Kapitel 2

Variablen, Datentypen und Konstanten

Variablen und Konstanten sind Werkzeuge,
die die Programmierung vereinfachen und strukturieren.

Das folgende Kapitel bildet die Voraussetzung für eine strukturierte Programmierung in Excel. Mit Variablen speichern Sie Informationen dauerhaft während der Laufzeit eines Makros, das heißt, Sie können Variablen auch mehrmals im Makro benutzen, indem Sie sie füllen und Werte hochzählen oder subtrahieren. Mit Konstanten legen Sie Informationen fest, die sich selten oder sogar nie ändern. Excel bietet für die Deklaration von Variablen und Konstanten eine ganze Auswahl an Datentypen. Je nach Aufgabe setzen Sie die vorgesehenen Datentypen ein.

2.1 Regeln für die Syntax von Variablen

Folgende Regeln gelten für die Benennung von Variablen:

- Das erste Zeichen muss aus einem Buchstaben bestehen. Als folgende Zeichen können Sie Buchstaben, Zahlen und einige Sonderzeichen verwenden.
- Sie dürfen keine Leerzeichen in einem Variablennamen verwenden. Wenn Sie einzelne Wörter trennen möchten, nehmen Sie dazu den Unterstrich, wie z. B. `Dim Miete_Januar as Currency`.
- Sonderzeichen wie #, %, &, ! oder ? sind nicht erlaubt.

Wenn Sie Ihre Variablennamen übersichtlich und auch aussagekräftig definieren möchten, empfiehlt sich die folgende Schreibweise:

```
Dim sTextMeldungFürFehler as String
```

Hier geht aus dem Namen der Variablen klar hervor, welchem Zweck sie dienen soll. Außerdem ist die Variable durch die Schreibweise leicht lesbar.

2.2 Variablen- und Datentypen

Variablen sollten immer zu Beginn eines Makros deklariert werden, also nach der `Sub`-Anweisung. Dabei spricht man von *lokalen Variablen*. Diese Variablen können nur in dem Makro verwendet werden, in dem sie deklariert wurden. Nachdem ein Makro durchgelaufen ist, wird eine solche Variable wieder aus dem Speicher gelöscht

Von *globalen Variablen* spricht man, wenn Sie sie allgemeingültig, also in mehreren Makros verwenden möchten. Dann muss die Variablendeklaration vor der `Sub`-Anweisung stattfinden.

Globale Variable können Sie für mehrere Makros verwenden. Sie werden nach dem Ende eines Makros nicht gelöscht und behalten ihren aktuellen Wert bei. Es gibt Beispiele, in denen diese Vorgehensweise sinnvoll ist. In den meisten Fällen sollten globale Variable aber weitestgehend vermieden werden, da sie wertvollen Speicherplatz auf dem Stapelspeicher belegen, was sich negativ auf das Laufzeitverhalten von Makros auswirken kann.

Eine Variablendeklaration beginnt immer mit der Anweisung `Dim`, gefolgt von einem Variablennamen, den Sie, unter Beachtung der unter Abschnitt 2.1 aufgeführten Regeln, frei wählen können. Danach geben Sie mit dem Schlüsselwort `As` an, welchen Datentyp die Variable erhalten soll. Tabelle 2.1 listet die gängigsten Datentypen auf.

Variablentyp	Wertebereich/Speicherbedarf
Byte	ganze Zahlen zwischen 0 und 255 (1 Byte)
Boolean	Wahrheitswert, entweder `True` oder `False` (2 Bytes)
Currency	Währungs-Datentyp: Festkommazahlen mit 15 Stellen vor und 4 Stellen nach dem Komma (8 Bytes)
Date	Datums- und Zeit-Datentyp (8 Bytes)
Decimal	Dezimalzahlen (14 Bytes)
Double	Fließkommazahlen mit einer Genauigkeit von 16 Stellen hinter dem Komma (8 Bytes)
Integer	ganze Zahlen zwischen –32.768 und +32.767 (2 Bytes)
Long	ganze Zahlen im Wertebereich von –2.147.483.648 bis +2.147.483.647 (4 Byte)

Tabelle 2.1 Die Datentypen für die Variablen

Variablentyp	Wertebereich/Speicherbedarf
Object	Datentyp gibt einen Verweis auf ein Objekt wieder. (4 Bytes)
Single	Fließkommazahlen mit einer Genauigkeit von 8 Stellen hinter dem Komma (4 Bytes)
String	der Datentyp für alle Texte (10 Bytes)
Variant	Standarddatentyp; wird automatisch gewählt, wenn kein anderer Datentyp definiert ist. (16 Bytes)

Tabelle 2.1 Die Datentypen für die Variablen (Forts.)

Möchten Sie möglicherweise die Variablennamen nicht mehr ganz so lang schreiben und auch bei der Datentyp-Anweisung weniger Schreibarbeit haben, dann verfahren Sie wie in Tabelle 2.2.

Ausführlich	Kurzform
`Dim Zähler as Integer`	`Dim Z%`
`Dim ZählerGroß as Long`	`Dim ZzGr&`
`Dim Betrag as Currency`	`Dim Bg@`
`Dim Meldung as String`	`Dim Meld$`

Tabelle 2.2 Variablendeklaration mit »DefType«

2.2.1 Variablendeklaration mit »DefType«

Wenn Sie eine größere Menge von Variablen des gleichen Typs verwenden, können Sie sich die Deklaration der einzelnen Variablen sparen, indem Sie die Anweisung `DefType` einsetzen. Dabei dürfen Sie diese Anweisung nicht innerhalb von Prozeduren, sondern nur auf Modulebene einsetzen.

So bedeutet die Anweisung

```
DefInt i-j
```

dass alle Variablen, die mit den Buchstaben `i` oder `j` beginnen, automatisch Integer-Variablen sein sollen. Entnehmen Sie Tabelle 2.3 weitere mögliche Anweisungen zu `DefType`.

Anweisung	Datentyp
DefBool	Boolean
DefByte	Byte
DefCur	Currency
DefDbl	Double
DefDate	Date
DefInt	Integer
DefLng	Long
DefObj	Object
DefStr	String
DefSng	Single
DefVar	Variant

Tabelle 2.3 »DefType«-Variablen

2.2.2 Statische Variablen

Sie haben die Möglichkeit, Variablen so zu definieren, dass sie über jedes Makroende hinaus »haltbar« sind. Sehen Sie sich dazu einmal die beiden folgenden Listings an.

```
Sub Variablen01()
  Dim i As Long

  i = i + 1
  MsgBox i
  End Sub
```

Listing 2.1 Variable zerfällt nach jedem Makroende.

In Listing 2.1 wird bei jedem Makrostart die Variable auf den Anfangswert 1 zurückgesetzt.

```
Sub Variablen02()
  Static i As Long
  i = i + 1
  MsgBox i
End Sub
```


Listing 2.2 Variable bleibt nach Makroende erhalten.

2.2.3 Private Variablen

Als Nächstes ist die Anweisung `Private` zu nennen. Setzen Sie diese Anweisung bei der Deklaration einer Variablen ein, ist diese für alle im Projekt befindlichen Makros gültig.

```
Private l As Long
Sub Variable03()
  l = l + 1
  MsgBox "Wert geändert von Sub Variable03: Wert:" & l
End Sub
```

Listing 2.3 Diese Variable kann nur von Makros im gleichen Modul geändert und abgefragt werden.

2.2.4 Öffentliche Variablen

In der Entwicklungsumgebung von Excel können Sie mehrere Module anlegen. Um Variablen modulübergreifend abfragen oder ändern zu können, müssen Sie sie als öffentlich deklarieren, und zwar mit der Anweisung `Public`.

```
Public l As Long

Sub Variable04()
  l = l + 1
  MsgBox "Wert geändert von Sub Variable04: Wert:" & l
End Sub
```

Listing 2.4 Diese Variable kann von allen Modulen des Projekts aufgerufen werden.

2.3 Variablendeklarationen erzwingen

Sie können Excel so einstellen, dass jede Variable vor ihrer ersten Verwendung deklariert werden muss. Vorher läuft kein einziges Makro an, sofern es mit Variablen arbeitet, die zuvor nicht deklariert wurden. Um diese wichtige Einstellung vorzunehmen, wechseln Sie in die Entwicklungsumgebung und rufen den Befehl Extras • Optionen auf. Wechseln Sie auf die Registerkarte Editor, und aktivieren Sie das Kontrollkästchen Variablendeklaration erforderlich.

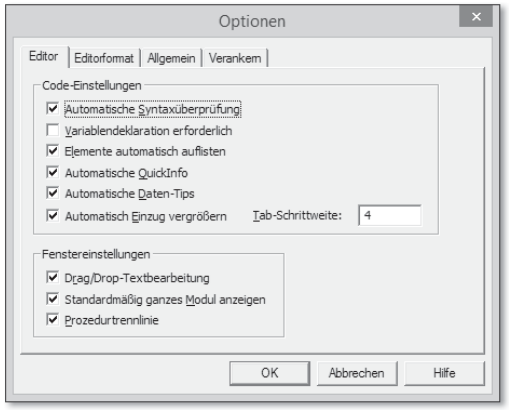

Abbildung 2.1 Variablendeklaration erzwingen

2.4 Die Konstanten

Im Gegensatz zu den Variablen ändern Konstanten ihre Werte nie und bleiben während der Programmausführung immer konstant. Auch hier wird zwischen lokalen und globalen Konstanten unterschieden. Globale Konstanten werden außerhalb der einzelnen Makros definiert und sind damit für alle Makros im Modul verwendbar. Lokale Konstanten hingegen gelten nur in dem Makro, in dem sie definiert wurden. Wie schon bei den Variablen sollten Sie darauf achten, nicht allzu viele globale Konstanten zu verwenden, da sich dies merklich auf Ihren Speicher auswirkt.

Nachfolgend ein paar typische Deklarationen mit Konstanten:

```
Const Mappe = "Mappe1.xls"
Const StartDatum = #1/1/2015#
Const Fehlermeldung1 =  _
 "Fehler beim Drucken aufgetreten!"
Const MWST = 1.19
```

Was kann hier noch verbessert werden? Was für die Variable gilt, hat auch bei den Konstanten Konsequenzen. In den obigen Beispielen ist noch nicht erklärt worden, welche Datentypen verwendet werden sollen. Zum aktuellen Zeitpunkt wird in allen vier Beispielen der Datentyp Variant eingesetzt. Es geht auch etwas genauer und speichersparender:

```
Const Mappe as String = "Mappe1.xls"
Const StartDatum As Date = #1/1/2015#
Const Fehlermeldung1 as String = _
 "Fehler beim Drucken!"
Const MWST as Single = 1.19
```


2.5 Die Objektvariable »Range«

Über die Objektvariable `Range` können Sie einzelne Zellen, ganze Zeilen, Spalten und Bereiche ansprechen.

> **Hinweis**
>
> Weitere Methoden und Eigenschaften zum Objekt `Range` finden Sie in Kapitel 9, »›Range‹-Objekt«.

2.5.1 »Value«-Eigenschaft

Über die Eigenschaft `Value` können Sie den Inhalt einer Zelle auslesen und auch den Inhalt für eine Zelle festlegen.

Syntax

```
Ausdruck.Value(RangeValueDataType)
```

Argument	Beschreibung
`Ausdruck`	Erforderlich. Ein Ausdruck, der ein `Range`-Objekt zurückgibt
`RangeValueDataType`	Optionaler Variant-Wert. Der Datentyp des Bereichswertes. Dies kann eine `xlRangeValueDataType`-Konstante sein.

Konstante	Beschreibung
`xlRangeValueDefault`	Standard. Wenn das angegebene `Range`-Objekt leer ist, wird der Wert `Empty` zurückgegeben (verwenden Sie die Funktion `IsEmpty`, um diesen Fall zu prüfen). Wenn das `Range`-Objekt mehr als eine Zelle enthält, gibt es eine Matrix von Werten zurück (verwenden Sie die Funktion `IsArray`, um diesen Fall zu prüfen).
`xlRangeValueMSPersistXML`	Gibt die Recordset-Darstellung des angegebenen `Range`-Objektes im XML-Format zurück.
`xlRangeValueXMLSpreadsheet`	Gibt die Werte, Formatierungen, Formeln und Namen des angegebenen `Range`-Objektes im XML-Tabellenformat zurück.

Tabelle 2.4 Die Konstanten für »RangeValueDataType«

Beispiele

Im nächsten Beispiel wird der Inhalt der Zelle A1 aus *Tabelle1* abgefragt.

```
Sub ZelleAuslesen()
 Dim Zelle As Range

 Set Zelle = Sheets("Tabelle1").Range("A1")
 Debug.Print Zelle.Value
End Sub
```

Listing 2.5 Zellinhalt abfragen

Über die Anweisung `Set` stellen Sie die Verbindung zum Objekt her. Über die Objektvariable `Zelle` vom Typ `Range` können Sie danach auf alle Eigenschaften und Methoden zurückgreifen, die für Zellen zur Verfügung stehen.

Im Makro aus Listing 2.6 schreiben Sie den Wert `0` in die Zelle A1 in *Tabelle1*.

```
Sub ZelleFüllen()
  Dim Zelle As Range

  Set Zelle = Sheets("Tabelle1").Range("A1")
  Zelle.Value = 0
End Sub
```

Listing 2.6 Zelle füllen

2.5.2 »Address«-Eigenschaft

Über die Eigenschaft `Address` aus Listing 2.7 ermitteln Sie die Adresse einer Zelle oder eines Bereichs.

Syntax

```
Ausdruck.Address(RowAbsolute, ColumnAbsolute,
ReferenceStyle, External, RelativeTo)
```

Argument	Beschreibung
`Ausdruck`	Erforderlich. Ein Ausdruck, der ein `Range`-Objekte zurückgibt
`RowAbsolute`	Optional. Bei `True` wird der Bezugsteil mit der Zeilenangabe als absoluter Bezug zurückgegeben.

Argument	Beschreibung
`ColumnAbsolute`	Optional. Bei `True` wird der Bezugsteil mit der Spaltenangabe als absoluter Bezug zurückgegeben.
`ReferenceStyle`	Standard: `xlA1`. Verwenden Sie `xlA1`, um einen Bezug in der A1-Schreibweise zurückzugeben, `xlR1C1`, um einen Bezug in der Z1S1-Schreibweise zurückzugeben.
`External`	Optional. Wenn `True`, wird ein externer Bezug zurückgegeben. Wenn `False`, wird ein lokaler Bezug zurückgegeben.
`RelativeTo`	Optional. Sind `RowAbsolute` und `ColumnAbsolute` beide `False` und geben Sie für `ReferenceStyle` den Wert `xlR1C1` an, müssen Sie mit `RelativeTo` einen Startpunkt für den relativen Bezug angeben. Dieses Argument ist ein `Range`-Objekt.

Beispiel

```
Sub ZellenAdresse()
 Dim Zelle As Range

 Set Zelle = Sheets("Tabelle1").Range("A1")
 Debug.Print Zelle.Address
 Debug.Print Zelle.Address(RowAbsolute:=False)
 Debug.Print Zelle.Address(ColumnAbsolute:=False)
 Debug.Print Zelle.Address(ReferenceStyle:=xlR1C1)
End Sub
```

Listing 2.7 Zellenadresse ermitteln

Abbildung 2.2 Zellenbezüge ausgeben

2.5.3 »Formula«-Eigenschaften

Über die Eigenschaften `Formula` bzw. `FormulaLocal` können Sie die Formel, Funktion oder auch eine Verknüpfung aus einer Zelle herauslesen.

Syntax

```
Ausdruck.Formula
```

Das erforderliche Argument `Ausdruck` ist ein Ausdruck, der ein `Range`-Objekt darstellt.

Beispiel

Das Beispiel aus Listing 2.8 ermittelt die Formel `Summe` aus Zelle A5 aus *Tabelle1*.

```
Sub ZellenFormel()
 Dim Zelle As Range

 Set Zelle = Sheets("Tabelle1").Range("A5")
 Debug.Print Zelle.Formula
 Debug.Print Zelle.FormulaLocal
End Sub
```

Listing 2.8 Formel aus einer Zelle ziehen

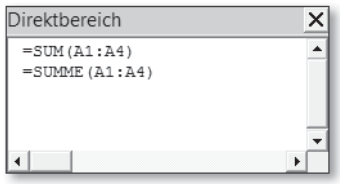

Abbildung 2.3 Formeltext aus einer Zelle auslesen

2.5.4 »Clear«-Methode

Über die Methode `Clear` löschen Sie den Inhalt einer Zelle, Zeile oder Spalte.

Syntax

```
Ausdruck.Clear
```

Das erforderliche Argument `Ausdruck` ist ein Ausdruck, der ein `Range`-Objekt darstellt.

Beispiel

Im Beispiel aus Listing 2.9 wird jeglicher Inhalt der Zeile 1 gelöscht.

```
Sub ZeileninhaltLöschen()
 Dim Zeile As Range
 Set Zeile = Sheets("Tabelle1").Rows(1)
 Zeile.Clear
End Sub
```

Listing 2.9 Inhalt einer ganzen Zeile löschen

2.5.5 »AutoFit«-Methode

Mit der Methode `AutoFit` bestimmen Sie die Breite einer oder mehrerer Spalten nach dem tatsächlichen Platzbedarf.

Syntax

```
Ausdruck.AutoFit
```

Das erforderliche Argument `Ausdruck` ist ein Ausdruck, der ein `Range`-Objekt darstellt.

Beispiel

Im Beispiel aus Listing 2.10 wird die Spaltenbreite für die Spalten A bis D in *Tabelle1* optimal eingestellt.

```
Sub Spaltenbreiteanpassen()
 Dim Spalte As Range

 Set Spalte = Sheets("Tabelle1").Columns("A:D")
 Spalte.AutoFit
End Sub
```

Listing 2.10 Spaltenbreite einstellen über »AutoFit«

2.5.6 »ColorIndex«-Eigenschaft

Über die Eigenschaft `Colorindex` legen Sie die Hintergrundfarbe von Zellen und Bereichen fest. Zur Verfügung stehen 56 Standardfarben, von denen Sie jede über einen eindeutigen Index ansprechen können.

Syntax

```
Ausdruck.ColorIndex
```

Das erforderliche Argument `Ausdruck` ist ein Ausdruck, der ein `Interior`-Objekt (Hintergrund), `Font`-Objekt (Schrift) oder ein `Border`-Objekt (Rahmen) darstellt.

