nämlich weitere Ribbons. Alle Ribbons außer dem ersten sind normalerweise nur Erweiterungen, müssen also nichts wiederholen, was es schon gibt. Daher steht deren `StartFromScratch`-Attribut auf `"False"`, sonst wäre das „Allgemein“-Ribbon selber bei ihrem Aufruf geleert.

Da das hier kein Ponyhof, sondern ein knallharter Konkurrenzkampf ist, halten auch die Tabs voneinander Abstand. Das bedeutet, dass ohne explizite Anforderung die im weiteren XML-Code enthaltenen Tabs neben den bisherigen stehen. Das ist nicht schlimm, sondern manchmal durchaus gewollt. Aber schon mal als Ausblick auf die Finalrunde: Die Buttons aus verschiedenen Ribbons lassen sich auch nebeneinander auf dem *gleichen* Tab anzeigen!

Jetzt geht es also darum, einem Formular zusätzliche eigene Icons im Ribbon mitzugeben. Dazu braucht es eine neue Zeile in der Tabelle „USysRibbon“ mit einem eindeutigen Namen. Ich benenne diese Erweiterungsribbons einfach mit dem Namen des Formulars, in dem sie benutzt werden.

Der zweite Datensatz mit der Bezeichnung „frmLogin“ enthält also diesen XML-Code:

```
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui">
  <ribbon startFromScratch="false">
    <contextualTabs>
      <tabSet idMso="TabSetFormReportExtensibility">
        <tab id="Fenster" label="Login-Fenster">
          <group id="grpLogIn" label="LogIn">
            <button id="btnLogInAlle" size="large" label="Alle Benutzer"
              imageMso="DatabaseUserLevelSecurityWizard" getEnabled="GetEnabled"
              onAction="OnActionButton" />
            <button id="btnLogInMeine" size="large" label="Benutzer mit meinem LogIn"
              imageMso="DatabaseUserLevelSecurityWizard" onAction="OnActionButton" />
            <separator id="sep01" />
            <button id="btnLogInDieser" size="large" label="Diesen Benutzer auswählen"
              imageMso="DatabaseUserLevelSecurityWizard" onAction="OnActionButton" />
          </group>
        </tab>
      </tabSet>
    </contextualTabs>
  </ribbon>
</customUI>
```

Dieser XML-Code enthält ausschließlich `<contextualTabs>`-Angaben. Im Formular „frmLogin“ muss für die „Name des Menübands“-Eigenschaft dann dieser Ribbon-Name „frmLogin“ eingegeben werden. Fertig.

Damit es funktioniert, ist ein erneuter Start der Datenbank notwendig und danach präsentieren sich die Tabs für dieses Formular so:

Abbildung 20: Das allgemeine contextualTab „Fenster“ sieht aus wie in anderen Formularen

Abbildung 21: Das spezielle contextualTab „Login-Fenster“ gibt es nur in diesem Formular

Zusätzlich zu dem allgemeinen contextualTab „Fenster“ erscheint nur in diesem Formular also daneben ein zweites contextualTab „Login-Fenster“.

Zweite Dschungelprüfung bestanden (aber wegen des Auftauchens nur im zweitem Tab mit leichten Abzügen in der B-Note).

Es ist nämlich dermaßen viel Platz übrig, dass es sinnvoller wäre, die zusätzlichen Icons *neben* den allgemeinen Icons auf dem gleichen Tab „Fenster“ unterzubringen. Wer jetzt versucht, einfach die gleichen `id`-Werte zu benutzen, also `id="Fenster"`, wird leider feststellen müssen, dass das nichts hilft. Die waren nämlich längst identisch, nur die Beschriftungen haben sich unterschieden.

Auch der nächste Anpassungsversuch, dass beide `<tab>`-Elemente mit `label="Fenster"` identisch beschriftet werden, bringt keine Änderung.

