Handbuch

BasicMaker 2010

Inhalt

Anweisungen und Funktionen von A-Z 279

Willkommen!

Willkommen bei BasicMaker!

Dieses Handbuch erläutert die Verwendung von BasicMaker, einer Programmierumgebung, mit der Sie TextMaker, PlanMaker und andere VBA-kompatible Programme über Scripts steuern können.

Das Handbuch ist in folgende Kapitel untergliedert:

- **Willkommen!**

  Das Kapitel, das Sie gerade lesen, enthält Informationen, wozu Sie BasicMaker verwenden können.

- **Bedienung des Script-Editors**

  Im zweiten Kapitel erfahren Sie alles über die Bedienung des Script-Editors von BasicMaker, mit dem Sie Ihre Scripts erstellen, ausführen und testen können.

- **Sprachelemente von SoftMaker Basic**

  Hier finden Sie grundlegende Informationen zur Syntax von SoftMaker Basic.

- **BasicMaker und TextMaker**

  BasicMaker wurde in erster Linie entwickelt, um eine Möglichkeit zu schaffen, TextMaker und PlanMaker programmieren zu können. Dieses Kapitel enthält alle Informationen zur Programmierung von TextMaker.

- **BasicMaker und PlanMaker**

  In diesem Kapitel finden Sie alle Informationen zur Programmierung von PlanMaker mit Hilfe von BasicMaker.

- **Anweisungen und Funktionen von A-Z**

  In diesem Kapitel finden Sie eine Beschreibung aller in SoftMaker Basic verfügbaren Anweisungen und Funktionen.

Was ist BasicMaker?

BasicMaker ist eine einfach zu bedienende Entwicklungsumgebung für die Programmiersprache *SoftMaker Basic*.

Was ist SoftMaker Basic?

SoftMaker Basic ist eine Programmiersprache, die sich am Standard *Visual Basic für Applikationen (VBA)* von Microsoft orientiert.

Es handelt sich hierbei um eine relativ einfach erlernbare Programmiersprache, die besonders auf die Zusammenarbeit mit *Applikationen* optimiert ist. So kann beispielsweise mit einigen einfachen Basic-Anweisungen die Schriftart in einem TextMaker-Dokument geändert, ein anderes Dokument geöffnet werden etc.

BasicMaker erzeugt keine direkt ausführbaren Programmdateien; es ist also kein Compiler zum Herstellen von ausführbaren Programmen enthalten. Vielmehr erstellen Sie mit BasicMaker sogenannte *Scripts*. Diese können in BasicMaker geöffnet und dann von BasicMaker aus ausgeführt werden.

Eine Übersicht über die Sprachelemente von SoftMaker Basic und deren Anwendung finden Sie im Kapitel "Sprachelemente von SoftMaker Basic". Eine A-Z-Referenz der Befehle gibt es im Kapitel "Anweisungen und Funktionen von A-Z".

Woraus besteht BasicMaker?

BasicMaker besteht aus folgenden Komponenten:

- Die Schaltzentrale von BasicMaker ist der *Script-Editor.* Damit erstellen und bearbeiten Sie SoftMaker Basic-Scripts. Die Bedienung des Editors wird im Kapitel "Bedienung des Script-Editors" erläutert.

- In diesen Editor ist ein *Interpreter* für die Programmiersprache SoftMaker Basic integriert. Dieser ist für die Ausführung der Scripts zuständig. SoftMaker Basic-Scripts können nicht zu ausführbaren Programmen kompiliert werden, sondern müssen stets vom Script-Editor aus gestartet werden.

  Zusätzlich gibt es die Möglichkeit, ein Script von TextMaker oder PlanMaker aus ausführen zu lassen. Rufen Sie darin den Befehl **Weiteres > Script starten** auf und wählen Sie das gewünschte Script, worauf BasicMaker startet und das Script ausführt.

  Weitere Informationen zum Ausführen von Scripts finden Sie im Abschnitt "Starten eines Scripts".

- Darüber hinaus ist in den Script-Editor ein *Debugger* zum Testen von Scripts integriert. Damit können Sie ein Script in Einzelschritten abarbeiten lassen und den Inhalt von Variablen überwachen, was die Fehlersuche erleichtert. Informationen dazu finden Sie im Abschnitt "Debuggen eines Scripts".

- Schließlich enthält BasicMaker einen grafischen *Dialogeditor*. Mit dessen Hilfe können Sie Dialogfenster erstellen, um einem Script Interaktionen mit dem Anwender zu ermöglichen. Informationen hierzu finden Sie im Abschnitt "Dialogeditor verwenden".

Bedienung des Script-Editors

In diesem Kapitel wird die Bedienung des Editors von BasicMaker erläutert:

- Starten von BasicMaker
- Befehle im Datei-Menü des Script-Editors
- Befehle im Bearbeiten-Menü des Script-Editors
- Befehle im Ansicht-Menü des Script-Editors
- Befehle im Einfügen-Menü des Script-Editors
- Befehle im Programm-Menü des Script-Editors
- Befehle im Weiteres-Menü des Script-Editors
- Befehle im Fenster-Menü des Script-Editors
- Starten eines Scripts
- Debuggen eines Scripts
- Dialogeditor verwenden

Starten von BasicMaker

Um BasicMaker aufzurufen, können Sie eine der folgenden Vorgehensweisen verwenden:

- **BasicMaker über das Start-Menü starten**

  Sie können BasicMaker starten, indem Sie im **Start**-Menü nacheinander auf die Einträge **Start > Programme > SoftMaker Office > BasicMaker** klicken.

  Es erscheint dann der *Script-Editor*. Mit diesem können Sie Ihre Scripts erstellen und ausführen. Die einzelnen Befehle des Editors stellen wir Ihnen in den folgenden Abschnitten vor.

- **BasicMaker von TextMaker/PlanMaker aus starten**

  Sie können BasicMaker auch von der Textverarbeitung TextMaker oder der Tabellenkalkulation PlanMaker aus starten. Rufen Sie dazu in TextMaker/PlanMaker den Befehl **Weiteres > Script bearbeiten** auf.

- **BasicMaker von TextMaker/PlanMaker aus starten und sofort ein Script ausführen lassen**

  Wenn Sie in TextMaker/PlanMaker den Befehl **Weiteres > Script starten** aufrufen, erscheint ein Dateidialog. Wählen Sie darin ein Script aus, wird BasicMaker gestartet, führt das Script aus und beendet sich danach wieder.

  Das Gleiche passiert, wenn Sie BasicMaker mit **basicmaker /s scriptname.bas** aufrufen: BasicMaker startet, führt das Script mit dem übergebenen Namen aus und beendet sich wieder.

Befehle im Datei-Menü des Script-Editors

Mit den Befehlen im Menü **Datei** des Script-Editors können Sie unter anderem Dateien öffnen, speichern, drucken und verwalten:

- **Datei > Neu**

  Legt ein neues Script-Fenster an.

- **Datei > Öffnen**

  Öffnet ein vorhandenes Script.

  Sie können auch VBA-Scripts (VBA = Visual Basic für Applikationen) öffnen, allerdings können nicht alle VBA-Befehle von BasicMaker ausgeführt werden.

- **Datei > Schließen**

  Schließt das aktuelle Fenster.

- **Datei > Speichern**

  Speichert das Script im aktuellen Fenster.

- **Datei > Speichern unter**

  Speichert das Script im aktuellen Fenster unter einem anderen Namen und/oder in einem anderen Ordner.

- **Datei > Alles speichern**

  Speichert alle Scripts in den derzeit geöffneten Fenstern, die seit dem letzten Speichern verändert wurden.

- **Datei > Dateimanager**

  Ruft den Dateimanager auf, mit dem Sie Dateien komfortabel suchen, öffnen, löschen und drucken können. Mehr dazu im Abschnitt "Dateimanager verwenden".

- **Datei > Seite einrichten**

  Lässt Sie Papierformat und Seitenränder für den Ausdruck einstellen.

- **Datei > Drucken**

  Gibt das aktuelle Script auf dem Drucker aus.

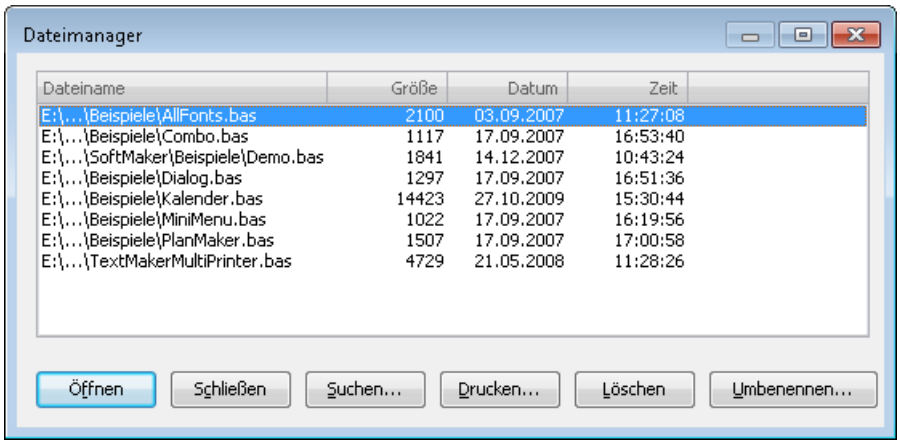
- **Datei > Beenden**

  Beendet BasicMaker.

Dateimanager verwenden

Der Dateimanager des Script-Editors zeigt eine Liste von Dateien aus einem oder mehreren Ordnern an und lässt Sie diese öffnen, löschen oder ausdrucken. Weiterhin können Sie nach bestimmten Dateien suchen.

Sie starten den Dateimanager mit dem Befehl **Datei > Dateimanager** oder durch Betätigen des Tastenkürzels F12.



Um eine Aktion durchzuführen, selektieren Sie mit den Richtungstasten oder per Mausklick eine Datei und klicken dann auf eine der Schaltflächen.

Die Schaltflächen haben folgende Funktionen:

Öffnen

Betätigen Sie diese Schaltfläche, wird die momentan selektierte Datei geöffnet.

Schließen

Diese Schaltfläche beendet den Dateimanager.

Suchen

Betätigen Sie diese Schaltfläche, um nach bestimmten Dateien zu suchen oder einfach nur den Ordner auszuwählen, der im Dateimanager angezeigt werden soll.

Dazu erscheint ein Dialogfenster mit folgenden Funktionen:

- **Dateiname**

  Hier können Sie den gesuchten Dateinamen oder eine mehrdeutige Dateimaske angeben. Bei der Standardeinstellung *.BAS findet die Suchfunktion alle Dateien mit der Namenserweiterung .bas, sprich: alle Basic-Scripts.

  Wenn Sie eine Suchmaske wie LIST*.BAS angeben, werden alle Scripts gefunden, deren Dateiname mit "list" beginnt.

  Wenn Sie einen eindeutigen Dateinamen wie LISTFONTS.BAS angeben, werden nur Dateien mit exakt diesem Namen gefunden.

- **Dateityp**

  Über diese Liste können Sie wählen, nach welchem Typ von Dateien gesucht werden soll.

- **Ordner**

  Hier legen Sie fest, in welchem Laufwerk und Ordner die Suche durchgeführt werden soll.

- **Untergeordnete Ordner einbeziehen**

  Ist dieser Schalter aktiviert, wird nicht nur der aktuelle Ordner durchsucht, sondern auch alle ihm untergeordneten Ordner.

- **Schaltfläche "Neue Dateiliste"**

  Startet eine neue Suche mit den aktuellen Einstellungen.

- **Schaltfläche "Liste ergänzen"**

  Hier wird ebenfalls mit der Suche begonnen – mit dem Unterschied, dass die Liste der bei der letzten Suche gefundenen Dateien nicht zuvor geleert wird. Die neuen Suchergebnisse werden den alten also hinzugefügt.

- **Schaltfläche "Schnellwahl"**

  Mit Hilfe von Schnellwahlpfaden können Sie die am häufigsten von Ihnen benötigten Ordner auf Knopfdruck abrufbar machen. Informationen dazu finden Sie in der Hilfe von TextMaker oder PlanMaker, Stichwort "Schnellwahlpfade".

Drucken

Betätigen Sie diese Schaltfläche, wird die selektierte Datei ausgedruckt.

Löschen

Betätigen Sie diese Schaltfläche, wird die selektierte Datei (nach Rückfrage) gelöscht.

Umbenennen

Betätigen Sie diese Schaltfläche, wird die selektierte Datei umbenannt. BasicMaker fragt Sie nach dem neuen Dateinamen.

Befehle im Bearbeiten-Menü des Script-Editors

Die Befehle im Menü **Bearbeiten** des Script-Editors dienen zum Bearbeiten von Scripts:

- **Bearbeiten > Rückgängig**

  Macht die zuletzt durchgeführte Textänderung im aktuellen Script-Fenster wieder rückgängig. Sie können diesen Befehl auch mehrmals hintereinander aufrufen, um die letzten x Änderungen ungeschehen zu machen.

- **Bearbeiten > Wiederherstellen**

  Stellt die zuletzt rückgängig gemachte(n) Textänderung(en) wieder her.

- **Bearbeiten > Ausschneiden**

  Schneidet den markierten Text in die Zwischenablage aus.

- **Bearbeiten > Kopieren**

  Kopiert den markierten Text in die Zwischenablage.

- **Bearbeiten > Einfügen**

  Fügt den Inhalt der Zwischenablage an der aktuellen Position der Schreibmarke ein.

- **Bearbeiten > Löschen**

  Löscht den markierten Text.

- **Bearbeiten > Alles markieren**

  Markiert den gesamten Text.

- **Bearbeiten > Suchen**

  Lässt Sie nach Text suchen. Mehr dazu im Abschnitt "Suchen und Ersetzen im Script-Editor".

- **Bearbeiten > Ersetzen**

  Lässt Sie Text suchen und durch anderen Text ersetzen. Mehr dazu im Abschnitt "Suchen und Ersetzen im Script-Editor".

- **Bearbeiten > Suchen wiederholen**

  Wiederholt den letzten Such- oder Ersetzungsvorgang. Mehr dazu im Abschnitt "Suchen und Ersetzen im Script-Editor".

- **Bearbeiten > Gehe zu...**

  Lässt Sie Textmarken (eine Art "Lesezeichen") im Script setzen und anspringen. Mehr dazu im Abschnitt "Textmarken und der Befehl Gehe zu...".

- **Bearbeiten > Dialoge bearbeiten**

  Ruft den grafischen Dialogeditor auf, mit dem Sie benutzerdefinierte Dialogfenster gestalten können. Informationen hierzu finden Sie im Abschnitt "Dialogeditor verwenden".

Suchen und Ersetzen im Script-Editor

Mit den Befehlen **Bearbeiten > Suchen** und **Bearbeiten > Ersetzen** können Sie nach einem bestimmten Text im Script suchen beziehungsweise diesen auch gleich durch einen anderen ersetzen lassen.

Suchen

Mit **Bearbeiten > Suchen** können Sie nach Text suchen. Geben Sie dazu den gewünschten Suchbegriff an und betätigen Sie die Schaltfläche **Suchen**.

Im Suchen-Dialog gibt es folgende Optionen:

Groß-/Kleinschreibung beachten: Wenn eingeschaltet, muss die Groß-/Kleinschreibung der Fundstelle exakt mit der des Suchbegriffs übereinstimmen. Bei der Suche nach "Print", wird also nur "Print" gefunden, nicht jedoch "print" oder "PRINT".

Nur ganze Wörter suchen: Wenn eingeschaltet, wird der Suchbegriff nur dann gefunden, wenn er als vollständiges Wort (also nicht als Wortteil) vorkommt.

Ab Dokumentanfang suchen: Wenn eingeschaltet, beginnt die Suche ab dem Beginn des Scripts, ansonsten ab der aktuellen Position der Schreibmarke.

Rückwärts suchen: Wenn eingeschaltet, wird rückwärts (von der Schreibmarke zum Beginn des Scripts) gesucht, ansonsten vorwärts.

Ersetzen

Mit dem Befehl **Bearbeiten > Ersetzen** können Sie Text suchen und diesen durch anderen Text ersetzen.

Geben Sie dazu den Suchbegriff und den Ersatzbegriff an.

Optionen: siehe oben

Starten Sie den Suchvorgang mit der Schaltfläche **Suchen**. Wenn der Suchbegriff gefunden wird, zeigt der Editor die Fundstelle an. Sie können nun die folgenden Schaltflächen verwenden, um zu entscheiden, ob der Begriff dort tatsächlich ersetzt werden soll:

Suchen: Begriff an der Fundstelle *nicht* ersetzen, Suche fortführen

Ersetzen: Begriff ersetzen, Suche dann fortführen

Alles ersetzen: Sämtliche weiteren Vorkommen des Begriffs ersetzen

Suchen wiederholen

Mit dem Befehl **Bearbeiten > Suchen wiederholen** können Sie den letzten Such- beziehungsweise Ersetzungsvorgang wiederholen.

Befehle im Ansicht-Menü des Script-Editors

Mit den Befehlen im Menü **Ansicht** des Script-Editors können Sie die Bildschirmdarstellung anpassen:

- **Ansicht > Symbolleisten**

  Lässt Sie die Symbolleisten (Funktionsleiste und Statuszeile) des Script-Editors ein-/ausschalten und konfigurieren. Mehr dazu im Abschnitt "Symbolleisten des Script-Editors anpassen".

- **Ansicht > Textmarken**

  Schaltet die Anzeige von Textmarken im Script-Editor ein-/aus.

- **Ansicht > Ausgabefenster**

  Zeigt das Ausgabefenster an. Wenn Sie in einem Script Ausgaben mit der **Print**-Anweisung vornehmen, werden diese darin angezeigt. Weiterhin werden dort Fehlermeldungen beim Abarbeiten von Scripts ausgegeben.

- **Ansicht > Variablenfenster**

  Zeigt das Variablenfenster an. Darin kann der Inhalt von Variablen während der Ausführung des Scripts überwacht werden. Mehr dazu im Abschnitt "Variablen beobachten".

- **Ansicht > Fensterpositionen speichern**

  Speichert Position und Größe des aktuellen Script-Fensters, des Ausgabefensters und des Variablenfensters.

- **Ansicht > Fensterpositionen abrufen**

  Ordnet das aktuelle Script-Fenster, das Ausgabefenster und das Variablenfenster wieder so an, wie es mit dem Befehl **Ansicht > Fensterpositionen speichern** gespeichert wurde.

- **Ansicht > Fensterpositionen zurücksetzen**

  Setzt Position und Größe des aktuellen Script-Fensters, des Ausgabefensters und des Variablenfensters wieder auf ihre Standardwerte zurück.

Befehle im Einfügen-Menü des Script-Editors

Im Menü **Einfügen** des Script-Editors stehen folgende Befehle zur Verfügung:

- **Einfügen > Sonderzeichen**

  Ruft die Zeichentabelle des Script-Editors auf, mit der Sie Sonderzeichen in den Text einfügen können. Wählen Sie dazu das gewünschte Sonderzeichen und bestätigen Sie mit **OK**.

- **Einfügen > Dokument**

  Fügt an der aktuellen Position der Schreibmarke ein anderes Script oder sonstiges Textdokument ein. Dazu erscheint ein Dateidialog, in dem Sie die einzufügende Datei auswählen.

- **Einfügen > Textbaustein**

  Fügt einen Textbaustein ein und lässt Sie Ihre Textbausteine bearbeiten. Mehr dazu im Abschnitt "Textbausteine verwenden".

  Tipp: Mit dieser Funktion können Sie sich viel Tipparbeit sparen, indem Sie die Namen häufig benötigter Befehle oder Routinen auf Textbausteine legen!

- **Einfügen > Textmarke**

  Setzt an der aktuellen Position eine Textmarke (eine Art "Lesezeichen"). Mit **Bearbeiten > Gehe zu...** kann diese jederzeit angesprungen werden. Mehr dazu im Abschnitt "Textmarken und der Befehl Gehe zu...".

Textbausteine verwenden

Genau wie in der Textverarbeitung TextMaker können Sie auch im Script-Editor von BasicMaker *Textbausteine* anlegen. Diese bieten eine enorme Arbeitserleichterung: Sie können sich Bausteine für häufig benötigte Namen oder Quelltextfragmente anlegen und diese dann blitzschnell abrufen.

Legen Sie beispielsweise einen Textbaustein namens "tma" mit dem Inhalt "tm.Application.ActiveDocument" an. Tippen Sie dann im Script einfach "tma" und die Leertaste, die Return-Taste oder ein Satzzeichen. Sofort wird "tma" durch "tm.Application.ActiveDocument" ersetzt.

Auf diese Weise können Sie beim Tippen viel Zeit sparen.

Textbausteine anlegen

Um beispielsweise einen Baustein mit dem Namen "tma" und dem Inhalt "tm.Application.ActiveDocument" anzulegen, gehen Sie wie folgt vor:

1. Rufen Sie den Befehl **Einfügen > Textbaustein** auf.
2. Klicken Sie auf die Schaltfläche **Neu**, um einen neuen Baustein anzulegen.
3. Geben Sie dem Baustein einen Namen (in unserem Beispiel also "tma"). Klicken Sie dann auf **OK**.

   Anhand seines Namens kann der Baustein später abgerufen werden.
4. Tippen Sie nun den Text für den Baustein ein (hier also: "tm.Application.ActiveDocument"). Klicken Sie dann auf **Speichern**.
5. Verlassen Sie den Dialog mit **Schließen**.

Der Baustein ist nun angelegt.

Textbausteine abrufen

Das Abrufen von Textbausteinen geht denkbar einfach: Tippen Sie im Script den Namen des Bausteins – in unserem Beispiel also "tma" – und dann ein Leerzeichen, ein Satzzeichen oder die Return-Taste. Sofort wird "tma" durch den Inhalt des Bausteins ersetzt.

Hinweis: Sollte dies nicht funktionieren, haben Sie die Option **Textbausteine automatisch ersetzen** deaktiviert. Rufen Sie den Befehl **Weiteres > Einstellungen** auf, wechseln Sie auf die Karteikarte **Allgemein** und schalten Sie diese Option wieder ein.

Alternativ können Sie den Baustein auch per Dialog einfügen, indem Sie den Befehl **Einfügen > Textbaustein** aufrufen, den gewünschten Baustein auswählen und dann die Schaltfläche **Einfügen** betätigen.

Textbausteine bearbeiten

Mit dem Befehl **Einfügen > Textbaustein** können Sie die bereits angelegten Textbausteine auch nachträglich bearbeiten:

- **Neuen Baustein anlegen**

  Betätigen Sie die Schaltfläche **Neu**, um einen neuen Textbaustein anzulegen (siehe oben).

- **Baustein löschen**

  Um einen Textbaustein zu löschen, selektieren Sie ihn in der Liste **Textbausteine** und betätigen dann die Schaltfläche **Löschen**.

- **Baustein umbenennen**

  Um den Namen eines Bausteins zu ändern, selektieren Sie ihn in der Liste, klicken auf die Schaltfläche **Umbenennen** und geben den neuen Namen ein.

- **Baustein bearbeiten**

  Um einen Baustein zu bearbeiten, selektieren Sie ihn in der Liste und klicken rechts in das Eingabefenster. Nun können Sie den Inhalt des Bausteins abändern.

- **Baustein einfügen**

  Um den Inhalt eines Bausteins in das Script einzufügen, selektieren Sie ihn in der Liste und klicken auf die Schaltfläche **Einfügen**.

- **Dialog schließen**

  Mit der Schaltfläche **Schließen** können Sie den Dialog verlassen.

Textmarken und der Befehl Gehe zu...

Genau wie in der Textverarbeitung TextMaker können Sie auch in Script-Editor von BasicMaker "Lesezeichen" anbringen, um bestimmte Stellen im Script schnell wieder finden zu können.

Rufen Sie dazu an der gewünschten Position den Befehl **Einfügen > Textmarke** auf und geben Sie der Textmarke einen Namen. Mit dem Befehl **Bearbeiten > Gehe zu** können Sie so markierte Textstellen nun jederzeit wieder aufschlagen.

Textmarken setzen

Um eine bestimmte Stelle in einem Script "mit einem Lesezeichen zu versehen", setzen Sie dort eine Textmarke. Dazu gehen Sie folgendermaßen vor:

1. Bewegen Sie die Schreibmarke an die Position, die mit einer Textmarke versehen werden soll.
2. Rufen Sie den Befehl **Einfügen > Textmarke** auf.
3. Tippen Sie einen beliebigen Namen für die Textmarke ein.

   Der Name darf nicht mit einer Ziffer beginnen und nicht länger als 20 Zeichen sein.
4. Klicken Sie auf **OK**, um die Textmarke zu setzen.

Sie können beliebig viele Textmarken definieren.

Textmarken abrufen

Um eine so gespeicherte Position wieder abzurufen, gehen Sie folgendermaßen vor:

1. Rufen Sie den Befehl **Bearbeiten > Gehe zu ...** auf oder drücken Sie F5.
2. Wählen Sie die gewünschte Textmarke aus der Liste oder tippen Sie ihren Namen von Hand ein.
3. Bestätigen Sie mit **OK**.

Die Schreibmarke wird nun an die Position gesetzt, an der Sie die Textmarke definiert hatten.

Textmarken löschen

Wird eine Textmarke nicht mehr benötigt, können Sie diese löschen. Dazu gehen Sie folgendermaßen vor:

1. Rufen Sie den Befehl **Einfügen > Textmarke** auf.
2. Wählen Sie die zu löschende Textmarke aus der Liste oder tippen Sie ihren Namen von Hand ein.
3. Klicken Sie auf **Löschen**.

Hinweis: Eine Textmarke wird automatisch gelöscht, wenn Sie die Textpassage, in der die Textmarke enthalten ist, löschen.

Schreibmarke in eine bestimmte Zeile setzen

Über den Befehl **Bearbeiten > Gehe zu** können Sie weiterhin die Schreibmarke in eine bestimmte Zeile des Scripts setzen. Rufen Sie den Befehl dazu auf und tippen Sie die gewünschte Zeilennummer ein.

Befehle im Programm-Menü des Script-Editors

Mit den Befehlen im Menü **Programm** des Script-Editors können Sie das aktuelle Script ausführen.

- **Programm > Starten** (Tastenkürzel: F9)

  Startet die Ausführung des Scripts.

  Ausführlichere Informationen zum Starten von Scripts erhalten Sie im Abschnitt "Starten eines Scripts".

Alle weiteren Befehle in diesem Menü dienen der Fehlersuche. So können Sie das Script beispielsweise in Einzelschritten ausführen oder Haltepunkte setzen, an denen die Ausführung automatisch unterbrochen wird.

Es gibt hierfür folgende Befehle:

- **Programm > Einzelschritt** (Tastenkürzel: F7)

  Führt die nächste Anweisung im Script aus und hält die Ausführung dann an.

- **Programm > Prozedurschritt** (Tastenkürzel: F8)

  Führt ebenfalls nur die nächste Anweisung aus – mit dem Unterschied, dass Prozeduren (Functions und Subs) hierbei nicht ebenfalls in Einzelschritten, sondern am Stück abgearbeitet werden.

- **Programm > Zurücksetzen** (Tastenkürzel: Strg+F2)

  Bricht die Einzelschritt-Ausführung ab und setzt das Script auf die erste Zeile zurück.

- **Programm > Haltepunkt setzen/löschen** (Tastenkürzel: F2)

  Setzt in der aktuellen Zeile einen Haltepunkt oder entfernt ihn wieder. Die Ausführung des Scripts wird automatisch unterbrochen, sobald es auf einen Haltepunkt stößt.

- **Programm > Alle Haltepunkte löschen** (Tastenkürzel: Alt+F2)

  Löscht alle gesetzten Haltepunkte im Script.

