

Microsoft Access97

Menü- und Symbolleisten per VBA steuern

von:	Anette Becker
eMail:	anette.becker@online.de
Version	1.0 vom 10.07.2000
BeispielDB	CreativBars97
Link	www.accessprofipool.com/Download

Inhaltsverzeichnis

Vorwort:	3
Die Objekt-Typen	3
CommandBars	3
CommandBar-Objekt	3
CommandBarControls	4
CommandBarControl-Objekt	4
CommandBarButton	4
CommandBarComboBox	4
CommandBarPopup	4
VBA-Zugriff auf Symbol- und Menüleisten	5
Deklarationen	5
CommandBars	5
Erstellen von CommandBar-Objekten	5
Konstanten von Position	5
Löschen von CommandBar-Objekten	6
Zurücksetzen von CommandBar-Objekten	6
CommandBarControls	7
Erstellen von CommandBarControl-Objekten	7
Konstanten von msoControlType zur Add-Methode	7
Löschen von CommandBarControl-Objekten	8
Zurücksetzen von CommandBar-Objekten	9
Deaktivieren vorhandener CommandBarControl-Objekte	9
Ausblenden von CommandBarControl-Objekten	9
Integrierte CommandBarControl-Objekte in eigene Befehlsleisten einfügen	10
Copy & Paste	10
Kopieren ohne Zwischenablage	10
Testen ob eine Befehlsleiste existiert	11
CommandBarsControl-Objekte suchen	11
Konstanten von msoControlType zur FindControl-Methode	11
Auslesen / Bestimmen des CommandBarButton-Zustands	12
Die Darstellungsart einer Schaltfläche	13
Konstanten-Auflistung zur Style-Methode bei CommandBarButton	13
Konstanten-Auflistung zur Style-Methode bei CommandBarComboBox	13
Symbolzuweisung mit "FaceID"	13
Menüs und Untermenüs erstellen	14
Nachträgliches Zufügen von Untermenüs	15
Nachträgliches Zufügen von Schaltflächen in Untermenüs	16
Einträge für msoControlComboBox und msoControlDropDown erstellen	16
Abfragen der gewählten Einträge	16
Laufzeit- und andere Fehler	17
Schlußwort	17

Wichtige Hinweise:

Die aufgeführten Beispiele sind ohne Gewähr. Die Benutzung erfolgt vollständig auf eigenes Risiko!

Ich weise daraufhin, daß die in der Dokumentation verwendeten Softwarebezeichnungen und Markennamen der jeweiligen Firmen im allgemeinen warenzeichen-, marken- oder patentrechtlichem Schutz unterliegen.

Copyright © 2000 PC-Creativ Anette Becker ♦ Altenmühlen 30 ♦ 25337 Kölln-Reisiek

Alle Rechte vorbehalten. Das Ändern, das Veröffentlichen und/oder Verbreiten dieser Dokumentation oder von Teilen aus dieser Dokumentation ist in jeglicher Form ausdrücklich untersagt und bedarf meiner schriftlichen Genehmigung.

Vorwort:

Menü- und Symbolleisten lassen sich im Access auch ohne VBA-Kenntnisse erstellen und anpassen (Ansicht/Symbolleisten/Anpassen). Wenn es aber darum geht, einzelne Elemente situationsbedingt zu deaktivieren, auszublenden oder zuzufügen, sind die Möglichkeiten ohne VBA sehr begrenzt.

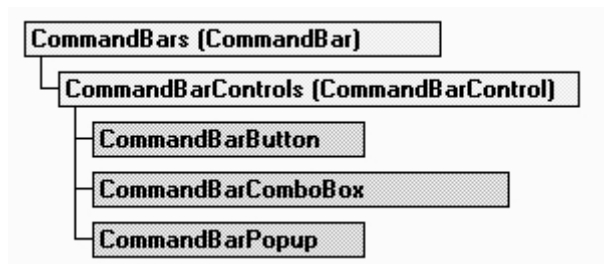
Auf den folgenden Seiten wird nur auf die Anpassung der "CommandBars" unter Verwendung von VBA eingegangen.

Im Downloadbereich des www.accessprofilpool.com finden Sie zur Verdeutlichung eine kleine Anwendung "CreativBars97", in der viele der hier aufgeführten Beispiele verwirklicht wurden und mit der Sie notwendige Id-Nummern der Objekte auslesen können.

Voraussetzung:

Es muß ein Verweis auf die Objektbibliothek »Microsoft Office 8.0 Objekt Library« gesetzt sein.

Die Objekt-Typen



(Abb. aus der Access-Online-Hilfe)

CommandBars

Über den CommandBars-Abschnitt können Sie Symbolleisten, Menüleisten und Kontextmenüs (und deren Komponenten) hinzufügen und ändern.

CommandBars ist die Container-Anwendung für die CommandBar-Objekte, diese wiederum stellen folgende Elemente dar:

- · Menüleisten, Symbolleisten und Shortcut-Menüs
- · Menüs in Menüleisten und in Symbolleisten
- · Untermenüs in Menüs, in Untermenüs und in Shortcut-Menüs

CommandBar-Objekt

Jedes integrierte CommandBar-Objekt hat einen eindeutigen Namen und eine Id-Nummer, über die es selektiert werden kann.

(Es gibt zusätzlich einen lokalisierten Namen "NameLocal", der eine Bezeichnung des Objektes in den Eigenschaften wiedergibt. Dieser erscheint z.B. in "Ansicht/Symbolleisten" oder auch als Überschrift in losgelösten Symbolleisten. Möchten Sie einen lokalisierten Namen auslesen, so geben Sie im Testfenster
 ? commandBars ("Name der Symbolleiste").NameLocal
 ein – als Rückgabewert erhalten Sie die Bezeichnung des Objekts

Die vorhandenen CommandBar-Objekte lassen sich anhand dieser Routine auslesen und auflisten:

```

Dim Z As CommandBar
Dim creaStr As String

For Each Z In CommandBars
  If Z.Type = msoBarTypeNormal Then
    creaStr = creaStr & Z.Name & vbCrLf
  End If
Next Z

MsgBox creaStr
  
```

CommandBarControls

CommandBarControls ist die Container-Anwendung für die CommandBarControl-Objekte, diese wiederum stellen folgende Elemente dar:

- Schaltflächen
- Textfelder
- Dropdown-Listenfelder
- Kombinationsfelder
- Popup-Steuerelemente (Steuerelemente, die ein Menü oder ein Untermenü anzeigen).

