

Kapitel 20

Befehlsleisten erstellen und verändern

In diesem Kapitel:

Generelles zu Befehlsleisten	624
Menüleisten verändern	630
Symbolleisten erstellen und bearbeiten	651
Kontextmenüs handhaben	672

In diesem Kapitel erfahren Sie alles rund um Befehlsleisten. Sie lernen den Unterschied zwischen den verschiedenen Befehlsleistentypen kennen und wie diese individuell verändert werden können.

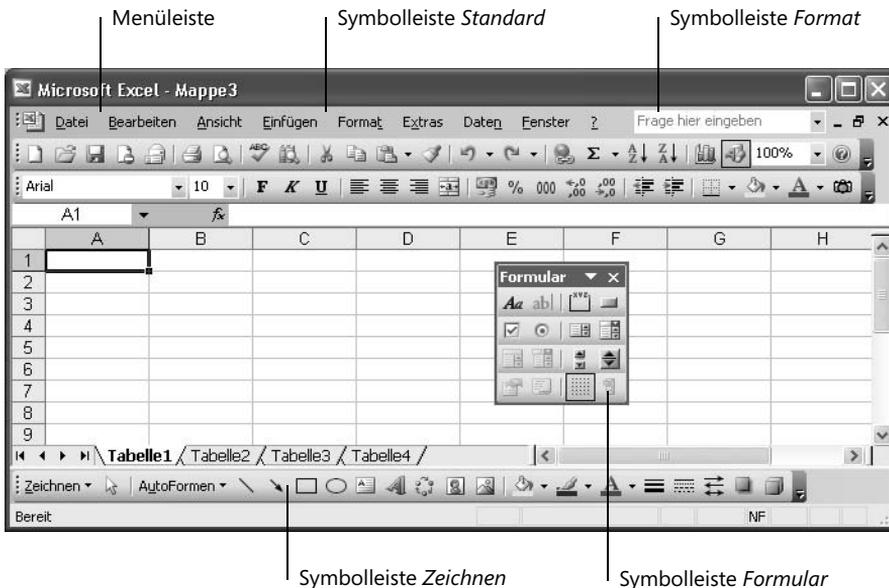
Generelles zu Befehlsleisten

In Excel unterscheidet man zwischen drei verschiedenen Befehlsleistentypen mit Schaltflächen, über die bestimmte Befehle ausgeführt werden können:

- Menüleiste
- Symbolleisten
- Kontextmenüs

Die *Menüleiste* befindet sich standardmäßig am oberen Rand der Applikation. Die *Symbolleisten* sind an verschiedenen Orten zu finden. In der Regel ist dies unterhalb der Menüleiste oder am unteren Rand der Applikation.

Bild 20.1 Menü- und Symbolleisten



Beide Befehlsleistentypen können beliebig platziert werden. Dazu wird der linke Rand der Leiste, in Excel 2003 gepunktet dargestellt, mit gedrückter linker Maustaste gehalten und an den gewünschten Ort verschoben. Menü- und Symbolleisten können an einem der vier Ränder der Applikation ange-dockt werden. Sie können jedoch auch losgelöst an einem bestimmten Ort in der Applikation erscheinen.

Um zusätzliche Symbolleisten einzublenden, verwenden Sie den Menübefehl *Ansicht/Symbolleisten* und aktivieren die gewünschte Symbolleiste. Auf dem gleichen Weg können überflüssige Symbolleisten ausgeblendet werden. Alternativ zum Menübefehl *Ansicht/Symbolleisten* können Sie die Symbolleisten über ein Kontextmenü ein- und ausblenden. Klicken Sie dazu mit der rechten Maustaste auf eine der bestehenden Menü- oder Symbolleisten und aktivieren bzw. deaktivieren einen Eintrag.

Ein weiterer Weg führt über den Menübefehl *Extras/Anpassen*. Auf der Registerkarte *Symbolleisten* des Dialogfeldes *Anpassen* sind sämtliche Menü- und Symbolleisten aufgeführt und können über das vorangestellte Kontrollkästchen aktiviert oder deaktiviert werden.

Auf der Registerkarte *Befehle* finden Sie sämtliche verfügbaren Menübefehle. Sie sind in einzelne Kategorien aufgeteilt, die unter anderem den Namen der Menübefehle entsprechen. Diese Befehlsschaltflächen können bestehenden Symbolleisten hinzugefügt werden. Ziehen Sie die gewünschte Befehlsschaltfläche mit gedrückter linker Maustaste an eine beliebige Stelle in einer der Symbolleisten.

Bild 20.2 Dialogfeld *Anpassen*



Kontextmenüs haben die Eigenschaft, dass sie erst angezeigt werden, wenn mit der rechten Maustaste auf eine Stelle der Applikation geklickt wird. Die verfügbaren Befehle des Kontextmenüs hängen davon ab, welches Objekt der Applikation mit der rechten Maustaste angeklickt wurde.

Befehlsleistentypen per VBA ermitteln

Wie Sie nun wissen, unterscheidet man in Excel zwischen drei Befehlsleistentypen. Zu Beginn mag es etwas verwirrend sein, dass alle über das Auflistungsobjekt `CommandBars` angesprochen werden. Sie mögen sich nun fragen, wie man die Befehlsleisten unterscheiden kann, oder wie man in Erfahrung bringt, welche Befehlsleiste welchen Typs ist.

Um den Typ einer Befehlsleiste zu ermitteln, verwenden Sie die Anweisung `CommandBars` in Kombination mit `Type`. Es stehen, entsprechend der drei Typen, drei Konstanten zur Verfügung.

Tabelle 20.1 Die drei Befehlsleistenkonstanten

Konstante	Wert	Beschreibung
<code>msoBarTypeMenuBar</code>	1	Menüleiste
<code>msoBarTypeNormal</code>	0	Symbolleiste
<code>msoBarTypePopup</code>	2	Kontextmenü

Vielleicht möchten Sie zusätzlich wissen, welche Befehlsleisten durch Excel selbst zur Verfügung gestellt werden und welche Sie benutzerdefiniert (`BuiltIn = False`) erstellt haben. Oder ob die Symboleiste eingblendet ist oder nicht (`Visible`).

Bild 20.3 Ein Auszug des Überblicks über die verschiedenen Befehlsleisten

	A	B	C	D	E	F
1	Symboleleistentyp	Benutzerdefiniert?	Eingblendet?	Englischer Name	Deutscher Name	Index
2	(Type)	(BuiltIn)	(Visible)	(Name)	(NameLocal)	(Index)
3	Menüleiste	Nein	Ja	Worksheet Menu Bar	Arbeitsblatt-Menüleiste	1
4	Menüleiste	Nein	Nein	Chart Menu Bar	Diagrammmenüleiste	2
5	Symboleiste	Nein	Ja	Standard	Standard	3
6	Symboleiste	Nein	Ja	Formatting	Format	4
7	Symboleiste	Nein	Nein	PivotTable	PivotTable	5
8	Symboleiste	Nein	Nein	Chart	Diagramm	6
9	Symboleiste	Nein	Nein	Reviewing	Überarbeiten	7
10	Symboleiste	Nein	Ja	Forms	Formular	8
11	Symboleiste	Nein	Nein	Stop Recording	Aufzeichnung beenden	9
12	Symboleiste	Nein	Nein	External Data	Externe Daten	10
13	Symboleiste	Nein	Nein	Formula Auditing	Formelüberwachung	11
14	Symboleiste	Nein	Nein	Full Screen	Ganzer Bildschirm	12
15	Symboleiste	Nein	Nein	Circular Reference	Zirkelverweis	13
16	Symboleiste	Nein	Nein	Visual Basic	Visual Basic	14
17	Symboleiste	Nein	Nein	Web	Web	15
18	Symboleiste	Nein	Nein	Control Toolbox	Steuerelement-Toolbox	16
19	Symboleiste	Nein	Nein	Exit Design Mode	Entwurfsmodus beenden	17
20	Symboleiste	Nein	Nein	Refresh	Aktualisieren	18
21	Symboleiste	Nein	Nein	Watch Window	Überwachungsfenster	19
22	Symboleiste	Nein	Nein	PivotTable Field List	PivotTable-Feldliste	20
23	Symboleiste	Nein	Nein	Borders	Rahmenlinien	21
24	Symboleiste	Nein	Nein	Protection	Schutz	22
25	Symboleiste	Nein	Nein	Text To Speech	Text zu Sprachein- und -ausgabe	23
26	Symboleiste	Nein	Nein	List	Liste	24
27	Symboleiste	Nein	Nein	Compare Side by Side	Nebeneinander vergleichen	25
28	Symboleiste	Nein	Nein	Drawing	Zeichnen	26
29	Kontextmenü	Nein	Nein	PivotChart Menu	PivotChart-Menü	27
30	Kontextmenü	Nein	Nein	Workbook tabs	Arbeitsmappen-Registerkarte	28
31	Kontextmenü	Nein	Nein	Call	Tabelle	29

Um einen Gesamtüberblick über den nicht geringen Umgang an Befehlsleisten zu erhalten, erstellen Sie am besten eine VBA-Prozedur, die die Informationen für Sie in einem Tabellenblatt ausgibt.

Listing 20.1 Informationen zu Befehlsleisten auslesen

```

Sub GetCommandBarType()
    Dim cmb As CommandBar
    Dim i As Long

    Application.ScreenUpdating = False

    With ThisWorkbook.Worksheets(1)
        ' Überschrift in Zeile 1 erstellen
        Range("A1") = "Symboleleistentyp"
        Range("B1") = "Benutzerdefiniert?"
        Range("C1") = "Eingblendet?"
        Range("D1") = "Englischer Name"
        Range("E1") = "Deutscher Name"
        Range("F1") = "Index"

        ' Überschrift in Zeile 2 erstellen
        Range("A2") = "(Type)"
        Range("B2") = "(BuiltIn)"
        Range("C2") = "(Visible)"
    
```

Listing 20.1 Informationen zu Befehlsleisten auslesen (Fortsetzung)

```

Range("D2") = "(Name)"
Range("E2") = "(NameLocal)"
Range("F2") = "(Index)"

' Überschriftenzeile formatieren
With Range("A1:F2")
    .Font.Bold = True
    .Interior.Color = vbYellow
End With

' -----
' Befehlsleisteninformationen auslesen
' -----

i = 3
For Each cmb In CommandBars
    With cmb
        ' Befehlsleistentyp ermitteln
        If .Type = msoBarTypeMenuBar Then
            Cells(i, 1) = "Menüleiste"
        ElseIf .Type = msoBarTypeNormal Then
            Cells(i, 1) = "Symbolleiste"
        ElseIf .Type = msoBarTypePopup Then
            Cells(i, 1) = "Kontextmenü"
        End If

        ' Benutzerdefiniert Ja/Nein
        If .BuiltIn = True Then
            Cells(i, 2) = "Nein"
        Else
            Cells(i, 2) = "Ja"
        End If

        ' Eingebliendet Ja/Nein
        If .Visible = True Then
            Cells(i, 3) = "Ja"
        Else
            Cells(i, 3) = "Nein"
        End If

        Cells(i, 4) = .Name           ' Englischer Name
        Cells(i, 5) = .NameLocal     ' Deutscher Name
        Cells(i, 6) = .Index         ' Index
    End With
    i = i + 1
Next cmb
' -----

End With

Application.ScreenUpdating = True

Columns("A:F").AutoFit
End Sub

```



Das obige Beispiel befindet sich auf der CD-ROM zum Buch im Ordner `\Buch\Kap20`. Die Mappe nennt sich `Bsp20_01.xls`. Die Prozedur ist im Modul `mdl_01_Type` untergebracht.

Den Speicherort der Befehlsleisten ermitteln

Sofern Sie mit der Excel-Version 2003 arbeiten, werden sämtliche darin enthaltenen Symbolleisten in einer Datei namens `Excel11.xlb` gespeichert. In der Version 2002 nennt sich die Datei `Excel10.xlb` und in der Version 2000 lediglich `Excel.xlb`, also noch ohne Versionsnummer.

Wenn Sie nicht sicher sind, welche Version auf Ihrem System installiert ist, rufen Sie den Menübefehl `?Info` auf.

Sie können die Versionsnummer auch per VBA in Erfahrung bringen.

Listing 20.2 Die Excel-Versionsnummer abfragen

```
Sub GetVersion()  
    MsgBox Application.Version  
End Sub
```

Bei den Gegebenheiten einer Standardinstallation befindet sich die Datei (`*.xlb`), die sämtliche Befehlsleisten enthält, in folgendem Verzeichnis:

`C:\Dokumente und Einstellungen\<Benutzername>\Anwendungsdaten\Microsoft\Excel`

WICHTIG

Die Datei ist erst verfügbar, wenn bereits Änderungen an den Symbolleisten vorgenommen wurden.

Beim ersten Teil des Pfades handelt es sich um einen Windows-Standardpfad, der nach der Installation von Windows bzw. Office automatisch zur Verfügung steht. Der `Benutzername` ist variabel. In den meisten Fällen wird dies Ihr angemeldeter Systemname sein.

`C:\Dokumente und Einstellungen\<Benutzername>\Anwendungsdaten`

Mittels einer VBA-Prozedur können Sie Ihren genauen Pfad ermitteln. Verwenden Sie dazu die `Environ`-Funktion und tragen im runden Klammernpaar umgeben von Anführungszeichen den Zeichenfolgenausdruck `APPDATA` ein.

Listing 20.3 Den Speicherpfad ermitteln

```
Sub GetPath()  
    MsgBox Environ("APPDATA")  
End Sub
```

Nach dem Ausführen des Codes wird in einem Meldungsfeld der gesuchte Pfad ausgegeben.

Bild 20.4 Den Speicherpfad ausgeben



Mittels der Environ-Funktion können Sie auch weitere Systeminformationen auslesen. Welche das sind, können Sie mittels einer VBA-Prozedur ermitteln.

Listing 20.4 Zeichenfolgenausdrücke von Environ ermitteln

```
Sub GetEnviron()
    Dim i As Integer
    i = 1

    Do While Environ(i) <> ""
        Debug.Print Environ(i)
        i = i + 1
    Loop
End Sub
```

Die Ausgabe der Informationen erfolgt im Direktfenster.



Die obigen Beispiele befinden sich auf der CD-ROM zum Buch im Ordner `\Buch\Kap20`. Die Mappe nennt sich `Bsp20_02.xls`. Die Prozeduren sind im Modul `mdl_01_Path` untergebracht.

Eine Sicherungskopie der Befehlsleistendatei erstellen

Wenn Sie selbst erstellte Symbolleisten oder Änderungen an bestehenden Symbolleisten auf anderen Computern zur Verfügung stellen möchten, muss diese Datei in das oben genannte Verzeichnis des Zielcomputers kopiert werden.