Ausführliche Informationen zu obigen Befehlen erhalten Sie im Abschnitt "Debuggen eines Scripts".

Befehle im Weiteres-Menü des Script-Editors

Mit den Befehlen im Menü **Weiteres** des Script-Editors können Sie den Editor konfigurieren.

- **Weiteres > Anpassen**

  Lässt Sie die Symbolleisten und die Tastenbelegung des Editors anpassen. Lesen Sie dazu die Abschnitte "Symbolleisten des Script-Editors anpassen" und "Tastaturbelegung des Script-Editors anpassen".

- **Weiteres > Einstellungen**

  Lässt Sie die Grundeinstellungen des Editors abändern. Lesen Sie dazu den Abschnitt "Einstellungen des Script-Editors ändern".

Einstellungen des Script-Editors ändern

Mit dem Befehl **Weiteres > Einstellungen** können Sie die Konfiguration des Script-Editors Ihren Arbeitsgewohnheiten anpassen.

Die verfügbaren Einstellungen sind auf mehrere Karteikarten verteilt:

Karteikarte "Ansicht"

Hier können Sie Einstellungen zum Erscheinungsbild des Programms vornehmen:

- **Schriftart** und **Größe**

  Hier lässt sich die Schriftart und Größe einstellen, die der Editor verwenden soll. Es empfiehlt sich, eine Schrift mit gleichen Zeichenbreiten (z.B. "Courier New") zu wählen.

- **Tabulator**

  Hier können Sie die Schrittweite für Tabulatoren einstellen. Diese bestimmt, um wie viele Zeichen der Text bei einem Tabulator eingerückt wird.

- **Textmarken anzeigen**

  Normalerweise sind Textmarken im Text nicht sichtbar. Wenn Sie diese Option hingegen aktivieren, werden Textmarken angezeigt. Informationen zum Einsatz von Textmarken finden Sie im Abschnitt "Textmarken und der Befehl Gehe zu...".

Karteikarte "Allgemein"

Hier können Sie allgemeine Einstellungen vornehmen:

- **Maximal widerrufbare Aktionen**

  Hier können Sie einstellen, wie viele Aktionen sich mit dem Befehl **Bearbeiten > Rückgängig** maximal widerrufen lassen.

- **Textbausteine automatisch ersetzen**

  Ist diese Option aktiviert, können Textbausteine direkt im Text ausgelöst werden. Dazu muss einfach das Kürzel für den Baustein und dann Leertaste, Return-Taste oder ein Satzzeichen getippt werden (siehe Abschnitt "Textbausteine verwenden").

  Ist die Option deaktiviert, können Bausteine nur über den Befehl **Einfügen > Textbaustein** abgerufen werden.

- **Registerkarten für Dokumente**

  Hier können Sie festlegen, ob *Registerkarten* für alle geöffneten Dokumentfenster unter der Funktionsleiste angezeigt werden sollen. (Siehe TextMaker-Handbuch, Stichwort "Registerkarten für Dokumente").

  Dokumentsymbol anzeigen: Ist diese Option aktiviert, erscheint in jeder Registerkarte links ein Symbol, das den Dateityp anzeigt.

  Schließen-Schaltfläche auch auf inaktiven Registerkarten anzeigen: Ist diese Option aktiviert, wird in jeder Registerkarte rechts ein **x**-Symbol angezeigt. Klicken Sie dieses an, wird das zugehörige Fenster geschlossen. Schalten Sie diese Option aus, wird das **x**-Symbol nur in der Registerkarte für das *aktive* Fenster angezeigt.

Karteikarte "Aussehen"

Hier können Sie Einstellungen vornehmen, die das Aussehen der Benutzeroberfläche von BasicMaker betreffen:

- **Dialogstil**

  Über die Option **Dialogstil** lässt sich das Aussehen der Dialogfenster und Symbolleisten von BasicMaker ändern. An der Bedienung des Programms ändert sich dabei nichts; verwenden Sie einfach die Einstellung, die Ihnen am besten gefällt.

- **Dialogsprache**

  Hier können Sie die Sprache wählen, in der Menüs und Dialogfenster angezeigt werden sollen. Nur anwendbar, wenn Sie bei der Installation mehrere Sprachen für die Benutzeroberfläche ausgewählt hatten (sofern verfügbar).

- **Schriftenliste mit echten Schriften**

  Ist diese Option aktiviert, zeigt BasicMaker in Schriftenlisten (zum Beispiel der Schriftenliste in den Einstellungen) alle Schriftnamen in der jeweiligen Schriftart an. So können Sie gleich sehen, wie die Schriften tatsächlich aussehen.

- **Quickinfos**

  Bestimmt, ob *Quickinfos* angezeigt werden sollen. Dabei handelt es sich um kurze Infotexte, die neben dem Mauszeiger angezeigt werden, wenn Sie mit der Maus auf ein Bildschirmelement zeigen.

- **Warnton bei Meldungen**

  Ist diese Option aktiviert, gibt der Editor bei Hinweis- und Fehlermeldungen einen Signalton aus.

- **Große Symbole verwenden**

  Wenn Sie diese Option aktivieren, werden in Symbolleisten und Menüs größere Symbole angezeigt.

Hinweis: Eine Änderung an dieser Einstellung wird erst wirksam, wenn Sie das Programm beenden und neu starten.

- **System-Dateidialoge verwenden**

  Diese Option bestimmt, welche Art von Dialogen bei Befehlen, die mit dem Öffnen und Speichern von Dateien zu tun haben, erscheinen sollen. Wählen Sie **Aus**, erscheinen BasicMakers eigene Dateidialoge. Wählen Sie **Ein**, erscheinen die Standard-Dateidialoge des Betriebssystems, wie Sie sie von den meisten anderen Applikationen her kennen.

- **Bildschirmschriftarten glätten**

  Wenn Sie diese Option aktivieren, verwendet BasicMaker eine Technologie, die die Kanten von Schriften auf dem Bildschirm glättet und so das Schriftbild verbessert – das sogenannte "Antialiasing".

Karteikarte "Dateien"

Hier können Sie Einstellungen zum Öffnen und Speichern von Dateien vornehmen:

- **.BAK-Dateien anlegen**

  Jedes Mal, wenn Sie ein Script speichern, legt der Editor auf Wunsch eine Sicherungskopie der letzten Fassung des Scripts mit der Namenserweiterung .BAK an. Wünschen Sie dies nicht, schalten Sie diese Option ab.

- **Automatisches Sichern alle ... Minuten**

  Macht auf Wunsch im eingetragenen Zeitabstand (1-100 Minuten) temporäre Sicherungskopien aller geöffneten Dateien.

  Wenn der Editor korrekt beendet wird, werden diese Kopien automatisch wieder gelöscht. Wurde er jedoch – zum Beispiel aufgrund eines Computerabsturzes – nicht korrekt beendet, erkennt er dies beim nächsten Programmstart. Er bietet Ihnen dann an, Sicherungskopien aller Dateien zu öffnen, die vor dem Absturz geändert wurden und noch nicht gespeichert waren.

  Überprüfen Sie dann bei jedem wiederhergestellten Script, welche der zuletzt gemachten Änderungen eventuell verloren gegangen sind, und speichern Sie es dann mit **Datei > Speichern**.

- **Dokumente in neuen Fenstern öffnen**

  Ist diese Option aktiviert, wird beim Anlegen oder Öffnen einer Datei mit **Datei > Neu** beziehungsweise **Datei > Öffnen** ein neues Fenster angelegt, ansonsten wird die Datei im gleichen Fenster geöffnet.

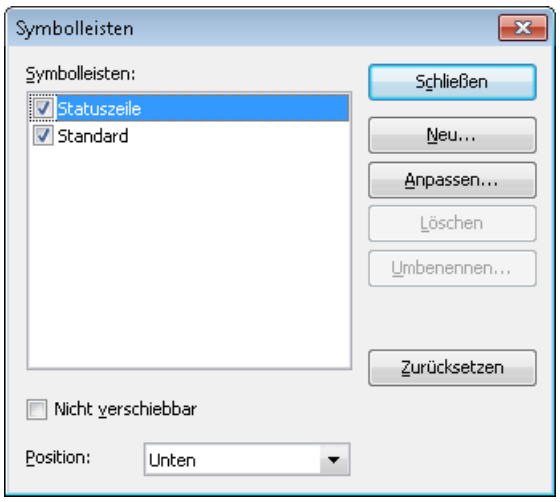
- **Einträge im Datei-Menü**

  Im Menü **Datei** wird eine Liste der zuletzt von Ihnen geöffneten Dateien angezeigt. Wählen Sie einen dieser Einträge, wird die entsprechende Datei sofort geöffnet. Hier lässt sich einstellen, wie viele Dateien dort angezeigt werden sollen.

Symbolleisten des Script-Editors anpassen

Symbolleisten wie zum Beispiel die Funktionsleiste ermöglichen blitzschnellen Zugriff auf die Funktionen des Editors. Jedes Symbol steht für einen bestimmten Befehl. Klicken Sie darauf, wird der Befehl aufgerufen.

Mit dem Befehl **Ansicht > Symbolleisten** können Sie Symbolleisten konfigurieren.



Rufen Sie den Befehl dazu auf und gehen Sie dann wie folgt vor:

- **Symbolleisten ein- und ausschalten**

  Klicken Sie auf das Kästchen vor der gewünschten Symbolleiste, um diese ein- beziehungsweise auszuschalten.

- **Symbolleisten positionieren**

  Um eine Symbolleiste zu positionieren, selektieren Sie diese erst in der Liste **Symbolleisten** und wählen dann bei **Position** die gewünschte Position: **Oben**, **Unten**, **Links**, **Rechts** oder **Frei** (in einem frei beweglichen Fenster).

  Tipp: Sie können Symbolleisten auch einfach mit der Maus packen und sie dann an die gewünschte Position ziehen.

  Verschieben mit der Maus verhindern: Wenn Sie verhindern möchten, dass eine Symbolleiste versehentlich mit der Maus verschoben werden kann, selektieren Sie die entsprechende Leiste und schalten die Option **Nicht verschiebbar** ein.

- **Neue Symbolleiste erstellen**

  Um eine neue Symbolleiste zu erstellen, klicken Sie auf die Schaltfläche **Neu** und tippen dann einen Namen für die Leiste ein (zum Beispiel "Meine Symbolleiste").

  Um zu erfahren, wie Sie die neue Symbolleiste mit Symbolen bestücken, lesen Sie bitte den nächsten Abschnitt.

- **Symbole einer Symbolleiste bearbeiten**

  Mit der Schaltfläche **Anpassen** können Sie die Symbole auf Symbolleisten bearbeiten. Lesen Sie dazu den nächsten Abschnitt.

- **Symbolleisten löschen**

  Um eine Symbolleiste zu löschen, selektieren Sie diese und klicken auf die Schaltfläche **Löschen**.

  Hinweis: Sie können nur selbst erstellte Symbolleisten löschen.

- **Symbolleisten umbenennen**

  Um eine Symbolleiste umzubenennen, selektieren Sie diese, klicken auf die Schaltfläche **Umbenennen** und geben den neuen Namen ein.

  Hinweis: Sie können nur selbst erstellte Symbolleisten umbenennen.

- **Symbolleisten zurücksetzen**

  Mit der Schaltfläche **Zurücksetzen** können Sie alle Änderungen an einer der Standard-Symbolleisten rückgängig machen.

  Hinweis: Diese Schaltfläche ist nur auf die Funktionsleiste und die Statuszeile anwendbar, nicht jedoch auf selbst erstellte Symbolleisten.

Symbole einer Symbolleiste bearbeiten

Sie können in Symbolleisten jederzeit Symbole hinzufügen, entfernen und verschieben sowie Leerräume einfügen und entfernen.

Dazu gehen Sie folgendermaßen vor:

1. Stellen Sie sicher, dass die betreffende Symbolleiste eingeschaltet ist. Ist dies nicht der Fall, rufen Sie **Ansicht > Symbolleisten** auf und schalten die Symbolleiste ein.
2. Rufen Sie den Befehl **Weiteres > Anpassen > Symbolleisten** auf. Alternativ können Sie diesen Befehl auch durch Betätigen der Schaltfläche **Anpassen** im Dialogfenster von **Ansicht > Symbolleisten** aufrufen.
3. Verwenden Sie eines der unten beschriebenen Verfahren, um Symbole hinzuzufügen, zu entfernen, zu verschieben etc.
4. Verlassen Sie den Dialog mit **Schließen**.

Beim Aufruf des Befehls **Weiteres > Anpassen > Symbolleisten** erscheint ein Dialogfenster, in dem alle Symbole aufgelistet sind, die Sie einer Symbolleiste hinzufügen können.

Die Symbole für die Befehle des Editors sind der Übersichtlichkeit halber auf Kategorien verteilt. Wählen Sie eine Kategorie in der Liste **Gruppe**, werden in der Liste **Befehl** alle verfügbaren Symbole dieser Kategorie angezeigt.

Das Bearbeiten der Symbole einer Symbolleiste funktioniert folgendermaßen:

- **Hinzufügen eines Symbols**

  Ziehen Sie das Symbol mit der Maus einfach direkt aus der Liste **Befehl** an die gewünschte Position in der Symbolleiste.

- **Löschen eines Symbols**

  Ziehen Sie das Symbol aus der Symbolleiste heraus (zum Beispiel in den Text), worauf es gelöscht wird.

- **Verschieben eines Symbols**

  Ziehen Sie das Symbol in der Symbolleiste an die gewünschte Position. Wenn Sie ein Symbol zwischen zwei andere Symbole ziehen, wird es dort eingefügt.

  Sie können auch ein Symbol von einer Symbolleiste in eine andere Symbolleiste verschieben.

- **Einfügen eines Leerraums**

  Ziehen Sie ein Symbol einige Millimeter nach rechts, wird links davon ein Leerraum eingefügt.

- **Entfernen eines Leerraums**

  Ziehen Sie das Symbol, das sich rechts vom Leerraum befindet, an das Symbol links davon, wird der Leerraum entfernt.

Tastaturbelegung des Script-Editors anpassen

Die gebräuchlichsten Befehle des Script-Editors lassen sich auch über "Tastenkürzel" aufrufen. So können Sie beispielsweise den Befehl **Datei > Speichern** blitzschnell mit der Tastenkombination Strg S aufrufen.

Mit dem Befehl **Weiteres > Anpassen > Tastaturbelegung** lassen sich diese Tastenkürzel nach Belieben anpassen:

Tastaturbelegung aktivieren

Folgendermaßen bestimmen Sie, welche Tastaturbelegung aktiv sein soll:

1. Rufen Sie den Befehl **Weiteres > Anpassen > Tastaturbelegung** auf.
2. Selektieren Sie die gewünschte Tastaturbelegung.
3. Klicken Sie auf **Anwenden**, um sie zu aktivieren.

Nun stehen alle in der Tastaturbelegung definierten Tastenkürzel zur Verfügung.

Standardmäßig sind die folgenden zwei Tastaturbelegungen vorgegeben:

- **Standard** (die Standardbelegung)
- **Classic** (eine weitgehend Wordstar-kompatible Tastaturbelegung)

Sie können diese zwei Standardbelegungen allerdings jederzeit abändern und auch eigene Tastaturbelegungen erstellen.

Neue Tastaturbelegung erstellen

Mit dem Befehl **Weiteres > Anpassen > Tastaturbelegung** lassen sich komplette *Tastaturbelegungen* zusammenstellen. So können Sie sich mehrere Tastaturbelegungen für unterschiedliche Einsatzzwecke anlegen und bei Bedarf zwischen diesen wechseln.

Hinweis: Wenn Sie nur einige Tastenkürzel hinzufügen oder ändern möchten, ist es nicht erforderlich, dafür extra eine eigene Tastaturbelegung anzulegen. Klicken Sie einfach auf die Schaltfläche **Bearbeiten**, und ändern Sie direkt die Standard-Tastaturbelegung.

Folgendermaßen erstellen Sie eine neue Tastaturbelegung:

1. Rufen Sie den Befehl **Weiteres > Anpassen > Tastaturbelegung** auf.
2. Wählen Sie die Tastaturbelegung aus, auf der die neue Tastaturbelegung basieren soll (in der Regel Standard).

   Hinweis: Die neue Tastaturbelegung übernimmt automatisch alle Tastenkürzel der Tastaturbelegung, die Sie hier auswählen.
3. Betätigen Sie die Schaltfläche **Neu**.
4. Es erscheint ein Dialogfenster, in das Sie einen Namen für die neue Tastaturbelegung eingeben – zum Beispiel "Meine Tastaturbelegung". Bestätigen Sie dann mit **OK**.

Die neue Tastaturbelegung wird nun angelegt. Es erscheint automatisch ein Dialogfenster zum Ändern der Tastenkürzel. Mehr dazu im nächsten Abschnitt.

Tastaturbelegung löschen

Selbst erstellte Tastaturbelegungen lassen sich wie folgt löschen:

1. Rufen Sie den Befehl **Weiteres > Anpassen > Tastaturbelegung** auf.
2. Selektieren Sie die zu löschende Tastaturbelegung.
3. Betätigen Sie die Schaltfläche **Löschen**.

Hinweis: Sie können nur selbst erstellte Tastaturbelegungen löschen.

Tastaturbelegung umbenennen

Selbst erstellte Tastaturbelegungen können Sie wie folgt umbenennen:

1. Rufen Sie den Befehl **Weiteres > Anpassen > Tastaturbelegung** auf.
2. Selektieren Sie die gewünschte Tastaturbelegung per Mausklick.
3. Betätigen Sie die Schaltfläche **Umbenennen**.
4. Tippen Sie den neuen Namen ein und bestätigen Sie mit **OK**.

Hinweis: Sie können nur selbst erstellte Tastaturbelegungen umbenennen.

Tastenkürzel einer Tastaturbelegung anpassen

Mit der Schaltfläche **Bearbeiten** können Sie die Tastenkürzel einer Tastaturbelegung anpassen. Informationen hierzu finden Sie im nächsten Abschnitt.

Tastenkürzel einer Tastaturbelegung bearbeiten

Mit dem Befehl **Weiteres > Anpassen > Tastaturbelegung** lassen sich Tastaturbelegungen nicht nur anlegen und verwalten. Die wichtigste Funktion dieses Befehls ist vielmehr das Ändern der enthaltenen Tastenkürzel. Hierzu dient die Schaltfläche **Bearbeiten**.

Weisen wir als Beispiel dem Befehl **Bearbeiten > Alles Markieren** das Kürzel F6 zu. Gehen Sie dazu wie folgt vor:

1. Rufen Sie den Befehl **Weiteres > Anpassen > Tastaturbelegung** auf.
2. Klicken Sie auf die Schaltfläche **Bearbeiten**.
3. Wählen Sie in der Liste **Gruppe** eine Befehlskategorie. Wählen Sie dann in der Liste **Befehl** den Befehl, dessen Tastenkürzel Sie verändern möchten.

   In unserem Beispiel wäre also bei **Gruppe** "Bearbeiten" und bei **Befehl** "Alles markieren" zu wählen.
4. Klicken Sie in das Eingabefeld **Bitte Tastenkürzel drücken** und betätigen Sie das gewünschte Tastenkürzel – hier also F6. Verwenden Sie die Rücktaste ←, falls Sie sich vertippt haben.
5. ***Nicht vergessen:*** Klicken Sie auf **Hinzufügen**, um dem Befehl dieses Tastenkürzel zuzuweisen.
6. Bestätigen Sie mit **OK** und verlassen Sie den Hauptdialog mit **Schließen**.

Zukünftig können Sie den Befehl **Bearbeiten > Alles markieren** mit der Funktionstaste F6 aufrufen.

Verfügbare Tastenkürzel

Beachten Sie, dass nicht alle Tastenkombinationen für Tastenkürzel erlaubt sind.

In der Regel sollten Sie für Tastenkürzel **Buchstabentasten**, **Zahlentasten** oder **Funktionstasten** verwenden. Diese können Sie mit den Tasten Strg, Alt und der Umschalttaste ⇧ kombinieren.

Sie können ganz einfach überprüfen, ob die von Ihnen gewünschte Tastenkombination zulässig ist: Betätigen Sie die Tastenkombination im Feld **Bitte Tastenkürzel drücken**. Wenn sie nicht erscheint, ist sie nicht zulässig.

Einige Beispiele für gültige Tastenkürzel:

- Strg A
- Alt A (Tastenkombinationen mit der Alt-Taste sind allerdings nicht zu empfehlen – Alt A ist beispielsweise für den Aufruf des Menüs **Ansicht** zuständig!)
- Strg Alt A
- Strg ⇧ A
- Strg Alt ⇧ A
- Strg F5
- etc.

Hinweis: Buchstaben allein sind *nicht* zulässig. Sie können also A oder ⇧ A nicht als Tastenkürzel verwenden.

Tastenkürzel bereits belegt: Wenn Sie ein Tastenkürzel drücken, das bereits belegt ist, wird unter dem Eingabefeld angezeigt, womit dieses Kürzel momentan belegt ist. Sie sollten dann die Rücktaste ← betätigen, um das Tastenkürzel wieder zu löschen und ein anderes Tastenkürzel verwenden. Andernfalls überschreiben Sie die bisherige Zuordnung dieses Kürzels.

Zweiteilige Tastenkürzel: Als Überbleibsel aus der WordStar-Zeit können Sie auch zweiteilige Tastenkürzel verwenden – zum Beispiel Strg X Y. Hierfür sind allerdings nur Tastenkürzel nach dem Schema "Strg + Buchstabe + Buchstabe" zulässig.

Ein Tastenkürzel wieder entfernen

Wenn Sie einem Befehl ein Tastenkürzel zugeordnet haben, können Sie dies jederzeit wieder rückgängig machen, indem Sie diese Zuordnung entfernen.

Dazu gehen Sie wie folgt vor:

1. Rufen Sie, wie oben beschrieben, den Dialog **Tastaturbelegung bearbeiten** auf.
2. Wählen Sie in der Liste **Gruppe** eine Befehlsgruppe und dann in der Liste **Befehl** den gewünschten Befehl.
3. Es erscheinen nun bei **Aktuelle Tastenkürzel** alle diesem Befehl zugewiesenen Kürzel. Selektieren Sie das zu entfernende Tastenkürzel und klicken Sie die Schaltfläche **Entfernen**.
4. Bestätigen Sie mit **OK** und verlassen Sie den Hauptdialog mit **Schließen**.

Das Tastenkürzel wurde nun entfernt – der Befehl kann zukünftig nicht mehr über dieses Kürzel aufgerufen werden.

Tastenkürzel einer Tastaturbelegung zurücksetzen

Klicken Sie im Dialogfenster von **Weiteres > Anpassen > Tastaturbelegung** auf die Schaltfläche **Zurücksetzen**, werden alle Tastenkürzel der aktuellen Tastaturbelegung wieder auf die Standardbelegung zurückgesetzt.

Hinweis: Dadurch gehen *alle* Änderungen, die Sie an den Tastenkürzeln dieser Tastaturbelegung vorgenommen haben, verloren.

Hinweis: Dies ist nur bei den vorgegebenen Tastaturbelegungen **Standard** und **Classic** möglich.

Befehle im Fenster-Menü des Script-Editors

Im Menü **Fenster** des Script-Editors stehen folgende Befehle zur Verfügung:

- **Fenster > Überlappend**

  Ordnet alle derzeit offenen Fenster hintereinander an (wie in einem Karteikasten).

- **Fenster > Nebeneinander**

  Ordnet die Fenster nebeneinander an.

- **Fenster > Untereinander**

  Ordnet die Fenster untereinander an.

- **Fenster > Symbole anordnen**

  Ordnet die Symbole aller minimierten Fenster in der linken unteren Ecke des Arbeitsfensters an.

- **Fenster > Alle schließen**

  Schließt alle derzeit geöffneten Fenster.

- **Fenster > Registerkarten**

  Bestimmt, ob unter der Funktionsleiste eine Leiste mit *Registerkarten* für alle derzeit geöffneten Dokumentfenster angezeigt werden soll. (Siehe auch TextMaker-Handbuch, Stichwort "Registerkarten für Dokumente".)

- **Fensterliste** im Menü **Fenster**

  Listet alle derzeit offenen Fenster auf. Klicken Sie einen Eintrag an, kommt das entsprechende Fenster in den Vordergrund.

Starten eines Scripts

Basic-Scripts können von BasicMaker, TextMaker oder von PlanMaker aus gestartet werden:

- **Starten von BasicMaker aus**

  Um ein Script auszuführen, rufen Sie in BasicMaker den Befehl **Programm > Starten** auf oder betätigen die Taste **F9**.

- **Starten von TextMaker/PlanMaker aus**

  Sie können ein Script auch von TextMaker oder PlanMaker aus starten. Rufen Sie dazu in TextMaker/PlanMaker den Befehl **Weiteres > Script starten** auf. Es erscheint ein Dialogfenster, in dem Sie das gewünschte Script auswählen. Wenn Sie mit **OK** bestätigen, wird BasicMaker gestartet, führt das Script aus und beendet sich danach wieder.

  Das Gleiche passiert, wenn Sie BasicMaker mit **basicmaker /s scriptname.bas** aufrufen: BasicMaker startet, führt das Script mit dem übergebenen Namen aus und beendet sich wieder.

Abbrechen eines Scripts

Ein laufendes Script kann jederzeit durch Betätigen der Tastenkombination Strg Untbr abgebrochen werden. (Falls eine andere Applikation im Vordergrund ist, müssen Sie dazu erst zum BasicMaker-Programmfenster wechseln.)

Debuggen eines Scripts

Der Script-Editor verfügt über Befehle, die Ihnen das Auffinden und Beseitigen von Fehlern (das "Debuggen") erleichtern:

- Script in Einzelschritten ausführen
- Haltepunkte verwenden
- Variablen beobachten

Script in Einzelschritten ausführen

Mit folgenden Befehlen können Sie ein Script in Einzelschritten ausführen:

Programm > Einzelschritt (Tastenkürzel: F7)

Wenn Sie den Befehl **Programm > Einzelschritt** aufrufen, wird nur eine einzelne Zeile des Scripts ausgeführt und die Ausführung dann angehalten. Wenn Sie den Befehl erneut aufrufen, wird die nächste Zeile abgerufen, dann wieder pausiert – usw.

So können Sie ein Script Zeile für Zeile in Einzelschritten ausführen.

Programm > Prozedurschritt (Tastenkürzel: F8)

Beim Befehl **Programm > Prozedurschritt** wird ebenfalls nur eine Zeile des Scripts ausgeführt und die Ausführung dann angehalten.

Der Unterschied zum Einzelschritt: Prozeduren werden nicht zeilenweise, sondern an einem Stück abgearbeitet.

Erläuterung: Wenn in Ihrem Quelltext eine Unterprozedur (also eine Function oder eine Sub) aufgerufen wird, springt der Befehl **Einzelschritt** in diese Prozedur hinein, arbeitet die erste Anweisung ab und wartet dann. Der Befehl **Prozedurschritt** behandelt Prozeduren hingegen wie eine Einzelanweisung: er arbeitet die komplette Prozedur auf einen Schlag ab.

Programm > Zurücksetzen (Tastenkürzel: Strg+F2)

Der Befehl **Programm > Zurücksetzen** bricht die Einzelschritt-Ausführung ab und setzt das Script auf die erste Zeile zurück.

Haltepunkte verwenden

Wenn Sie in einer Zeile des Scripts einen Haltepunkt setzen und das Script dann ausführen, wird die Ausführung in dieser Zeile angehalten.

Um die Ausführung anschließend wieder fortzusetzen, können Sie **Programm > Starten** beziehungsweise **Programm > Einzelschritt** oder **Programm > Prozedurschritt** aufrufen.