CommandBarControl-Objekt

Jedes integrierte CommandBarControl-Objekt hat eine eindeutige Id-Nummer, über die es sich selektieren läßt.

Um die Id-Nummer der Objekte auszulesen und in eine txt-Datei zu speichern können Sie die folgende Prozedur¹ verwenden:

```
Sub outputIDs()  
Dim cbr As Variant  
Dim btn As Variant  
Dim i As Long  
  
Const maxID = 4000  
Open "c:\id_Nr.txt" For Output As #1  
Set cbr = CommandBars.Add("Temporary", msoBarTop, False, True)  
For i = 1 To maxID  
    On Error Resume Next  
    cbr.Controls.Add ID:=i  
Next  
On Error GoTo 0  
For Each btn In cbr.Controls  
    Write #1, btn.ID, btn.Caption  
Next  
cbr.Delete  
Close #1  
End Sub
```

Es wird eine Datei namens "id-Nr.txt" in Ihrem Stammverzeichnis c:\ angelegt.

In der Beispieldanwendung "CreativBars97" habe ich das Modul "mdlSuchPfad" aus der KnowHow von Klaus Oberdalhoff eingefügt, damit ist es möglich, die Datei in einem Verzeichnis nach Wahl und nicht zwingend unter C:\ abzulegen

CommandBarButton

Ist eine Schaltfläche auf einer Symbolleiste oder in einem Menü. Durch das Anklicken können vordefinierte Aktionen (z.B. löschen, speichern, drucken usw.), aber auch eigene Abläufe (Makros, Funktionen usw.) gestartet werden. Außerdem kann man den Zustand auslesen – mehr unter "Auslesen / Bestimmen des CommandBarButton-Zustands"

CommandBarComboBox

Ist ein Dropdown-Listefeld, ein Kombinationsfeld oder benutzerdefiniertes Textfeld, es kann mit und ohne Beschriftung angezeigt werden. Integrierte Objekte dieses Typs lassen sich nicht in den Eigenschaften und Methoden ändern.

CommandBarPopup

Hierbei handelt es sich meist um ActiveX-Steuerelemente, die beim Anklicken ein Menü anzeigen (Bsp. Schriftfarbe in der Symbolleiste). Außerdem sind die Menüelemente in der Menüleiste (die beim Anklicken das Menü öffnen) und die Einträge im Menü, die bei Focuserhalt ein Untermenü anzeigen, auch diesem Typ zugehörig. Auch hier gilt, daß sich integrierte Objekte in des Eigenschaft und Methoden nicht ändern lassen.

¹ Quellenangabe: Microsoft® Office97 Visual Basic® Programmier Handbuch

VBA-Zugriff auf Symbol- und Menüleisten

Jede Access-Anwendung (beziehungsweise: jede Office-Anwendung) beinhaltet ein integriertes Menüsystem. Kleine Änderungen an diesem sind schnell vorgenommen und stellen keinen großen Aufwand dar. Wenn Sie aber sehr viel anpassen möchten, dann ist es vorteilhafter eigene Menüs oder Symbolleisten zu definieren. Die integrierten Menüelemente lassen umfangreiche Änderungen oftmals nicht zu oder reagieren recht empfindlich. Mein persönlicher Rat: Wenn Sie Ihre Anwendung mit speziellen Menüs und Symbolleisten ausstatten möchten, dann nutzen Sie keine Access-integrierten Menü- oder Symbolleisten, sondern erstellen benutzerdefinierte, mit denen Sie beliebig variieren können.

Bei dem Erstellen und Löschen von Befehlsleisten über VBA ist ein besonderes Augenmerk auf die Fehlerrountinen zu legen, lassen Sie diese auch nur zu Testzwecken weg, werden Sie erleben wie schnell sich die Anwendung oder auch das System verabschieden kann....

Deklarationen

Zu Vereinfachung habe ich in der BeispielDB im Deklarationsbereich folgende Einträge vorgenommen:

Dim CreaBar As CommandBar

Dim CreaBarCtrl As CommandBarControl

Dim CreaBarBtn As CommandBarButton

Wenn also in einem Beispiel die expliziten Deklarationen fehlen, dann müssen Sie diese nachholen

CommandBars

Erstellen von CommandBar-Objekten

Die grundlegenden Befehle zum Hinzufügen und Löschen von Symbol- und Menüleisten sind Add (zum Hinzufügen) und Delete (zum Löschen).

Über die Add-Methode können Sie ein neues CommandBar-Objekt anlegen, dabei wird über den Parameter "Position" die Eigenschaft als Symbolleiste oder Kontextmenü und über "Menubar"=True die Funktion als Menüleiste bestimmt.

Syntax:

Ausdruck.Add([Name], [Position], [Menubar], [Temporary])

Ausdruck	ist erforderlich, es wird das CommandBars-Objekt zurückgegeben (die Container-Anwendung).
Name	der Name, den das CommandBar-Objekt erhalten soll, dieser wird in der Auflistung der CommandBar-Objekte angezeigt und gilt auch als Überschrift, wenn das Objekt aus der Verankerung gelöst wird.
Position	damit wird die Position der neuen Befehlsleiste bestimmt – Auflistung der Konstanten unterhalb dieses Absatzes.
Menubar	Standardwert ist False, wird hier True eingetragen, so wird die aktuelle Menüleiste durch diese neu-erstellte ersetzt.
Temporary	Standardwert ist False, damit bleibt die Befehlsleiste erhalten, setzten Sie True ein, so wird dies CommandBar-Objekt beim Schließen der Anwendung gelöscht.