Bevor Sie eine bestehende Datei ersetzen, sollten Sie unbedingt eine Sicherungskopie der vorhandenen Datei erstellen. So haben Sie die Möglichkeit, die »alten« Symbolleisten wieder zurückzuholen. Manuell kopieren Sie die Datei und weisen Ihr einen aussagekräftigen Namen zu wie zum Beispiel: `Backup_Excel11.xlb`.

Sie können die Sicherungskopie auch per VBA erstellen. Die folgende Prozedur ist auf die Version 11.0, also Excel 2003 ausgerichtet. Wenn Sie mit einer anderen Version arbeiten, brauchen Sie lediglich den Dateinamen zu ändern.

Listing 20.5 Eine Sicherungskopie der *.xlb für die Version 11 erstellen

```
Sub CreateBackup()
    Dim strPath As String
    Dim strFile As String

    strPath = Environ("APPDATA") & "\Microsoft\Excel\"
    strFile = "Excel11.xlb"
```

Listing 20.5 Eine Sicherungskopie der *.xlb für die Version 11 erstellen (Fortsetzung)

```

If Dir(strPath & strFile) <> "" Then
    FileCopy strPath & strFile, _
        strPath & "Backup_" & strFile
    MsgBox "Die Sicherungskopie wurde erstellt."
Else
    MsgBox "Der Pfad oder die Version ist ungültig, oder " & _
        "die *.xlb besteht noch nicht."
End If
End Sub

```



Das obige Beispiel befindet sich auf der CD-ROM zum Buch im Ordner \Buch\Kap20. Die Mappe nennt sich *Bsp20_03.xls*. Die Prozedur ist im Modul *mdl_01_Backup* untergebracht.

Menüleisten verändern

In Excel gibt es zwei unterschiedliche Menüleisten. Die eine ist die *Arbeitsblatt-Menüleiste*. Sie wird standardmäßig am oberen Rand der Applikation angezeigt. Des Weiteren gibt es die *Diagrammmenüleiste*. Wenn ein Diagramm im Tabellenblatt aktiv ist, wird die normale Menüleiste durch die *Diagrammmenüleiste* ersetzt. Sie weist zum Teil unterschiedliche Menübefehle auf.

Welche Menüleisten gibt es?

Um die Menüleisten per VBA zu ermitteln, verwenden Sie folgende Prozedur. Im Code werden in einer For-Schleife sämtliche verfügbaren Befehlsleisten durchlaufen. In der If-Entscheidung wird festgelegt, dass nur die Menüleisten aufgelistet werden (msoBarTypeMenuBar).

HINWEIS

Einen Code, um sämtliche Befehlsleisten zu ermitteln, finden Sie zu Beginn dieses Kapitels im Abschnitt »Befehlsleistentypen per VBA ermitteln«.

Listing 20.6 Namen der Menüleisten ermitteln

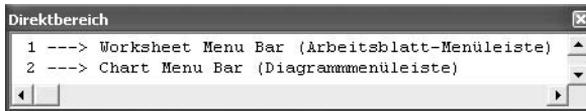
```

Sub MenuBarNames()
    Dim i As Integer

    For i = 1 To Application.CommandBars.Count
        If CommandBars(i).Type = msoBarTypeMenuBar Then
            Debug.Print i & " ---> " & _
                Application.CommandBars(i).Name & _
                " (" & Application.CommandBars(i).NameLocal & ")"
        End If
    Next i
End Sub

```

Bild 20.5 Index und Name der Menüleisten



Die Menüleiste kann über ihren englischen Namen `Worksheet Menu Bar` oder über den Index 1 angesprochen werden. Der Name der Diagrammmenüleiste lautet `Chart Menu Bar`. Deren Index ist ebenfalls 1, denn es kann immer nur eine der beiden angezeigt werden und diese ist automatisch die erste.

Das Objekt `CommandBar` wird gleichermaßen für Menü- und Symbolleisten sowie für Kontextmenüs eingesetzt. Die Leisten unterscheiden sich hauptsächlich durch deren Namen bzw. Index. Wenn wir von *Befehlsleisten* sprechen, sind damit alle drei Leistentypen gemeint.

Befehlsleisten sind nicht an eine bestimmte Applikation gebunden. Das dem Objekt `CommandBar` übergeordnete Objekt ist somit das Objekt `Application`.



Das obige Beispiel befindet sich auf der CD-ROM zum Buch im Ordner `\Buch\Kap20`. Die Mappe nennt sich `Bsp20_04.xls`. Die Prozedur ist im Modul `mdl_01_MenuBarNames` untergebracht.

Eine Menüleiste ein- oder ausblenden

Um sicherzustellen, dass die Menüleiste verfügbar ist, verwenden Sie die Eigenschaft `Enabled = True`. Wenn Sie die Menüleiste per VBA ausblenden möchten, ersetzen Sie den Wert `True` durch `False`.

Um die Menü- oder Diagrammleiste einzublenden, verwenden Sie den Index 1.

Listing 20.7 Eine Menüleiste über den Index ansprechen

```
Sub EnableMenu()
    Application.CommandBars(1).Enabled = True
End Sub
```

An Stelle des Indexes können Sie die Menüleiste auch über den Namen `Worksheet Menu Bar` ansprechen.

Listing 20.8 Die Menüleiste über den Namen ansprechen

```
Sub EnableMenu()
    Application.CommandBars("Worksheet Menu Bar").Enabled = True
End Sub
```

Wenn Sie die Diagrammleiste über den Namen ansprechen möchten, verwenden Sie als Name `Chart Menu Bar`.

Listing 20.9 Die Diagrammleiste über den Namen ansprechen

```
Sub EnableMenu()
    Application.CommandBars("Chart Menu Bar").Enabled = True
End Sub
```

TIPP

Die zurzeit aktive Menüleiste kann auch über die Anweisung `Application.ActiveMenuBar` angesprochen werden.

Ein Menüelement anfügen

Wenn Sie auf einen Menübefehl klicken, wird eine Liste weiterer Befehle angezeigt. Sie werden Menüelemente genannt. Das VBA-Objekt dazu lautet `CommandBarButton`.

Sie haben die Möglichkeit, einem bestehenden Menübefehl ein weiteres Element hinzuzufügen. Dazu müssen Sie die englischen Namen der Menübefehle kennen.

Tabelle 20.2 Namen der Menübefehle

Menübefehlsname	Englische Bezeichnung	Index
Datei	File	30002
Bearbeiten	Edit	30003
Ansicht	View	30004
Einfügen	Insert	30005
Format	Format	30006
Extras	Tools	30007
Daten	Data	30011
Fenster	Window	30009
?	Help	30010

Type Ein zusätzliches Menüelement wird über die Anweisung `Controls.Add` erzeugt. Es können mehrere Argumente übergeben werden. Wenn Sie eine Schaltfläche erzeugen möchten, wie es im folgenden Beispiel geschieht, weisen Sie dem Argument `Type` die Konstante `msoControlButton` zu.

Es stehen fünf unterschiedliche `Type`-Konstanten zur Verfügung:

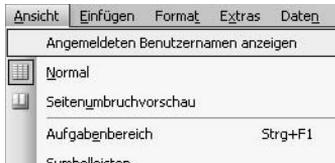
Tabelle 20.3 `Type`-Konstanten

Konstante	Beschreibung
<code>msoControlButton</code>	Schaltfläche
<code>msoControlEdit</code>	Textbox
<code>msoControlDropDown</code>	Dropdown-Feld
<code>msoControlComboBox</code>	Kombinationsfeld
<code>msoControlPopup</code>	Menüleistenelement mit einer weiteren Ebene

Before Mittels des Arguments `Before` können Sie bestimmen, an welcher Position die Schaltfläche eingefügt werden soll. Wenn Sie den Wert 1 verwenden, wird die Schaltfläche an erster Stelle eingefügt. Wenn Sie das Argument weglassen, wird die Schaltfläche an letzter Stelle platziert.

- Temporary** Wenn Sie das Argument `Temporary:=True` mitliefern, bleibt die Schaltfläche nur so lange erhalten, bis die Applikation geschlossen wird.
- Caption** Die Eigenschaft `Caption` wird verwendet, um der neuen Schaltfläche den Text zuzuweisen, der als Schaltflächenbeschriftung gilt.
- OnAction** Der Eigenschaft `OnAction` wird der Name der Prozedur zugewiesen, die beim Anklicken der Schaltfläche aufgerufen werden soll.

Bild 20.6 Ein zusätzliches Menüelement an erster Stelle



Listing 20.10 Menüelement einfügen

```
Sub AddMenuElement()
    Dim cmbb As CommandBarButton

    ' Menübefehl "Ansicht" ergänzen
    Set cmbb = Application.CommandBars("View")._
        Controls.Add(Type:=msoControlButton, _
            Before:=1, _
            Temporary:=True)

    With cmbb
        ' Beschriftung der Schaltfläche
        .Caption = "Angemeldeten Benutzernamen anzeigen"

        ' Prozedur aufrufen
        .OnAction = "ShowUser"
    End With

    Set cmbb = Nothing
End Sub
```

Listing 20.11 Aufzurufende Prozedur

```
Sub ShowUser()
    MsgBox Environ("UserName")
End Sub
```



Das obige Beispiel befindet sich auf der CD-ROM zum Buch im Ordner `\Buch\Kap20`. Die Mappe nennt sich `Bsp20_05.xls`. Die Prozedur ist im Modul `mdl_01_AddMenuElement` untergebracht.

Ein Menüelement löschen

Wenn Sie ein Menüelement löschen möchten (`Delete`), geben Sie den Namen des Menübefehls sowie den Index des Elementes an. Da sich an der angegebenen Position eine andere Schaltfläche

befinden könnte, sollte vor dem Löschen der Name der Schaltfläche überprüft werden. Ansonsten würde womöglich eine falsche Schaltfläche entfernt.

Listing 20.12 Eingefügtes Menüelement über den Index entfernen

```
Sub DeleteMenuElement()  
    With Application.CommandBars("View").Controls(1)  
        If .Caption = "Angemeldeten Benutzernamen anzeigen" Then  
            .Delete  
        End If  
    End With  
End Sub
```

Wenn Sie die Position, also den Index des Menüelements nicht kennen, können Sie mittels einer For Each-Schleife sämtliche Menüelemente des Menübefehls durchlaufen. Es wird dabei der angegebene Name mit den Menüelementen verglichen. Sobald eine Übereinstimmung stattfindet, wird das betreffende Menüelement gelöscht und die Prozedur wird verlassen.

Listing 20.13 Eingefügtes Menüelement über den Namen entfernen

```
Sub DeleteMenuElement2()  
    Dim ctr As CommandBarControl  
  
    With Application.CommandBars("View")  
        For Each ctr In .Controls  
            If ctr.Caption = "Angemeldeten Benutzernamen anzeigen" Then  
                ctr.Delete  
                Exit Sub  
            End If  
        Next ctr  
    End With  
End Sub
```



Die obigen Beispiele befinden sich auf der CD-ROM zum Buch im Ordner `\Buch\Kap20`. Die Mappe nennt sich `Bsp20_05.xls`. Die Prozeduren sind im Modul `mdl_02_DeleteMenuElement` untergebracht.

Die Menüleiste zurücksetzen

Sie haben die Möglichkeit, mittels der `Reset`-Methode die Menüleiste in ihren Originalzustand zurückzusetzen.

WICHTIG

Bedenken Sie vor dem Ausführen des Codes, dass sämtliche benutzerdefinierten Einstellungen, die Sie an der Menüleiste vorgenommen haben, verloren gehen. Wenn Sie auf der sicheren Seite sein möchten, erstellen Sie vor dem Ausführen des Codes eine Sicherungskopie der `*.xlb`-Datei, so wie dies zu Beginn des Kapitels beschrieben wurde.

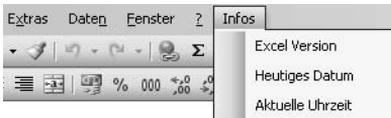
Listing 20.14 Menüleiste zurücksetzen

```
Sub ResetMenubar()
    Application.CommandBars("Worksheet Menu Bar").Reset
End Sub
```

Einen neuen Menübefehl erstellen

Neben der Möglichkeit, ein Element einem bestehenden Menübefehl anzufügen, können Sie auch einen neuen Menübefehl mit einer Reihe an Menüelementen erzeugen. Es gibt dazu verschiedene Techniken. Wir verwenden hier das Objekt `CommandBarPopup`, das auch tatsächlich dafür vorgesehen ist.

Bild 20.7 Neuer Menübefehl mit drei Elementen



Zu Beginn der Prozedur wird zuerst eine weitere Prozedur aufgerufen (`Call DeleteNewItem`). Den Inhalt dieser Prozedur werden wir uns in einem zweiten Schritt ansehen. Die Prozedur löscht einen bereits vorhandenen Menübefehl mit diesem Namen. Dies tun wir, um zu vermeiden, dass der Menübefehl mehrmals in die Menüleiste eingefügt wird. Dies für den Fall, dass die Prozedur wiederholt ausgeführt wird.

In der Prozedur werden zwei Objekte referenziert. Das erste Objekt ist die Menüleiste, die wir der Variablen `cmb` zuweisen. Das zweite Objekt ist der Menübefehl, den wir der Variablen `cmbp` übergeben. Nach der Referenzierung weisen wir dem neuen Menübefehl den Namen »Infos« zu.

In den drei `With`-Anweisungen werden die drei Menüelemente erstellt (`Controls.Add`).

Listing 20.15 Einen neuen Menübefehl mit drei Elementen erzeugen

```
Sub AddNewMenu()
    Dim cmb As CommandBar
    Dim cmbp As CommandBarPopup

    Call DeleteNewMenu

    ' 1. Bezug auf aktive Menüleiste
    ' 2. Neuen Menübefehl erzeugen
    Set cmb = Application.CommandBars("Worksheet Menu Bar")
    Set cmbp = cmb.Controls.Add(Type:=msoControlPopup)
    cmbp.Caption = "Infos"

    ' Erstes Menüelement
    With cmbp.Controls.Add
        .Caption = "Excel Version"
        .OnAction = "Version"
    End With

    ' Zweites Menüelement
    With cmbp.Controls.Add
```

Listing 20.15 Einen neuen Menübefehl mit drei Elementen erzeugen (*Fortsetzung*)

```
.Caption = "Heutiges Datum"  
.OnAction = "DateToday"  
End With  
  
' Drittes Menüelement  
With cmbp.Controls.Add  
.Caption = "Aktuelle Uhrzeit"  
.OnAction = "TimeNow"  
End With  
  
Set cmb = Nothing  
Set cmbp = Nothing  
End Sub
```

Listing 20.16 Die drei zugehörigen Prozeduren

```
Sub Version()  
MsgBox Application.Version  
End Sub  
  
Sub DateToday()  
MsgBox Date  
End Sub  
  
Sub TimeNow()  
MsgBox Time  
End Sub
```

Sehen wir uns nun die Prozedur an, die den Menübefehl wieder entfernt. Die Anweisung, um einen Menübefehl zu löschen, kann aus nur einer Codezeile bestehen:

```
Application.CommandBars("Worksheet Menu Bar").Controls("Infos").Delete
```

Bei dieser Methode wird allerdings der Debugger gestartet, wenn der Menübefehl nicht oder nicht mehr existiert. Um dies zu vermeiden, können Sie beispielsweise die Anweisung `On Error Resume Next` voranstellen.

