

16 Menüs und Symbolleisten

CommandBars	2
Symbolschaltflächen	7
Symbolleisten erstellen	13
Zusammenfassung	16

In allen Microsoft Office-Versionen bis einschließlich der Version 2003 spielen Menüs und Symbolleisten eine wichtige Rolle. Nicht nur sind fast alle internen Anwendungsbefehle darüber zu erreichen. Auch der Entwickler stellt dadurch seine Werkzeuge zur Verfügung.

Hinweis Beginn

Obwohl es auf den ersten Blick vielleicht nicht offensichtlich ist, sind auch Menüs eigentlich Symbolleisten mit Symbolschaltflächen.

Hinweis Ende

Word unterstützt, wie in Kapitel 14 beschrieben, die Erstellung von Symbolleisten für den globalen Gebrauch, oder für Dokument spezifische Aufgaben. Solange der Code hinter den Symbolschaltflächen sich in der gleichen Datei mit der Symbolleiste befindet, werden Symbolleisten eher selten programmtechnisch; sondern über die Benutzerschnittstelle (Menübefehl *Extras/Anpassen*) bei der Vorbereitung der Vorlage oder Datei erstellt. Für die Fernsteuerung außerhalb von Word sieht die Lage anders aus. Ein COM-Add-In oder VSTO-Lösung beispielsweise wird die für die Aufgabe benötigten Symbolleisten meistens (beim Laden) dynamisch erstellen.

In diesem Kapitel werden wir zuerst die allgemein beim Programmablauf nützlichen Eigenschaften der Symbolleisten vorstellen. Danach werden die Symbolschaltflächen behandelt. Und abschließend zeigen wir ein Beispiel, wie eine Symbolleiste vom Grund auf programmtechnisch erstellt wird.



2007

Ab Office 2007 werden die Menüs und Symbolleisten durch die Multifunktionsleiste abgelöst. Wie diese bearbeitet abgelöst. Wie diese bearbeitet wird, ist in Kapitel 17 beschrieben. Alle Symbolleisten, die in Dokumenten oder Symbolleisten, die in Dokumenten oder Dokumentvorlagen hinterlegt sind, sowie die dynamisch erstellten, werden die dynamisch erstellten, werden in Word 2007 auf der Registerkarte Add-Ins angezeigt. Automatisierungscode, der angezeigt. Automatisierungscode, der nicht unterstützte Befehle und Funktionen aufruft (aufruft (

Listing 16.2 beispielsweise), läuft ergebnislos und ohne Fehlermeldungen ab. Das in diesem Kapitel beschriebene CommandBars-Objektmodell ist einzig für das Erstellen und Verwalten von Kontextmenüs in Word 2007 von Interesse (siehe den

Abschnitt »Symbolleisten erstellen« sowie das Listing 16.3 in diesem Kapitel).

Aus Zeitgründen wird es wohl nicht möglich sein, alle in früheren Versionen erstellten Werkzeuge sofort an die neue Benutzerschnittstelle anzupassen. Sie können Befehle aus der Registerkarte *Add-Ins* besser sichtbar machen, indem diese der *Symbolleiste für den Schnellzugriff* zugewiesen werden:

1. Öffnen Sie die bestehende Dokumentvorlage und speichern Sie diese im neuen Dateiformat ab (.dotm).
2. In den *Word-Optionen* aktivieren Sie die Kategorie *Anpassen*.
3. Wählen Sie in der Liste *Symbolleiste für den Schnellzugriff anpassen* den Speicherort aus.
4. Wählen Sie in der Liste *Befehle auswählen* die Registerkarte *Add-Ins* aus.
5. Wählen Sie den Eintrag *Benutzerdefinierte Symbolleisten* aus, klicken Sie auf *Hinzufügen* und bestätigen Sie das Dialogfeld *Word-Optionen* mit *OK*.

CommandBars

Im Objektmodell stellt das Objekt `CommandBar` eine Symbol- oder Menüleiste dar. Ein `CommandBar` ist eine Eigenschaft des `Application`-Objekts, stammt jedoch vom *Office*-Objektmodell, was bei der Deklaration einer Objektvariablen offensichtlich wird. Die Symbolleisten eines *Kontextes* werden in einer `CommandBars`-Auflistung geführt. Eine spezifische Symbolleiste kann durch einen numerischen Indexwert oder durch seine **englische** Beschriftung angesprochen werden:

```
Dim cb1 as Office.CommandBar
Dim cb2 as Office.CommandBar
Set cb1 = Application.CommandBars(1)
Set cb2 = Application.CommandBars("Menu Bar")
```

Bezeichnung

Das Objektmodell stellt für das `CommandBar`-Objekt drei Eigenschaften zur Verfügung, womit der Indexwert sowie die englische und deutsche Bezeichnung ermittelt werden können: `Index`, `Name` bzw. `NameLocal`. Zudem ist die Eigenschaft `BuiltIn` nützlich, um festzustellen, ob es sich um eine benutzerdefinierte oder Word-interne Symbolleiste handelt. Eine Tabelle der im gegenwärtigen Kontext zur Verfügung stehenden Symbolleisten ist damit schnell mit einer Prozedur wie die in Listing 16.1 erstellt.

Hinweis Beginn

Eine entsprechende Liste für die standardmäßige Dokumentvorlage *Normal.dot* finden Sie im Anhang C.