Beispiel

Im Beispiel aus Listing 2.11 wird einer Objektvariablen ein Bereich zugewiesen und anschließend die Eigenschaft `ColorIndex` auf den Hintergrund des Bereiches angewendet, um diesen einzufärben.

```
Sub BereichFärben()
 Dim Bereich As Range

 Set Bereich = Sheets("Tabelle1").Range("A1:D5")
 Bereich.Interior.ColorIndex = 3
End Sub
```

Listing 2.11 Bereich einfärben über »ColorIndex«

2.5.7 »Union«-Methode

Über die Methode `Union` können Sie mehrere Bereiche in einer Tabelle zusammenfassen.

Syntax

```
Ausdruck.Union(Arg1, Arg2, ...)
```

Argumente	Beschreibung
`Ausdruck`	Das erforderliche Argument `Ausdruck` ist ein Ausdruck, der ein `Application`-Objekt darstellt.
`Arg1`, `Arg2`	Erforderlich. Es müssen mindestens zwei `Range`-Objekte angegeben werden.

Beispiel

Im Makro aus Listing 2.12 werden drei nicht zusammenhängende Bereiche zu einem Gesamtbereich zusammengefasst und dann mit einem Wert gefüllt.

```
Sub MehrereBereiche()
 Dim Bereich1 As Range
 Dim Bereich2 As Range
 Dim Bereich3 As Range
 Dim Gesamt As Range
```



```
With Worksheets("Tabelle3")
 Set Bereich1 = .Range("A1:A5")
 Set Bereich2 = .Range("C1:C5")
 Set Bereich3 = .Range("E1:E5")
 Set Gesamt = Union(Bereich1, Bereich2, Bereich3)
 Gesamt.Value = 0 With
End
End Sub
```

Listing 2.12 Mit »Union« einen Gesamtbereich bilden

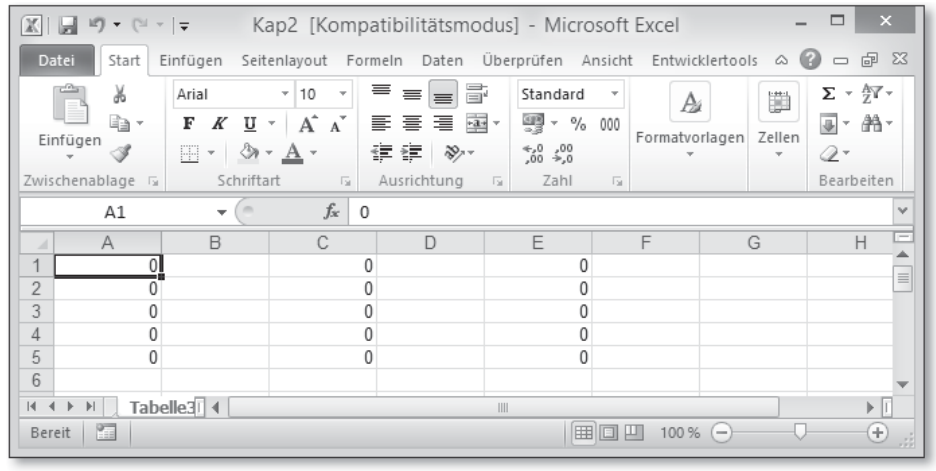

Abbildung 2.4 Einen Gesamtbereich ansprechen

2.6 Die Objektvariable »Comment«

Über die Objektvariable `Comment` können Sie Kommentare in Zellen einfügen, ändern, löschen und vieles mehr.

2.6.1 »AddComment«-Methode

Die Methode `AddComment` fügt dem Bereich einen Kommentar hinzu.

Syntax

```
Ausdruck.AddComment(Text)
```

Argument	Beschreibung
Ausdruck	Erforderlich. Ein Ausdruck, der ein Range-Objekt zurückgibt.
Text	Optional. Der Kommentartext

Beispiel

Das Beispiel aus Listing 2.13 fügt in Zelle A1 in *Tabelle2* einen Kommentar ein.

```
Sub KommentarAnlegen()
 Dim Cmt As Comment

 Set Cmt = Sheets("Tabelle2").Range("A1").AddComment
 Cmt.Text "Neuer Kommentar"
End Sub
```

Listing 2.13 Kommentar einfügen über die Methode »AddComment«

Die Methode `AddComment` fügt ein noch leeres Kommentarfenster in Zelle A1 in *Tabelle2* ein. Den Text des Kommentars bestimmen Sie über die Methode `Text`.

2.6.2 »Text«-Eigenschaft

Die Eigenschaft `Text` legt den Kommentartext fest.

Syntax

```
Ausdruck.Text(Text, Start, Overwrite)
```

Argument	Beschreibung
Ausdruck	Erforderlich. Ein Ausdruck, der eines der oben aufgeführten Objekte zurückgibt.
Text	Optional. Der hinzuzufügende Text
Start	Optional. Die Nummer des Zeichens, an dem der neue Text eingefügt wird. Wird dieses Argument nicht angegeben, wird bereits bestehender Text im Kommentar gelöscht.
Overwrite	Optional. Wenn `True`, wird bereits bestehender Text überschrieben.

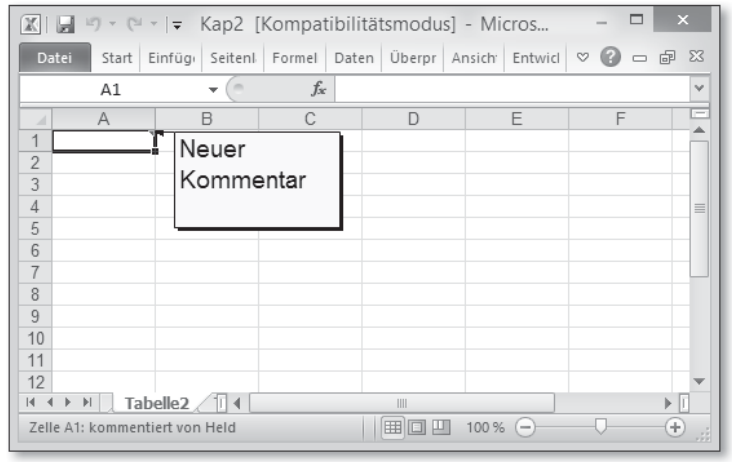

Abbildung 2.5 Kommentar einfügen und beschriften

2.6.3 »Parent«-Eigenschaft

Um die Zellenadresse eines Kommentars zu ermitteln, setzen Sie die Eigenschaft `Address` ein. Diese Eigenschaft haben wir bereits besprochen. Damit dies auch funktioniert, muss diese Eigenschaft mit Hilfe der Eigenschaft `Parent` auf das übergeordnete Objekt angewendet werden.

Syntax

```
Ausdruck.Parent
```

Das erforderliche Argument `Ausdruck` ist ein Ausdruck, der ein übergeordnetes Objekt darstellt.

Beispiel

Das Beispiel aus Listing 2.14 fragt einen Kommentar auf einer Tabelle ab. Dabei sollen die Zellenadresse sowie der Inhalt des Kommentars ermittelt werden.

```
Sub KommentarAuslesen()
 Dim Cmt As Comment

 Set Cmt = Sheets("Tabelle2").Range("A1").Comment
 MsgBox Cmt.Parent.Address & vbLf & _
        Cmt.Text, vbInformation
End Sub
```

Listing 2.14 Kommentarinhalt auslesen

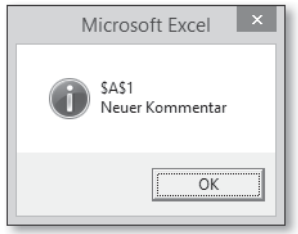

Abbildung 2.6 Kommentarinhalt auslesen

2.7 Die Objektvariable »Worksheet«

Über die Objektvariable `Worksheet` können Sie Tabellen ansprechen und programmieren.

> **Hinweis**
>
> Weitere Methoden, Eigenschaften zum Objekt `Worksheet` finden Sie in Kapitel 8, »›Worksheet‹-Objekt«.

2.7.1 »Name«-Eigenschaft

Über die Eigenschaft `Name` lesen Sie den Namen einer Tabelle aus.

Syntax

```
Ausdruck.Name
```

Das erforderliche Argument `Ausdruck` ist ein Ausdruck, der ein `Worksheet`-Objekt darstellt.

Beispiele

Das folgende Beispiel ermittelt den Namen der ersten Tabelle in der Arbeitsmappe.

```
Sub TabelleAnsprechen()
 Dim Tabelle As Worksheet

 Set Tabelle = Sheets(1)
 MsgBox Tabelle.Name
End Sub
```

Listing 2.15 Erste Tabelle in der Mappe identifizieren

Im Beispiel aus Listing 2.16 wird die erste Tabelle in der Arbeitsmappe umbenannt.

```
Sub TabelleBenennen()
 Dim Tabelle As Worksheet

 Set Tabelle = Worksheets.(1)
 Tabellen.Name = "Dokumentation"
End Sub
```

Listing 2.16 Tabelle umbenennen

2.7.2 »Copy«-Methode

Über die Methode `Copy` kopieren Sie eine Tabelle.

Syntax

```
Ausdruck.Copy(Before, After)
```

Argument	Beschreibung
`Ausdruck`	Das erforderliche Argument `Ausdruck` ist ein Ausdruck, der ein `Worksheet`-Objekt darstellt.
`Before`	Optional. Das Blatt, vor das dieses Blatt kopiert wird. Sie können `Before` nicht angeben, wenn Sie `After` angeben.
`After`	Optional. Das Blatt, hinter das dieses Blatt kopiert wird. Sie können `After` nicht angeben, wenn Sie `Before` angegeben haben.

Beispiele

Im Beispiel aus Listing 2.17 wird die erste Tabelle der Arbeitsmappe in eine neue Arbeitsmappe kopiert.

```
Sub TabelleKopierenAndereMappe()
 Dim Tabelle As Worksheet

 Set Tabelle = Sheets(1)
 Tabelle.Copy
End Sub
```

Listing 2.17 Tabelle in neue Mappe kopieren

Soll die Kopie der Tabelle in derselben Arbeitsmappe verbleiben, dann geben Sie entweder das Argument `Before` oder das Argument `After` an. Im Beispiel aus Listing 2.18

wird eine Tabelle kopiert und danach an die erste Stelle in der Arbeitsmappe angeordnet.

```
Sub TabelleBeginnMappe()
 Dim Tabelle As Worksheet

 Set Tabelle = Sheets(1)
 Tabelle.Copy Before:=Worksheets(1)
End Sub
```

Listing 2.18 Tabelle kopieren und am Anfang der Mappe einfügen

2.7.3 »Move«-Methode

Über die Methode `Move` verschieben Sie eine Tabelle an eine andere Position in der Arbeitsmappe.

Syntax

```
Ausdruck.Move(Before, After)
```

Argument	Beschreibung
`Ausdruck`	Ausdruck, der ein `Worksheet`-Objekte zurückgibt.
`Before`	Optional. Das Blatt, vor das dieses Blatt geschoben wird. Sie können `Before` nicht angeben, wenn Sie `After` angeben.
`After`	Optional. Das Blatt, hinter das dieses Blatt geschoben wird. Sie können `After` nicht angeben, wenn Sie `Before` angeben.

Beispiele

Im Beispiel aus Listing 2.19 wird die erste Tabelle in der Arbeitsmappe ans Ende der Mappe verschoben.

```
Sub TabelleAnsEndeVerschieben()
 Dim Tabelle As Worksheet

 Set Tabelle = Worksheets(1)
 Tabelle.Move After:=Worksheets(Worksheets.Count)
End Sub
```

Listing 2.19 Erste Tabelle ans Ende der Mappe verschieben

Soll die letzte Tabelle einer Arbeitsmappe an den Beginn der Mappe verschoben werden, dann starten Sie das Makro aus Listing 2.20.

```
Sub LetzteTabelleAnBeginnVerschieben()
 Dim Tabelle As Worksheet

 Set Tabelle = Worksheets(Worksheets.Count)
 Tabelle.Move Before:=Worksheets(1)
End Sub
```

Listing 2.20 Letzte Tabelle an den Beginn der Mappe verschieben

2.7.4 »Add«-Methode

Die Methode `Add` erstellt ein neues Arbeitsblatt, Diagramm oder Makroblatt. Das neue Arbeitsblatt wird zum aktiven Blatt.

Syntax

```
Ausdruck.Add(Before, After, Count, Type)
```

Argument	Beschreibung
`Ausdruck`	Ausdruck, der ein `Worksheet`-Objekt zurückgibt
`Before`	Optional. Ein Objekt, das das Blatt festlegt, vor dem das neue Blatt eingefügt werden soll
`After`	Optional. Ein Objekt, das das Blatt festlegt, nach dem das neue Blatt eingefügt werden soll
`Count`	Optional. Die Anzahl der hinzuzufügenden Blätter. Der Standardwert ist `1`.
`Type`	Optional. Legt den Blatttyp fest. Kann eine der folgenden `XlSheetType`-Konstanten sein: `xlWorksheet`, `xlChart`, `xlExcel4MacroSheet` oder `xlExcel4IntlMacroSheet`. Wenn Sie ein Blatt einfügen, das auf einer vorhandenen Vorlage basiert, müssen Sie den Pfad zu der Vorlage angeben. Der Standardwert ist `xlWorksheet`.

Beispiele

Das Beispiel aus Listing 2.21 fügt eine neue Tabelle ganz am Ende der Arbeitsmappe ein.

```
Sub TabelleEinfügen()
 Dim Tabelle As Worksheet

 Set Tabelle = Worksheets.Add _
 (After:=Worksheets(Worksheets.Count))
End Sub
```

Listing 2.21 Neue Tabelle am Ende der Mappe einfügen

Soll die neue Tabelle gleich zu Beginn der Arbeitsmappe eingefügt werden, dann starten Sie das Makro aus Listing 2.22.

```
Sub TabelleEinfügenStart()
 Dim Tabelle As Worksheet

 Set Tabelle = Worksheets.Add(Before:=Worksheets(1))
End Sub
```

Listing 2.22 Neue Tabelle am Anfang der Mappe einfügen

Hinweis

Wenn Sie die Argumente `Before` und `After` nicht angeben, wird das neue Blatt vor dem aktiven Blatt eingefügt.

2.7.5 »Delete«-Methode

Über die Methode `Delete` entfernen Sie eine Tabelle aus einer Arbeitsmappe.

Syntax

```
Ausdruck.Delete()
```

Das erforderliche Argument `Ausdruck` ist ein Ausdruck, der ein `Worksheet`-Objekt darstellt.

Beispiele

Im Beispiel aus Listing 2.23 wird die erste Tabelle in einer Arbeitsmappe gelöscht.

```
Sub TabelleLöschen()
 Dim Tabelle As Worksheet
 Set Tabelle = Worksheets(1)
 Tabelle.Delete
End Sub
```

Listing 2.23 Tabelle entfernen über die Methode »Delete«

Wenn Sie dieses Makro starten, dann erfolgt eine Rückfrage, ob die Tabelle wirklich gelöscht werden soll.

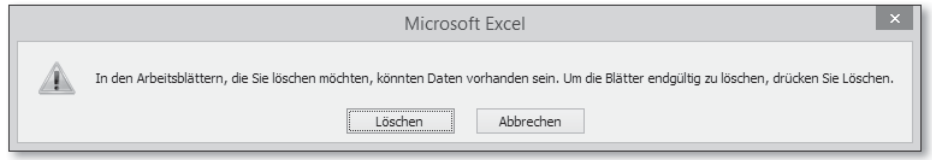

Abbildung 2.7 Rückfrage bestätigen

Da diese Rückfrage besonders bei größeren Software-Projekten recht lästig ist, kann diese Meldung übergangen werden, indem Sie das Makro aus Listing 2.24 starten.

```
Sub TabelleLöschenOhneRückfrage()
 Dim Tabelle As Worksheet

 Set Tabelle = Worksheets(1)
 Application.DisplayAlerts = False
 Tabelle.Delete
 Application.DisplayAlerts = True
End Sub
```

Listing 2.24 Tabelle ohne Rückfrage löschen

2.8 Die Objektvariable »Picture«

Über die Objektvariable `Picture` sprechen Sie Bilder in Excel-Tabellen an.

2.8.1 »Insert«-Methode

Mit der Methode `Insert` fügen Sie eine Bilddatei in eine Excel-Tabelle ein.

Syntax

```
Ausdruck.Insert
```

Das erforderliche Argument `Ausdruck` ist ein Ausdruck, der ein `Picture`-Objekt darstellt.

Beispiel

Das Beispiel aus Listing 2.25 fügt eine Grafikdatei in eine Tabelle ein.

```
Sub BildEinfügen()
 Dim Bild As Picture

 Set Bild = _
 Worksheets("Tabelle1").Pictures.Insert _("C:\Excel.jpg")
End Sub
```

Listing 2.25 Bild in Tabelle einfügen über die Methode »Insert«

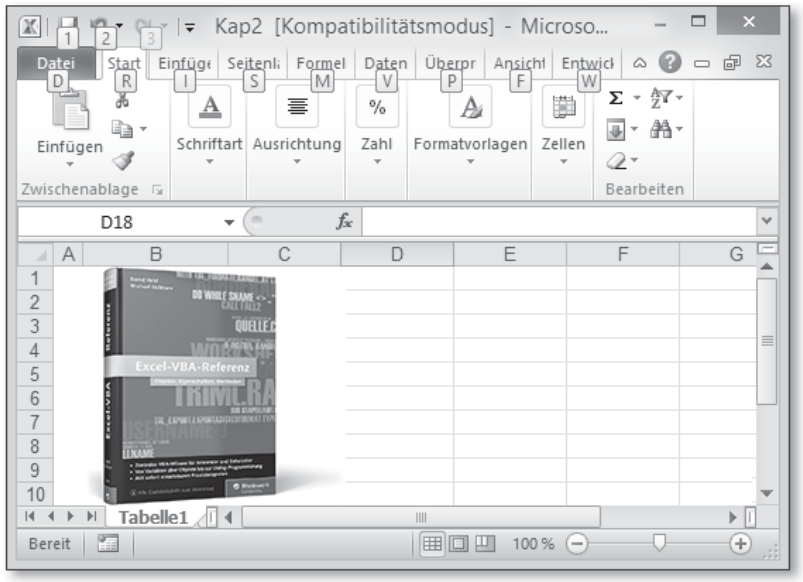

Abbildung 2.8 Bild in eine Tabelle einfügen

2.8.2 »Top«-Eigenschaft

Die Eigenschaft `Top` repräsentiert die linke obere Ecke einer Zelle.