Konstanten von Position

msoBarLeft	Plaziert am linken Fensterrand
msoBarTop	Plaziert am oberen Fensterrand
msoBarRight	Plaziert am rechten Fensterrand
msoBarBottom	Plaziert am unteren Fensterrand
msoBarFloating	Bestimmt, das die Befehlsleiste nicht verankert wird
msoBarPopup	erstellt die Befehlsleiste als Kontextmenü

Im folgenden Beispiel wird per ButtonKlick (btnNew1) eine leere Menüleiste mit Namen "Test1" erzeugt, diese wird am oberen Fensterrand verankert und ist nicht temporär.

!!Vorsicht!! – ist die Menüleiste bereits vorhanden und Sie haben keine Fehlerroutine eingefügt, kommt es zum Absturz. Im angezeigten Beispiel reagiere ich auf diesen Fehler mit einer Anzeige (MsgBox), daß die Menüleiste schon angelegt wurde. Eine andere Möglichkeit wäre, den Objektverweis auf die bestehende Menüleiste zu übergeben (siehe Beispielanwendung "CreativeBars"), der Vorteil – die Schreibweise ist kürzer ;-) der Nachteil – falls auch Steuerelemente eingefügt werden, erscheinen die durch erneutes Aufrufen der Sub auch mehrmals auf einer Befehlsleiste.

BEISPIEL:

```
Private Sub btnNew1_Click()  
On Error GoTo btnNew1_Err
```

```
    Set CreaBar = CommandBars.Add("Test1", msoBarTop, True, False)  
    CommandBars("Test1").Visible = True
```

'erstellt die Menüleiste
'macht sie sichtbar

```
btnNew1_Err:  
If Err.Number = 5 Then  
    MsgBox "Diese Menüleiste besteht bereits."  
Elseif Err.Number <> 0 Then  
    MsgBox "Fehler " & Err.Number & " " & Error$  
End If  
End Sub
```

Es ist vollkommen egal, ob Sie eine Menüleiste, eine Symbolleiste oder ein Kontextmenü erstellen möchten, die Syntax ist immer gleich, lediglich anhand der angegebenen Konstanten entscheiden Sie, welche Art das CommandBar-Objekt annimmt.

Löschen von CommandBar-Objekten

Das Löschen von CommandBar-Objekten ist weitaus einfacher.

Syntax:

Ausdruck.Delete

Löschen Sie eine der Access- Befehlsleisten, so können diese über "Reset" wiederhergestellt werden, doch Vorsicht mit dem Löschen von benutzerdefinierten Befehlsleisten – diese werden endgültig gelöscht und sie müssen bei Bedarf neu erstellt werden.

BEISPIEL:

```
CommandBars("Test1").Delete
```

Zurücksetzen von CommandBar-Objekten

Access-integrierte CommandBars können zurückgesetzt werden, haben Sie eine integrierte Befehlsleiste gelöscht oder in der Zusammensetzung geändert, so können Sie sie mit "Reset" wieder in den alten Zustand versetzen.

Syntax:

Ausdruck.Reset

Ausdruck ist erforderlich. Es wird CommandBar-Objekt zurückgegeben

Im folgenden Beispiel wird die standardmäßige Menüleiste zurückgesetzt

BEISPIEL:

```
CommandBars("Menu Bar").Reset
```

CommandBarControls

Erstellen von CommandBarControl-Objekten

Mit Add kann ein neues Befehlsleisten-Steuerelement erstellt und in CommandBarControls aufgenommen werden. Je nach Einsatz wird durch Add eines der Objekte:

CommandBarButton
CommandBarComboBox
CommandBarPopup

zurückgegeben.

Syntax:

Ausdruck.Add([Type], [Id], [Parameter], [Before], [Temporary])

Ausdruck	ist erforderlich, es wird das CommandBarControls-Objekt zurückgegeben (die Container-Anwendung)
Type	ist optional. Es wird die Art des Steuerelements bestimmt (siehe Auflistung "Konstanten von msoControlType zur Add-Methode")
Id	ist optional. Mittels der Id-Nummer können Sie auf eins der Access-integrierten Steuerelemente zurückgreifen. Geben Sie z.B. "Id:=4" ein um den Button "Drucken" einzufügen. Wird dies Argument ausgelassen oder eine "1" eingefügt, so wird der Symbolleiste ein leeres ActiveX-Steuerelement des angegebenen Typs zugefügt.
Parameter	ist optional. Wird bei Access-integrierten Steuerelementen benutzt um den Befehl auszuführen, bei ActiveX-Steuerelementen können Daten an Prozeduren geschickt werden.
Before	ist optional. Tragen Sie nichts ein, so erscheint das Control-Objekt am Ende der Befehlsleiste, ansonsten erscheint es vor dem Objekt dessen Wert Sie eingegeben haben.
Temporary	ist optional. Standardwert ist False – geben Sie True ein, so wird dies Steuerelement gelöscht, wenn Access geschlossen wird.

Konstanten von msoControlType zur Add-Methode

msoControlButton	Schaltfläche (Button)
msoControlEdit	ein Eingabefeld
msoControlDropdown	ähnlich dem vorhergehenden, nur das ein Pfeil die Liste aufklappen läßt (z.B. Schriftgröße)
msoControlComboBox	ein Kombinationsfeld
msoControlPopup	Das typische Menüelement, das beim Klicken das Menü öffnet und Untermenüs

Im folgenden Beispiel wird auf ButtonKlick (btnSym1) eine temporäre Symbolleiste "Test2" erstellt, auf diese werden dann verschiedene Access-integrierte Steuerelemente eingefügt.