Es braucht einen sogenannten „Namespace“. Das ist so eine Art Vereinbarung mit dem Ribbon, dass sich gleichnamige Objekte wirklich im gleichen Namensraum bewegen sollen. Dieser Namespace muss allen Beteiligten mitgeteilt werden, also sowohl dem Haupt-Ribbon als auch allen Erweiterungen. Der XML-Code im Ribbon „Allgemein“ ändert sich ganz am Anfang in:

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

Der Buchstabe `a` ist übrigens ebenso beliebig wie die folgende Bezeichnung `"RTV"`. In den weiteren Ribbons muss dieser Namespace dann anhand seiner Kurzbezeichnung `a` identisch angemeldet und in den Details benutzt werden:

```
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui" xmlns:a="RTV">
  <ribbon startFromScratch="false">
    <contextualTabs>
      <tabSet idMso="TabSetFormReportExtensibility">
        <tab idQ="a:Fenster" label="Fenster">
              <!– ... weiterer Code ... -->
```

Diese Benutzung des gleichen Namensraums findet für das `<tab>` statt. Damit der Namespace als solcher erkannt wird, ist es nicht mehr das `id`-Attribut, sondern ein `idQ`-Attribut.

Und plötzlich stehen die Icons aus zwei Ribbons im gleichen Register:

Abbildung 22: Allgemeines und spezielle contextualTab werden im gleichen Register angezeigt

Anstatt also, wie ich es leider immer wieder sehe, für jedes Formular ein komplettes Tab zu definieren, geht es viel einfacher. Ein *allgemeines* contextualTab gibt die überall gewünschten Standard-Buttons vor und ein *spezielles* contextualTab ergänzt weitere Buttons, falls es besondere Anforderungen gibt.

Nächste Dschungelprüfung bestanden.

Ach ja, die Menüs. Gibt es immer noch jemanden, der nur darauf wartet, wie mit den neuen Ribbons die alten Menüs erzeugt werden? Echt jetzt? Okay, also ganz kurz: Es gibt zwei Arten Menüs, die statischen (`menu`) und die dynamischen (`dynamicMenu`). Die statischen stehen komplett schon im XML-Code, die dynamischen werden erst beim Klick auf ihren Menütitel per VBA zusammengebaut.

Das statische Menü ist im Grunde nur eine Extra-Gruppe um ganz normale Buttons, die dann bloß etwas anders aussehen.

```
<!– ... vorheriger Code ... -->
</group>
<group id="grpFormUeberblick" label="Überblick-Formulare">
  <menu id="menUeberblicke" size="large" label="Verschiedene Überblicke"
   imageMso="CreateStoredProcedure" itemSize="large">
    <button id="btnFUNiederlassungen1" label="Überblick 1: nebeneinander"
      onAction="OnActionButton" image="UeberblickNr1.gif"
      description="Erste Variante, alle ... nebeneinander"/>
    <button id="btnFUNiederlassungen2" label="Überblick 2: Register manuell"
      onAction="OnActionButton" image="UeberblickNr2.png"
      description="Zweite Variante, alle ... zusammengefasst"/>
    <button id="btnFUNiederlassungen3"
      label="Überblick 3: Register, Formular-Umschaltung per VBA" onAction="OnActionButton"
      image="UeberblickNr3.png" description="Dritte Variante, alle ... Inhalte"/>
    <button id="btnFUNiederlassungen4"
      label="Überblick 4: Register, Datenquellen-Umschaltung per VBA"
      onAction="OnActionButton" image="UeberblickNr4.png"
      description="Vierte Variante, ein ... VBA"/>
  </menu>
</group>
<!– ... weiterer Code ... -->
```

Dieser XML-Code[8] innerhalb des Haupt-Ribbons erzeugt ein Menü mit einer (wegen der Angabe `size = "large"`) großen Titel-Startfläche und darunter vier einzelnen Menüzeilen. Diese sind (wegen `itemSize = "large"`) ebenfalls groß und zeigen daher auch die Inhalte Ihrer `description`-Eigenschaft darunter:

