RibbonX - Tutorial

(Einführung in die Programmierung der Multifunktionsleiste)

Excel 2007

Inhaltsverzeichnis

Zum Inhaltsverzeichnis !

RibbonX - Teil 1 – Einführung

Dieses Tutorial richtet sich an Anwender, welche bereits Erfahrung mit der Programmierung in Office gesammelt haben.

Wichtig:

Wenn Sie die Datei auch in älteren Office Versionen nutzen möchten oder müssen, dann sollten Sie auf RibbonX verzichten und dafür benutzerdefinierte Symbolleisten über VBA programmieren.

RibbonX ist die neue Benutzeroberfläche in Office 2007. Es ist möglich, eigene Ribbon zu programmieren. Hierbei gibt es zwei Möglichkeiten.

1. Das Ribbon wird an die Standard Tabs angehängt (startFormSratch="False").
2. Das Ribbon wird ohne die Standard Ribbon angezeigt (startFormSratch="True").

Zum Einfügen eines RibbonX-Codes in eine Office 2007-Datei benötigen sie den kostenlosen CostumUI-Editor, mit welchem Sie den Code in Dateien für Excel 2007, Word 2007 und PowerPoint 2007 einfügen können. Access-Dateien können Sie damit nicht bearbeiten, da die Vorgehensweise hierfür anders ist. Für Access 2007 besuchen Sie die Seite Access Ribbon erstellen.

RibbonX folgt einer klar definierten Struktur, welche hier kurz erklärt werden soll.

Die Grundstruktur im Überblick :

```
<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui"
onLoad="onLoad">
<commands>
<!-- Hier werden die Commands aufgelistet -->
</commands>
<ribbon>
<qat><documentControls>
<!-- Hier werden die Elemente aufgelistet -->
</documentControls></qat>
<officeMenu>
<!-- Hier werden die Elemente aufgelistet -->
</officeMenu>
<contextualTabs>
      <!-- Hier wird der TabSetname angegeben>
       <!-- visible="True", wenn nur ein Tab ausgelendet
werden soll -->
       <tabSet idMso="TabSetNameAufEnglisch" visible="false">
       <!-- Hier werden einzelne Tabs ausgeblendet -->
        <!-- visibel weglassen, wenn nur einzelne Tabs ausgeblendet
werden sollen -->
       <tab idMso="TabNameAufEnglisch" visible="false">
         <!-- Hier werden einzelne Gruppen ausgeblendet -->
          <group idMso="GruppenNameAufEnglisch"
                visible="false">
          </group>
      </tab>
       </tabSet>
    </contextualTabs>
```

```
<tabs>
<tab>
<group>
<!-- Hier werden die Elemente aufgelistet -->
</group>
<group>
<!-- Hier werden die Elemente aufgelistet -->
</group>
</tab>
<tab>
<group>
<!-- Hier werden die Elemente aufgelistet -->
</group>
<group>
<!-- Hier werden die Elemente aufgelistet -->
</group>
</tab>
</tabs>
</ribbon>
</customUI>
```

Das Onload-Ereignis. Dieses ist optional und sorgt dafür, dass das Ribbon dynamisch zur Laufzeit aktualisiert werden kann. Dafür kopieren Sie den folgenden Code in ein Standardmodul der Datei. Die Datei muss dabei mit der Endung *.xlsm (Excel 2007 Arbeitsmappe mit Makros), *.xlam (Excel 2007 Makro), *.docm (Word 2007 Dokument mit Makros) oder *.dotm (Word 2007 Vorlage mit Makros) gespeichert sein. Das Makro muss an oberster Stelle im Modul stehen (unter der Zeile Option Explicit)

```
Public objRibbon As IRibbonUI
Public Sub rx_onload(ribbon As IRibbonUI)
Set objRibbon = ribbon
End Sub
```

An erster Stelle steht die Commandoebene, über welche zum Beispiel die Exceloptionen deaktiviert werden können. Diese Einstellung hat nur Auswirkung auf die Datei, in welcher die Anweisung steht.

Erst jetzt wird das Ribbon eröffnet. Im Ribbon-Stammtag wird auch startFromScratch angegeben.

An erster Stelle im Ribbon-Tag wird die Schnellstartleiste (Quick Access Toolbar -> QAT) beeinflusst.

Als nächstes kann das Officemenü beeinflusst werden.

Hinweis:
Das Beeinflussen der QAT hat nur Einfluss, wenn startFromScratch auf True gestellt wird.
Das Beeinflussen des Office Menüs hat nur Auswirkung auf die Datei, in welcher der RibbonX-Code die Anweisung enthält. Nach dem Schließen der Arbeitsmappe oder ein Wechsel in eine andere Arbeitsmappe stellt die Standardeinstellungen wieder her.

Jetzt können Kontextregisterkarten (contextualTabs), einzelne Tabs derselben oder einzelne Gruppen der Tabs deaktiviert werden.

Jetzt können die Tabs erstellt werden. Es ist möglich, mehrere Tabs zu erstellen. Welche Elemente genutzt werden können, wird in weiteren Teilen erklärt. Jedes Tab enthält mindestens eine Gruppe mit mindestens einem Element.

OfficeControls.zip

Hier gibt es ein Add-In für Excel 2007. Mit diesem können Sie sich die in Office 2007 eingebauten Icons anzeigen lassen. Entpacken Sie das Add-In in ein beliebiges Verzeichnis und binden es dann in Excel ein. Danach wird im Tab Entwicklertools eine neue Gruppe mit 9 Galerien eingefügt, über welcher die verschiedenen Icons angezeigt werden. Nach einem Klick auf ein Icon können Sie eine Codezeile in die active Zelle schreiben lassen. Den Namen des Icons benötigen Sie für das Tag imageMso im RibbonX-Code.

Übersicht eingebaute Steuerelemente.zip

Hier gibt es die Übersichten über die eingebauten Steuerelemente. Die Übersichten liegen als Excel 2007 Tabellen vor. Es gibt jeweils eine Übersicht für Word 2007, Excel 2007, PowerPoint 2007 und Access 2007. Für die Korrektheit der Angaben wird keine Garantie übernommen.

Dateien im Anhang:
Liste der VBA-Prozeduren
Liste der RibbonX-Elemente
Liste der RibbonX-Attribute
Beschreibung der RibbonX-Attribute

Zum Inhaltsverzeichnis !

RibbonX - Teil 2 - Schaltflächen (Button)

In diesem Teil wollen wir einen Button erstellen. Hierfür wird folgender RibbonX-Code verwendet. Der Beispielcode erstellt zwei Schaltflächen.

Erstellen Sie eine neue Arbeitsmappe und speichern Sie diese mit der Endung *.xslm. Fügen Sie ein neues Modul ein, in welches Sie die folgenden Makros kopieren.

```
Option Private Module
Public objRibbon As IRibbonUI
Public Sub onload(ribbon As IRibbonUI)
Set objRibbon = ribbon
End Sub
Sub Button1_OnAction(control As IRibbonControl)
If ThisWorkbook.Sheets("Tabelle1").Columns("A:A"). _
EntireColumn.Hidden = False Then
ThisWorkbook.Sheets("Tabelle1").Columns("A:A"). _
EntireColumn.Hidden = True
MsgBox "Spalte A wurde ausgeblendet", vbOKOnly + _
vbInformation, "Hinweis"
Else
ThisWorkbook.Sheets("Tabelle1").Columns("A:A"). _
EntireColumn.Hidden = False
MsgBox "Spalte A wurde eingeblendet", vbOKOnly + _
vbInformation, "Hinweis"
End If
End Sub
Sub Button2_getLabel(control As IRibbonControl, ByRef label)
If ThisWorkbook.Sheets("Tabelle1").Columns("B:B"). _
EntireColumn.Hidden = False Then
label = "Spalte B eingeblendet"
Else
label = "Spalte B ausgeblendet"
End If
End Sub
Sub Button2_OnAction(control As IRibbonControl)
If ThisWorkbook.Sheets("Tabelle1").Columns("B:B"). _
EntireColumn.Hidden = False Then
ThisWorkbook.Sheets("Tabelle1").Columns("B:B"). _
EntireColumn.Hidden = True
objRibbon.Invalidate
MsgBox "Spalte B wurde ausgeblendet", vbOKOnly + _
vbInformation, "Hinweis"
Else
ThisWorkbook.Sheets("Tabelle1").Columns("B:B"). _
EntireColumn.Hidden = False
objRibbon.Invalidate
MsgBox "Spalte B wurde eingeblendet", vbOKOnly + _
vbInformation, "Hinweis"
End If
End Sub
```