Wenn Sie einen eventuellen Fehler sauber ausprogrammieren möchten, können Sie mit einer `For Each`-Schleife arbeiten, die alle Menübefehle der Menüleiste durchläuft. In einer `If`-Entscheidung wird geprüft, ob ein Menübefehl mit dem Namen »Infos« vorhanden ist. Wenn ja, wird er gelöscht. Ansonsten wird die Prozedur verlassen, ohne im Debugger zu enden. Auf Wunsch können Sie eine `Else`-Verzweigung einfügen und dort über `MsgBox` eine entsprechende Nachricht ausgeben, dass der Menübefehl nicht gefunden wurde.

Listing 20.17 Einen Menübefehl löschen

```
Sub DeleteNewMenu()  
Dim cmb As CommandBar  
Dim cmbp As CommandBarPopup  
  
Set cmb = Application.CommandBars("Worksheet Menu Bar")  
  
For Each cmbp In cmb.Controls
```

Listing 20.17 Einen Menübefehl löschen (Fortsetzung)

```

    If cmbp.Caption = "Infos" Then
        cmbp.Delete
    End If
Next cmbp

Set cmb = Nothing
End Sub

```

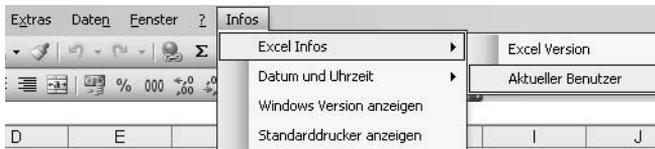


Die obigen Beispiele befinden sich auf der CD-ROM zum Buch im Ordner `\Buch\Kap20`. Die Mappe nennt sich `Bsp20_06.xls`. Die Prozeduren sind im Modul `mdl_01_NewMenu` untergebracht.

Menüelemente mit weiteren Verzweigungen

Menüelemente können weitere Unterelemente enthalten. Innerhalb des Menübefehls sind diese am rechten Rand durch ein Dreieck gekennzeichnet, das drauf hindeutet, dass weitere Elemente hinterlegt sind. Wie Sie dem Bild 20.8 entnehmen können, sind im Menübefehl zwei Menüelemente enthalten, die weitere Unterelemente beinhalten.

Bild 20.8 Untermenüelemente



Um Unterelemente zu programmieren, muss ein weiteres Objekt vom Typ `CommandBarPopup` referenziert werden. In der nachfolgenden Prozedur werden zwei Menüelemente erzeugt, die Unterelemente enthalten. Es werden zudem zwei Menüelemente ohne Unterelemente erzeugt.

Die Referenzierung für die Unterelemente findet nicht zu Beginn der Prozedur statt, sondern erst an der Stelle, wo auch die zugehörigen Elemente erzeugt werden.

Listing 20.18 Menübefehl mit Menüelementen und Menüunterelementen

```

Sub AddSubMenuElement()
    Dim cmb As CommandBar
    Dim cmbp As CommandBarPopup
    Dim cmbpSub As CommandBarPopup

    Call DeleteNewMenu

    ' 1. Bezug auf aktive Menüleiste
    ' 2. Neuen Menübefehl erzeugen
    Set cmb = Application.CommandBars("Worksheet Menu Bar")
    Set cmbp = cmb.Controls.Add(Type:=msoControlPopup)
    cmbp.Caption = "Infos"

    ' Erstes Menüelement

```

Listing 20.18 Menübefehl mit Menüelementen und Menüunterelementen (Fortsetzung)

```

Set cmbpSub = cmbp.Controls.Add(Type:=msoControlPopup)
cmbpSub.Caption = "Excel Infos"
    ' Erstes Untermenüelement
    With cmbpSub.Controls.Add
        .Caption = "Excel Version"
        .OnAction = "Version"
    End With
    ' Zweites Untermenüelement
    With cmbpSub.Controls.Add
        .Caption = "Aktueller Benutzer"
        .OnAction = "User"
    End With
    ' Drittes Untermenüelement
    With cmbpSub.Controls.Add
        .Caption = "Standarddrucker anzeigen"
        .OnAction = "Printer"
    End With

    ' Zweites Menüelement
Set cmbpSub = cmbp.Controls.Add(Type:=msoControlPopup)
cmbpSub.Caption = "Datum und Uhrzeit"
    ' Erstes Untermenüelement
    With cmbpSub.Controls.Add
        .Caption = "Heutiges Datum"
        .OnAction = "DateToday"
    End With
    ' Zweites Untermenüelement
    With cmbpSub.Controls.Add
        .Caption = "Aktuelle Uhrzeit"
        .OnAction = "TimeNow"
    End With

Set cmb = Nothing
Set cmbp = Nothing
Set cmbpSub = Nothing
End Sub

```



Das obige Beispiel befindet sich auf der CD-ROM zum Buch im Ordner `\Buch\Kap20`. Die Mappe nennt sich `Bsp20_07.xls`. Die Prozedur ist im Modul `mdl_01_SubMenu` untergebracht.

Die zugehörigen Prozeduren befinden sich im selben Modul. Es wurde darauf verzichtet, diese hier abzdrukken.

Eigene Menübefehle nur in bestimmten Mappen anzeigen

Wie Sie bereits erfahren haben, kann durch das Argument `Temporary` ein Menübefehl so eingerichtet werden, dass er nach dem Beenden der Applikation wieder entfernt wird.

In vielen Fällen ist es jedoch sinnvoller, wenn ein Menübefehl nur angezeigt wird, wenn eine bestimmte Mappe geöffnet wird. Nach dem Schließen der Mappe soll der Menübefehl wieder entfernt werden.

Um dies zu realisieren, muss mit zwei Ereignisprozeduren gearbeitet werden, die im Modul *DieseArbeitsmappe* hinterlegt werden. Im Ereignis `Workbook_Open` wird der Menübefehl erzeugt. Im Ereignis `Workbook_BeforeClose` wird der Menübefehl wieder entfernt.

Die zugehörigen Prozeduren, die über den Menübefehl aufgerufen werden können, werden separat in einem Modul untergebracht.

HINWEIS Menübefehle lassen sich auch pro Tabellenblatt ein- oder ausblenden. Dazu werden die Ereignisprozeduren `Worksheet_Activate` und `Worksheet_Deactivate` verwendet. Die beiden Ereignisprozeduren werden dem Tabellenblatt hinterlegt, in dem der Menübefehl erscheinen soll.

Listing 20.19 Menübefehl beim Öffnen der Mappe erzeugen

```
Private Sub Workbook_Open()
    Dim cmb As CommandBar
    Dim cmbp As CommandBarPopup

    Call DeleteNewMenu

    Set cmb = Application.CommandBars("Worksheet Menu Bar")
    Set cmbp = cmb.Controls.Add(Type:=msoControlPopup)
    cmbp.Caption = "Infos"

    With cmbp.Controls.Add
        .Caption = "Excel Version"
        .OnAction = "Version"
    End With

    MsgBox "Der Menübefehl ""Infos"" wurde erzeugt."

    Set cmb = Nothing
    Set cmbp = Nothing
End Sub
```

Listing 20.20 Menübefehl beim Schließen der Mappe wieder entfernen

```
Private Sub Workbook_BeforeClose(Cancel As Boolean)
    Dim cmb As CommandBar
    Dim cmbp As CommandBarPopup

    Set cmb = Application.CommandBars("Worksheet Menu Bar")

    For Each cmbp In cmb.Controls
        If cmbp.Caption = "Infos" Then
            cmbp.Delete
        End If
    Next cmbp

    MsgBox "Der Menübefehl ""Infos"" wurde entfernt."

    Set cmb = Nothing
End Sub
```



Das obige Beispiel befindet sich auf der CD-ROM zum Buch im Ordner `\Buch\Kap20`. Die Mappe nennt sich `Bsp20_08.xls`. Die Ereignisprozeduren befinden sich im Modul *DieseArbeitsmappe*.

Einen Menübefehl deaktivieren oder ausblenden

Es gibt zwei unterschiedliche Möglichkeiten, um Menübefehle auszublenden, sodass der Benutzer keine Möglichkeit hat, die darin enthaltenen Befehle zu verwenden.

Der erste Weg führt darüber, den Menübefehl lediglich zu deaktivieren. Der Menübefehl wird dabei grau hinterlegt, ist jedoch weiterhin sichtbar.

Bild 20.9 Der Menübefehl *Ansicht* wird als »nicht wählbar« dargestellt



Der Vorgang geschieht in unserem Beispiel beim Wechsel in das zweite Tabellenblatt. Das heißt, wenn Sie das zweite Tabellenblatt aktivieren, wird die Ereignisprozedur ausgeführt. Der Code ist somit dem zweiten Tabellenblatt hinterlegt.

Die Eigenschaft, die verwendet wird, um den Menübefehl zu deaktivieren, lautet `Enabled = False`. Um den Menübefehl wieder zu aktivieren, wird der boolesche Wert `False` durch `True` ersetzt.

Listing 20.21 Menübefehl *Ansicht* deaktivieren

```
Private Sub Worksheet_Activate()
    Dim cmb As CommandBar
    Set cmb = Application.CommandBars("Worksheet Menu Bar")

    ' Menübefehl "Ansicht" grau hinterlegen
    Application.CommandBars("View").Enabled = False

    Set cmb = Nothing
End Sub
```

Listing 20.22 Menübefehl *Ansicht* wieder aktivieren

```
Private Sub Worksheet_Deactivate()
    Dim cmb As CommandBar
    Set cmb = Application.CommandBars("Worksheet Menu Bar")

    ' Menübefehl "Ansicht" wieder aktivieren
    Application.CommandBars("View").Enabled = True

    Set cmb = Nothing
End Sub
```



Das obige Beispiel befindet sich auf der CD-ROM zum Buch im Ordner `\Buch\Kap20`. Die Mappe nennt sich `Bsp20_09.xls`. Die Ereignisprozeduren befinden sich im Modul `Tabelle1`.

Eine zweite Möglichkeit, den Zugriff auf einen Menübefehl zu verweigern, besteht darin, diesen gänzlich auszublenden.

Bild 20.10 Menüleiste vor und nach dem Ausblenden des Menübefehls *Ansicht*



Der erste Gedanke wäre wohl, dass der obige Code verwendet werden kann und lediglich die Eigenschaft `Enabled` durch `Visible` vertauscht werden muss. Dem ist jedoch nicht so. Um den gewünschten Effekt zu erreichen, muss der Menübefehl über dessen ID-Nummer angesprochen werden. Dazu wird die Methode `FindControl` verwendet. Die Indexnummern für `FindControl` können Sie der Tabelle 20.2 entnehmen.

Listing 20.23 Menübefehl ausblenden

```
Sub DisableControl()
    Dim cmb As CommandBar
    Set cmb = Application.CommandBars("Worksheet Menu Bar")

    ' Menübefehl "Ansicht" ausblenden
    cmb.FindControl(ID:=30004).Visible = False

    Set cmb = Nothing
End Sub
```

Listing 20.24 Menübefehl wieder einblenden

```
Sub ShowControl()
    Dim cmb As CommandBar
    Set cmb = Application.CommandBars("Worksheet Menu Bar")

    ' Menübefehl "Ansicht" einblenden
    cmb.FindControl(ID:=30004).Visible = True

    Set cmb = Nothing
End Sub
```

HINWEIS In `Workbook_BeforeClose` im Modul `DieseArbeitsmappe` wird der Menübefehl `Ansicht` zusätzlich wieder aktiviert und eingeblendet. Auf diese Weise wird sichergestellt, dass der Menübefehl in jedem Fall beim Verlassen der Mappe wieder verfügbar ist.

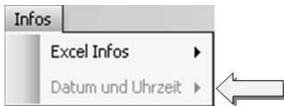


Die obigen Beispiele befinden sich auf der CD-ROM zum Buch im Ordner `\Buch\Kap20`. Die Mappe nennt sich `Bsp20_09.xls`. Die Prozeduren sind im Modul `mdl_01_Control` untergebracht.

Untermenüs deaktivieren

Wenn Sie einen Untermenübefehl deaktivieren möchten, geben Sie zuerst die Menüleiste an, dann den Menübefehl und schließlich das Untermenü. Menübefehl und Untermenüs werden gleichermaßen über die Eigenschaft `Controls` angesprochen.

Bild 20.11 Ausgeblendeter Untermenübefehl

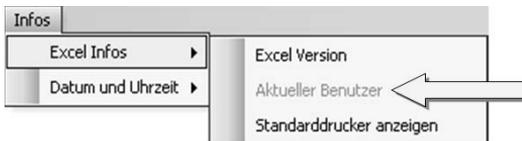


Listing 20.25 Ein Untermenü deaktivieren

```
Sub DeactivateSubmenu1()
    Application.CommandBars("Worksheet Menu Bar"). _
        Controls("Infos"). _
            Controls("Datum und Uhrzeit"). _
                Enabled = False
End Sub
```

Mittels der oben genannten Methode können Sie problemlos auch einen Befehl ausblenden, der sich in einer tieferen Ebene befindet.

Bild 20.12 Ausgeblendeter Untermenübefehl



Listing 20.26 Einen Untermenübefehl ausblenden

```
Sub DeactivateSubmenu2()
    Application.CommandBars("Worksheet Menu Bar"). _
        Controls("Infos"). _
            Controls("Excel Infos"). _
                Controls("Aktueller Benutzer") _
                    .Enabled = False
End Sub
```

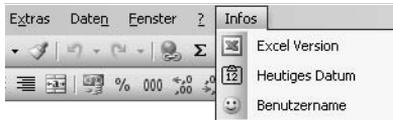


Die obigen Beispiele befinden sich auf der CD-ROM zum Buch im Ordner `\Buch\Kap20`. Die Mappe nennt sich `Bsp20_10.xls`. Die Prozeduren sind im Modul `mdl_01_SubMenu` untergebracht.

Menüelemente mit Symbolen

In Excel steht eine Unmenge von Symbolen zur Verfügung, die Menüelementen vorangestellt werden können. Jedes der Symbole hat eine eigene Identifikationsnummer. Innerhalb der Prozedur wird die gewünschte Nummer der Methode FaceId zugewiesen.

Bild 20.13 Menübefehle mit vorangestellten Symbolen



HINWEIS Später in diesem Kapitel werden Sie erfahren, wie die Symbole und deren zugehörige Nummer ermittelt werden können.

