HANSER

Leseprobe

Jens Hansen

KochbuchCATIA V5 automatisieren

Vom Powercopy bis zur C#-Programmierung

ISBN: 978-3-446-41621-5

Weitere Informationen oder Bestellungen unter

http://www.hanser.de/978-3-446-41621-5

sowie im Buchhandel.

3 Oft benutzt und nützlich

So, kommen wir zum Wesentlichen und endlich zum Grund, warum Sie dieses Buch gekauft haben – die „Kochrezepte“. Alle hier aufgeführten Rezepte entstanden durch Anregung und Anforderungen aus der Wirtschaft und dem täglichen Leben.

Ich habe im Vorfeld in den von mir moderierten CATIA-Foren auf CAD.DE eine Wunschliste angeregt, wo jeder sein „Wunsch-Makro“ oder seine Powercopy eintragen konnte. Des Weiteren habe ich bei meinen Kunden ebenfalls viele Anregungen und Wünsche aufnehmen können.

Bei all diesen Leuten möchte ich mich an dieser Stelle ganz herzlich bedanken.

Die Rezepte gliedern sich nach den Anwendungsbereichen:

- Dateisystem und Infrastruktur
- Part- und Assembly-Design
- Flächenkonstruktion
- Drafting

In diesen Bereichen werden die jeweiligen Makros, VBA-Makros und externen Applikationen erläutert. Powercopies gibt es nur im Part-Design und in der Flächenkonstruktion. Da es keinen Sinn macht, den gesamten Quellcode auf etlichen Seiten zu verteilen, wird nur der relevante Bereich jeweils erklärt.

Das entsprechende fertige Makro inkl. Quellcode finden Sie im Internet zum Herunterladen.

Dieses Symbol bedeutet, dass Sie mit dem gegebenen Suchbegriff in der V5-Automation.chm[1] zusätzliche Infos finden.

3.1 Grundrezepte

Es gibt viele Funktionen oder Objekte, die immer wieder benutzt werden. Die Syntax ist (fast) immer gleich oder meist sehr ähnlich. Es bietet sich daher an, einige „Grundrezepte“ direkt zur Hand zu haben, ohne lange überlegen zu müssen, wie denn noch mal die genaue Syntax für das Objekt oder die Funktion lautet.

1 V5Automation.chm ist die online-Hilfe für das Scripting und befindet sich im Verzeichnis „..\Dassault Systemes\B18\intel_a\code\bin\“

Syntax	Beschreibung
CATIA.DisplayFileAlerts = False	Unterdrücken von Dateisicherungsmeldungen
Set selection1 = CATIA.ActiveDocument.Selection	Deklaration einer Selektion

On Error Resume Next
Set activedoc = CATIA.ActiveDocument
If Err.Number <> 0 Then
MsgBox "Es ist kein Dokument geöffnet", 16
Exit Sub
End If
	Prüfen, ob ein Dokument geöffnet ist. Bei Fehler Meldung an den Anwender und Makro beenden

If (Right(activedoc.Name, 7) <> "CATPart") Then
MsgBox "Aktives Dokument ist kein Bauteil", 16
Exit Sub
End If
	CATPart anhand des Dokumentnamens identifizieren

If (Right(activedoc.Name, 10) <> "CATProduct") Then
MsgBox "Aktives Dokument ist keine Baugruppe", 16
Exit Sub
End If
	CATProduct anhand des Dokumentnamens identifizieren

If (Right(activedoc.Name, 10) <> "CATDrawing") Then
MsgBox "Aktives Dokument ist kein Bauteil", 16
Exit Sub
End If
	CATDrawing anhand des Dokumentnamens identifizieren

Dim wert1 as Integer wert1 = InStrRev(**String**, ",")	Erstes Vorkommen des Zeichens „," (Komma) im Text ermitteln
Dim InputObjectType() InputObjectType(0) = "Point"	Setzen des Auswahlfilters auf PUNKTE bei einer User-Selection
Status = selection1.SelectElement2(InputObjectType, "Wählen Sie das Objekt aus", False)	Interaktive Auswahl eines Objektes durch den User
Set visPropertySet1 = Selection1.VisProperties	Deklaration der Sichtbarkeits- und Farbsteuerung einer Auswahl