Syntax

```
Ausdruck.Top
```

Das erforderliche Argument `Ausdruck` ist ein Ausdruck, der ein `Picture`-Objekt darstellt.

2.8.3 »Left«-Eigenschaft

Die Eigenschaft `Left` repräsentiert den Abstand von der linken Seite des Objekts zur linken Seite von Spalte A.

Syntax

```
Ausdruck.Left
```

Das erforderliche Argument `Ausdruck` ist ein Ausdruck, der ein `Picture`-Objekt darstellt.

Beispiel

Im Beispiel aus Listing 2.26 wird eine Grafik, die sich bereits in einer Tabelle befindet, neu angeordnet.

```
Sub BildPositionieren()
 Dim Bild As Picture

 Set Bild =
  Set Bild = _
   Worksheets("Tabelle1").Pictures(1)
   Bild.Top = Range("C3").Top
   Bild.Left = Range("C3").Left
End Sub
```

Listing 2.26 Bild neu in der Tabelle anordnen

2.8.4 »Width«-Eigenschaft

Die Eigenschaft `Width` gibt die Breite eines Objektes zurück bzw. legt diese fest.

Syntax

```
Ausdruck.Width
```

Das erforderliche Argument `Ausdruck` ist ein Ausdruck, der ein `Picture`-Objekt darstellt.

2.8.5 »Height«-Eigenschaft

Die Eigenschaft `Height` gibt die Höhe eines Objektes zurück bzw. legt diese fest.

Syntax

```
Ausdruck.Height
```

Das erforderliche Argument `Ausdruck` ist ein Ausdruck, der ein `Picture`-Objekt darstellt.

Beispiele

Das Beispiel aus Listing 2.27 legt die Breite sowie die Höhe eines Bildes fest. Dabei wird das Bild auf die doppelte Größe gebracht.

```
Sub BildVergrößern()
 Dim Bild As Picture

 Set Bild = _
 Worksheets("Tabelle1").Pictures(1)
 Bild.Width = Bild.Width * 2
 Bild.Height = Bild.Height * 2
End Sub
```

Listing 2.27 Bild vergrößern

Soll das Bild hingegen verkleinert werden, dann starten Sie das Makro aus Listing 2.28. Dabei wird die Größe des Bildes auf die Hälfte reduziert.

```
Sub BildVerkleinern()
 Dim Bild As Picture

 Set Bild = _
 Worksheets("Tabelle1").Pictures(1)
 Bild.Width = Bild.Width * 0.5
 Bild.Height = Bild.Height * 0.5
End Sub
```

Listing 2.28 Bild verkleinern

2.8.6 »Duplicate«-Methode

Die Methode `Duplicate` dupliziert das Objekt und gibt einen Bezug auf die neue Kopie zurück.

Syntax

```
Ausdruck.Duplicate
```

Das erforderliche Argument `Ausdruck` ist ein Ausdruck, der ein `Picture`-Objekt darstellt.

Beispiel

Beim Beispiel aus Listing 2.29 wird eine Grafik dupliziert und das Duplikat dann drei Spalten weiter rechts angeordnet.

```
Sub BildDuplizieren()
 Dim Bild As Picture
 Dim BildNeu As Picture

 Set Bild = _
 Worksheets("Tabelle1").Pictures(1)
 Bild.Top = Range("C3").Top
 Bild.Left = Range("C3").Left

 Set BildNeu = Bild.Duplicate
 BildNeu.Top = Range("C3").Offset(0, 3).Top
 BildNeu.Left = Range("C3").Offset(0, 3).Left
End Sub
```

Listing 2.29 Bild wird dupliziert und neu angeordnet

2.8.7 »CopyPicture«-Methode

Die Methode `CopyPicture` kopiert das ausgewählte Objekt als Bild in die Zwischenablage.

Syntax

```
Ausdruck.CopyPicture(Appearance, Format, Size)
```

Argument	Beschreibung
`Ausdruck`	Das erforderliche Argument `Ausdruck` ist ein Ausdruck, der ein `Picture`-Objekt darstellt.
`Appearance`	Optional. Legt fest, wie das Bild kopiert werden soll.
`Format`	Optional. Das Bildformat

Argument	Beschreibung
Size	Optional. Gibt die Größe der kopierten Grafik an, wenn das Objekt ein Diagramm ist, das sich auf einem Diagrammblatt befindet und nicht in einem Arbeitsblatt eingebettet ist.

Als Appearance-Wert stehen Ihnen die Konstanten aus Tabelle 2.5 zur Verfügung.

Konstante	Beschreibung
xlPrinter	Das Bild wird so kopiert, wie es beim Drucken dargestellt wird.
xlScreen	Standard. Das Bild wird so kopiert, dass es seiner Darstellung auf dem Bildschirm so weit wie möglich entspricht.

Tabelle 2.5 Die möglichen »Appearance«-Konstanten

Als Format-Wert, mit dem Sie das Bildformat festlegen, stehen Ihnen die Konstanten aus Tabelle 2.6 zur Verfügung.

Konstante	Beschreibung
xlBitmap	Bild wird als Bitmap gespeichert.
xlPicture	Standard

Tabelle 2.6 Die möglichen »Format«-Konstanten

Als Size-Wert, mit dem Sie die Größe bestimmen, stehen Ihnen die Konstanten aus Tabelle 2.7 zur Verfügung.

Konstante	Beschreibung
xlPrinter	Standard. Das Bild wird so kopiert, dass es seiner gedruckten Größe so weit wie möglich entspricht.
xlScreen	Das Bild wird so kopiert, dass es seiner Darstellung auf dem Bildschirm so weit wie möglich entspricht.

Tabelle 2.7 Die möglichen »Size«-Konstanten

Beispiel

Das Beispiel aus Listing 2.30 kopiert ein Bild aus *Tabelle1* und fügt es in Zelle A1 von *Tabelle2* ein.


```
Sub BildKopieren()
 Dim Bild As Picture

 Set Bild = _
 Worksheets("Tabelle1").Pictures(1)
 Bild.CopyPicture
 Worksheets("Tabelle2").Paste _
 Destination:=Worksheets("Tabelle2").Range("A1")
End Sub
```

Listing 2.30 Bild kopieren und in andere Tabelle einfügen

2.9 Die Objektvariable »Shape«

Jedes Shape-Objekt stellt ein Objekt in der Zeichnungsebene dar, wie z. B. eine AutoForm, eine Freihandform, ein OLE-Objekt oder ein Bild.

2.9.1 Methode »AddCallout«

Die Methode AddCallout erstellt eine rahmenlose Legende mit Linie. Die Methode gibt ein Shape-Objekt zurück, das die neue Legende darstellt.

Syntax

```
Ausdruck.AddCallout(Type, Left, Top, Width, Height)
```

Argument	Beschreibung
Ausdruck	Dieses Argument gibt ein Shape-Objekt zurück.
Type	Erforderlich. Legt den Typ einer Legendenlinie fest.
Left	Erforderlich. Die Position (in Punkt) der oberen linken Ecke des Begrenzungsrechtecks der Legende, relativ zur oberen linken Ecke des Dokuments
Top	Erforderlich. Die Position (in Punkt) der oberen linken Ecke des Begrenzungsrechtecks der Legende, relativ zur oberen linken Ecke des Dokuments
Width	Erforderlich. Breite (in Punkt) des Begrenzungsrechtecks der Legende
Height	Erforderlich. Höhe (in Punkt) des Begrenzungsrechtecks der Legende

Folgende Konstanten sind für das Argument Type möglich:

Konstante	Beschreibung
msoCalloutOne	eine horizontale oder vertikale Legendenlinie mit einem Abschnitt
msoCalloutTwo	eine frei drehbare Legendenlinie mit einem Abschnitt
msoCalloutMixed	Mischung aus den ersten beiden Konstanten
msoCalloutThree	eine Linie mit zwei Abschnitten
msoCalloutFour	eine Linie mit drei Abschnitten

Tabelle 2.8 Die Konstanten für das Argument »Type«

Beispiel

Im Beispiel aus Listing 2.31 wird eine Legende in *Tabelle3* eingefügt und ein Text darin erfasst.

```
Sub LegendeEinfügen()
 Dim Tabelle As Worksheet
 Dim shp As Shape

 Set Tabelle = Worksheets("Tabelle3")
 Set shp = _
 Tabelle.Shapes.AddCallout(Type:=msoCalloutTwo, _
  Left:=50, Top:=50, Width:=200, Height:=100)
  shp.TextFrame.Characters.Text = "TEXT"
End Sub
```

Listing 2.31 Legende einfügen und beschriften

Verwandte Methoden

Verwandte Methoden sind: AddCurve, AddConnector, AddDiagram, AddFormControl, AddLabel, AddLine, AddOLEObject, AddPicture, AddPolyLine, AddShape, AddTextbox.

2.9.2 »AddCurve«-Methode

Über die Methode AddCurve fügen Sie eine Bézierkurve in eine Tabelle ein.

Syntax

```
Ausdruck.AddCurve(SafeArrayOfPoints)
```


Argument	Beschreibung
Ausdruck	Dieses Argument gibt ein Shape-Objekt zurück.
SafeArrayOfPoints	Erforderlich. Eine Matrix von Koordinatenpaaren (Koordinatenpaar: Ein Wertepaar, das die x- und y-Koordinaten eines Punktes darstellt und in einem zweidimensionalen Array gespeichert ist, der Koordinaten für viele Punkte enthalten kann), die Scheitelpunkte und Steuerpunkte der Kurve angibt.

Beispiel

Beim Beispiel aus Listing 2.32 wird auf Basis einiger Werte aus *Tabelle4* eine Bézierkurve gezeichnet.

```
Sub KurveZeichnenLassen()
 Dim Tabelle As Worksheet
 Dim shp As Shape
 Dim pts(1 To 7, 1 To 2) As Single

 With Sheets("Tabelle4")
  pts(1, 1) = .Cells(1, 1).Value
  pts(1, 2) = .Cells(1, 2).Value
  pts(2, 1) = .Cells(2, 1).Value
  pts(2, 2) = .Cells(2, 2).Value
  pts(3, 1) = .Cells(3, 1).Value
  pts(3, 2) = .Cells(3, 2).Value
  pts(4, 1) = .Cells(4, 1).Value
  pts(4, 2) = .Cells(4, 2).Value
  pts(5, 1) = .Cells(5, 1).Value
  pts(5, 2) = .Cells(5, 2).Value
  pts(6, 1) = .Cells(6, 1).Value
  pts(6, 2) = .Cells(6, 2).Value
  pts(7, 1) = .Cells(7, 1).Value
  pts(7, 2) = .Cells(7, 2).Value
 End With

 Set Tabelle = Worksheets("Tabelle4")
 Set shp = _
 Tabelle.Shapes.AddCurve(SafeArrayOfPoints:=pts)
 shp.Fill.ForeColor.RGB = RGB(128, 0, 0)
End Sub
```

Listing 2.32 Bézierkurve einfügen und färben

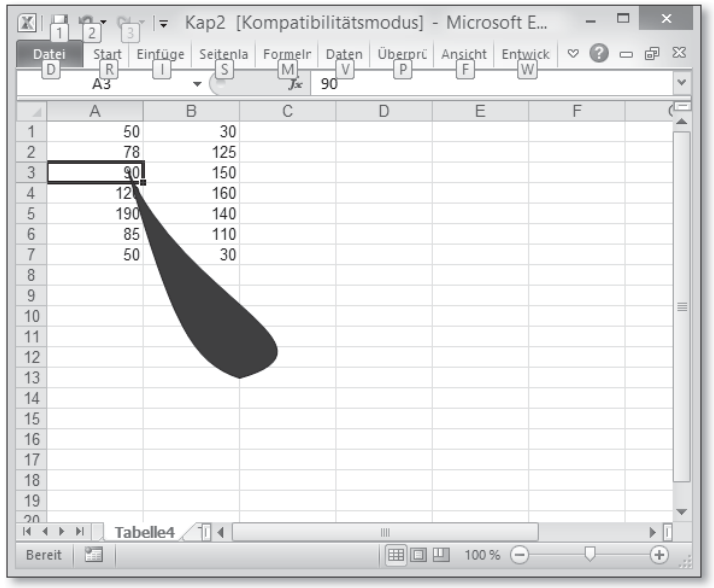

Abbildung 2.9 Eine Bézierkurve einfügen und füllen

Verwandte Methoden

Verwandte Methoden sind: `AddCallout`, `AddConnector`, `AddDiagram`, `AddFormControl`, `AddLabel`, `AddLine`, `AddOLEObject`, `AddPicture`, `AddPolyLine`, `AddShape`, `AddTextbox`.

2.9.3 »Fill«-Eigenschaft

Über die Eigenschaft `Fill` legen Sie die Füllung eines `Shape`-Objektes oder auch eines Diagramms fest. Diese Eigenschaft gibt ein `FillFormat`-Objekt zurück, das Füllformateigenschaften des angegebenen Diagramms oder der angegebenen Form darstellt.

2.9.4 »ForeColor«-Eigenschaft

Über die Eigenschaft `ForeColor` weisen Sie die Füllfarbe des Vordergrunds oder die Farbfläche eines `Shape`-Objektes oder eines Diagramm zu.

Beispiel

Im Beispiel aus Listing 2.33 wird die Füllfarbe des `Shape`-Objekts in *Tabelle3* festgelegt.

```
Sub ShpFormatieren()
 Dim shp As Shape

 Set shp = Worksheets("Tabelle3").Shapes(1)
 With shp
```



```
  .TextFrame.Characters.Text = "TEXT Neu"
  .Fill.ForeColor.RGB = RGB(256, 0, 0)
 End With
End Sub
```

Listing 2.33 Ein »Shape«-Objekt füllen

2.9.5 »RGB«-Funktion

Über die Funktion `RGB` erzeugen Sie einen Farbwert.

Syntax

```
RGB(red, green, blue)
```

Teil	Beschreibung
`red`	Erforderlich. Zahl im Bereich von 0 bis 255 (einschließlich), die die Rot-Komponente der Farbe darstellt.
`green`	Erforderlich. Zahl im Bereich 0 bis 255 (einschließlich), die die Grün-Komponente der Farbe darstellt.
`blue`	Erforderlich. Zahl im Bereich von 0 bis 255 (einschließlich), die die Blau-Komponente der Farbe darstellt.

Die Tabelle 2.9 enthält die wichtigsten Standardfarben.

Farbe	Rot- Komponente	Grün-Komponente	Blau-Komponente
Schwarz	0	0	0
Blau	0	0	255
Grün	0	255	0
Cyan	0	255	255
Rot	255	0	0
Magenta	255	0	255
Gelb	255	255	0
Weiß	255	255	255

Tabelle 2.9 Die wichtigsten Standardfarben

2.9.6 »AddTextbox«-Methode

Über die Methode `AddTextbox` erstellen Sie ein Textfeld. Dabei gibt diese Methode ein `Shape`-Objekt zurück, das das neue Textfeld darstellt.

Syntax

```
Ausdruck.AddTextbox(Orientation, Left, Top, Width, Height)
```

Argument	Beschreibung
`Ausdruck`	Dieses Argument gibt ein `Shape`-Objekt zurück.
`Orientation`	Erforderlich. Die Ausrichtung des Textfeldes
`MsoTextOrientation`	Stellt eine der Konstanten aus Tabelle 2.10 dar.
`Left`	Erforderlich. Die Position (in Punkt) der oberen linken Ecke des Textfeldes, relativ zur oberen linken Ecke des Dokuments
`Top`	Erforderlich. Die Position (in Punkt) der oberen linken Ecke des Textfeldes, relativ zum oberen Rand des Dokuments
`Width`	Erforderlich. Die Breite des Textfeldes in Punkt
`Height`	Erforderlich. Die Höhe des Textfeldes in Punkt

Für das Argument `MsoTextOrientation` stehen folgende Konstanten zur Verfügung:

Konstante	Beschreibung
`msoTextOrientationDownward`	Text wird vertikal, von oben nach unten angeordnet. Lesbar, wenn der Kopf nach rechts geneigt wird
`msoTextOrientationHorizontal`	Text wird horizontal von links nach rechts angeordnet.
`msoTextOrientationUpward`	Text wird vertikal, von oben nach unten angeordnet. Lesbar, wenn der Kopf nach links geneigt wird
`msoTextOrientationVertical`	Text wird vertikal, von oben nach unten angeordnet. Dabei wird Zeichen für Zeichen untereinander angeordnet; ist das Ende der Textbox erreicht, wird in der nächsten Spalte wieder oben angefangen.

Tabelle 2.10 Die möglichen Konstanten für die Textausrichtung

Beispiele

Im Beispiel aus Listing 2.34 wird in *Tabelle7* eine Textbox eingefügt. Der Inhalt dieser Textbox wird aus Zelle A1 bezogen.

```
Sub TextboxEinfügen()
 Dim Tabelle As Worksheet
 Dim shp As Shape

 Set Tabelle = Worksheets("Tabelle7")
 Set shp = Tabelle.Shapes.AddTextbox _
 (msoTextOrientationHorizontal, 50, 50, 100, 50)
 shp.TextFrame.Characters.Text = _
 Tabelle.Range("A1").Value
End Sub
```

Listing 2.34 Textbox über die Methode »AddTextbox« einfügen

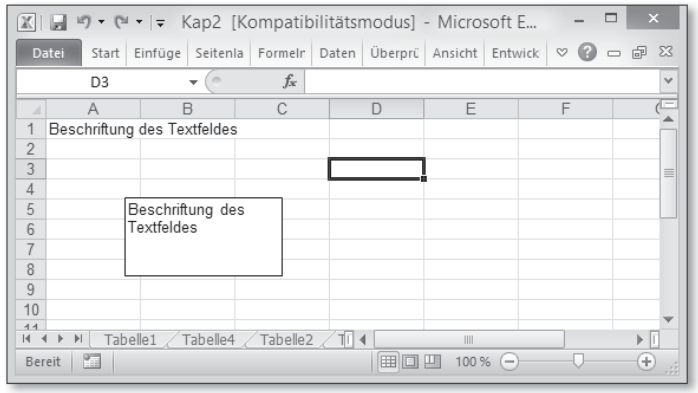

Abbildung 2.10 Textbox einfügen und mit Inhalt füllen

Soll die Ausrichtung des Textes geändert werden, dann starten Sie das Makro aus Listing 2.35.

```
Sub TextboxInhaltAusrichten()
 Dim Tabelle As Worksheet
 Dim shp As Shape

 Set Tabelle = Worksheets("Tabelle7")
 Set shp = Tabelle.Shapes.AddTextbox _
 (msoTextOrientationVertical, 50, 50, 100, 50)
 shp.TextFrame.Characters.Text = _
 Tabelle.Range("A1").Value
End Sub
```

Listing 2.35 Textbox-Inhalt ausrichten

Verwandte Methoden

Verwandte Methoden sind: `AddCallout`, `AddCurve`, `AddConnector`, `AddFormControl`, `AddLabel`, `AddLine`, `AddOLEObject`, `AddPicture`, `AddPolyLine`, `AddShape`.