Hinweis Ende

Listing 16.1: Eine Tabelle der verfügbaren Symbolleisten generieren, die die englischen und deutschen Bezeichnungen gegenüber stellt

```
Sub SymbolleistenAuflisten()
    Dim cb As Office.CommandBar
    Dim s As String
    Dim sTrenn As String
```

```

Dim rng As Word.Range
Dim tbl As Word.Table

sTrenn = ","
'Tabellenüberschriften
s = "Deutsche Bezeichnung" & sTrenn & "Englische Bezeichnung" & _
sTrenn & "Index" & sTrenn & "Benutzerdefiniert"
For Each cb In Application.CommandBars
    s = s & vbCr & cb.NameLocal & sTrenn & cb.Name & sTrenn & _
        cb.Index & sTrenn & (Not cb.BuiltIn)
Next
'Eine Tabelle wird anstelle der gegenwärtigen Markierung erstellt.
Set rng = Selection.Range
rng.Text = s
Set tbl = rng.ConvertToTable(sTrenn)
tbl.Rows(1).Range.Bold = True
End Sub

```

CD-ROM Beginn

Die Beispieldatei *Bsp16_01.doc* finden Sie auf der CD-ROM zum Buch im Ordner *\Beilagen\Kap16_Menüs_und_Symbolleisten\Bsp.*

CD-ROM Ende

Typ

Das Objektmodell führt auch eine Type-Eigenschaft für das CommandBar-Objekt auf. Dafür gibt es drei Konstanten: `msoBarTypeMenuBar`, `msoBarTypeNormal` sowie `msoBarTypePopup`. Lediglich eine Symbolleiste pro Kontext darf als Menüleiste bezeichnet werden. Die Menüs, die beim Anklicken aufklappen, sind Symbolleisten des Typs `msoBarTypePopup`. Die restlichen sind normale Symbolleisten. Type ist nur lesbar; der Typ einer Symbolleiste wird bei ihrer Erstellung festgelegt.

Positionierung

Symbolleisten können entweder an allen vier Seiten des Dokumentfensters verankert oder frei auf dem Bildschirm oder frei auf dem Bildschirm positioniert werden. Im Objektmodell wird dies über die Eigenschaft `Position` die Eigenschaft `Position` geregelt, in Kombination mit Eigenschaften wie `Left` (links), `Top` (oben), `Width` (links), `Top` (oben), `Width` (Breite) und `Height` (Höhe). Zusätzlich wird für Symbolleisten, die angedockt Symbolleisten, die angedockt werden, mit `RowIndex` die jeweilige Zeile festgelegt, wie die Abbildung 16.1, die wie die Abbildung 16.1, die Abbildung 16.2 sowie das

Listing 16.2 verdeutlichen.

Das Code-Beispiel veranschaulicht die Verwendung dieser Eigenschaften. Die Symbolleiste »Benutzerdefiniert 1« wird am oberen Rand (`msoBarTop`), ganz links (`Left = 0`), in der Zeile unter allen anderen oben angedockten Symbolleisten (`RowIndex`) angedockt. Die zweite Symbolleiste bleibt freistehend (`msoBarFloating`), über der linken oberen Ecke des Dokuments. Ihre Symbolschaltflächen werden, durch Festlegung der Breite, untereinander aufgereiht.

Abbildung 16.1: Beliebige Ausgangslage: die Symbolleisten befinden sich »irgendwo«

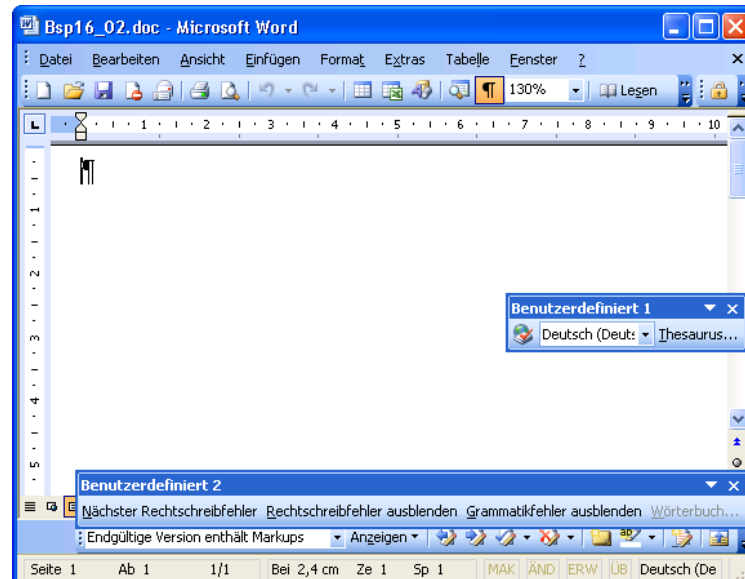
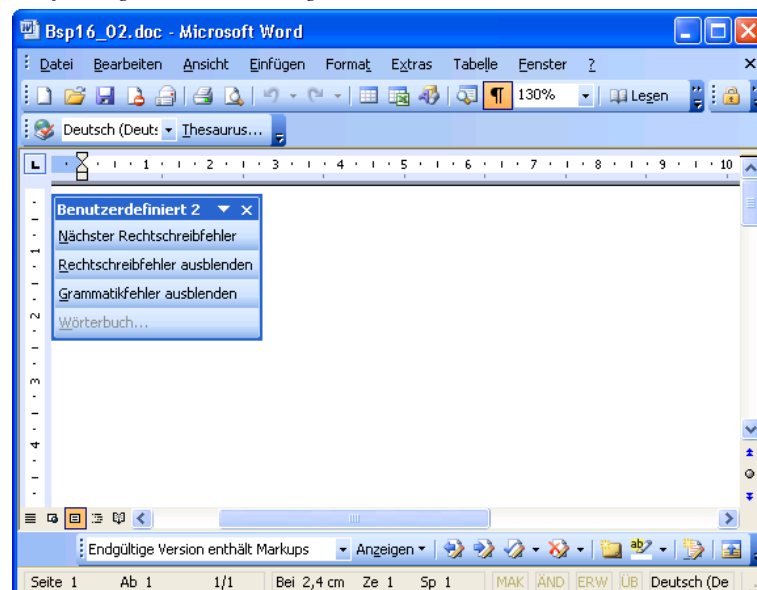