Syntax	Beschreibung
MsgBox "Makro ist beendet", 64, "Name des Makros"	Meldungsfenster
Input = InputBox("Geben Sie den neuen Namen ein", "Titel", „Standardtext in der Inputbox“)	Fenster mit Eingabefeld
CATIA.ActiveDocument.Product.PartNumber	Teilenummer eines Parts bzw. Produktes – *geht nicht bei Drawing!!*
Selection1.Item(1).Value	Objekt einer Auswahl inkl. Typendeklaration -> „Pad.1“ oder „Point.4“
CATIA.StartCommand "Exit workbench"	Direkten Befehl an V5 senden – hier Workbench verlassen. – *Befehl wird immer asynchron zum laufenden Makro ausgeführt!*
Exit Function, Exit Sub	Abbruch einer Funktion bzw. Routine
if Err.Number <> 0 then End if	Abfrage eines Fehlers anhand der Fehlernummer
Dim meineAuflistung(2) MeineAuflistung(0) = „Zeile1“ MeineAuflistung(1) = „Hier kommt der 2. Punkt“ MeineAuflistung(2) = „Hier kommt der 3 Punkt“	Erstellen einer Auflistung mit drei *(2+1)* Zeilen
Redim preserve MeineAuflistung(9)	Erweiterung der Auflistung auf zehn *(9+1)* Zeilen, wobei der bisherige Inhalt behalten wird. Der zusätzliche Inhalt muss noch definiert werden.
UBound(meineAuflistung)	Max. Anzahl meiner Auflistung ermitteln
Dim meineTabelle(1,1) meineTabelle (0,0) = „1. Zeile, 1. Spalte“ meineTabelle (0,1) = „1. Zeile, 2. Spalte“ meineTabelle (1,0) = „2. Zeile, 1. Spalte“ meineTabelle (1,1) = „2. Zeile, 2. Spalte“	zweidimensionales Array => vergleichbar mit einer normalen Tabelle mit Zeilen und Spalten. WICHTIG: Kann nicht Redim Preserve erweitert werden.

Syntax	Beschreibung
Set Zeilenzahl = UBound(meineTabelle,0)	Max. Anzahl der Zeilen ermitteln
Dim Shell As Object Set Shell = CreateObject("Shell.Application") Ordner = Shell.BrowseForFolder(0, "Wählen Sie bitte den Ordner mit den Dateien aus", 0).Self.Path	Deklaration des Ordner-auswahl-Dialoges
Dim index Do While index < 100 Index = index +1 Loop	Schleife mit Abbruch-bedingung
For i = 1 To CATIA.Documents.Count Next	Schleife mit fester Durch-laufzahl
On Error Resume Next	Fehler ignorieren und mit Makro weitermachen
On Error Goto 0	Ignorieren der Fehler ausschalten
& Chr(13)	Zeilenumbruch in einem Text (für Ausgabe)
Set Excel1 = GetObject(, "Excel.Application")	Zugriff au die LAUFENDE Excel-Instanz
Excel1.Workbooks.Open ("C:\Temp\meineexceldatei.xls")	Öffnen der Excel-Datei

3.2 Objekte für die Selektion

Bei einer Selektion durch den Anwender muss oft ein Filter gesetzt werden, um sicherzustellen, dass der Anwender nur bestimmte Objekte auswählen kann. Hier eine Auflistung der *(wichtigsten)* Objekttypen, die in dem Auswahlfilter der Selektion verwendet werden können. Hybrid Shapes Automation Objects.