Anschließend beenden Sie Excel und öffnen die Datei mit dem CustomUI-Editor. Fügen Sie in das Codefenster den folgenden Code ein.

```
<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui"
onLoad="onload">
<ribbon startFromScratch="true">
<tabs>
<tab id="tab01" label="Schaltflächen">
<group id="grp01" label="Spalte A" >
<button id="tgb01" label="Spalte A" imageMso="ExchangeFolder"
onAction = "Button1_onAction"
size="large"/>
</group>
<group id="grp02" label="Spalte B" >
<button id="tgb02" getLabel="Button2_getLabel" imageMso="ExchangeFolder"
onAction = "Button2_onAction"
size="large"/>
</group></tab></tabs></ribbon></customUI>
```

Speichern Sie die Änderungen und schließen den Editor. Öffnen Sie die Datei mit Excel. Jetzt sollte ein Ribbon mit zwei Schaltflächen erscheinen. Die erste blendet Spalte A im Wechsel ein oder aus. Die Schaltflächenbeschriftung ändert sich nicht. Schaltfläche 2 blendet Spalte B im Wechsel ein oder aus, wobei die Schaltflächenbeschriftung den jeweiligen Zustand anzeigt.

Schaltfläche 1:
Diese Schaltfläche hat eine feste Beschriftung.

Schaltfläche 2:
Diese Schaltfläche bezieht ihre Beschriftung aus dem Makro Button2_getLabel.

Beschreibung:

id: Legt die control.ID fest.

onAction: Gibt das zu startende Makro an

label: Beschriftung der Schaltfläche (für Schaltfläche 1)
getLabel: Gibt das Makro an, welches die Beschriftung anhand bestimmter Einstellungen einträgt.

Wichtig: Es ist immer nur eines möglich. Entweder label oder getLabel.

imageMso: Gibt das Icon an, welches genutzt werden soll.

size: gibt die Größe der Schaltfläche an. large=groß. normal=klein

Sie können auch die Beispielmappe **XL07_Button.**xlsm nutzen (im Anhang).

RibbonX - Teil 3 - Umschaltflächen (Tooglebutton)

In diesem Teil wollen wir eine Umschaltfläche (ToggleButton) erstellen. Hierfür wird folgender RibbonX-Code verwendet. Der Beispielcode erstellt eine Schaltfläche.

Bei Umschaltflächen muss der jeweilige Zustand gespeichert werden, damit dieser beim erneuten Öffnen der Arbeitsmappe wieder zur Verfügung steht. Ohne speichern des Zustandes geht selbiger beim Schließen der Arbeitsmappe verloren. Es gibt mehrere Möglichkeiten, den Zustand zu speichern. Die Registry, eine INI-Datei, eine Textdatei, In einer Zelle der Arbeitsmappe oder die DocumentProperties. Da letzteres am geeignetsten erscheint, wird hier dieses Beispiel vorgestellt.

Erstellen Sie eine neue Arbeitsmappe und speichern Sie diese mit der Endung .xslm. Fügen Sie ein neues Modul ein, in welches Sie die folgenden Makros kopieren. Mit diesem wird ein neuer Eintrag in den DocPorperties erstellt. Für jede Umschaltfläche benötigen Sie einen eigenen Eintrag. Nach dem Erstellen des Eintrages können Sie das Makro wieder löschen.

```
Sub DocumentProbertieFürToggleButtonAnlegen()
ActiveWorkbook.CustomDocumentProperties.Add Name:="ToggleButton01", _
LinkToContent:=False, Type:=msoPropertyTypeNumber, Value:=0
End Sub
```

Nachdem Sie den Eintrag erstellt haben, fügen Sie die folgenden Makros in ein Standardmodul ein.

```
Option Private Module
Public objRibbon As IRibbonUI
Public Sub onload(ribbon As IRibbonUI)
Set objRibbon = ribbon
End Sub
Sub ToggleButton_getPressed(control As IRibbonControl, ByRef returnedValue)
returnedValue = ThisWorkbook.CustomDocumentProperties _
("ToggleButton01").Value
End Sub
Sub ToggleButton_OnAction(control As IRibbonControl, pressed As Boolean)
If ThisWorkbook.CustomDocumentProperties("ToggleButton01").Value = 0 Then
ActiveWorkbook.CustomDocumentProperties("ToggleButton01").Value = 1
objRibbon.Invalidate
MsgBox "Umschaltfläche 1 eingerastet", vbOKOnly + vbInformation, "Hinweis"
ThisWorkbook.Save
Else
ActiveWorkbook.CustomDocumentProperties("ToggleButton01").Value = 0
objRibbon.Invalidate
MsgBox "Umschaltfläche 1 ausgerastet", vbOKOnly + vbInformation, "Hinweis"
ThisWorkbook.Save
End If
End Sub
Sub ToggleButton_getLabel(control As IRibbonControl, ByRef label)
If ThisWorkbook.CustomDocumentProperties("ToggleButton01").Value = 1 Then
label = "Umschaltfläche 1 eingerastet"
Else
label = "Umschaltfläche 1 ausgerastet"
End If
End Sub
```

Speichern Sie jetzt die Änderungen und schließen die Arbeitsmappe. Öffnen Sie die Arbeitsmappe mit dem CustomUI-Editor und fügen Sie in das Codefenster folgenden Code ein.

```
<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui"
onLoad="onload">
<ribbon startFromScratch="true">
<tabs>
<tab id="tab01" label="Umschaltflächen">
<group id="grp01" label="Zustand in DocumentProperties" >
<toggleButton id="tgb01" getLabel="ToggleButton_getLabel"
imageMso="ExchangeFolder"
getPressed="ToggleButton_getPressed" onAction="ToggleButton_onAction"
size="large"/> </group>
 </tab>
  </tabs>
   </ribbon>
    </customUI>
```

Beschreibung:

id: Gibt die control.ID an

getLabel: Das Makro, welches die Beschriftung je nach Zustand der Schaltfläche angibt.

size: Größe der Schaltfläche. large=groß, normal=klein.

imageMso: Gibt das Icon an, welches genutzt werden soll.

getPressed: Fragt beim Öffnen der Arbeitsmappe den Wert für den Zustand aus dem gespeicherten Wert ab. Bei 1 wird die Schaltfläche gedrückt.

onAction: Das auszuführende Makro.

Sie können auch die Beispielmappe **XL07_ToggleButton.xlsm** nutzen (im Anhang).

RibbonX - Teil 4 – Checkboxen

In diesem Teil wollen wir eine Checkbox erstellen. Hierfür wird folgender RibbonX-Code verwendet. Der Beispielcode erstellt eine Checkbox.

Das folgende Beispiel erstellt eine Checkbox. Mit dieser wird geprüft, ob die aktive Tabelle geschützt ist.

Erstellen Sie eine neue Arbeitsmappe. Fügen Sie den folgenden VBA-Code in den Codebereich einer jeden Tabelle ein. Dieser Code ist notwendig, um den Schutzzustand beim Aktivieren des Tabellenblattes auslesen zu können.

```
Private Sub worksheet_activate()
On Error Resume Next
objRibbon.Invalidate
End Sub
```

Fügen Sie jetzt ein neues Modul ein und in dieses die folgenden Makros.

```
Option Private Module
Public objRibbon As IRibbonUI
Public Sub rx_onload(ribbon As IRibbonUI)
Set objRibbon = ribbon
End Sub
Public Sub Checkbox_onAction(control As IRibbonControl, pressed As Boolean)
If pressed = True Then
ActiveSheet.Protect
objRibbon.Invalidate
Else
ActiveSheet.Unprotect
objRibbon.Invalidate
End If
End Sub
Public Sub Checkbox_getLabel(control As IRibbonControl, ByRef label)
If ActiveSheet.ProtectContents = False Then
label = "Tabelle freigegeben"
Else
label = "Tabelle geschützt"
End If
End Sub
Public Sub Checkbox_getPressed(control As IRibbonControl, ByRef returnValue)
If ActiveSheet.ProtectContents = True Then returnValue = 1
End Sub
```

Anschließend speichern Sie die Arbeitsmappe und schließen sie. Öffnen Sie jetzt die Arbeitsmappe mit dem CustomUI-Editor und fügen folgenden Code in das Codefenster ein.