Abbildung 16.2: Nach Ausführung des Positionierungscodes



Hinweis Beginn

Bitte bedenken Sie, dass eine genaue Positionierung von Symbolleisten ein recht komplexes Unterfangen ist. Die Ausgangslage – wie viele und welche Symbolleisten wo platziert sind – wird von Benutzer zu Benutzer variieren. Zudem beeinflusst die Bildschirmauflösung die Wirkung der Maßangaben.

Hinweis Ende

Listing 16.2: Symbolleisten positionieren

```
Sub SymbolleistePositionieren()
    Dim cb1 As Office.CommandBar
    Dim cb2 As Office.CommandBar
    Dim doc As Word.Document
    Dim win As Word.Window
    Dim rng As Word.Range
    Dim lLinks As Long, lTop As Long, lWidth As Long, lHeight As Long
    Dim sngKorrektur As Single

    Set cb1 = Application.CommandBars("Benutzerdefiniert 1")
    Set cb2 = Application.CommandBars("Benutzerdefiniert 2")
    Set doc = ActiveDocument
    Set win = ActiveWindow

    'Die erste Symbolleiste ganz links andocken
    'in der nächsten, leeren Zeile, im oberen Teil
    'ganz links andocken
    cb1.Position = msoBarTop
    cb1.RowIndex = IndexErmitteln + 1
    'Dafür sorgen, dass sie die erste der Zeile ist
    cb1.Left = 0

    'Die zweite Symbolleiste nahe des oberen linken Fensterrahmens positionieren
    cb2.Position = msoBarFloating
    cb2.Width = 100
    cb2.Left = Application.Left + 30
    Set rng = doc.Range(0, 0)
    win.GetPoint lLinks, lTop, lWidth, lHeight, rng
    If win.View.DisplayPageBoundaries Then
        With doc.Sections(1).PageSetup
            sngKorrektur = .TopMargin + .HeaderDistance
        End With
    End If
    cb2.Top = CSng(lTop) - sngKorrektur
End Sub

Function IndexErmitteln() As Long
    Dim cb As Office.CommandBar
    Dim index As Long

    For Each cb In Application.CommandBars
        If cb.Position = msoBarTop Then
            If cb.RowIndex > index Then
                index = cb.RowIndex
            End If
        End If
    Next
    IndexErmitteln = index
End Function
```

Schutz

Wird Code benutzt, um Symbolleisten zu positionieren, ist es möglich, dass sie vom Anwender nicht verschoben oder angepasst werden dürfen. Wenn der Anwender gar nichts mit einer Symbolleiste machen soll, muss ihre `Enabled`-Eigenschaft auf `False` festgelegt werden. Somit wird sie auch unsichtbar und im Untermenübefehl *Ansicht/Symbolleisten* nicht aufgelistet.

Im Gegensatz dazu blendet die Eigenschaft `Visible` (sichtbar) eine Symbolleiste ein oder aus, bleibt aber im Untermenü zum Befehl *Ansicht/Symbolleisten* verfügbar.

Eine etwas verfeinerte Kontrolle bietet die Eigenschaft `Protection` an. Die in Tabelle 16.1 aufgelisteten `MsoBarProtection`-Konstanten dürfen nicht nur einzeln, sondern miteinander addiert zugewiesen werden. Soll beispielsweise eine Symbolleiste nicht ausgeblendet und auch nicht am rechten oder linken Rand andockt werden dürfen, verwenden Sie die folgende Anweisung:

```
cb.Protection = msoBarNoVerticalDock + msoBarNoChangeVisible
```

Tabelle 16.1: Mögliche Schutzeinstellungen für Symbolleisten

MsoBarProtection-Enum	Wert	Beschreibung
<code>msoBarNoChangeDock</code>	16	Die Symbolleiste kann nicht anderswo andockt werden.
<code>msoBarNoChangeVisible</code>	8	Die Symbolleiste kann nicht ausgeblendet werden, wenn sie sichtbar ist, bzw. nicht eingeblendet werden, wenn sie nicht sichtbar ist.
<code>msoBarNoCustomize</code>	1	Die Symbolleiste kann weder über den Word-Befehl <i>Extras/Anpassen</i> noch mit VBA geändert werden.
<code>msoBarNoHorizontalDock</code>	64	Die Symbolleiste darf weder oben noch unten andockt werden.
<code>msoBarNoMove</code>	4	Die Symbolleiste kann überhaupt nicht verschoben werden.
<code>msoBarNoProtection</code>	0	Es sind keine Schutzmaßnahmen aktiviert.
<code>msoBarNoResize</code>	2	Die Größe der Symbolleiste darf nicht geändert werden.
<code>msoBarNoVerticalDock</code>	32	Die Symbolleiste darf weder links noch rechts andockt werden.