2.9.7 »AddShape«-Methode

Über die Methode `AddShape` fügen Sie eine AutoForm aus der Symbolleiste ZEICHNEN ein.

Syntax

```
Ausdruck.AddShape(Type, Left, Top, Width, Height)
```

Argument	Beschreibung
`Ausdruck`	Erforderlich. Ein Ausdruck, der ein `Shape`-Objekt zurückgibt
`Type`	Erforderlich. Gibt den Typ der zu erstellenden AutoForm an. Exemplarisch werden einige davon in Tabelle 2.11 dargestellt.
`Left`, `Top`	Erforderlich. Die Position (in Punkt) der oberen linken Ecke des Begrenzungsrechtecks der AutoForm, relativ zur oberen linken Ecke des Dokuments
`Width`, `Height`	Erforderlich. Höhe und Breite des Begrenzungsrechtecks der AutoForm in Punkt

Ein Auszug aus den möglichen AutoForms sehen Sie in Tabelle 2.11.

Konstante	Beschreibung
`msoShapeActionButtonEnd`	ein Stop-Symbol
`msoShapeActionButtonBeginning`	ein Play-Symbol
`msoShapeActionButtonHelp`	ein Hilfe-Symbol
`msoShapeActionButtonHome`	ein Haus-Symbol
`msoShapeActionButtonInformation`	ein Informations-Symbol
`msoShapeActionButtonSound`	ein Sound-Symbol
`msoShapeBalloon`	eine Sprechblase

Tabelle 2.11 Mögliche AutoForms

Konstante	Beschreibung
`msoShapeBentArrow`	ein gebogener Pfeil
`msoShapeCan`	eine Dose
`msoShapeCross`	ein Kreuz
`msoShapeCube`	ein Würfel
`msoShapeDonut`	ein Donut
`msoShapeExplosion1`	eine Explosion
`msoShapeHeart`	ein Herz
`msoShapeLightningBolt`	ein Blitz
`msoShapeLineCallout1`	ein Kommentar
`msoShapeMoon`	ein Mond
`msoShapeSmileyFace`	ein lächelndes Gesicht
`msoShapeSun`	eine Sonne
`msoShapeWave`	eine Welle

Tabelle 2.11 Mögliche AutoForms (Forts.)

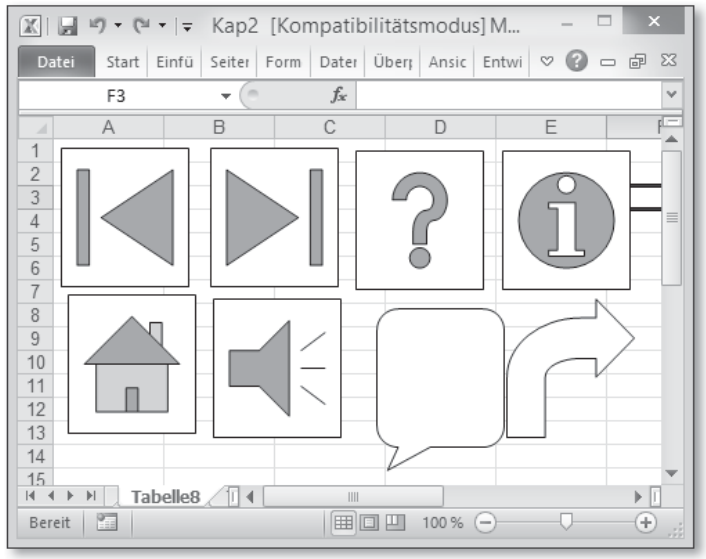

Abbildung 2.11 Typische AutoForms in Excel

Beispiele

Das Beispiel aus Listing 2.36 fügt ein lächelndes Gesicht in *Tabelle8* ein.

```
Sub SymbolEinfügen()
 Dim Tabelle As Worksheet
 Dim shp As Shape

 Set Tabelle = Worksheets("Tabelle8")
 Set shp = Tabelle.Shapes.AddShape _
 (msoShapeSmileyFace, 10, 10, 69, 75)
End Sub
```

Listing 2.36 AutoForm einfügen über die Methode »AddShape«

Die einzelnen `Shape`-Objekte können Sie auch über einen Index ansprechen. Jede AutoForm hat in Excel eine eindeutige Nummer. Im Beispiel aus Listing 2.37 werden in *Tabelle9* die ersten 100 AutoForms eingefügt.

```
Sub SymboleEinfügen()
 Dim Tabelle As Worksheet
 Dim shp As Shape
 Dim i As Integer
 Dim e As Integer

 i = 12
 Set Tabelle = Worksheets("Tabelle9")
 For e = 1 To 100
  Set shp = Tabelle.Shapes.AddShape _
  (e, i, 10, 10, 10)
  i = i + 12
 Next e
End Sub
```

Listing 2.37 Die ersten 100 AutoForms werden in eine Tabelle eingefügt.

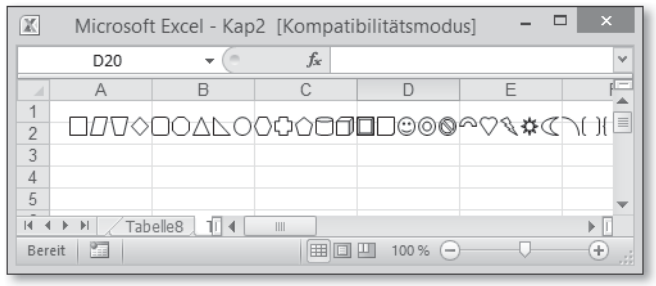

Abbildung 2.12 Von links nach rechts (Nr. 1 bis Nr. 100)

Verwandte Methoden

Verwandte Methoden sind: `AddCallout`, `AddCurve`, `AddConnector`, `AddFormControl`, `AddLabel`, `AddLine`, `AddOLEObject`, `AddPicture`, `AddPolyLine`.

2.10 Die Objektvariable »Workbook«

Über die Objektvariable `Workbook` sprechen Sie Arbeitsmappen an.

> **Hinweis**
>
> Weitere Methoden, Eigenschaften zum Objekt `Workbook` finden Sie in Kapitel 7, »›Workbook‹-Objekt«.

2.10.1 »Save«-Methode

Die Methode `Save` speichert Änderungen in der angegebenen Arbeitsmappe.

Syntax

```
Ausdruck.Save
```

Das erforderliche Argument `Ausdruck` ist ein Ausdruck, der ein `Workbook`-Objekt darstellt.

Beispiel

Im Beispiel aus Listing 2.38 wird die aktuell geöffnete und aktive Arbeitsmappe gespeichert.

```
Sub ArbeitsmappeSpeichern()
 Dim Mappe As Workbook

 Set Mappe = ActiveWorkbook
 Mappe.Save
End Sub
```

Listing 2.38 Arbeitsmappe speichern über die Methode »Save«

Verwandte Methoden

Verwandte Methoden sind: `SaveAs`, `SaveCopyAs`.

2.10.2 »Close«-Methode

Die Methode `Close` schließt eine Arbeitsmappe.

Syntax

```
Ausdruck.Close(SaveChanges, Filename, RouteWorkbook)
```

Argument	Beschreibung
`Ausdruck`	Das erforderliche Argument `Ausdruck` ist ein Ausdruck, der ein `Workbook`-Objekt darstellt.
`SaveChanges`	Optional. Hiermit entscheiden sie, ob Sie die Mappe beim Schließen gleichzeitig speichern möchten oder nicht. Wenn ja, dann weisen Sie `True` zu. Wenn nicht, dann weisen Sie den Wert `False` zu.
`FileName`	Optional. Gibt den Dateinamen an, unter dem die Änderungen gespeichert werden sollen. Fehlt dieser, werden Änderungen selbstverständlich in der zu schließenden Mappe durchgeführt.
`RouteWorkbook`	Optional. Wenn die Arbeitsmappe nicht an den nächsten Empfänger weiterzuleiten ist (weil sie keinen Verteiler hat oder schon weitergeleitet wurde), wird dieses Argument ignoriert.

Beispiel

Im Beispiel aus Listing 2.39 wird die aktive Arbeitsmappe geschlossen.

```
Sub ArbeitsmappeSchließen()
 Dim Mappe As Workbook

 Set Mappe = ActiveWorkbook
 Mappe.Close savechanges:=True
End Sub
```

Listing 2.39 Mit der Methode »Close« die Mappe schließen

2.10.3 »Name«-Eigenschaft

Über die Eigenschaft `Name` ermitteln Sie den Namen einer Arbeitsmappe.

Syntax

```
Ausdruck.Name
```


Das erforderliche Argument `Ausdruck` ist ein Ausdruck, der ein `Workbook`-Objekt darstellt.

Beispiel

Im Beispiel aus Listing 2.40 wird der Name der aktiven Arbeitsmappe am Bildschirm ausgegeben.

```
Sub ArbeitsmappenNamen()
 Dim Mappe As Workbook

 Set Mappe = ActiveWorkbook
 MsgBox Mappe.Name
End Sub
```

Listing 2.40 Den Namen der Mappe über die Eigenschaft »Name« abfragen

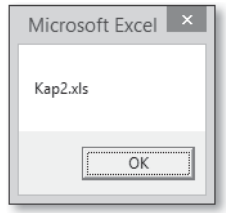

Abbildung 2.13 Den Namen einer Mappe ermitteln

Verwandte Eigenschaften

Verwandte Eigenschaften sind: `FullName`.

2.10.4 »FullName«-Eigenschaft

Über die Eigenschaft `FullName` ermitteln Sie den Namen inklusive des Speicherpfades einer Arbeitsmappe.

Syntax

```
Ausdruck.FullName
```

Das erforderliche Argument `Ausdruck` ist ein Ausdruck, der ein `Workbook`-Objekt darstellt.

Beispiel

Im Beispiel aus Listing 2.41 wird der komplette Speicherpfad der aktuell geöffneten Arbeitsmappe ermittelt und am Bildschirm angezeigt.

```
Sub ArbeitsmappenPfad()
 Dim Mappe As Workbook

 Set Mappe = ActiveWorkbook
 MsgBox Mappe.FullName
End Sub
```

Listing 2.41 Den Pfadnamen der Mappe über die Eigenschaft »FullName« ermitteln

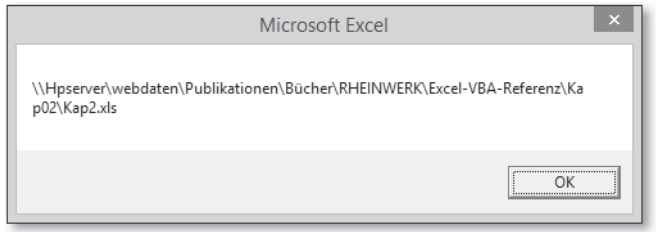

Abbildung 2.14 Der komplette Pfad der Mappe wird ermittelt.

Verwandte Eigenschaften

Verwandte Eigenschaften sind: `Name`.

2.11 Die Objektvariable »RecentFile«

Mit der Objektvariable `RecentFile` greifen Sie auf die Liste der zuletzt verwendeten Dateien zu.

2.11.1 »Name«-Eigenschaft

Über die Eigenschaft `Name` fragen Sie die Liste der zuletzt geöffneten Arbeitsmappen ab.

Syntax

```
Ausdruck.Name
```

Das erforderliche Argument `Ausdruck` ist ein Ausdruck, der ein `RecentFile`-Objekt darstellt.

Beispiele

Das Beispiel aus Listing 2.42 gibt den Namen der zuletzt geöffneten Arbeitsmappe aus.

```
Sub ZuletztGeöffneteMappe()
 Dim Rf As RecentFile

 Set Rf = Application.RecentFiles(1)
 MsgBox Rf.Name
End Sub
```

Listing 2.42 Zuletzt geöffnete Mappe ermitteln

Um eine komplette Liste der zuletzt geöffneten Arbeitsmappen zu erstellen, starten Sie das Makro aus Listing 2.43.

```
Sub LetzteDateienAnzeigen()
 Dim Rf As RecentFile
 Dim i As Integer

 For i = 1 To Application.RecentFiles.Count
   Debug.Print Application.RecentFiles(i).Name
 Next i
End Sub
```

Listing 2.43 Liste der zuletzt geöffneten Mappen erzeugen

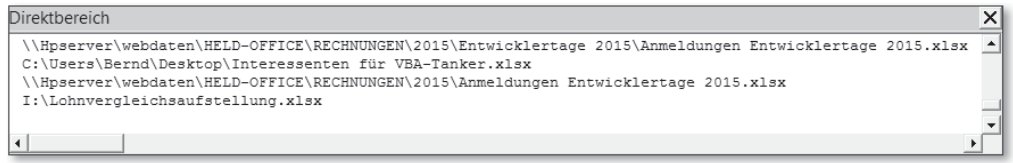

Abbildung 2.15 Diese Mappen waren zuletzt geöffnet.

2.12 Die Objektvariable »AddIn«

Über die Objektvariable `AddIn` sprechen Sie ein einzelnes installiertes oder nicht installiertes Add-In an.

2.12.1 »Installed«-Eigenschaft

Mit der Eigenschaft `Installed` binden Sie ein Add-In ein oder deaktivieren es wieder.

Syntax

```
Ausdruck.Installed
```

Das erforderliche Argument `Ausdruck` ist ein Ausdruck, der ein `AddIn`-Objekt darstellt.

Beispiele

Im Beispiel aus Listing 2.44 wird das Add-In Analyse-Funktionen aktiviert.

```
Sub AddInAktivieren()
 Dim AI As AddIn

 Set AI = AddIns("Analyse-Funktionen")
 AI.Installed = True
End Sub
```

Listing 2.44 Über die Eigenschaft »Installed« ein Add-In aktivieren

Um das Add-In aus dem Add-Ins-Manager wieder zu entladen, starten Sie das Makro aus Listing 2.45:

```
Sub AddInDeAktivieren()
 Dim AI As AddIn

 Set AI = AddIns("Analyse-Funktionen")
 AI.Installed = False
End Sub
```

Listing 2.45 Ein Add-In wieder deaktivieren

Das Makro aus Listing 2.46 schreibt die Namen aller Add-Ins, die derzeit im Add-Ins-Manager eingebunden sind, in das Direktfenster der Entwicklungsumgebung.

```
Sub AddInListe()
 Dim AI As AddIn

 For Each AI In Application.AddIns
  If AI.Installed = True Then
   Debug.Print AI.FullName
  End If
 Next AI
End Sub
```

Listing 2.46 Liste der geladenen Add-Ins

2.13 Die Objektvariable »Button«

Über die Objektvariable `Button` sprechen Sie eine Schaltfläche in einer Tabelle an.

2.13.1 »Add«-Methode

Über die Methode `Add` fügen Sie eine Schaltfläche in eine Tabelle ein.

Syntax

```
Ausdruck.Buttons.Add(Left, Top, Width, Height)
```

Argument	Beschreibung
`Ausdruck`	Das erforderliche Argument `Ausdruck` ist ein Ausdruck, der ein `Button`-Objekt darstellt.
`Left`	Legt die linke Ecke der Schaltfläche in Pixeln fest.
`Top`	Legt die obere linke Ecke der Schaltfläche in Pixeln fest.
`Width`	Legt die Breite der Schaltfläche in Pixeln fest.
`Height`	Legt die Höhe der Schaltfläche in Pixeln fest.

2.13.2 »Caption«-Eigenschaft

Über die Eigenschaft `Caption` legen Sie die Beschriftung einer Schaltfläche fest.

Syntax

```
Ausdruck.Caption = "Text"
```

Das erforderliche Argument `Ausdruck` ist ein Ausdruck, der ein `Button`-Objekt darstellt.

2.13.3 »OnAction«-Eigenschaft

Über die Eigenschaft `OnAction` geben Sie den Namen des Makros an, das ausgeführt wird, wenn auf die Schaltfläche geklickt wird.

Syntax

```
Ausdruck.OnAction = "Makroname"
```

Das erforderliche Argument `Ausdruck` ist ein Ausdruck, der ein `Button`-Objekt darstellt.

Beispiele

Beim Beispiel aus Listing 2.47 wird eine Schaltfläche in *Tabelle10* eingefügt und das Makro `ExcelBeenden` hinterlegt.

```
Sub SchaltflächeIntegrieren()
 Dim Schaltfläche As Button

 Set Schaltfläche = _
 Sheets("Tabelle10").Buttons.Add(10, 20, 100, 50)

 With Schaltfläche
   .Caption = "Anwendung beenden"
   .OnAction = "ExcelBeenden"
 End With
 Set Schaltfläche = Nothing
End Sub
```

Listing 2.47 Schaltfläche in Tabelle integrieren

Zur Vollständigkeit noch das Makro aus Listing 2.48, das Excel ohne Rückfrage beendet.

```
Sub ExcelBeenden()
 Application.DisplayAlerts = False
 Application.Quit
End Sub
```

Listing 2.48 Excel ohne Rückfrage beenden

Kapitel 6
»Application«-Objekt

Das Objekt »Application« ist das oberste Objekt in der Hierarchie der Excel-Objekte. Über dieses Objekt lassen sich Excel-Einstellungen anpassen.

In Tabelle 6.1 sehen Sie das Objekt `Application`, das die höchste Einordnung im Objektmodell von Excel einnimmt.

Objekt	Verwendung
`Workbook`	komplette Arbeitsmappe
`AddIn`	ein Add-In
`Answer`	Antwort-Assistent
`AutoCorrect`	Autokorrektur
`Assistant`	Office-Assistent
`AutoRecover`	automatische Wiederherstellungsfunktion einer Arbeitsmappe
`CellFormat`	Suchkriterien für ein Zellenformat
`COMAddIn`	`COMAddIn`-Objekt
`Debug`	Testumgebung
`Dialog`	integriertes Dialogfeld
`CommandBar`	Menü- oder Symbolleisten
`ErrorCheckingOptions`	Optionen zur Fehlerprüfung
`LanguageSettings`	Spracheinstellungen
`Name`	Name der Applikation

Tabelle 6.1 Die untergeordneten Objekte von »Application«

Objekt	Verwendung
Window	Fenster der Anwendung
WorksheetFunction	Tabellenfunktion
RecentFile	Liste der zuletzt verwendeten Dateien
SmartTagRecognizers	Erkennung von SmartTags
Speech	Spracheigenschaften
SpellingOptions	Rechtschreibprüfung
FileSearch	standardisierte Dateisuche
VBE	Zugriff auf Entwicklungsumgebung
ODBCError	Fehlerbeschreibung bei ODBC-Zugriff
OLEDBError	Fehlerbeschreibung bei OLEDB-Zugriff
DefaultWebOptions	webbasierte Einstellungen
UsedObjects	benutzte Objekte einer Mappe
Watch	Überwachungsfenster (ab Excel 2003)

Tabelle 6.1 Die untergeordneten Objekte von »Application« (Forts.)