Für Haltepunkte gibt es folgende Befehle:

Programm > Haltepunkt setzen/löschen (Tastenkürzel: F2)

Setzt in der aktuellen Zeile einen Haltepunkt oder entfernt ihn wieder.

Programm > Alle Haltepunkte löschen (Tastenkürzel: Alt+F2)

Löscht alle gesetzten Haltepunkte.

Variablen beobachten

Mit Hilfe des *Variablenfensters* können Sie sich den Inhalt von Variablen während der Ausführung eines Scripts anzeigen lassen. Dies ist natürlich besonders beim Testen eines Scripts in Einzelschritten nützlich.

Um eine Variable zu beobachten, gehen Sie wie folgt vor:

1. Falls das Variablenfenster derzeit nicht sichtbar ist, aktivieren Sie es mit **Ansicht > Variablenfenster**.
2. Klicken Sie im Script auf den Namen der gewünschten Variable. Öffnen Sie dann mit einem Rechtsklick das Kontextmenü und wählen Sie darin den Befehl **Variable zeigen** aus.

   Tipp: Sie können auch einfach den Namen der Variable von Hand in eine leere Zeile des Variablenfensters eintragen.
3. Starten Sie das Script nun mit **Programm > Starten** beziehungsweise **Programm > Einzelschritt** oder **Programm > Prozedurschritt**.

Der Inhalt der Variable wird nun im Variablenfenster angezeigt und laufend aktualisiert.

Dialogeditor verwenden

In diesem Abschnitt wird die Bedienung des in BasicMaker enthaltenen Dialogeditors erläutert:

- Allgemeines
- Dialogeditor aufrufen/beenden
- Befehle im Datei-Menü des Dialogeditors
- Befehle im Bearbeiten-Menü des Dialogeditors
- Befehle im Einfügen-Menü des Dialogeditors

Allgemeines

Sie können in SoftMaker Basic eigene Dialogfenster erstellen, um einem Script Interaktionen mit dem Anwender zu erlauben.

Um ein Dialogfenster herzustellen, müssen Sie einen Dialog definieren. Sie können die dazu erforderliche *Dialogdefinition* entweder manuell in das Script eintragen (siehe Abschnitt "Dialogdefinition") oder wahlweise den integrierten Dialogeditor (siehe nachfolgende Abschnitte) verwenden.

Der Dialogeditor lässt Sie Dialogfenster grafisch gestalten. Sie können neue Dialogelemente über die Funktionsleiste oder die Befehle im Menü **Einfügen** einfügen. Bestehende Elemente lassen sich wie in einem Zeichenprogramm durch Ziehen mit der Maus vergrößern, verkleinern und verschieben; ihre Eigenschaften können über das Menü **Bearbeiten** editiert werden.

Dialogeditor aufrufen/beenden

Der Dialogeditor kann mit dem Befehl **Bearbeiten > Dialoge bearbeiten** aufgerufen werden.

Erstellen eines neuen Dialogs

Um mit Hilfe des Dialogeditors ein *neues* Dialogfenster zu erstellen, sind folgende Schritte nötig:

1. Setzen Sie die Schreibmarke an die Position im Quelltext, an der die Dialogdefinition (**BeginDialog** ... **EndDialog**) eingefügt werden soll.
2. Rufen Sie den Befehl **Bearbeiten > Dialoge bearbeiten** auf.
3. Klicken Sie auf die Schaltfläche **Neu**.
4. Der Dialogeditor wird nun gestartet und Sie können den Dialog darin entwerfen (Informationen zur Bedienung des Dialogeditors finden Sie in den nächsten Abschnitten).
5. Wenn der Dialog fertiggestellt ist, beenden Sie den Dialogeditor mit **Datei > Beenden**.
6. Verlassen Sie das Dialogfenster mit **Schließen**.

Die Dialogdefinition wird nun in den Quelltext eingefügt.

Bearbeiten eines vorhandenen Dialogs

Um eine bereits *vorhandene* Dialogdefinition zu bearbeiten, sind folgende Schritte nötig:

1. Rufen Sie den Befehl **Bearbeiten > Dialoge bearbeiten** auf.
2. Wählen Sie den zu bearbeitenden Dialog in der Liste **Dialogname**.
3. Klicken Sie auf die Schaltfläche **Bearbeiten**.
4. Der Dialogeditor wird nun gestartet und Sie können den Dialog darin bearbeiten.
5. Wenn alle Änderungen vorgenommen wurden, beenden Sie den Dialogeditor mit **Datei > Beenden**.
6. Verlassen Sie das Dialogfenster mit **Schließen**.

Die Dialogdefinition wird nun im Quelltext entsprechend verändert.

Löschen eines vorhandenen Dialogs

Um eine Dialogdefinition zu löschen, entfernen Sie diese von Hand aus dem Quelltext oder rufen **Bearbeiten > Dialoge bearbeiten** auf, selektieren den Dialog in der Liste **Dialogname** und betätigen dann die Schaltfläche **Löschen**.

Befehle im Datei-Menü des Dialogeditors

- **Datei > Dialog zurücksetzen**

  Macht sämtliche Änderungen am Dialog rückgängig.

- **Datei > Abbrechen**

  Beendet den Dialogeditor – *ohne* Ihre Änderungen zu speichern.

- **Datei > Beenden**

  Speichert Ihre Änderungen und beendet den Dialogeditor.

Befehle im Bearbeiten-Menü des Dialogeditors

Die Befehle des Menüs **Bearbeiten** dienen zum Bearbeiten bereits vorhandener Dialogelemente.

Bevor Sie einen dieser Befehle aufrufen, müssen Sie in der Regel erst das Dialogelement selektieren, auf das sich der Befehl auswirken soll. Klicken Sie es dazu mit der Maus an. Wenn mehrere Objekte selektiert werden sollen, klicken Sie diese nacheinander bei gedrückter **Umschalttaste** an oder ziehen bei gedrückter Maustaste einen Rahmen um diese Objekte.

- **Bearbeiten > Ausschneiden**

  Schneidet Dialogelemente in die Zwischenablage aus.

- **Bearbeiten > Kopieren**

  Kopiert Dialogelemente in die Zwischenablage.

- **Bearbeiten > Einfügen**

  Fügt den Inhalt der Zwischenablage ein.

- **Bearbeiten > Löschen**

  Löscht Dialogelemente.

- **Bearbeiten > Alles löschen**

  Leert das komplette Dialogfenster.

- **Bearbeiten > Auf Gitter einpassen**

  Richtet Dialogelemente am Gitter aus. Die Auflösung dieses Gitters lässt sich mit dem Befehl **Bearbeiten > Gitter** einstellen.

- **Bearbeiten > Nach vorne**

  Für den Fall, dass sich Dialogelemente überlappen, können Sie mit diesem Befehl ein Element in den Vordergrund bringen.

- **Bearbeiten > Nach hinten**

  Für den Fall, dass sich Dialogelemente überlappen, können Sie mit diesem Befehl ein Element in den Hintergrund stellen.

- **Bearbeiten > Ausrichtung**

  Lässt Sie die Ausrichtung der derzeit markierten Dialogelemente ändern:

  Keine Änderung: Ausrichtung wird nicht verändert.

  Linke Seiten: Die Dialogelemente werden an der linken Kante des am weitesten links befindlichen markierten Elements ausgerichtet.

  Zentriert: Die Dialogelemente werden untereinander zentriert.

Rechte Seiten: Die Dialogelemente werden an der rechten Kante des am weitesten rechts befindlichen Elements ausgerichtet.

Gleicher Abstand: Die Elemente werden zwischen dem am weitesten links und dem am weitesten rechts befindlichen Element gleichmäßig verteilt.

Zentrieren im Fenster: Die Dialogelemente werden innerhalb des Dialogfensters zentriert.

Die Optionen in der Rubrik **Vertikal** funktionieren entsprechend.

- **Bearbeiten > Größe**

  Lässt Sie die Größe der derzeit markierten Dialogelemente ändern:

  Keine Änderung: Es wird keine Änderung vorgenommen.

  Minimale Breite: Die Breite wird der des schmalsten markierten Elements angepasst.

  Maximale Breite: Die Breite wird der des breitesten markierten Elements angepasst.

  Breite: Lässt Sie die Breite auf einen festen Wert (in Pixeln) setzen.

  Änderungen an der **Höhe** funktionieren entsprechend.

- **Bearbeiten > Gitter**

  Lässt Sie das Gitter konfigurieren. Das Gitter ist eine Positionierungshilfe. Wenn es aktiviert ist, lassen sich Dialogelemente nicht mehr frei verschieben, sondern nur von Gitterpunkt zu Gitterpunkt bewegen.

  Gitter anzeigen: Bestimmt, ob die Gitterpunkte im Dialogfenster angezeigt werden sollen.

  Auf Gitter springen: Bestimmt, ob das Gitter aktiviert sein soll.

  X- und **Y-Inkrement**: Bestimmt den Abstand der Gitterpunkte.

  Mit dem Befehl **Bearbeiten > Auf Gitter einpassen** können Sie die Elemente eines Dialogs auch nachträglich auf das Gitter einpassen.

Befehle im Einfügen-Menü des Dialogeditors

Mit Hilfe der Befehle des Menüs **Einfügen** können Sie einem Dialogfenster neue Elemente hinzufügen. Alternativ können Sie dazu die Symbole auf der Funktionsleiste oder die Funktionstasten **F2** bis **F10** verwenden:

Dialogelement	Symbol	Taste
OK-Schaltfläche		F2
Abbrechen-Schaltfläche		F3
Schaltfläche		F4
Optionsfeld		F5
Kontrollkästchen		F6
Text		F7
Eingabefeld		F8
Gruppenfeld		F9
Listenfeld		F10

Kombinationsfeld 

Dropdown-Liste 

Wählen Sie zunächst aus, welche Art von Dialogelement einfügt werden soll. Ziehen Sie dann im Dialogfenster einen Rahmen auf, um Größe und Position des Elements festzulegen.

Eine Beschreibung aller Elemente, die Sie in Dialogfenster einfügen können, finden Sie im Abschnitt "Steuerelemente eines Dialogfensters".

Sprachelemente von SoftMaker Basic

In diesem Kapitel finden Sie grundlegende Informationen zu den Script-Befehlen von BasicMaker:

- Grundlegendes zur Syntax
- Datentypen
- Variablen
- Arrays
- Operatoren
- Kontrollstrukturen
- Unterprogramme und Funktionen
- Aufruf von Funktionen in DLLs
- Dateioperationen
- Dialogfenster
- OLE Automation

Grundlegendes zur Syntax

Kommentare

Text, dem das Schlüsselwort **Rem** oder ein Apostrophzeichen (') vorausgeht, wird als *Kommentar* angesehen und nicht ausgeführt. Sie können Kommentare zur Dokumentation Ihrer Scripts verwenden.

```
'Dies ist ein Kommentar
rem Dies auch
REM Dies auch
Rem Dies auch
```

Wie Sie sehen, spielt die Groß-/Kleinschreibung der **Rem**-Anweisung keine Rolle. Dies ist bei allen Schlüsselwörtern von SoftMaker Basic der Fall.

Kommentare können auch am Ende einer Zeile angebracht werden:

```
MsgBox Msg   ' Zeige Meldung an
```

Der Text hinter dem Apostroph ist ein Kommentar.

Mehrere Anweisungen in einer Zeile

Sie können mehrere Anweisungen in die gleiche Zeile schreiben, indem Sie diese durch einen Doppelpunkt trennen:

```
X.AddPoint(25,100) :  Y.AddPoint(0,75)
```

... ist gleichbedeutend mit ...

```
X.AddPoint(25,100)
Y.AddPoint(0,75)
```

Anweisungen über mehrere Zeilen

Sie können eine Anweisung auf mehrere Zeilen aufteilen, indem Sie am Ende jeder Zeile ein Leerzeichen und einen Unterstrich (_) eingeben.

```
X.AddPoint _
(25,100)
```

... ist gleichbedeutend mit ...

```
X.AddPoint(25,100)
```

Zahlen

Sie können Zahlen auf drei Arten schreiben: dezimal, oktal und hexadezimal:

- **Dezimalzahlen:** Bei den meisten Zahlen in den Programmbeispielen dieses Handbuch handelt es sich um Dezimalzahlen (Basis 10).
- **Oktalzahlen:** Möchten Sie Oktalzahlen (Basis 8) verwenden, stellen Sie der Zahl **&O** voran – zum Beispiel &O37.
- **Hexadezimale Zahlen:** Hexadezimalen Zahlen (Basis 16) ist **&H** voranzustellen – zum Beispiel &H1F.

Namen

Variablen, Konstanten, Unterprogramme und Funktionen werden über ihren Namen angesprochen. Für diese Namen gelten folgende Regeln:

- Nur die Buchstaben A-Z und a-z, der Unterstrich (_) und die Ziffern 0-9 sind erlaubt.
- Groß-/Kleinschreibung spielt keine Rolle.
- Das erste Zeichen des Namens muss ein Buchstabe sein.
- Die Länge darf nicht mehr als 40 Zeichen betragen.
- Schlüsselwörter von SoftMaker Basic dürfen nicht verwendet werden.

Datentypen

Es gibt folgende Datentypen:

Typ	Suffix	Syntax der Deklaration	Größe
String	$	Dim <*Name*> As String	0 bis 65,500 Zeichen
String*n		Dim <*Name*> As String*n	genau *n* Zeichen
Integer	%	Dim <*Name*> As Integer	2 Bytes
Long	&	Dim <*Name*> As Long	4 Bytes
Single	!	Dim <*Name*> As Single	4 Bytes
Double	#	Dim <*Name*> As Double	8 Bytes
Variant		Dim <*Name*> As Variant	je nach Inhalt
		Oder: Dim <*Name*> As Any	
		Oder einfach nur: Dim <*Name*>	
Object		(siehe Abschnitt "OLE Automation")	
Benutzerdefiniert		(siehe Abschnitt "Benutzerdefinierte Datentypen")	

Informationen zur Verwendung von Variablen finden Sie im Abschnitt "Variablen".

Besonderheiten beim Variant-Datentyp

Bei SoftMaker Basic muss eine Variable nicht unbedingt deklariert zu werden, bevor sie zum ersten Mal verwendet wird (Ausnahme: wenn **Option Explicit** gesetzt wurde). SoftMaker Basic deklariert sie dann bei ihrem ersten Auftreten automatisch – und zwar als Datentyp **Variant**.

Der Datentyp **Variant** kann benutzt werden, um *wahlweise* Zahlen, Zeichenketten oder Datum/Uhrzeit-Werte zu speichern. Typumwandlungen geschehen dabei automatisch.

Sie können Variablen auch explizit als Variant deklarieren, zum Beispiel mit `Dim x As Variant` oder einfach nur `Dim x` ohne Typangabe.

Ein Beispiel für die Verwendung von Variant-Variablen:

```
Sub Main
    Dim x         'Variant-Variable
    x = 10
    x = x + 8
    x = "F" & x
    print x       'Ergebnis: "F18"
End Sub
```

Wenn Zahlen in einer Variant-Variablen gespeichert werden, wird automatisch der kompakteste Datentyp gewählt, mit dem die Zahl dargestellt werden kann (in der Reihenfolge **Integer**, **Long**, **Single**, **Double**).

Der von einer Variant-Variablen beherbergte Datentyp kann jederzeit wechseln. Um den aktuellen Datentyp zu ermitteln, können Sie die Funktion **VarType** verwenden. Mit der Funktion **IsNumeric** können Sie prüfen, ob die Variable momentan einen numerischen Wert enthält.

Variant-Variablen können zwei spezielle Werte annehmen, die es bei anderen Datentypen nicht gibt:

- **Empty** ist der Wert einer Variant-Variablen, die noch nicht initialisiert wurde. Er kann mit der Funktion **IsEmpty** abgefragt werden. Bei numerischen Operationen wird **Empty** als 0, bei Zeichenkettenoperationen als eine leere Zeichenkette interpretiert.
- Der Wert **Null** dient zur Darstellung der Tatsache, dass kein (gültiger) Wert verfügbar ist. Er kann mit der Funktion **IsNull** abgefragt werden. Jede Operation mit einem **Null**-Wert ergibt **Null**.

Verkettung von Variant-Variablen

Wenn eine Zeichenkette und eine Zahl mit dem Operator **+** verkettet werden, resultiert daraus eine Zeichenkette.

Verketten Sie zwei Zahlen, ist das Ergebnis hingegen eine Zahl. Möchten Sie stattdessen eine Zeichenkette erhalten, ist statt **+** der Operator **&** zu verwenden, der – unabhängig vom Datentyp der Argumente – stets eine Zeichenkette liefert.

Benutzerdefinierte Datentypen

Mit Hilfe der Anweisung **Type** können Sie eigene Datentypen definieren. Dies muss vor der Deklaration von Prozeduren geschehen – benutzerdefinierte *Datentypen* sind nämlich stets global gültig. Die *Variablen* eines benutzerdefinierten Typs können hingegen lokal oder global deklariert werden.

Hinweis: Die Verwendung von Arrays in benutzerdefinierten Typen ist nicht zulässig. Weiterhin können Variablen benutzerdefinierten Typs nicht an DLLs übergeben werden, die C-Strukturen erwarten.

```
Type Person
   Name As String
   Vorname As String
   Geschlecht As String*1    ' ("m" oder "w")
   Geburtsdatum As String
End Type
```

Variablen dieses Typs können wie andere Variablen mit **Dim** oder **Static** angelegt werden. Auf die einzelnen Elemente kann mit der Punktnotation *Variable.Element* zugegriffen werden (siehe auch **With**-Anweisung).

```
Dim p As Person
p.Name = "Maier"
```

Variablen

Deklarieren von Variablen

Variablen werden mit den Anweisungen **Dim** oder **Static** angelegt. Standardmäßig haben Variablen den Typ Variant. Wird ein anderer Datentyp gewünscht, müssen Sie diesen bei der Deklaration mit **As** *Typ* oder mittels Typsuffix (z.B. % für **Integer**) angeben (siehe Abschnitt "Datentypen").

```
Dim X               ' X als Variant-Variable deklarieren

Dim X As Integer    ' X als Integer-Variable deklarieren

Dim X%              ' Gleichbedeutend mit obiger Anweisung

Dim X%, Name$       ' Mehrere Deklarationen in einer Zeile
```

Geltungsbereich von Variablen

Variablen können entweder lokal oder global gelten:

- Globale Variablen werden mit einer **Dim**-Anweisung *außerhalb* einer Prozedur angelegt. Auf sie kann überall zugegriffen werden.
- Lokale Variablen werden mit einer **Dim**- oder **Static**-Anweisung *innerhalb* einer Prozedur (Unterprogramm oder Funktion) angelegt. Sie sind nur innerhalb der Prozedur verfügbar.

Arrays

SoftMaker Basic unterstützt ein- und mehrdimensionale *Arrays (Felder)*. In solchen Arrays können Felder von Werten unter einem einheitlichen Namen abgelegt werden. Auf die enthaltenen Werte kann über einen Index zugegriffen werden.

Alle Elemente in einem Array haben den gleichen Datentyp. Als Datentypen sind zulässig: **Integer**, **Long**, **Single**, **Double** oder **String**.

Hinweis: In anderen Basic-Varianten dürfen Arrays ohne vorherige Deklaration verwendet werden. Bei SoftMaker Basic ist dies *nicht* zulässig – Felder müssen vor Benutzung mit **Dim** oder **Static** angelegt werden.

Um die Größe des Arrays zu bestimmen, geben Sie die Obergrenze und optional die Untergrenze für den Index an. Wird die Untergrenze weggelassen, wird der per **Option Base** festgelegte Standardwert genommen – in der Regel also Null.

```
Dim a(10) As Integer            'a(0)..a(10)
Dim b(-10 To 10) As Double      'b(-10)..b(10)
```

Um effizient auf die Elemente eines Arrays zuzugreifen, empfiehlt sich der Einsatz von Schleifen. Folgende **For**-Schleife initialisiert beispielsweise alle Elemente des Arrays "A" mit 1:

```
Static A (1 To 20) As Integer
Dim i As Integer

For i = 1 To 20
   A (i) = 1
Next i
```

Mehrdimensionale Arrays

Arrays können sich auch über mehrere Dimensionen erstrecken:

```
Static a(10, 10) As Double   'zweidimensional
Dim b(5, 3, 2)               'dreidimensional
```

Operatoren

SoftMaker Basic unterstützt die folgenden Operatoren:

Arithmetische Operatoren

Operator	Funktion	Beispiel
+	Addition	x = a + b
-	Subtraktion	x = a - b
	auch: Negation	x = -a
*	Multiplikation	x = a * 3
/	Division	x = a / b
Mod	Modulo	x = a Mod b%
^	Potenzierung	x = a ^ b

Zeichenkettenoperatoren

Operator	Funktion	Beispiel
+	Verkettung	x = "Guten " + "Tag"
&	Verkettung	x = "Guten " & "Tag"

Der Unterschied zwischen den Operatoren **+** und **&** besteht in der Behandlung von Variant-Variablen, die *Zahlen* erhalten: Der Operator **+** addiert diese Zahlen, wohingegen der Operator **&** sie als Zeichenketten verknüpft (siehe Beispiel).

Beispiel:

```
Sub Main
    Dim a, b as Any    '2 Variant-Variablen
    a = 2
    b = 3
    print a + b        ' Ergibt die Zahl 5
    print a & b        ' Ergibt die Zeichenkette "23"
End Sub
```

Vergleichsoperatoren

Operator	Funktion	Beispiel
<	Kleiner als	If x < y Then ...
<=	Kleiner oder gleich	If x <= y Then ...
=	Gleich	If x = y Then ...
>=	Größer oder gleich	If x >= y Then ...
>	Größer als	If x > y Then ...
<>	Ungleich	If x <> y Then ...

Das Ergebnis von Vergleichen mit diesen Operatoren ist ein **Integer**-Wert:

- -1 (**True**) falls die Bedingung zutrifft
- 0 (**False**) falls die Bedingung nicht zutrifft

Logische und Bit-Operatoren

Operator	Funktion	Beispiel
Not	Logische Negation	If Not (x = a) Then ...
And	Logisches Und	If (x > a) And (x < b) Then ...
Or	Logisches Oder	If (x = y) Or (x = z) Then ...

Diese Operatoren wirken bitweise, man kann sie also sowohl für logische Verknüpfungen als auch für Bitmanipulationen benutzen.

Rangordnung von Operatoren

Operatoren werden nach folgender Hierarchie abgearbeitet:

Operator	Funktion	Vorrangstufe
( )	Klammern	höchste
^	Potenzierung	
+ -	Positives/Negatives Vorzeichen	
/ *	Division/Multiplikation	
+ -	Addition/Subtraktion	
Mod	Modulo	
= <> > < <= >=	Vergleichsoperatoren	
Not	Logische Negation	
And	Logisches Und	
Or	Logisches Oder	niedrigste

Kontrollstrukturen

Kontrollstrukturen dienen dazu, abhängig von einer Bedingung Anweisungen auszuführen, zu überspringen oder zu wiederholen. Es gibt folgende Varianten:

Goto-Verzweigung

```
Goto Label1
.
.
.
Label1:
```

Die **Goto**-Anweisung veranlasst einen unbedingten Sprung auf eine Sprungmarke – in obigem Beispiel die Sprungmarke "Label1".

Gosub-Verzweigung

```
Gosub Label1
.
.
.
Label1:
    Anweisung(en)...
Return
```

Auch bei der **Gosub**-Anweisung muss ein Sprungziel angegeben werden. Der Unterschied zur **Goto**-Anweisung liegt darin, dass die **Gosub**-Anweisung zur ursprünglichen Programmstelle zurückkehrt, sobald sie auf eine **Return**-Anweisung trifft.

Do-Schleifen

Mit einer **Do ... Loop**-Schleife kann eine Gruppe von Anweisungen mehrere Male ausgeführt werden. Es gibt folgende Varianten:

```
Do While|Until Bedingung
    Anweisung(en)...
    [Exit Do]
    Anweisung(en)...
Loop
```

Oder:

```
Do
    Anweisung(en)...
Loop While|Until Bedingung
```

Der Unterschied:

Do While und **Do Until** prüfen die Bedingung, *bevor* sie mit der Abarbeitung der Anweisungen innerhalb der Schleife beginnen. Diese werden also *nur dann* ausgeführt, wenn die Bedingung zutrifft.

Bei **Do ... Loop While** und **Do ... Loop Until** hingegen findet die Prüfung erst statt, wenn die Schleife bereits zum ersten Mal durchlaufen ist. Hier werden die Anweisungen innerhalb der Schleife also in jedem Falle *mindestens ein Mal* abgearbeitet.

While-Schleifen

While ... Wend-Schleifen ähneln **Do While ... Loop**-Schleifen. Auch hier wird die Bedingung *vor* dem ersten Abarbeiten der Anweisungen innerhalb der Schleife geprüft.

```
While Bedingung
    Anweisung(en)...
Wend
```

For ... Next-Schleifen

Eine **For ... Next**-Schleife wiederholt die darin enthaltenen Anweisungen anhand eines Zählers genau *n* mal. Bei jedem Durchlaufen der Schleife wird dieser Zähler um die angegebene Schrittweite herauf- beziehungsweise herabgesetzt. Wenn Sie keine Schrittweite angegeben, wird als Schrittweite Eins verwendet.

```
For Zähler = Startwert To Zielwert [Step Schrittweite]
    Anweisung(en)...
Next
```

If-Verzweigungen

Bei einem **If ... Then**-Block werden Anweisungen nur dann ausgeführt, wenn die angegebene Bedingung wahr ist. Diese Bedingung muss ein Ausdruck sein, dessen Ergebnis True oder False ist (zum Beispiel **If a<b Then ...**).

Ein **If ... Then**-Block kann eine oder mehrere Zeilen umfassen. Wenn er sich über mehrere Zeilen erstreckt, muss er mit einer **End If**-Anweisung abgeschlossen werden.

```
If Bedingung Then Anweisung(en)...   'Einzeilige Syntax
```

Oder:

```
If Bedingung Then                    'Mehrzeilige Syntax
    Anweisung(en)...
End If
```

Eine Variante hiervon ist die **If ... Then ... Else**-Anweisung. Die Anweisungen nach **Else** werden ausgeführt, wenn die Bedingung *nicht* zutrifft.

```
If Bedingung Then
    Anweisung(en)...
Else
    Anweisung(en)...
End If
```

Weitere Verzweigungen könnten mit zusätzlichen **If ... Then ... ElseIf**-Anweisungen erreicht werden. Dies führt jedoch leicht zu unübersichtlichem Code, weshalb man hier die **Select Case**-Anweisung vorziehen sollte (siehe unten).

```
If Bedingung Then
    Anweisung(en)...
ElseIf Bedingung Then
    Anweisung(en)...
Else
    Anweisung(en)...
End If
```

Select Case-Verzweigungen

Bei der **Select Case**-Anweisung wird eine Variable auf verschiedene Werte überprüft.

```
Select Case Variable
       Case Wert1
          Anweisung(en)...
       Case Wert2
          Anweisung(en)...
       Case Wert3
          Anweisung(en)...
       [Case Else
          Anweisung(en)...]
End Select
```

Trägt die Variable den Inhalt "Wert1", werden die Anweisungen bei **Case Wert1** abgearbeitet etc. Hat sie keinen der angegebenen Werte, wird zu den Anweisungen bei **Case Else** verzweigt (sofern vorhanden, ansonsten wird die Struktur einfach verlassen).

Unterprogramme und Funktionen

Sie können eigene Funktionen und Unterprogramme definieren, die sich anschließend wie die eingebauten Funktionen und Anweisungen von SoftMaker Basic benutzen lassen. Weiterhin ist es möglich, Funktionen in beliebigen DLLs aufzurufen.