BEISPIEL:

```
Private Sub btnSym1_Click()
Dim SymbolTest2 As CommandBar
On Error GoTo btnSym1_err

Set SymbolTest2 = CommandBars.Add("Test2", msoBarTop, , True)
    With SymbolTest2
        .Controls.Add msoControlButton, 2, , , True           ' Rechtschreibung
        .Controls.Add msoControlButton, 21, , , True          ' Ausschneiden
        .Controls.Add msoControlButton, 19, , , True           ' Kopieren
        .Controls.Add msoControlButton, 22, , , True           ' Einfügen
        .Controls.Add msoControlButton, 4, , , True            ' Drucken
        .Visible = True
    End With

btnSym1_err:
Set SymbolTest2 = CommandBars("Test2")
Resume Next
End Sub
```

Die Schaltflächen sind anhand der Id-Nummer ausgewählt. Am Schluß der With-Anweisung wird die Symbolleiste auf "sichtbar" gestellt.

Das folgende Beispiel erstellt wieder eine temporäre Symbolleiste, jedoch wird diese nicht am Fensterrand verankert. Als Befehlsleisten-Steuerelemente füge ich ein Eingabefeld, ein Kombinationsfeld und ein Popup ein.

BEISPIEL:

```
Sub SymLeiste2anlegen()
Dim SymLeiste2 As CommandBar
Dim CtrlEdit As CommandBarControl
Dim CtrlCombo As CommandBarControl
Dim CtrlPopup As CommandBarControl

On Error GoTo Symleiste2_Err

Set SymLeiste2 = CommandBars.Add("SymLeiste2", msoBarFloating, , True)
    with SymLeiste2.Controls
        Set CtrlEdit = .Add(msoControlEdit)           ' Eingabefeld
        Set CtrlCombo = .Add(msoControlComboBox)      ' Kombinationsfeld
        Set CtrlPopup = .Add(msoControlPopup)         ' Popup
    End With
SymLeiste2.Visible = True

btnSymLeiste2_err:
Set SymLeiste2 = CommandBars("SymLeiste2")
Resume Next
End Sub
```

Diese Symbolleiste ist noch funktionslos – es wurden nur die Control-Objekte eingefügt, denen aber keine Aktion zugewiesen.

Löschen von CommandBarControl-Objekten

Auch hier ist das Löschen wieder einfacher – mit Delete werden CommandBarControls entfernt – Vorsicht, wenn es sich um benutzerdefinierte Control-Objekte handelt, sie müssen neu erstellt werden da sie nicht wiederhergestellt werden können.

Syntax:

Ausdruck.Delete(Temporary)

Ausdruck	erforderlich. Gibt ein CommandBarControl-Objekt zurück
Delete	löscht
Temporary	Standardwert ist False. Wenn Sie ein Control-Objekt nur für diese Arbeitssitzung löschen möchten, dann können Sie temporär löschen. Nach dem nächsten Start der Anwendung ist das Objekt wieder verfügbar.

Integrierte CommandBarControl-Objekte kann man über die Id-Nummer angeben, bei benutzerdefinierten Objekten lautet diese aber immer "1". Wenn man benutzerdefinierte Control-Objekte angeben möchte, so kann man das entweder über den beim erstellen gegebenen Namen machen, oder (und das ist der Normalfall) über die Position auf der Befehlsleiste (von oben nach unten und von links nach rechts).

BEISPIEL:

```
CommandBars("Test2").Controls(6).Delete
```

Damit wird das 6te Objekt aus der Symbolleiste "Test2" gelöscht.

Mit diesem kurzen Befehl soll es auch möglich sein, einzelne Einträge aus der Menüleiste zu löschen. Allerdings habe ich die Erfahrung gemacht, daß sich Access dabei gern verabschiedet (abstürzt). So gehe ich lieber den Umweg über eine Variable und dem Befehl "FindControl"

Im nächsten Beispiel wird über FindControl das Control-Objekt gesucht, als Id-Nummer habe ich 30002 angegeben (das ist die Id-Nummer für "Datei" in der Menüleiste). Das entsprechende Objekt wird an "CreaBarCtrl" übergeben und in der nächsten Zeile gelöscht. Die Id-Nummern der Menüelemente lassen sich auch auslesen, dazu finden Sie eine Function in der Beispieldanwendung "CreativBars97". (Zu dem Befehl "FindControl" im Abschnitt "CommandBarsControl-Objekte suchen" mehr.)

BEISPIEL:

```
Dim CreaBarCtrl As CommandBarControl
```

```
Set CreaBarCtrl = CommandBars("Menu Bar").FindControl(Id:=30002)  
CreaBarCtrl.Delete
```

Zurücksetzen von CommandBar-Objekten

Wie eingangs schon geschrieben, können Access-integrierte Objekte wieder hergestellt werden, wenn also das Control "Datei" wieder in die Menüleiste eingefügt werden soll, erteilen Sie folgenden Befehl:

Syntax:

Ausdruck.Reset

Ausdruck ist erforderlich. Es wird CommandBar-Objekt zurückgegeben

BEISPIEL:

```
CommandBars("Menu Bar").Reset
```

Deaktivieren vorhandener CommandBarControl-Objekte

Wenn Sie eine Schaltfläche deaktivieren möchten, können Sie sie auf "Enable = False" setzen.

2 Beispiele zum Deaktivieren des Buttons "Ansicht" aus der Symbolleiste "Formularansicht"

BEISPIEL 1: (Hier wird über die Id-Nummer das CommandBarControl deaktiviert.)

```
Dim CreaBar As CommandBarControl
```

```
Set CreaBar = CommandBars("Form View").FindControl(Id:=212)  
CreaBar.Enabled = False
```

BEISPIEL 2: (Hier wird die Schaltfläche Nr.1 deaktiviert und die Id-Nummer spielt keine Rolle.)

```
CommandBars("Form View").Controls(1).Enabled = False
```

Wenn die Anpassung der Symbolleisten in Ihrer Anwendung nicht unterbunden werden soll und jeder User diese nach seinen Wünschen zusammenstellen kann, dann ist es auf jeden Fall ratsam die Schaltflächen oder Menüeinträge über die Id-Nummer anzusprechen. Ordnet ein User die Schaltflächen nach seinen Wünschen, würde (bei Beispiel2) das falsche Symbol deaktiviert werden.