Listing 20.27 Den Menüelementen ein Symbol voranstellen

```
Sub AddNewMenuWithIcons()
    Dim cmb As CommandBar
    Dim cmbp As CommandBarPopup

    Call DeleteNewMenu

    Set cmb = Application.CommandBars("Worksheet Menu Bar")
    Set cmbp = cmb.Controls.Add(Type:=msoControlPopup, Temporary:=True)
    cmbp.Caption = "Infos"

    With cmbp.Controls.Add
        .FaceId = 263 ' Ein Excel-Symbol
        .Caption = "Excel Version"
        .OnAction = "Version"
    End With

    With cmbp.Controls.Add
        .FaceId = 370 ' Ein Kalendersymbol
        .Caption = "Heutiges Datum"
        .OnAction = "DateToday"
    End With

    With cmbp.Controls.Add
        .FaceId = 59 ' Ein Smiley
        .Caption = "Benutzername"
        .OnAction = "User"
    End With

    Set cmb = Nothing
    Set cmbp = Nothing
End Sub
```



Das obige Beispiel befindet sich auf der CD-ROM zum Buch im Ordner \Buch\Kap20. Die Mappe nennt sich *Bsp20_11.xls*. Die Prozedur ist im Modul *mdl_01 Icons* untergebracht.

Die zugehörigen Prozeduren befinden sich im selben Modul. Es wurde darauf verzichtet, diese hier abzdrukken.

Menübefehle mit Umschaltflächen

Wenn Sie Symbole vor den Menüelementen anzeigen, haben Sie die Möglichkeit, deren Status zu bestimmen. Die Symbole sind damit vergleichbar mit Umschaltflächen. In Excel 2003 werden die Symbole orange dargestellt, wenn sie aktiv sind. In älteren Versionen werden sie vertieft angezeigt.

Bild 20.14 Aktivierte und nicht aktivierte Schaltfläche



Beim Erstellen der Menüelemente wird der Eigenschaft *State* die Konstante *msoButtonUp* zugewiesen. Erst in der Prozedur, die bei Auswahl des Elements aufgerufen wird, wird der Status auf *msoButtonDown* umgestellt.

Listing 20.28 Menübefehl mit Umschaltflächen erstellen

```

Sub AddNewMenuToggleButton()
    Dim cmb As CommandBar
    Dim cmbp As CommandBarPopup

    Call DeleteNewMenu

    Set cmb = Application.CommandBars("Worksheet Menu Bar")
    Set cmbp = cmb.Controls.Add(Type:=msoControlPopup, _
        Temporary:=True)
    cmbp.Caption = "Optionen"

    With cmbp.Controls.Add
        .FaceId = 217 ' Ein Symbol für Gitternetzlinien
        .State = msoButtonUp ' Schaltfläche normal
        .Caption = "Gitternetzlinien"
        .OnAction = "GridelinesOnOff"
    End With

    With cmbp.Controls.Add
        .FaceId = 488 ' Ein Symbol für Registerkarten
        .State = msoButtonUp ' Schaltfläche orange
        .Caption = "Blattregisterkarten"
        .OnAction = "SheetTabsOnOff"
    End With

    Set cmb = Nothing
    Set cmbp = Nothing
End Sub

```

In der Prozedur, die an das Menüelement gebunden ist, muss ebenfalls auf die Menüleiste und deren Elemente referenziert werden. Dies, damit der Status des Elementes verändert werden kann.

In der folgenden Prozedur wird in einer If-Entscheidung geprüft, ob die Gitternetzlinien momentan eingeblendet sind oder nicht. Die Anzeige wird verändert und der Status des Elementes wird umgestellt.

Listing 20.29 Prozedur zum Umschalten zwischen ein- und ausgeblendeten Gitternetzlinien

```
Sub GridelinesOnOff()
    Dim cmb As CommandBar
    Dim cmbc As CommandBarControl

    Set cmb = Application.CommandBars("Worksheet Menu Bar")
    Set cmbc = cmb.Controls("Optionen")

    ' Zustand der Gitternetzlinien abfragen und ändern
    With ActiveWindow
        If .DisplayGridlines = False Then
            .DisplayGridlines = True
            cmbc.Controls("Gitternetzlinien").State = msoButtonUp
        Else
            .DisplayGridlines = False
            cmbc.Controls("Gitternetzlinien").State = msoButtonDown
        End If
    End With

    Set cmb = Nothing
    Set cmbc = Nothing
End Sub
```



Das komplette obige Beispiel befindet sich auf der CD-ROM zum Buch im Ordner `\Buch\Kap20`. Die Mappe nennt sich `Bsp20_12.xls`. Die Prozeduren sind im Modul `mdl_01_Toggle` untergebracht.

Menübefehl mit Hyperlinks

Der Aufbau einer Prozedur, die in der Menüleiste einen Menübefehl mit Hyperlinks erstellt, ist im Grunde genommen identisch mit den vorangegangenen Beispielen. Es sind lediglich Eigenschaften erforderlich, die wir bis jetzt noch nicht benutzt haben.

Bild 20.15 Hyperlinks als Menübefehle



Eine der neuen Eigenschaften nennt sich `HyperlinkType`. Ihr wird die Konstante `msoCommandBarButtonHyperlinkOpen` zugewiesen. Diese veranlasst, dass die Schaltfläche als Hyperlink erkannt wird.

Die zweite Eigenschaft, die hinzukommt, nennt sich `ToolTipText`. Ihr wird die eigentliche Webadresse übergeben.

Diese beiden Angaben reichen aus, um ein Menüelement als Hyperlink zu verwenden. Die Eigenschaft `OnAction` fällt in diesem Fall weg, da keine zusätzliche Prozedur aufgerufen werden muss.

Listing 20.30 Menübefehle mit Hyperlinks

```

Sub WebMail()
    Dim cmb As CommandBar
    Dim cmbp As CommandBarPopup

    Call DeleteNewMenu

    Set cmb = Application.CommandBars("Worksheet Menu Bar")
    Set cmbp = cmb.Controls.Add(Type:=msoControlPopup, _
        Temporary:=True)
    cmbp.Caption = "Internet"

    With cmbp.Controls.Add
        .FaceId = 589
        .Caption = "Office-Help-Desk"
        .HyperlinkType = msoCommandBarButtonHyperlinkOpen
        .ToolTipText = "http://www.jumper.ch"
    End With

    With cmbp.Controls.Add
        .FaceId = 69
        .Caption = "Microsoft Deutschland"
        .HyperlinkType = msoCommandBarButtonHyperlinkOpen
        .ToolTipText = "http://www.microsoft.de"
    End With

    With cmbp.Controls.Add
        .FaceId = 64
        .Caption = "Microsoft Press"
        .HyperlinkType = msoCommandBarButtonHyperlinkOpen
        .ToolTipText = "http://mspress.microsoft.de"
    End With

    Set cmb = Nothing
    Set cmbp = Nothing
End Sub

```



Das obige Beispiel befindet sich auf der CD-ROM zum Buch im Ordner `\Buch\Kap20`. Die Mappe nennt sich `Bsp20_13.xls`. Die Prozedur ist im Modul `mdl_01_WebMail` untergebracht.

Menübefehle über Tastenkombinationen aufrufen

Die meisten integrierten Menübefehle lassen sich über Tastenkombinationen aufrufen. In der bestehenden Menüleiste sind einzelne Buchstaben der Menübefehle unterstrichen dargestellt. Die Tastenkombination setzt sich immer aus der `[Alt]`-Taste und dem unterstrichenen Buchstaben zusammen. Um beispielsweise das Dialogfeld *Öffnen* anzuzeigen, verwenden Sie die Tastenkombination `[Alt]+[D],[F]`.

Bild 20.16 Tastenkombinationen programmieren

Bei eigenen Menübefehlen lässt sich dies sehr einfach umsetzen. Stellen Sie einfach den Buchstaben für die Tastenkombination ein kaufmännisches Und (&) voran. Dies geschieht immer bei der Beschriftung Caption.

Der Menübefehl, der in Bild 20.16 zu sehen ist, lässt sich über die Tastenkombination **[Alt] + [I], [I]** ansteuern.

Listing 20.31 Tastenkombinationen zuweisen

```
Sub ShortCut()
    Dim cmb As CommandBar
    Dim cmbp As CommandBarPopup

    Call DeleteNewMenu

    Set cmb = Application.CommandBars("Worksheet Menu Bar")
    Set cmbp = cmb.Controls.Add(Type:=msoControlPopup, _
        Temporary:=True)
    cmbp.Caption = "&Internet"

    With cmbp.Controls.Add
        .FaceId = 589
        .Caption = "Off&ice-Help-Desk"
        .HyperlinkType = msoCommandBarButtonHyperlinkOpen
        .TooltipText = "http://www.jumper.ch"
    End With

    Set cmb = Nothing
    Set cmbp = Nothing
End Sub
```



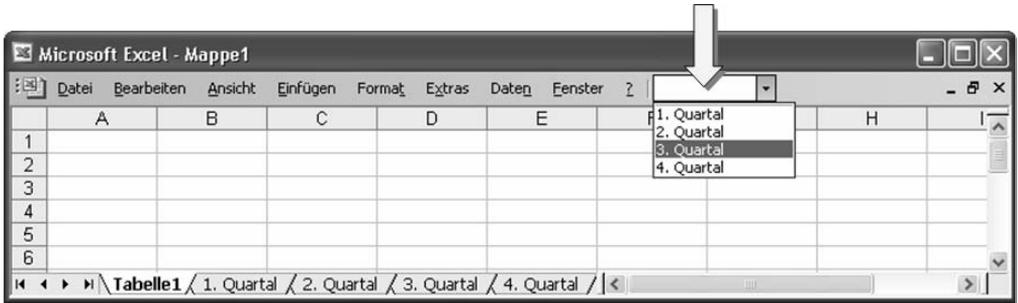
Das obige Beispiel befindet sich auf der CD-ROM zum Buch im Ordner `\Buch\Kap20`. Die Mappe nennt sich `Bsp20_14.xls`. Die Prozedur ist im Modul `mdl_01_ShortCut` untergebracht.

Ein Kombinationsfeld in der Menüleiste erstellen

Anstelle von Menübefehlen können Sie auch Kombinationsfelder in einer Menü- oder Symbolleiste erstellen.

In unserem Beispiel erstellen wir ein Kombinationsfeld an letzter Stelle in der Menüleiste (siehe Bild 20.17). Es soll zum Navigieren innerhalb der Arbeitsmappe dienen. Per Auswahl einer der Einträge im Kombinationsfeld wird zum gleichnamigen Tabellenblatt gesprungen.

Bild 20.17 Ein Kombinationsfeld in der Menüleiste



Die Konstante zum Erstellen eines Kombinationsfeldes lautet `msoControlComboBox`. Um die einzelnen Einträge hinzuzufügen wird jeweils die Methode `AddItem` verwendet. Ihr folgt der Text, der im Kombinationsfeld erscheinen soll.

Caption Um das spätere Ansprechen, zum Beispiel das Löschen des Kombinationsfeldes zu erleichtern, sollten Sie es benennen. Dies geschieht über die Eigenschaft `Caption`.

DropDownLines Über die Eigenschaft `DropDownLines` können Sie festlegen, wie viele Einträge beim Aufklicken des Kombinationsfeldes angezeigt werden sollen. Sollte die Anzahl an Zeilen im Kombinationsfeld größer sein als der Wert, den Sie bei `DropDownLines` festlegen, so wird im Kombinationsfeld eine Bildlaufleiste eingeblendet. Über die Bildlaufleiste können die »versteckten« Zeilen erreicht werden (siehe Bild 20.18).

DropDownWidth Mittels der Eigenschaft `DropDownWidth` können Sie die Breite des aufgeklappten Kombinationsfeldes in Pixel bestimmen.

ListHeaderCount Der Eigenschaft `ListHeaderCount` können Sie einen Wert zuweisen. Dieser gibt an, wie viele Zeilen des Kombinationsfeldes als Überschrift angezeigt werden sollen. Wenn Sie z.B. den Wert 2 eintragen, wird nach dem zweiten Eintrag eine Doppellinie angezeigt.



Listing 20.32 Ein Kombinationsfeld zur Navigation in der Arbeitsmappe

```
Sub AddComboBox()
    ' Kombinationsfeld einfügen
    With Application.CommandBars("Worksheet Menu Bar").Controls. _
        Add(Type:=msoControlComboBox)

        ' -----
        ' Einträge hinzufügen
        ' -----
        .AddItem Text:="1. Quartal"
        .AddItem Text:="2. Quartal"
        .AddItem Text:="3. Quartal"
        .AddItem Text:="4. Quartal"
        ' -----

        ' Einstellungen am Kombinationsfeld
        .Caption = "Monate"
        .DropDownLines = 4
        .DropDownWidth = 100
        .ListHeaderCount = 0
    End With
End Sub
```

Listing 20.32 Ein Kombinationsfeld zur Navigation in der Arbeitsmappe (*Fortsetzung*)

```
.TooltipText = "Tabellenblätter auswählen"
.BeginGroup = True
.OnAction = "ActivateSheets"
End With
End Sub
```

Wir benötigen nun noch die Prozedur, die durch `OnAction` aufgerufen werden soll. Damit per Klick auf die Kombinationsfeldeinträge je eine andere Aktion ausgelöst wird, müssen wir diese über deren `Index` ansprechen (`ListIndex`). In einer Entscheidung prüfen wir, welcher Index aktiviert wurde und selektieren dann das entsprechende Tabellenblatt.

Listing 20.33 Jedem Kombinationsfeldeintrag eine Aktion zuweisen

```
Sub ActivateSheets()
    Dim bytIndex As Byte

    bytIndex = Application.CommandBars("Worksheet Menu Bar"). _
        Controls("Monate").ListIndex

    ' Je nach Auswahl im Kombinationsfeld wird über
    ' eine andere Prozedur aufrufen
    Select Case bytIndex
        Case 1
            Worksheets("1. Quartal").Select
        Case 2
            Worksheets("2. Quartal").Select
        Case 3
            Worksheets("3. Quartal").Select
        Case 4
            Worksheets("4. Quartal").Select
    End Select
End Sub
```



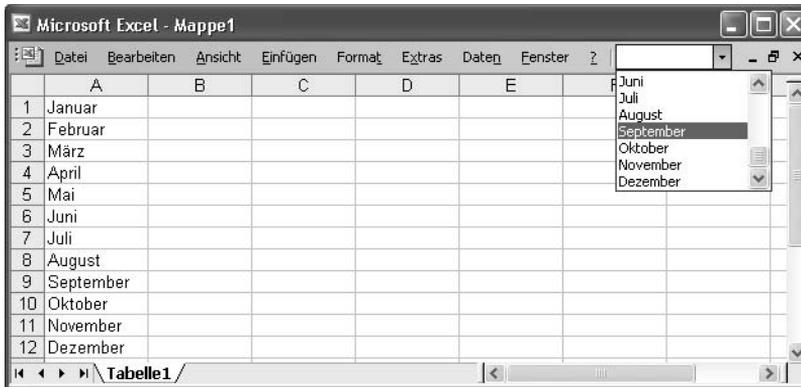
Das obige Beispiel befindet sich auf der CD-ROM zum Buch im Ordner `\Buch\Kap20`. Die Mappe nennt sich `Bsp20_15.xls`. Die Prozedur ist im Modul `mdl_01_Combo` untergebracht.