- Benutzerdefinierte *Unterprogramme* werden mit der **Sub**-Anweisung definiert.
- Benutzerdefinierte *Funktionen* werden mit der **Function**-Anweisung definiert.
- *Funktionen in DLLs* müssen mit der **Declare**-Anweisung deklariert werden (siehe Abschnitt "Aufruf von Funktionen in DLLs").

Hinweise zur Namensgebung von Unterprogrammen und Funktionen

Namen für Unterprogramme und Funktionen dürfen nur die Buchstaben A-Z und a-z, den Unterstrich (_) und die Ziffern 0-9 enthalten. Das erste Zeichen muss stets ein Buchstabe sein. Der Name darf nicht mehr als 40 Zeichen umfassen. Er darf nicht aus einem Schlüsselwort von SoftMaker Basic bestehen.

Parameterübergabe ByRef oder ByVal

Parameter können an Prozeduren als Referenz (**ByRef**) oder als Wert (**ByVal**) übergeben werden:

- **ByRef**

  Die Übergabe **ByRef** ("als Referenz") ermöglicht es der aufgerufenen Prozedur, den Wert der übergebenen Variablen zu ändern.

  ByRef ist die Standardmethode für die Parameterübergabe und muss daher nicht explizit angegeben werden. `Sub Test(j As Integer)` ist also gleichbedeutend mit `Sub Test(ByRef j As Integer)`.

- **ByVal**

  Bei der Übergabe **ByVal** ("als Wert") erhält die Prozedur lediglich eine Kopie der Variable, sodass Änderungen des Parameters innerhalb der Prozedur sich nicht auf die übergebene Variable selbst auswirken.

  Die Übergabe als Wert kann bei der Prozedurdefinition festgelegt werden, indem vor dem Parameter das Schlüsselwort **ByVal** geschrieben wird: `Sub Joe(ByVal j As Integer)`.

  Alternativ kann man dies auch beim Aufruf der Prozedur erzwingen, indem man den Parameter in Klammern setzt. Hier wird beispielsweise der Parameter Var3 *ByVal* übergeben:

  ```
  SubOne Var1, Var2, (Var3)
  ```

Aufruf von Funktionen in DLLs

Um eine Funktion in einer DLL aufzurufen, muss diese zunächst mit einer **Declare**-Anweisung deklariert werden. Wenn die aufzurufende Prozedur keinen Wert zurückgibt, wird sie als **Sub**, ansonsten als **Function** deklariert.

Beispiel:

```
Declare Function GetPrivateProfileString Lib "Kernel32" (ByVal lpApplicationName As
String, ByVal lpKeyName As String, ByVal lpDefault As String, ByVal lpReturnedString As
String, ByVal nSize As Integer, ByVal lpFileName As String) As Integer

Declare Sub InvertRect Lib "User32" (ByVal hDC As Integer, aRect As Rectangle)
```

Sobald die Prozedur deklariert wurde, kann sie wie jede andere Basic-Funktion beziehungsweise -Anweisung verwendet werden.

Dateioperationen

Mit SoftMaker Basic können Sie alle gängigen Dateioperationen durchführen. Nachfolgend ein kleines Beispiel. Ausführliche Informationen zu den einzelnen Anweisungen können Sie im Abschnitt "Anweisungen und Funktionen von A-Z" nachlesen.

Beispiel:

```
Sub FileIO_Example
    Dim Msg                             'Variablen deklarieren
    Call Make3Files()
    Msg = "Drei Testdateien wurden angelegt. "
    Msg = Msg & "Mit OK werden sie wieder gelöscht."
    MsgBox Msg
    For I = 1 To 3
        Kill "TEST" & I                 'Dateien löschen
        Next I
End Sub

Sub Make3Files
    Dim I, FNum, Fname
    For I = 1 To 3
        FNum = FreeFile                 'Nächste Dateinummer ermitteln
        FName = "TEST" & FNum
        Open FName For Output As FNum   'Datei öffnen
        Print #I, "This is test #" & I  'In Datei schreiben
        Print #I, "Here is another "; "line"; I
```

```
    Next I
    Close                                   'Alle Dateien schließen
End Sub
```

Dialogfenster

Sie können in SoftMaker Basic eigene Dialogfenster definieren und diese dann mit der Funktion **Dialog** aufrufen und auswerten.

Dialoge lassen sich wahlweise durch manuelle Eingabe der Dialogdefinition oder über den integrierten Dialogeditor anlegen.

Ein Dialog kann optional mit einer *Dialogfunktion* verknüpft werden, die das Aktivieren/Deaktivieren und Einblenden/Ausblenden von Dialogelementen erlaubt und das Anlegen verschachtelter Dialoge ermöglicht.

Dialogdefinition

Um ein Dialogfenster zu erstellen, müssen Sie eine *Dialogdefinition* in das Script einfügen. Sie können dazu entweder den integrierten Dialogeditor verwenden (siehe Abschnitt "Dialogeditor verwenden") oder die Dialogdefinition von Hand in das Script eintragen.

In den nachfolgenden Abschnitten wollen wir uns eine solche Dialogdefinition einmal genauer ansehen.

Syntax einer Dialogdefinition

Dialogdefinitionen müssen von den Anweisungen **Begin Dialog** und **End Dialog** umgeben werden:

```
Begin Dialog DialogName [X, Y] Breite, Höhe, Titel$ [,.Dialogfunktion]
   ' Hier Steuerelemente definieren
End Dialog
```

Die Parameter haben folgende Bedeutung:

Parameter	Beschreibung
DialogName	Name der Dialogdefinition. Nach erfolgter Dialogdefinition kann eine Variable dieses Typs dimensioniert werden (**Dim** Name **As** Dialogname).
X, Y	Optional. Legt die Koordinaten für die linke obere Ecke des Dialogfensters fest (in Pixel).
Breite, Höhe	Bestimmt die Breite und Höhe des Dialogs (in Pixel).
Titel$	Der Titel des Dialogs – wird in der Titelleiste des Dialogfensters angezeigt.
.Dialogfunktion	Dialogfunktion für diesen Dialog (optional). Ermöglicht das Aktivieren/Deaktivieren und Einblenden/Ausblenden von Dialogelementen sowie das Erzeugen verschachtelter Dialoge (siehe Abschnitt "Die Dialogfunktion").

Innerhalb der Dialogdefinition selbst sind die im Dialog anzuzeigenden Steuerelemente einzutragen. Hierfür können die im nächsten Abschnitt aufgeführten Schlüsselwörter verwendet werden.

Beispiel:

```
Sub Main
   Begin Dialog QuitDialogTemplate 16,32,116,64,"Beenden?"
      Text 4,8,108,8,"Möchten Sie das Programm beenden?"
      CheckBox 32,24,63,8,"Änderungen speichern",.SaveChanges
      OKButton 12,40,40,14
      CancelButton 60,40,40,14
   End Dialog

Dim QuitDialog As QuitDialogTemplate

rc% = Dialog(QuitDialog)
```

```
' Hier können Sie das Ergebnis (rc%) des Dialogs auswerten
End Sub
```

Steuerelemente eines Dialogfensters

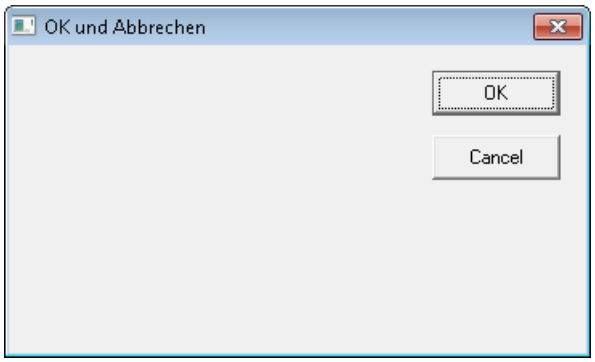
Folgende Steuerelemente können in Dialogfenstern verwendet werden:

- Befehlsschaltflächen
- Text und Eingabefelder
- Listenfelder, Kombinationsfelder und Dropdown-Listenfelder
- Kontrollkästchen
- Optionsfelder und Gruppenfelder

Befehlsschaltflächen

Die Schaltflächen **OK** und **Abbrechen** bezeichnet man als *Befehlsschaltflächen*.

Hinweis: Jeder Dialog muss mindestens eine Befehlsschaltfläche enthalten.



Syntax: **OKButton** *X, Y, Breite, Höhe*

CancelButton *X, Y, Breite, Höhe*

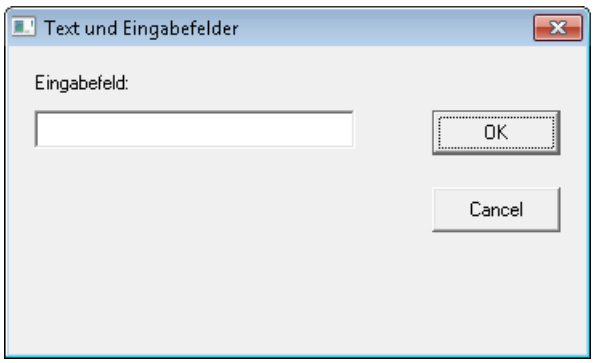
Beispiel:

```
Sub Main

Begin Dialog ButtonSample 16,32,180,96,"OK und Abbrechen"
    OKButton 132,8,40,14
    CancelButton 132,28,40,14
End Dialog

Dim Dlg1 As ButtonSample

rc% = Dialog (Dlg1)

End Sub
```

Text und Eingabefelder

Mit *Text* können die Bestandteile eines Dialogs beschriftet werden.

In einem *Eingabefeld* (Schlüsselwort "TextBox") kann der Anwender Eingaben vornehmen.



Syntax: **Text** *X, Y, Breite, Höhe, Text*

TextBox *X, Y, Breite, Höhe, .ID*

ID ist eine Variable, die den aktuellen Text enthält.

Beispiel:

```
Sub Main

Begin Dialog TextBoxSample 16,30,180,96,"Text und Eingabefelder"
    OKButton 132,20,40,14
    CancelButton 132,44,40,14
    Text 8,8,32,8,"Eingabefeld:"
    TextBox 8,20,100,12,.TextBox1
End Dialog

Dim Dlg1 As TextBoxSample
rc% = Dialog(Dlg1)

End Sub
```

Listenfelder, Kombinationsfelder und Dropdown-Listenfelder

Listenfelder zeigen Listen an, in denen der Anwender eine Auswahl treffen kann.

Es gibt drei Arten von Listenfeldern:

- **Gewöhnliches Listenfeld** (engl. "List Box")

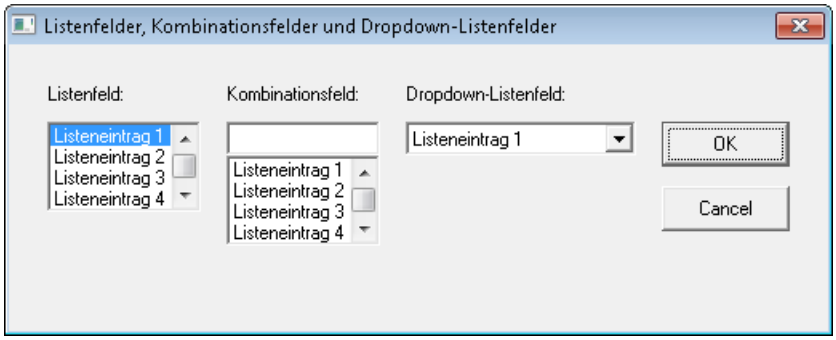
  Hier kann der Anwender einen der Listeneinträge auswählen.

- **Kombinationsfeld** (engl. "Combo Box")

  Hier kann der Anwender entweder einen der Listeneinträge auswählen oder selbst eine Eingabe vornehmen.

- **Dropdown-Listenfeld** (engl. "Drop-down List Box")

  Eine platzsparende Variante von Listenfeldern: Der Anwender muss sie aufklappen, bevor er eine Auswahl treffen kann.



Syntax: **ListBox** *X, Y, Breite, Höhe, Inhalt, .ID*

ComboBox *X, Y, Breite, Höhe, Inhalt, .ID*

DropListBox *X, Y, Breite, Höhe, Inhalt, .ID*

Die einzelnen Texte werden über das String-Array *Inhalt* initialisiert, das vor Aufruf von **Dialog** gefüllt werden sollte.

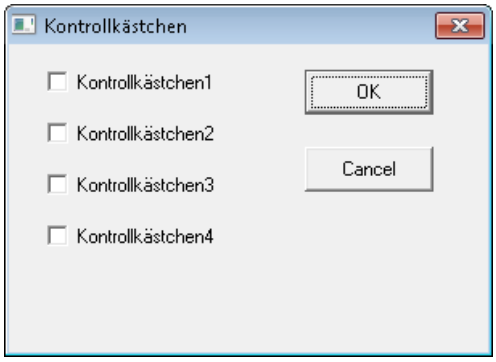
ID ist eine Variable, die die aktuell markierte Auswahl enthält.

Beispiel:

```
Sub Main

Dim MyList$(5)
MyList(0) = "Listeneintrag 1"
MyList(1) = "Listeneintrag 2"
MyList(2) = "Listeneintrag 3"
MyList(3) = "Listeneintrag 4"
MyList(4) = "Listeneintrag 5"
MyList(5) = "Listeneintrag 6"

Begin Dialog BoxSample 16,35,256,89,"Listenfelder, Kombinationsfelder und Dropdown-
Listenfelder"
    OKButton 204,24,40,14
    CancelButton 204,44,40,14
    ListBox 12,24,48,40, MyList$(),.Lstbox
    DropListBox 124,24,72,40, MyList$(),.DrpList
    ComboBox 68,24,48,40, MyList$(),.CmboBox
    Text 12,12,32,8,"Listenfeld:"
    Text 124,12,68,8,"Dropdown-Listenfeld:"
    Text 68,12,44,8,"Kombinationsfeld:"
End Dialog

Dim Dlg1 As BoxSample
rc% = Dialog(Dlg1)

End Sub
```

Kontrollkästchen

Kontrollkästchen ("Check Box") eignen sich für "Ja/Nein"- oder "An/Aus"-Optionen.



Syntax: **CheckBox** *X, Y, Breite, Höhe, Text, .ID*

ID ist eine Variable, die den aktuellen Zustand enthält.

Beispiel:

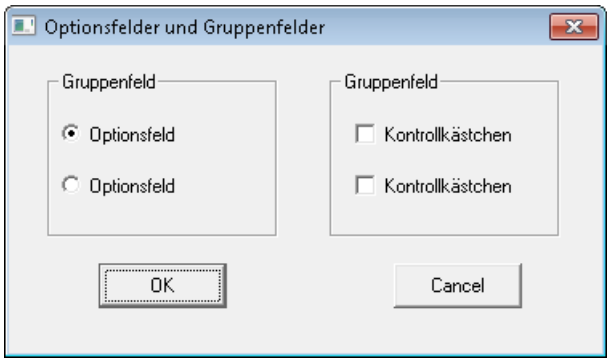
```
Sub Main

Begin Dialog CheckSample 15,32,149,96,"Kontrollkästchen"
    OKButton 92,8,40,14
    CancelButton 92,32,40,14
    CheckBox 12, 8,60,8,"Kontrollkästchen1",.CheckBox1
    CheckBox 12,24,60,8,"Kontrollkästchen2",.CheckBox2
    CheckBox 12,40,60,8,"Kontrollkästchen3",.CheckBox3
    CheckBox 12,56,60,8,"Kontrollkästchen4",.CheckBox4
End Dialog

Dim Dlg1 As CheckSample
```

```
rc% = Dialog(Dlg1)

End Sub
```

Optionsfelder und Gruppenfelder

Verwenden Sie *Optionsfelder* ("Option Buttons" oder "Radio Buttons"), wenn dem Anwender mehrere Möglichkeiten zur Wahl gestellt werden sollen, von denen er jedoch nur *eine* auswählen darf.

Eine zusammengehörige Gruppe von Optionsfeldern wird meist in einem *Gruppenfeld* ("Group Box") zusammengefasst. Sie können Gruppenfelder aber auch verwenden, um beliebige andere Steuerelemente zu einer Gruppe zusammenzufassen.



Syntax: **OptionButton** *X, Y, Breite, Höhe, Text, .ID1*

OptionGroup *.ID2*

ID1 ist eine Variable, die den aktuellen Zustand des Feldes enthält.

ID2 ist eine Variable, die den Index der aktuell markierten Option enthält.

Beispiel:

```
Sub Main

Begin Dialog GroupSample 31,32,185,96,"Optionsfelder und Gruppenfelder"
    OKButton 28,68,40,14
    CancelButton 120,68,40,14
    GroupBox 12,8,72,52,"Gruppenfeld",.GroupBox1
    GroupBox 100,8,72,52,"Gruppenfeld",.GroupBox2
    OptionGroup .OptionGroup1
    OptionButton 16,24,54,8,"Optionsfeld",.OptionButton1
    OptionButton 16,40,54,8,"Optionsfeld",.OptionButton2
    CheckBox 108,24,50,8,"Kontrollkästchen",.CheckBox1
    CheckBox 108,40,50,8,"Kontrollkästchen",.CheckBox2
End Dialog

Dim Dlg1 As GroupSample
Button = Dialog (Dlg1)

End Sub
```

Die Dialogfunktion

Ein benutzerdefinierter Dialog kann bei Bedarf mit einer *Dialogfunktion* verknüpft werden. Diese Funktion wird aufgerufen, wenn das Dialogfeld initialisiert wird und wenn der Benutzer ein Kontrollelement betätigt. Mit Hilfe einer Dialogfunktion ist es möglich, Dialoge zu verschachteln und Kontrollelemente zu deaktivieren oder auszublenden.

Um ein Dialogfenster mit einer Dialogfunktion zu verknüpfen, wird der Funktionsname in die Dialogdefinition eingefügt, wobei ein Punkt vorangestellt wird. Hier wird beispielsweise dem Dialog **MyDlg** eine Dialogfunktion namens **MyDlgFunc** zugewiesen:

```
Begin Dialog MyDlg 60, 60, 260, 188, "3", .MyDlgFunc
```

Die zu überwachenden Steuerelemente

Jedem Steuerelement des Dialogfensters, das von der Dialogfunktion überwacht werden soll, muss ein eindeutiger Bezeichner zugewiesen werden. Dieser muss als letzter Parameter in der Definition des Steuerelements angegeben werden und mit einem Punkt beginnen.

```
CheckBox 8,56,203,16, "Alles anzeigen",.Chk1
```

Hier wird dem Kontrollkästchen der Bezeichner "Chk1" zugewiesen.

Syntax der Dialogfunktion

Die Syntax der Dialogfunktion ist wie folgt aufgebaut:

```
Function FunctionName(ControlID$, Action%, SuppValue%)
    [Anweisungen]
    FunctionName = ReturnValue
End Function
```

Die Dialogfunktion liefert einen Wert zurück, wenn der Anwender auf **OK** oder **Abbrechen** klickt. Setzen Sie diesen *ReturnValue* in der Dialogfunktion auf 0, wird der Dialog geschlossen, bei einem anderen Wert bleibt der Dialog offen.

Die Parameter der Dialogfunktion:

- **ControlID$**

  Wenn Action = 2 ist, enthält dieser Parameter die (bei der Dialogdefinition angegebene) *ID* des vom Anwender betätigten Kontrollelements.

- **Action%**

  1, wenn der Dialog initialisiert wird (die anderen Parameter haben dann keine Bedeutung)

  2, wenn der Benutzer ein Kontrollelement betätigt hat. Das Kontrollelement wird durch *ControlId*$ identifiziert, *SuppValue*% enthält weitere Informationen.

- **SuppValue%:**

  Informationen, welche Art von Änderung durchgeführt wurde, abhängig vom Typ des Kontrollelements:

  Kontrollkästchen: 0, wenn ausgeschaltet, beziehungsweise 1, wenn eingeschaltet

  Optionsfeld: Nummer des gewählten Optionsfelds, wobei das erste Feld der Gruppe die Nummer 0 trägt

  Befehlsschaltfläche: Keine Bedeutung

  OK: 1

  Abbrechen: 2

Im folgenden Beispiel wird die Dialogfunktion eines Dialogs mittels einer **Case**-Verzweigung ausgewertet. Der Parameter **SuppValue** wird dabei nicht berücksichtigt.

```
Sub Main

Begin Dialog UserDialog1 60,60, 260, 188, "Dialogfunktion", .Dialogfn
    Text 8,10,73,13, "Text:"
    TextBox 8, 26, 160, 18, .FText
    CheckBox 8, 56, 203, 16, "Alles anzeigen",. Chk1
    GroupBox 8, 79, 230, 70, "Gruppenfeld:", .Group
    CheckBox 18,100,189,16, "Beschriftung der Schaltfläche ändern", .Chk2
    PushButton 18, 118, 159, 16, "Schaltfläche", .History
    OKButton 177, 8, 58, 21
    CancelButton 177, 32, 58, 21
End Dialog

Dim Dlg1 As UserDialog1
x = Dialog( Dlg1 )

End Sub   ' (Main)
```

```
Function Dialogfn(ControlID$, Action%, SuppValue%)
Begin Dialog UserDialog2 160,160, 260, 188, "Dialogfunktion", .Dialogfunktion
    Text 8,10,73,13, "Eingabefeld"
    TextBox 8, 26, 160, 18, .FText
    CheckBox 8, 56, 203, 16, "Kontrollkästchen",. ch1
    CheckBox 18,100,189,16, "Kontrollkästchen", .ch2
    PushButton 18, 118, 159, 16, "Schaltfläche", .but1
    OKButton 177, 8, 58, 21
    CancelButton 177, 32, 58, 21
End Dialog

Dim Dlg2 As UserDialog2
Dlg2.FText = "Dies ist die Vorgabe"

Select Case Action%

    Case 1
        DlgEnable "Group", 0
        DlgVisible "Chk2", 0
        DlgVisible "History", 0

    Case 2
        If ControlID$ = "Chk1" Then
                              DlgEnable "Group"
                              DlgVisible "Chk2"
        End If

        If ControlID$ = "Chk2" Then
                              DlgText "History", "Weiteren Dialog anzeigen"
        End If

        If ControlID$ = "History" Then
                                 Dialogfn =1
                                 x = Dialog(Dlg2)
        End If

    Case Else

End Select

Dialogfn=1

End Function
```

OLE Automation

Mit Hilfe von OLE Automation können geeignete Applikationen (zum Beispiel die Textverarbeitung TextMaker oder die Tabellenkalkulation PlanMaker) mit SoftMaker Basic-Scripts gesteuert werden.

> ***Tipp:*** Ausführliche Informationen zum Programmieren von TextMaker und PlanMaker finden Sie in den Kapiteln "BasicMaker und TextMaker" sowie "BasicMaker und PlanMaker".

Was ist ein OLE Automation-Objekt?

Jedes für OLE-Automation geeignete Programm stellt bestimmte *Objekte* zur Verfügung. Die Art dieser Objekte hängt von der Applikation ab. Eine Textverarbeitung wie TextMaker bietet beispielsweise Objekte wie die derzeit geöffneten Dokumente oder die Formatierung des derzeit markierten Textes an.

Ein OLE Automation-Objekt bietet zwei Arten von Zugriffsmöglichkeiten:

- Die **Eigenschaften (Properties)** von OLE Automation-Objekten sind Werte, die gelesen und/oder geschrieben werden können und einen bestimmten Aspekt des Objekts beschreiben. Ein Dokumentfenster einer Textverarbeitung hat zum Beispiel die Eigenschaften Name (des darin geöffneten Dokuments), Breite und Höhe des Fensters usw.
- **Methoden** sind Funktionen, die eine Aktion an einem OLE Automation-Objekt auslösen. Ein in der Applikation geöffnetes Dokument hat zum Beispiel eine Methode zum Speichern des Dokuments.

Zugriff auf OLE Automation-Objekte

Um auf ein OLE Automation-Objekt zuzugreifen, muss zunächst eine Variable vom Typ **Object** deklariert werden.

Beispiel:

```
Dim MyObj As Object
```

Dann muss diese mit der Applikation "verbunden" werden. Dazu gibt es zwei Funktionen: Während **CreateObject** die Applikation nötigenfalls automatisch startet, kann **GetObject** nur eine Verbindung zu einer bereits gestarteten Applikation herstellen.

Beispiel:

```
Set MyObj = CreateObject("TextMaker.Application")
```

Die Variable `MyObj` enthält nun eine Referenz auf das Haupt-OLE Automation-Objekt der Applikation, dessen Name übrigens immer **Application** lautet. Über die Punktnotation – zum Beispiel **MyObj.Application.Documents** – können Sie auf die untergeordneten Objekte zugreifen (siehe auch nächster Abschnitt).

Wenn die OLE Automation-Verbindung nicht mehr benötigt wird, sollte die Variable wieder von dem Objekt getrennt werden, indem sie auf **Nothing** gesetzt wird:

Beispiel:

```
Set MyObj = Nothing ' Variable vom Objekt trennen
```

Eigenschaften

Um auf die Eigenschaften eines Objekts zuzugreifen, wird die Punktnotation *Objekt.Eigenschaft* verwendet.

Beispiel:

```
x = MyObj.Application.Width ' Breite des Programmfensters auslesen
```

Oder:

```
MyObj.Application.Width = 5 ' Breite des Programmfensters setzen
```

Methoden

Zum Aufrufen von Methoden wird ebenfalls die Punktnotation eingesetzt: *Objekt.Methode*

Beispiel:

```
MyObj.Application.Quit ' Applikation beenden
```

Sammlungen verwenden

Neben einfachen Objekten gibt es auch *Sammlungen* (Collections) von Objekten.

TextMaker bietet beispielsweise die Sammlung **Documents** (eine Sammlung aller geöffneten Dokumente) an. Eine Sammlung ist selbst ein Objekt, das meist als Eigenschaft ihres Elternobjekts zugänglich ist.

Um auf alle Elemente einer Sammlung zuzugreifen, kann man die **For Each ... Next**-Anweisung benutzen.

Alle Sammlungen bieten standardmäßig die folgenden Eigenschaften und Methoden:

Count	Liefert die Anzahl der Elemente (nur lesbar).
Item(*i*)	Liefert das *i*-te Element.
Add	Fügt der Sammlung ein neues Objekt hinzu.

Beispiel

Abschließend ein Beispiel, das die Anwendung von OLE Automation in der Praxis demonstriert. Das Beispiel verwendet die Sammlung **Documents** von TextMaker, die für alle derzeit geöffneten Dokumente steht. Erst wird ermittelt, wie viele Dokumente gerade geöffnet sind. Anschließend werden die Namen der geöffneten Dokumente ausgegeben. Schließlich werden die Dokumente geschlossen.

> ***Tipp:*** Ausführliche Informationen zu den Themen "BasicMaker und TextMaker" sowie "BasicMaker und PlanMaker" finden Sie in den gleichnamigen Kapiteln.

```
Sub Main

    Dim tm As Object
    Set tm = CreateObject("TextMaker.Application")

    tm.Visible = TRUE     ' TM sichtbar machen
    tm.Activate           ' TM in den Vordergrund bringen

    tm.Documents.Add      ' Drei neue Dokumente anlegen
    tm.Documents.Add
    tm.Documents.Add

    Print tm.Documents.Count & " Dokumente geöffnet"

    Dim x As Object
    For Each x in tm.Documents
        Print x.Name      ' Namen der Dokumente ausgeben
    Next

    tm.Documents.Close    ' Alle Dokumente schließen

    Set tm = Nothing      ' Verbindung zu TM beenden

End Sub
```

BasicMaker und TextMaker

BasicMaker wurde in erster Linie entwickelt, um eine Möglichkeit zu schaffen, TextMaker und PlanMaker programmieren, quasi "fernsteuern" zu können. Dieses Kapitel enthält alle Informationen zur TextMaker-Programmierung. Es ist in folgende Abschnitte gegliedert:

- **Programmierung von TextMaker**

  Dieser Abschnitt enthält grundlegende Informationen zur Programmierung der Textverarbeitung TextMaker mit BasicMaker.