DisableCustomize

Falls es dem Anwender untersagt werden soll, irgendwelche Anpassungen vorzunehmen, kann die `DisableCustomize`-Eigenschaft der `CommandBars`-Auflistung auf `True` festgelegt werden. Damit wird der Menübefehl *Extras/Anpassen* abgeblendet dargestellt und ist somit nicht mehr wählbar. Bitte beachten Sie, dass diese Eigenschaft kontextbezogen ist. Bestimmen Sie daher sorgfältig die `CustomizationContext`-Eigenschaft, bevor `DisableCustomize` festgelegt wird:

```
Application.CustomizationContext = ActiveDocument
Application.CommandBars.DisableCustomize = True
```

CD-ROM Beginn

Die Beispieldatei *Bsp16_02.doc* finden Sie auf der CD-ROM zum Buch im Ordner `\Beilagen\Kap16_Menüs_und_Symbolleisten\Bsp.`

CD-ROM Ende

*AdaptiveMenus**AdaptiveMenu*

Die Eigenschaft `AdaptiveMenus` gehört der Auflistung `CommandBars` des `Application`-Objekts und entspricht der Option *Menüs immer vollständig anzeigen* unter *Extras/Anpassen/Optionen*. Ist die Einstellung `True`, werden nur die zuletzt oder am häufigsten benutzten Befehle in den Symbolleisten angezeigt.

Die Eigenschaft `AdaptiveMenu` bezieht sich auf eine einzelne Symbolleiste (`CommandBar`) und legt dieses Verhalten dafür fest. Die Eigenschaft kommt jedoch nur zur Geltung, wenn `AdaptiveMenus` für die ganze Umgebung `True` ist.

Priority

Damit verwandt ist die Symbolschaltflächen- (Control) Eigenschaft `Priority`. Diese legt fest, welche Symbolschaltflächen in jedem Fall sichtbar sein sollen, egal wie oft sie vom Anwender ausgeführt werden. Eine interessante Nebenwirkung ist, dass damit *alle* Symbolschaltflächen einer Symbolleiste zwingend eingeblendet werden können, auch wenn dadurch eine angedockte Symbolleiste über mehrere Zeilen umbrochen wird.

DisableAskAQuestionDropdown

Auch interessant ist die Eigenschaft `DisableAskAQuestionDropdown` der `CommandBars`-Auflistung. Wird ihr der Wert `True` zugewiesen, verschwindet das Eingabefeld *Frage hier eingeben* aus der Menüleiste. Im Gegensatz zu `DisableCustomize` bezieht sich diese Eigenschaft immer auf die gesamte Word-Umgebung.

Symbolschaltflächen

Ohne Symbolschaltflächen ist eine Symbolleiste bedeutungslos. Die meisten Schaltflächen, die Sie in der Benutzerschnittstelle sehen sind des Typs `msoControlButton` (entspricht dem `CommandBarButton`-Objekt im Word-Objektmodell).

Wer im Objektkatalog des VB-Editors die Liste der `MsoControlType` betrachtet und auf hochfliegende Gedanken kommt, was alles damit angestellt werden könnte, wird schnell wieder auf den Boden der Realität geholt. Von den 27 aufgelisteten Konstanten stehen davon lediglich vier dem VBA-Entwickler für die Definition eigener Symbolleisten zur Verfügung: `msoControlButton`, `msoControlComboBox`, `msoControlEdit` sowie `msoControlPopup`. (Die restlichen sind nur lesbar, als Rückgabewert bei Prüfung des Typs.)

Während die internen Symbolleisten von Word über die englische Bezeichnung angesprochen werden müssen, ist der beschreibende Indexwert einer Symbolschaltflächen lokalisiert; die Beschriftung (`Caption`-Eigenschaft) ist gleichzeitig ein Indexwert.

ID

Dieser Umstand würde, was die Word-eigenen Symbolschaltflächen betrifft, eine Internationalisierung fast zu einem Ding der Unmöglichkeit machen, gäbe es nicht die `ID`-Eigenschaft des `Control`-Objekts. Haben Sie den `ID`-Wert einer Word-

internen Symbolschaltfläche ermittelt, kann diese, egal in welcher Sprachumgebung, durch den Einsatz der Methode `FindControl` eindeutig angesprochen werden.

Ein Beispiel hilft, dies zu veranschaulichen. In der `Dialogs`-Auflistung befindet sich keine Konstante für das Dialogfeld zum Menübefehl *Datei/Eigenschaften*. Folglich ist es scheinbar nicht möglich, dieses programmtechnisch einzublenden. Es gibt jedoch einen Weg: über die Methode `Execute` einer Symbolschaltfläche kann deren Befehl direkt ausgeführt werden.

Um den ID-Wert einer Symbolschaltfläche zu ermitteln:

```
Dim IID as Long
IID = Application.CommandBars("Menu Bar").Controls( _
    "Datei").CommandBar.Controls("Eigenschaften").Id 'Ergebnis: 750
```

Profitipp Beginn

Bitte beachten Sie, wie untere Menüebenen angesprochen werden. Ein `Control`-Objekt des Typs `msoControlPopup` besitzt eine `CommandBar`-Eigenschaft (auch wenn sie nicht in der `IntelliSense`-Liste erscheint). Diese ihrerseits hat ebenfalls eine `Control`-Eigenschaft. Ist dieses Steuerelement ebenfalls vom Typ `msoControlPopup`, geht's weiter im gleichen Stil, bis eine Symbolschaltfläche einer anderen Art vorliegt, die ausführbar ist.