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

```
<tabs><tab id="tab01" label="Checkbox">
<group id="grpCBX" label="Checkbox" >
<checkBox id="cbx" getLabel="Checkbox_getLabel"
onAction="Checkbox_onAction" getPressed="Checkbox_getPressed" />
</group>
</tab></tabs></ribbon></customUI>
```

Speichern Sie nun die Änderung und schließen den Editor.

Öffnen Sie nun die Arbeitsmappe und testen Sie die Makros. Beim Öffnen wird nun geprüft, ob das aktive Tabellenblatt geschützt ist und setzt den Wert returnValue bei geschützter Tabelle auf 1. Bei 1 ist die Checkbox aktiv (Haken gesetzt. beim Wechsel in eine andere Tabelle wird die Prüfung (objRibbon.Invalidate) erneut durchgeführt.

Beschreibung:
id: Gibt die control.ID an

getPressed: Ruft beim Öffnen der Arbeitsmappe das Makro auf, welches den definierten Zustand abfragt.

onAction: Das aufzurufende Makro.

getlabel: Das Makro, welches die Beschriftung des Labels an den definierten Zustand anpasst.

Sie können auch die Beispielmappe **XL07_Checkbox.xlsm** nutzen (im Anhang).

RibbonX - Teil 5 - Auswahl- und Kombinationsfelder

In diesem Teil wollen wir ein Auswahlfeld und ein Kombinationsfeld erstellen. Hierfür wird folgender RibbonX-Code verwendet. Der Beispielcode erstellt zwei Auswahlfelder.
Mit diesen sollen verschiedene Einträge aus zwei Listen in ein "Rechnungsblatt" übertragen werden. Hierbei wird der jeweils erste Eintrag in das "Rechnungsblatt" übertragen. Die restlichen Angaben werden durch SVerweis übernommen.

Zuerst das Auswahlfeld (DropDown).
Erstellen Sie eine neue Arbeitsmappe. Fügen Sie ein neues Modul und in dieses die folgenden für das erste Auswahlfeld gültigen Makros ein.

```
Option Private Module
Public objRibbon As IRibbonUI
Public Sub rx_onload(ribbon As IRibbonUI)
Set objRibbon = ribbon
End Sub
Public Sub Artikel_getItemCount(control As IRibbonControl, _
ByRef returnedVal)
returnedVal = 200
End Sub
Public Sub Artikel_getItemID(control As IRibbonControl, _
index As Integer, ByRef id)
id = "Eintrag" & index + 3
End Sub
Public Sub Artikel_getItemLabel(control As IRibbonControl, _
index As Integer, ByRef returnedVal)
 returnedVal = ThisWorkbook.Sheets("Artikelliste").Cells(index + 3, 3).Value
End Sub
Public Sub Artikel_onAction(control As IRibbonControl, _
id As String, index As Integer)
Rem Trägt den gewählten Artikel in die Rechnung ein
ThisWorkbook.Sheets("Rechnung-Ausfüllen").Cells _
(Cells(1048576, 1).End(xlUp).Row + 1, 1).Value = _
ThisWorkbook.Sheets("Artikelliste").Cells _
(index + 3, 4).Offset(0, -3).Value
End Sub
```

Fügen Sie nun ein zweites Modul und in dieses die für das zweite Auswahlfeld gültigen Makros ein.

```
Option Private Module
Public Sub Kunde_getItemCount(control As IRibbonControl, _
ByRef returnedVal)
returnedVal = 200
End Sub
Public Sub Kunde_getItemID(control As IRibbonControl, _
index As Integer, ByRef id)
id = "Eintrag" & index + 3
End Sub
Public Sub Kunde_getItemLabel(control As IRibbonControl, _
index As Integer, ByRef returnedVal)
```

```
returnedVal = ThisWorkbook.Sheets("Kundendaten"). _
Cells(index + 3, 4).Value
End Sub
Public Sub Kunde_onAction(control As IRibbonControl, id As String, _
index As Integer)
Rem Trägt die Posi des Kunden in die Rechnung ein
With ThisWorkbook
.Sheets("Rechnung-Ausfüllen").Range("H2").Value = _
.Sheets("Kundendaten").Cells(index + 3, 4).Offset(0, -3).Value
End With
End Sub
```

Speichern Sie nun die Änderung und schließen die Arbeitsmappe. Öffnen Sie selbige mit dem CustomUI-Editor und fügen in das Codefenster folgenden Code ein.

```
<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui"
onLoad="rx_onLoad">
<ribbon startFromScratch="true">
<tabs><tab id="tab01" label="Rechnungserstellung">
<group id="grpCombo" label="Kundenwahl" >
<dropDown id="ddcKunde" label="Kunde:" screentip="Kunden auswählen"
supertip="Die Kundenkennung (Posi) auswählen" sizeString="xxxxxxxxxxxxxx"
onAction="Kunde_onAction" getItemCount="Kunde_getItemCount"
getItemID="Kunde_getItemID"
getItemLabel="Kunde_getItemLabel" >
</dropDown>
<dropDown id="ddcArtikel" label="Artikel:" screentip="Artikel auswählen"
supertip="Die Artikelkennung (Posi) auswählen" sizeString="xxxxxxxxxxxxxx"
onAction="Artikel_onAction" getItemCount="Artikel_getItemCount"
getItemID="Artikel_getItemID"
getItemLabel="Artikel_getItemLabel" >
</dropDown></group>
</tab></tabs></ribbon></customUI>
```

Speichern Sie die Änderung und schließen den Editor.

Beschreibung:

id: Gibt die control.ID an
label: Die Beschreibung des Elementes
getItemID: Die ID des Eintrages
getItemLabel: Der Eintrag
getItemCount: Die Anzahl der möglichen Einträge
onAction: Das auszuführende Makro

Jetzt kommen wir zum Kombinationsfeld. Anders als bei Auswahlfedern, bei denen nur feste Einträge gewählt werden können, kann man in einem Kombinationsfeld auch Eingaben tätigen. Kombinationsfelder kombinieren Auswahlfelder mit Eingabefeldern (editBox) in einem Element.

Erstellen Sie eine neue Arbeitsmappe. Fügen Sie ein neues Modul und in dieses die folgenden für das Kombinationsfeld gültigen Makros ein.

Die folgenden Codebeispiele stammen von *Melanie Breden* !

```
Option Explicit

Public objRibbon As IRibbonUI
Rem Callback for customUI.onLoad
Public Sub rx_onLoad(ribbon As IRibbonUI)
Set objRibbon = ribbon
End Sub

Rem Sub wird beim ersten Anklicken des Pfeils der ComboBox
Rem und dann bei jeder Neuinitialisierung ausgeführt

Rem Callback for cboCombo getItemCount
Public Sub cboCombo_getItemCount(control As IRibbonControl, ByRef returnedVal)
Rem Anzahl Einträge festlegen (Monate)
  returnedVal = 12
End Sub

Rem Sub wird nach Wert der Variable returnedVal aus getItemCount
Rem n mal durchlaufen
Public Sub cboCombo_getItemID(control As IRibbonControl, _
                        index As Integer, ByRef id)
Rem Eindeutiger Index je Eintrag festlegen
  id = &amp;quot;Month&amp;quot; &amp; index
End Sub

Rem Callback for cboCombo getItemLabel
Public Sub cboCombo_getItemLabel(control As IRibbonControl, _
                    index As Integer, ByRef returnedVal)
Rem Beschriftung je Eintrag festlegen
  returnedVal = MonthName(index + 1)
End Sub

Rem Callback für cboCombo onChange
Public Sub cboCombo_onChange(control As IRibbonControl, text As String)
Rem Auswahl aus Kombinationsfeld in aktive Zelle schreiben
  ActiveCell.Value = text
End Sub
```

Speichern Sie nun die Änderung und schließen die Arbeitsmappe. Öffnen Sie selbige mit dem CustomUI-Editor und fügen in das Codefenster folgenden Code ein.

```
<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui"
onLoad="rx_onLoad">
<ribbon><tabs>
<!--Benutzerdefinierte Registerkarte am Anfang erstellen-->
```

```
<tab id="tb1" label="Combobox" insertBeforeMso="TabHome">
<group id="grpCombo" label="Combobox">
<comboBox id="cboCombo" label="Monate:" screentip=
"Dynamisches Kombinationsfeld" supertip=
"Die Auswahl eines Eintrages wird in die aktive Zelle geschrieben"
onChange="cboCombo_onChange" getItemCount="cboCombo_getItemCount"
getItemID="cboCombo_getItemID" getItemLabel="cboCombo_getItemLabel">
</comboBox></group>
</tab></tabs>
</ribbon></customUI>
```