- **Objektstruktur von TextMaker**

  In diesem Abschnitt werden alle von TextMaker zur Verfügung gestellten Objekte beschrieben.

Der Programmierung von PlanMaker ist ein **separates Kapitel** gewidmet.

Programmierung von TextMaker

Die Programmierung der Textverarbeitung *TextMaker* und die der Tabellenkalkulation *PlanMaker* unterscheiden sich nur darin, dass einige Schlüsselwörter andere Namen haben (zum Beispiel PlanMaker.Application statt TextMaker.Application). Wenn Sie also den Abschnitt "Programmierung von PlanMaker" bereits kennen, werden Sie feststellen, dass dieser hier nahezu identisch ist.

TextMaker stellt aber natürlich andere Objekte zur Verfügung als PlanMaker. Eine Auflistung der exponierten Objekte finden Sie im nächsten Abschnitt "Objektstruktur von TextMaker".

Um TextMaker mit BasicMaker zu programmieren, verwenden Sie in erster Linie *OLE Automation-Befehle*. Allgemeine Informationen zu diesem Thema erhalten Sie im Abschnitt "OLE Automation".

Prinzipiell ist folgendermaßen vorzugehen (Details folgen im Anschluss):

1. Deklarieren Sie eine Variable vom Typ **Object**:

   ```
   Dim tm as Object
   ```

2. Stellen Sie über OLE Automation eine Verbindung zu TextMaker her (TextMaker wird dazu nötigenfalls automatisch gestartet):

   ```
   Set tm = CreateObject("TextMaker.Application")
   ```

3. Setzen Sie die Eigenschaft **Application.Visible** auf **True**, damit TextMaker sichtbar wird:

   ```
   tm.Application.Visible = True
   ```

4. Jetzt können Sie TextMaker programmieren, indem Sie "Eigenschaften" von TextMaker auslesen und abändern und die von TextMaker bereitgestellten "Methoden" anwenden.

5. Wird das TextMaker-Objekt nicht mehr benötigt, sollten Sie die Verbindung zu TextMaker trennen:

   ```
   Set tm = Nothing
   ```

Soweit in aller Kürze. Auf den nächsten Seiten folgen ausführlichere Informationen zur Programmierung von TextMaker. Eine Aufstellung aller TextMaker-Objekte und der darauf anwendbaren Eigenschaften und Methoden finden Sie anschließend im Abschnitt "Objektstruktur von TextMaker".

Verbindung zu TextMaker herstellen

Wenn Sie TextMaker mit BasicMaker steuern wollen, müssen Sie zuerst über eine OLE Automation eine Verbindung zu TextMaker herstellen. Dazu ist eine Variable vom Typ **Object** zu deklarieren, der anschließend mit der Funktion **CreateObject** das Objekt "TextMaker.Application" zugewiesen wird:

```
Dim tm as Object
```

```
Set tm = CreateObject("TextMaker.Application")
```

Wenn TextMaker bereits läuft, wird dadurch lediglich eine Verbindung zu ihm aufgebaut; wenn TextMaker noch nicht gestartet ist, wird er automatisch gestartet.

Die Objektvariable "tm" enthält nun eine Referenz auf TextMaker.

Wichtig: TextMaker sichtbar machen

Bitte beachten Sie: Wenn Sie TextMaker wie gerade beschrieben starten, ist das Programmfenster standardmäßig *unsichtbar*. Soll TextMaker sichtbar gemacht werden, muss die Eigenschaft **Visible** auf **True** gesetzt werden.

Der vollständige Aufruf von TextMaker sollte also lauten:

```
Dim tm as Object
Set tm = CreateObject("TextMaker.Application")
tm.Application.Visible = True
```

Das Objekt "Application"

Das *grundlegende* Objekt, das TextMaker für die Programmierung exponiert, ist **Application**. Alle anderen Objekte – wie zum Beispiel Listen der geöffneten Dokumente und Fenster – "hängen" am **Application**-Objekt.

Das **Application**-Objekt enthält einerseits eigene Eigenschaften (zum Beispiel **Application.Left** für die X-Koordinate des Programmfensters) und Methoden (wie **Application.Quit** zum Beenden von TextMaker), andererseits enthält es Zeiger auf andere Objekte wie **Application.Options**, die wiederum eigene Eigenschaften und Methoden enthalten, und Zeiger auf Sammlungen ("Collections") wie **Documents** (die Liste der gerade geöffneten Dokumente).

Schreibweisen

Wie Sie aus dem vorherigen Abschnitt schon ersehen können, ist für den Zugriff auf die bereitgestellten Eigenschaften, Methoden usw. die bei OLE-Automation übliche Punktnotation zu verwenden.

Mit **Application.Left** wird beispielsweise die Eigenschaft **Left** des Objekts **Application** angesprochen. **Application.Documents.Add** bezeichnet die Methode **Add** der Sammlung **Documents**, die wiederum ein Objekt von **Application** ist.

Eigenschaften (Properties) von TextMaker auslesen und ändern

Wurde die Verbindung zu TextMaker hergestellt, können Sie das Programm "fernsteuern". Dazu gibt es, wie im Abschnitt "OLE Automation" beschrieben, *Eigenschaften (Properties)* und *Methoden (Methods)*.

Beschäftigen wir uns zunächst mit den *Eigenschaften*. Als Eigenschaften bezeichnet man Optionen und Einstellungen, die abgefragt und teilweise verändert werden können.

Möchten Sie zum Beispiel den Programmnamen von TextMaker ermitteln, verwenden Sie die Eigenschaft **Name** des Objekts **Application**:

```
MsgBox "Der Name dieser Applikation ist: " & tm.Application.Name
```

Bei **Application.Name** handelt es sich um eine Eigenschaft, die nur gelesen werden kann. Andere Eigenschaften lassen sich sowohl auslesen als auch von einem BasicMaker-Script aus abändern. So sind die Koordinaten des TextMaker-Programmfensters in den Eigenschaften **Left, Top**, **Width** und **Height** des Application-Objekts abgelegt. Sie können sie wieder auslesen:

```
MsgBox "Der linke Fensterrand liegt bei: " & tm.Application.Left
```

Sie können diese Eigenschaft aber auch verändern:

```
tm.Application.Left = 200
```

TextMaker reagiert sofort und verschiebt den linken Fensterrand der Applikation auf dem Bildschirm an die Pixelposition 200. Sie können Lesen und Schreiben von Eigenschaften auch mischen, etwa:

```
tm.Application.Left = tm.Application.Left + 100
```

Hier wird der aktuelle linke Rand ausgelesen, um 100 erhöht und als neuer linker Rand an TextMaker übergeben. Auch hier reagiert TextMaker sofort und schiebt seinen linken Fensterrand um 100 Pixel nach rechts.

Es gibt eine große Anzahl von Eigenschaften des **Application**-Objekts. Eine Auflistung finden Sie im Abschnitt "Objektstruktur von TextMaker".

Methoden (Methods) von TextMaker verwenden

Neben Eigenschaften gibt es *Methoden*. Methoden sind Befehle, die TextMaker anweisen, etwas Bestimmtes zu tun.

So können Sie zum Beispiel mit **Application.Quit** TextMaker anweisen, sich zu beenden; mit **Application.Activate** erzwingen Sie, dass das TextMaker-Programmfenster in den Vordergrund kommt, wenn es gegenwärtig von Fenstern anderer Programme überdeckt wird:

```
tm.Application.Activate
```

Unterschied zwischen Funktions- und Prozedurmethoden

Es gibt zwei Arten von Methoden: solche, die einen Wert an das Basic-Programm zurückliefern und solche ohne Rückgabewert. Erstere bezeichnen wir – in Anlehnung an andere Programmiersprachen – als "Funktionsmethoden" oder einfach "Funktionen", letztere als "Prozedurmethoden" oder "Prozeduren".

Diese Unterscheidung mag Ihnen vielleicht übertrieben feinsinnig erscheinen, sie ist es aber nicht, weil sie Auswirkungen auf die Schreibweise der Befehle hat.

Solange Sie eine Methode ohne Parameter aufrufen, gibt es keinen syntaktischen Unterschied:

Aufruf als Prozedur:

```
tm.Documents.Add ' Ein Dokument zu den offenen Dokumenten hinzufügen
```

Aufruf als Funktion:

```
Dim newDoc as Object
Set newDoc = tm.Documents.Add ' jetzt mit dem Document-Objekt als Rückgabewert
```

Bei Methoden *mit* Parametern sind aber unterschiedliche Schreibweisen erforderlich:

Aufruf als Prozedur:

```
tm.ActiveDocument.Tables.Add 3, 3 ' Eine 3*3-Tabelle einfügen
```

Aufruf als Funktion:

```
Dim newTable as Object
Set newTable = tm.ActiveDocument.Tables.Add(3, 3) ' jetzt mit Rückgabewert
```

Sie sehen: Beim Aufruf als Prozedur dürfen Sie die Parameter *nicht* mit Klammern umgeben, beim Aufruf als Funktion *müssen* Sie es.

Zeiger auf andere Objekte verwenden

Eine dritte Gruppe von Elementen des **Application**-Objekts sind *Zeiger auf andere Objekte*.

Stellen Sie sich hier bitte nichts großartig Kompliziertes vor. Es ist lediglich unübersichtlich, alle Eigenschaften und Methoden von TextMaker unmittelbar an das Application-Objekt zu hängen, da die Objektstruktur dadurch sehr unübersichtlich würde. Deshalb sind bestimmte Reihen von Eigenschaften und Methoden zu logischen Gruppen zusammengefasst. So kennt TextMaker beispielsweise das Objekt **Options**, mit dem Sie viele grundlegende Programmeinstellungen auslesen und verändern können:

```
tm.Application.Options.CreateBackup = True

MsgBox "Überschreibmodus eingeschaltet? " & tm.Application.Options.Overtype
```

Sammlungen verwenden

Die vierte Gruppe von Elementen des **Application**-Objekts sind Zeiger auf *Sammlungen* ("Collections").

Sammlungen enthalten, wie der Name schon sagt, eine Ansammlung von gleichartigen Objekten. Es gibt zum Beispiel eine Sammlung **Application.Documents**, die alle geöffneten Dokumente enthält und eine Sammlung **Application.RecentFiles** mit allen Dateien, die im Datei-Menü aufgelistet werden.

Es existieren zwei standardisierte Arten, um auf Sammlungen zuzugreifen, und TextMaker unterstützt beide. Die simplere Art ist die Eigenschaft **Item**, die in jeder Sammlung vorhanden ist:

```
' Namen des ersten geöffneten Dokuments ausgeben:
MsgBox tm.Application.Documents.Item(1).Name

' Schließt das (geöffnete) Dokument "Test.tmd":
tm.Application.Documents.Item("Test.tmd").Close
```

Wollen Sie beispielsweise alle geöffneten Dokumente auflisten, lassen Sie sich zuerst mit der standardisierten Eigenschaft **Count** die Zahl der offenen Dokumente geben und greifen dann sukzessive auf die einzelnen Elemente, also Dokumente, zu:

```
' Gibt die Namen aller geöffneten Dokumente aus:
For i=1 To tm.Application.Documents.Count
   MsgBox tm.Application.Documents.Item(i).Name
Next i
```

Jede Sammlung besitzt also per Definition die Eigenschaft **Count**, die die Zahl der Einträge in der Sammlung ermittelt, und die Eigenschaft **Item**, mit der Sie gezielt an einen Eintrag in der Sammlung herankommen.

Item akzeptiert als Argument stets die Nummer des gewünschten Eintrags. Soweit es sinnvoll ist, akzeptiert **Item** als Argument auch andere Argumente, zum Beispiel Dateinamen. Sie haben dies bereits weiter oben gesehen, als wir **Item** einmal eine Zahl übergeben haben und einmal einen Dateinamen.

Zu den meisten Sammlungen gibt es einen passenden Objekttyp für deren einzelne Elemente. Bei der Sammlung **Windows** ist beispielsweise ein einzelner Eintrag, der von **Item** zurückgeliefert wird, vom Typ **Window** – man beachte den Singular! Ein Element der **Documents**-Sammlung heißt **Document**, ein Element der **RecentFiles**-Sammlung eben **RecentFile**.

Eleganter Zugriff auf Sammlungen: For Each ... Next

Eine elegantere Methode, hintereinander auf alle Einträge einer Sammlung zuzugreifen, sei hier ebenfalls beschrieben: BasicMaker unterstützt auch die **For Each**-Anweisung:

```
' Namen aller geöffneten Dokumente ausgeben

Dim x As Object

For Each x In tm.Application.Documents
   MsgBox x.Name
Next x
```

Das ist gleichbedeutend mit der oben vorgestellten Schreibweise:

```
For i=1 To tm.Application.Documents.Count
   MsgBox tm.Application.Documents.Item(i).Name
Next i
```

Eigene Eigenschaften und Methoden von Sammlungen

Sammlungen besitzen neben **Item** und **Count** gegebenenfalls eigene Eigenschaften und Methoden, mit denen die jeweilige Sammlung verwaltet werden kann. Möchten Sie beispielsweise in TextMaker ein leeres Dokument anlegen, so bedeutet dieser Vorgang für BasicMaker, dass Sie der **Documents**-Sammlung einen neuen Eintrag hinzufügen:

```
tm.Application.Documents.Add   ' leeres Dokument anlegen
```

Tipps für die Vereinfachung von Schreibweisen

Wenn Sie sich nun langsam wundern, ob wirklich so viel Tipparbeit nötig ist, um ein einzelnes Dokument anzusprechen, können wir Sie beruhigen: ist es nicht! Es gibt diverse Abkürzungen, die Ihnen viel Zeit ersparen.

Verwenden der With-Anweisung

Die erste Abkürzung ist, dass Sie zum Zugriff auf *mehrere* Eigenschaften eines Objekts die **With**-Anweisung verwenden können.

Zunächst die herkömmliche Schreibweise:

```
tm.Application.Left = 100
tm.Application.Top = 50
tm.Application.Width = 500
tm.Application.Height = 300
tm.Application.Options.CreateBackup = True
MsgBox tm.Application.ActiveDocument.Name
```

Dieser Code sieht bei Verwendung der **With**-Anweisung wesentlich übersichtlicher aus:

```
With tm.Application
   .Left = 100
   .Top = 50
   .Width = 500
   .Height = 300
   .Options.CreateBackup = True
   MsgBox .ActiveDocument.Name
End With
```

Objektvariablen einrichten

Die nächste Arbeitsvereinfachung ist, dass Sie sich eigene Objektvariablen für den schnellen Zugriff einrichten können. Vergleichen Sie folgende Anweisungen:

Umständlich:

```
Sub Kompliziert
   Dim tm As Object
   Set tm = CreateObject("TextMaker.Application")
   tm.Application.Visible = True ' TextMaker sichtbar machen
   tm.Application.Documents.Add ' Dokument hinzufügen
   tm.Application.ActiveDocument.Left = 100
   tm.Application.ActiveDocument.Top = 50
   tm.Application.ActiveDocument.Width = 222
   tm.Application.ActiveDocument.Height = 80
End Sub
```

Einfacher:

```
Sub Besser
   Dim tm As Object
   Dim NeuesDokument As Object
   Set tm = CreateObject("TextMaker.Application")
   tm.Application.Visible = True ' TextMaker sichtbar machen
   NeuesDokument = tm.Application.Documents.Add ' Dokument hinzufügen
   NeuesDokument.Left = 100
   NeuesDokument.Top = 50
   NeuesDokument.Width = 222
   NeuesDokument.Height = 80
End Sub
```

Nachdem Sie im unteren Beispiel in der Objektvariablen "NeuesDokument" eine Referenz auf das Dokument angelegt haben (die von der **Add**-Methode der Sammlung **Documents** praktischerweise zurückgegeben wird), können Sie über diese Objektvariable viel handlicher auf das neue Dokument zugreifen.

Weglassen von Standardeigenschaften

Es geht in vielen Fällen noch einfacher: Jedes Objekt (zum Beispiel **Application** oder **Application.Documents**) besitzt unter seinen Eigenschaften jeweils eine Eigenschaft, die als *Standardeigenschaft* markiert ist. Das Praktische daran ist, dass Sie sich dadurch nochmals Tipparbeit ersparen können, denn die Standardeigenschaft kann einfach weglassen werden.

Die Standardeigenschaft von **Application** ist beispielsweise **Name**. Folgende beiden Befehle sind daher gleichbedeutend:

```
MsgBox tm.Application.Name   ' gibt den Namen von TextMaker aus

MsgBox tm.Application        ' tut dasselbe
```

Typischerweise ist die am häufigsten benötigte Eigenschaft eines Objekts als Standardeigenschaft markiert. So ist sicherlich die am häufigsten benötigte Eigenschaft einer Sammlung die **Item**-Eigenschaft. Denn im Allgemeinen will man ja auf ein bestimmtes Element einer Sammlung zugreifen. Folgende Anweisungen sind daher wieder gleichbedeutend:

```
MsgBox tm.Application.Documents.Item(1).Name

MsgBox tm.Application.Documents(1).Name
```

So wird das Ganze doch langsam übersichtlicher! Es kommt aber noch besser: **Name** ist die Standardeigenschaft eines einzelnen **Document**-Objekts (aufgepasst: "Document", nicht "Documents"!). Jedes **Item** der **Documents**-Sammlung ist vom Typ **Document**. Da also **Name** die Standardeigenschaft ist, können Sie **Name** wieder weglassen:

```
MsgBox tm.Application.Documents(1)
```

Immer noch nicht einfach genug? Also... **Application** ist die Standardeigenschaft von TextMaker an sich. Lassen wir **Application** also einfach weg! Das sieht dann so aus:

```
MsgBox tm.Documents(1)
```

Mit diesem Grundwissen sind Sie nun gerüstet, um die Objektstruktur von TextMaker zu verstehen und können sich dem Abschnitt "Objektstruktur von TextMaker" widmen, der eine detaillierte Liste aller von TextMaker bereitgestellten Objekte enthält.

Objektstruktur von TextMaker

TextMaker stellt BasicMaker und anderen OLE Automation-fähigen Programmiersprachen die im Folgenden aufgelisteten Objekte zur Verfügung.

Hinweise:

- Mit "R/O" gekennzeichnete Eigenschaften sind "Read Only" (also schreibgeschützt). Sie können zwar ausgelesen, aber nicht verändert werden.
- Die Default-Eigenschaft eines Objekts ist durch *Kursivschrift* gekennzeichnet.

Die folgende Tabelle führt alle in TextMaker verfügbaren Objekte und Sammlungen auf.

Name	Typ	Beschreibung
Application	Objekt	"Wurzelobjekt" von TextMaker
Options	Objekt	Globale Einstellungen
UserProperties	Sammlung	Sammlung aller Bestandteile der privaten und geschäftlichen Adresse
UserProperty	Objekt	Ein einzelner Bestandteil der Adresse
CommandBars	Sammlung	Sammlung aller Symbolleisten
CommandBar	Objekt	Eine einzelne Symbolleiste
AutoCorrect	Objekt	Automatische Textkorrektur und Textbausteine
AutoCorrectEntries	Sammlung	Sammlung aller Textbausteine

Name	Typ	Beschreibung
AutoCorrectEntry	Objekt	Ein einzelner Textbaustein
Documents	Sammlung	Sammlung aller geöffneten Dokumente
Document	Objekt	Ein einzelnes geöffnetes Dokument
DocumentProperties	Sammlung	Sammlung aller Dokumenteigenschaften eines Dokuments
DocumentProperty	Objekt	Eine einzelne Dokumenteigenschaft
PageSetup	Objekt	Die Seiteneinstellungen eines Dokuments
Selection	Objekt	Die Selektion oder Schreibmarke in einem Dokument
Font	Objekt	Die zur Selektion gehörende Zeichenformatierung
Paragraphs	Sammlung	Sammlung aller Absätze eines Dokuments
Paragraph	Objekt	Ein einzelner Absatz
Range	Objekt	Start- und Endposition eines Absatzes im Dokument
DropCap	Objekt	Der Initialbuchstabe eines Absatzes
Tables	Sammlung	Sammlung aller Tabellen eines Dokuments
Table	Objekt	Eine einzelne Tabelle
Rows	Sammlung	Sammlung aller Tabellenzeilen einer Tabelle
Row	Objekt	Eine einzelne Tabellenzeile
Cells	Sammlung	Sammlung aller Zellen einer Tabellenzeile
Cell	Objekt	Eine einzelne Tabellenzelle
Borders	Sammlung	Sammlung aller Umrandungslinien (links, rechts, oben, unten etc.) eines Absatzes, einer Tabelle, einer Tabellenzeile oder Tabellenzelle
Border	Objekt	Eine einzelne Umrandungslinie
Shading	Objekt	Die Schattierung von Absätzen, Tabellen, Tabellenzeilen und Tabellenzellen
FormFields	Sammlung	Sammlung aller Formularobjekte eines Dokuments
FormField	Objekt	Ein einzelnes Formularobjekt
TextInput	Objekt	Ein einzelnes Formularobjekt, betrachtet als Textfeld
CheckBox	Objekt	Ein einzelnes Formularobjekt, betrachtet als Kontrollkästchen
DropDown	Objekt	Ein einzelnes Formularobjekt, betrachtet als Auswahlliste
ListEntries	Sammlung	Sammlung aller Einträge einer Auswahlliste
ListEntry	Objekt	Ein einzelner Eintrag einer Auswahlliste
Windows	Sammlung	Sammlung aller geöffneten Dokumentfenster
Window	Objekt	Ein einzelnes geöffnetes Dokumentfenster
View	Objekt	Einstellungen zur Darstellung eines Dokumentfensters
Zoom	Objekt	Die Vergrößerungsstufe eines Dokumentfensters
RecentFiles	Sammlung	Sammlung aller im Datei-Menü aufgeführten zuletzt geöffneten Dateien
RecentFile	Objekt	Eine einzelne der im Datei-Menü angezeigten zuletzt geöffneten Dateien
FontNames	Sammlung	Sammlung aller installierten Schriftarten
FontName	Objekt	Eine einzelne installierte Schriftart

Im Anschluss werden alle Objekte und Sammlungen im Detail beschrieben.

Application (Objekt)

Zugriffspfad: **Application**

1 Beschreibung

Application ist das "Wurzelobjekt" aller anderen Objekte in TextMaker. Es ist das zentrale Steuerobjekt, über das die gesamte Kommunikation zwischen Ihrem Basic-Script und TextMaker abgewickelt wird.

2 Zugriff auf das Objekt

Es existiert genau eine Instanz des **Application**-Objekts. Diese ist während der gesamten Laufzeit von TextMaker verfügbar und wird direkt über die von **CreateObject** zurückgegebene Objektvariable angesprochen:

```
Set tm = CreateObject("TextMaker.Application")
MsgBox tm.Application.Name
```

Da **Application** die Defaulteigenschaft von TextMaker ist, kann es generell weggelassen werden:

```
Set tm = CreateObject("TextMaker.Application")
MsgBox tm.Name ' gleichbedeutend mit tm.Application.Name
```

3 Eigenschaften, Objekte, Sammlungen und Methoden

Eigenschaften:

- **FullName** R/O
- ***Name*** R/O (Defaulteigenschaft)
- **Path** R/O
- **Build** R/O
- **Bits** R/O
- **Visible**
- **Caption** R/O
- **Left**
- **Top**
- **Width**
- **Height**
- **WindowState**
- **DisplayScrollBars**

Objekte:

- **ActiveDocument → Document**
- **ActiveWindow → Window**
- **Options → Options**
- **UserProperties → UserProperties**
- **CommandBars → CommandBars**
- **AutoCorrect → AutoCorrect**
- **Application → Application**

Sammlungen:

- **Documents → Documents**
- **Windows → Windows**
- **RecentFiles → RecentFiles**
- **FontNames → FontNames**

Methoden:

- **CentimetersToPoints**
- **MillimetersToPoints**

- **InchesToPoints**
- **PicasToPoints**
- **LinesToPoints**
- **Activate**
- **Quit**

FullName (Eigenschaft, R/O)

Datentyp: **String**

Liefert Namen und Pfad des Programms (z.B. "C:\Programme\SoftMaker Office\TextMaker.exe").

Name (Eigenschaft, R/O)

Datentyp: **String**

Liefert den Namen des Programms (in diesem Fall also "TextMaker").

Path (Eigenschaft, R/O)

Datentyp: **String**

Liefert den Pfad des Programms, zum Beispiel "C:\Programme\SoftMaker Office\".

Build (Eigenschaft, R/O)

Datentyp: **String**

Liefert die Revisionsnummer des Programms als Zeichenkette, zum Beispiel "460".

Bits (Eigenschaft, R/O)

Datentyp: **String**

Liefert eine Zeichenkette, die der Bit-Version des Programms entspricht: "16" bei der 16 Bit-Version, "32" bei der 32 Bit-Version von TextMaker.

Visible (Eigenschaft)

Datentyp: **Boolean**

Liest oder setzt die Sichtbarkeit des Programmfensters:

```
tm.Application.Visible = True  ' TextMaker wird sichtbar
tm.Application.Visible = False ' TextMaker wird unsichtbar
```

Wichtig: Standardmäßig ist **Visible** auf **False** gesetzt – TextMaker startet also unsichtbar, bis Sie ihn explizit sichtbar machen.

Caption (Eigenschaft, R/O)

Datentyp: **String**

Liefert eine Zeichenkette mit dem Inhalt der Titelleiste des Programmfensters (z.B. "TextMaker - Liesmich.tmd").

Left (Eigenschaft)

Datentyp: **Long**

Liest oder setzt die X-Koordinate (= linker Rand) des Programmfensters auf dem Bildschirm. Die Maßeinheit sind Bildschirmpixel.

Top (Eigenschaft)

Datentyp: **Long**

Liest oder setzt die Y-Koordinate (= oberer Rand) des Programmfensters auf dem Bildschirm. Die Maßeinheit sind Bildschirmpixel.

Width (Eigenschaft)

Datentyp: **Long**

Liest oder setzt die Breite des Programmfensters auf dem Bildschirm. Die Maßeinheit sind Bildschirmpixel.

Height (Eigenschaft)

Datentyp: **Long**

Liest oder setzt die Höhe des Programmfensters auf dem Bildschirm. Die Maßeinheit sind Bildschirmpixel.

WindowState (Eigenschaft)

Datentyp: **Long** (SmoWindowState)

Liest oder setzt die Fensterdarstellung des Programmfensters. Mögliche Werte:

```
smoWindowStateNormal   = 1 ' normal
smoWindowStateMinimize = 2 ' minimiert
smoWindowStateMaximize = 3 ' maximiert
```

DisplayScrollBars (Eigenschaft)

Datentyp: **Boolean**

Liest oder setzt die Einstellung von TextMaker, ob bei Dokumenten sowohl der horizontale als auch der vertikale Rollbalken angezeigt werden.

ActiveDocument (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das gerade aktive **Document**-Objekt, über das Sie auf das aktuelle Dokument zugreifen können.

ActiveWindow (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das gerade aktive **Window**-Objekt, über das Sie auf das aktuelle Dokumentfenster zugreifen können.

Options (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das **Options**-Objekt, mit dem Sie auf diverse globale Programmeinstellungen von TextMaker zugreifen können.

UserProperties (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das **UserProperties**-Objekt, mit dem Sie auf den Namen und die Adresse des Anwenders zugreifen können (sofern er dies im Dialog des Befehls **Weiteres > Einstellungen**, Karteikarte **Allgemein**, eingetragen hat).