Profitipp Ende

FindControl
Execute

Mit `FindControl` wird die Symbolschaltfläche über den ID-Wert angesprochen, und mit `Execute` wird sie ausgeführt, genau so, als ob der Anwender darauf geklickt hätte. In unserem Beispiel wird das Dialogfeld, egal in welcher Sprachumgebung, eingeblendet:

```
Application.CommandBars.FindControl(ID:=750).Execute
```

Enabled, Visible

Während eines Programmablaufs sind sonst hauptsächlich die Eigenschaften `Visible` (sichtbar) und `Enabled` (zugänglich) interessant. Im Gegensatz zu den gleichnamigen Eigenschaften des `CommandBar`-Objekts bleibt eine Symbolschaltfläche sichtbar, wenn `Enabled` auf `False` gesetzt ist. Allerdings wird sie abgeblendet dargestellt und ist somit nicht mehr wählbar.

Eine unsichtbare Schaltfläche verschwindet aus der Symbolleiste, kann jedoch weiterhin mit `Execute` ausgeführt werden.

Symbolschaltflächen erstellen

In der Benutzerschnittstelle werden Symbolschaltflächen über den Menübefehl Extras/Anpassen auf der Extras/Anpassen auf der Registerkarte Befehle definiert. Der Word-Befehl oder das Makro wird aus der Liste Makro wird aus der Liste Befehle (rechts in

Abbildung 16.3) in die Symbolleiste oder Menü gezogen, wo die Symbolschaltfläche stehen soll. Über die Schaltfläche *Auswahl ändern* wird das Aussehen angepasst: die Beschriftung (`Caption`) wird festgelegt und bestimmt, ob

und welches Symbol angezeigt werden soll (Style).

Abbildung 16.3: Symbolleisten in der Benutzeroberfläche definieren



Hinweis Beginn

Obwohl in der Word-Umgebung vier Steuerelementtypen definiert werden können, lassen sich über die Benutzerschnittstelle nur zwei erstellen: eine Symbolschaltfläche zum Anklicken (`msoControlButton`) und ein Untermenü (`msoControlPopup`). Um ein Untermenü zu erstellen, wird im Dialogfeld *Anpassen* auf der Registerkarte *Befehle* aus der Liste *Kategorien* der Eintrag *Neues Menü* gewählt. Dann wird der Befehl *Neues Menü* in eine Symbolleiste, Menüleiste oder Popup-Menü gezogen.

Hinweis Ende

Über die Automatisierungsschnittstelle stehen mehr Gestaltungsmöglichkeiten zur Verfügung. Quick-Info (`ToolTipText`), Tastaturzuweisung (`ShortcutText`), Sichtbarkeit (`Visible`) sowie Zugang (`Enabled`) können einer Schaltfläche zusätzlich zugewiesen werden.

Add-Methode

Eine neue Symbolschaltfläche wird mit der Add-Methode der Controls-Auflistung hinzugefügt, die folgende Syntax aufweist:

```
Add(Type, Id, Parameter, Before, Temporary)
```

Das `Type`-Argument ist das einzig erforderliche; es erwartet irgendeinen der vier `MsoControlType`-Werte. `ID` wird nur gebraucht, wenn die Symbolschaltfläche

einen Word-internen Befehl ausführen soll (wie der ID-Wert ermittelt wird, wurde weiter oben beschrieben). Mit `Parameter` kann eine unsichtbare Zeichenkette mitgespeichert werden. Um die Symbolschaltfläche an einer bestimmten Stelle einzufügen, wird dem Argument `Before` eine Ganzzahl zugewiesen, die die Ordinalzahl in der Symbolleiste angibt. `Temporary` hat keine Bedeutung in der Word-Umgebung; eine Schaltfläche wird nie automatisch beim Schließen des Kontextes entfernt.

Hinweis Beginn

Wenn Sie außerhalb von Word Symbolschaltflächen, die in mehreren Dokumenten benutzt werden, erstellen und über ein Ereignis mit Code verbinden, müssen Sie der `Tag`-Eigenschaft jeder Symbolschaltfläche den gleichen eindeutigen Wert zuweisen. Sonst werden sie nur im ersten Dokument lauffähig sein. Mehr Informationen finden Sie im Artikel »Hooking a COM Add-in Up to a Command Bar Control« unter der URL [http://msdn2.microsoft.com/en-us/library/Aa165301\(office.10\).aspx](http://msdn2.microsoft.com/en-us/library/Aa165301(office.10).aspx).

Hinweis Ende

Word-interner Befehl

Um den Word-Formatierungsbefehl »Kursiv« einer eigenen Symbolleiste an erster Stelle zuzuweisen, können Sie die folgenden Anweisungen verwenden:

```
Application.CustomizationContext = ActiveDocument
CommandBars("Meine Befehle").Controls.Add(msoControlButton, 114, 1)
```

OnAction

Das Listing 16.3 veranschaulicht, wie man dem allgemeinen Kontextmenü `Text` eine Schaltfläche zuweist, die mit einem Makro verbunden ist. Wird eine Symbolschaltfläche nicht mit einem Word-internen Befehl verbunden, muss der `OnAction`-Eigenschaft der Name einer Prozedur zugewiesen werden, wenn die Schaltfläche irgendeine Handlung ausführen soll.