6.1 »Applications«-Eigenschaften

6.1.1 »ActiveCell«-Eigenschaft

Die Eigenschaft `ActiveCell` gibt die aktive Zelle zurück.

Beispiel

Das Beispiel aus Listing 6.1 gibt die aktuell markierte Adresse sowie den Inhalt dieser Zelle am Bildschirm aus.

```
Sub Activecell_Beispiel()
 MsgBox ActiveCell.Address
 MsgBox ActiveCell.Value
End Sub
```

Listing 6.1 Die aktive Zelle ermitteln und auslesen

6.1.2 »ActiveChart«-Eigenschaft

Die Eigenschaft `ActiveChart` stellt das aktive Diagramm (entweder ein eingebettetes Diagramm oder ein Diagrammblatt) dar.

Beispiel

Das Beispiel aus Listing 6.2 gibt den Namen sowie den Speicherort des markierten Diagrammobjekts zurück.


```
Sub ActiveChart_Beispiel()
 MsgBox ActiveChart.Name
End Sub
```

Listing 6.2 Namen des Diagramms ermitteln

6.1.3 »ActivePrinter«-Eigenschaft

Über die Eigenschaft `ActivePrinter` ermitteln können Sie den Namen des aktuell eingestellten Druckers.

Beispiel

```
Sub ActivePrinter_Beispiel()
 MsgBox Application.ActivePrinter
End Sub
```

Listing 6.3 Den Namen des eingestellten Standarddruckers ausgeben

6.1.4 »ActiveSheet«-Eigenschaft

Die Eigenschaft `ActiveSheet` gibt den Namen der aktiven Tabelle bzw. das aktive Diagrammblatt zurück oder legt ihn fest.

Beispiel

Im Beispiel aus Listing 6.4 wird der Namen der aktiven Tabelle festgelegt.

```
Sub ActiveSheet_Beispiel()
 ActiveSheet.Name = "Tabelle1"
End Sub
```

Listing 6.4 Die aktive Tabelle umbenennen

6.1.5 »ActiveWindow«-Eigenschaft

Über die Eigenschaft `ActiveWindow` sprechen Sie das aktive Fenster an.

Beispiel

Das Beispiel aus Listing 6.5 erstellt ein neues Fenster auf Basis des aktiven Fensters, also eine Kopie.

```
Sub ActiveWindow_Beispiel()
 ActiveWindow.NewWindow
End Sub
```

Listing 6.5 Ein neues Fenster erstellen

6.1.6 »ActiveWorkbook«-Eigenschaft

Über die Eigenschaft `ActiveWorkbook` sprechen Sie die aktive Arbeitsmappe an.

Beispiel

Das Beispiel aus Listing 6.6 gibt den Namen sowie den Speicherpfad der aktiven Arbeitsmappe aus.

```
Sub Activeworkbook_Beispiel()
 MsgBox ActiveWorkbook.FullName
End Sub
```

Listing 6.6 Namen und Speicherpfad der aktiven Mappe ermitteln

6.1.7 »AddIns«-Eigenschaft

Die Eigenschaft `AddIns` gibt eine Add-Ins-Auflistung zurück, die alle im Dialogfeld ADD-INS (Menü EXTRAS) aufgeführten Add-Ins darstellt.

Das Beispiel aus Listing 6.7 gibt alle derzeit aktivierten Add-Ins im Direktfenster der Entwicklungsumgebung aus.

```
Sub AddIns_Beispiel()
Dim Addi As AddIn

For Each Addi In Application.AddIns
 If Addi.Installed = True Then
  Debug.Print Addi.Name
 End If
Next Addi
End Sub
```

Listing 6.7 Alle installierten Add-Ins ermitteln

6.1.8 »AlertBeforeOverwriting«-Eigenschaft

Die Eigenschaft `AlertBeforeOverWriting` meldet den Wert `True`, wenn Microsoft Excel eine Meldung einblenden soll, bevor nicht leere Zellen während einer Drag-and-Drop-Operation überschrieben werden.

Beispiel

Das Beispiel aus Listing 6.8 schaltet diese Eigenschaft aus.

```
Sub AlertBeforeOverWriting_Beispiel()
 Application.AlertBeforeOverwriting = False
End Sub
```

Listing 6.8 Meldung beim Überschreiben von Zellen bei Drag & Drop ausschalten

6.1.9 »AltStartupPath«-, »StartUpPath«-Eigenschaft

Die Eigenschaft `AltStartupPath` gibt den Namen des alternativen Startordners zurück bzw. legt diesen fest. Alle Excel-Mappen, die in diesem Ordner gespeichert sind, werden beim Starten von Excel automatisch geöffnet.

Die Eigenschaft `StartUpPath` gibt den vollständigen Pfad des Startordners ohne das abschließende Trennzeichen zurück.

Beispiel

Beim Beispiel aus Listing 6.9 wird ein zusätzlicher Startordner festgelegt.

```
Sub AltStartupPath_Beispiel()
 Application.AltStartupPath = "C:\Wichtig"
End Sub
```

Listing 6.9 Einen Zusatzstartordner festlegen

6.1.10 »AnswerWizard«-Eigenschaft

Die Eigenschaft `AnswerWizard` gibt das `AnswerWizard`-Objekt für Microsoft Excel zurück.

Beispiel

Das Beispiel aus Listing 6.10 setzt die Dateiliste des Antwort-Assistenten zurück.

```
Sub AnswerWizard_Beispiel()
 Application.AnswerWizard.ResetFileList
End Sub
```

Listing 6.10 Die Dateiliste des Antwort-Assistenten zurücksetzen

6.1.11 »ArbitraryXMLSupportAvailable«-Eigenschaft

Diese Eigenschaft steht ab Excel 2003 zur Verfügung und prüft, ob die XML-Features in Microsoft Excel verfügbar sind.

Beispiel

Das Beispiel aus Listing 6.11 überprüft, ob die XML-Features zur Verfügung stehen.

```
Sub ArbitraryXMLSupportAvailable_Beispiel()
  If Application.ArbitraryXMLSupportAvailable Then
   MsgBox "XML-Features verfügbar!"
  Else
   MsgBox "XML-Features nicht verfügbar!"
  End If
End Sub
```

Listing 6.11 XML-Features überprüfen

6.1.12 »AskToUpdateLinks«-Eigenschaft

Die Eigenschaft `AskToUpdateLinks` meldet den Wert `True`, wenn Microsoft Excel den Benutzer fragen soll, ob beim Öffnen von Dateien mit Verknüpfungen diese Verknüpfungen aktualisiert werden sollen. Wenn Sie den Wert `False` einstellen, werden Verknüpfungen automatisch ohne Dialogfeld aktualisiert.

Beispiel

Das Beispiel aus Listing 6.12 legt fest, dass beim Öffnen von verknüpften Dateien die Warnmeldung angezeigt werden soll.

```
Sub AskToUpdateLinks_Beispiel()
 Application.AskToUpdateLinks = True
End Sub
```

Listing 6.12 Datei-Verknüpfungsabfrage beim Öffnen anzeigen

6.1.13 »Assistant«-Eigenschaft

Die Eigenschaft `Assistant` gibt ein `Assistant`-Objekt für Microsoft Excel zurück.

Beispiel

Das Beispiel aus Listing 6.13 ruft die Onlinehilfe von Excel auf.


```
Sub Assistant_Beispiel()
 Application.Assistant.Help
End Sub
```

Listing 6.13 Onlinehilfe aufrufen

6.1.14 »AutoCorrect«-Eigenschaft

Die Eigenschaft `AutoCorrect` gibt ein `AutoCorrect`-Objekt zurück, das die Autokorrektur-Attribute darstellt.

Beispiel

Das Beispiel aus Listing 6.14 legt einen zusätzlichen Autokorrektureintrag an.

```
Sub AutoCorrect_Beispiel()
 With Application.AutoCorrect
    .AddReplacement "Sei", "Sie"
 End With
End Sub
```

Listing 6.14 Einen Autokorrektureintrag einfügen

6.1.15 »AutoFormatAsYouTypeReplaceHyperlinks«-Eigenschaft

Setzen Sie die Eigenschaft `AutoFormatAsYouTypeReplaceHyperlinks` auf den Wert `True`, wenn Microsoft Excel Hyperlinks automatisch formatieren soll, während Sie diese eingeben. Wenn Sie dieser Eigenschaft den Wert `False` zuweisen, unterbleibt die automatische Formatierung.

Beispiel

Nach Aufruf des Beispiels aus Listing 6.15 wird die automatische Formatierung von Internetseiten und E-Mail-Adressen zukünftig unterbunden.

```
Sub AutoFormatAsYouTypeReplaceHyperlinks_Beispiel()
 Application.AutoFormatAsYouTypeReplaceHyperlinks = False
End Sub
```

Listing 6.15 Hyperlinkkonvertierung verhindern

6.1.16 »AutomationSecurity«-Eigenschaft

Die Eigenschaft `AutomationSecurity` gibt eine `MsoAutomationSecurity`-Konstante zurück oder legt eine solche Konstante fest. Diese steht für den Sicherheitsmodus, den Microsoft Excel beim programmatischen Öffnen von Dateien verwendet.

Als Konstanten stehen folgende zur Verfügung:

Konstante	Beschreibung
msoAutomationSecurityByUI	Verwendet die Sicherheitseinstellung, die im Dialogfeld SICHERHEIT angegeben wurde.
msoAutomationSecurityForceDisable	Deaktiviert alle Makros in allen programmgesteuert geöffneten Dateien, ohne Sicherheitswarnungen anzuzeigen.
msoAutomationSecurityLow	Aktiviert alle Makros. Dies ist der Standardwert beim Starten einer Anwendung.

Tabelle 6.2 Die Sicherheitskonstanten der Eigenschaft »AutomationSecurity«

Beispiel

Das Beispiel aus Listing 6.16 öffnet eine Mappe, die Makros enthält, ohne dass die Makros aktiviert werden.

```
Sub AutomationSecurity_Beispiel()
 Application.AutomationSecurity =  _
 msoAutomationSecurityForceDisable
 Workbooks.Open("C:\Eigene Dateien\Mappe1.xls")
End Sub
```

Listing 6.16 Arbeitsmappe öffnen mit deaktivierten Makros

6.1.17 »AutoRecover«-Eigenschaft

Die Eigenschaft `AutoRecover` gibt ein `AutoRecover`-Objekt zurück, das in festgelegten Intervallen alle Dateiformate sichert.

Beispiel

Das Beispiel aus Listing 6.17 bestimmt als `AutoRecover`-Einstellung alle 10 Minuten.

```
Sub AutoRecover_Beispiel()
 Application.AutoRecover.Time = 10
End Sub
```

Listing 6.17 Die »AutoRecover«-Einstellungen vornehmen

> **Hinweis**
>
> Gültige Zeitintervalle liegen zwischen 1 und 120 Minuten.

6.1.18 »CalculateBeforeSave«-, »Calculation«-Eigenschaft

Setzen Sie die Eigenschaft `CalculateBeforeSave` auf den Wert `True`, um Arbeitsmappen vor dem Speichern neu zu berechnen.

Die Eigenschaft `Calculation` gibt die Berechnungsart zurück oder legt sie fest. Dazu stehen folgende Möglichkeiten zur Verfügung:

Konstante	Beschreibung
xlCalculationAutomatic	automatische Berechnung (Standardeinstellung)
xlCalculationManual	manuelle Berechnung
xlCalculationSemiautomatic	automatisch außer bei Mehrfachoperationen

Tabelle 6.3 Die Berechnungskonstanten der Eigenschaft »Calculation«

Beispiel

Beim folgenden Beispiel wurde bei einer etwas größeren Arbeitsmappe die Berechnungseigenschaft `Calculation` ausgeschaltet, um zügiger arbeiten zu können. Über die Eigenschaft `CalculateBeforeSave` wird die Mappe vor der Speicherung neu berechnet, also aktualisiert.

```
Sub CalculationBeforeSave_Beispiel()
  Application.Calculation = xlManual
  Application.CalculateBeforeSave = True
End Sub
```

Listing 6.18 Mappe wird vor dem Speichern neu berechnet.

6.1.19 »Caller«-Eigenschaft

Die Eigenschaft `Caller` gibt Informationen darüber zurück, wie Visual Basic aufgerufen wurde.

Beispiel

Das Beispiel aus Listing 6.19 prüft mit Hilfe der Eigenschaft `Caller`, welche Schaltfläche in einer Tabelle geklickt wurde. Dabei wird das Makro den beiden Schaltflächen zugewiesen.

```
Sub Caller_Bespiel()
 Select Case Application.Caller
    Case "Schaltfläche 2"
          MsgBox "Schaltfläche 2 geklickt!"
    Case "Schaltfläche 3"
```

```
        MsgBox "Schaltfläche 3 geklickt!"
 End Select
End Sub
```

Listing 6.19 Welche Schaltfläche wurde angeklickt?

6.1.20 »Caption«-Eigenschaft

Die Eigenschaft `Caption` legt den Namen in der Titelleiste des Microsoft-Excel-Hauptfensters fest bzw. gibt ihn zurück.

Beispiel

Das Beispiel aus Listing 6.20 passt den Namen des Titelfensters an.

```
Sub Caption_Beispiel()
  Application.Caption = "Referenz-Buch"
End Sub
```

Listing 6.20 Namen des Titelfensters anpassen

6.1.21 »CellDragAndDrop«-Eigenschaft

Die Eigenschaft `CellDragAndDrop` legt fest, ob Drag & Drop für Zellenbearbeitung zugelassen wird oder nicht.

Beispiel

Das Beispiel aus Listing 6.21 schaltet die Drag-and-Drop-Eigenschaft ab.

```
Sub CellDragAndDrop_Beispiel()
  Application.CellDragAndDrop = False
End Sub
```

Listing 6.21 Drag & Drop für Zellenbearbeitung abschalten

6.1.22 »CutCopyMode«-Eigenschaft

Die Eigenschaft `CutCopyMode` gibt den Status des Ausschneide- oder Kopiermodus zurück bzw. stellt diesen ein. Folgende Konstanten stehen zur Verfügung:

Rückgabewert	Beschreibung
`False`	Weder im Ausschneide- noch im Kopiermodus. Entfernt den Laufrahmen.

Tabelle 6.4 Die Konstanten der Eigenschaft »CutCopyMode«

Rückgabewert	Beschreibung
`xlCopy`	im Kopiermodus
`xlCut`	im Ausschneidemodus

Tabelle 6.4 Die Konstanten der Eigenschaft »CutCopyMode« (Forts.)

Beispiel

Im Beispiel in Listing 6.22 werden alle benutzten Zellen einer Tabelle kopiert und in einer anderen Tabelle eingefügt. Danach erfolgt die Deaktivierung des Laufrahmens, der nach dem Einfügen noch läuft.

```
Sub CutCopyMode_Beispiel()
 Tabelle2.UsedRange.Copy Destination:=Tabelle3.Range("A1")
 Application.CutCopyMode = False
End Sub
```

Listing 6.22 Laufrahmen nach dem Kopieren von Daten entfernen

6.1.23 »DecimalSeparator«-, »ThousandsSeparator«-, »UseSystemSeparators«-Eigenschaft

Die Eigenschaft `DecimalSeparator` gibt das als Dezimaltrennzeichen verwendete Zeichen als String-Wert zurück oder legt es fest.

Die Eigenschaft `ThousandsSeparator` gibt das als Tausendertrennzeichen verwendete Zeichen als String-Wert zurück oder legt es fest.

Über die Eigenschaft `UseSystemSeparators` bestimmen Sie, ob Sie die in der Systemsteuerung hinterlegten Einstellungen anwenden möchten.

Beispiel

Im Beispiel aus Listing 6.23 werden die Dezimaltrennzeichen sowie das Tausendertrennzeichen, wie es beispielsweise in der Schweiz üblich ist, eingestellt.

```
Sub DecimalSeparator_Beispiel()
 Application.DecimalSeparator = "."
 Application.ThousandsSeparator = ","
 Application.UseSystemSeparators = False
End Sub
```

Listing 6.23 Trennzeichen austauschen

6.1.24 »DefaultFilePath«-, »TemplatesPath«-Eigenschaft

Die Eigenschaft `DefaultFilePath` gibt den Standardpfad zurück, den Microsoft Excel beim Öffnen von Dateien verwendet, oder legt ihn fest.

Die Eigenschaft `TemplatesPath` gibt den lokalen Pfad zurück, unter dem Vorlagen gespeichert sind. Diese Eigenschaft kann nur abgefragt und nicht gesetzt werden.

Beispiel

Das Beispiel aus Listing 6.24 fragt den Standardpfad von Excel ab.

```
Sub DefaultFilePath_Beispiel()
 MsgBox Application.DefaultFilePath
End Sub
```

Listing 6.24 Den Standardpfad von Excel abfragen

6.1.25 »DefaultSaveFormat«-Eigenschaft

Die Eigenschaft `DefaultSaveFormat` gibt das Standardformat beim Speichern von Dateien zurück oder legt es fest. Wenn Sie beispielsweise standardmäßig Excel-Arbeitsmappen für eine ältere Excel-Version erstellen müssen, dann können Sie diese Eigenschaft einsetzen.

Beispiel

Das Beispiel aus Listing 6.25 stellt das Standardspeicherformat für Excel-Arbeitsmappen auf das neue Format ein.

```
Sub DefaultSaveFormat_Beispiel()
 Application.DefaultSaveFormat = xlOpenXMLWorkbook
End Sub
```

Listing 6.25 Das Standardspeicherformat in Excel einstellen

6.1.26 »DisplayAlerts«-Eigenschaft

Über die Eigenschaft `DisplayAlerts` schalten Sie Standardmeldungen von Excel temporär ein bzw. aus.