CommandBars (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das **CommandBars**-Objekt, mit dem Sie auf die Symbolleisten von TextMaker zugreifen können.

AutoCorrect (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das **AutoCorrect**-Objekt, mit dem Sie auf die Autokorrektur-Einstellungen von TextMaker zugreifen können.

Application (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das **Application**-Objekt, also sich selbst. Dieser Objektzeiger ist eigentlich unnötig und nur der Vollständigkeit halber vorhanden.

Documents (Zeiger auf Sammlung)

Datentyp: **Object**

Liefert die **Documents**-Sammlung, eine Sammlung aller momentan geöffneten Dokumente.

Windows (Zeiger auf Sammlung)

Datentyp: **Object**

Liefert die **Windows**-Sammlung, eine Sammlung aller momentan geöffneten Dokumentfenster.

RecentFiles (Zeiger auf Sammlung)

Datentyp: **Object**

Liefert die **RecentFiles**-Sammlung, eine Sammlung der zuletzt geöffneten Dokumente.

FontNames (Zeiger auf Sammlung)

Datentyp: **Object**

Liefert die **FontNames**-Sammlung, eine Sammlung aller installierten Schriftarten.

CentimetersToPoints (Methode)

Konvertiert den angegebenen Wert von Zentimetern (cm) in Punkt (pt). Diese Funktion ist nützlich, wenn Sie mit Zentimetern rechnen, eine TextMaker-Funktion als Maßeinheit aber nur Punkt akzeptiert.

Syntax:

```
CentimetersToPoints(Centimeters)
```

Parameter:

Centimeters (Typ: **Single**) gibt den umzurechnenden Wert an.

Rückgabetyp:

Single

Beispiel:

```
' Den oberen Rand des aktiven Dokuments auf 3 Zentimeter setzen
tm.ActiveDocument.PageSetup.TopMargin = tm.Application.CentimetersToPoints(3)
```

MillimetersToPoints (Methode)

Konvertiert den angegebenen Wert von Millimetern (mm) in Punkt (pt). Diese Funktion ist nützlich, wenn Sie mit Millimetern rechnen, eine TextMaker-Funktion als Maßeinheit aber nur Punkt akzeptiert.

Syntax:

```
MillimetersToPoints(Millimeters)
```

Parameter:

Millimeters (Typ: **Single**) gibt den umzurechnenden Wert an.

Rückgabetyp:

Single

Beispiel:

```
' Den oberen Rand des aktiven Dokuments auf 30 Millimeter setzen
tm.ActiveDocument.PageSetup.TopMargin = tm.Application.MillimetersToPoints(30)
```

InchesToPoints (Methode)

Konvertiert den angegebenen Wert von Zoll (Inch) in Punkt (pt). Diese Funktion ist nützlich, wenn Sie mit Zoll rechnen, eine TextMaker-Funktion als Maßeinheit aber nur Punkt akzeptiert.

Syntax:

```
InchesToPoints(Inches)
```

Parameter:

Inches (Typ: **Single**) gibt den umzurechnenden Wert an.

Rückgabetyp:

Single

Beispiel:

```
' Den unteren Rand des aktiven Dokuments auf 1 Zoll setzen
tm.ActiveDocument.PageSetup.BottomMargin = tm.Application.InchesToPoints(1)
```

PicasToPoints (Methode)

Konvertiert den angegebenen Wert von Pica in Punkt (pt). Diese Funktion ist nützlich, wenn Sie mit Pica rechnen, eine TextMaker-Funktion als Maßeinheit aber nur Punkt akzeptiert.

Syntax:

PicasToPoints(Picas)

Parameter:

Picas (Typ: **Single**) gibt den umzurechnenden Wert an.

Rückgabetyp:

Single

Beispiel:

```
' Den unteren Rand des aktiven Dokuments auf 6 Pica setzen
tm.ActiveDocument.PageSetup.BottomMargin = tm.Application.PicasToPoints(6)
```

LinesToPoints (Methode)

Identisch mit **PicasToPoints** (siehe dort).

Syntax:

LinesToPoints(Lines)

Parameter:

Lines (Typ: **Single**) gibt den umzurechnenden Wert an.

Rückgabetyp:

Single

Beispiel:

```
' Den unteren Rand des aktiven Dokuments auf 6 Pica setzen
tm.ActiveDocument.PageSetup.BottomMargin = tm.Application.LinesToPoints(6)
```

Activate (Methode)

Bringt das Programmfenster in den Vordergrund und setzt den Fokus darauf.

Syntax:

Activate

Parameter:

keine

Rückgabetyp:

keiner

Beispiel:

```
' TextMaker in den Vordergrund holen
tm.Application.Activate
```

Hinweis: Damit die Funktion erfolgreich ausgeführt werden kann, muss **Application.Visible** = **True** sein.

Quit (Methode)

Beendet das Programm.

Syntax:

```
Quit
```

Parameter:

keine

Rückgabetyp:

keiner

Beispiel:

```
' TextMaker beenden
tm.Application.Quit
```

Sind noch ungespeicherte Dokumente geöffnet, wird der Benutzer gefragt, ob diese gespeichert werden sollen. Wenn Sie diese Frage vermeiden wollen, sollten Sie entweder alle offenen Dokumente von Ihrem Programm aus schließen oder bei diesen Dokumenten die Eigenschaft **Saved** auf **True** setzen (siehe **Document**).

Options (Objekt)

Zugriffspfad: Application → **Options**

1 Beschreibung

Im **Options**-Objekt sind diverse globale Programmeinstellungen zusammengefasst, von denen Sie in TextMaker die meisten im Dialogfenster **Weiteres** > **Einstellungen** finden.

2 Zugriff auf das Objekt

Es existiert während der gesamten Laufzeit von TextMaker genau eine Instanz des **Options**-Objekts. Diese wird über **Application.Options** angesprochen:

```
Set tm = CreateObject("TextMaker.Application")
tm.Application.Options.EnableSound = True
```

3 Eigenschaften, Objekte, Sammlungen und Methoden

Eigenschaften:

- **AutoFormatReplaceQuotes**
- **CheckSpellingAsYouType**
- **ShowSpellingErrors**
- **ShowGermanSpellingReformErrors**
- **CreateBackup**
- **DefaultFilePath**
- **DefaultTemplatePath**
- **EnableSound**
- **Overtype**
- **SaveInterval**
- **SavePropertiesPrompt**
- **AutoWordSelection**
- **PasteAdjustWordSpacing**
- **TabIndentKey**

- **DefaultFileFormat**

Objekte:
- **Application → Application**
- ***Parent* → Application**

AutoFormatReplaceQuotes (Eigenschaft)

Datentyp: **Long** (SmoQuotesStyle)

Liest oder setzt die Einstellung, ob beim Tippen neutrale Anführungszeichen automatisch in typographische Anführungszeichen gewandelt werden sollen. Mögliche Werte:

```
smoQuotesNeutral  = 0 ' Neutral = aus
smoQuotesGerman   = 1 ' Deutsch
smoQuotesSwiss    = 2 ' Schweizerisches Deutsch
smoQuotesEnglish  = 3 ' Englisch
smoQuotesFrench   = 4 ' Französisch
smoQuotesAuto     = 5 ' Automatisch nach der Sprache
```

CheckSpellingAsYouType (Eigenschaft)

Datentyp: **Boolean**

Liest oder setzt die Einstellung "Rechtschreibprüfung im Hintergrund" (**True** oder **False**).

ShowSpellingErrors (Eigenschaft)

Datentyp: **Boolean**

Liest oder setzt die Einstellung "Tippfehler rot unterstreichen" (**True** oder **False**).

ShowGermanSpellingReformErrors (Eigenschaft)

Datentyp: **Boolean**

Liest oder setzt die Einstellung "Alte Schreibweisen blau unterstreichen" (**True** oder **False**).

CreateBackup (Eigenschaft)

Datentyp: **Boolean**

Liest oder setzt die Einstellung ".BAK-Dateien anlegen" (**True** oder **False**).

DefaultFilePath (Eigenschaft)

Datentyp: **String**

Liest oder setzt den Dateipfad, unter dem Dokumente standardmäßig gespeichert und geöffnet werden.

Dies ist lediglich eine temporäre Einstellung: Beim nächsten Aufruf von **Datei > Öffnen** oder **Datei > Speichern unter** erscheint der hier gewählte Pfad. Ändert der Benutzer diesen aber ab, wird ab diesem Zeitpunkt der vom Benutzer ausgewählte Pfad voreingestellt.

DefaultTemplatePath (Eigenschaft)

Datentyp: **String**

Liest oder setzt den Dateipfad, unter dem Dokumentvorlagen standardmäßig gespeichert werden.

Diese Einstellung wird dauerhaft gespeichert. Bei jedem Aufruf von **Datei > Neu** erscheinen die Dokumentvorlagen im hier angegebenen Pfad.

EnableSound (Eigenschaft)

Datentyp: **Boolean**

Liest oder setzt die Einstellung "Warnton bei Meldungen" (**True** oder **False**).

Overtype (Eigenschaft)

Datentyp: **Boolean**

Liest oder setzt den Überschreibe-/Einfügemodus (**True**=Überschreiben, **False**=Einfügen).

SaveInterval (Eigenschaft)

Datentyp: **Long**

Liest oder setzt die Einstellung "Automatisches Sichern alle *n* Minuten" (0=aus).

SavePropertiesPrompt (Eigenschaft)

Datentyp: **Boolean**

Liest oder setzt die Einstellung "Beim Speichern nach Dokumentinfo fragen" (**True** oder **False**).

AutoWordSelection (Eigenschaft)

Datentyp: **Boolean**

Liest oder setzt die Einstellung "Beim Markieren immer ganzes Wort markieren" (**True** oder **False**).

PasteAdjustWordSpacing (Eigenschaft)

Datentyp: **Boolean**

Liest oder setzt die Einstellung "Leerzeichen beim Einfügen hinzufügen oder löschen" (**True** oder **False**).

TabIndentKey (Eigenschaft)

Datentyp: **Boolean**

Liest oder setzt die Einstellung "Linken Einzug und Erstzeileneinzug mit Tabulator- und Rückschritt-Taste setzen" (**True** oder **False**).

DefaultFileFormat (Eigenschaft)

Datentyp: **Long** (TmDefaultFileFormat)

Liest oder setzt das Standarddateiformat, in dem TextMaker neu angelegte Dokumente standardmäßig speichert. Mögliche Werte:

```
tmDefaultFileFormatTextMaker     = 0 ' TextMaker (.tmd)
tmDefaultFileFormatWinWordXP     = 1 ' Microsoft Word XP/2003 (.doc)
tmDefaultFileFormatWinWord97     = 2 ' Microsoft Word 97/2000 (.doc)
tmDefaultFileFormatWinWord6      = 3 ' Microsoft Word 6.0/95 (.doc)
tmDefaultFileFormatOpenDoc       = 4 ' OpenDocument (.odt)
tmDefaultFileFormatRTF           = 5 ' RTF Rich Text Format (.rtf)
```

Application (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das **Application**-Objekt.

Parent (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das übergeordnete Objekt, also **Application**.

UserProperties (Sammlung)

Zugriffspfad: Application → **UserProperties**

1 Beschreibung

Die Sammlung **UserProperties** enthält die Privat- und Geschäftsadresse des Benutzers (sofern er dies im Dialog des Befehls **Weiteres > Einstellungen**, Karteikarte **Allgemein**, eingetragen hat).

Die einzelnen Elemente dieser Sammlung sind vom Typ **UserProperty**.

2 Zugriff auf die Sammlung

Es existiert während der gesamten Laufzeit von TextMaker genau eine Instanz der **UserProperties**-Sammlung. Diese wird über **Application.UserProperties** angesprochen:

```
' Zeige die erste UserProperty (den Nachnamen des Benutzers) an
MsgBox tm.Application.UserProperties.Item(1).Value
```

3 Eigenschaften, Objekte, Sammlungen und Methoden

Eigenschaften:
- **Count** R/O

Objekte:
- ***Item*** → **UserProperty** (Defaultobjekt)
- **Application** → **Application**
- **Parent** → **Application**

Count (Eigenschaft, R/O)

Datentyp: **Long**

Liefert die Anzahl der **UserProperty**-Objekte in der Sammlung, also die Zahl aller Adressbestandteile (Nachname, Vorname, Straße etc. – jeweils für Privat- und Geschäftsadresse).

Dieser Wert ist konstant 24, da es genau 24 solche Elemente gibt.

Item (Zeiger auf Objekt)

Datentyp: **Object**

Liefert ein einzelnes **UserProperty**-Objekt, mit dem Sie einen einzelnen Adressbestandteil (Nachname, Vorname, Straße etc.) der privaten oder geschäftlichen Adresse des Benutzers lesen oder setzen können.

Welches UserProperty-Objekt Sie erhalten, hängt von dem Zahlenwert ab, den Sie an **Item** übergeben. Die folgende Tabelle zeigt die erlaubten Werte:

```
smoUserHomeAddressName             = 1  ' Nachname (privat)
smoUserHomeAddressFirstName        = 2  ' Vorname (privat)
smoUserHomeAddressStreet           = 3  ' Straße (privat)
smoUserHomeAddressZip              = 4  ' Postleitzahl (privat)
smoUserHomeAddressCity             = 5  ' Stadt (privat)
smoUserHomeAddressPhone1           = 6  ' Telefon (privat)
smoUserHomeAddressFax              = 7  ' Telefax (privat)
smoUserHomeAddressEmail            = 8  ' E-Mail-Adresse (privat)
smoUserHomeAddressPhone2           = 9  ' Mobiltelefon o.ä. (privat)
smoUserHomeAddressHomepage         = 10 ' Homepage (privat)

smoUserBusinessAddressName         = 11 ' Nachname (geschäftlich)
smoUserBusinessAddressFirstName    = 12 ' Vorname (geschäftlich)
smoUserBusinessAddressCompany      = 13 ' Firma (geschäftlich)
smoUserBusinessAddressDepartment   = 14 ' Abteilung (geschäftlich)
smoUserBusinessAddressStreet       = 15 ' Straße (geschäftlich)
smoUserBusinessAddressZip          = 16 ' Postleitzahl (geschäftlich)
smoUserBusinessAddressCity         = 17 ' Stadt (geschäftlich)
smoUserBusinessAddressPhone1       = 18 ' Telefon (geschäftlich)
smoUserBusinessAddressFax          = 19 ' Telefax (geschäftlich)
smoUserBusinessAddressEmail        = 20 ' E-Mail-Adresse (geschäftlich)
smoUserBusinessAddressPhone2       = 21 ' Mobiltelefon o.ä. (geschäftlich)
smoUserBusinessAddressHomepage     = 22 ' Homepage (geschäftlich)

smoUserHomeAddressInitials         = 23 ' Initialen des Benutzers (privat)
smoUserBusinessAddressInitials     = 24 ' Initialen des Benutzers (geschäftlich)
```

Beispiele:

```
' Den Nachnamen des Benutzers (privat) anzeigen
MsgBox tm.Application.UserProperties.Item(1).Value

' Die geschäftliche E-Mail-Adresse auf test@example.com ändern
With tm.Application
  .UserProperties.Item(smoUserBusinessAddressEmail).Value = "test@example.com"
End With
```

Application (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das **Application**-Objekt.

Parent (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das übergeordnete Objekt, also **Application**.

UserProperty (Objekt)

Zugriffspfad: Application → UserProperties → **Item**

1 Beschreibung

Ein **UserProperty**-Objekt repräsentiert einen einzelnen Teil (zum Beispiel Straße oder Postleitzahl) der vom Benutzer eingegebenen privaten und geschäftlichen Adresse.

Für jeden dieser Bestandteile existiert ein eigenes **UserProperty**-Objekt. Die Zahl dieser Objekte ist konstant, da Sie zwar die einzelnen Adressbestandteile bearbeiten, nicht aber neue anlegen können.

2 Zugriff auf das Objekt

Die einzelnen **UserProperty**-Objekte können ausschließlich durch Aufzählung der Elemente der Sammlung **Application.UserProperties** angesprochen werden. Der Typ dieser Sammlung ist **UserProperties**.

Beispiel:

```
' Den Inhalt des ersten Adressbestandteils (Nachname privat) anzeigen
MsgBox tm.Application.UserProperties.Item(1).Value
```

3 Eigenschaften, Objekte, Sammlungen und Methoden

Eigenschaften:
- ***Value*** (Defaulteigenschaft)

Objekte:
- **Application → Application**
- **Parent → UserProperties**

Value (Eigenschaft)

Datentyp: **String**

Liest oder setzt den Inhalt des Adressbestandteils. Das folgende Beispiel setzt den Firmennamen des Benutzers:

```
Sub Beispiel()
  Set tm = CreateObject("TextMaker.Application")
  tm.UserProperties(smoUserBusinessAddressCompany).Value = "ACME Corp."
End Sub
```

Application (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das **Application**-Objekt.

Parent (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das übergeordnete Objekt, also **UserProperties**.

CommandBars (Sammlung)

Zugriffspfad: Application → **CommandBars**

1 Beschreibung

Die Sammlung **CommandBars** enthält alle Symbolleisten von TextMaker. Die einzelnen Elemente dieser Sammlung sind vom Typ **CommandBar**.

2 Zugriff auf die Sammlung

Es existiert während der gesamten Laufzeit von TextMaker genau eine Instanz der **CommandBars**-Sammlung. Diese wird über **Application.CommandBars** angesprochen:

```
' Zeige den Namen der ersten Symbolleiste von TextMaker an
MsgBox tm.Application.CommandBars.Item(1).Name

' Dasselbe einfacher durch Nutzung der Defaulteigenschaft
MsgBox tm.CommandBars(1)
```

3 Eigenschaften, Objekte, Sammlungen und Methoden

Eigenschaften:
- **Count** R/O
- **DisplayFonts**
- **DisplayTooltips**

Objekte:
- ***Item*** → **CommandBar** (Defaultobjekt)
- **Application** → **Application**
- **Parent** → **Application**

Count (Eigenschaft, R/O)

Datentyp: **Long**

Liefert die Anzahl der **CommandBar**-Objekte in der Sammlung, also die Zahl aller verfügbaren Symbolleisten.

DisplayFonts (Eigenschaft)

Datentyp: **Boolean**

Liest oder setzt die Einstellung "Schriftenliste mit echten Schriften" (**True** oder **False**).

DisplayTooltips (Eigenschaft)

Datentyp: **Boolean**

Liest oder setzt die Einstellung von TextMaker, ob QuickInfos (Tooltips) angezeigt werden, wenn die Maus über eine Schaltfläche in den Symbolleisten bewegt wird.

Entspricht der Einstellung "QuickInfos" im Dialogfenster des Befehls **Weiteres > Einstellungen**.

Item (Zeiger auf Objekt)

Datentyp: **Object**

Liefert ein einzelnes **CommandBar**-Objekt, mit dem Sie auf eine einzelne Symbolleiste von TextMaker zugreifen können.

Welches CommandBar-Objekt Sie erhalten, hängt von dem Parameter ab, den Sie an **Item** übergeben. Dies kann entweder der numerische Index oder der Name der gewünschten Symbolleiste sein. Beispiele:

```
' Mache die erste Symbolleiste unsichtbar
tm.Application.CommandBars.Item(1).Visible = False

' Mache die Symbolleiste "Formatleiste" unsichtbar
tm.Application.CommandBars.Item("Format").Visible = False
```

Sie sollten aber *Namen* von Symbolleisten nicht fest in Ihr Script eintragen, da sich diese Namen an die Sprache der gewählten Benutzeroberfläche von TextMaker anpassen. Betreiben Sie TextMaker beispielsweise in englischer Sprache, heißt die Formatleiste nicht "Format" sondern "Formatting".

Daher ist es empfehlenswerter, die folgenden symbolischen Konstanten für die Symbolleisten zu verwenden:

```
tmBarStatusShort        = 1  ' Statusleiste (ohne geöffnete Dokumente)
tmBarStandardShort      = 2  ' Funktionsleiste (ohne geöffnete Dokumente)
tmBarStatus             = 3  ' Statusleiste
tmBarStandard           = 4  ' Funktionsleiste
tmBarFormatting         = 5  ' Formatleiste
tmBarOutliner           = 6  ' Gliederungsleiste
tmBarObjects            = 7  ' Objektleiste
tmBarFormsEditing       = 8  ' Formularleiste
tmBarMailMerge          = 9  ' Serienbriefleiste
tmBarDatabase           = 10 ' Funktionsleiste im Datenbankfenster
tmBarDatabaseStatus     = 11 ' Statusleiste im Datenbankfenster
tmBarPicture            = 12 ' Grafikleiste
tmBarReviewing          = 13 ' Überarbeiten-Leiste
tmBarHeaderAndFooter    = 14 ' Kopf- und Fußzeilenleiste
tmBarFullscreen         = 15 ' Vollbildleiste
tmBarTable              = 16 ' Tabellenleiste
```

Application (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das **Application**-Objekt.

Parent (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das übergeordnete Objekt, also **Application**.

CommandBar (Objekt)

Zugriffspfad: Application → CommandBars → **Item**

1 Beschreibung

Ein **CommandBar**-Objekt repräsentiert eine einzelne Symbolleiste von TextMaker.

Für jede Symbolleiste existiert ein eigenes **CommandBar**-Objekt. Richten Sie neue Symbolleisten ein oder löschen diese, werden die zugehörigen **CommandBar**-Objekte dynamisch angelegt bzw. entfernt.

2 Zugriff auf das Objekt

Die einzelnen **CommandBar**-Objekte können ausschließlich durch Aufzählung der Elemente der Sammlung **Application.CommandBars** angesprochen werden. Der Typ dieser Sammlung ist **CommandBars**.

Beispiel:

```
' Zeige den Namen der ersten Symbolleiste von TextMaker an
MsgBox tm.Application.CommandBars.Item(1).Name

' Dasselbe einfacher durch Nutzung der Defaulteigenschaft
MsgBox tm.CommandBars(1)
```

3 Eigenschaften, Objekte, Sammlungen und Methoden

Eigenschaften:
- ***Name*** (Defaulteigenschaft)
- **Visible**

Objekte:
- **Application → Application**
- **Parent → CommandBars**

Name (Eigenschaft)

Datentyp: **String**

Liest oder setzt den Namen der Symbolleiste.

Beispiel:

```
' Zeige den Namen der ersten Symbolleiste an
MsgBox tm.Application.CommandBars.Item(1).Name
```

Visible (Eigenschaft)

Datentyp: **Boolean**

Liest oder setzt die Sichtbarkeit der Symbolleiste. Das folgende Beispiel macht die Formatleiste unsichtbar:

```
Sub Beispiel()
  Set tm = CreateObject("TextMaker.Application")
  tm.Application.CommandBars.Item("Format").Visible = False
End Sub
```

Application (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das **Application**-Objekt.

Parent (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das übergeordnete Objekt, also **CommandBars**.

AutoCorrect (Objekt)

Zugriffspfad: Application → **AutoCorrect**

1 Beschreibung

Das **AutoCorrect**-Objekt lässt Sie Einstellungen an der automatischen Textkorrektur von TextMaker vornehmen und die Textbausteine bearbeiten.

2 Zugriff auf das Objekt

Es existiert während der gesamten Laufzeit von TextMaker genau eine Instanz des **AutoCorrect**-Objekts. Diese wird über **Application.AutoCorrect** angesprochen:

```
Set tm = CreateObject("TextMaker.Application")
tm.Application.AutoCorrect.CorrectInitialCaps = True
```

3 Eigenschaften, Objekte, Sammlungen und Methoden

Eigenschaften:
- **CorrectInitialCaps**
- **CorrectSentenceCaps**
- **ReplaceText**

Objekte:
- **Application → Application**
- **Parent → Application**

Sammlungen:
- **Entries → AutoCorrectEntries**

CorrectInitialCaps (Eigenschaft)

Datentyp: **Boolean**

Liest oder setzt die Einstellung "Zwei Großbuchstaben am Anfang eines Wortes korrigieren".

Wenn diese Eigenschaft **True** ist, korrigiert TextMaker automatisch doppelte Großbuchstaben am Wortanfang (zum Beispiel wird "HEnry" in "Henry" geändert).

CorrectSentenceCaps (Eigenschaft)

Datentyp: **Boolean**

Liest oder setzt die Einstellung "Ersten Buchstaben eines Satzes groß schreiben".

Wenn diese Eigenschaft **True** ist, ändert TextMaker am Satzanfang versehentlich in Kleinschreibung eingegebene Buchstaben automatisch in Großschreibung.

ReplaceText (Eigenschaft)

Datentyp: **Boolean**

Liest oder setzt die Einstellung "Textbausteine automatisch ersetzen".

Wenn diese Eigenschaft **True** ist, werden im Text eingegebene Textbaustein-Namen automatisch durch den jeweiligen Textbaustein ausgetauscht (zum Beispiel: Sie tippen "mfg", und TextMaker expandiert dies automatisch zu "Mit freundlichen Grüßen").

Application (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das **Application**-Objekt.

Parent (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das übergeordnete Objekt, also **Application**.

Entries (Zeiger auf Sammlung)

Datentyp: **Object**

Liefert die **AutoCorrectEntries**-Sammlung, die alle Textbausteine von TextMaker enthält.

AutoCorrectEntries (Sammlung)

Zugriffspfad: Application → AutoCorrect → **Entries**

1 Beschreibung

Die Sammlung **AutoCorrectEntries** enthält alle in TextMaker definierten Textbausteine. Die einzelnen Elemente dieser Sammlung sind vom Typ **AutoCorrectEntry**.

2 Zugriff auf die Sammlung

Es existiert während der gesamten Laufzeit von TextMaker genau eine Instanz der **AutoCorrectEntries**-Sammlung. Diese wird über **Application.AutoCorrect.Entries** angesprochen:

```
Set tm = CreateObject("TextMaker.Application")
tm.Application.AutoCorrect.Entries.Add "lax", "Los Angeles"
```

3 Eigenschaften, Objekte, Sammlungen und Methoden

Eigenschaften:
- **Count** R/O

Objekte:
- ***Item*** → **AutoCorrectEntry** (Defaultobjekt)
- **Application** → **Application**
- **Parent** → **AutoCorrect**

Methoden:
- **Add**

Count (Eigenschaft, R/O)

Datentyp: **Long**

Liefert die Anzahl der **AutoCorrectEntry**-Objekte in der Sammlung, also die Zahl der momentan definierten Textbausteine.

Item (Zeiger auf Objekt)

Datentyp: **Object**

Liefert ein einzelnes **AutoCorrectEntry**-Objekt, die Definition eines einzelnen Textbausteins.

Welches AutoCorrect-Objekt Sie erhalten, hängt von dem Parameter ab, den Sie an **Item** übergeben. Dies kann entweder der numerische Index oder der Name des gewünschten Textbausteins sein. Beispiele:

```
' Den Inhalt des ersten definierten Textbausteins anzeigen
MsgBox tm.Application.AutoCorrect.Entries.Item(1).Value

' Den Inhalt des Textbausteins mit dem Namen "mfg" anzeigen
MsgBox tm.Application.AutoCorrect.Entries.Item("mfg").Value
```

Application (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das **Application**-Objekt.

Parent (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das übergeordnete Objekt, also **AutoCorrect**.

Add (Methode)

Fügt einen neuen **AutoCorrectEntry**-Eintrag hinzu.

Syntax:

```
Add Name, Value
```

Parameter:

Name (Typ: **String**): Der gewünschte Name für den neuen Textbaustein. Wenn der Name leer ist oder bereits existiert, schlägt der Aufruf der Methode fehl.

Value (Typ: **String**): Der gewünschte Text für den neuen Textbaustein. Wenn die übergebene Zeichenkette leer ist, schlägt der Aufruf der Methode fehl.