Listing 16.3: Eine Schaltfläche im Kontextmenü erstellen, die mit einem Makro verbunden ist

```
Sub SymbolschaltflächeDefinieren1()
    Dim ctl As Office.CommandBarButton

    Application.CustomizationContext = ActiveDocument
    Set ctl = CommandBars("Text").Controls.Add(msoControlButton)
    ctl.Caption = "Dokumentschutz aktivieren"
    ctl.OnAction = "DokumentschutzAktivieren"
    ctl.DescriptionText = _
        "Aktiviert 'Änderungen verfolgen' und schützt das Dokument für Revisionen"
    ctl.Style = msoButtonCaption
    ctl.TooltipText = "Dokument für Revisionen schützen"
End Sub

Sub DokumentSchutzAktivieren()
    Dim doc As Word.Document

    Set doc = ActiveDocument
    doc.TrackRevisions = True
    If doc.ProtectionType = wdNoProtection Then
        doc.Protect Type:=wdAllowOnlyRevisions
    End If
End Sub
```

Hinweis Beginn

Es können nur `Public Sub`-Prozeduren, die keine Argumente verwenden, einer Symbolschaltfläche zugewiesen werden. Falls eine Schaltfläche einen Wert dieser Prozedur liefern soll, kann er in der `Parameter`-Eigenschaft gespeichert werden.

Hinweis Ende

ActionControl

Wollen Sie feststellen, welche Symbolschaltfläche betätigt wurde, können Sie die `ActionControl`-Eigenschaft der `CommandBars`-Auflistung benutzen. In Listing 16.4 werden mehrere Symbolschaltflächen einer Symbolleiste hinzugefügt. Als Parameter wird ein Farbname festgelegt. Der `OnAction`-Eigenschaft aller Objekte wird das gleiche Makro `MarkierungHervorheben` zugewiesen. Klickt der Anwender auf eine dieser vier Symbolschaltflächen (in Abbildung 16.4 abgebildet), wird diese durch `ActionControl` ermittelt. Je nach Parameter (dem Farbennamen), wird der Wert für die Hervorhebungsfarbe festgelegt.

Listing 16.4: Mehrere Schaltflächen erstellen, die mit dem gleichen Makro verbunden sind. Die Parameter-Eigenschaft entscheidet, welche Handlung auszuführen ist.

```
Sub SymbolschaltflächeDefinieren2()
    Dim cb As Office.CommandBar

    Application.CustomizationContext = ActiveDocument
    Set cb = CommandBars("Meine Befehle")
    HervorhebungsSchaltflächen cb, "Gelb"
    HervorhebungsSchaltflächen cb, "Grün"
    HervorhebungsSchaltflächen cb, "Rosa"
    HervorhebungsSchaltflächen cb, "Keine"
End Sub

Sub HervorhebungsSchaltflächen(ByVal cb As Office.CommandBar, ByRef sFarbe As String)
    Dim ctl As Office.CommandBarButton

    Set ctl = cb.Controls.Add(Type:=msoControlButton, Parameter:=sFarbe)
    ctl.Caption = sFarbe & " hervorheben"
    ctl.OnAction = "MarkierungHervorheben"
    ctl.DescriptionText = "Hervorhebung der gegenwärtigen Markierung " & sFarbe
    ctl.Style = msoButtonCaption
    ctl.ToolTipText = "Markierung " & sFarbe & " hervorheben"
    ctl.BeginGroup = True
    Set ctl = Nothing
End Sub

Sub MarkierungHervorheben()
    Dim ctl As Office.CommandBarButton
    Dim lFarbenWert As Long

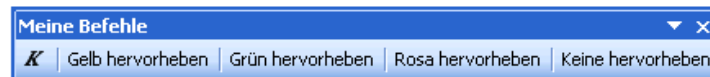
    Set ctl = Application.CommandBars.ActionControl
    Select Case ctl.Parameter
        Case "Gelb"
            lFarbenWert = wdYellow
        Case "Grün"
            lFarbenWert = wdBrightGreen
```

```

Case "Rosa"
    lFarbenWert = wdPink
Case "Keine"
    lFarbenWert = wdWhite
Case Else
    lFarbenWert = wdRosa
End Select
Selection.Range.HighlightColorIndex = lFarbenWert
End Sub

```

Abbildung 16.4: Schaltflächen, die über die Automatisierungsschnittstelle erstellt wurden



Wenn Sie im Text rechts anklicken, können Sie das Dokument für Revisionen schützen.

Die Schaltflächen in der Symbolleiste helfen beim Formatieren.

CD-ROM Beginn

Die Beispieldatei *Bsp16_03.doc* finden Sie auf der CD-ROM zum Buch im Ordner *\Beilagen\Kap16_Menüs_und_Symbolleisten\Bsp.*

CD-ROM Ende

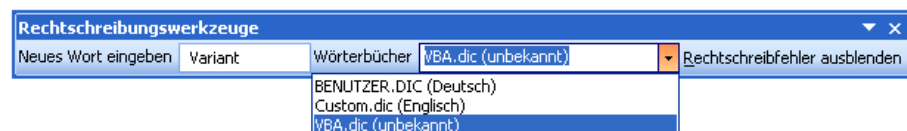
Bearbeitungsfelder und Dropdownlisten

Obwohl der Word-Umgebung vier Befehlsleisten-Steuerelementtypen zur Verfügung stehen, können über die Benutzerschnittstelle nur zwei davon definiert werden: Schaltflächen (`msoControlButton`) sowie Popup-Menüs (`msoControlPopup`). Um Bearbeitungsfelder oder Dropdownlisten einer Symbolleiste hinzuzufügen, braucht es ein Makro.

Diese zwei Control-Typen sind in Abbildung 16.5 ersichtlich. Deren `Style`-Eigenschaft wurde jeweils auf `msoComboLabel` festgelegt (ein Bearbeitungsfeld ist eine Dropdownliste ohne die Liste). Das `OnAction`-Makro eines Bearbeitungsfelds wird durch Drücken der (Eingabe)-Taste ausgeführt; das einer Dropdownliste durch die Auswahl eines Eintrags.