Beispielarbeitsmappen: **XL07_DropDown.xlsm** und **xl07_combobox_neu.xlsm (im Anhang)**

Auswahlfeld mit variabler Anzahl an Einträgen

Im Eingangsbeitrag haben Sie gelernt, wie man ein Auswahlfeld erstellt. In diesem Beispiel ist die Anzahl der Einträge auf 200 Einträge begrenzt. Mit einer kleinen Änderung können Sie die Anzahl an die Einträge der gewählten Spalte anpassen. Somit können Sie auf das Einfügen oder Löschen von Einträgen reagieren.
Ersetzen Sie einfach die bereits vorhandenen Makros Artikel_getItemCount und Kunde_getItemCount durch die beiden folgenden Makros.

```
Public Sub Artikel_getItemCount(control As IRibbonControl, ByRef returnedVal)
returnedVal = ThisWorkbook.Sheets("Artikelliste").Cells(Rows.Count, 3).End(xlUp).Row
End Sub

Public Sub Kunde_getItemCount(control As IRibbonControl, ByRef returnedVal)
returnedVal = ThisWorkbook.Sheets("Kundendaten").Cells(Rows.Count, 3).End(xlUp).Row
End Sub
```

Fügen Sie zusätzlich in den onAction-Ereignissen die folgende Zeile ein.

```
objRibbon.Invalidate
```

RibbonX - Teil 6 - Eingebaute Steuerelemente

Sie können in Ihren Ribbon auch die eingebauten Steuerelemente nutzen.
Eine Liste der Steuerelemente erhalten Sie bei Microsoft. Für jedes Office-2007-Programm gibt es eine eigene Liste als Excel-2007-Datei.

Wichtig:
Einige Elemente sind in den Listen falsch deklariert. Ein Beispiel ist der FormatPainter (Format übertragen). Entgegen den Angaben handelt es sich bei dieser Schaltfläche um ein Control (Kontrollschaltfläche) und nicht, wie angegeben, um einen ToggleButton.

2007OfficeControlIDsExcel2007.EXE

Mit dem folgenden Beispielcode fügen Sie den FormatPainter in Ihr Ribbbon ein. Ein VBA-Makro ist für eingebaute Steuerelemente nicht erforderlich.

```
<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui"
onLoad="onload">
<ribbon startFromScratch="true">
<officeMenu >
<button idMso="FileClose" visible="true" />
</officeMenu>
<tabs>
<tab id="tab01" label="FormatPainter" >
<group id="grp01" label="Format übertragen" >
<control idMso="FormatPainter" size="large"/>
</group>
 </tab>
  </tabs>
   </ribbon>
    </customUI>
```

RibbonX - Teil 7 - Eingabefeld (Editbox)

In diesem Teil bauen wir ein Eingabefeld (Editbox) ein. Im folgenden Beispiel wird das Eingabefeld genutzt, um über TAPI eine >>>Telefonnummer zu wählen. Zuerst wird die Nummer eingegeben. Ein Klick auf die Entertaste löst dann das Ereignis aus.

Erstellen Sie eine neue Arbeitsmappe. Fügen Sie ein neues Modul und in dieses den folgenden Makrocode ein.

```
Option Private Module
Option Explicit
Declare Function tapiRequestMakeCall Lib "Tapi32.dll" _
(ByVal DestAddress As String, _
ByVal AppName As String, ByVal CalledParty As String, _
ByVal Comment As String) As Long
Public A$
Private Declare Function CallWindowProc Lib "user32.dll" _
Alias "CallWindowProcA" _
(ByVal lpPrevWndFunc As Long, ByVal hWnd As Long, ByVal Msg As Long, _
ByVal wParam As Long, _
ByVal lParam As Long) As Long
Private Const WM_COMMAND As Long = &H111
Public objRibbon As IRibbonUI
Public Sub onload(ribbon As IRibbonUI)
Set objRibbon = ribbon
End Sub
Sub BoxWahlRaus(control As IRibbonControl, ByRef text)
Dim i As Integer
Dim Zelle As Integer
Dim A$
A$ = text
Telefonieren A, " "
End Sub
Sub Telefonieren(TelefonNr$, derName$)
Application.EnableCancelKey = False
Dim retval As Long
retval = tapiRequestMakeCall(TelefonNr, "", derName, "")
If retval <> 0 Then
MsgBox "Beim Verbindungsaufbau ist ein Fehler aufgetreten!"
End If
End Sub
```

Speichern und schließen Sie die Arbeitsmappe und öffnen diese dann mit dem CustomUI-Editor. Fügen Sie dann im Codefenster folgenden XML-Code ein.

```
<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui"
onLoad="onload">
<ribbon><tabs>
<tab id="tabExcelvorlage" label="Telefonieren">
```

```
<group id="groupTelefonieren" label="Telefonieren">
<editBox id="ebx_Schnellwahl" onChange="BoxWahlRaus"
supertip = "Telefonnummernschnellwahl. Nummer eingeben und
Entertaste drücken!"
sizeString="wwwwwwwwww" />
</group> </tab></tabs></ribbon></customUI>
```

Sie können auch die Beispielmappe **XL07_EditBox.xlsm** nutzen (im Anhang).

RibbonX - Teil 8 - Menü

In diesem Teil beschäftigen wir uns mit dem Menü. In einem Menü können Schaltflächen, Umschaltflächen und Checkboxen und Trennstriche (Menüseparator)eingebaut werden.
Legen Sie jetzt eine neue Arbeitsmappe an und speichern diese als *.xlsm.
Fügen Sie nun ein neues Modul und in dieses den folgenden Code ein.

```
Option Private Module
Public objRibbon As IRibbonUI
Public Sub onload(ribbon As IRibbonUI)
Set objRibbon = ribbon
End Sub
Sub Button01_onAction(control As IRibbonControl)
Range("A2").Value = Range("A2").Value + 1
MsgBox "Sie haben die Menüschaltfläche 1 gedrückt", _
vbInformation + vbOKOnly, "Hinweis"
End Sub
Sub Button02_onAction(control As IRibbonControl)
Range("B2").Value = Range("B2").Value + 1
MsgBox "Sie haben die Menüschaltfläche 2 gedrückt", _
vbInformation + vbOKOnly, "Hinweis"
End Sub
Sub Checkbox01_onAction(control As IRibbonControl, pressed As Boolean)
If pressed = True Then
MsgBox "Checkbox 1 aktiviert", vbInformation + vbOKOnly, "Hinweis"
Range("C2").Value = 1
Else
MsgBox "Checkbox 1 deaktiviert", vbInformation + vbOKOnly
Range("C2").Value = 0
End If
End Sub
Sub Checkbox01_getPressed(control As IRibbonControl, ByRef returnValue)
returnValue = ThisWorkbook.Sheets("Tabelle1").Range("C2")
End Sub
Sub ToggleButton01_onAction(control As IRibbonControl, pressed As Boolean)
If pressed = True Then
MsgBox "Umschaltlfäche 1 aktiviert", vbInformation + vbOKOnly, "Hinweis"
Range("D2").Value = 1
Else
MsgBox "Umschaltfläche 1 deaktiviert", vbInformation + vbOKOnly
Range("D2").Value = 0
End If
End Sub
Sub ToggleButton01_getPressed(control As IRibbonControl, ByRef returnValue)
returnValue = ThisWorkbook.Sheets("Tabelle1").Range("D2")
End Sub
Sub Checkbox02_onAction(control As IRibbonControl, pressed As Boolean)
If pressed = True Then
MsgBox "Checkbox 2 aktiviert", vbInformation + vbOKOnly, "Hinweis"
Range("E2").Value = 1
Else
MsgBox "Checkbox 2 deaktiviert", vbInformation + vbOKOnly
```

```
Range("E2").Value = 0
End If
End Sub
Sub Checkbox02_getPressed(control As IRibbonControl, ByRef returnValue)
returnValue = ThisWorkbook.Sheets("Tabelle1").Range("E2")
End Sub
Sub ToggleButton02_onAction(control As IRibbonControl, pressed As Boolean)
If pressed = True Then
MsgBox "Umschaltlfäche 2 aktiviert", vbInformation + vbOKOnly, "Hinweis"
Range("F2").Value = 1
Else
MsgBox "Umschaltfläche 2 deaktiviert", vbInformation + vbOKOnly
Range("F2").Value = 0
End If
End Sub
Sub ToggleButton02_getPressed(control As IRibbonControl, ByRef returnValue)
returnValue = ThisWorkbook.Sheets("Tabelle1").Range("F2")
End Sub
```