Rückgabetyp:

Object (ein **AutoCorrectEntry**-Objekt, das den neuen Textbaustein repräsentiert)

Beispiel:

```
' Den Baustein "lax" mit dem Inhalt "Los Angeles" belegen
tm.Application.AutoCorrect.Entries.Add "lax", "Los Angeles"
```

AutoCorrectEntry (Objekt)

Zugriffspfad: Application → AutoCorrect → Entries → **Item**

1 Beschreibung

Ein **AutoCorrectEntry**-Objekt repräsentiert einen einzelnen Textbaustein in TextMaker, zum Beispiel "mfg" für "Mit freundlichen Grüßen".

Für jeden Textbaustein existiert ein eigenes **AutoCorrectEntry**-Objekt. Legen Sie Textbausteine an oder löschen diese, werden die zugehörigen **AutoCorrectEntry**-Objekte dynamisch angelegt bzw. entfernt.

2 Zugriff auf das Objekt

Die einzelnen **AutoCorrectEntry**-Objekte können ausschließlich durch Aufzählung der Elemente der Sammlung **Application.AutoCorrect.Entries** angesprochen werden. Der Typ dieser Sammlung ist **AutoCorrectEntries**.

Beispiel:

```
' Den Namen des ersten Textbausteins anzeigen
MsgBox tm.Application.AutoCorrect.Entries.Item(1).Name
```

3 Eigenschaften, Objekte, Sammlungen und Methoden

Eigenschaften:
- ***Name*** (Defaulteigenschaft)
- **Value**

Objekte:
- **Application → Application**
- **Parent → AutoCorrectEntries**

Methoden:
- **Delete**

Name (Eigenschaft)

Datentyp: **String**

Liest oder setzt den Namen des Textbausteins (z.B. "mfg").

Value (Eigenschaft)

Datentyp: **String**

Liest oder setzt den Inhalt des Textbausteins (z.B. "Mit freundlichen Grüßen").

Application (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das **Application**-Objekt.

Parent (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das übergeordnete Objekt, also **AutoCorrectEntries**.

Delete (Methode)

Löscht ein **AutoCorrectEntry**-Objekt aus der **AutoCorrectEntries**-Sammlung.

Syntax:

```
Delete
```

Parameter:

keine

Rückgabetyp:

keiner

Beispiele:

```
' Den ersten Textbaustein löschen
tm.Application.AutoCorrect.Entries.Item(1).Delete

' Den Textbaustein mit dem Namen "mfg" löschen
tm.Application.AutoCorrect.Entries.Item("mfg").Delete
```

Documents (Sammlung)

Zugriffspfad: Application → **Documents**

1 Beschreibung

Die Sammlung **Documents** enthält alle geöffneten Dokumente. Die einzelnen Elemente dieser Sammlung sind vom Typ **Document**.

2 Zugriff auf die Sammlung

Es existiert während der gesamten Laufzeit von TextMaker genau eine Instanz der **Documents**-Sammlung. Diese wird über **Application.Documents** angesprochen:

```
' Die Anzahl der offenen Dokumente anzeigen
MsgBox tm.Application.Documents.Count

' Den Namen des ersten geöffneten Dokuments anzeigen
MsgBox tm.Application.Documents(1).Name
```

3 Eigenschaften, Objekte, Sammlungen und Methoden

Eigenschaften:
- **Count** R/O

Objekte:
- ***Item*** → **Document** (Defaultobjekt)
- **Application** → **Application**

- **Parent → Application**

Methoden:
- **Add**
- **Open**
- **Close**

Count (Eigenschaft, R/O)

Datentyp: **Long**

Liefert die Anzahl der **Document**-Objekte in der Sammlung, also die Zahl der momentan geöffneten Dokumente.

Item (Zeiger auf Objekt)

Datentyp: **Object**

Liefert ein einzelnes **Document**-Objekt, also ein einzelnes geöffnetes Dokument.

Welches Document-Objekt Sie erhalten, hängt von dem Parameter ab, den Sie an **Item** übergeben. Dies kann entweder der numerische Index oder der Dateiname des gewünschten Dokuments sein. Beispiele:

```
' Den Namen des ersten Dokuments anzeigen
MsgBox tm.Application.Documents.Item(1).FullName

' Den Namen des Dokuments "Test.tmd" anzeigen (sofern gerade geöffnet)
MsgBox tm.Application.Documents.Item("Test.tmd").FullName

' Sie können auch den kompletten Pfad angeben
MsgBox tm.Application.Documents.Item("c:\Dokumente\Test.tmd").FullName
```

Application (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das **Application**-Objekt.

Parent (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das übergeordnete Objekt, also **Application**.

Add (Methode)

Legt ein neues leeres Dokument an, wahlweise basierend auf der Standarddokumentvorlage **Normal.tmv** oder einer anderen von Ihnen gewählten Dokumentvorlage.

Syntax:

```
Add [Template]
```

Parameter:

Template (optional; Typ: **String**): Der Pfad und Dateiname der Dokumentvorlage, auf der das Dokument basieren soll. Wird dieser Parameter nicht angegeben, basiert das Dokument auf der Standardvorlage **Normal.tmv**.

Lassen Sie den Pfad weg oder geben nur einen relativen Pfad an, wird automatisch der Standardvorlagenpfad von TextMaker vorangestellt. Lassen Sie die Dateierweiterung **.tmv** weg, wird sie automatisch angehängt.

Rückgabetyp:

Object (ein **Document**-Objekt, das das neue Dokument repräsentiert)

Beispiel:

```
Sub Sample()
  Dim tm as Object
  Dim newDoc as Object

  Set tm = CreateObject("TextMaker.Application")
  tm.Visible = True
  Set newDoc = tm.Documents.Add
  MsgBox newDoc.Name
End Sub
```

Mit dem von **Add** zurückgegebenen **Document**-Objekt können Sie arbeiten wie mit jedem anderen Dokument. Sie können aber auch den Rückgabewert von **Add** ignorieren und sich das neue Dokument beispielsweise über **ActiveDocument** holen.

Open (Methode)

Öffnet ein bestehendes Dokument.

Syntax:

```
Open FileName, [ReadOnly], [Password], [WritePassword], [Format]
```

Parameter:

FileName (Typ: **String**): Pfad und Dateiname des zu öffnenden Dokuments beziehungsweise der zu öffnenden Dokumentvorlage

ReadOnly (optional; Typ: **Boolean**): Gibt an, ob das Dokument nur zum Lesen geöffnet werden soll.

Password (optional; Typ: **String**): Gibt bei kennwortgeschützten Dokumenten das Lesekennwort an. Lassen Sie diesen Parameter bei einem kennwortgeschützten Dokument weg, wird der Benutzer nach dem Lesekennwort gefragt.

WritePassword (optional; Typ: **String**): Gibt bei kennwortgeschützten Dokumenten das Schreibkennwort an. Lassen Sie diesen Parameter bei einem kennwortgeschützten Dokument weg, wird der Benutzer nach dem Schreibkennwort gefragt.

Format (optional; Typ: **Long** bzw. **TmSaveFormat**): Dateiformat des zu öffnenden Dokuments. Mögliche Werte:

```
tmFormatDocument          =  0 ' Dokument, ist der Standardwert
tmFormatTemplate          =  1 ' Dokumentvorlage
tmFormatWinWord97         =  2 ' Microsoft Word für Windows 97 und 2000
tmFormatOpenDocument      =  3 ' OpenDocument, OpenOffice.org, StarOffice
tmFormatRTF               =  4 ' Rich Text Format
tmFormatPocketWordPPC     =  5 ' Pocket Word auf dem Pocket PC
tmFormatPocketWordHPC     =  6 ' Pocket Word auf dem Handheld PC (Windows CE)
tmFormatPlainTextAnsi     =  7 ' Textdatei mit Windows-Zeichensatz
tmFormatPlainTextDOS      =  8 ' Textdatei mit DOS-Zeichensatz
tmFormatPlainTextUnicode  =  9 ' Textdatei mit Unicode-Zeichensatz
tmFormatPlainTextUTF8     = 10 ' Textdatei mit UTF8-Zeichensatz
tmFormatHTML              = 12 ' HTML
tmFormatWinWord6          = 13 ' Microsoft Word für Windows 6.0
tmFormatPlainTextUnix     = 14 ' Textdatei für UNIX, Linux, FreeBSD
tmFormatWinWordXP         = 15 ' Microsoft Word für Windows XP und 2003
```

Wenn Sie diesen Parameter weglassen, wird **tmFormatDocument** angenommen.

Unabhängig vom übergebenen Parameter **FileFormat** versucht TextMaker stets, das Dateiformat selbst zu erkennen, und ignoriert offensichtlich falsche Angaben.

Rückgabetyp:

Object (ein **Document**-Objekt, das das geöffnete Dokument repräsentiert)

Beispiele:

```
' Ein Dokument öffnen
tm.Documents.Open "c:\doks\test.tmd"

' Ein Dokument nur zum Lesen öffnen
tm.Documents.Open "c:\doks\Test.tmd", True
```

Close (Methode)

Schließt alle momentan geöffneten Dokumente.

Syntax:

```
Close [SaveChanges]
```

Parameter:

SaveChanges (optional; Typ: **Long** bzw. **SmoSaveOptions**) gibt an, ob die seit dem letzten Speichern veränderten Dokumente gespeichert werden sollen oder nicht. Lassen Sie den Parameter weg, wird stattdessen gegebenenfalls der Benutzer gefragt. Mögliche Werte:

```
smoDoNotSaveChanges = 0       ' Nicht fragen, nicht speichern
smoPromptToSaveChanges = 1    ' Den Benutzer fragen
smoSaveChanges = 2            ' Ohne Rückfrage speichern
```

Rückgabetyp:

keiner

Beispiel:

```
' Alle offenen Dokumente schließen, ohne sie zu speichern
tm.Documents.Close smoDoNotSaveChanges
```

Document (Objekt)

Zugriffspfade:
- Application → Documents → **Item**
- Application → **ActiveDocument**
- Application → Windows → Item → **Document**
- Application → ActiveWindow → **Document**

1 Beschreibung

Ein **Document**-Objekt repräsentiert ein einzelnes in TextMaker geöffnetes Dokument.

Für jedes Dokument existiert ein eigenes **Document**-Objekt. Öffnen oder schließen Sie Dokumente, werden die zugehörigen **Document**-Objekte dynamisch angelegt bzw. entfernt.

2 Zugriff auf das Objekt

Die einzelnen **Document**-Objekte können auf folgenden Wegen angesprochen werden:

- Alle zu einem Zeitpunkt geöffneten Dokumente werden in der Sammlung **Application.Documents** (Typ: **Documents**) verwaltet:

```
' Die Namen aller geöffneten Dokumente anzeigen
For i = 1 To tm.Application.Documents.Count
```

```
  MsgBox tm.Application.Documents.Item(i).Name
Next i
```

- Das aktive Dokument erhalten Sie über **Application.ActiveDocument**:

```
' Den Namen des aktuellen Dokuments anzeigen
MsgBox tm.Application.ActiveDocument.Name
```

- **Document** ist der **Parent** diverser Objekte, die daran angebunden sind, zum Beispiel **BuiltInDocumentProperties** oder **Selection**:

```
' Den Namen des aktuellen Dokuments über einen Umweg anzeigen
MsgBox tm.Application.ActiveDocument.BuiltInDocumentProperties.Parent.Name
```

- Die Objekte **Window** und **Selection** enthalten Objektzeiger auf das ihnen zugehörige Dokument:

```
' Über das aktive Dokumentfenster an das aktive Dokument kommen
MsgBox tm.Application.ActiveWindow.Document.Name
```

3 Eigenschaften, Objekte, Sammlungen und Methoden

Eigenschaften:
- ***Name*** R/O
- **FullName** R/O
- **Path** R/O
- **PageCount** R/O
- **Saved**
- **ReadOnly**
- **EnableCaretMovement**
- **MergeFileName**
- **MergeFileFormat**
- **MergeFileHeader**
- **MergeRecord**

Objekte:
- **PageSetup → PageSetup**
- **Selection → Selection**
- **ActiveWindow → Window**
- **Application → Application**
- **Parent → Documents**

Sammlungen:
- **BuiltInDocumentProperties → DocumentProperties**
- **Paragraphs → Paragraphs**
- **Tables → Tables**
- **FormFields → FormFields**

Methoden:
- **Activate**
- **Close**
- **Save**
- **SaveAs**
- **Select**
- **MailMerge**
- **PrintOut**
- **MergePrintOut**

Name (Eigenschaft, R/O)

Datentyp: **String**

Liefert den Namen des Dokuments (z.B. Müller.tmd).

FullName (Eigenschaft, R/O)

Datentyp: **String**

Liefert den Pfad und Namen des Dokuments (z.B. c:\Briefe\Müller.tmd).

Path (Eigenschaft, R/O)

Datentyp: **String**

Liefert den Pfad des Dokuments (z.B. c:\Briefe).

PageCount (Eigenschaft, R/O)

Datentyp: **Long**

Liefert die Anzahl der Seiten im Dokument.

Saved (Eigenschaft)

Datentyp: **Boolean**

Liest oder setzt die **Saved**-Eigenschaft des Dokuments. Diese bezeichnet, ob ein Dokument nach seiner letzten Speicherung verändert wurde:

- Wenn **Saved** auf **True** steht, wurde das Dokument seit dem letzten Speichern nicht mehr verändert.
- Wenn **Saved** auf **False** steht, wurde das Dokument seit dem letzten Speichern verändert. Der Anwender wird beim Schließen des Dokuments gefragt, ob es gespeichert werden soll.

Note: Sobald der Anwender an einem Dokument etwas ändert, wird **Saved** auf **False** gesetzt.

ReadOnly (Eigenschaft)

Datentyp: **Boolean**

Liest oder setzt die **ReadOnly**-Eigenschaft des Dokuments.

Wenn diese Eigenschaft **True** ist, ist das Dokument gegen Änderungen durch den Benutzer geschützt. Er kann dann nichts mehr editieren, löschen oder einfügen.

Setzen Sie diese Eigenschaft auf **True**, wird automatisch die Eigenschaft **EnableCaretMovement** (siehe dort) auf **False** gesetzt. Dadurch kann dann im Dokument die Schreibmarke nicht mehr versetzt werden. Sie können aber **EnableCaretMovement** auch wieder auf **True** setzen, sodass dieses wieder möglich ist.

EnableCaretMovement (Eigenschaft)

Datentyp: **Boolean**

Liest oder setzt die **EnableCaretMovement**-Eigenschaft des Dokuments. Diese Eigenschaft ist nur sinnvoll in Kombination mit der **ReadOnly**-Eigenschaft (siehe dort).

Wenn **EnableCaretMovement True** ist, kann die Schreibmarke im (schreibgeschützten) Dokument frei bewegt werden. Wird die Eigenschaft auf **False** gesetzt, ist das Versetzen der Schreibmarke nicht mehr möglich.

MergeFileName (Eigenschaft)

Datentyp: **String**

Liest oder setzt den Namen der Serienbriefdatenbank, die dem Dokument zugeordnet ist.

MergeFileFormat (Eigenschaft)

Datentyp: **Long** (TmMergeType)

Liest oder setzt das Dateiformat der Serienbriefdatenbank, die dem Dokument zugeordnet ist. Mögliche Werte:

```
tmMergeCSVAnsi         =  3
tmMergeDBaseAnsi       =  5
tmMergeCSVDos          = 64
tmMergeDBaseDos        = 66
tmMergeDBaseUnicode    = 69
```

MergeFileHeader (Eigenschaft)

Datentyp: **Boolean**

Liest oder setzt die Option **Feldnamen aus dem 1. Datensatz holen** (bei TextMaker finden Sie die Option in **Weiteres > Datenbank zuordnen**).

Diese Eigenschaft ist nur sinnvoll einsetzbar bei CSV-Dateien (**tmMergeCSVAnsi**, **tmMergeCSVDos**).

MergeRecord (Eigenschaft)

Datentyp: **Long**

Liest oder setzt bei einem Serienbriefdokument die Nummer des angezeigten Datensatzes. Entspricht der Einstellung **Datensatz anzeigen** im Dialogfenster des Befehls **Datei > Eigenschaften**, Karteikarte **Ansicht**.

PageSetup (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das **PageSetup**-Objekt, das Sie auf die Seitenformatierung (Papierformat, Ränder etc.) des Dokuments zugreifen lässt.

Selection (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das **Selection**-Objekt, das Sie auf den gerade selektierten (markierten) Text des Dokuments zugreifen lässt. Wenn nichts markiert ist, liefert das Objekt die aktuelle Schreibmarke.

ActiveWindow (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das **Window**-Objekt, das die Fenstereinstellungen (zum Beispiel Höhe und Breite auf dem Bildschirm) des Dokuments enthält.

Application (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das **Application**-Objekt.

Parent (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das übergeordnete Objekt, also **Documents**.

BuiltInDocumentProperties (Zeiger auf Sammlung)

Datentyp: **Object**

Liefert die **DocumentProperties**-Sammlung, die Sie auf die Dokumentinfos (Titel, Thema, Autor etc.) des Dokuments zugreifen lässt.

Paragraphs (Zeiger auf Sammlung)

Datentyp: **Object**

Liefert die **Paragraphs**-Sammlung, eine Sammlung aller Absätze des Dokuments.

Tables (Zeiger auf Sammlung)

Datentyp: **Object**

Liefert die **Tables**-Sammlung, eine Sammlung aller Tabellen des Dokuments.

FormFields (Zeiger auf Sammlung)

Datentyp: **Object**

Liefert die **FormFields**-Sammlung, eine Sammlung aller Formularobjekte des Dokuments.

Activate (Methode)

Bringt das Dokumentfenster in den Vordergrund (sofern **Visible** für das Dokument True ist) und setzt den Fokus auf das Dokumentfenster.

Syntax:

```
Activate
```

Parameter:

keine

Rückgabetyp:

keiner

Beispiel:

```
' Das erste Dokument der Documents-Sammlung in den Vordergrund bringen
tm.Documents(1).Activate
```

Close (Methode)

Schließt das Dokument.

Syntax:

```
Close [SaveChanges]
```

Parameter:

SaveChanges (optional; Typ: **Long** bzw. **SmoSaveOptions**) gibt an, ob das Dokument gespeichert werden soll oder nicht. Lassen Sie den Parameter weg, wird stattdessen der Benutzer gefragt – jedoch nur dann, wenn das Dokument seit der letzten Speicherung verändert wurde. Mögliche Werte für **SaveChanges**:

```
smoDoNotSaveChanges = 0       ' Nicht fragen, nicht speichern
smoPromptToSaveChanges = 1    ' Den Benutzer fragen
smoSaveChanges = 2            ' Ohne Rückfrage speichern
```

Rückgabetyp:

keiner

Beispiel:

```
' Das aktive Dokument schließen, ohne es zu speichern
tm.ActiveDocument.Close smoDoNotSaveChanges
```

Save (Methode)

Speichert das Dokument.

Syntax:

```
Save
```

Parameter:

keine

Rückgabetyp:

keiner

Beispiel:

```
' Das aktive Dokument speichern
tm.ActiveDocument.Save
```

SaveAs (Methode)

Speichert das Dokument unter einem anderen Namen und/oder in einem anderen Pfad.

Syntax:

```
SaveAs FileName, [FileFormat]
```

Parameter:

FileName (Typ: **String**): Pfad und Dateiname, unter dem das Dokument gespeichert werden soll

FileFormat (optional; Typ: **Long** bzw. **TmSaveFormat**) bestimmt das Dateiformat. Dieser Parameter kann folgende Werte annehmen (links die symbolische Konstante, rechts der entsprechende numerische Wert):

```
tmFormatDocument          =  0 ' Dokument, ist der Standardwert
tmFormatTemplate          =  1 ' Dokumentvorlage
tmFormatWinWord97         =  2 ' Microsoft Word für Windows 97 und 2000
tmFormatOpenDocument      =  3 ' OpenDocument, OpenOffice.org, StarOffice
```

```
tmFormatRTF              =  4 ' Rich Text Format
tmFormatPocketWordPPC    =  5 ' Pocket Word auf dem Pocket PC
tmFormatPocketWordHPC    =  6 ' Pocket Word auf dem Handheld PC (Windows CE)
tmFormatPlainTextAnsi    =  7 ' Textdatei mit Windows-Zeichensatz
tmFormatPlainTextDOS     =  8 ' Textdatei mit DOS-Zeichensatz
tmFormatPlainTextUnicode =  9 ' Textdatei mit Unicode-Zeichensatz
tmFormatPlainTextUTF8    = 10 ' Textdatei mit UTF8-Zeichensatz
tmFormatHTML             = 12 ' HTML
tmFormatWinWord6         = 13 ' Microsoft Word für Windows 6.0
tmFormatPlainTextUnix    = 14 ' Textdatei für UNIX, Linux, FreeBSD
tmFormatWinWordXP        = 15 ' Microsoft Word für Windows XP und 2003
```

Wenn Sie diesen Parameter weglassen, wird **tmFormatDocument** angenommen.

Rückgabetyp:

keiner

Beispiel:

```
' Das aktuelle Dokument unter dem angegebenen Namen im RTF-Format speichern
tm.ActiveDocument.SaveAs "c:\doks\test.rtf", tmFormatRTF
```

Select (Methode)

Selektiert (markiert) das gesamte Dokument.

Syntax:

```
Select
```

Parameter:

keine

Rückgabetyp:

keiner

Beispiel:

```
' Das aktuelle Dokument selektieren
tm.ActiveDocument.Select
```

Sie können dann über das **Selection**-Objekt beispielsweise die Textformatierung ändern oder den markierten Text in die Zwischenablage übertragen.

PrintOut (Methode)

Druckt das Dokument auf dem aktuell gewählten Drucker aus.

Syntax:

```
PrintOut [From], [To]
```

Parameter:

From (optional; Typ: **Long**) gibt an, ab welcher Seite gedruckt werden soll. Lassen Sie diesen Parameter weg, wird ab der ersten Seite gedruckt.

To (optional; Typ: **Long**) gibt an, bis zu welcher Seite gedruckt werden soll. Lassen Sie diesen Parameter weg, wird bis zur letzten Seite gedruckt.

Rückgabetyp:

Boolean (True, wenn das Drucken erfolgreich)

Beispiel:

```
' Seiten 2-5 des aktuellen Dokuments ausdrucken
tm.ActiveDocument.PrintOut 2, 5
```

MailMerge (Methode)

Überträgt aus der einem Dokument zugeordneten Datenbank die Datenfelder des in **Datei > Eigenschaften** eingestellten Datensatzes in das Dokument.

Syntax:

```
MailMerge Options, [ReplaceFields]
```

Parameter:

Options (Typ: **Long** bzw. **TmMergeOption**) gibt an, welche Art von Datenmischung durchgeführt wird. Mögliche Werte:

```
tmSingleFax       = 1
tmSingleAddress   = 2
tmMultipleFax     = 3
tmMultipleAddress = 4
```

ReplaceFields (optional; Typ: **Boolean**) bestimmt, ob die im Dokument eingefügten Datenbankfelder physikalisch durch ihre Feldinhalte ersetzt werden sollen. Standardmäßig ist der Wert **False**.

Rückgabetyp:

keiner

Beispiel:

```
' In das Dokument den Datensatz 5 aus der zugeordneten Datenbank einfügen
tm.ActiveDocument.MergeRecord = 5
tm.ActiveDocument.MailMerge tmSingleAddress, True
```

MergePrintOut (Methode)

Druckt das Dokument auf dem aktuell gewählten Drucker als Serienbrief aus.

Syntax:

```
MergePrintOut [From], [To]
```

Parameter:

From (optional; Typ: **Long**) gibt die Nummer des ersten auszudruckenden Datensatzes an. Lassen Sie diesen Parameter weg, wird ab dem ersten Datensatz gedruckt.

To (optional; Typ: **Long**) gibt die Nummer des letzten auszudruckenden Datensatzes an. Lassen Sie diesen Parameter weg, wird bis zum letzten Datensatz gedruckt.

Rückgabetyp:

Boolean (True wenn Drucken erfolgreich)

Beispiel:

```
' Das aktuelle Serienbriefdokument mit Datensätzen 99-105 drucken
tm.ActiveDocument.MergePrintOut 99, 105
```

DocumentProperties (Sammlung)

Zugriffspfade:
- Application → Documents → Item → **DocumentProperties**
- Application → ActiveDocument → **DocumentProperties**

1 Beschreibung

Die Sammlung **DocumentProperties** enthält alle Dokumenteigenschaften eines Dokuments. Dazu gehören zum Beispiel der Titel, der Autor, die Anzahl der Wörter usw.

Die einzelnen Elemente dieser Sammlung sind vom Typ **DocumentProperty**.

2 Zugriff auf die Sammlung

Jedes geöffnete Dokument besitzt genau eine **DocumentProperties**-Sammlung. Diese wird über **Document.BuiltInDocumentProperties** angesprochen:

```
' Den Titel des aktiven Dokuments auf "Meine Memoiren" setzen
tm.ActiveDocument.BuiltInDocumentProperties(smoPropertyTitle) = "Meine Memoiren"

' Die Anzahl der Wörter des aktiven Dokuments ausgeben
MsgBox tm.ActiveDocument.BuiltInDocumentProperties("Number of words")
```

3 Eigenschaften, Objekte, Sammlungen und Methoden

Eigenschaften:
- **Count** R/O

Objekte:
- ***Item*** → **DocumentProperty** (Defaultobjekt)
- **Application** → **Application**
- **Parent** → **Document**

Count (Eigenschaft, R/O)

Datentyp: **Long**

Liefert die Anzahl der **DocumentProperty**-Objekte in der Sammlung, also die Zahl der Dokumenteigenschaften eines Dokuments. Der Wert ist unveränderlich, da alle TextMaker-Dokumente dieselbe Zahl von Dokumenteigenschaften besitzen.

Item (Zeiger auf Objekt)

Datentyp: **Object**

Liefert ein einzelnes **DocumentProperty**-Objekt, also eine einzelne Dokumenteigenschaft.

Welches DocumentProperty-Objekt Sie erhalten, hängt von dem Parameter ab, den Sie an **Item** übergeben. Dies kann entweder der numerische Index oder der Name der gewünschten Dokumenteigenschaft sein.

Die folgende Tabelle enthält sowohl die erlaubten Zahlenwerte als auch die zugehörigen Namen:

```
smoPropertyTitle              =  1 ' "Title"
smoPropertySubject            =  2 ' "Subject"
smoPropertyAuthor             =  3 ' "Author"
smoPropertyKeywords           =  4 ' "Keywords"
```

```
smoPropertyComments              =  5 ' "Comments"
smoPropertyAppName               =  6 ' "Application name"
smoPropertyTimeLastPrinted       =  7 ' "Last print date"
smoPropertyTimeCreated           =  8 ' "Creation date"
smoPropertyTimeLastSaved         =  9 ' "Last save time"
smoPropertyKeystrokes            = 10 ' "Number of keystrokes"
smoPropertyCharacters            = 11 ' "Number of characters"
smoPropertyWords                 = 12 ' "Number of words"
smoPropertySentences             = 13 ' "Number of sentences"
smoPropertyParas                 = 14 ' "Number of paragraphs"
smoPropertyChapters              = 15 ' "Number of chapters"
smoPropertySections              = 16 ' "Number of sections"
smoPropertyLines                 = 17 ' "Number of lines"
smoPropertyPages                 = 18 ' "Number of pages"
smoPropertyCells                 = 19 ' - (bei TextMaker nicht verfügbar)
smoPropertyTextCells             = 20 ' - (bei TextMaker nicht verfügbar)
smoPropertyNumericCells          = 21 ' - (bei TextMaker nicht verfügbar)
smoPropertyFormulaCells          = 22 ' - (bei TextMaker nicht verfügbar)
smoPropertyNotes                 = 23 ' - (bei TextMaker nicht verfügbar)
smoPropertySheets                = 24 ' - (bei TextMaker nicht verfügbar)
smoPropertyCharts                = 25 ' - (bei TextMaker nicht verfügbar)
smoPropertyPictures              = 26 ' "Number of pictures"
smoPropertyOLEObjects            = 27 ' - (bei TextMaker nicht verfügbar)
smoPropertyDrawings              = 28 ' - (bei TextMaker nicht verfügbar)
smoPropertyTextFrames            = 29 ' "Number of text frames"
smoPropertyTables                = 30 ' "Number of tables"
smoPropertyFootnotes             = 31 ' "Number of footnotes"
smoPropertyAvgWordLength         = 32 ' "Average word length"
smoPropertyAvgCharactersSentence = 33 ' "Average characters per sentence"
smoPropertyAvgWordsSentence      = 34 ' "Average words per sentence"
```

Diese Liste führt *alle* Dokumenteigenschaften auf, die in SoftMaker Office verfügbar sind, auch solche, die es bei TextMaker nicht gibt. Diese sind mit "bei nicht TextMaker verfügbar" gekennzeichnet.