Objekte für den Auswahlfilter

„Geometric Element Automation Objects"

Objekt	Beschreibung
AnyObject	Jedes beliebige Objekt
Drahtgitter-Elemente	
Point2D	Punkt in Skizze oder Drafting
Point	Punkte, die im WSD erzeugt wurden

Damit nun alles reibungslos abläuft, müssen nur noch die If-Anweisung für die Typabfrage und natürlich die For-Schleife geschlossen werden. Zum Schluss wird die Fehlernummer erneut auf 0 zurückgesetzt.

```
    End If
  Next
Err.Clear
```

Damit wäre das Makro so gut wie fertig. Für etwas mehr Komfort können wir bei aufgetretenen Fehlern eine konkretere Beschreibung ausgeben lassen. Das funktioniert mit ERR.DESCRIPTION.

```
iErr = Err.Number
  If (iErr <> 0) Then
    MsgBox (Err.Description)
    Exit Sub
  End If
```

Ein abschließender Hinweis für den Anwender, dass das Makro beendet ist, macht das Rezept nun endgültig komplett.

```
MsgBox "Makro ist beendet", 64, makroname + " " + version
End Sub
```

R4 Umbenennen aller Instanzen in einer Baugruppe

Wer als Automobilzulieferer tätig ist, kennt die Vorgaben, dass die Dateinamen, Teilenummern und Instanznamen den Kennzeichnungsschlüssel des Kunden enthalten müssen. Wer nicht an das PDM-System des Kunden angeschlossen ist, muss diesen Schlüssel entweder von Anfang an in die Teilenummer und den Dateinamen oder nachträglich in alle relevanten Dateien eintragen. Letzteres ist allerdings ziemlich arbeitsaufwendig und auch fehleranfällig – also für ein Makro prädestiniert.

Makro-Typ:	catvbs
V5-Level:	V5R16 und höher
Dialogsprache:	Deutsch
Voraussetzungen:	Es muss ein CATProduct geöffnet sein.

Strategie

1. Ausschalten der Fehlerroutine und der Dateisicherungsmeldung
2. Prüfen, ob ein Dokument geöffnet ist
3. Überprüfen des Dateityps

4. Deklaration des Trennzeichens zur Unterscheidung von Schlüssel und Instanznamen
5. Aufruf einer Inputbox zur Eingabe des Schlüssels
6. Erstellen eines neuen Unterordners mit Name des Schlüssels
7. Auflistung aller geöffneten Dokumente und Ansprache jedes einzelnen Dokumentes in einer Schleife
8. Durchsuchen nach etwaigen Instanzen in aktuell betrachtetem Dokument
9. Umbenennen der Instanz anhand des Schlüssels
10. Sichern des aktuell betrachteten Dokumentes unter neuen Namen im neuen Unterordner
11. Beenden des Makros durch Meldung an Anwender

Quellcode

```
Dim version, makroname

Sub CATMain()
version = "1.0"
makroname = "Instance_rename"
  On Error Resume Next
  CATIA.DisplayFileAlerts = False
```

Wir schon im ersten Rezept R1, beginnen wir mit der generellen Definition der Hauptroutine SUB CATMAIN(). Im Anschluss wird die Fehlerroutine mit ON ERROR RESUME NEXT deaktiviert. Die ist notwendig, um später eine Abfrage nach einem geöffneten Dokument zu machen. Danach werden die Dateisicherungsmeldungen mit CATIA.DISPLAYFILEALERTS=FALSE deaktiviert.

Als Nächstes wird getestet, ob ein Dokument geöffnet ist. Hier deklarieren wir das Objekt „activedoc“ als aktives Dokument.

```
Set activedoc = CATIA.ActiveDocument
```

Sollte es dabei zu einem Fehler kommen, bedeutet dies, dass kein Dokument geöffnet ist. Diese Situation fangen wir mit der Abfrage nach der Fehlernummer ERR.NUMBER ab. Ist die Fehlernummer ungleich 0, wird dem Anwender eine Meldung anhand einer MESSAGEBOX angezeigt und das Makro mit EXIT SUB beendet.

```
If Err.Number <> 0 Then
  MsgBox "Es ist kein Dokument geöffnet", 16, makroname + " " + version
  Exit Sub
End If
```