Ausblenden von CommandBarControl-Objekten

Im vorherigen Abschnitt wurden die Symbole deaktiviert, d.h. sie sind noch sichtbar, lassen sich aber nicht betätigen. Sie können Symbole und Menüleisteneinträge aber auch ganz ausblenden:

Beispiel:

Ausblenden des Menüeintrags "Fenster" aus der Menüleiste

```
Set CreaBar = CommandBars("Menu Bar").FindControl(Id:=30009)  
CreaBar.Visible = False
```

Integrierte CommandBarControl-Objekte in eigene Befehlsleisten einfügen

Alle integrierten CommandBarControl-Objekte können Sie kopieren und in eigene Befehlsleisten einfügen. Dazu gibt es unterschiedliche Methoden:

Copy & Paste

Sie können über Copy&Paste eine Schaltfläche aus einer vorhandenen Symbolleiste in die Zwischenablage kopieren und in eine andere Befehlsleiste einfügen. Dabei muß aber die Schaltfläche, auf die der Inhalt der Zwischenablage eingefügt werden soll, bereits erstellt sein. Hat diese Schaltfläche ein anderes Symbol und/oder eine andere Aufgabe, wird dies beim Einfügen überschrieben. Möchten Sie eine Schaltfläche (Button) in die Zwischenablage kopieren, so lautet die Syntax:

Ausdruck.CopyFace

Ausdruck ist erforderlich. Es wird ein CommandBarButton-Objekt zurückgegeben

CopyFace Kopiert das angegebene CommandBarButton-Objekt in die Zwischenablage. Gilt nur für Button-Objekte

Im folgenden Beispiel kopieren Sie das 3te Symbol aus der Symbolleiste "Formularansicht" (Form View) in die Zwischenablage

BEISPIEL:

```
Sub BtnKopieren()  
    CommandBars("Form View").Controls(3).CopyFace  
End Sub
```

Zum Einfügen des Symbols aus der Zwischenablage lautet die Syntax:

Ausdruck.PasteFace

Ausdruck ist erforderlich. Es wird ein CommandBarButton-Objekt zurückgegeben

CopyFace Fügt das angegebene CommandBarButton-Objekt aus der Zwischenablage ein.

Mit nachfolgendem Beispiel fügen Sie das in die Zwischenablage kopierte Control-Objekt in die benutzerdefinierte Symbolleiste "Test2" ein. Es wird an die 4te Stelle platziert und würde damit eine vorhandene Schaltfläche überschreiben.

BEISPIEL:

```
sub BtnEinfuegen()  
    CommandBars("Test2").Controls(4).PasteFace  
End Sub
```

Zum Einfügen muß sich an der angegebenen Stelle ein Control-Objekt des Typs Button befinden ansonsten kommt es zu einer Fehlermeldung. Über PasteFace kann kein neuer Button erstellt werden, es kann nur einem vorhandenen Button die Eigenschaften des kopierten zugewiesen werden.

Kopieren ohne Zwischenablage

Die eben aufgezeigte Vorgehensweise bezieht sich nur auf CommandBarControl-Objekte vom Typ Button, wenn Sie andere Control-Objekte aus den integrierten Objekten in Ihre benutzerdefinierte Befehlsleiste einfügen möchten, so verwenden Sie folgende Syntax:

Ausdruck.Copy(Bar, Before)

Ausdruck ist erforderlich. Es wird ein CommandBarButton-, CommandBarControl-, CommandBarComboBox- oder CommandBarPopup-Objekt zurückgegeben.

Bar ist optional. Geben Sie die Befehlsleiste an, auf die das kopierte Objekt eingefügt werden soll

Before ist optional. Geben Sie die Zahl des Objektes an, vor dem das neue eingefügt werden soll. Machen Sie keine Eingabe, so wird das Objekt an das Ende gestellt.

Unterschied zwischen Copy&Paste und Copy... Bei Copy&Paste muß das Button-Objekt schon hergestellt sein – bei Copy nicht.

Copy&Paste bezieht sich nur auf Button-Objekte – Copy können Sie auch für andere Typen verwenden.

Im nachfolgenden Beispiel wird das CommandBarControl-Objekt "Bearbeiten" aus der Menüleiste "Menu Bar" (Standard Menüleiste) in die benutzerdefinierte Menüleiste "Test1" an erste Stelle eingefügt.

BEISPIEL:

```
Dim CreaBar As CommandBar
```

```
Dim CreaBarCtrl As CommandBarControl
```

```
Set CreaBar = CommandBars("Test1")
```

```
Set CreaBarCtrl = CommandBars("Menu Bar").Controls("Bearbeiten")
```

```
CreaBarCtrl.Copy Bar:=CreaBar, Before:=1
```

```
CreaBarCtrl.Visible = True
```

Auch hierbei ist es wichtig, eine Fehlerroutine einzufügen – ist die Befehlsleiste "Test1" nicht geöffnet, so kommt es zur Fehlermeldung und Access wird geschlossen. Das gleiche geschieht, wenn eine neue Befehlsleiste erstellt werden soll, die schon vorhanden ist. Ich nutze die Prozedur "BarFind" um zu testen, ob eine Befehlsleiste existiert.

Testen ob eine Befehlsleiste existiert

Fügen Sie die nachfolgendes in ein Modul

```
Public Function BarFind(CreaName As String) As Boolean
```

```
BarFind = False
```

```
For Each CreaBar In CommandBars
```

```
    If CreaBar.Name = CreaName Then
```

```
        BarFind = True
```

```
    End If
```

```
Next CreaBar
```

```
End Function
```

Nun können Sie, bevor Sie auf die Befehlsleiste greifen, mit If...Then...testen ob diese schon vorhanden ist...

```
If BarFind("Test2") = True Then....
```

CommandBarsControl-Objekte suchen

Benötigen Sie den Wert eines bestimmten CommandBarControl-Objektes, so können Sie die FindControl-Anweisung nutzen.