[8] Die `label`-Texte sind hier im Code gekürzt, damit es übersichtlicher ist. Die `image`-Eigenschaft mit Angabe einer Datei (empfohlen gif oder png mit 64*64 Pixeln) funktioniert erst, wenn auch `LoadImage` und der passende VBA-Code vorhanden ist. Da das über den hier möglichen Umfang hinausgeht, guckt Ihr bitte in den Beispieldateien nach. Zum Testen reichen stattdessen auch `imageMso`-Attribute mit den integrierten Icons.

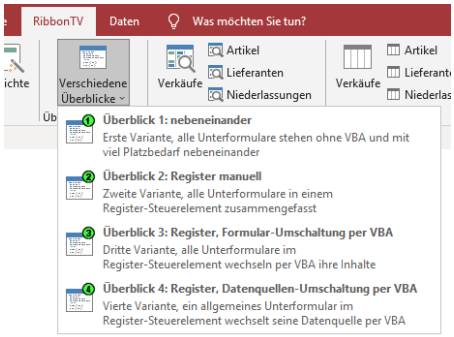

Abbildung 23: Ausschnitt aus dem Ribbon mit einem statischen Menü

Wer gerne die Optik alter Menüzeilen haben möchte, ändert die Angaben auf `size ="normal"` bzw. `itemSize="normal"`. Da das die Standardwerte sind, können diese Angaben auch weggelassen werden. Dann sieht es so aus:

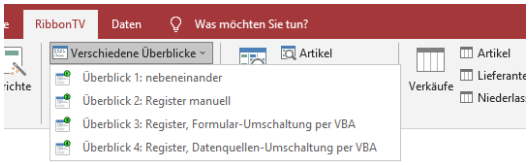

Abbildung 24: Statisches Menü mit kleinen Elementen

Das gleiche `size`-Attribut funktioniert bei „normalen“ Buttons übrigens genau so, wie für die im Beispiel sichtbaren Buttons „Verkäufe“ (`large`) und „Artikel“ (`normal` oder Attribut weglassen) zu sehen ist.

Wirklich hilfreich ist aber das `dynamicMenu`-Element. Es erzeugt seine Menüeinträge erst zur Laufzeit, nämlich beim Klick auf seinen Titel. Damit ist eines der lästigsten Probleme in Ribbons gelöst: Die dauernd notwendige und leider nicht immer reibungslose Aktualisierung aller sichtbaren Icons. Die Menüeinträge werden ja erst dann erzeugt und sichtbar, sobald der Klick erfolgt.

In den Beispieldatenbanken gibt es ein Login-Formular, mit welchem ich verschiedene Rollen annehmen kann oder alternativ anhand meines Windows-Logins erkannt werde. Je nach Rolle sehe ich unterschiedlich viele Menüeinträge, die dynamisch erzeugt werden. Als einfacher und ziemlich rechtloser Gast sehe ich bloß einen Menüeintrag mit drei Untermenüeinträgen:

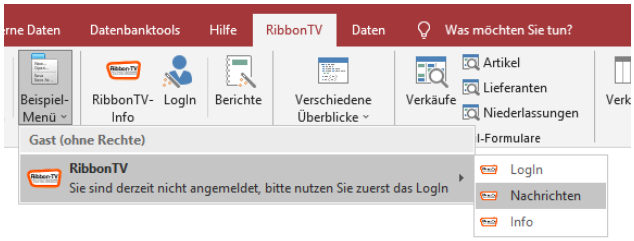

Abbildung 25: Dynamisches Menü mit wenigen Rechten

Diese Kurzfassung des Menüs zeigt der Code an, wenn jemand nicht eingeloggt ist[9].