Beispiel

Das Beispiel in Listing 6.26 löscht die angegebene Tabelle. Damit keine Rückfrage erfolgt, wird die Standardmeldung für diesen Vorgang ausgeschaltet.

```
Sub DisplayAlerts_Beispiel()
 DisplayAlerts = False
 Sheets("Tabelle3").Delete
 DisplayAlerts = True
End Sub
```

Listing 6.26 Standardmeldungen in Excel unterdrücken

6.1.27 »DisplayClipboardWindow«-Eigenschaft

Über die Eigenschaft `DisplayClipboardWindow` können Sie die Zwischenablage von Office anzeigen, indem Sie diese Eigenschaft auf den Wert `True` setzen.

6.1.28 »DisplayCommentIndicator«-, »DisplayNoteIndicator«-Eigenschaft

Über die Eigenschaft `DisplayCommentIndicator` legen Sie fest, wie Zellenkommentare und Indikatoren (rotes Dreieck) in Excel angezeigt werden. Folgende Konstanten stehen dabei zur Verfügung:

Konstante	Beschreibung
`xlNoIndicator`	Es wird kein Indikator angezeigt.
`xlCommentIndicatorOnly`	Es wird nur der Indikator angezeigt.
`xlCommentAndIndicator`	Es werden der Indikator sowie das Kommentarfester angezeigt.

Tabelle 6.5 Die Konstanten der Eigenschaft »DisplayCommentIndicator«

Über die Eigenschaft `DisplayNoteIndicator` legen Sie fest, ob Indikatoren (rotes Dreieck) in Excel angezeigt werden sollen.

Beispiel

Beim Beispiel aus Listing 6.27 machen Sie es von einem bestimmten Zellinhalt abhängig, ob Kommentarfester und Indikator angezeigt werden oder nicht.

```
Sub DisplayCommentIndicator_Beispiel()
  If Sheets("Tabelle1").Range("A1").Value = 1 Then
   Application.DisplayCommentIndicator = xlNoIndicator
  Else
   Application.DisplayCommentIndicator = xlCommentAndIndicator
  End If
End Sub
```

Listing 6.27 Kommentarfenster und Indikator anzeigen

6.1.29 »DisplayFormulaBar«-, »DisplayStatusBar«-Eigenschaft

Über die Eigenschaft `DisplayFormulaBar` blenden Sie die Bearbeitungsleiste von Excel ein (`True`) oder aus (`False`).

Über die Eigenschaft `DisplayStatusBar` blenden Sie die Statusleiste von Excel ein (`True`) oder aus (`False`).

6.1.30 »DisplayFullScreen«-Eigenschaft

Über die Eigenschaft `DisplayFullScreen` zeigen Sie in Excel im Vollbildmodus an (`True`) oder stellen die Normalansicht ein (`False`).

6.1.31 »DisplayFunctionToolTips«-Eigenschaft

Über die Eigenschaft `DisplayFunctionToolTips` zeigen Sie QuickInfos für Funktionen an (`True`) oder blenden sie aus (`False`).

6.1.32 »DisplayRecentFiles«-, »RecentFiles«-Eigenschaft

Über die Eigenschaft `DisplayRecentFiles` entscheiden Sie, ob Sie die Wiedervorlageliste im Menü DATEI unten anzeigen (`True`) oder ausblenden (`False`) möchten.

Die Eigenschaft `RecentFiles` gibt eine Liste mit den zuletzt geöffneten Dateien zurück.

Beispiel

Das Beispiel aus Listing 6.28 gibt die Namen der zuletzt geöffneten Arbeitsmappen im Direktfenster der Entwicklungsumgebung aus.

```
Sub RecentFiles_Beispiel()
 Dim Rf As RecentFile

 For Each Rf In Application.RecentFiles
   Debug.Print Rf.Name
 Next Rf
End Sub
```

Listing 6.28 Die zuletzt geöffneten Dateien ermitteln

6.1.33 »DisplayScrollBars«-Eigenschaft

Die Eigenschaft `DisplayScrollBars` legt fest, ob die horizontalen und vertikalen Bildlaufleisten angezeigt (`True`) oder ausgeblendet (`False`) werden sollen.

Hinweis

Sollen die beiden Leistenarten (horizontal und vertikal) einzeln eingeblendet bzw. ausgeblendet werden können, dann stehen hierfür die Eigenschaften `DisplayHorizontalScrollBar` und `DisplayVerticalScrollBar` zur Verfügung.

6.1.34 »EditDirectlyInCell«-Eigenschaft

Über die Eigenschaft `EditDirectlyInCell` können Sie festlegen, ob in Microsoft Excel eine direkte Zellbearbeitung zugelassen werden soll oder nicht. Wenn Sie diese Eigenschaft auf den Wert `False` setzen, wird die direkte Zellbearbeitung deaktiviert, d. h., es ist danach nicht mehr möglich, über einen Doppelklick direkt in die Zelle zu springen.

6.1.35 »EnableAnimations«-Eigenschaft

Setzen Sie die Eigenschaft `EnableAnimations` auf den Wert `True`, wenn der Animationseffekt für Einfügen und Löschen aktiviert werden soll. Dadurch werden in das Arbeitsblatt eingefügte Zeilen und Spalten langsam eingeblendet; aus dem Arbeitsblatt gelöschte Zeilen und Spalten verschwinden langsam.

6.1.36 »EnableAutoComplete«-Eigenschaft

Über die Eigenschaft `EnableAutoComplete` aktivieren (`True`) oder deaktivieren (`False`) Sie die automatische Vervollständigung von bereits bekannten Wörtern in Zellen.

6.1.37 »EnableCancelKey«-Eigenschaft

Über die Eigenschaft `EnableCancelKey` deaktivieren Sie die normale Funktion der Taste Esc, die Unterbrechung eines laufenden Makros. Dazu stehen folgende Konstanten zur Verfügung:

Konstante	Beschreibung
`xlDisabled`	Das Drücken der Abbruchtaste wird nicht beachtet.
`xlErrorHandler`	Die Unterbrechung wird als Fehler an die momentan ausgeführte Prozedur gesendet. Dieser Fehler kann dort durch eine Fehlerbehandlungsroutine behandelt werden, die mit einer `On Error GoTo`-Anweisung festgelegt wurde. Der Fehlercode des auffangbaren Fehlers ist 18.

Tabelle 6.6 Die Konstanten der Eigenschaft »EnableCancelKey«

Konstante	Beschreibung
xlInterrupt	Die aktuelle Prozedur wird unterbrochen, und der Benutzer kann die Prozedur testen oder beenden.

Tabelle 6.6 Die Konstanten der Eigenschaft »EnableCancelKey« (Forts.)

6.1.38 »EnableEvents«-Eigenschaft

Über die Eigenschaft `EnableEvents` schalten Sie die Ereignissteuerung von Excel ein (`True`) oder aus (`False`).

6.1.39 »FileDialog«-Eigenschaft

Siehe dazu Kapitel 10, »Dialogprogrammierung«.

6.1.40 »FindFormat«-, »ReplaceFormat«-Eigenschaft

Über die Eigenschaft `FindFormat` können Sie die Suchkriterien für den Typ der zu suchenden Zellenformate ermitteln oder festlegen. Sie haben damit die Möglichkeit, nach Formatierungen zu suchen und diese dann über die Eigenschaft `ReplaceFormat` zu ersetzen.

Beispiel

Im Beispiel aus Listing 6.29 wird in der aktiven Tabelle nach einer bestimmten Formatierung gesucht und diese dann durch eine andere ersetzt.

```
Sub FindFormat_Beispiel()
 With Application.FindFormat.Font
    .Name = "Arial"
    .Size = 10
 End With
 With Application.ReplaceFormat.Font
    .Name = "Arial Narrow"
    .Size = 12
 End With
 Cells.Replace What:="", Replacement:="", _
 LookAt:=xlPart, SearchOrder:=xlByRows, _
 MatchCase:=False, SearchFormat:=True, _
 ReplaceFormat:=True
End Sub
```

Listing 6.29 Formate suchen und ersetzen

6.1.41 »FixedDecimal«-, »FixedDecimalPlaces«-Eigenschaft

Mit der Eigenschaft `FixedDecimal` werden alle eingegebenen Daten mit einer festen Anzahl an Dezimalstellen formatiert, die durch die `FixedDecimalPlaces`-Eigenschaft festgelegt wurde.

Beispiel

Das Beispiel aus Listing 6.30 teilt alle Eingaben von Zahlenwerten in Excel automatisch durch 1.000.

```
Sub FixedDecimal_Beispiel()
 Application.FixedDecimal = True
 Application.FixedDecimalPlaces = 3
End Sub
```

Listing 6.30 Daten mit fester Dezimalstellenanzahl formatieren

6.1.42 »Height«-, »Width«-, »Top«-, »Left«-Eigenschaft

Über die Eigenschaft `Height` legen Sie die Höhe des Anwendungsfensters fest oder fragen sie ab. Wenn das Fenster zum Symbol minimiert wurde, ist diese Eigenschaft schreibgeschützt und gibt die Höhe des Symbols an. Ist das Fenster maximiert, kann diese Eigenschaft nicht festgelegt werden.

Über die Eigenschaft `Width` legen Sie den Abstand zwischen dem linken und dem rechten Rand des Anwendungsfensters fest.

Die Eigenschaft `Top` liefert der Abstand vom oberen Rand des Bildschirms zum oberen Rand des Microsoft-Excel-Hauptfensters.

Die Eigenschaft `Left` liefert den Abstand von der linken Seite des Bildschirms zur linken Seite des Microsoft-Excel-Hauptfensters.

Beispiel

Das Beispiel aus Listing 6.31 ermittelt die Abmessungen des Anwendungsfensters.

```
Sub HeightWidth_Beispiel()
 MsgBox "Das Fenster hat die Abmessungen" & vbLf & _
 Application.Height & " auf " & Application.Width
End Sub
```

Listing 6.31 Die Abmessungen des Anwendungsfensters ermitteln

6.1.43 »IgnoreRemoteRequests«-Eigenschaft

Über die Eigenschaft `IgnoreRemoteRequests` können Sie DDE-Fernabfragen ignorieren. Setzen Sie dafür die Eigenschaft auf den Wert `True`.

6.1.44 »Interactive«-Eigenschaft

Über die Eigenschaft `Interactive` bringen Sie Microsoft Excel in den interaktiven Modus (`True`). Setzen Sie diese Eigenschaft auf `False`, so ignoriert Microsoft Excel jegliche Eingabe von Tastatur oder Maus. Sehr praktisch ist diese Eigenschaft, wenn während DDE-Abfragen, Web-Abfragen oder OLE-Automatisierungen keine Störungen zugelassen werden sollen.

6.1.45 »International«-Eigenschaft

Die Eigenschaft `International` gibt Informationen über die aktuellen landesspezifischen/regionalen und internationalen Einstellungen zurück. Diese Eigenschaft kann nicht gesetzt werden. Eine komplette Auflistung der Konstanten finden Sie in der Onlinehilfe von VBA.

Beispiel

Das Beispiel aus Listing 6.32 gibt den Ländercode sowie das Dezimaltrennzeichen der installierten Anwendung aus.

```
Sub International_Beispiel()
 MsgBox Application.International(xlCountryCode) & _
 vbLf & Application.International(xlListSeparator)
End Sub
```

Listing 6.32 Ländereinstellungen auslesen

6.1.46 »Iteration«-, »MaxChange«-, »MaxIteration«-Eigenschaft

Setzen Sie die Eigenschaft `Iteration` auf den Wert `True`, wenn Zirkelbezüge in Microsoft Excel durch Iteration aufgelöst werden sollen.

Die Eigenschaft `MaxChange` gibt die maximal zulässige Anzahl an Änderungen zwischen jeder Iteration beim Auflösen von Zirkularbezügen in Microsoft Excel zurück oder legt sie fest.

Die Eigenschaft `MaxIteration` gibt die maximale Anzahl von Iterationsschritten zurück, die bei der Auflösung von Zirkularbezügen durchlaufen werden können, oder legt sie fest.

6.1.47 »MailSystem«-Eigenschaft

Die Eigenschaft `MailSystem` gibt den Namen des Mailsystems zurück, das auf Ihrem Computer installiert ist.

Beispiel

Das Beispiel aus Listing 6.33 ermittelt das installierte Mailsystem.

```
Sub MailSystem_Beispiel()
  Select Case Application.MailSystem
      Case xlMAPI
          MsgBox "Mailsystem: Microsoft Mail"
      Case xlPowerTalk
          MsgBox "Mailsystem: PowerTalk"
      Case xlNoMailSystem
          MsgBox "Kein Mailsystem installiert!"
  End Select
End Sub
```

Listing 6.33 Installiertes Mailsystem ermitteln

6.1.48 »MemoryFree«-, »MemoryUsed«-Eigenschaft

Die Eigenschaft `MemoryFree` gibt die Größe des Arbeitsspeichers in Byte zurück, der Microsoft Excel noch zur Verfügung steht.

Die Eigenschaft `MemoryUsed` gibt den Arbeitsspeicherplatz in Byte zurück, den Microsoft Excel gegenwärtig verwendet.

6.1.49 »MouseAvailable«-Eigenschaft

Über der Eigenschaft `MouseAvailable` prüfen Sie, ob eine Maus am Computer angeschlossen ist. In diesem Fall gibt diese Eigenschaft den Wert `True` zurück.

6.1.50 »MoveAfterReturn«-, »MoveAfterReturnDirection«-Eigenschaft

Setzen Sie die Eigenschaft `MoveAfterReturn` auf den Wert `True`, wenn die aktive Zelle nach dem Drücken der Taste ↵ verschoben werden soll.

Über die Eigenschaft `MoveAfterReturnDirection` legen Sie fest, wohin die aktive Zelle nach dem Drücken der Taste ↵ verschoben wird. Dazu stehen folgende Richtungskonstanten zur Verfügung:

Konstante	Beschreibung
`xlDown`	Markierung wird nach unten verschoben.
`xlToLeft`	Markierung wird nach links verschoben.

Tabelle 6.7 Die Richtungskonstanten der Eigenschaft »MoveAfterReturnDirection«

Konstante	Beschreibung
xlToRight	Markierung wird nach rechts verschoben.
xlUp	Markierung wird nach oben verschoben.

Tabelle 6.7 Die Richtungskonstanten der Eigenschaft »MoveAfterReturnDirection« (Forts.)

6.1.51 »Name«-, »Names«-Eigenschaft

Über die Eigenschaft `Name` fragen Sie den Namen der Anwendung ab. So liefert die Anweisung `MsgBox Application.Name` das Ergebnis »Microsoft Excel«.

Über die Eigenschaft `Names` können Sie alle verwendeten Namen der Arbeitsmappe ermitteln und die genaue Zellenposition aufspüren.

Beispiel

Das Beispiel aus Listing 6.34 gibt alle Namen einer Mappe im Direktfenster der Entwicklungsumgebung aus.

```
Sub Names_Beispiel()
  Dim Namen As Name

   For Each Namen In Application.Names
    Debug.Print Namen.Value; vbTab; Namen.Name
   Next Namen
End Sub
```

Listing 6.34 Alle verwendeten Namen einer Arbeitsmappe ermitteln

6.1.52 »NetWorkTemplatePath«-Eigenschaft

Die Eigenschaft `NetWorkTemplatePath` gibt den Netzwerkpfad zurück, unter dem Vorlagen gespeichert sind. Existiert der Netzwerkpfad nicht, liefert die Eigenschaft eine leere Zeichenfolge zurück.

6.1.53 »OperatingSystem«-, »Version«-, »Build«-Eigenschaft

Die Eigenschaft `OperatingSystem` gibt die Versionsnummer des aktuellen Betriebssystems, also Windows, aus.

Die Eigenschaft `Version` liefert die Versionsnummer von Microsoft Excel.

Die Eigenschaft `Build` gibt die interne Versionsnummer von Microsoft Excel zurück.

6.1.54 »OrganizationName«-, »UserName«-Eigenschaft

Die Eigenschaft `OrganizationName` gibt den Namen der registrierten Organisation aus.

Die Eigenschaft `UserName` liefert den Namen des aktuellen Benutzers zurück oder legt ihn fest.

6.1.55 »Parent«-Eigenschaft

Die Eigenschaft `Parent` gibt das übergeordnete Objekt für das angegebene Objekt zurück.

Beispiel

Das Beispiel aus Listing 6.35 stattet die Fußzeile einer Tabelle mit der Dokumenteigenschaft »Firma« aus.

```
Sub Parent_Beispiel()
 With ActiveSheet
   .PageSetup.RightFooter = _
   .Parent.BuiltinDocumentProperties("Company")
 End With
End Sub
```

Listing 6.35 Fußzeile mit Dokumenteigenschaft füllen

6.1.56 »Path«-Eigenschaft

Über die Eigenschaft `Path` fragen Sie das Installationsverzeichnis von Excel ab.

Beispiel

Das Beispiel aus Listing 6.36 fragt das Installationsverzeichnis von Excel ab.

```
Sub Path_Beispiel()
 MsgBox Application.Path
End Sub
```

Listing 6.36 Das Installationsverzeichnis von Office abfragen

6.1.57 »PreviousSelections«-Eigenschaft

Über die Eigenschaft `PreviousSelections` ermitteln Sie die vier zuletzt markierten Bereiche oder Namen, sofern diese über die Methode `Goto` bzw. über den Dialog GEHE ZU aktiviert und markiert wurden.

Beispiel

Das Beispiel aus Listing 6.37 ermittelt die zuletzt markierten Bereiche.

```
Sub PreviousSelection_Beispiel()
 On Error Resume Next
 For i = LBound(Application.PreviousSelections) To _
         UBound(Application.PreviousSelections)
    Debug.Print _
    Application.PreviousSelections(i).Address
 Next i
End Sub
```

Listing 6.37 Die zuletzt markierten Bereiche auslesen

6.1.58 »PromptForSummaryInfo«-Eigenschaft

Die Eigenschaft `PromptForSummaryInfo` kann eingesetzt werden, um Microsoft Excel so einzustellen, dass beim ersten Speichern von Dateien der Datei-Eigenschaften-Dialog aufgerufen wird.

6.1.59 »Ready«-Eigenschaft

Über die Eigenschaft `Ready` fragen Sie ab, ob die Anwendung bereit ist oder ob sie gerade beschäftigt ist.

6.1.60 »RecordRelative«-, »ReferenceStyle«-Eigenschaft

Die Eigenschaft `RecordRelative` betrifft die Einstellung des Makrorekorders. Setzen Sie diese Eigenschaft auf den Wert `True`, um Makros mit relativen Bezügen aufzuzeichnen. Stellen Sie hingegen den Wert `False` ein, um die Aufzeichnung absolut durchzuführen.