Speichern Sie die Änderungen und schließen Sie die Arbeitsmappe. Öffnen Sie selbige jetzt mit dem CustomUI-Editor. Fügen Sie dann den folgenden Code in das Codefenster ein.

```
<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui"
onLoad="onLoad" >
<ribbon startFromScratch="true">
<tabs><tab id="tab01" label="Menü">
<group id="grp01" label="Menü">
<menu id="mnu0" size="large" itemSize="normal" label="Menü"
imageMso="FileSaveAsExcelXlsxMacro">
<button id="menBtn01" label="Menüschaltfläche 1"
 onAction="Button01_OnAction" />
<menuSeparator id="menSpt01" />
<toggleButton id="menTgb01" label="Umschaltfläche 1"
onAction="ToggleButton01_OnAction"
getPressed="ToggleButton01_getPressed"/>
<menuSeparator id="menSpt02" />
<menu id="men02" itemSize="normal" label="Untermenü 1">
<button id="menBtn02" label="Menüschaltfläche 2"
 onAction="Button02_OnAction" />
<menuSeparator id="menSpt03" />
<toggleButton id="menTgb02" label="Umschaltfläche 2"
 onAction="ToggleButton02_OnAction"
getPressed="ToggleButton02_getPressed" />
<menuSeparator id="menSpt04" />
<checkBox id="menCbx02" label="Checkbox 2" onAction="Checkbox02_OnAction"
getPressed="Checkbox02_getPressed"/>
</menu>
<menuSeparator id="menSpt05" />
<checkBox id="menCbx01" label="Checkbox 1" onAction="Checkbox01_OnAction"
getPressed="Checkbox01_getPressed" />
</menu></group></tab>
</tabs></ribbon></customUI>
```

Speichern sie die Änderung und öffnen Sie die Arbeitsmappe. Jetzt wird ein neues Ribbon mit einem Menü angezeigt. Die Werte für die Checkboxen und Umschaltflächen (jeweils 2 Stück) werden in der Tabelle gespeichert. Idealer Weise sollten Sie die Werte in den DocumentProperties speichern. Gehen Sie hierzu so vor, wie Sie es in Teil 3 gelernt haben. Für jede Checkbox und jede Umschaltfläche benötigen Sie einen eigenen Eintrag in den DocumentProperties.

Sie können auch die Beispielmappe **XL07_Menü.xlsm** nutzen (im Anhang).

Zum Inhaltsverzeichnis !

RibbonX - Teil 9 - SplitButton

In diesem Teil lernen wir den Splitbutton kennen. Dieses Element wird selten genutzt. SplitButton können genutzt werden, wenn nicht genügend Platz auf dem Ribbon vorhanden ist.

Ein Spiltbutton ist in zwei Teile aufgeteilt. Dem Button und dem Menü. Dem Button wird dabei das aus dem Splitbutton-Menü zuletzt genutzte Makro zugewiesen. Normalerweise ist diese Zuweisung temporär, geht also beim Schließen der Arbeitsmappe verloren. Damit dies nicht geschieht, nutzen wir wieder die DocumentProperties. Zu diesem Zweck wird ein neuer Eintrag in den DocumentProperties angelgt. Anders als bei Umschaltflächen muss der Eintrag in diesem Fall vom Typ String sein. Für jeden Splitbutton benötigen Sie einen eigenen Eintrag.

Erstellen Sie eine neue Arbeitsmappe (*.xslm). Fügen Sie ein neues Modul und in dieses das folgende Makro ein. Führen Sie selbiges einmalig aus. Danach können Sie es wieder löschen.

```
Sub DocumentProbertieFürToggleButtonAnlegen()
ActiveWorkbook.CustomDocumentProperties.Add Name:="SplitButton01", _
LinkToContent:=False, Type:=msoPropertyTypeString, Value:="Spalte A"
End Sub
```

Fügen Sie anschließend die folgenden Makros ein:

```
Option Explicit
Dim objRibbon As IRibbonUI
Dim strLabel As String
Public Sub rx_onLoad(ribbon As IRibbonUI)
  Set objRibbon = ribbon
End Sub
Public Sub SplitButton_getLabel(control As IRibbonControl, ByRef label)
If strLabel = "" Then strLabel = _
ThisWorkbook.CustomDocumentProperties("SplitButton01").Value
label = strLabel
End Sub
Public Sub SplitButton_onAction(control As IRibbonControl)
 SplitMacro strLabel
End Sub
Public Sub SplitMenüButton_onAction(control As IRibbonControl)
Select Case control.ID
Case "btn01"
strLabel = "Spalte A"
ThisWorkbook.CustomDocumentProperties("SplitButton01").Value = "Spalte A"
SplitMacro strLabel
Case "btn02"
strLabel = "Spalte B"
ThisWorkbook.CustomDocumentProperties("SplitButton01").Value = "Spalte B"
SplitMacro strLabel
Case "btn03"
strLabel = "Spalte C"
ThisWorkbook.CustomDocumentProperties("SplitButton01").Value = "Spalte C"
SplitMacro strLabel
End Select
objRibbon.Invalidate
```

```
End Sub
Private Sub SplitMacro(ByVal label As String)
  Select Case strLabel
    Case "Spalte A"
      If Columns("A:A").EntireColumn.Hidden = False Then
        Columns("A:A").EntireColumn.Hidden = True
        Else
        Columns("A:A").EntireColumn.Hidden = False
        End If
    Case "Spalte B"
      If Columns("B:B").EntireColumn.Hidden = False Then
        Columns("B:B").EntireColumn.Hidden = True
        Else
        Columns("B:B").EntireColumn.Hidden = False
        End If
    Case "Spalte C"
      If Columns("C:C").EntireColumn.Hidden = False Then
        Columns("C:C").EntireColumn.Hidden = True
        Else
        Columns("C:C").EntireColumn.Hidden = False
        End If
  End Select
End Sub
```

Speichern und schließen Sie jetzt die Arbeitsmappe und öffnen Sie sie mit dem CustomUI-Editor. Fügen Sie dann den folgenden Code in das Codefenster ein.

```
<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui"
 onLoad="rx_onLoad">
<ribbon startFromScratch="true">
<tabs>
<tab id="tab01" label="Mein Tab">
<group id="grpDemo" label="SplitButton Demo">
<splitButton id="spbImage" size="large"  >
<button id="btnImage" getLabel="SplitButton_getLabel"
onAction="SplitButton_onAction"
imageMso="TableSharePointListsModifyColumnsAndSettings"/>
<menu id="menSbt01">
<button id="btn01" label="Spalte A" onAction="SplitMenüButton_onAction"/>
<button id="btn02" label="Spalte B" onAction="SplitMenüButton_onAction"/>
<button id="btn03" label="Spalte C" onAction="SplitMenüButton_onAction"/>
</menu>
 </splitButton>
  </group>
   </tab>
    </tabs>
     </ribbon>
      </customUI>
```

Die Codebeispiele in diesem Beitrag stammen von *Melanie Breden* !

Beispielarbeitsmappe **xl07_splitbutton_neu.xlsm**

RibbonX - Teil 10 - Command, QAT, Office Menü anpassen

In diesem Beitrag lernen wir, wie man auf Command Ebene Befehle deaktiviert und die Schnellstartleiste und das Office Menü dokumentabhängig anpasst.