Im Anschluss muss geprüft werden, ob das aktive Dokument ein CATProduct ist. Dies lässt sich bewerkstelligen, indem das Ende des Dokumentnamens mit der Zei-

chenfolge „CATProduct“ übereinstimmt. Für diesen Vergleich benutzen wir wieder die VB-Funktion RIGHT(STRING,ANZAHL DER ZEICHEN).

```
Set activedoc = CATIA.ActiveDocument
If (Right(activedoc.Name, 10) <> ”CATProduct”) Then
   MsgBox ”Aktives Dokument ist keine Baugruppe”, 16, makroname + ” ” + version
   Exit Sub
End If
```

Bisher haben wir dasselbe gemacht wie bereits schon im ersten Rezept R1 – der Quellcode kann also bis auf die Änderung des Makronamens kopiert werden. Einzig die Abfrage, ob ein CATPart gerade geöffnet ist, entfällt.

```
Dim Trennung
Trennung = ”_”
```

Mit „Trennung“ definieren wir ein Zeichen bzw. eine Zeichenfolge, anhand derer später im Dateinamen bzw. im Instanznamen unterschieden werden kann, was der Kundenschlüssel und was der alte Namen des Bauteils bzw. der Baugruppe war; z.B. „Kundenschlüssel_Instanzname“ -> 0815_Ausgleichsblech.1.

```
UserName = InputBox(”Geben Sie die neue Bezeichnung ein”, makroname + ” ” +
version, ”4711”)
```

Der nächste Schritt ist die Eingabe des Kundenschlüssels durch den Anwender. Die Eingabe des Anwenders realisieren wir über eine Inputbox. Hier geben wir an, was der Anwender in das Eingabefeld eintragen soll – „Geben Sie die neue Bezeichnung ein“ –, und geben noch zusätzlich mit „4711“ schon mal eine mögliche Eingabe im Feld an.

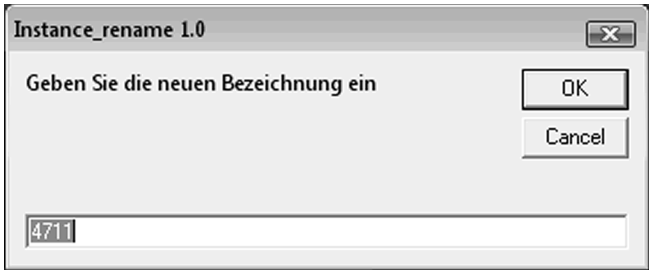

Bild 4.7:
Inputbox für
Anwendereingabe

```
If (UserName = ””) Then
   MsgBox ”Makro wurde abgebrochen”, 16, makroname & ” ” & version
   Exit Sub
End If
```

Hier hat der Anwender die Möglichkeit, die CANCEL-Taste zu drücken. Dies würde den Abbruch der Eingabeaufforderung darstellen, und aus der Inputbox würde kein

Text an das Makro übergeben. Um diese Situation abzufangen, fragen wir den eingegebenen Text der Inputbox mittels einer If-Anweisung ab. Sollte also die Eingabe des Anwenders ein leeren Text *(unabhängig von der Standardeingabe „4711“)* sein, wird das Makro nach einem entsprechenden Hinweis an den Anwender mit „Exit Sub“ beendet.

```
Dim targetfolder
targetfolder = CATIA.ActiveDocument.Path + "\" + UserName
CATIA.FileSystem.CreateFolder (targetfolder)
```

Es geht weiter mit den nächsten Schritten – dem Erzeugen eines Verzeichnisses, in dem die umbenannten Dateien gespeichert werden. Zwar könnte man die geänderten Dateien im selben Verzeichnis lassen wie die Originale, allerdings würde dies bei größeren Baugruppen schnell im Datenchaos enden – daher der eigene Ordner. Wir deklarieren eine neue Variable mit dem Namen TARGETFOLDER. Dieser Variablen weisen wir eine Zeichenfolge zu, die sich aus dem Pfad der aktuellen V5-Datei *(CATIA.ActiveDocument.Path)* und der Eingabe der Inputbox *(Username)* zusammensetzt. Erzeugt wird das Verzeichnis dann mit dem Befehl CATIA.FILESYSTEM.CREATE FOLDER(TARGETFOLDER).

```
CATIA.DisplayFileAlerts = False
```