Nachdem ich (in diesem Fall als Admin mit maximalen Rechten) eingeloggt bin und damit klar ist, welche Rechte ich habe und welche Aktionen/Objekte für mich erlaubt sind, ändert sich das gleiche Menü[10] so:

[9] Diesen Gaststatus zeigt das Menü praktischerweise auch nach schweren Laufzeitfehlern, wenn globale Variablen leer sind. Nach dem so erzwungenen neuen Einloggen befindet sich die Datenbank dann wieder in einem definierten Zustand.
[10] Die hier gezeigten Menüeinträge sind nur Dummies und von verschiedenen meiner echten Datenbanken inspiriert.

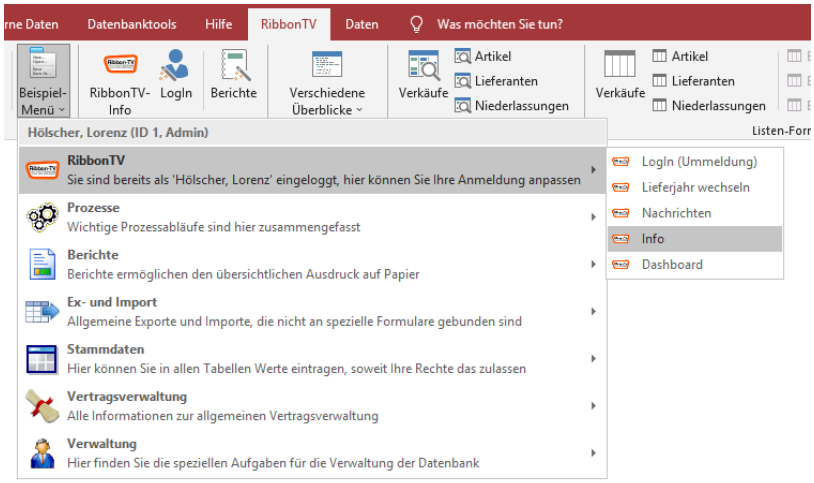

Abbildung 26: Dynamisches Menü mit allen Rechten

Die Definition dieses dynamischen Menüs ist sehr einfach, weil im XML-Code eigentlich nur der Menü-Titel und die Callback-Prozedur genannt werden müssen

```
  <button idMso="DatabaseLinedTableManager" size="large" label="Tabellen verknüpfen"/>
  <separator id="sepD02" />
  <dynamicMenu id="menHaupt" label="Beispiel- Menü" size="large" image="icnMenue.gif"
    getContent="GetContent" invalidateContentOnDrop="true" />
</group>
<group id="grpSpezial" label="Spezial-Formulare">
```

Das `invalidateContentOnDrop`-Attribut sorgt dafür, dass der Menüinhalt auch bei jedem Öffnen wirklich neu erzeugt wird, das ließe sich sonst auch nach erstmaligem Erzeugen unterdrücken. Die eigentliche Arbeit macht die VBA-Prozedur `GetContent()`, welche eine Zeichenkette mit dem XML-Code für das gezeigte Menü zurückliefern muss.

Anstatt hier jetzt nur eine statische Zeichenkette zu erzeugen, wird das ganze eigentlich erst wirklich praxistauglich, wenn beispielsweise die je nach Rechtegruppe erlaubten Menü-Einträge aus einer Tabelle ausgelesen werden. Das wiederum braucht außer den beteiligten Tabellen rund 100 Zeilen VBA-Code, die ich hier nicht alle abdrucken möchte. Ihr könnt es Euch im Modul „USys_modRibbonDynamicMenue“ ansehen[11].

Dort ganz am Ende der Prozedur ist auch noch `Debug.Print strXML` kommentiert. Wer sich das XML-Ergebnis detailliert ansehen will, entfernt den Kommentar und guckt nach Menüaufruf im Direkt-Fenster nach.

Letzte Dschungelprüfung bestanden.

Das Ribbon wird hiermit zum Dschungelkönig ernannt!