Beginnen wir mit der Command Ebene. Diese muss zwingend vor dem Ribbon initialisiert werden.
In diesem Beispiel werden folgende Befehle deaktiviert:
Excel-Optionen, Beenden-Schaltfläche.
Schließkreuz, Minimieren, Wiederherstellen der Tabelle und der Hilfebutton.
Die Tastenkombination ALT+F4 wird nicht deaktiviert, die Anwendung kann noch immer über selbige geschlossen werden.

```
<commands>
   <!-- Deaktiviert die Hilfe -->
   <command idMso="Help" enabled="false"/>
   <!-- Deaktiviert das Schließkreuz -->
   <command idMso="WindowClose" enabled="false"/>
   <!-- Deaktiviert "Wiederherstellen" -->
   <command idMso="WindowRestore" enabled="false"/>
   <!-- Deaktiviert "Minimieren" -->
   <command idMso="WindowMinimize" enabled="false"/>
   <!-- Deaktiviert das Optionsmenü -->
   <command idMso="ApplicationOptionsDialog" enabled="false"/>
   <!-- Deaktiviert die Beenden-Schaltfläche -->
   <command idMso="FileExit" enabled="false" />
</commands> <!-- Ende Command-Ebene -->
```

Als Nächstes fügen wir Elemente der Schnellstartleiste hinzu. Kann nur genutzt werden, wenn startFromScratch auf True gesetzt. Die QAT muss noch vor den Tabs und vor dem Office Menü initialisiert werden. In diesem Beispiel werden hinzugefügt:
Speichern, Rückgängig, Wiederholen und Neue Datei.

```
<!-- Schnellstartleiste - Quick Access Toolbar (QAT)
    Hier können Sie verschiedene eingebaute Steuer-
    elemente der QAT zuweisen
    Hat nur Wirkung bei "startFromScratch="true"" -->
<qat><documentControls>
<!-- Fügt "Speichern" hinzu -->
<control idMso="FileSave" imageMso="FileSave"/>
<!-- Fügt "Rückgängig" ein -->
<control idMso="Undo" imageMso="Undo"/>
<!-- Fügt "Wiederholen" ein -->
<control idMso="Redo" imageMso="Redo"/>
<!-- Fügt "Neue Datei" hinzu -->
<control idMso="FileNewDefault" imageMso="FileNewDefault"/>
</documentControls></qat><!-- Ende QAT -->
```

Zum Schluss widmen wir uns dem Office Menü. Dieses muss noch vor den Tabs und nach der QAT initialisiert werden. Diese Einstellung hat nur Wirkung, wenn die Office Menü-Ebene im XML-Code eingebaut ist. Ist sie nicht eingebaut und startFromScratch auf True gesetzt, dann werden bis auf drei

Einträge alle anderen Menüeinträge entfernt. Steht startFromScratch auf False, werden beim Fehlen der OfficeMenü-Ebene alle Einträge im Office-Menü angezeigt.
In diesem Beispiel werden folgende Einstellungen für das Office Menü vorgenommen.
Schließen->aktiviert. Neu->deaktiviert. Datei öffnen->deaktiviert.
Speichern->deaktiviert. Drucken->aktiviert.
Speichern unter->aktiviert. Senden->aktiviert.

```
<!-- Officemenü. Hier können Sie dem Menü
    standardmäßig inaktive Elemente hinzufügen oder
    standardmäßig aktive Elemente deaktivieren. -->
<officeMenu >
<!-- Fügt "Schließen" hinzu (Standardmäßig inaktiv) -->
<button idMso="FileClose" visible="true" />
<!-- Deaktiviert "Neu" (Standardmäßig aktiv) -->
<button idMso="FileNew" visible="false" />
<!-- Deaktiviert "Datei öffnen" (Standardmäßig aktiv) -->
<button idMso="FileOpen" visible="false" />
<!-- Deaktiviert "Speichern" (Standardmäßig aktiv) -->
<button idMso="FileSave" visible="false" />
<!-- Aktiviert "Drucken" (Standardmäßig inaktiv) -->
<splitButton idMso="FilePrintMenu" visible="true" />
<!-- Aktiviert "Speichern unter" (Standardmäßig inaktiv) -->
<splitButton idMso="FileSaveAsMenu" visible="true" />
<!-- Aktiviert "Senden an" (Standardmäßig inaktiv)  -->
<menu idMso="FileSendMenu" visible="true"></menu> </officeMenu>
<!-- Ende Officemenü -->
```

Den gesamten RibbonX-Code finden Sie in der Beispielmappe **Commands.xlsm (im Anhang)** .

RibbonX - Teil 11 - Dialogbox-Launcher

In diesem Teil stelle ich kurz den DialogboxLauncher vor. Das ist die kleine Schaltfläche in der unteren rechten Ecke einer Gruppe.

Pro Gruppe darf nur ein DialogboxLauncher eingefügt werden. Der DBL muss zwingend an das Ende der Gruppe. Über das OnAction-Ereignis können Sie ein Makro auslösen, z.B. eine Infobox oder eine Userform aufrufen.

```
<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui" >
<ribbon>
<tabs>
<tab id="tab01" label="DialogBoxLauncher">
<group id="grp01" label="Launcher">
<button id="btn01" label="Schaltfläche 1" onAction="Button01_onAction"
screentip="Schaltfläche 1" supertip="Schaltfläche 1. Zum Testen anklicken"/>
<dialogBoxLauncher>
<button id="myLauncher1" supertip = "Impressum"
onAction="BildschirmAnzeigen" />
</dialogBoxLauncher>
</group> </tab></tabs></ribbon></customUI>
```

RibbonX - Teil 12 - Elemente gruppieren

In diesem Teil lernen wir, wie man Schaltflächen gruppieren kann. Ein Beispiel dafür ist die Funktion "Textausrichtung" (Tab Home->Gruppe Ausrichtung).

Zum Testen können Sie den folgenden VBA-Code verwenden, den Sie in ein Modul einer Arbeitsmappe speichern.

```
Sub Button_onAction(control As IRibbonControl)
MsgBox control.ID
End Sub
```

Den RibbonX-Code fügen Sie über den CustomUI-Editor ein.

```
<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui">
<ribbon startFromScratch="true">
<tabs>
<tab id="tab01" label="ButtonGroup">
<group id="grp01" label="ButtonGroup">
<buttonGroup id="btn_grp_01">
<button id="button01" label="01" onAction="Button_onAction" />
<button id="button02" label="02" onAction="Button_onAction" /></buttonGroup>
<buttonGroup id="btn_grp_2">
<button id="button03" label="03" onAction="Button_onAction" />
<button id="button04" label="04"  onAction="Button_onAction" /></buttonGroup>
<buttonGroup id="btn_grp_3">
<button id="button05" label="05" onAction="Button_onAction" />
<button id="button06" label="06" onAction="Button_onAction" /></buttonGroup>
</group></tab></tabs></ribbon></customUI>
```

RibbonX - Teil 13 - Kontext-Registerkarten

In diesem Teil lernen wir, wie man Kontext-Registerkarten deaktivieren kann. Kontext-Registerkarten sind Tabs, welche nur unter bestimmten Bedingungen erscheinen. Dazu gehört zum Beispiel die Registerkarte Diagrammtools.

Um eine Kontextregisterkarte auszublenden, benutzt man folgenden RibbonX-Code. Dieser muss vor den Tabs, aber nach dem Office Menü deklariert werden.

```
<contextualTabs>
  <tabSet idMso="TabSetChartTools" >
    <tab idMso="TabChartToolsFormat" visible="false"></tab>
  </tabSet>
</contextualTabs>
```

Sollen nur bestimmte Tabs der Kontext-Register ausgeblendet werden, benutzt man folgenden Code. In diesem Beispiel wird das Tab Format der Diagrammtools ausgelendet.

```
<contextualTabs>
  <tabSet idMso="TabSetChartTools" >
    <tab idMso="TabChartToolsFormat" visible="false"></tab>
  </tabSet>
</contextualTabs>
```

Sollen nur bestimmte Gruppen eines Tabs deaktiviert werden, benutzt man folgenden Code. In diesem Beispiel wird die Gruppe Anordnen des Tabs Format ausgeblendet.

```
<contextualTabs>
  <tabSet idMso="TabSetChartTools" >
    <tab idMso="TabChartToolsFormat">
      <group idMso="GroupArrange" visible="false">
          </group>
    </tab>
  </tabSet>
</contextualTabs>
```

Die Namen der Tabs und Gruppen finden Sie in der bereits erwähnten Übersicht ExcelRibbonControls (für die anderen Office Programme nehmen Sie die entsprechende Übersicht).

RibbonX - Teil 14 - Vorhandene Tabs anpassen

In diesem Teil lernen wir, wie man in vorhandene Tabs eigene Gruppen einbindet.

Es ist nicht möglich, vorhandene Tabs oder Gruppen zu beeinflussen. Sie können aber in vorhandene Tabs eigene Gruppen einfügen. Im folgenden Beispiel wird in die Registerkarte Entwicklertools eine neue Gruppe mit einer (großen) Schaltfläche eingefügt.

```
<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui"
onLoad="onLoad">
<ribbon>
<tabs>
<tab idMso="TabDeveloper">
<group id="grpMeineGruppe" label="Meine Gruppe">
<button id="btn01" label="Mein Button" size="large"
imageMso="_1" onAction="MeinMakro"/>
</group>
 </tab>
  </tabs>
  </ribbon>
    </customUI>
```

Die Namen der Tabs finden Sie in den bereits erwähnten Übersichten.