Über die Eigenschaft `ReferenceStyle` bestimmen Sie, wie Microsoft Excel Zellbezüge sowie Zeilen- und Spaltenköpfe anzeigt. Dabei gibt es die Möglichkeit des Z1S1-Stils oder der A1-Bezugsart.

6.1.61 »RollZoom«-Eigenschaft

Die Eigenschaft `RollZoom` bestimmt, wie eine IntelliMouse reagieren soll. Setzen Sie diese Eigenschaft auf den Wert `True`, damit die IntelliMouse bei Betätigung des Rädchens den Bereich vergrößert, anstatt einen Bildlauf durchzuführen.

6.1.62 »ScreenUpdating«-Eigenschaft

Die Eigenschaft `ScreenUpdating` kann die Laufzeit von Makros erheblich verkürzen. Dabei können Sie die Bildschirmaktualisierung abschalten, indem Sie dieser Eigenschaft den Wert `False` zuweisen.

6.1.63 »SheetsInNewWorkbook«-Eigenschaft

Über die Eigenschaft `SheetsInNewWorkbook` legen Sie fest, wie viele Tabellen automatisch erstellt werden, wenn Sie eine neue Arbeitsmappe anlegen.

Beispiel

Beim Beispiel aus Listing 6.38 wird eine neue Arbeitsmappe angelegt, die bereits zwölf Tabellen enthält. Speichern Sie davor die aktuell eingestellte Anzahl, und setzen Sie sie am Ende des Makros wieder auf diesen Wert zurück.

```
Sub SheetsInNewWorkbook_Beispiel()
 Dim iAnz As Integer

 iAnz = Application.SheetsInNewWorkbook
 Application.SheetsInNewWorkbook = 12
 Workbooks.Add
 Application.SheetsInNewWorkbook = iAnz
End Sub
```

Listing 6.38 Anzahl Tabellen einer neuen Mappe festlegen

6.1.64 »ShowWindowsInTaskbar«-Eigenschaft

Setzen Sie die Eigenschaft `ShowWindowsInTaskBar` auf den Wert `True`, um für jede geöffnete Arbeitsmappe eine eigene Schaltfläche auf der Windows-Taskleiste anzuzeigen.

6.1.65 »StandardFont«-, »StandardFontSize«-Eigenschaft

Die Eigenschaft `StandardFont` gibt den Namen der Standardschriftart zurück oder legt ihn fest.

Die Eigenschaft `StandardFontSize` gibt den Standardschriftgrad in Punkt zurück oder legt ihn fest.

Beide Einstellungen werden erst nach erneutem Excel-Start wirksam.

Beispiel

Das Beispiel aus Listing 6.39 legt als Standardschriftart Arial sowie als Standardschriftgröße 10 Punkte fest.

```
Sub StandardFont_Beispiel()
 With Application
  .StandardFont = "Arial"
  .StandardFontSize = 10
 End With
End Sub
```

Listing 6.39 Standardschriftart und -größe festlegen

6.1.66 »StatusBar«-Eigenschaft

Über die Eigenschaft `StatusBar` können Sie den Text der Statusleiste abfragen oder festlegen. Gerade bei länger laufenden Makros wird die Statusleiste oft dazu eingesetzt, die einzelnen Schritte eines Makros aufzuzeigen.

Beispiel

Das Makro aus Listing 6.40 ruft jede Minute ein bestimmtes Makro auf. Dieser Vorgang wird in der Statusleiste vermerkt.

```
Sub Zeitmakro()
  Application.OnTime Now + TimeValue("00:01:00"), "StatusBar_Beispiel"
End Sub

Sub StatusBar_Beispiel()
  Application.StatusBar = Date & "," & Time
  Call Zeitmakro
End Sub
```

Listing 6.40 Die Statusleiste einsetzen

6.1.67 »UsableHeight«-, »UsableWidth«-Eigenschaft

Die Eigenschaft `UsableHeight` gibt in Punkt die maximale Höhe der Fläche zurück, die ein Fenster innerhalb des Anwendungsfensters einnehmen kann.

Die Eigenschaft `UsableWidth` gibt die Breite der Fläche zurück, die ein Fenster innerhalb des Anwendungsfensters einnehmen kann.

Hinweis

Beide Eigenschaften werden in der Einheit Punkte ausgegeben. Ein Punkt bezieht sich auf die Höhe eines gedruckten Zeichens und entspricht ca. 0,03527 cm.

6.1.68 »UsedObjects«-Eigenschaft

Die Eigenschaft `UsedObjects` gibt ein `UsedObjects`-Objekt zurück, das Objekte darstellt, die in einer Arbeitsmappe zugewiesen sind.

Beispiel

Das Beispiel aus Listing 6.41 gibt alle verwendeten Objekte der Anwendung im Direktfenster der Entwicklungsumgebung aus.

```
Sub UsedObjects_Beispiel()
 Dim obj As Object

  For Each obj In Application.UsedObjects
    Debug.Print TypeName(obj)
  Next obj
End Sub
```

Listing 6.41 Die verwendeten Objekte einer Anwendung ausgeben

6.1.69 »UserControl«-Eigenschaft

Die Eigenschaft `UserControl` meldet den Wert `True`, wenn die Anwendung sichtbar ist oder vom Benutzer selbst erstellt oder gestartet wurde. Wurde die Anwendung als Programm mit Hilfe der Funktionen `CreateObject` oder `GetObject` erstellt bzw. gestartet und ist sie nicht sichtbar, dann meldet die Eigenschaft den Wert `False`.

6.1.70 »UserLibraryPath«-Eigenschaft

Die Eigenschaft `UserLibraryPath` liefert den Pfad zu dem Speicherort auf dem Computer des Benutzers zurück, an dem die COM-Add-Ins installiert sind.

6.1.71 »VBE«-Eigenschaft

Die Eigenschaft VBE gibt ein `VBE`-Objekt zurück, das den Visual-Basic-Editor darstellt.

Beispiel

Das Beispiel aus Listing 6.42 ruft die Entwicklungsumgebung per Makro auf.

```
Sub VBE_Beispiel()
  Application.VBE.MainWindow.Visible = True
End Sub
```

Listing 6.42 Die Entwicklungsumgebung per Makro aufrufen

6.1.72 »Visible«-Eigenschaft

Die Eigenschaft `Visible` bestimmt, ob ein Objekt angezeigt wird oder nicht.

Beispiel

Im Beispiel aus Listing 6.43 wird eine neue Excel-Applikation erstellt, eine neue Mappe eingefügt und sichtbar gemacht.

```
Sub Visible_Beispiel()
 Dim Xl As Excel.Application

 Set Xl = New Excel.Application
 Xl.Workbooks.Add
 Xl.Visible = True
End Sub
```

Listing 6.43 Neue Anwendungssitzung aufrufen und sichtbar machen

6.1.73 »Watches«-Eigenschaft

Die Eigenschaft (ab Excel 2002) `Watches` gibt ein `Watches`-Objekt zurück, das einen Bereich darstellt, der bei der Neuberechnung eines Arbeitsblattes protokolliert wird.

Beispiel

Im Beispiel aus Listing 6.44 wird eine bestimmte Zelle überwacht und dazu im Überwachungsfenster eingefügt.

```
Sub Watch_Beispiel()
 With Application
    .Range("A1").Value = 1
    .Range("A2").Value = 2
    .Range("A3").Formula = "=Sum(A1:A2)"
    .Watches.Add Source:=Range("A3").Address
```



```
 End With
End Sub
```

Listing 6.44 Überwachung hinzufügen

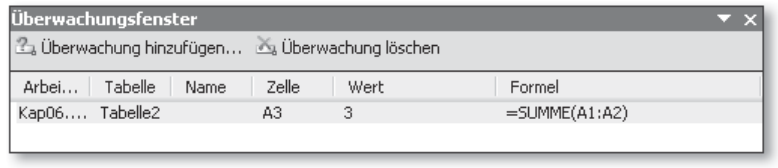

Abbildung 6.1 Zelle A3 wird überwacht.

6.1.74 »Windows«-Eigenschaft

Die Eigenschaft `Windows` gibt für ein `Application`-Objekt eine Windows-Auflistung zurück, die alle Fenster in allen Arbeitsmappen darstellt.

Beispiel

Das Beispiel aus Listing 6.45 gibt den Windows-Titel jeder geöffneten Excel-Mappe im Direktbereich der Entwicklungsumgebung aus.

```
Sub Windows_Beispiel()
 Dim fenster As Window

 For Each fenster In Application.Windows
   Debug.Print fenster.Caption
 Next fenster
End Sub
```

Listing 6.45 Fenstertitel ermitteln

6.1.75 »WindowState«-Eigenschaft

Über die Eigenschaft `WindowState` können Sie den Status des Fensters abfragen oder festlegen. Dazu stehen folgende Konstanten zur Verfügung:

Konstante	Beschreibung
`xlMaximized`	Fenster ist maximiert.
`xlNormal`	Fenster liegt in Normalgröße vor.
`xlMinimized`	Fenster ist minimiert.

Tabelle 6.8 Die Konstanten der Eigenschaft »WindowState«

Beispiel

Das Beispiel aus Listing 6.46 fragt den momentanen Fensterstatus ab.

```
Sub WindowState_Beispiel()
 Dim s As String

  Select Case Application.WindowState
      Case xlMaximized
          s = "xlMaximized"
      Case xlMinimized
          s = "xlMinimized"
      Case xlNormal
          s = "xlNormal"
  End Select
  MsgBox "Window-Status: " & s
End Sub
```

Listing 6.46 Fensterstatus abfragen

6.2 »Applications«-Methoden

6.2.1 »ActivateMicrosoftApp«-Methode

Die Methode `ActivateMicrosoftApp` aktiviert eine Microsoft-Anwendung. Falls die Anwendung bereits ausgeführt wird, aktiviert diese Methode das betreffende Anwendungsfenster. Wird die Anwendung noch nicht ausgeführt, startet die Methode eine neue Instanz der Anwendung. Folgende Konstanten stehen dabei zur Verfügung:

Konstante	Anwendung
`xlMicrosoftWord`	Word
`xlMicrosoftPowerPoint`	PowerPoint
`xlMicrosoftMail`	Outlook, Outlook Express
`xlMicrosoftAccess`	Access
`xlMicrosoftFoxPro`	FoxPro
`xlMicrosoftProject`	Project

Tabelle 6.9 Die Konstanten der Methode »ActivateMicrosoftApp«

Beispiel

Das Beispiel aus Listing 6.47 startet die Anwendung Outlook.

```
Sub ActivateMicrosoftApp_Beispiel()
 Application.ActivateMicrosoftApp xlMicrosoftMail
End Sub
```

Listing 6.47 Outlook starten über die Methode »ActivateMicrosoftApp«

6.2.2 »AddChartAutoFormat«-, »DeleteChartAutoFormat«-Methode

Die Methode `AddChartAutoFormat` fügt der Liste verfügbarer Diagramme ein benutzerdefiniertes Diagramm hinzu.

Syntax

```
Ausdruck.AddChartAutoFormat(Chart, Name, Description)
```

Argument	Beschreibung
`Ausdruck`	Erforderlich. Ein Ausdruck, der ein `Application`-Objekt zurückgibt
`Chart`	Erforderlich. Ein Diagramm mit dem Format, das dem neuen Diagramm als AutoFormat zugeordnet werden soll
`Name`	Erforderlich. Der Name des AutoFormats
`Description`	Optional. Eine Beschreibung des benutzerdefinierten AutoFormats

Die Methode `DeleteChartAutoFormat` entfernt ein benutzerdefiniertes Diagramm-AutoFormat aus der Liste verfügbarer Diagramme.

Beispiel

Das Beispiel aus Listing 6.48 stellt ein Diagramm im Diagramm-Assistenten im Register Benutzerdefinierte Typen ein.

```
Sub AddChartAutoFormat_Beispiel()
 Application.AddChartAutoFormat _
 Chart:=Charts("Diagramm1"), _
 Name:="Eigenes Diagramm", _
 Description:="Mein eigenes Diagramm"
End Sub
```

Listing 6.48 Ein benutzerdefiniertes Diagramm hinzufügen

6.2.3 »AddCustomList«-, »GetCustomListContents«-, »DeleteCustomList«-Methode

Die Methode `AddCustomList` fügt eine benutzerdefinierte Liste für benutzerdefiniertes AutoAusfüllen und benutzerdefiniertes Sortieren den bereits bestehenden Listen hinzu.

Die Methode `GetCustomListContents` gibt eine benutzerdefinierte Liste (ein Array mit Zeichenfolgen) zurück.

Die Methode `DeleteCustomList` löscht eine benutzerdefinierte Liste. Es wird eine Fehlermeldung erzeugt, wenn die Listennummer kleiner 5 ist oder die entsprechende benutzerdefinierte Liste nicht existiert.

Beispiele

Das Beispiel aus Listing 6.50 legt eine Liste mit Automobilherstellern an.

```
Sub AddCustomList_Beispiel()
 Application.AddCustomList Array _
 ("DaimlerChrysler", "Volkswagen", "Volvo", "Audi", _
 "Opel", "Suzuki", "Toyota", "Mitsubishi", "Porsche")
End Sub
```

Listing 6.49 Eine Autoherstellerliste generieren

Das Makro aus Listing 6.50 fügt die gerade angelegte Liste in eine neue Tabelle ein.

```
Sub GetCutstomListContents_Beispiel()
 Dim Tabelle As Worksheet
 Dim VList As Variant
 Dim i As Integer

 Set Tabelle = Worksheets.Add
 VList = Application.GetCustomListContents(5)
 For i = LBound(VList, 1) To UBound(VList, 1)
     Tabelle.Cells(i, 1).Value = VList(i)
 Next i
End Sub
```

Listing 6.50 Eine benutzerdefinierte Liste in eine Tabelle einfügen

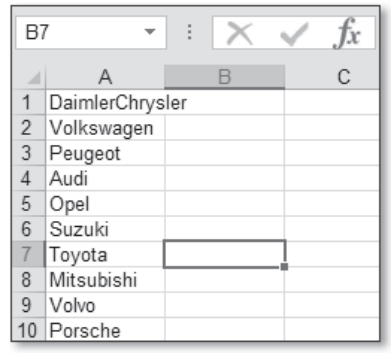

Abbildung 6.2 Die benutzerdefinierte Liste wurde eingefügt.

6.2.4 »Calculate«-, »CalculateFull«-, »CalculateFullRebuild«-Methode

Die Methode `Calculate` berechnet alle geöffneten Arbeitsmappen, ein bestimmtes Arbeitsblatt einer Arbeitsmappe oder einen bestimmten Zellbereich in einem Arbeitsblatt.

Die Methode `CalculateFull` erzwingt eine vollständige Berechnung der Daten in allen geöffneten Arbeitsmappen.

Die Methode `CalculateFullRebuild` erzwingt für alle geöffneten Arbeitsmappen eine vollständige Berechnung der Daten und erstellt die Abhängigkeiten erneut.

6.2.5 »CentimetersToPoints«-, »InchesToPoints«-Methode

Die Methode `CentimetersToPoints` wandelt eine Maßangabe von Zentimeter in Punkt (0,35 mm) um.

Die Methode `InchesToPoints` wandelt eine Maßangabe von Zoll in Punkt um. Ein Punkt entspricht ca. 0,3527 mm (1/72 Zoll).

6.2.6 »CheckAbort«-Methode

Die Methode `CheckAbort` stoppt die Neuberechnung in einer Microsoft-Excel-Anwendung.

6.2.7 »CheckSpelling«-Methode

Die Methode `CheckSpelling` prüft die Rechtschreibung eines einzelnen Wortes. Sie gibt den Wert `True` zurück, wenn das Wort in einem der Wörterbücher gefunden wird.

Syntax

```
Ausdruck.CheckSpelling(Word, CustomDictionary, IgnoreUppercase)
```

Argument	Beschreibung
Ausdruck	Erforderlich. Ein Ausdruck, der ein Application-Objekt zurückgibt
Word	Erforderlich. Das zu überprüfende Wort
CustomDictionary	Optional. Das Benutzerwörterbuch, in dem das Word gesucht werden soll, das nicht im Standardwörterbuch zu finden ist
IgnoreUppercase	Optional. True, Microsoft Excel ignoriert alle Wörter in Großbuchstaben. Ist der Wert False, prüft Microsoft Excel auch Wörter, die in Großbuchstaben geschrieben sind. Wenn Sie dieses Argument nicht angeben, wird die aktuelle Einstellung verwendet.

Beispiel

Das Beispiel aus Listing 6.51 prüft, ob das Wort »Scooter« bereits im Wörterbuch vorhanden ist.

```
Sub CheckSpelling()
  If Application.CheckSpelling("Scooter") = True Then
   MsgBox "Wort gefunden!"
  Else
   MsgBox "Wort nicht gefunden!"
  End If
End Sub
```

Listing 6.51 Wort über die Methode »SpellChecking« überprüfen

6.2.8 »ConvertFormula«-Methode

Die Methode ConvertFormula konvertiert Zellbezüge in Formeln zwischen den Bezugsarten A1 und Z1S1, zwischen relativen und absoluten Bezügen oder beides.

Syntax

```
Ausdruck.ConvertFormula(Formula, FromReferenceStyle, ToReferenceStyle,
ToAbsolute, RelativeTo)
```


Argument	Beschreibung
Ausdruck	Erforderlich. Ein Ausdruck, der ein Application-Objekt zurückgibt
Formula	Erforderlich. Eine Zeichenfolge, die die zu konvertierende Formel enthält. Es muss eine gültige Formel sein, die mit einem Gleichheitszeichen beginnt.
FromReferenceStyle	Erforderlich. XlReferenceStyle-Wert
XlReferenceStyle	Erforderlich. Kann eine der folgenden XlReferenceStyle-Konstanten sein: xlA1 oder xlR1C1.
ToReferenceStyle	Optional. XlReferenceStyle-Wert. Die Bezugsart, die Sie zurückgeben möchten
XlReferenceStyle	Kann eine der folgenden XlReferenceStyle-Konstanten sein: xlA1 oder xlR1C1.
ToAbsolute	Optional. XlReferenceStyle-Wert: ▶ xlAbsolute ▶ xlAbsRowRelColumn ▶ xlRelRowAbsColumn ▶ xlRelative
RelativeTo	Optionaler Variant-Wert. Ein Range-Objekt, das eine Zelle enthält. Relative Bezüge verweisen auf diese Zelle.