[11] Auch hier verlangt das `image`-Attribut die `LoadImage`-Callback-Prozedur. Stattdessen ist `imageMso` mit Angabe eines integrierten Icons möglich.

00:30 Das Literarische Duett

Natürlich kann ich auf diesen wenigen Seiten bzw. in der kurzen Zeit nicht alles erklären, was sich mit Ribbons so machen lässt. Dafür gibt es einfach zu viele Möglichkeiten.

Wie schon kurz erwähnt, können mit dem Ribbon-XML außerdem die internen Befehle wie Speichern, Aktualisieren, etc. umgeleitet werden. Selbstverständlich gehören eigene Icons schon fast zum Pflichtprogramm einer professionellen Ribbon-Oberfläche und brauchen nur ein paar wenige Zeilen VBA-Code. Den Backstage-Bereich (der sich hinter dem [Datei]-Register verbirgt) habe ich überhaupt noch nicht erwähnt, weil dessen Möglichkeiten hier viel zu umfangreich wären.

Ihr findet in den beiliegenden Beispiel-Datenbanken daher noch viel mehr, als ich hier erläutert habe. Nehmt sie als Anregung und nehmt sie auseinander, um zu sehen, warum und wie etwas funktioniert. Lest die XML-Anweisungen und den VBA-Code und lasst Euch inspirieren.

Die erste richtige Lese-Empfehlung aber betrifft die AEK 17. Michael Zimmermann hat dort bereits einen Vortrag über den technischen Aufbau von Ribbons gehalten und das lässt sich als PDF zum Nachlesen auf http://www.donkarl.com/?aek downloaden.

Denjenigen, die sich noch intensiver damit beschäftigen möchten, kann ich unbedingt das folgende Buch empfehlen. Auch wenn es laut Titel eigentlich für die Access-Version 2007 gilt, enthält es immer noch alles, was Ihr zu Ribbons wissen müsst:

André Minhorst/Melanie Breden:
Ribbon-Programmierung für Office 2007
ISBN 978-3-8273-2738-3, Addison-Wesley

Nachträge nach der AEK:

Es ist für demnächst eine neue Version des obigen Buches angekündigt.

Thomas Pfoch empfiehlt für die Erstellung der Icons, auf umfangreiche Icon-Bibliotheken zurückzugreifen, beispielsweise in SyncFusion.

Sabine Konschak beschreibt in https://www.konschak.de/eigenes-icon-fuer-ribbon-erstellen/, wie sich Icons sogar in PowerPoint erstellen lassen.

01:30 Sendepause

Ich hoffe, ich konnte zeigen, dass Ribbons sehr leicht zu erstellen sind, wenn die Anfangshürde geschafft ist. Ja, zuerst sind sie sperrig und bieten scheinbar mehr Fehlerquellen als Lösungen. Aber wenn die Struktur einmal steht und nur noch erweitert werden muss, geht es wirklich schnell und einfach.

Für uns Programmierer:innen vereinfacht sich vieles. Ich muss keine Standard-Schaltflächen-Prozeduren mehr kopieren, ich muss keine VBA-Tricks zum Ein- und Ausblenden erfinden und die Befehle sind sogar vollautomatisch angeordnet. Die Technik erlaubt weiterhin, sich nur auf Menüs zu beschränken, aber selbst diese können viel mehr als früher.

Und aus Sicht der Benutzer:innen wirken Ribbons endlich nicht mehr wie das altmodische Design der 1980er Jahre, sie passen sich sogar neuen Access-Versionen sofort optisch an. Sie bieten eine robuste, einheitliche Bedienungsoberfläche und lassen sich bequem ein- und ausblenden.

Mit diesen Gedanken zur Nacht beenden wir das Programm und ich wünsche Euch viel Erfolg dabei, eigene Ribbons in Euren Access-Datenbanken zu nutzen.

Lorenz Hölscher