In diesem Beispiel wird das vom Anwender in das Bearbeitungsfeld eingegebene Wort dem aktuellen Benutzerwörterbuch hinzugefügt. Das aktuelle Wörterbuch kann aus der Liste der geladenen Wörterbücher in der Dropdownliste festgelegt werden (das aktive Wörterbuch erscheint im Textfeld der Dropdownliste).

Abbildung 16.5: Bearbeitungsfelder und Dropdownlisten, wie hier abgebildet, können nur programmtechnisch erstellt werden



Der Code, um diese beiden Symbolschaltflächen zu erstellen, befindet sich in Listing 16.5. Elemente werden einer Dropdownliste durch die `AddItem`-Methode

hinzugefügt.

Listing 16.5: Edit- und ComboBox-Schaltflächen einer Symbolleiste hinzufügen

```
Sub EditSchaltflächeErstellen()  
    Dim ctl As Office.CommandBarControl  
  
    Set ctl = Application.CommandBars(" _  
        RechtschreibungsWerkzeuge").Controls.Add(msoControlEdit)  
    ctl.Style = msoComboLabel  
    ctl.Caption = "Neues Wort eingeben"  
    'Wird beim Drücken der Eingabe-Taste ausgelöst  
    ctl.OnAction = "WortHinzufügen"  
End Sub  
  
Sub WörterbuchListeErstellen()  
    Dim cbo As Office.CommandBarComboBox  
    Dim dic As Word.Dictionary  
    Dim lZaehler As Long  
  
    Application.CustomizationContext = ActiveDocument  
    Set cbo = Application.CommandBars(" _  
        RechtschreibungsWerkzeuge").Controls.Add(Type:=msoControlComboBox)  
    cbo.Style = msoComboLabel  
    cbo.Caption = "Wörterbücher"  
    cbo.Width = 250  
    cbo.OnAction = "AktivesWörterbuchAendern"  
    lZaehler = 1  
    For Each dic In Application.CustomDictionaries  
        cbo.AddItem dic.Name & Sprache(dic.LanguageID)  
        If Application.CustomDictionaries.ActiveCustomDictionary.Name _  
            = dic.Name Then  
            cbo.ListIndex = lZaehler  
        End If  
        lZaehler = lZaehler + 1  
    Next  
End Sub
```

CD-ROM Beginn

Die Beispieldatei *Bsp16_04.doc* mit der gesamten Lösung finden Sie auf der CD-ROM zum Buch im Ordner *\Beilagen\Kap16_Menüs_und_Symbolleisten\Bsp*.

CD-ROM Ende

Symbolleisten erstellen

Neue Symbolleisten werden mit der Add-Methode der CommandBars-Auflistung erstellt:

```
Add(Name, Position, MenuBar, Temporary)
```

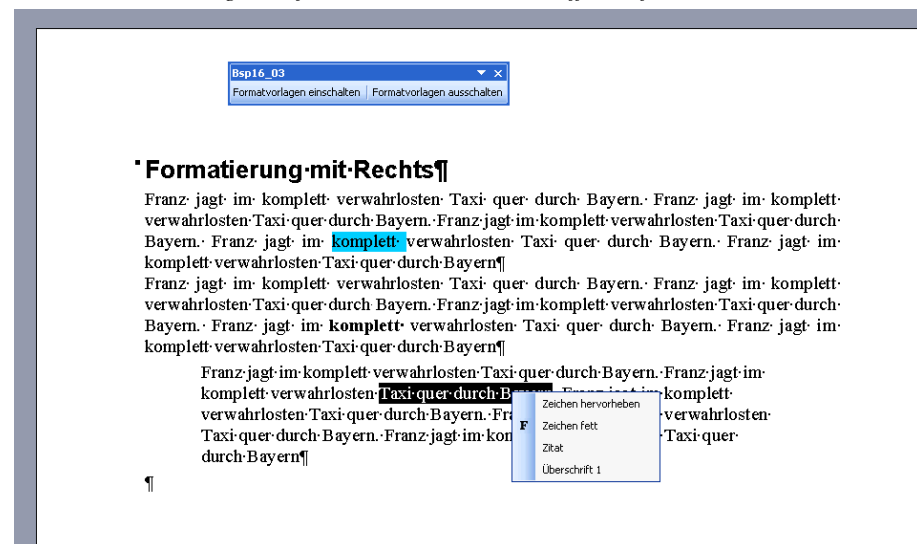
Alle Argumente sind optional. Die Werte für Position wurden bereits im Abschnitt »CommandBars« am Anfang dieses Kapitels vorgestellt. Soll die neue Symbolleiste die Menüleiste des gegenwärtigen Kontextes ersetzen, muss MenuBar

auf True festgelegt werden. Für Temporary bedeutet True, dass die Symbolleiste beim Schließen des Kontextes verworfen wird.

Interessant als Beispiel, das das Erstellen einer Symbolleiste samt Steuerelemente veranschaulicht, ist das Erstellen eines Kontextmenüs, das eines der Word-eigenen ersetzt. Die Abbildung 16.6 zeigt dazu ein einfaches Beispiel. Um Text in einem Dokument schnell zu formatieren, werden die benötigten Formatvorlagen in einem Kontextmenü aufgelistet. Dadurch muss die Maus nicht bis zu einer Symbolleiste, die entweder weit entfernt liegt oder deren Sicht darauf verdeckt ist, bewegt werden, sondern kann unmittelbar neben der Markierung den gewünschten Befehl auswählen.

Mit den Symbolschaltflächen in der »normalen« Symbolleiste (oben in Abbildung 16.6) wird die Aktivierung dieses Menü ein- und ausgeschaltet (das Word-eigene Kontextmenü wird wieder aktiv).