Kombinationsfeldeinträge aus Zelle beziehen (Schleife)

Sie können Ihre Kombinationsfeldeinträge auch aus Zellen beziehen. Dies ist vor allem dann sinnvoll, wenn es sich um eine umfangreiche Liste handelt. Der Vorteil beim Beziehen der Beiträge aus Zellen besteht darin, dass Sie mit einer Schleife arbeiten können. Das reduziert den VBA-Code.

Nehmen wir an, Sie möchten sämtliche Monatsnamen in einem Kombinationsfeld erscheinen lassen. Tragen Sie diese in die Zellen `A1` bis `A12` ein.

Bild 20.18 Kombinationsfeldeinträge aus Zellen beziehen



Im VBA-Code muss nur der Teil geändert werden, der die Methode AddItem enthält. Wir umgeben diesen mit einer Schleife, die alle Zellen der Spalte A durchläuft, bis die erste leere Zelle gefunden wird.

Listing 20.34 Kombinationsfeldeinträge mittels Schleife aufbereiten

```
Sub AddComboBoxLoop()
    Dim ws As Worksheet
    Dim i As Long

    Set ws = Worksheets(1)

    ' Kombinationsfeld einfügen
    With Application.CommandBars("Worksheet Menu Bar").Controls. _
        Add(Type:=msoControlComboBox)

        ' -----
        ' Einträge aus Zellen beziehen
        ' -----
        i = 1
        Do Until ws.Cells(i, 1) = ""
            .AddItem Text:=ws.Cells(i, 1)
            i = i + 1
        Loop
        ' -----

        ' Einstellungen am Kombinationsfeld
        .Caption = "Monate"
        .DropDownLines = 7
        .DropDownWidth = 100
        .ListHeaderCount = 0
        .TooltipText = "Monate auswählen"
        .BeginGroup = True
        .OnAction = "ActivateSheets"
    End With
End Sub
```



Das komplette obige Beispiel befindet sich auf der CD-ROM zum Buch im Ordner `\Buch\Kap20`. Die Mappe nennt sich `Bsp20_16.xls`. Die Prozedur ist im Modul `mdl_01_Combo` untergebracht.

Das Kombinationsfeld löschen

Um einen Kombinationsfeldeintrag aus der Menüleiste zu löschen, geben Sie den Namen des Controls an. Der Name entspricht dem Text, den Sie beim Erstellen des Kombinationsfeldes der Eigenschaft `Caption` zugewiesen haben.

Listing 20.35 Ein Kombinationsfeld löschen

```
Sub DeleteDropDown()
    Application.CommandBars("Worksheet Menu Bar")._
        Controls("Monate").Delete
End Sub
```

Wenn Sie mehrere Kombinationsfelder in der Menüleiste erstellt haben, so können Sie eine Schleife verwenden, um alle zu löschen. Wichtig dabei ist, dass Sie in einer Entscheidung den Typ auf `msoControlComboBox` überprüfen. Damit stellen Sie sicher, dass wirklich nur Kombinationsfelder und nicht auch benutzerdefinierte Menübefehle gelöscht werden.

Optional können Sie zusätzlich prüfen, ob es sich bei dem Kombinationsfeld um ein integriertes oder benutzerdefiniertes Element handelt. `BuiltIn = False` bedeutet benutzerdefiniert.

Listing 20.36 Alle benutzerdefinierten Kombinationsfelder löschen

```
Sub DeleteAllBuiltInDropDowns()
    Dim ctr As CommandBarControl

    For Each ctr In CommandBars("Worksheet Menu Bar").Controls
        If ctr.Type = msoControlComboBox and ctr.BuiltIn = False Then
            ctr.Delete
        End If
    Next ctr
End Sub
```



Das obige Beispiel befindet sich auf der CD-ROM zum Buch im Ordner `\Buch\Kap20`. Die Mappe nennt sich `Bsp20_16.xls`. Die Prozedur ist im Modul `mdl_01_DeleteCombo` untergebracht.

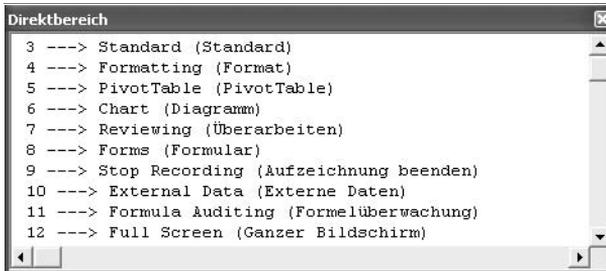
Symbolleisten erstellen und bearbeiten

In Excel gibt es eine ganze Reihe von bestehenden Symbolleisten, die ein- oder ausgeblendet werden können. Wenn Sie einen Teil der integrierten Symbolleisten einsehen möchten, verwenden Sie den Menübefehl *Ansicht/Symbolleisten/Anpassen* und aktivieren die Registerkarte *Symbolleisten*.

Welche Symbolleisten gibt es?

Wenn Sie alle verfügbaren Symbolleisten, bzw. deren Namen ermitteln möchten, verwenden Sie am besten eine geeignete Prozedur. Der nachfolgende Code listet die Symbolleisten im Direktfenster auf.

Bild 20.19 Alle integrierten Befehlsleisten im Direktfenster ausgeben



Über die Eigenschaft `Name` werden alle englischen Namen ausgegeben. Mittels der Eigenschaft `NameLocal` werden die Namen sprachabhängig ermittelt. Das bedeutet, dass die Sprache der installierten Office-Version maßgebend ist.

Um nur Symbolleisten zu ermitteln, ergänzen Sie dem Code eine `If`-Entscheidung, die prüft, ob es sich um den Typ `Normal` (`msoBarTypeNormal`) handelt.

HINWEIS

Eine Alternative zur Auflistung aller Befehlsleisten finden Sie zu Beginn dieses Kapitels im Abschnitt »Befehlsleistentypen per VBA ermitteln«.

Listing 20.37 Index und Name der verfügbare Symbolleisten ermitteln

```
Sub CommandBarNamesTypeNormal()
    Dim i As Integer

    For i = 1 To Application.CommandBars.Count
        If CommandBars(i).Type = msoBarTypeNormal Then
            Debug.Print i & " ---> " &
                Application.CommandBars(i).Name &
                " (" & Application.CommandBars(i).NameLocal & ")"
        End If
    Next i
End Sub
```



Das obige Beispiel befindet sich auf der CD-ROM zum Buch im Ordner `\Buch\Kap20`. Die Mappe nennt sich `Bsp20_17.xls`. Die Prozedur ist im Modul `mdl_01_GetNames` untergebracht.

Symbolleistenbefehle ermitteln

Nachdem Sie die Namen der Symbolleisten kennen, können Sie deren Symbolleistenbefehle ermitteln. Diese zu kennen ist unerlässlich, da sie in späteren Codes wieder verwendet werden.

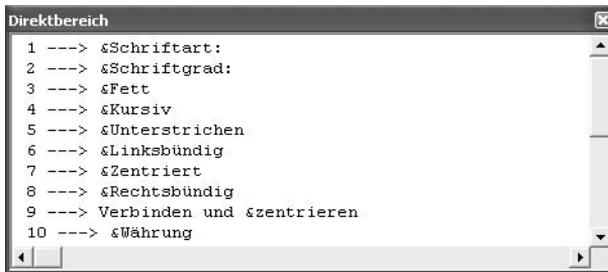
Listing 20.38 Befehle einer Symbolleiste ermitteln

```

Sub ItemIndexName()
    Dim i As Integer

    With Application.CommandBars("Formatting")
        For i = 1 To .Controls.Count
            Debug.Print i & " ---> " & _
                .Controls(i).Caption
        Next i
    End With
End Sub

```

Bild 20.20 Befehle der Symbolleiste *Format*

Das obige Beispiel befindet sich auf der CD-ROM zum Buch im Ordner `\Buch\Kap20`. Die Mappe nennt sich `Bsp20_17.xls`. Die Prozedur ist im Modul `mdl_01_GetNames` untergebracht.

Eine Symbolleiste ein- oder ausblenden

Um eine bestehende Symbolleiste per VBA ein- oder auszublenden, verwenden Sie das Objekt `Application`, gefolgt vom Objekt `CommandBar`, sowie dem Index oder Namen der Symbolleiste. Der Eigenschaft `Visible` weisen Sie den Wert `True` zu, wenn die Symbolleiste eingeblendet werden soll.

Listing 20.39 Symbolleiste über den Index einblenden

```

Sub ShowStandardIndex
    Application.CommandBars(3).Visible = True
End Sub

```

Listing 20.40 Symbolleiste über den Namen einblenden

```

Sub ShowStandardName
    Application.CommandBars("Standard").Visible = True
End Sub

```

Verwenden Sie `False`, um die Symbolleiste auszublenden.

Listing 20.41 Symbolleiste über den Index ausblenden

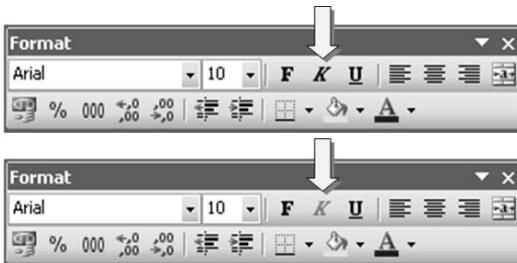
```
Sub ShowStandardIndex
    Application.CommandBars(3).Visible = False
End Sub
```

Listing 20.42 Symbolleiste über den Namen ausblenden

```
Sub ShowStandardName
    Application.CommandBars("Standard").Visible = False
End Sub
```

Einen Symbolleistenbefehl deaktivieren oder ausblenden

Integrierte Symbolleistenbefehle sind in der Regel automatisch inaktiv, wenn kein Objekt dazu vorhanden ist, das momentan damit verändert werden kann.

Bild 20.21 Symbolleistenbefehle aktivieren oder deaktivieren

Per VBA haben Sie die Möglichkeit Symbolleistenbefehle explizit zu deaktivieren. Geben Sie dazu den Namen oder Index der Symbolleiste und des Befehls an, gefolgt von `Enabled = False`.

Listing 20.43 Einen Symbolleistenbefehl deaktivieren

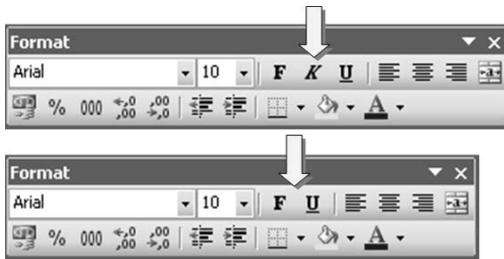
```
Sub DeactivateItem()
    Application.CommandBars("Formatting")._
        Controls("&Kursiv").Enabled = False
End Sub
```

Um einen Symbolleistenbefehl wieder zu aktivieren, ersetzen Sie den booleschen Wert `False` durch `True`.

Listing 20.44 Einen Symbolleistenbefehl aktivieren

```
Sub ActivateItem()
    Application.CommandBars("Formatting")._
        Controls("&Kursiv").Enabled = True
End Sub
```

Bild 20.22 Einen Symbolleistenbefehl ein- oder ausblenden



Anstatt einen Symbolleistenbefehl zu deaktivieren können Sie ihn auch gänzlich ausblenden. Ersetzen Sie dabei die Eigenschaft `Enabled` durch `Visible`.

Listing 20.45 Einen Symbolleistenbefehl ausblenden

```
Sub FadeOutItem()
    Application.CommandBars("Formatting")._
        Controls("&Kursiv").Visible = False
End Sub
```

Listing 20.46 Einen Symbolleistenbefehl einblenden

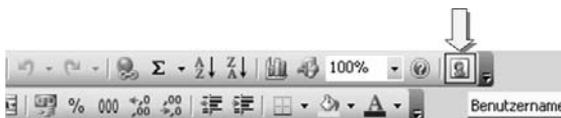
```
Sub FadeInItem()
    Application.CommandBars("Formatting")._
        Controls("&Kursiv").Visible = True
End Sub
```

Eine integrierte Symbolleiste ergänzen

Das Programmieren von Symbolleisten verhält sich im Grunde genommen ähnlich wie das Programmieren von Menüleisten. Der Unterschied besteht zum Teil im Gebrauch der Objekte. So findet beispielsweise das Objekt `ActiveMenuBar` keine Verwendung, denn es bezieht sich nur auf die aktive Menüleiste.

Eine Symbolleiste wird über das Objekt `CommandBar` angesprochen und die darin enthaltenen Schaltflächen über das Objekt `CommandBarButton`.

Bild 20.23 Ein Symbol in die bestehende Symbolleiste *Standard* einfügen



In der folgenden Prozedur wird temporär ein Symbol in der Symbolleiste *Standard* erzeugt. Temporär bedeutet auch hier, dass die Symbolleiste nach dem Beenden der Applikation wieder gelöscht wird.

Der Name, der über `Caption` zugewiesen wird, erscheint nicht als Text in der Symbolleiste, sondern wird lediglich in einer `QuickInfo` angezeigt, wenn ohne zu Klicken der Mauszeiger eine Weile auf dem Symbol verbleibt.

Mittels der Eigenschaft `BeginGroup` wird eine Trennlinie vor dem Symbol eingefügt. Der Rest der Prozedur verhält sich genauso wie bei den Menüleisten.

Listing 20.47 Die Symbolleiste *Standard* ergänzen

```
Sub AddItemInStandard ()
    Dim cmb As CommandBar
    Dim cmbb As CommandBarButton

    Set cmb = Application.CommandBars("Standard")
    Set cmbb = cmb.Controls.Add(Type:=msoControlButton, _
        Temporary:=True)

    With cmbb
        .Caption = "Benutzername"      ' QuickInfo
        .BeginGroup = True             ' Trennlinie
        .FaceId = 682                  ' Symbol
        .OnAction = "UserName"         ' Prozeduraufruf
    End With

    Set cmb = Nothing
    Set cmbb = Nothing
End Sub
```

Listing 20.48 Die zugehörige Prozedur

```
Sub UserName()
    MsgBox Application.UserName
End Sub
```



Das obige Beispiel befindet sich auf der CD-ROM zum Buch im Ordner `\Buch\Kap20`. Die Mappe nennt sich `Bsp20_18.xls`. Die Prozeduren sind im Modul `mdl_01_AddItem` untergebracht.

Einen benutzerdefinierten Symbolleistenbefehl löschen

Benutzerdefinierte Symbolleistenbefehle können, im Gegensatz zu Integrierten, wieder gelöscht werden. Verwenden Sie dazu die Methode `Delete`.

```
Application.CommandBars("Standard").Controls("Benutzername").Delete
```

Die folgende Prozedur enthält neben dem Löschen des Symbolleistenbefehls eine Fehlerbehandlung. In der Schleife werden alle Symbolleistenbefehle der Symbolleiste *Standard* durchlaufen. Sobald ein Element die Beschriftung »Benutzername« aufweist, wird es gelöscht. Ohne die Fehlerbehandlung würde das Debugger-Fenster angezeigt, wenn der Versuch unternommen würde, einen nicht existierenden Befehl zu löschen.