Beispiel

Das Beispiel aus Listing 6.52 entfernt die Absolutbezüge aus allen markierten Zellen und ersetzt sie durch relative Bezüge.

```
Sub ConvertFormular_Beispiel()
 Dim zelle As Range

 For Each zelle In _
 Selection.SpecialCells(xlCellTypeFormulas)
   zelle.Formula = _
    Application.ConvertFormula(zelle.Formula, _
    xlA1, , xlRelative, zelle)
 Next zelle
End Sub
```

Listing 6.52 Absolutbezüge durch Relativbezüge tauschen

6.2.9 »DoubleClick«-Methode

Die Methode `DoubleClick` entspricht einem Doppelklick auf die aktive Zelle.

6.2.10 »FindFile«-, »GetOpenFilename«-, »GetSaveAsFilename«-Methode

Siehe Kapitel 10, »Dialogprogrammierung«

6.2.11 »GoTo«-Methode

Die Methode `GoTo` markiert einen Bereich oder eine Visual-Basic-Prozedur in einer beliebigen Arbeitsmappe und aktiviert diese, falls sie nicht bereits aktiv ist.

Syntax

```
Ausdruck.GoTo(Reference, Scroll)
```

Argument	Beschreibung
`Ausdruck`	Erforderlich. Ein Ausdruck, der ein `Application`-Objekt zurückgibt
`Reference`	Optional. Gibt das Ziel an. Kann ein `Range`-Objekt sein, eine Zeichenfolge, die einen Zellbezug in Z1S1-Bezugsart enthält, oder aber eine Zeichenfolge, die den Namen einer Visual-Basic-Prozedur enthält. Wenn Sie dieses Argument nicht angeben, ist das Ziel der letzte mit der `Goto`-Methode ausgewählte Bereich.
`Scroll`	Optional. Wenn `True`, führt Microsoft Excel die nötigen Bildläufe durch, damit die obere linke Zelle des Bereichs als obere linke Zelle des Fensters erscheint. Wenn `False`, führt Microsoft Excel keine Bildläufe durch. Der Standardwert ist `False`.

Beispiel

Das Beispiel aus Listing 6.53 steuert in *Tabelle2* Zelle D200 an und stellt die Bildlaufleisten dementsprechend ein.

```
Sub Goto_Beispiel()
 Application.Goto Reference:= _
 Worksheets("Tabelle1").Range("D200"), scroll:=True
End Sub
```

Listing 6.53 Zellenbereiche ansteuern über die Methode »GoTo«

6.2.12 »Help«-Methode

Über die Methode `Help` zeigen Sie ein Hilfethema an.

Syntax

```
Ausdruck.Help(HelpFile, HelpContextID)
```

Argument	Beschreibung
`Ausdruck`	Erforderlich. Ein Ausdruck, der ein `Application`-Objekt zurückgibt
`HelpFile`	Optional. Der Name der Hilfedatei, die angezeigt werden soll. Wenn Sie dieses Argument nicht angeben, wird die Microsoft-Excel-Hilfe angezeigt.
`HelpContextID`	Optional. Hier können Sie die Kontext-ID für das betreffende Hilfethema angeben. Wenn Sie dieses Argument nicht leer lassen, wird das Dialogfeld HILFETHEMEN angezeigt.

Beispiel

Das Beispiel aus Listing 6.54 ruft die Excel-Onlinehilfe auf.

```
Sub Help_Beispiel()
 Application.Help
End Sub
```

Listing 6.54 Excel-Onlinehilfe aufrufen

6.2.13 »InputBox«-Methode

Siehe Kapitel 10, »Dialogprogrammierung«.

6.2.14 »Intersect«-Methode

Die Methode `Intersect` gibt ein `Range`-Objekt zurück, das die rechteckige Schnittmenge von zwei oder mehreren Bereichen darstellt.

Beispiel

Das Beispiel aus Listing 6.55 prüft, ob die aktive Zelle in einem vorher definierten Bereich liegt.

```
Sub Intersect_Bespiel()
 Dim Bereich As Range

 Set Bereich = Sheets("Tabelle1").Range("A1:B5")
 If Application.Intersect _
 (Bereich, ActiveCell) Is Nothing Then
```

```
    MsgBox "Die aktive Zelle liegt nicht im Bereich"
 Else
    MsgBox "Die aktive Zelle liegt im Bereich"
 End If
End Sub
```

Listing 6.55 Prüfung, ob eine Zelle in einem bestimmten Bereich liegt

6.2.15 »MacroOptions«-Methode

Die Methode `MacroOptions` legt die Eigenschaften eines Makros/einer Funktion fest.

Syntax

```
Ausdruck.MacroOptions(Macro, Description, HasMenu, MenuText,
HasShortcutKey, ShortcutKey, Category, StatusBar, HelpContextID, HelpFile)
```

Argument	Beschreibung
`Ausdruck`	Erforderlich. Ein Ausdruck, der ein `Application`-Objekt zurückgibt
`Macro`	Optional. Der Name des Makros
`Description`	Optional. Die Beschreibung des Makros
`HasMenu`	Optional. Dieses Argument wird ignoriert.
`MenuText`	Optional. Dieses Argument wird ignoriert.
`HasShortcutKey`	Optional. Wenn `True`, wird dem Makro eine Tastenkombination zugeordnet (`ShortcutKey` muss ebenfalls angegeben werden). Wenn das Argument den Wert `False` besitzt, wird dem Makro keine Tastenkombination zugewiesen.
`ShortcutKey`	Optional. Erforderlich, wenn `HasShortcutKey` den Wert `True` besitzt. Andernfalls wird das Argument ignoriert.
`Category`	Optional. Eine Ganzzahl, die die Funktionskategorie des Makros angibt
`StatusBar`	Optionaler Variant-Wert. Der Statusleistentext des Makros
`HelpContextId`	Optional. Eine Ganzzahl, die die Kontext-ID für das Hilfethema angibt, das dem Makro zugeordnet ist
`HelpFile`	Optional. Der Name der Hilfedatei, die das Hilfethema enthält, das durch das Argument `HelpContextId` angegeben wurde

Das Argument `Category` hat folgende Bedeutung:

Zahl	Funktionskategorie
1	Finanzmathematik
2	Datum & Zeit
3	Math. & Trigonom.
4	Statistik
5	Matrix
6	Datenbank
7	Text
8	Logik
9	Information
10	Benutzerdefiniert

Tabelle 6.10 Die Bedeutung des Arguments »Category«

Beispiel

Das Beispiel aus Listing 6.56 weist eine Funktion der Funktionskategorie INFORMATION zu. Standardmäßig werden eigene Funktionen der Funktionskategorie BENUTZERDEFINIERT zugewiesen.

```
Function TabName()
 TabName = ActiveSheet.Name
End Function

Sub MacroOptions_Beispiel()
  Application.MacroOptions _
  Macro:="TabName", _
  Description:="Tabellennamen ermitteln", _
  Category:=9
End Sub
```

Listing 6.56 Eine Funktion einer Funktionskategorie zuweisen

6.2.16 »MailLogOn«-, »MailLogoff«-Methode

Die Methode `MailLogOn` meldet sich bei MAPI Mail oder Microsoft Exchange an und stellt eine Mailsitzung her.

Syntax

```
Ausdruck.MailLogon(Name, Password, DownloadNewMail)
```

Argument	Beschreibung
Ausdruck	Erforderlich. Ein Ausdruck, der ein Application-Objekt zurückgibt
Name	Optional. Der Name des Mailkontos oder des Microsoft-Exchange-Profils. Ohne Angabe wird der Standardname des Mailkontos eingesetzt.
Password	Optional. Das Kennwort des Mailkontos. Dieses Argument wird in Microsoft Exchange ignoriert.
DownloadNewMail	Optional. Wenn True, werden neue Nachrichten sofort heruntergeladen.

Die Methode MailLogoff beendet eine von Microsoft Excel hergestellte MAPI-Mail-Sitzung.

6.2.17 »OnKey«-Methode

Die Methode OnKey führt die angegebene Prozedur aus, wenn eine bestimmte Taste oder Tastenkombination gedrückt wird.

Syntax

```
Ausdruck.OnKey(Key, Procedure)
```

Argument	Beschreibung
Ausdruck	Erforderlich. Ein Ausdruck, der ein Application-Objekt zurückgibt
Key	Erforderlich. Eine Zeichenfolge, die die zu drückende Taste angibt
Procedure	Optional. Eine Zeichenfolge, die den Namen der auszuführenden Prozedur festlegt

Beispiel

Das Beispiel aus Listing 6.57 weist einem Makro eine Tastenkombination zu.

```
Sub TastenkombinationZuweisen()
    Application.OnKey "%9", "Onkey_Beispiel"
End Sub
```



```
Sub Onkey_Beispiel()
 With Application.ActiveCell
   MsgBox "Aktive Zelle ist in Zeile " & _
   .Row & " und Spalte " & .Column
 End With
End Sub
```

Listing 6.57 Über die Tastenkombination Alt + 9 wird die Adresse der aktiven Zelle am Bildschirm ausgegeben.

Hinweis

Die Belegung der einzelnen Tasten der Methode OnKey können Sie übersichtlich in der Onlinehilfe nachschlagen.

6.2.18 »OnRepeat«-, »OnUndo«-Methode

Über die Methode OnRepeat hinterlegen Sie den Befehl WIEDERHOLEN aus dem Menü BEARBEITEN mit einem anderen Text sowie mit einem anderen Makro.

Mit der Methode OnUndo weisen Sie dem Befehl RÜCKGÄNGIG aus dem Menü BEARBEITEN einen anderen Text sowie ein anderes Makro zu.

Beispiel

Das Beispiel aus Listing 6.58 benennt die beiden Befehle WIEDERHOLEN und RÜCKGÄNGIG aus dem Menü BEARBEITEN um und hinterlegt jeweils ein anderes Makro.

```
Sub OnRepeat_Beispiel()
 Application.OnRepeat "Aktion wiederholen", _
    "WiederholenMakro"
 Application.OnUndo "Aktion widerrufen", _
    "WiderrufenMakro"
End Sub
```

Listing 6.58 Auf die Befehle »Wiederholen« bzw. »Rückgängig« reagieren

6.2.19 »OnTime«-Methode

Über die Methode OnTime starten Sie ein Makro zu einem bestimmten Zeitpunkt.

Syntax

```
Ausdruck.OnTime(EarliestTime, Procedure, LatestTime, Schedule)
```

Argument	Beschreibung
Ausdruck	Erforderlich. Ein Ausdruck, der ein Application-Objekt zurückgibt
EarliestTime	Erforderlich. Die Zeit, zu der diese Prozedur ausgeführt werden soll
Procedure	Erforderlich. Der Name der auszuführenden Prozedur
LatestTime	Optional. Der späteste Zeitpunkt zum Ausführen der Prozedur
Schedule	Optional. True führt eine neue OnTime-Prozedur aus. False löscht eine vorher eingestellte Prozedur. Der Standardwert ist True.

Beispiel

Das Beispiel aus Listing 6.59 führt ein bestimmtes Makro an einem zukünftigen Datum um eine bestimmte Uhrzeit aus.

```
Sub OnTime_Beispiel()
  Application.OnTime DateValue("25.05.15") + _
  TimeValue("08:30:00"), "Abfrage"
End Sub
```

Listing 6.59 Am 25.05.2015 um 8:30 wird das Makro »Abfrage« automatisch gestartet.

Hinweis

Um diesen Countdown einzuleiten, starten Sie zu einem früheren Zeitpunkt das Makro und lassen den Computer eingeschaltet.

6.2.20 »Quit«-Methode

Die Methode Quit beendet Excel.

Beispiel

Das Beispiel aus Listing 6.60 speichert alle noch geöffneten Arbeitsmappen und beendet anschließend Excel.

```
Sub Quit_Beispiel()
 Dim Mappe As Workbook

 For Each Mappe In Application.Workbooks
  Mappe.Save
 Next Mappe
 Application.Quit
End Sub
```

Listing 6.60 Die Anwendung über die Methode »Quit« beenden

6.2.21 »Repeat«-Methode

Die Methode Repeat wiederholt die letzte Benutzeraktion.

6.2.22 »Run«-Methode

Die Methode Run führt ein Makro aus oder ruft eine Funktion auf. Diese Syntax eignet sich zur Ausführung eines Makros, das in Visual Basic oder der Microsoft-Excel-Makrosprache geschrieben wurde, oder zur Ausführung einer Funktion in einer DLL oder XLL.

Syntax

```
Ausdruck.Run(Macro, Arg1, Arg2..Arg30)
```

Argument	Beschreibung
Ausdruck	Erforderlich. Ein Ausdruck, der ein Application-Objekt zurückgibt
Macro	Optional. Das auszuführende Makro. Sie können entweder eine Zeichenfolge mit dem Makronamen angeben oder ein Range-Objekt, das die Position der Funktion bezeichnet, oder die Registerkennnummer einer registrierten DLL-(XLL-)Funktion.
Arg1-Arg30	Optional. Die Argumente, die an die Funktion weiterzuleiten sind

6.2.23 »SaveWorkspace«-Methode

Die Methode SaveWorkspace speichert den aktuellen Arbeitsbereich. Dabei werden alle geöffneten Arbeitsmappen unter einem Arbeitsbereich abgelegt. Dies ermöglicht ein schnelleres Öffnen aller im Arbeitsbereich verzeichneten Arbeitsmappen.

Syntax

```
Ausdruck.SaveWorkspace(Filename)
```

Argument	Beschreibung
Ausdruck	Erforderlich. Ein Ausdruck, der ein Application-Objekt zurückgibt
Filename	Optional. Der Name der gespeicherten Datei

6.2.24 »SendKeys«-Methode

Die Methode `SendKeys` sendet Tastenkombinationen an die aktive Anwendung.

Syntax

```
Ausdruck.SendKeys(Keys, Wait)
```

Argument	Beschreibung
`Ausdruck`	Optional. Ein Ausdruck, der ein `Application`-Objekt zurückgibt
`Keys`	Erforderlich. Die Taste oder Tastenkombination, die Sie als Text an die Anwendung senden möchten
`Wait`	Optional. Falls `True`, wartet Microsoft Excel das Ende der Verarbeitung ab, bevor die Steuerung an das Makro zurückgegeben wird. Bei `False` oder keiner Angabe wird das Makro weiter ausgeführt, ohne dass auf eine Tastenkombination gewartet wird.

Beispiel

Im eher theoretischen Beispiel aus Listing 6.61 wird eine Tabelle umbenannt, indem über die Methode `SendKeys` nacheinander einzelne Tastenkombinationen an die Anwendung gesendet werden.

```
Sub SendKeys_Bespiel()
 SendKeys "%T"
 SendKeys "b"
 SendKeys "~"
 SendKeys "Neue Tabelle"
 SendKeys "~"
End Sub
```

Listing 6.61 Über die Methode »SendKeys« Tastenkombinationen senden

> **Hinweis**
>
> Die Tastenbelegung der Methode `SendKeys` können Sie übersichtlich und schnell in der Onlinehilfe nachsehen.

6.2.25 »SetDefaultChart«-Methode

Über die Methode `SetDefaultChart` können Sie den Namen der Diagrammvorlage angeben, die Microsoft Excel beim Erstellen neuer Diagramme standardmäßig verwenden soll.

Syntax

```
Ausdruck.SetDefaultChart(FormatName)
```

Argument	Beschreibung
`Ausdruck`	Erforderlich. Ein Ausdruck, der ein `Application`-Objekt zurückgibt
`FormatName`	Optional. Gibt den Namen eines benutzerdefinierten AutoFormats an. Dieser Name kann ein benutzerdefiniertes AutoFormat (als eine Zeichenfolge) oder aber die spezielle Konstante `xlBuiltIn` sein, mit der die integrierte Diagrammvorlage angegeben wird.

Beispiel

Das Beispiel aus Listing 6.62 ersetzt den standardmäßig eingestellten Diagrammtyp »Säulen« durch ein Liniendiagramm.

```
Sub SetDefaultChart_Beispiel()
 Application.SetDefaultChart _
 FormatName:=xlLineMarkers
End Sub
```

Listing 6.62 Das Standarddiagramm über die Methode »SetDefaultChart« ändern

6.2.26 »Undo«-Methode

Die Methode `Undo` macht die letzte Benutzeraktion rückgängig.

6.2.27 »Union«-Methode

Über die Methode `Union` vereinen Sie mehrere Bereiche einer Tabelle miteinander.

Beispiel

Das Beispiel aus Listing 6.63 vereint zwei nicht zusammenhängende Bereiche miteinander.

```
Sub Union_Beispiel()
 Dim Bereich1 As Range
 Dim Bereich2 As Range
 Dim Gesamt As Range

 Set Bereich1 = Range("A1:A5")
 Set Bereich2 = Range("C1:D2")
 Set Gesamt = Application.Union(Bereich1, Bereich2)
```

```
 Gesamt.Select
End Sub
```

Listing 6.63 Nicht zusammenhängende Bereiche vereinen

6.2.28 »Volatile«-Methode

Über die Methode `Volatile` wird eine benutzerdefinierte Funktion als veränderlich gekennzeichnet. Solche Funktionen werden immer neu berechnet, wenn in einer beliebigen Zelle des Arbeitsblattes eine Berechnung durchgeführt wird. Nicht veränderliche Funktionen werden nur dann neu berechnet, wenn sich die Eingabevariablen ändern.

6.2.29 »Wait«-Methode

Die Methode `Wait` hält das aktuell ausgeführte Makro bis zu einem angegebenen Zeitpunkt an.

Syntax

```
Ausdruck.Wait(Time)
```

Argument	Beschreibung
`Ausdruck`	Erforderlich. Ein Ausdruck, der ein `Application`-Objekt zurückgibt
`Time`	Erforderlich. Der Zeitpunkt, zu dem das Makro fortgesetzt werden soll (im Datumsformat von Microsoft Excel)

Beispiel

Im Beispiel aus Listing 6.64 wird ein Bild in eine Tabelle eingefügt und nach fünf Sekunden wieder gelöscht.

```
Sub Wait_Beispiel()
 Dim Pic As Picture

 ActiveSheet.Pictures.Insert("C:\Bild.jpg")
 AStunde = Hour(Now())
 AMinute = Minute(Now())
 ASekunde = Second(Now()) + 5
 PauseZeit = TimeSerial(AStunde, AMinute, ASekunde)
 Application.Wait PauseZeit
 ActiveSheet.Pictures(1).Delete
End Sub
```

Listing 6.64 Bild einfügen und nach fünf Sekunden wieder entfernen