Syntax:

```
Ausdruck.FindControl([Type], [Id], [Tag], [Visible], [Recursive])
```

Ausdruck ist erforderlich. Gibt den Wert für das CommandBarControl zurück. Geben Sie hier den Namen der Befehlsleiste mit an, so wird nur in dieser nach dem Control-Objekt gesucht, lassen Sie die Namensangabe für das CommandBars weg, so wird in allen vorhandenen CommandBar-Objekten aus dem MS-Office Bereich dieses Control-Objekt gesucht.

Type ist optional. Geben Sie die Art des Control-Objekts an. Welche Typen dafür in Fragen kommen entnehmen Sie bitte der Auflistung "Konstanten von msoControlType zur FindControl-Methode".

Id ist optional. Sie können die Id-Nummer oder den Namen des Control-Objekts angeben.

Tag ist optional. Besitzt das Control-Objekt eine Tag-Markierung, so kann diese zur Suche hier eingegeben werden.

Visible ist optional. Wenn die Suche nur in den sichtbaren Befehlsleisten durchgeführt werden soll.

Recursive ist optional. Standardwert ist False. Wenn die Unterbefehlsleisten bei der Suche eingeschlossen werden sollen.

Konstanten von msoControlType zur FindControl-Methode

msoControlButton

Button (Schaltfläche)

msoControlButtonDropDown	Button mit DropDown-Symbolleiste
msoControlButtonPopup	Button mit Popup (wie im Menü: Button auf Menüleiste)
msoControlComboBox	Kombinations-Box
msoControlDropDown	DropDown-Listefeld (z.B. Prozente der Zoomeinstellung)
msoControlEdit	Eingabefeld
msoControlExpandingGrid	Schaltfläche oder Popup zum Aufklappen des msoControlGrid
msoControlGauge	Element zur Prozenteinstellung
msoControlGraphicCombo	Kombinations-Box mit grafischer Darstellung (z.B. im Word: Kombinationsfeld für die Überschriften)
msoControlGraphicDropDown	DropDown-Listefeld mit grafischer Darstellung (z.B. Linienstärken auswählen)
msoControlGrid	Element im aufgeklappten Feld zur Wertübernahme (z.B. die Farben in Farbauswahl bei Schriften oder Linien)
msoControlOCXDropDown	DropDown-Listefeld mit allen ActiveX- oder OCX-Controls
msoControlPopup	Menüelement
msoControlSplitButtonMRUPopup	Popup, das im aufgeklappten Menü grafische Darstellungen anzeigen kann
msoControlSplitButtonPopup	Button, über den ein Menü mit grafischer Darstellung geöffnet werden kann
msoControlSplitDropDown	Schaltfläche, die eine Liste mit verfügbaren Einträgen anzeigt (z.B. Rückgängig)

Werden mehrere Steuerlemente gefunden, so wird immer der erste gefundene Wert zurückgegeben. Ist die Suche erfolglos, wird "Nothing" zurückgegeben.

Ein Beispiel zu "FindControl" finden Sie im Abschnitt "Löschen von CommandBarControl-Objekte" und im nächsten Abschnitt

Auslesen / Bestimmen des CommandBarButton-Zustands

Das CommandBarButton kann drei verschiedene Zustände annehmen:

msoButtonDown	Gedrückter Button
msoButtonMixed	Unbestimmter Zustand des Buttons
msoButtonUp	Nicht gedrückter Button

Syntax:

Ausdruck.State

Ausdruck	erforderlich. Gibt den Wert des CommandBarControl-Objekts zurück, dabei muß es sich um ein Element "CommandBarButton" handeln
State	Gibt eine der obengenannten Konstanten zurück oder setzt sie neu fest.

BEISPIEL:

```
Set CreaBar = CommandBars("Test1")
    With CreaBar.Controls
        .Add(msoControlButton).State = msoButtonDown
    End With
```

Dieses Beispiel fügt auf die Befehlsleiste "Test1" einen gedrückten Button ein. Denken Sie daran eine Fehlerroutine einzubinden.

Die Darstellungsart einer Schaltfläche

Bestimmen Sie, wie ein Button oder eine ComboBox angezeigt werden soll.

Syntax:

Ausdruck.Style

Ausdruck	erforderlich. Gibt den Wert eines CommandBarButton-Objekts oder eines CommandBarComboBox-Objekts zurück.
Style	Bestimmt das Aussehen nach vordefinierten Konstanten (s. "Konstanten-Auflistung zur Style-Methode").

Konstanten-Auflistung zur Style-Methode bei CommandBarButton

msoButtonAutomatic	Standardwert. Die Anzeigeart bestimmt sich nach Befehlsleiste. D.h. erstellen Sie eins der o.a. Control-Objekte auf einer Symbolleiste, so wird dies automatisch als Icon dargestellt (sofern ein vordefiniertes Icon dafür vorhanden ist, erstellen Sie es dagegen auf einer Menüleiste, wird es als Text dargestellt).
msoButtonCaption	zeigt auf der Schaltfläche nur Text an.
msoButtonIcon	zeigt auf der Schaltfläche nur ein Symbol an.
msoButtonIconAndCaption	zeigt Text und Icon an.

Konstanten-Auflistung zur Style-Methode bei CommandBarComboBox

msoComboLabel	Anzeige mit Beschriftung
msoComboNormal	Standardanzeige

Über die "FaceID" können Sie der Schaltfläche ein anderes Symbol zuweisen.

Symbolzuweisung mit "FaceID"

Diese Zuweisung macht nur Sinn, wenn Sie eine Schaltfläche über ein Icon darstellen lassen.

Ausdruck.FaceId

Ausdruck	erforderlich. Gibt den Wert eines CommandBarButton-Objekts oder eines CommandBarComboBox-Objekts zurück
FaceId	Id-Nummer der vordefinierten CommandBarButton-Objekte.