Listing 20.49 Einen Symboleistenbefehl löschen

```

Sub DeleteItemInStandard()
    Dim i As Integer

    With Application.CommandBars("Standard")
        For i = 1 To .Controls.Count
            If .Controls(i).Caption = "Benutzername" Then
                .Controls(i).Delete
            End If
        Next i
    End With
End Sub

```



Das obige Beispiel befindet sich auf der CD-ROM zum Buch im Ordner `\Buch\Kap20`. Die Mappe nennt sich `Bsp20_18.xls`. Die Prozedur ist im Modul `mdl_02_DeleteItem` untergebracht.

Eine Symboleiste zurücksetzen

Genauso wie Menüleisten können auch Symboleisten mittels der `Reset`-Methode zurückgesetzt werden.

ACHTUNG Durch das Zurücksetzen von Symboleisten werden alle manuellen Anpassungen entfernt!

In der nachfolgenden Prozedur wird lediglich die Symboleiste *Standard* in den Originalzustand zurückversetzt.

Listing 20.50 Symboleiste *Standard* zurücksetzen

```

Sub ResetCommandBar()
    Application.CommandBars("Standard").Reset
End Sub

```

Wenn Sie sämtliche Symboleisten zurücksetzen möchten, verwenden Sie eine Schleife. Wenn Sie mit einer `For Each`-Schleife arbeiten, müssen Sie `On Error Resume Next` verwenden, denn es können nicht alle Symboleisten zurückgesetzt werden. Beispielsweise wird beim Versuch, eine selbst erstellte Symboleiste zurückzusetzen, der Debugger gestartet. Die Symboleiste *EuroPlaceholder_2000* verursacht ebenfalls einen Fehler.

Listing 20.51 Sämtliche Symboleisten zurücksetzen

```

Sub ResetAllCommandBars1()
    Dim cmb As CommandBar

    On Error Resume Next
    For Each cmb In Application.CommandBars
        cmb.Reset
    Next cmb
End Sub

```

Die Anweisung `On Error Resume Next` zu umgehen erweist sich in diesem Falle als nicht ganz einfach. Sie könnten eine Zählschleife verwenden und zusätzlich in einer `If`-Entscheidung die Symbolleisten abfangen, die Probleme verursachen. Dazu müssen Sie allerdings den Index oder die Namen der Symbolleisten kennen, die den Fehler erzeugen. Die folgende Prozedur muss deshalb je nach Version, die Sie im Einsatz haben, angepasst werden.

Listing 20.52 Sämtliche Symbolleisten zurücksetzen, mit Fehlerbehandlung

```
Sub ResetAllCommandBars2()
    Dim i As Integer

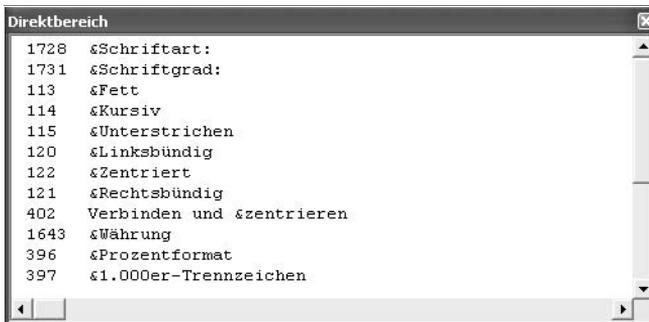
    For i = 1 To Application.CommandBars.Count
        If i <> 27 And i < 122 Then
            Application.CommandBars(i).Reset
        End If
    Next i
End Sub
```

Eine eigene Symbolleiste erzeugen

Wenn Sie eine neue eigene Symbolleiste erzeugen möchten, können Sie bestehende Excel-Befehle einbinden oder eigene erzeugen und diesen eine Prozedur zuweisen. Mit Excel-Befehlen sind hier Schaltflächen gemeint, die auf integrierten Symbolleisten vorhanden sind. Um Excel-eigene Schaltflächen verwenden zu können, müssen Sie deren Identifikationsnummer (ID) kennen.

In Excel gibt es über 100 Symbolleisten, wobei in jeder Symbolleiste mehrere Symbole enthalten sind. Es würde zu weit führen, diese hier alle aufzuführen. Um die gewünschte Identifikationsnummer zu ermitteln, verwenden Sie am besten eine einfache Prozedur, die eine `For`-Schleife enthält.

Bild 20.24 ID bestehender Schaltflächen



Geben Sie in der `With`-Anweisung den Namen der Symbolleiste ein, deren Schaltflächen-IDs Sie ermitteln möchten. Im Direktfenster wird sowohl die Identifikationsnummer als auch der Name des Symbols ausgegeben.

Listing 20.53 ID ermitteln

```
Sub GetID()
    Dim i As Integer

    With Application.CommandBars("Formatting")
        For i = 1 To .Controls.Count
            Debug.Print .Controls(i).ID & Chr(9) & .Controls(i).Caption
        Next i
    End With
End Sub
```



Das obige Beispiel befindet sich auf der CD-ROM zum Buch im Ordner `\Buch\Kap20`. Die Mappe nennt sich `Bsp20_19.xls`. Die Prozedur ist im Modul `mdl_01_GetID` untergebracht.

Um eine neue Symbolleiste zu erzeugen, wird zu Beginn der Prozedur das Objekt `CommandBar` als Variable `cmb` deklariert und anschließend referenziert. Noch vor der Referenzierung wird über `Call` eine Prozedur aufgerufen, die eine bereits vorhandene Symbolleiste mit dem Namen *Meine Symbolleiste* löscht. Auf diese Weise wird verhindert, dass bei mehrfachem Aufruf der Prozedur der Debugger gestartet wird. Die Löschroutine werden wir uns im nächsten Schritt ansehen.

Bild 20.25 Eine selbst definierte Symbolleiste



In der äußeren `With`-Anweisung wird auf das Objekt `CommandBar` verwiesen. In der ersten darin enthaltenen `With`-Anweisung werden drei Excel-eigene Symbole erzeugt. Das Zuweisen einer Prozedur ist in diesem Fall nicht erforderlich, da automatisch auf Excel-interne Befehle zugegriffen wird, die den Symbolen standardmäßig zugewiesen sind.

In den darauf folgenden drei `With`-Anweisungen werden eigene Symbole erzeugt. Mittels `BeginGroup` wird eine Trennlinie eingefügt. Die restlichen Eigenschaften entsprechen denen, die wir bereits beim Erstellen von Menübefehlen verwendet haben.

Listing 20.54 Eine eigene Symbolleiste erstellen

```
Sub CreateNewCommandBar()
    Dim cmb As CommandBar

    Call DeleteNewCommandBar

    Set cmb = Application.CommandBars.Add(Name:="Meine Symbolleiste")

    With cmb
        ' Excel-eigene Symbole einbinden
        With .Controls
            .Add Type:=msoControlButton, ID:=113 ' Fett
            .Add Type:=msoControlButton, ID:=114 ' Kursiv
            .Add Type:=msoControlButton, ID:=115 ' Unterstrichen
        End With

        ' Eigene Symbole erzeugen
        With .Controls.Add(Type:=msoControlButton)
```

Listing 20.54 Eine eigene Symbolleiste erstellen (*Fortsetzung*)

```

        .BeginGroup = True           ' Trennlinie einfügen
        .Caption = "Benutzername"    ' QuickInfo
        .FaceId = 59                 ' Symbol Smiley
        .OnAction = "UserName"       ' Prozedur
    End With

    With .Controls.Add(Type:=msoControlButton)
        .Caption = "Aktuelle Uhrzeit" ' QuickInfo
        .FaceId = 33                 ' Symbol Uhr
        .OnAction = "TimeNow"        ' Prozedur
    End With

    With .Controls.Add(Type:=msoControlButton)
        .Caption = "Heutiges Datum"  ' QuickInfo
        .FaceId = 370                ' Symbol Datum
        .OnAction = "DateToday"      ' Prozedur
    End With

    ' Symbolleiste anzeigen
    .Visible = True
    End With

    Set cmb = Nothing
End Sub

```

Listing 20.55 Zugehörige Prozeduren

```

Sub UserName()
    MsgBox Application.UserName
End Sub

Sub TimeNow()
    MsgBox Time
End Sub

Sub DateToday()
    MsgBox Date
End Sub

```



Das obige Beispiel befindet sich auf der CD-ROM zum Buch im Ordner `\Buch\Kap20`. Die Mappe nennt sich `Bsp20_19.xls`. Die Prozeduren sind im Modul `mdl_02_CreateCommandbar` untergebracht.

Eine benutzerdefinierte Symbolleiste löschen

Im zuvor behandelten Beispiel wurde zu Beginn des Codes mit `Call` die nachfolgend beschriebene Prozedur `DeleteNewCommandBar` aufgerufen. Die Prozedur löscht eine selbst erstellte Symbolleiste.

In der `For`-Schleife werden sämtliche vorhandenen Symbolleisten durchlaufen. In der `If`-Entscheidung wird der Name der Symbolleiste hinterlegt, die gelöscht werden soll. Sobald in der `For`-Schleife eine Symbolleiste gefunden wird, die mit dem Namen übereinstimmt, wird sie gelöscht.

HINWEIS Symbolleisten, die standardmäßig durch Excel zur Verfügung stehen, können nicht gelöscht werden. Beim Versuch, eine solche Symbolleiste zu löschen, wird der Debugger gestartet.

Listing 20.56 Eine selbst erstellte Symbolleiste löschen

```
Sub DeleteNewCommandBar()
    Dim cmb As CommandBar

    For Each cmb In Application.CommandBars
        If cmb.Name = "Meine Symbolleiste" Then
            cmb.Delete
        End If
    Next cmb
End Sub
```



Das obige Beispiel befindet sich auf der CD-ROM zum Buch im Ordner `\Buch\Kap20`. Die Mappe nennt sich `Bsp20_19.xls`. Die Prozedur ist im Modul `mdl_03_DeleteCommandbar` untergebracht.

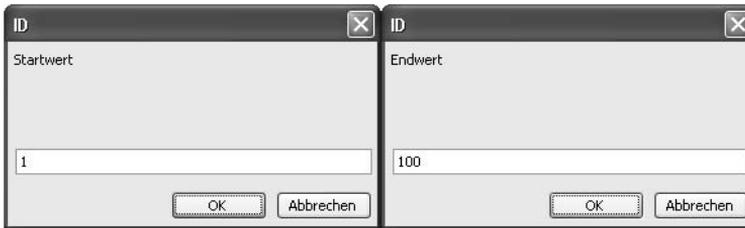
Welche Symbole gibt es?

Es gibt eine Unmenge von Symbolen, die als Schaltflächen verwendet werden können. Da es sich um mehrere Hundert handelt, können diese hier nicht alle aufgeführt werden. Dennoch ist es hilfreich, die Symbole und deren ID sehen zu können. Wir erstellen zu diesem Zweck eine Symbolleiste, die beides anzeigt.

Bild 20.26 Symbole und deren ID



Da unmöglich alle Symbole in einer Symbolleiste aufgeführt werden können, werden beim Aufrufen der Prozedur zwei Eingabefelder `InputBox` eingeblendet. Im ersten Eingabefeld kann ein Startwert eingegeben werden und im zweiten ein Endwert. Es sollten nicht mehr als maximal 200 Symbole ausgewählt werden. Die Rechenzeit der Prozedur würde sonst zu lange dauern.

Bild 20.27 Start- und Endwert eingeben


Die Prozedur ist so aufgebaut, dass sie mehrmals ausgeführt werden kann. Zu Beginn der Prozedur wird die alte Symbolleiste gelöscht. Danach erfolgt die Abfrage für die zwei Eingabefelder. Beide Werte werden an eine Variable übergeben und in der For-Schleife als Zähler verwendet.

Innerhalb der Schleife werden pro Durchlauf zwei Symbole kreiert. Das Erste erzeugt das Symbol und das Zweite die zugehörige Nummer. Damit ein Symbol und deren ID als Block dargestellt werden, wird jeweils eine Trennlinie (BeginGroup) eingefügt.

Es wird zudem über Height die Höhe der Symbolleiste auf 320 Pixel festgelegt. Damit wird erreicht, dass die Symbolleiste aus mehreren Zeilen besteht.

Listing 20.57 Symbole und deren Nummern ermitteln

```

Sub IconsAndIDs()
    Dim cmb As CommandBar
    Dim intFirst As Integer
    Dim intLast As Integer
    Dim i As Integer

    ' Eventuell vorhandene Symbolleiste löschen
    For Each cmb In Application.CommandBars
        If cmb.Name = "ID & Symbole" Then
            cmb.Delete
        End If
    Next cmb

    ' Anzahl der gewünschten Symbole abfragen
    intFirst = Application.InputBox("Startwert", "ID", 1)
    intLast = Application.InputBox("Endwert", "ID", 100)

    ' Symbolleiste erzeugen
    Set cmb = Application.CommandBars.Add(Name:="ID & Symbole")

    With cmb
        ' Schleife für die gewünschte Anzahl an Symbolen durchlaufen
        For i = intFirst To intLast

            ' Die Symbole einfügen
            With .Controls.Add(msoControlButton)
                .BeginGroup = True
                .FaceId = i
            End With

            ' Die ID einfügen
            With .Controls.Add(msoControlButton)
                .FaceId = 0
            End With
        Next i
    End With
End Sub

```

Listing 20.57 Symbole und deren Nummern ermitteln (Fortsetzung)

```

        .Caption = i
        .Style = msoButtonWrapCaption
    End With
Next i

' Symbolleiste anzeigen und deren Höhe festlegen
.Visible = True
.Height = 320
End With

Set cmb = Nothing
End Sub

```

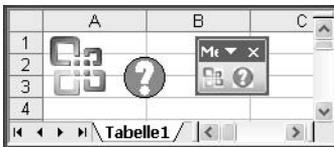


Das obige Beispiel befindet sich auf der CD-ROM zum Buch im Ordner `\Buch\Kap20`. Die Mappe nennt sich `Bsp20_20.xls`. Die Prozedur ist im Modul `mdl_01_Icons` untergebracht.

Eigene Symbole erzeugen

Neben den verfügbaren Standardsymbolen haben Sie zusätzlich die Möglichkeit, eigene Bilder in eine Symbolleiste einzubinden.

Bild 20.28 Eigene Symbole in Symbolleisten einbinden



Das Symbol, das als Schaltfläche in der Symbolleiste erscheinen soll, muss zuvor in die Zwischenablage kopiert werden. Sie können ein beliebiges Bild verwenden, beispielsweise ein Bild aus dem Internet. Mittels der Methode `PasteFace` wird es in die Symbolleiste eingefügt.