Abbildung 16.6: Ein Dokument mit einem eigens dafür erstellten Kontextmenü effizient formatieren



Das Kontextmenü wurde mit der Prozedur *KontextMenuErstellen* in Listing 16.6 erstellt. Bitte beachten Sie dabei den Control-Typ *msoControlPopup*. Stellvertretend für die Erstellung der einzelnen Schaltflächen und ihre *OnAction*-Makros, die alle nach dem gleichen Muster ablaufen, befinden sich *MenuEintrag1Erstellen* bzw. *MenuEintrag1* im aufgeführten Code.

Listing 16.6: Diese Prozedur einmal im erwünschten Speicherort durchführen, um das Kontextmenü zu erstellen

```
Sub KontextMenuErstellen()
    Dim cb As Office.CommandBar

    Application.CustomizationContext = ActiveDocument
    Set cb = Application.CommandBars.Add(Name:="Kontext", Position:=msoBarPopup)
    MenuEintrag1Erstellen cb
    MenuEintrag2Erstellen cb
    MenuEintrag3Erstellen cb
    MenuEintrag4Erstellen cb
End Sub
```

```

Sub MenuEintrag1Erstellen(cb As Office.CommandBar)
    Dim ctl As Office.CommandBarButton

    Set ctl = cb.Controls.Add(Office.msoControlButton)
    ctl.Caption = "Zeichen hervorheben"
    ctl.Enabled = True
    ctl.OnAction = "MenuEintrag1"
    ctl.Style = msoButtonCaption
    ctl.Visible = True
End Sub

Public Sub MenuEintrag1()
    Dim rng As Word.Range
    Set rng = Application.Selection.Range
    rng.Style = "Zeichen hervorheben"
    rng.Shading.ForegroundPatternColor = wdColorSkyBlue
End Sub

```

Ein Popup-Menü kann nur programmtechnisch eingeblendet werden; die Eigenschaft `Visible` verursacht einen Laufzeitfehler. Dieses Beispiel bedient sich des Ereignisses `WindowBeforeRightClick`. Das Ereignis wird mit der Prozedur *EreignisseEinschalten* in Listing 16.7 aktiviert und mit *EreignisseAusschalten* wieder außer Kraft gesetzt. Die beiden sind den Schaltflächen der »normalen« Symbolleiste zugewiesen, so dass der Benutzer das Kontextmenü nach Bedarf benutzen kann.

Hinweis Beginn

Einzelheiten über den Einsatz von Ereignissen sind in Kapitel 8 beschrieben.

Hinweis Ende

Die Ereignis-Prozedur *app_WindowBeforeRightClick* ruft `ShowKontext` auf, die das Kontextmenü einblendet. Die standardmäßige Handlung des Rechtsklicks wird unterdrückt (`Cancel=True`).

Listing 16.7: Die Teile des Ereignisses, das das Rechtsklick-Verhalten steuert

```

'Inhalt des normalen Moduls
Public appEvents As New clsBsp16_03

Public Sub EreignisseEinschalten()
    Set appEvents.app = Word.Application
End Sub

Public Sub EreignisseAusschalten()
    Set appEvents.app = Nothing
End Sub

Public Sub ShowKontext(cb As Office.CommandBar)
    cb.ShowPopup
End Sub

'Inhalt des Klassenmoduls
Public WithEvents app As Word.Application
Private Sub app_WindowBeforeRightClick(ByVal sel As Selection, Cancel As Boolean)
    Dim cb As Office.CommandBar

```

```
app.CustomizationContext = ActiveDocument
Set cb = app.CommandBars("Kontext")
ShowKontext cb
Cancel = True
End Sub
```



2007

Bis einschließlich Word 2003 werden die Kontextmenüs über das Dialogfeld Anpassen (Anpassen (

Abbildung 16.3) verwaltet. In Word 2007 gibt es keine ähnliche Möglichkeit – Anpassungen erfolgen zwingend über das Objektmodell. Um ein bestehendes Kontextmenü mit Befehlen zu ergänzen, müssen Sie dessen Namen kennen. Die Beispieldatei *Bsp16_06.doc* enthält Code, der eine Liste vorhandener Kontextmenüs samt den Steuerelementen erstellt.

Hinweis Beginn

Müssen viele komplexe Symbolleisten erstellt und entfernt werden, ist es mühsam, alle Einzelheiten im Code aufzuschreiben und zu verwalten. Für solche Aufgaben lohnt es sich, die Einzelheiten des Aufbaus einer Symbolleiste mit allen Eigenschaften in einer externen Datei festzuhalten. Diese könnte beispielsweise eine Excel-Tabelle, eine Access-Datenbank oder eine XML-Datei sein. Der Code öffnet die Datei, arbeitet diese ab, und erstellt nach deren Angaben die Symbolleisten.

Hinweis Ende

CD-ROM Beginn

Die Beispieldateien *Bsp16_05.doc* und *Bsp16_06.doc* finden Sie auf der CD-ROM zum Buch im Ordner *\Beilagen\Kap16_Menüs_und_Symbolleisten\Bsp*.

CD-ROM Ende

Zusammenfassung

Dieses Kapitel beschäftigte sich mit der programmtechnischen Erstellung von Symbolleisten. Beschrieben wurde dabei,

- wie mit den Eigenschaften einer Symbolleiste umzugehen ist (Seite 2),
- wie sich verschiedene Arten von Bedienelementen für eine Symbolleiste erstellen und mit einem Befehl verbinden lassen (Seite 8),
- wie Symbolleisten erstellt werden (Seite 13).