Als Nächstes wird die Dateisicherungsmeldung von V5 ausgeschaltet. Diese wird immer wieder aufgerufen, wenn der Anwender bei der Sicherung der Datei eine Änderung von z.B. links oder das Überschreiben von Dateien bestätigen muss.

Die bisherigen Schritte waren quasi die Aufwärmübung; jetzt geht's ans Eingemachte. Alle Bauteile und Unterbaugruppen umzubenennen ist die eine Sache; wir müssen allerdings sicherstellen, dass alle Verknüpfungen innerhalb der Baugruppe(en) korrekt umgeschrieben werden. Da wir keinen direkten Einfluss auf Verknüpfungen in V5 nehmen können, müssen wir dies über eine intelligente Speicherung aller Bauteile und Baugruppen lösen.

Sonnen-Regel für das Linkmanagement

> Das CATPart steht als Sonne immer im Mittelpunkt unseres CATIA-Sonnensystems. Baugruppen (CATProduct) und Zeichnungen (CATDrawing) umkreisen das CATPart wie Planeten die Sonne. Ändert sich das CATPart, müssen sich auch das CATProduct und CATDrawing ändern – und zwar genau in dieser Reihenfolge: CATPart -> CATProduct -> CATDrawing

Nimmt man sich diese Regel zu Herzen, müssen demnach zuerst alle Bauteile umbenannt und gespeichert werden. Erst danach dürfen etwaige Unterbaugruppen umbenannt und gespeichert werden. Ganz zum Schluss kommt dann die eigentliche Hauptbaugruppe dran – unser aktives Dokument. Was ist also zu tun: Jede Instanz des aktiven Dokumentes aufrufen und prüfen, ob es ein Part oder ein Produkt ist. Wenn es ein Part ist, dann muss es umbenannt und unter neuem Namen gespeichert

werden. Das geht so lange, bis keine Parts mehr gefunden werden. Dasselbe dann noch einmal, allerdings dann mit den Unterbaugruppen. Aber schön der Reihe nach:

```
Set Products1 = CATIA.ActiveDocument.Product.Products
Dim Partnumber1, Filename
```

Wir listen erst einmal mit „PRODUCTS1“ alle Instanzen des aktiven Dokumentes auf. Dann deklarieren wir zwei neue Variablen mit den Namen „Partnumber1“ und „Filename“.

```
For i = 1 To CATIA.Documents.Count
  If (Right(CATIA.Documents.Item(i).Name, 7) = "CATPart") Then
    Partnumber1 = CATIA.Documents.Item(i).Product.PartNumber
```

Als Nächstes setzen wir eine For-Schleife auf, die als untere Grenze für die Laufvariable „i“ den Wert 1 und als Obergrenze die Summe aller geladenen Dokumente in V5 hat. Diese Schleife stellt somit unseren äußeren Rahmen dar, indem die ganzen Aktionen wie Namensvergabe und Speichern stattfinden. Gemäß unserer Sonnen-Regel müssen wir zuerst nur Bauteile (CATParts) betrachten. Mit der If-Abfrage nach dem Namen *(wie schon zu Anfang des Makros benutzt)* filtern wir in der laufenden Betrachtung eines Dokumentes ein CATPart aus. Mit Partnumber1 bekommen wir die aktuelle Teilenummer des Bauteils.