RibbonX - Teil 15 - Quickinfo anpassen

In diesem Teil lernen wir, wie man die Quickinfo anpasst. Voraussetzung für die Anzeige der Featurebeschreibung ist, dass sie aktiviert ist.

So aktivieren Sie die Featurebeschreibung:

Officebutton->Excel-Optionen->Häufig verwendet. Hier bei Quickinfo-Format wählen Sie Featurebeschreibungen in Quickinfo anzeigen.

Der untere Teil der Quickinfo ist Standard und kann nicht geändert werden.

Es gibt zwei Attribute für die Featurebeschreibung.

1. screentip: Dieser sollte so kurz als möglich sein. Wird er weggelassen, dann wird hier der Name der Schaltfläche angezeigt.

2. supertip. Dieser darf lang sein.

Beispielcode:

```
<button id="btn123" label="Telefonverzeichnis."
ScreenTip = "Telefonverzeichnis laden"
supertip = "Laden Sie ein selbsterstelltes Telefonverzeichnis."
onAction="VerzeichnisÖffnen"/>
```

Es besteht auch die Möglichkeit, die Texte variabel zu halten. Hierzu ist es notwendig, das Ribbon nach Betätigen der Schaltfläche zu aktualisieren.

Dieser Code gehört in ein Standardmodul:

```
Option Private Module
Option Explicit
Public objRibbon As IRibbonUI
Public Sub onload(ribbon As IRibbonUI)
Set objRibbon = ribbon
End Sub
Sub Button1_onAction(control As IRibbonControl, pressed As Boolean)
If pressed = True Then
Range("A1").Value = "eingeschaltet"
Else
Range("A1").Value = "ausgeschaltet"
End If
objRibbon.Invalidate
End Sub
Sub Button1_getScreentip(control As IRibbonControl, ByRef text)
If Range("A1").Value = "eingeschaltet" Then
text = "Aktiviert"
Else
text = "Deaktiviert"
End If
End Sub
```

```
Sub Button1_getSupertip(control As IRibbonControl, ByRef text)
If Range("A1").Value = "eingeschaltet" Then
text = "Sie haben die Schaltfläche aktiviert"
Else
text = "Sie haben die Schaltfläche deaktiviert"
End If
End Sub
Sub Button1_getLabel(control As IRibbonControl, ByRef label)
If Range("A1").Value = "eingeschaltet" Then
label = "Aktiviert"
Else
label = "Deaktiviert"
End If
End Sub
Sub Button1_getPressed(control As IRibbonControl, ByRef returnedVal)
If Range("A1").Value = "eingeschaltet" Then
returnedVal = 1
Else
returnedVal = 0
End If
End Sub
```

Anschließend die Datei speichern und mit dem CustomUi-Editor öffnen. Dann den folgenden Code einfügen.

```
<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui"
onLoad="onload">
<ribbon>
<tabs>
<tab id="Quickinfo" label="Quickinfo">
<group id="grp01" label="Quickinfo">
<toggleButton id="tgb01" getLabel="Button1_getLabel"
getScreentip = "Button1_getScreentip"
onAction="Button1_onAction" getPressed="Button1_getpressed"
getSupertip="Button1_getSupertip" imageMso="AutoDial" size="large"/>
</group> </tab></tabs></ribbon></customUI>
```

Beispielmappe **quickinfo_ribbonX.xlsm (im Anhang)**

Zum Inhaltsverzeichnis !

RibbonX - Teil 16 - Galerie

In diesem Teil lernen wir die Galerie kennen. Die Galerie ist eine Art Bildervorschau. Den Items in einer Gallery werden Bilder (Icons) zugewiesen. Die Auswertung der Auswahl erfolgt über selectedId. Ein Beispiel dafür ist die OfficeControls.xlam, in welchem die in Office 2007 möglichen Icons dargestellt werden und welches Sie im Teil 1 dieses Workshops herunterladen können.

Erstellen Sie zuerst eine neue Datei mit Makros (*.dotm, *.docm, *.xlsm usw.). Fügen Sie dann ein neues Modul ein, in welches Sie das folgende Makro einfügen.

```
Public Sub onAction(control As IRibbonControl, _
selectedId As String, selectedIndex As Integer)
Select Case selectedId
Case "itm1"
MsgBox "Schaltfläche ""Telefon"" gedrückt"
Case "itm2"
MsgBox "Schaltfäche ""Speichern"" gedrückt"
End Select
End Sub
```

Speichern Sie die Änderung, schließen die Datei und öffnen Sie es dann mit dem CustomUI-Editor. Fügen Sie dort den folgenden Code ein.

```
<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui">
<ribbon>
<tabs>
<tab id="tabGallery" label="Gallery">
<group id="grp" label="Gallerie">
<gallery id="gal" tag="large" label="Galerie" size="large"
imageMso="VisualBasic"
showItemLabel="true" itemWidth="64" itemHeight="64" onAction="OnAction">
<item id="itm1" imageMso="AutoDial" label="Telefon" />
<item id="itm2" imageMso="FileSave" label="Speichern"/>
</gallery>
</group>
</tab>
</tabs>
</ribbon>
</customUI>
```

Speichern und schließen Sie nun die Datei. Jetzt sollte die Galerie angezeigt werden.

Erklärung:
showItemLabel: Hier können Sie festlegen, ob das Label angezeigt werden soll.
onAction: Das auszuführende Makro
item id: Hier wird die ID zur Identifikation angegeben
image/imageMso: Hier wird das Bild/Name des integrierten Icons angegeben
label: Bezeichnung für showItemLabel
itemWidth/itemHeight: Größe des Elementes. Empfohlen werden jeweils 32.
columns/rows: Anzahl Zeilen/Spalten. Angabe ist optional.
size: Größe der Schaltfläche (normal = klein, large = groß)

Sie können auch die Beispieldatei **Galery.xlsm (im Anhang)** nutzen.

Zum Inhaltsverzeichnis !

RibbonX - Teil 17 - RibbonX in Access

In diesem Teil lernen wir, wie man in Access 2007 die Multifunktionsleiste beeinflussen kann.

Anders als in Excel, Word und PowerPoint kann man in Access-Datenbanken RibbonX-Code nicht mit dem CustomUI-Editor erstellen, da es sich bei Datenbanken nicht um XML-Dateien handelt. Bei Access wird der Code stattdessen direkt in die Datenbank geschrieben. Hierbei wird eine neue Tabelle mit dem Namen USysRibbon erstellt, in welche der RibbonX-Code geschrieben wird. Eine gute Anleitung finden Sie unter Access Ribbon - Gunter Avenius mit vielen Beispieldatenbanken.
In Access können Sie eigene Ribbon ganz komfortabel mit dem Access Ribbon Creator (Version 1.1019) erstellen und direkt in die Datenbank schreiben. Der Ribbon Creator beruht auf WYSIWYG, sie sehen also sofort, wie das Ribbon aussehen wird. In einigen Fällen kann jedoch eine Nachbearbeitung des RibbonX-Codes erforderlich werden. Diese können Sie direkt in der Datenbank vornehmen. Sie können den RibbonX-Code natürlich auch in einem Texteditor schreiben und anschließend in die Datenbank schreiben. Hierzu können Sie die Beispielcodes aus den vorangegangenen RibbonX-Beiträgen nutzen.

Hinweis:

Sie können den Ribbon Creator auch für Excel und Word einsetzen.
Der Ribbon Creator nutzt für gleiche Elemente nur ein Callback. Hierbei wird Select Case eingesetzt. Wenn Sie für einzelne Elemente eigene Callbacks nutzen möchten, dann müssen Sie den RibbonX-Code anschließend über den CustomUI-Editor nachbearbeiten. Für Access-Datenbanken nehmen Sie die Anpassung stattdessen direkt in der Datenbank (Tabelle USysRibbon) vor.

Für Excel- und Worddateien kann es außerdem erforderlich werden, einige Callbackaufrufe aus der customUI zu entfernen. Hierzu gehört zum Beispiel LoadImage, da dieses in Excel und Word nicht zwingend ist. Für Excel und Word ist es daher empfehlenswert, den RibbonX-Code nicht direkt in die Datei zu schreiben. Speichern Sie daher den RibbonX-Code zuerst in eine XML-Datei. Anschließend kopieren Sie den Code aus der XML-Datei in die Excel- bzw. Worddatei. Sie können dann auf Wunsch den Code mit dem CustomUI-Editor nachbearbeiten.