In unserem Beispiel wurden die Bilder zuerst ins Tabellenblatt eingefügt. In der Prozedur werden die Bilder kopiert und danach in die Symbolleiste aufgenommen.

Listing 20.58 Symbolleiste mit eigenen Symbolen erzeugen

```

Sub MyIcons ()
    Dim cmb As CommandBar

    Call DeleteCommandBar

    Set cmb = Application.CommandBars.Add(Name:="Meine Symbole")

    With cmb
        ' Eigene Symbole erzeugen
        With .Controls.Add(Type:=msoControlButton)
            ActiveSheet.Pictures(1).Copy ' Erstes Bild kopieren
            .PasteFace ' Bild als Symbol einfügen
            .Caption = "Excel Version"
        End With
    End With
End Sub

```

Listing 20.58 Symbolleiste mit eigenen Symbolen erzeugen (*Fortsetzung*)

```

        .OnAction = "MyVersion"
    End With

    With .Controls.Add(Type:=msoControlButton)
        ActiveSheet.Pictures(2).Copy ' Zweites Bild kopieren
        .PasteFace ' Bild als Symbol einfügen
        .Caption = "Benutzername"
        .OnAction = "MyName"
    End With

    ' Symbolleiste einblenden
    .Visible = True
End With

Set cmb = Nothing
End Sub

```

Listing 20.59 Zugehörige Prozeduren

```

Sub MyVersion()
    MsgBox "Excel " & Application.Version
End Sub

Sub MyName()
    MsgBox Application.UserName
End Sub

```



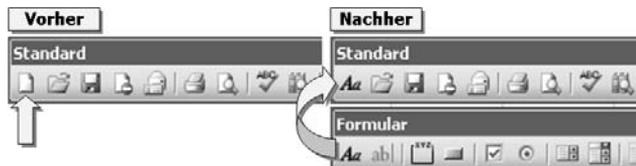
Das obige Beispiel befindet sich auf der CD-ROM zum Buch im Ordner `\Buch\Kap20`. Die Mappe nennt sich `Bsp20_21.xls`. Die Prozeduren sind im Modul `mdl_01_Mylcons` untergebracht.

Symbole ersetzen

Mittels der Methode `CopyFace` können Sie Symbole einer bestehenden Symbolleiste ersetzen.

Das Bild 20.29 zeigt einen Teil der *Standard*-Symbolleiste vor und nach dem Ersetzen des ersten Symbols. Der Befehl *Neu* der Symbolleiste *Standard* wurde durch das erste Symbol der Symbolleiste *Formular* ersetzt. Der Befehl *Neu* hat lediglich das neue Bild übernommen. Seine Funktion, eine neue Mappe zu erzeugen, hat er beibehalten.

Bild 20.29 *Standard*-Symbolleiste vorher und nachher



Um das Ganze umzusetzen, sind lediglich zwei Codezeilen erforderlich. In der ersten Codezeile wird angegeben, welches Symbol kopiert werden soll (`CopyFace`). Dabei werden der Name der Symbolleiste sowie die Position des zu kopierenden Symbols angegeben.

Über die zweite Codezeile wird festgelegt, welches Symbol ersetzt werden soll (PasteFace). Es wird ebenfalls Bezug genommen auf den Namen der Symbolleiste sowie den Index des zu ersetzenden Symbols.

Listing 20.60 Symbole ersetzen

```
Sub ReplaceIcon()
    ' Symbol kopieren
    CommandBars("Forms").Controls(1).CopyFace

    ' Symbol ersetzen
    CommandBars("Standard").Controls(1).PasteFace
End Sub
```



Das obige Beispiel befindet sich auf der CD-ROM zum Buch im Ordner `\Buch\Kap20`. Die Mappe nennt sich `Bsp20_22.xls`. Die Prozedur ist im Modul `mdl_01_CopyIcons` untergebracht.

Symbolleisten positionieren

Beim Erstellen einer neuen Symbolleiste können Sie festlegen, wo diese erscheinen soll. Sie können eine Symbolleiste am rechten, linken, oberen oder unteren Rand der Applikation andocken. Des Weiteren haben Sie die Möglichkeit, die Symbolleiste frei schwebend einzublenden. Letzteres ist die Standardeinstellung.

Beim Erzeugen der Symbolleiste `Add` wird dabei das Argument `Position`, gefolgt von einer Konstanten, übergeben.

Tabelle 20.4 Konstanten zur Positionierung von Symbolleisten

Beschreibung	Konstante	Index
Am linken Bildschirmrand	<code>msoBarLeft</code>	0
Am oberen Bildschirmrand	<code>msoBarTop</code>	1
Am rechten Bildschirmrand	<code>msoBarRight</code>	2
Am unteren Bildschirmrand	<code>msoBarBottom</code>	3
Frei stehend (Standard)	<code>msoBarFloating</code>	4

Listing 20.61 Symbolleiste am linken Rand erscheinen lassen

```
Sub CommandBarPosition()
    Dim cmb As CommandBar
    Call DeleteCommandBar

    ' Die Position mit angeben
    Set cmb = Application.CommandBars.Add(Name:="Meine Symbolleiste", _
        Position:=msoBarLeft)

    With cmb.Controls.Add(Type:=msoControlButton)
        .FaceId = 64
        .OnAction = "MyName"
    End With
End Sub
```

Listing 20.61 Symbolleiste am linken Rand erscheinen lassen (Fortsetzung)

```
End With

cmb.Visible = True
Set cmb = Nothing
End Sub
```

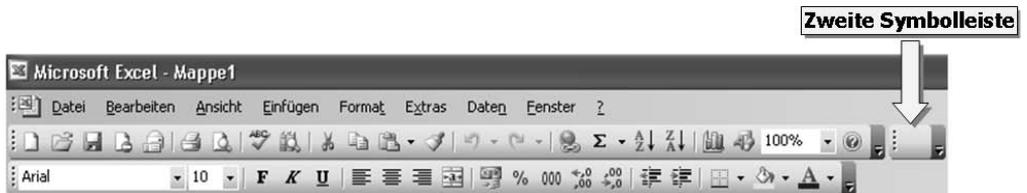


Das obige Beispiel befindet sich auf der CD-ROM zum Buch im Ordner `\Buch\Kap20`. Die Mappe nennt sich `Bsp20_23.xls`. Die Prozedur ist im Modul `mdl_01_Position` untergebracht.

Zwei Symbolleisten nebeneinander anordnen

Sie können zwei Symbolleisten nebeneinander platzieren, sofern sie am oberen Rand der Applikation angedockt sind.

Bild 20.30 Zwei Symbolleisten nebeneinander platzieren



In der folgenden Prozedur wird eine neue Symbolleiste erzeugt. Zuvor wird sie gelöscht, sofern bereits vorhanden. Dies, damit der Code fehlerfrei mehrmals ausgeführt werden kann. Die Anweisung `On Error Resume Next` verwenden wir hier, damit der Code nicht unnötig kompliziert wird.

Mittels der Eigenschaft `Position` wird festgelegt, dass die neue Symbolleiste am selben Ort erscheinen soll wie die bestehende. Hier somit am oberen Rand der Applikation.

Über die Eigenschaft `RowIndex` wird die Andockreihenfolge der bestehenden Symbolleiste übernommen. Sie befindet sich unterhalb der Titelleiste an erster Stelle.

Der Eigenschaft `Left` (links) der neuen Symbolleiste wird die Breite (`Width`) der bestehenden Symbolleiste übergeben. Damit wird die neue Symbolleiste rechts neben der bestehenden eingefügt. Wenn Sie diese Codezeile weglassen, wird die neue Symbolleiste linkerhand der bestehenden erscheinen.

Listing 20.62 Eine neue Symbolleiste neben einer bestehenden einfügen

```
Sub SideBySide()
    Dim cmb As CommandBars
    Set cmb = Application.CommandBars

    ' Eventuell bestehende Symbolleiste löschen
    On Error Resume Next
    cmb("Meine Symbolleiste").Delete
    On Error GoTo 0

    ' Neue (leere) Symbolleiste erstellen
```

Listing 20.62 Eine neue Symboleiste neben einer bestehenden einfügen (Fortsetzung)

```
cmb.Add Name:="Meine Symboleiste"
cmb("Meine Symboleiste").Visible = True

' Neue Symboleiste neben "Standard" platzieren
With cmb("Meine Symboleiste")
    .Position = cmb("Standard").Position
    .RowIndex = cmb("Standard").RowIndex
    .Left = cmb("Standard").Width
End With

Set cmb = Nothing
End Sub
```



Das obige Beispiel befindet sich auf der CD-ROM zum Buch im Ordner `\Buch\Kap20`. Die Mappe nennt sich `Bsp20_24.xls`. Die Prozedur ist im Modul `mdl_01_SideBySide` untergebracht.

Symboleisten mit Text

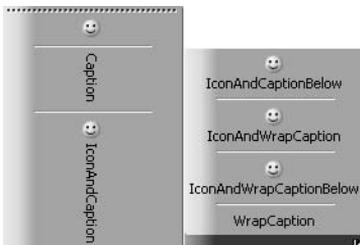
Wenn Sie an Stelle eines Symbols lieber den Text auf der Symbolschaltfläche erscheinen lassen möchten, müssen Sie eine weitere Eigenschaft verwenden. Die Eigenschaft nennt sich `Style`. Ihr können verschiedene Konstanten zugewiesen werden. Um nur den Text anzuzeigen verwenden Sie die Konstante `msoButtonCaption`. Wenn Sie den Text und ein Symbol anzeigen möchten, verwenden Sie die Konstante `msoButtonIconAndCaptionBelow`. Der Tabelle 20.5 können Sie alle verfügbaren Konstanten entnehmen.

HINWEIS Wenn Sie die Symboleiste am linken oder rechten Rand der Applikation andocken, haben einzelne Konstanten einen anderen Effekt. Beispielsweise wird der Text von oben nach unten geschrieben, also vertikal. Sie können u.a. die Konstante `msoButtonWrapCaption` verwenden, um den Text zu drehen.

Bild 20.31 Symboleiste oben andockt oder frei stehend



Bild 20.32 Symboleiste links oder rechts andockt



In der dritten Spalte der Tabelle 20.5 sind die Konstanten der Eigenschaft `Style` aufgeführt. Den Konstanten muss jeweils die Zeichenfolge `msoButton` vorangestellt werden. Aus Platzgründen wurde hier nur der veränderliche Teil aufgeführt.

Tabelle 20.5 *Style*-Konstanten

Symboleiste oben, unten oder freistehend	Symboleiste links oder rechts	Konstante (msoButton)	Index
Nur Symbol (Standard)	Nur Symbol (Standard)	<code>Automatic</code>	0
Nur Text, horizontal	Nur Text, vertikal	<code>Caption</code>	2
Symbol links und Text horizontal	Symbol oben und Text vertikal	<code>IconAndCaption</code>	3
Symbol oben und Text horizontal	Symbol oben und Text horizontal	<code>IconAndCaptionBelow</code>	11
Symbol links und Text horizontal	Symbol oben und Text horizontal	<code>IconAndWrapCaption</code>	7
Symbol oben und Text horizontal	Symbol oben und Text horizontal	<code>IconAndWrapCaptionBelow</code>	15
Nur Text, horizontal	Nur Text, horizontal	<code>WrapCaption</code>	14

Listing 20.63 Eine Schaltfläche mit Symbol und Text

```

Sub WithText()
    Dim cmb As CommandBar
    Call DeleteCommandBar

    Set cmb = Application.CommandBars.Add(Name:="msoButton", _
                                         Position:=msoBarFloating)

    With cmb
        ' Eine Schaltfläche mit Text
        With .Controls.Add(Type:=msoControlButton)
            .Caption = "IconAndWrapCaption "
            .Style = msoButtonIconAndWrapCaption
            .FaceId = 59
            .OnAction = "MyName"
        End With

        .Visible = True
    End With

    Set cmb = Nothing
End Sub

```



Das obige Beispiel befindet sich auf der CD-ROM zum Buch im Ordner `\Buch\Kap20`. Die Mappe nennt sich `Bsp20_25.xls`. Die Prozedur ist im Modul `mdl_01_Text` untergebracht.

Ein Symbolleistenbefehl mit Unterbefehlen erzeugen

Das Verfahren, um eine Symbolleiste mit einem Befehl zu erstellen, der Untermenüs aufweist, ist dem der Menüleiste sehr ähnlich.

Bild 20.33 Symbolleistenbefehl mit Unterbefehlen



Der grundlegende Unterschied besteht darin, dass die Symbolleiste zuerst erzeugt werden muss. Die Beschreibungen der Codeblöcke sind als Kommentare in der folgenden Prozedur untergebracht.

Listing 20.64 Eine Symbolleiste mit Untermenüs erzeugen

```

Sub NewCommandBarSubMenu()
    Dim cmb As CommandBar
    Dim cmbp As CommandBarPopup
    Dim cmbpSub As CommandBarPopup

    ' Eventuell bestehende Symbolleiste löschen
    On Error Resume Next
    Application.CommandBars("Meine Symbolleiste").Delete
    On Error GoTo 0

    ' Neue Symbolleiste erzeugen
    Set cmb = Application.CommandBars.Add(Name:="Meine Symbolleiste")

    ' Unterbefehl erzeugen
    Set cmbp = cmb.Controls.Add(Type:=msoControlPopup)
    cmbp.Caption = "Infos"

    ' Symbolleitenbefehl mit Unterbefehlen
    With cmb

        Set cmbpSub = cmbp.Controls.Add(Type:=msoControlPopup)
        cmbpSub.Caption = "Excel Infos"
        ' Erster Untermenübefehl
        With cmbpSub.Controls.Add
            .Caption = "Excel Version"
            .OnAction = "Version"
        End With
        ' Zweiter Untermenübefehl
        With cmbpSub.Controls.Add
            .Caption = "Aktueller Benutzer"
            .OnAction = "User"
        End With
        ' Dritter Untermenübefehl
        With cmbpSub.Controls.Add
            .Caption = "Standarddrucker anzeigen"
            .OnAction = "Printer"
        End With
    End With
End Sub

```

Listing 20.64 Eine Symbolleiste mit Untermenüs erzeugen (Fortsetzung)

```

End With

' Symbolleiste anzeigen
.Visible = True
End With

Set cmb = Nothing
Set cmbp = Nothing
Set cmbpSub = Nothing
End Sub

```



Das komplette obige Beispiel befindet sich auf der CD-ROM zum Buch im Ordner `\Buch\Kap20`. Die Mappe nennt sich `Bsp20_26.xls`. Die Prozedur ist im Modul `mdl_01_SubMenu` untergebracht.

Einen Symbolleistenbefehl mit Hyperlink erstellen

Ein Symbolleistenbefehl kann mit einem Hyperlink verknüpft werden. Das bedeutet, dass beim Klick auf das Symbolleistenelement nicht eine Prozedur gestartet, sondern eine Webseite geöffnet wird. Voraussetzung dafür ist eine bestehende Internetverbindung.