Bis jetzt haben wir es geschafft, aus der Liste aller geladenen Dateien jeweils ein CATPart auszufiltern. Nur – ist dieses CATPart auch wirklich in der aktuellen Baugruppe eingebaut? Diese Frage klären wir mittels Vergleich der ermittelten Teilenummer Partnumber1 und des Instanznamens des aktiven Dokumentes.

```
Dim index
For j = 1 To Products1.Count
  index = InStrRev(Products1.Item(j).Name, ".")
  If (Left(Products1.Item(j).Name, index - 1) = Partnumber1) Then
    Set instance1 = Products1.Item(j)
    Exit For
  End If
Next
```

In einer zweiten For-Schleife (innerhalb der ersten For-Schleife) lassen wir alle Instanzen durchlaufen und die jeweiligen Namen mit „Partnumber1“ vergleichen. Da Instanznamen allerdings mit „.1“ oder „.5“ enden *(z.B. 6kant-Schrb. M8x26.6),* würde ein direkter Textvergleich uns nicht weiter führen, da „Partnumber1“ diese Aufzählung am Ende nicht hat. Also müssen dieser Punkt und der Rest des Instanznamens für den Vergleich verschwinden. Dies erreichen wir, indem wir mit dem VB-Befehl INSTRREV die Stelle des Punktes im Instanznamen ermitteln und mit LEFT nur den Teil des Instanznamens bis zu dieser ermittelten Stelle auslesen und in der

5 Bauteile und Baugruppen

R7 Aktives Fenster dauerhaft drehen

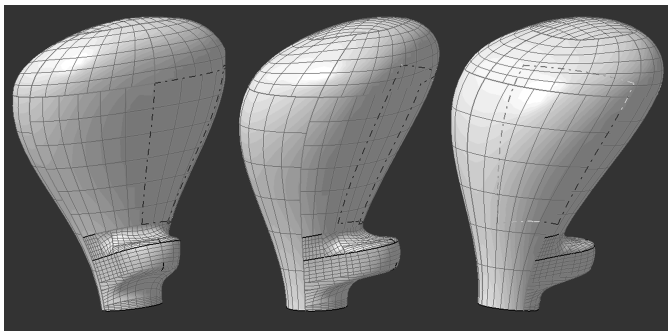

Bild 5.1:
Rotierender Schaltknauf

Gerade bei Messen oder Demos im eigenen Hause muss es auf dem Bildschirm immer schön bunt und interessant aussehen. Vor allem muss sich da was bewegen. Da bietet sich doch an, ein tolles Bauteil oder eine Baugruppe recht groß darzustellen und das Ding permanent langsam rotieren zu lassen. Wenn man nicht gerade einen Praktikanten oder eine Praktikantin dafür abstellen kann, ständig mit der Maus zu rotieren, ist dieses Makro hier genau das Richtige. Das Makro rotiert das Fenster mit dem tollen Bauteil so lange, bis jemand die ESCAPE-Taste drückt. Dabei kann sogar weiterhin das Bauteil bewegt und gedreht werden.

Makro-Typ:	VBA
V5-Level:	V5R16 und höher
Dialogsprache:	Deutsch
Voraussetzungen:	Es muss ein Dokument geöffnet sein.

Strategie

1. Deklarieren der nötigen Windows-Api
2. Ausschalten der Fehlerroutine
3. Prüfen, ob ein Dokument geöffnet ist
4. Hinweis zur ESCAPE-Taste an den Anwender ausgeben
5. Aktives Fenster deklarieren
6. Die senkrechte Achse des Fensters ermitteln
7. Schleife für Rotation erstellen
8. Innerhalb der Schleife die Escape-Taste abfragen

9. Fensterinhalt drehen
10. Prozess für zehn Millisekunden anhalten

Quellcode der Windows-Api

```
Declare Sub Sleep Lib "kernel32" (ByVal milliseconds As Long)
Private Declare Function GetAsyncKeyState Lib "user32" ( _
ByVal vKey As Long) As Integer
```

Dank der im Internet umfangreichen Fülle an Quellcode-Beispielen braucht man sich mit der Windows-Api selbst nicht mehr herumzuschlagen. Daher sei hier nur der nötige Quellcode gezeigt. Zum einen wird die Prozedur SLEEP benötigt, um eine Pause im Prozess zu realisieren. Die Funktion GetAsyncKeyState ist für das Abfragen der ESCAPE-Taste zuständig.