Zum Inhaltsverzeichnis !

RibbonX - Teil 18 - individuelle Schnellzugriffleiste

Normalerweise wird die Schnellzugriffleiste über das Office Programm angepasst. Dies hat jedoch den Nachteil, dass die Schaltflächensymbole (Icons) der integrierten Schaltflächen nicht geändert werden können. Schaltflächensymbole (Icons) für benutzerdefinierte Makroschaltflächen sind im Umfang reduziert, es steht also nur eine begrenzte Anzahl der eigentlich möglichen Schaltflächensymbole (Icons) zur Verfügung. Möchte man andere Schaltflächensymbole nutzen, muss man die userCustomization extern manipulieren. Wie man das macht, lernen wir hier.

Zuerst wird der Unterschied zwischen der QAT-Ebene in RibbonX und der userCustomization erklärt.

Über die QAT-Ebene in RibbonX lässt sich die Schnellstartleiste beeinflussen. Dies funktioniert jedoch nur, wenn startFromScratch auf True gesetzt ist. Wenn Sie dieses vergessen, wird der RibbonX-Code nicht ausgeführt. In RibbonX ist es nicht möglich, eigene Schaltflächen mit benutzerdefinierten Makros in die Schnellzugriffleiste zu integrieren. Sie können lediglich integrierte Schaltflächen aktivieren oder deaktivieren oder zusätzliche integrierte Schaltflächen zufügen (sharedControls). Hierzu muss aber startFromScratch auf True gesetzt werden. Das hat jedoch zur Folge, dass die Standard Ribbon deaktiviert werden. Daher empfiehlt es sich, eigene Makroschaltflächen über die userCustomization in die Schnellzugriffleiste zu integrieren.

Mit Hilfe der userCostumization ist es möglich, eigene Schaltflächen mit benutzerdefinierten Makros in die Schnellzugriffleiste zu integrieren. Die über die userCostumization erstellte Schnellzugriffleiste kann nicht durch RibbonX deaktiviert werden, da es sich hierbei um eine eigenständige Symbolleiste innerhalb der Schnellzugriffleiste handelt. Vorteil hierbei ist, dass die Standard Ribbon aktiviert bleiben.

Weder in RibbonX noch in der userCustomization können eigene Icons genutzt werden, das image-Tag steht nicht zur Verfügung. Bei fehlerhaften Einträgen in RibbonX und der userCustomization werden die Codes nicht ausgeführt.

Um eine eigene userCustomiation zu erstellen gehen Sie wie folgt vor. Da die xml-datei der userCustomization in der Regel denselben Namen trägt wie die RibbonX-Datei (customUI), habe ich für die userCustomization den Namen icontoolbar gewählt. Die erfordert jedoch, dass diese in der .rels-Datei berücksichtigt wird.

Erstellen Sie eine neue Datei und speichern diese ab, zum Beispiel als *.xlam (Excel 2007-AddIn) oder für Word als Vorlage (*.dotm, Vorlage mit Makros).

Erstellen Sie zuerst einen Ordner mit dem Namen userCustomization. Achten Sie dabei auf die korrekte Schreibweise wie hier angegeben. Erstellen Sie nun mit Notepad eine neue Textdatei, geben Sie dieser den Namen icontoolbar und speichern Sie diese mit der Endung .xml im zuvor erstellten Ordner ab. Kopieren Sie den folgenden Code nun in der icontoolbar.xml, speichern die Änderung und schließen Sie zum Abschluss die Datei.

```
<mso:customUI xmlns:doc="http://schemas.microsoft.com/office/2006/01/customui/
currentDocument"
xmlns:mso="http://schemas.microsoft.com/office/2006/01/customui">
<mso:ribbon><mso:qat>
<mso:documentControls>
<mso:button idMso="FileSaveAs" visible="true" imageMso="FileSaveAs"/>
<mso:button id="FS" visible="true" label="Ruft den Oeffnen-Dialog auf"
imageMso="FileOpen" onAction="Oeffnen"/>
</mso:documentControls></mso:qat></mso:ribbon></mso:customUI>
```

Sie können nun den Code entsprechend anpassen und um eigene Schaltflächen erweitern. Achten Sie darauf, dass Sie die Grundstruktur des Codes nicht ändern. Der Beispielcode enthält jeweils eine Standardschaltfläche und eine benutzerdefinierte Schaltfläche mit eigenem Makro. Das Makro befindet sich dabei in einem Modul der Datei.

Speichern Sie nun die Wordvorlage im Word 2007 Startup-Ordner. Das Excel-AddIn binden Sie wie gewohnt über den Excel 2007 AddIn-Manager ein.

Wichtig:
Die über userCustomization erstellte Schnellzugriffleiste hat Vorrang vor der QAT-Ebene in RibbonX. Die über die userCustomization zur Schnellzugriffleiste hinzugefügten Elemente können nicht über RibbonX deaktiviert werden, sofern die Vorlage (bei Word) im Startup-Ordner gespeichert wurde oder die Elemente über AddIn (bei Excel) eingebunden wurden. Dasselbe gilt, wenn Ihre Datei sowohl RibbonX als auch userCustomization enthält.

Öffnen Sie nun mit Winzip die zuvor Erstellte Officedatei. Kopieren Sie dann den Ordner in den Datei Container. Öffnen Sie jetzt mit Wordpad die Datei .rels aus dem Datei Container heraus und ersetzen Sie den darin enthaltenen Text durch folgenden Code.

```
<?xml version="1.0" encoding="utf-8"?>
<Relationships xmlns="http://schemas.openxmlformats.org/package/2006/
relationships">
<Relationship Type="http://schemas.openxmlformats.org/package/2006/
relationships/metadata/core-properties"
Target="docProps/core.xml" Id="rId3" />
<Relationship Type="http://schemas.microsoft.com/office/2006/
relationships/ui/userCustomization"
Target="userCustomization/icontoolbar.xml" Id="rId2" />
<Relationship Type="http://schemas.openxmlformats.org/officeDocument/2006/
relationships/officeDocument"
Target="word/document.xml" Id="rId1" />
<Relationship Type="http://schemas.openxmlformats.org/officeDocument/2006/
relationships/extended-properties"
Target="docProps/app.xml" Id="rId4" />
</Relationships>
```

Sie können auch die Beispieldatei herunter laden, und somit die Beispielcodes über Winzip anschauen.
Wenn Sie den Aufwand scheuen und trotzdem andere Schaltflächensymbole (Icons) haben möchten, dann haben Sie auch die Möglichkeit, die entsprechende QAT-Datei zu ändern. Suchen Sie nach der entsprechenden Datei (Word.qat, Excel.qat) und öffnen Sie diese mit Wordpad. Der Code entspricht nahezu dem oben genannten. Auch in den QAT-Dateien können keine eigenen Icons genutzt werden, das image-Tag steht nicht zur Verfügung. Es können nur integrierte Schaltflächensymbole genutzt werden (imageMso).

Um den Namen der Schaltflächensymbole zu erhalten, laden Sie sich die Officecontrol.xlam herunter. Die Namen der Schaltflächen entnehmen Sie den entsprechenden Tabellen. Zum herunterladen der Dateien gehen Sie zum Beitrag **RibbonX – Teil 1 – Einführung**.

Der vorgenannte Code für die .rels kann nur in Word verwendet werden. Für Excel nutzen Sie bitte folgenden XML-Code

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Relationships xmlns="http://schemas.openxmlformats.org/package/2006/relationships">
<Relationship Id="rId7"
Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/extended-properties"
Target="docProps/app.xml"/>
<Relationship Id="rId2"
Type="http://schemas.microsoft.com/office/2006/relationships/ui/userCustomization"
Target="userCustomization/icontoolbar.xml"/>
<Relationship Id="rId1"
Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/officeDocument"
Target="xl/workbook.xml"/>
<Relationship Id="rId6"
Type="http://schemas.openxmlformats.org/package/2006/relationships/metadata/core-properties"
Target="docProps/core.xml"/>
</Relationships>
```

Hinweis für Nutzer von Office 2010:

In Office 2010 werden die Einstellungen für die Schnellzugriffleiste nicht mehr in den QAT-Dateien abgelegt, sondern in Dateien mit der Endung officeUI. Die officeUI-Dateien sind in zwei "Abschnitte" unterteilt. Der erste Abschnitt (<mso>) enthält die Einstellungen für die Schnellzugriffleiste, der zweite Abschnitt (<mso>) die Einstellung für das Menüband.