Wenn Sie Ihrer Schaltfläche, oder einer vordefinierten Schaltfläche ein neues oder anderes Symbol zuweisen möchten, übergeben Sie eine FaceId Anweisung in die Erstellung. Dabei stehen Ihnen nur die Symbole der vorhandenen Schaltflächen zur Verfügung

im folgenden Beispiel füge ich 4x die Schaltfläche "Suchen" (Control-Objekt mit der Id-Nummer 141 – Standardsymbol = Fernglas) in die benutzerdefinierte Symbolleiste "Test2" ein. Die ersten 3 Buttons lasse ich auf die verschiedenen Arten (wie unter "Die Darstellungsart einer Schaltfläche" beschrieben) darstellen und den letzten versehen ich mit dem Symbol "Drucken" vom Typ "msoButtonIconAndCaption". Ich nutze dabei zu Beginn wieder die Kontrolle "BarFind" um Fehler ("Befehlsleiste ist nicht vorhanden!") abzufangen.

BEISPIEL:

```
On Error GoTo btnSym9_err
```

```
If BarFind("Test2") = True Then  
    Set CreaBar = CommandBars("Test2")
```

```
        With CreaBar  
            With .Controls.Add(msoControlButton, 141)  
                .Caption = "Suchen"  
                .BeginGroup = True  
                .Style = msoButtonCaption  
            End With
```

```
            With .Controls.Add(msoControlButton, 141)  
                .Caption = "Suchen"  
                .BeginGroup = True  
                .Style = msoButtonIcon  
            End With
```

```
            With .Controls.Add(msoControlButton, 141)  
                .Caption = "Suchen"  
                .BeginGroup = True  
                .Style = msoButtonIconAndCaption  
            End With
```

```
            With .Controls.Add(msoControlButton, 141)  
                .Caption = "Suchen"  
                .BeginGroup = True  
                .Faceld = 4  
                .Style = msoButtonIconAndCaption  
            End With
```

```
        End With  
        CreaBar.Visible = True
```

```
Else  
    MsgBox "Die Symbolleiste Test2 ist nicht vorhanden!"  
End If
```

```
btnSym9_err:  
If Err.Number <> 0 Then  
    MsgBox "Fehler " & Err.Number & " " & Error$  
End If  
End Sub
```

Menüs und Untermenüs erstellen

Wenn Sie ein Menü mit Untermenüs erstellen möchten, so benötigen Sie:

- eine Befehlsleiste als Menüleiste
- ein Control-Objekt als Popup-Element auf der Menüleiste
- ein Control-Objekt als Pop-Element im Menü
- Control-Objekte als Einträge in dem Untermenü

In dieser Reihenfolge werden diese Menüs angelegt. Möchten Sie im Nachhinein einen weiteren Eintrag in das Untermenü einfügen, so müssen Sie sich auch an diese Reihenfolge halten, damit das Untermenü einwandfrei identifiziert werden kann.

Anhand einiger Beispiele soll dies verdeutlicht werden:

BEISPIEL:

Es wird eine Menüleiste "Test5" erstellt, diese erhält ein msoControlPopup namens "MeinMenu". In das Menü füge ich die Schaltflächen "Drucken" und "Speichern", sowie ein Untermenü "Zwischenablage" ein. Das Untermenü "Zwischenablage" erhält die Schaltflächen "Kopieren" und "Einfügen".

Sie können diese Elemente alle auf einmal in verschachtelten Add-Anweisungen oder Schritt für Schritt zufügen, da ich persönlich die schrittweise Erstellung für übersichtlicher halte, verwende ich nur diese.

```
Dim CreaBar As CommandBars
Dim Button1 As CommandBarButton, Button2 As CommandBarButton
Dim Button3 As CommandBarButton, Button4 As CommandBarButton
Dim UMenu As CommandBarPopup

On Error GoTo btnNew2_Err

Set CreaBar = CommandBars.Add("Test5", msoBarTop, True, True)
    CommandBars("Test5").Visible = True
    With CreaBar.Controls.Add(msoControlPopup)
        .Caption = "MeinMenu"
        .TooltipText = "Popup-Button auf Menüleiste"
    End With
    Set Button1 = CreaBar.Controls("MeinMenu").Controls.Add(Id:=4)
    Set Button2 = CreaBar.Controls("MeinMenu").Controls.Add(Id:=3)
    Set UMenu = CreaBar.Controls("MeinMenu").Controls.Add(msoControlPopup)
    With UMenu
        .Caption = "Zwischenablage"
        .BeginGroup = True
    End With
Set Button3 = UMenu.Controls.Add(Id:=19)
Set Button4 = UMenu.Controls.Add(Id:=22)

btnNew2_Err:
If Err.Number = 5 Then
MsgBox "Diese Menüleiste besteht bereits. Sie müssen Sie erst löschen, bevor sie neu erstellt werden kann"
ElseIf Err.Number <> 0 Then
MsgBox "Fehler " & Err.Number & " " & Error$
End If
```

Nachträgliches Zufügen von Untermenüs

Wenn Sie bereits ein Menü-Element und ein Menü erstellt haben und nachträglich ein Untermenü einfügen möchten, müssen Sie erst die Menüleiste und das Menüelement angeben und dann den Eintrag erstellen.

BEISPIEL:

```
Dim NeuUnterMenu As CommandBarPopup

Set NeuUnterMenu = CommandBars("Test5").Controls("MeinMenu").Controls.Add(msoControlPopup)
    With NeuUnterMenu
        .Caption = "Neues Untermenü"
        .TooltipText = "Nachträglich erstelltes Untermenü"
        .BeginGroup = True
    End With
```

Nachträgliches Zufügen von Schaltflächen in Untermenüs

Manchmal ist es notwendig zu vorhandenen Untermenüs weitere Schaltflächen hinzuzufügen.

Im folgenden Beispiel füge ich der Menüleiste "Test5", Menüelement "Mein Menü", Untermenü "Neues Menü" zwei Schaltflächen (DS sortieren "Aufsteigend" und "Absteigend") zu.