Durch die Verwendung der Windows-API ist das Makro leider nur als VBA-Makro zu verwenden.

Quellcode

```
Dim version, makroname
Sub CATMain()
version = "1.0"
makroname = " DemoRotor "
On Error Resume Next
Set activedoc = CATIA.ActiveDocument
If Err.Number <> 0 Then
  MsgBox "Es ist kein Dokument geöffnet", 16, makroname + " " + version
  Exit Sub
End If
```

Nach der Deklaration des Makronamens und der Version wird zuerst die Fehlerroutine mit ON ERROR RESUME NEXT ausgeschaltet. Dies ist notwendig, um herauszufinden, ob ein Dokument in V5 geöffnet ist. Dieser Test wird durchgeführt, indem versuchsweise das aktuelle Dokument mit SET ACTIVEDOC = CATIA.ACTIVEDOCUMENT angesprochen wird. Sollte es dabei zu einem Fehler kommen, da kein Dokument geöffnet ist, ist die resultierende Fehlernummer ERR.NUMBER ungleich 0. In diesem Fall wird der Anwender darüber informiert, und das Makro wird mit EXIT SUB beendet.

```
MsgBox "Abbruch der Rotation durch Drücken der ESCAPE-Taste", vbInformation,
makroname + " " + version
```

In der folgenden MessageBox wird der Anwender darüber informiert, dass er/sie die Roation durch Betätigen der Escape-Taste stoppen kann.

```
Set viewer1 = CATIA.Application.ActiveWindow.ActiveViewer
Set Viewpoint3D1 = viewer1.Viewpoint3D
Dim Direction(2)
Viewpoint3D1.GetUpDirection Direction
```

Als Nächstes wird über das aktive Fenster ACTIVEWINDOW der aktive „Viewer“ mit ACTIVEVIEWER deklariert. Der Viewer ist für die ganze grafische Darstellung innerhalb des Fensters zuständig, so z.B. auch für die Hintergrundfarbe. Anschließend wird der aktuelle Blickpunkt mit VIEWPOINT3D definiert. Von diesem Blickpunkt wird nun mittels GETUPDIRECTION die senkrechte Achse ermittelt. Um diese Achse wird später rotiert. Da das Fenster hier wie ein ebenes Bild fungiert, gibt es daher nur eine senkrechte und eine waagerechte Achse. Die Achse muss in einem Array mit drei Eintragsmöglichkeiten gespeichert werden. Dafür wird das Array DIRECTION deklariert und an die Funktion GETUPDIRECTION übergeben.

```
Do While Zeit <> ”Abbruch”
  If GetAsyncKeyState(vbKeyEscape) Then
    Zeit = ”Abbruch”
  End If
```

Als Nächstes wird eine Do While-Schleife benötigt, in der die Rotation permanent ablaufen soll. Dabei soll die Schleife so lange durchlaufen werden, bis für die Variable ZEIT der Wert „Abbruch“ definiert wurde. Der Wert „Abbruch“ wird dann gesetzt, wenn die Funktion GetAsyncKeyState() ermittelt, dass die ESCAPE-Taste gedrückt wurde. Für ESCAPE wird der Wert VBKEYESCAPE an die Funktion übergeben.

```
  DoEvents
  viewer1.Rotate Direction, 0.5
  viewer1.Update
  Sleep 10
Loop
```