Bild 20.34 Symbolleistenbefehl, der zu einer Webseite führt



An Stelle von `OnAction` wird bei einem Hyperlink die Eigenschaft `HyperlinkType`, gefolgt von der Konstanten `msoCommandBarButtonHyperlinkOpen`, verwendet.

Der URL, also die Webadresse, wird im `TooltipText` angegeben.

Listing 20.65 Symbolleistenbefehl mit Hyperlink erstellen

```

Sub CommandBarHyperlink()
' Allfällig bestehende Symbolleiste löschen
On Error Resume Next
Application.CommandBars("Meine Symbolleiste").Delete
On Error GoTo 0

' Neue Symbolleiste erzeugen
With Application.CommandBars.Add(Name:="Meine Symbolleiste")
' Befehl mit Hyperlink erzeugen
With .Controls.Add
.FaceId = 589
.Caption = "Office-Help-Desk"
.Style = msoButtonIconAndWrapCaption
.HyperlinkType = msoCommandBarButtonHyperlinkOpen
.TooltipText = "http://www.jumper.ch"
End With

' Symbolleiste anzeigen

```

Listing 20.65 Symbolleistenbefehl mit Hyperlink erstellen (Fortsetzung)

```
.Visible = True
End With
End Sub
```



Das obige Beispiel befindet sich auf der CD-ROM zum Buch im Ordner `\Buch\Kap20`. Die Mappe nennt sich `Bsp20_27.xls`. Die Prozedur ist im Modul `mdl_01_Hyperlink` untergebracht.

Ein Kombinationsfeld in eine Symbolleiste einfügen

Auch das Verfahren, um ein Kombinationsfeld in eine Symbolleiste einzufügen, ist dem der Menüleiste sehr ähnlich. Nachfolgend finden Sie ein Beispiel dazu.

Bild 20.35 Ein Kombinationsfeld in einer Symbolleiste



Eine detaillierte Beschreibung zu den Befehlen finden Sie in diesem Kapitel im Abschnitt »Ein Kombinationsfeld in der Menüleiste erstellen«.

Listing 20.66 Ein Kombinationsfeld in eine Symbolleiste einfügen

```
Sub AddComboBox()
    ' Allfällig bestehende Symbolleiste löschen
    On Error Resume Next
    Application.CommandBars("Meine Symbolleiste").Delete
    On Error GoTo 0

    ' Symbolleiste erstellen
    With Application.CommandBars.Add(Name:="Meine Symbolleiste")

        ' Kombinationsfeld einfügen
        With .Controls.Add(Type:=msoControlComboBox)

            ' Einträge hinzufügen
            .AddItem Text:="1. Quartal"
            .AddItem Text:="2. Quartal"
            .AddItem Text:="3. Quartal"
            .AddItem Text:="4. Quartal"

            ' Einstellungen am Kombinationsfeld
            .Caption = "Monate"
            .DropDownLines = 4
            .DropDownWidth = 100
            .ListHeaderCount = 0

            .TooltipText = "Tabellenblätter auswählen"
```

Listing 20.66 Ein Kombinationsfeld in eine Symbolleiste einfügen (Fortsetzung)

```

        .BeginGroup = True
        .OnAction = "ActivateSheets"
    End With

    ' Symbolleiste einblenden
    .Visible = True
    End With
End Sub

```



Das komplette obige Beispiel befindet sich auf der CD-ROM zum Buch im Ordner `\Buch\Kap20`. Die Mappe nennt sich `Bsp20_28.xls`. Die Prozedur ist im Modul `mdl_01_Combo` untergebracht.

Kontextmenüs handhaben

Kontextmenüs werden immer dann eingeblendet, wenn mit der rechten Maustaste auf ein Objekt der Applikation geklickt wird. Der Inhalt des Kontextmenüs hängt davon ab, auf welches Objekt ein Rechtsklick erfolgt ist. Dabei kann es sich um eine Grafik, ein Diagramm, ein Diagrammelement, eine Pivot-Tabelle, das Tabellenblatt usw. handeln.

Die Handhabung von Kontextmenüs erfolgt im Grunde genommen genauso wie die von Befehlsleisten generell. Nachfolgend finden Sie einige Besonderheiten beschrieben. Ansonsten können Sie die Codes von Symbolleisten genauso für Kontextmenüs verwenden. Sie brauchen lediglich den entsprechenden Index oder Namen zu ändern.

Der Tabelle 20.6 können Sie die wichtigsten Kontextmenüs entnehmen, die sich in Bezug auf Mappen und Tabellen verwenden lassen.

Tabelle 20.6 Kontextmenüs in Bezug auf Mappen und Tabellenblätter

Kontextmenüname deutsch	Kontextmenüname englisch
Zelle	Cell
Zeilen	Row
Spalten	Column
Tabellenreiter	Ply
Menü- und Symbolleisten	Toolbar List
Arbeitsmappen-Registerkarte	Workbook tabs
Applikationssymbol oben links	System

Wenn Sie beispielsweise das Zellen-Kontextmenü deaktivieren möchten, verwenden Sie die folgende Codezeile:

```
Application.CommandBars("Cell").Enabled = False
```

Um das Kontextmenü wieder zu aktivieren, ändern Sie den Wert von False auf True und führen die Prozedur nochmals aus.

Welche Kontextmenüs gibt es?

Wenn Sie sämtliche Kontextmenüs ermitteln möchten, die in Excel verfügbar sind, verwenden Sie am besten einen VBA-Code. Die nachfolgende Prozedur gibt alle Kontextmenüs und deren Index im Direktfenster aus.

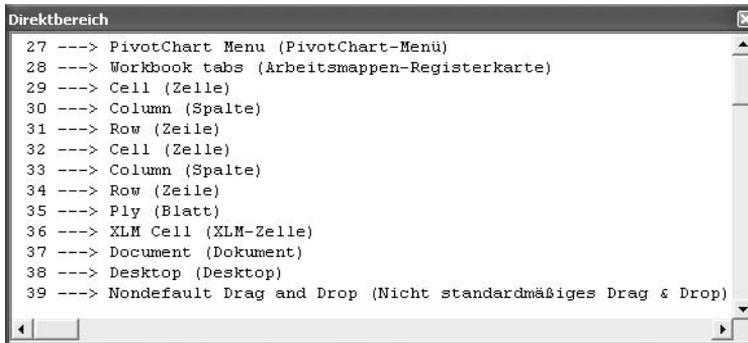
Listing 20.67 Index und Name aller Kontextmenüs ermitteln

```
Sub PopupNames()
    Dim i As Integer

    For i = 1 To Application.CommandBars.Count
        If CommandBars(i).Type = msoBarTypePopup Then
            Debug.Print i & " ---> " &
                Application.CommandBars(i).Name &
                " (" & Application.CommandBars(i).NameLocal & ")"
        End If
    Next i
End Sub
```

HINWEIS Einen Code, um sämtliche Befehlsleisten zu ermitteln, finden Sie zu Beginn dieses Kapitels im Abschnitt »Befehlsleistentypen per VBA ermitteln«.

Bild 20.36 Kontextmenüs und deren Index

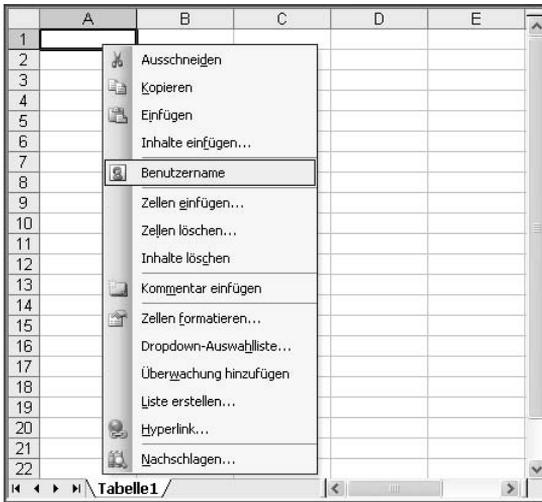


Das obige Beispiel befindet sich auf der CD-ROM zum Buch im Ordner `\Buch\Kap20`. Die Mappe nennt sich `Bsp20_29.xls`. Die Prozedur ist im Modul `mdl_01_PopupNames` untergebracht.

Kontextmenüs ergänzen und zurücksetzen

Das Ergänzen eines Kontextmenüs erfolgt auf gleiche Weise wie bei einer Symbolleiste. In unserem Beispiel wird das Zellenkontextmenü ergänzt.

Bild 20.37 Der Eintrag *Benutzername* wurde hinzugefügt.



Zuerst wird das Kontextmenü als Variable `cmb` deklariert und anschließend referenziert. Direkt nach der Referenzierung wird das Kontextmenü zurückgesetzt. Dies tun wir hier, damit die Prozedur mehrmals ausgeführt werden kann, ohne dass der Befehl mehrfach vorhanden ist.

In einem zweiten Schritt wird das Hinzufügen des Befehls referenziert. Der Befehl soll an fünfter Stelle erscheinen, deshalb `Before:=5`.

In der `With`-Anweisung wird zuerst eine Trennlinie eingefügt und danach die Beschriftung festgelegt. Es wird ein Symbol zugewiesen und schließlich der Name der zu startenden Prozedur übergeben. Wie Sie sehen können, werden exakt dieselben Eigenschaften wie beim Erstellen von Symbolleisten verwendet.

Listing 20.68 Ein Kontextmenü ergänzen

```
Sub AddCommand()
    Dim cmb As CommandBar
    Dim cmbb As CommandBarButton

    Set cmb = Application.CommandBars("Cell")
    cmb.Reset ' Kontextmenü zurücksetzen

    Set cmbb = cmb.Controls.Add(Type:=msoControlButton, _
                                Temporary:=True, Before:=5)

    With cmbb
        .BeginGroup = True ' Trennlinie einfügen
        .Caption = "Benutzername" ' Beschriftung
        .FaceId = 682 ' Symbol
        .OnAction = "UserName" ' Prozeduraufruf
    End With

    Set cmb = Nothing
    Set cmbb = Nothing
End Sub
```



Das obige Beispiel befindet sich auf der CD-ROM zum Buch im Ordner `\Buch\Kap20`. Die Mappe nennt sich `Bsp20_30.xls`. Die Prozedur ist im Modul `mdl_01_AddCommand` untergebracht.

Kontextmenübefehle ausblenden

Um Kontextmenübefehle auszublenden, verwenden Sie die Eigenschaft `Visible` und übergeben ihr den booleschen Wert `False`. Kontextmenübefehle können über deren Index oder Beschriftungsname `Caption` angesprochen werden.

ACHTUNG Achten Sie beim Verwenden von Namen darauf, dass Buchstaben, die in der Beschriftung unterstrichen dargestellt werden sollen, ein `&` vorausgestellt werden muss.

Bild 20.38 Zellenkontextmenü auf zwei Befehle reduzieren



In der folgenden Prozedur werden in einer For-Schleife alle Zellen-Kontextmenübefehle ausgeblendet, bis auf zwei. Welche das sind, wird in der If-Entscheidung festgelegt.

Listing 20.69 Alle Zellen-Kontextmenübefehle bis auf zwei ausblenden

```
Sub FadeOut()
    Dim cmb As CommandBar
    Dim i As Integer

    Set cmb = Application.CommandBars("Cell")

    With cmb
        .Reset
        For i = 1 To .Controls.Count
            If i <> 8 And .Controls(i).Caption <>
                "Zellen &formatieren..." Then
                .Controls(i).Visible = False
            End If
        Next i
    End With

    Set cmb = Nothing
End Sub
```



Das obige Beispiel befindet sich auf der CD-ROM zum Buch im Ordner `\Buch\Kap20`. Die Mappe nennt sich `Bsp20_30.xls`. Die Prozedur ist im Modul `mdl_02_FadeOut` untergebracht.

Ein Kontextmenü ersetzen

Wenn Sie ein Kontextmenü komplett ersetzen möchten, können Sie einfach die beiden zuvor behandelten Beispiele anpassen und kombinieren.

Listing 20.70 Das Zellen-Kontextmenü komplett ersetzen

```
Sub ReplaceCommandBar ()
    Dim cmb As CommandBar
    Dim cmbb As CommandBarButton
    Dim i As Integer

    Set cmb = Application.CommandBars("Cell")

    With cmb
        .Reset

        ' Bestehende Befehle ausblenden
        For i = 1 To .Controls.Count
            .Controls(i).Visible = False
        Next i

        ' Neuen Befehl einfügen
        Set cmbb = .Controls.Add(Type:=msoControlButton, _
            Temporary:=True, Before:=5)

        With cmbb
            .BeginGroup = True           ' Trennlinie einfügen
            .Caption = "Benutzername"    ' Beschriftung
            .FaceId = 682                ' Symbol
            .OnAction = "UserName"       ' Prozeduraufruf
        End With
    End With

    Set cmb = Nothing
    Set cmbb = Nothing
End Sub
```



Das obige Beispiel befindet sich auf der CD-ROM zum Buch im Ordner `\Buch\Kap20`. Die Mappe nennt sich `Bsp20_30.xls`. Die Prozedur ist im Modul `mdl_03_Replace` untergebracht.

Ein verändertes Kontextmenü für einen bestimmten Bereich

Ein benutzerdefiniertes Kontextmenü kann auch nur auf einen bestimmten Zellenbereich angewendet werden. Um dies zu realisieren, muss das Ereignis `Worksheet_BeforeRightClick` verwendet werden. Die Ereignis-Prozedur wird dem gewünschten Tabellenblatt hinterlegt.

In der Ereignis-Prozedur wird in einer If-Entscheidung die Schnittmenge des Bereichs festgelegt, in dem das veränderte Kontextmenü erscheinen soll. Wenn eine Zelle außerhalb des Bereiches mit der rechten Maustaste angeklickt wird, wird das Kontextmenü in den Originalzustand zurückversetzt (Reset). Wenn der Rechtsklick im festgelegten Bereich ausgeführt wird, wird die Prozedur mit dem

benutzerdefinierten Kontextmenü aufgerufen. Es handelt sich hier um eines der vorangegangenen Beispiele (FadeOut aus Listing 20.69).

HINWEIS Sie können das Aufbereiten des benutzerdefinierten Kontextmenüs auch direkt in die Ereignisprozedur integrieren.

Listing 20.71 Benutzerdefiniertes Kontextmenü nur für festgelegten Bereich

```
Private Sub Worksheet_BeforeRightClick_  
    (ByVal Target As Range, Cancel As Boolean)  
  
    If Intersect(Target, Range("B3:D10")) Is Nothing Then  
        Application.CommandBars("Cell").Reset  
        Exit Sub  
    Else  
        Call FadeOut  
    End If  
End Sub
```



Das obige Beispiel befindet sich auf der CD-ROM zum Buch im Ordner `\Buch\Kap20`. Die Mappe nennt sich `Bsp20_31.xls`. Die Ereignisprozedur ist im Modul `Tabelle1` untergebracht.