BEISPIEL:

```
Dim CreaBarCtrl As CommandBarControl
Dim Button6 As CommandBarControl
Dim Button7 As CommandBarControl
```

```
Set CreaBarCtrl = CommandBars("Test5").Controls("MeinMenu").Controls("Neues Untermenü")
Set Button6 = CreaBarCtrl.Controls.Add(msoControlButton, 210)
Set Button7 = CreaBarCtrl.Controls.Add(msoControlButton, 211)
```

Vergessen Sie nicht eine Fehlerroutine einzufügen!

Einträge für msoControlComboBox und msoControlDropDown erstellen

Da die bisher erstellten Kombinationsfelder und Listenfelder noch keine Inhalte haben, können Sie dies mit der AddItem-Methode nachholen.

Syntax:

Ausdruck.AddItem(Text, [Index])

Ausdruck	erforderlich. es wird ein CommandBarComboBox-Objekt zurückgegeben. (Auch bei Listenfeldern wird dieser Typ zurückgegeben, da sich die beiden Typen sehr ähneln)
Text	erforderlich. Das dem angegebenen Steuerelement hinzuzufügende Element.
Index	ist optional. Die Position des Elements innerhalb der Liste. Lassen Sie den Wert weg, so wird das Element an das Ende gestellt.

Im folgenden Beispiel werden der ComboBox aus SymLeiste2 drei Einträge zugefügt.

Schaltfläche 1 / Schaltfläche 2 / Schaltfläche 3 – Schaltfläche 3 wird durch die Index-Anweisung an den Anfang gestellt

BEISPIEL:

```
Set CreaBarCtrl = CommandBars("SymLeiste2").FindControl(msoControlComboBox)
With CreaBarCtrl
    .AddItem "Schaltfläche 1"
    .AddItem " Schaltfläche 2"
    .AddItem " Schaltfläche 3", 1
End With
```

Abfragen der gewählten Einträge

Über die ListIndex-Methode können Sie abfragen, welcher Eintrag gewählt wurde

Syntax:

Ausdruck.ListIndex

Ausdruck	erforderlich. Es wird ein CommandBarComboBox-Objekt zurückgegeben. (Auch bei Listenfeldern wird dieser Typ zurückgegeben, da sich die beiden Typen sehr ähneln)
ListIndex	Es wird eine Indexnummer des jeweiligen Eintrags zurückgegeben

Um die bisher erstellten Einträge überhaupt abfragen zu können, ergänze ich das o.a. Beispiel um eine weitere Eigenschaft "OnAction" zu. Mit dieser Eigenschaft wird bestimmt, was passiert wenn ein Eintrag ausgewählt (bei anderen Typen – z.B. wenn ein Button angeklickt) wird.

BEISPIEL:

```
Set CreaBarCtrl = CommandBars("SymLeiste2").FindControl(msoControlComboBox)
    With CreaBarCtrl
        .AddItem "Eintrag 1"
        .AddItem "Eintrag 2"
        .AddItem "Eintrag 3", 1
        .OnAction = "=EintragAbfragen()"
    End With
```

OnAction ist auf eine Prozedur gesetzt, die folgenden Aufbau hat:

Function EintragAbfragen()

```
Set CreaBar = CommandBars("SymLeiste2")
Select Case CreaBar.Controls(2).ListIndex
    Case 1
        MsgBox "der erste Eintrag wurde gewählt"
    Case 2
        MsgBox "der zweite Eintrag wurde gewählt"
    Case 3
        MsgBox "der dritte Eintrag wurde gewählt"
End Select
End Function
```

Laufzeit- und andere Fehler

Hier eine kurze Auflistung der Fehler, die mir beim Erstellen der BeispielDB "CreativBars97" am häufigsten begegnet sind:

Fehlernummer 5 Unzulässiger Prozeduraufruf oder ungültiges Argument
Dieser Fehler tritt auf, wenn eine vorhandene Befehlsleiste nochmals erstellt, oder eine nicht vorhandene gelöscht werden soll. Wird der Fehler nicht abgefangen, so reagiert Access ziemlich zickig – es kommt (beim Löschen) zum Absturz der Anwendung und (je nach Befehlsleiste) auch mal zum Absturz des Systems.
Weiterhin erscheint diese Meldung, wenn Sie versuchen den Inhalt aus integrierten Listen- oder Kombinationsfelder zu löschen oder aus anderen Typen per ListIndex einzulesen.

Fehlernummer 438 Laufzeitfehler – Objekt unterstützt diese Eigenschaft oder Methode nicht
Versuchen Sie eine Methode auf ein nicht dafür geeignetes Objekt anzuwenden, ist Ihnen diese Fehlermeldung bald vertraut. Beispiel: Versuchen Sie ein Control, das nicht vom Typ DropDown-Listenfeld oder Kombinationsfeld ist, mit AddItem zu füllen, erscheint diese Meldung.

»CreativBars97 kann das Makro 'TestContext' nicht finden« – so oder ähnlich reagiert Access auf Befehlsleisten, die angesprochen werden und nicht vorhanden sind. Die hier angegebene Meldung bezieht sich auf ein nicht vorhandenes Kontextmenü. Ähnlich lautet auch die Fehlermeldung, wenn Sie eine "OnAction"-Anweisung eingebunden haben und Access auf diese nicht zugreifen kann.

Die Eigenschaften Id und Type von vorhandenen Control-Objekten können nicht geändert werden, da diese schreibgeschützt sind

Schlußwort

Es ist mir bekannt, daß diese Dokumentation nicht vollständig ist – es gibt noch einige weitere Methoden und Befehle im Zusammenhang mit Befehlsleisten. Ich habe versucht, die gängigsten und (in meinen Augen) wichtigsten hier anzuführen – wenn Sie der Meinung sind, daß wichtige Dinge hier verschwiegen wurden oder unbedingt eingefügt werden sollten, dann würde ich mich über eine kurze Mail sehr freuen.