Mit DOEVENTS wird sichergestellt, dass auch weiterhin Aktionen seitens des Anwenders möglich sind – sonst könnte ja ESCAPE nicht betätigt werden. Dann erfolgt endlich die eigentliche Rotation des Fensters mit der Funktion ROTATE. Als Parameter wird dabei die Achse der Rotation angegeben, die oben bereits ermittelt und im Array DIRECTION gespeichert wurde. Der zweite Parameter gibt den Winkel der Rotation mit „0,5“ Grad an. Sofort danach wird das Fenster aktualisiert, damit die Änderung auch zu sehen ist. Im Anschluss wird der laufende Prozess mit der eigenen Prozedur SLEEP angehalten. Der Parameter der Sleep-Prozedur definiert in Millisekunden, wie lange diese Pause dauern soll. Die Kombination zwischen der Winkelangabe der Rotation und der Pause in Millisekunden bestimmen die Qualität und Geschwindigkeit der Rotation. Je größer der Winkel, desto schneller die Rotation; je größer die Pause, desto „ruckeliger“ die Darstellung – hier ist ein bisschen Feintuning gefragt. Mit LOOP muss die DO WHILE-Schleife geschlossen werden.

10 Küche – das Verwalten von Makros

Es gibt viele Wege bzw. Möglichkeiten, sein Essen zuzubereiten. Je nachdem, ob man für sich selbst oder für Gäste kocht, sind der Aufwand und die Qualität recht unterschiedlich. Je nachdem, ob man zu Hause am eigenen Herd brutzelt, auf einem Campingkocher auf einer Wiese oder vielleicht sogar in einer Restaurantküche mit zwei bis drei Hilfsköchen eine kleine Speisekarte auswendig kochen muss, der Aufwand und der Geschmack richten sich stark nach den Gästen bzw. den Konsumenten unserer kulinarischen Ideen.

Genauso verhält es sich auch mit der Erstellung und Weitergabe von Makros. Je nachdem, ob ich ein paar Makros „auf Zuruf" für meine Kollegen oder für mich schreibe oder ob ich viele Makros als externer Dienstleister programmiere, der Aufwand bezüglich der Verwaltung variiert recht stark. Kommen dann noch verschiedene Sprachen für die Benutzeroberfläche oder bei der Meldung an den Benutzer hinzu oder vielleicht sogar verschiedene Versionen eines Makros, kommt man mit der einfachen Dateiablage selbst in dem strukturiertesten Ordner recht schnell an die Grenzen. Ich möchte daher eine Möglichkeit aufzeigen, wie man mit den Bordmitteln des VBA-Editors seine Makros – auch VBA-Makros – verwalten kann. Diese Möglichkeit ist weniger für das große Küchenteam von 20 Programmierern in einem Fünf-Sterne-Hotel geeignet. Es ist eher an den Restaurantkoch mit seinen Helferlein gerichtet, an die Hausfrau, die eine Großfamilie bekochen und dabei auf unterschiedlichste Einschränkungen Rücksicht nehmen muss, und an den ambitionierten Hobbykoch, der für viele seiner Mätressen „außer Haus" kochen geht und dabei immer die Vorlieben der jeweiligen Damen in Petto haben muss. Kurz: für alle Administratoren, Ein-Mann-Programmierer-Teams, Dienstleister; alle, die eine Anzahl von Makros je nach Wunsch oder Restriktionen verwalten und das richtige Makro in der richtigen Ausführung weitergeben müssen. Ganz ausgenommen sind bei den folgenden Betrachtungen Applikationen, die nicht mit CATIA-Bordmitteln erstellt werden – also auf dem .NET Framework basieren (C#, C++, VB.NET ...) oder auf Excel-Makros.

10.1 Mehrsprachigkeit

Seit ein paar Jahren reicht es nicht mehr aus, nur seine lokalen Mitarbeiter bei Laune zu halten. Ist das Unternehmen auch in anderen Ländern tätig, wurde versucht, möglichst viele Prozesse zu standardisieren. War anfangs von einem einheitlichen Prozess die Rede, der überall gleich funktionieren soll und alle Eventualitäten abdeckt, so haben die Praxis und die Zeit gezeigt, dass das so nicht ganz funktioniert. Einheitliche Konzernsprache z.B. ist zwar nützlich auf der Managerebene, aber der gemeine Konstrukteur redet/schreibt doch lieber in seiner Muttersprache. Zwar kann man eine V5-Schulung mit der englischen Sprachumgebung halten, die Schulungs-