Application (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das **Application**-Objekt.

Parent (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das übergeordnete Objekt, also ein Objekt vom Typ **Document**.

DocumentProperty (Objekt)

Zugriffspfade:
- Application → Documents → Item → BuiltInDocumentProperties → **Item**
- Application → ActiveDocument → BuiltInDocumentProperties → **Item**

1 Beschreibung

Ein **DocumentProperty**-Objekt repräsentiert eine einzelne Dokumenteigenschaft eines Dokuments, etwa den Titel, den Autor oder die Zahl der Wörter eines Dokuments.

2 Zugriff auf das Objekt

Die einzelnen **DocumentProperty**-Objekte können ausschließlich durch Aufzählung der Elemente von Sammlungen des Typs **DocumentProperties** angesprochen werden.

Für jedes geöffnete Dokument existiert genau eine Instanz dieser **DocumentProperties**-Sammlung, nämlich **BuiltInDocumentProperties** im **Document**-Objekt:

```
' Den Titel des aktiven Dokuments auf "Meine Memoiren" setzen
tm.ActiveDocument.BuiltInDocumentProperties.Item(smoPropertyTitle) = "Meine Memoiren"
```


Eigenschaften:
- **Name** R/O
- ***Value*** (Defaulteigenschaft)
- **Valid**
- **Type**

Objekte:
- **Application → Application**
- **Parent → BuiltInDocumentProperties**

Name (Eigenschaft, R/O)

Datentyp: **String**

Liefert den Namen der Dokumenteigenschaft. Beispiele:

```
' Den Namen der Dokumenteigenschaft smoPropertyTitle ausgeben, also "Title"
MsgBox tm.ActiveDocument.BuiltInDocumentProperties.Item(smoPropertyTitle).Name

' Den Namen der Dokumenteigenschaft "Author" ausgeben, also "Author"
MsgBox tm.ActiveDocument.BuiltInDocumentProperties.Item("Author").Name
```

Value (Eigenschaft)

Datentyp: **String**

Liest oder setzt den Inhalt der Dokumenteigenschaft.

Das folgende Beispiel setzt die Dokumenteigenschaft "Titel" über die numerische Konstante **smoPropertyTitle** und liest sie gleich wieder über die Stringkonstante "Title" aus:

```
Sub Beispiel()
  Dim tm as Object

  Set tm = CreateObject("TextMaker.Application")
  tm.Documents.Add ' Neues leeres Dokument hinzufügen

  With tm.ActiveDocument

    ' Neuen Titel setzen (mit Hilfe der numerischen Konstante smoPropertyTitle)
    .BuiltInDocumentProperties.Item(smoPropertyTitle).Value = "Neuer Titel"

    ' Genau diese Eigenschaft wieder auslesen (diesmal über den String)
    MsgBox .BuiltInDocumentProperties.Item("Title").Value

  End With
End Sub
```

Da **Item** das Defaultobjekt von **DocumentProperties** ist und **Value** die Defaulteigenschaft von **DocumentProperty**, lässt sich dieses Beispiel übersichtlicher wie folgt schreiben:

```
Sub Beispiel()
```

```
  Dim tm as Object

  Set tm = CreateObject("TextMaker.Application")
  tm.Documents.Add ' Neues leeres Dokument hinzufügen

  With tm.ActiveDocument

    ' Neuen Titel setzen (mit Hilfe der numerischen Konstante smoPropertyTitle)
    .BuiltInDocumentProperties(smoPropertyTitle) = "Neuer Titel"

    ' Genau diese Eigenschaft wieder auslesen (diesmal über den String)
    MsgBox .BuiltInDocumentProperties("Title")

  End With
End Sub
```

Valid (Eigenschaft, R/O)

Datentyp: **Boolean**

Liefert **True** zurück, wenn die Dokumenteigenschaft bei TextMaker verfügbar ist.

Hintergrund: Die Liste der möglichen Dokumenteigenschaften enthält auch solche, die nur bei PlanMaker verfügbar sind (zum Beispiel **smoPropertySheets**, "Number of sheets"). Sie dürfen bei TextMaker nur diejenigen Dokumenteigenschaften abfragen, die TextMaker kennt – sonst wird ein Leerwert zurückgegeben (VT_EMPTY).

Die Eigenschaft **Valid** lässt Sie vor der Abfrage prüfen, ob die jeweilige Dokumenteigenschaft bei TextMaker vorhanden ist. Beispiel:

```
Sub Test
  Dim tm as Object
  Dim i as Integer

  Set tm = CreateObject("TextMaker.Application")

  tm.Visible = True
  tm.Documents.Add ' leeres Dokument hinzufügen

  With tm.ActiveDocument
   For i = 1 to .BuiltInDocumentProperties.Count
     If .BuiltInDocumentProperties(i).Valid then
        print i, .BuiltInDocumentProperties(i).Name, "=", _
           .BuiltInDocumentProperties(i).Value
     Else
        print i, "Nicht bei TextMaker verfügbar"
     End If
   Next i
  End With

End Sub
```

Type (Eigenschaft, R/O)

Datentyp: **Long** (SmoDocProperties)

Liefert den Datentyp der Dokumenteigenschaft. Damit Sie eine Dokumenteigenschaft richtig auswerten können, müssen Sie ihren Typ wissen. Beispielsweise ist **Title** (smoPropertyTitle) ein String, **Creation Date** (smoPropertyTimeCreated) hingegen ein Datum. Mögliche Werte:

```
smoPropertyTypeBoolean   = 0 ' Boolean
smoPropertyTypeDate      = 1 ' Datum
smoPropertyTypeFloat     = 2 ' Fließkommawert
smoPropertyTypeNumber    = 3 ' Ganzzahl
smoPropertyTypeString    = 4 ' Zeichenkette
```

Application (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das **Application**-Objekt.

Parent (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das übergeordnete Objekt, also **BuiltInDocumentProperties**.

PageSetup (Objekt)

Zugriffspfade:
- Application → Documents → Item → **PageSetup**
- Application → ActiveDocument → **PageSetup**

1 Beschreibung

Das **PageSetup**-Objekt enthält die Seiteneinstellungen des **Document**-Objekts, zu dem es gehört. Sie können damit das Papierformat, Seitengröße und -ränder sowie die Druckrichtung eines Dokuments ermitteln und verändern.

2 Zugriff auf das Objekt

Jedes geöffnete Dokument besitzt genau eine Instanz des **PageSetup**-Objekts. Diese wird über **Document.PageSetup** angesprochen:

```
' Den linken Blattrand auf 2 cm setzen
tm.ActiveDocument.PageSetup.LeftMargin = tm.CentimetersToPoints(2)
```

Hinweis: TextMaker erlaubt es, ein Dokument in mehrere Kapitel aufzuteilen und in diesen unterschiedliche Seiteneinstellungen festzulegen. Das **PageSetup**-Objekt bezieht sich in einem solchen Dokument stets auf die Seiteneinstellungen desjenigen Kapitels, in dem sich die Schreibmarke befindet.

3 Eigenschaften, Objekte, Sammlungen und Methoden

Eigenschaften:
- **LeftMargin**
- **RightMargin**
- **TopMargin**
- **BottomMargin**
- **PageHeight**
- **PageWidth**
- **Orientation**
- **PaperSize**

Objekte:
- **Application** → **Application**
- **Parent** → **Document**

LeftMargin (Eigenschaft)

Datentyp: **Single**

Liest oder setzt den linken Seitenrand des Dokuments in Punkt (1 Punkt entspricht 1/72 Zoll).

RightMargin (Eigenschaft)

Datentyp: **Single**

Liest oder setzt den rechten Seitenrand des Dokuments in Punkt (1 Punkt entspricht 1/72 Zoll).

TopMargin (Eigenschaft)

Datentyp: **Single**

Liest oder setzt den oberen Seitenrand des Dokuments in Punkt (1 Punkt entspricht 1/72 Zoll).

BottomMargin (Eigenschaft)

Datentyp: **Single**

Liest oder setzt den unteren Seitenrand des Dokuments in Punkt (1 Punkt entspricht 1/72 Zoll).

PageHeight (Eigenschaft)

Datentyp: **Single**

Liest oder setzt die Seitenhöhe des Dokuments in Punkt (1 Punkt entspricht 1/72 Zoll).

Wenn Sie diese Eigenschaft setzen, ändert sich die **PaperSize**-Eigenschaft (siehe unten) automatisch auf das passende Papierformat.

PageWidth (Eigenschaft)

Datentyp: **Single**

Liest oder setzt die Seitenbreite des Dokuments in Punkt (1 Punkt entspricht 1/72 Zoll).

Wenn Sie diese Eigenschaft setzen, ändert sich die **PaperSize**-Eigenschaft (siehe unten) automatisch auf das passende Papierformat.

Orientation (Eigenschaft)

Datentyp: **Long** (SmoOrientation)

Liest oder setzt die Ausrichtung der Seite. Folgende Konstanten sind erlaubt:

```
smoOrientLandscape  = 0 ' Querformat
smoOrientPortrait   = 1 ' Hochformat
```

PaperSize (Eigenschaft)

Datentyp: **Long** (SmoPaperSize)

Liest oder setzt die Papiergröße des Dokuments. Folgende Konstanten sind erlaubt:

```
smoPaperCustom                = -1
smoPaperLetter                = 1
smoPaperLetterSmall           = 2
smoPaperTabloid               = 3
smoPaperLedger                = 4
smoPaperLegal                 = 5
smoPaperStatement             = 6
smoPaperExecutive             = 7
```

```
smoPaperA3                  = 8
smoPaperA4                  = 9
smoPaperA4Small             = 10
smoPaperA5                  = 11
smoPaperB4                  = 12
smoPaperB5                  = 13
smoPaperFolio               = 14
smoPaperQuarto              = 15
smoPaper10x14               = 16
smoPaper11x17               = 17
smoPaperNote                = 18
smoPaperEnvelope9           = 19
smoPaperEnvelope10          = 20
smoPaperEnvelope11          = 21
smoPaperEnvelope12          = 22
smoPaperEnvelope14          = 23
smoPaperCSheet              = 24
smoPaperDSheet              = 25
smoPaperESheet              = 26
smoPaperEnvelopeDL          = 27
smoPaperEnvelopeC5          = 28
smoPaperEnvelopeC3          = 29
smoPaperEnvelopeC4          = 30
smoPaperEnvelopeC6          = 31
smoPaperEnvelopeC65         = 32
smoPaperEnvelopeB4          = 33
smoPaperEnvelopeB5          = 34
smoPaperEnvelopeB6          = 35
smoPaperEnvelopeItaly       = 36
smoPaperEnvelopeMonarch     = 37
smoPaperEnvelopePersonal    = 38
smoPaperFanfoldUS           = 39
smoPaperFanfoldStdGerman    = 40
smoPaperFanfoldLegalGerman  = 41
```

Application (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das **Application**-Objekt.

Parent (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das übergeordnete Objekt, also ein Objekt vom Typ **Document**.

Selection (Objekt)

Zugriffspfade:
- Application → Documents → Item → **Selection**
- Application → ActiveDocument → **Selection**

1 Beschreibung

Selection bezeichnet die aktuelle Selektion (Markierung) in einem Dokument.

Wenn im Text etwas markiert ist, steht **Selection** für den Inhalt dieser Selektion. Ist nichts markiert, bezeichnet **Selection** die aktuelle Position der Schreibmarke. Fügen Sie also (zum Beispiel mit der Methode **Selection.TypeText**) Text ein, wird der Inhalt der Selektion durch diesen Text ersetzt, falls zuvor etwas markiert wurde. Ist nichts markiert, wird der Text an der aktuellen Position der Schreibmarke eingefügt.

Über das **Font**-Objekt, das Sie von **Selection** aus erreichen, können Sie auch die Formatierung von Text ändern. Beispiel: **tm.ActiveDocument.Selection.Font.Size = 24** ändert die Schriftgröße des im aktuellen Dokument markierten Textabschnitts auf 24 Punkt.

2 Zugriff auf das Objekt

Jedes geöffnete Dokument besitzt genau eine Instanz des **Selection**-Objekts. Diese wird über **Document.Selection** angesprochen:

```
' Die Selektion des aktuellen Dokuments in die Zwischenablage kopieren
tm.ActiveDocument.Selection.Copy
```

3 Eigenschaften, Objekte, Sammlungen und Methoden

Objekte:

- **Document → Document**
- **Font → Font**
- **Application → Application**
- **Parent → Document**

Methoden:

- **Copy**
- **Cut**
- **Paste**
- **Delete**
- **TypeText**
- **TypeParagraph**
- **TypeBackspace**
- **InsertBreak**
- **GoTo**
- **ConvertToTable**
- **SetRange**
- **InsertPicture**

Document (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das zur aktuellen Selektion gehörende **Document**-Objekt.

Font (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das zur aktuellen Selektion gehörende **Font**-Objekt. Dieses enthält Eigenschaften zum Lesen und Ändern der in der Selektion gewählten Zeichenformatierung.

Application (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das **Application**-Objekt.

Parent (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das übergeordnete Objekt, also ein Objekt vom Typ **Document**.

Copy (Methode)

Kopiert den Inhalt der Selektion in die Zwischenablage.

Syntax:

Copy

Parameter:

keine

Rückgabetyp:

keiner

Beispiel:

```
' Die aktuelle Selektion in die Zwischenablage kopieren
tm.ActiveDocument.Selection.Copy
```

Cut (Methode)

Schneidet den Inhalt der Selektion in die Zwischenablage aus.

Syntax:

Cut

Parameter:

keine

Rückgabetyp:

keiner

Beispiel:

```
' Die aktuelle Selektion ausschneiden und in die Zwischenablage übertragen
tm.ActiveDocument.Selection.Cut
```

Paste (Methode)

Fügt den Inhalt der Zwischenablage in die Selektion ein.

Syntax:

Paste

Parameter:

keine

Rückgabetyp:

keiner

Beispiel:

```
' Die aktuelle Selektion durch den Inhalt der Zwischenablage ersetzen
tm.ActiveDocument.Selection.Paste
```

Delete (Methode)

Löscht den Inhalt der Selektion.

Syntax:

```
Delete
```

Parameter:

keine

Rückgabetyp:

keiner

Beispiel:

```
' Die aktuelle Selektion löschen
tm.ActiveDocument.Selection.Delete
```

TypeText (Methode)

Fügt eine Zeichenkette in die Selektion ein.

Syntax:

```
TypeText Text
```

Parameter:

Text (Typ: **String**) ist die Zeichenkette, die eingefügt werden soll.

Rückgabetyp:

keiner

Beispiel:

```
' An der aktuellen Schreibmarke des aktiven Dokuments Text einfügen
tm.ActiveDocument.Selection.TypeText "Programmieren mit BasicMaker"
```

TypeParagraph (Methode)

Fügt einen Wagenrücklauf in die Selektion ein.

Syntax:

```
TypeParagraph
```

Parameter:

keine

Rückgabetyp:

keiner

Beispiel:

```
' Wagenrücklauf an der aktuellen Schreibmarke des aktiven Dokuments einfügen
tm.ActiveDocument.Selection.TypeParagraph
```

TypeBackspace (Methode)

Löst das Drücken der Rücktaste aus.

Syntax:

```
TypeBackspace
```

Parameter:

keine

Rückgabetyp:

keiner

Beispiel:

```
' Backspace an der aktuellen Schreibmarke des aktiven Dokuments durchführen
tm.ActiveDocument.Selection.TypeBackspace
```

InsertBreak (Methode)

Fügt einen manuellen Umbruch ein.

Syntax:

```
InsertBreak [Type]
```

Parameter:

Type (optional; Typ: **Long** bzw. **TmBreakType**) legt den gewünschten Typ des Umbruchs fest. Mögliche Werte:

```
tmLineBreak    = 0 ' Zeilenumbruch
tmColumnBreak  = 1 ' Spaltenumbruch
tmSectionBreak = 2 ' Bereichsumbruch
tmPageBreak    = 3 ' Seitenumbruch
tmChapterBreak = 4 ' Kapitelumbruch
```

Wenn Sie **Type** nicht angeben, wird **tmPageBreak** angenommen.

Rückgabetyp:

keiner

Beispiel:

```
' Einen Seitenumbruch an der aktuellen Schreibmarke einfügen
tm.ActiveDocument.Selection.InsertBreak tmPageBreak
```

GoTo (Methode)

Verschiebt die Schreibmarke an die von Ihnen gewünschte Stelle.

Syntax:

```
GoTo [What], [Which], [Count], [NumRow], [NumCol]
```

Parameter:

What (optional; Typ: **Long** bzw. **TmGoToItem**) legt fest, ob das Ziel der Bewegung eine Tabelle oder ein Absatz ist:

```
tmGoToParagraph    = 1 ' Absatz
tmGoToTable  = 2 ' Tabelle
```

Lassen Sie **What** weg, wird **tmGoToParagraph** angenommen.

Which (optional; Typ: **Long** bzw. **TmGoToDirection**) legt fest, ob die Verschiebung absolut oder relativ zur aktuellen Position der Schreibmarke erfolgen soll:

```
tmGoToAbsolute  = 1 ' absolut
tmGoToRelative  = 2 ' relativ
```

Lassen Sie **Which** weg, wird **tmGoToAbsolute** angenommen.

Count (optional; Typ: **Long**) legt fest, das wievielte Element (also die wievielte Tabelle oder der wievielte Absatz des Dokuments) erreicht werden soll.

Lassen Sie **Count** weg, wird 1 angenommen.

NumRow (optional; Typ: **Long**): Wenn **What** auf **tmGoToTable** steht, können Sie hier auf Wunsch die Zeile der Tabelle angeben, in die die Schreibmarke gesetzt werden soll.

NumCol (optional; Typ: **Long**): Wenn **What** auf **tmGoToTable** steht, können Sie hier auf Wunsch die Spalte der Tabelle angeben, in die die Schreibmarke gesetzt werden soll.

Rückgabetyp:

keiner

Beispiele:

```
' Schreibmarke in den vierten Absatz des Dokuments setzen
tm.ActiveDocument.Selection.GoTo tmGoToParagraph, tmGoToAbsolute, 4

' Schreibmarke in den vorherigen Absatz setzen
tm.ActiveDocument.Selection.GoTo tmGoToParagraph, tmGoToRelative, -1

' Schreibmarke in die erste Zelle der ersten Tabelle setzen
tm.ActiveDocument.Selection.GoTo tmGoToTable, tmGoToAbsolute, 1, 1, 1
```

ConvertToTable (Methode)

Wandelt den Text innerhalb der Selektion in eine Tabelle um.

Syntax:

```
ConvertToTable [NumRows], [NumCols], [Separator], [RemoveQuotationMarks],
[RemoveSpaces]
```

Parameter:

NumRows (optional; Typ: **Long**) legt fest, wie viele Zeilen die neue Tabelle enthalten soll. Wenn Sie diesen Parameter weglassen, rechnet TextMaker die Zahl der Zeilen selbsttätig aus.

NumCols (optional; Typ: **Long**) legt fest, wie viele Spalten die neue Tabelle enthalten soll. Wenn Sie diesen Parameter weglassen, rechnet TextMaker die Zahl der Spalten selbsttätig aus.

Separator (optional; Typ: entweder **String** oder **Long** bzw. **TmTableFieldSeparator**) gibt ein oder mehrere Zeichen an, das/die TextMaker zum Erkennen der Spaltengrenzen verwenden soll. Geben Sie entweder einen String an oder eine der folgenden symbolischen Konstanten:

```
tmSeparateByCommas       = 0 ' Spaltentrennung durch Komma
tmSeparateByParagraphs   = 1 ' Spaltentrennung durch Absatzende
tmSeparateByTabs         = 2 ' Spaltentrennung durch Tabulator
tmSeparateBySemicolons   = 3 ' Spaltentrennung durch Strichpunkt
```

Wenn Sie den Parameter weglassen, wird **tmSeparateByTabs** verwendet.

RemoveQuotationMarks (optional; Typ: **Boolean**): Setzen Sie diesen Parameter auf **True**, wenn TextMaker führende und abschließende Anführungszeichen entfernen soll. Lassen Sie den Parameter aus, wird **False** angenommen.

RemoveSpaces (optional; Typ: **Boolean**): Setzen Sie diesen Parameter auf **True**, wenn TextMaker führende und abschließende Leerzeichen entfernen soll. Lassen Sie den Parameter aus, wird **True** angenommen.

Rückgabetyp:

Object (ein **Table**-Objekt, das die neue Tabelle repräsentiert)

Beispiele:

```
' Aktuelle Selektion in eine Tabelle umwandeln. Spaltentrenner ist das Komma
tm.ActiveDocument.Selection.ConvertToTable Separator := tmSeparateByCommas

' Hier werden Schrägstriche als Trenner verwendet
tm.ActiveDocument.Selection.ConvertToTable Separator := "/"
```

SetRange (Methode)

Setzt Anfangs- und Endpunkt der Selektion, indem Sie deren Zeichenpositionen übergeben.

Syntax:

```
SetRange Start, End
```

Parameter:

Start (Typ: **Long**) legt die Startposition der neuen Selektion fest, gezählt als Anzahl der Zeichen vom Dokumentanfang.

End (Typ: **Long**) legt die Endposition der neuen Selektion fest, gezählt als Anzahl der Zeichen vom Dokumentanfang.

Rückgabetyp:

keiner

Beispiele:

```
' Selektiere von Zeichen 1 bis Zeichen 4 des aktuellen Dokuments
tm.ActiveDocument.Selection.SetRange 1, 4
```

Tipp: Sie können hiermit auch ganze Absätze selektieren. Dazu können Sie mit **Paragraph.Range.Start** und **Paragraph.Range.End** die Anfangs- und Endposition des Absatzes an die hier beschriebene **SetRange**-Methode übergeben.

InsertPicture (Methode)

Fügt eine Grafik aus einer Datei in die Selektion ein.

Syntax:

```
InsertPicture PictureName
```

Parameter:

PictureName (Typ: **String**) ist der Pfad- und Dateiname der einzufügenden Grafik.

Rückgabetyp:

keiner

Beispiele:

```
' An der aktuellen Position eine Grafik einfügen
tm.ActiveDocument.Selection.InsertPicture "c:\windows\Zapotek.bmp"
```

Font (Objekt)

Zugriffspfade:
- Application → Documents → Item → Selection → **Font**
- Application → ActiveDocument → Selection → **Font**

1 Beschreibung

Das **Font**-Objekt beschreibt die Zeichenformatierung eines Textstücks. Es ist ein Tochterobjekt von **Selection** und lässt Sie alle Zeichenattribute der aktuellen Selektion lesen und setzen.

2 Zugriff auf das Objekt

Jedes geöffnete Dokument besitzt genau eine Instanz des **Font**-Objekts. Diese wird über **Document.Selection.Font** angesprochen:

```
' Der aktuellen Selektion die Schriftart Arial zuweisen
tm.ActiveDocument.Selection.Font.Name = "Arial"
```

3 Eigenschaften, Objekte, Sammlungen und Methoden

Eigenschaften:
- ***Name*** (Defaulteigenschaft)
- **Size**
- **Bold**
- **Italic**
- **Underline**
- **StrikeThrough**
- **Superscript**
- **Subscript**
- **AllCaps**
- **SmallCaps**
- **PreferredSmallCaps**
- **Blink**
- **Color**
- **ColorIndex**
- **BColor**
- **BColorIndex**
- **Spacing**
- **Pitch**

Objekte:
- **Application → Application**
- **Parent → Selection**

Name (Eigenschaft)

Datentyp: **String**

Liest oder setzt den Namen der Schriftart (als Zeichenkette).

Falls innerhalb der Selektion mehrere Schriftarten vorkommen, wird eine leere Zeichenkette zurückgeliefert.

Size (Eigenschaft)

Datentyp: **Single**

Liest oder setzt die Schriftgröße in Punkt (pt).

Falls innerhalb der Selektion mehrere Schriftgrößen vorkommen, wird **smoUndefined** (9.999.999) zurückgeliefert.

Beispiel:

```
' Setze die Schriftgröße des selektierten Textes auf 10,3 pt
tm.ActiveDocument.Selection.Font.Size = 10.3
```

Bold (Eigenschaft)

Datentyp: **Long**

Liest oder setzt die Zeichenformatierung "Fettdruck":

- **True:** Fettdruck ein
- **False:** Fettdruck aus
- **smoToggle** (nur beim Setzen): Aktueller Zustand wird ins Gegenteil gekehrt.
- **smoUndefined** (nur beim Lesen): Die Selektion ist teils fett, teils nicht.

Italic (Eigenschaft)

Datentyp: **Long**

Liest oder setzt die Zeichenformatierung "Kursivschrift":

- **True:** Kursivschrift ein
- **False:** Kursivschrift aus
- **smoToggle** (nur beim Setzen): Aktueller Zustand wird ins Gegenteil gekehrt.
- **smoUndefined** (nur beim Lesen): Die Selektion ist teils kursiv, teils nicht.

Underline (Eigenschaft)

Datentyp: **Long** (TmUnderline)

Liest oder setzt die Zeichenformatierung "Unterstreichen". Folgende Werte sind zulässig:

```
tmUnderlineNone         = 0 ' aus
tmUnderlineSingle       = 1 ' einfach durchgehend
tmUnderlineDouble       = 2 ' doppelt durchgehend
tmUnderlineWords        = 3 ' einfach wortweise
tmUnderlineWordsDouble = 4 ' doppelt wortweise
```

Lesen Sie die Eigenschaft aus und die Selektion ist teils unterstrichen, teils nicht, wird **smoUndefined** zurückgeliefert.

StrikeThrough (Eigenschaft)

Datentyp: **Long**

Liest oder setzt die Zeichenformatierung "Durchstreichen":

- **True:** Durchstreichen ein
- **False:** Durchstreichen aus
- **smoToggle** (nur beim Setzen): Aktueller Zustand wird ins Gegenteil gekehrt.
- **smoUndefined** (nur beim Lesen): Die Selektion ist teils durchgestrichen, teils nicht.

Superscript (Eigenschaft)

Datentyp: **Long**

Liest oder setzt die Zeichenformatierung "Hochstellen":

- **True:** Hochstellen ein
- **False:** Hochstellen aus
- **smoToggle** (nur beim Setzen): Aktueller Zustand wird ins Gegenteil gekehrt.
- **smoUndefined** (nur beim Lesen): Die Selektion ist teils hochgestellt, teils nicht.

Subscript (Eigenschaft)

Datentyp: **Long**

Liest oder setzt die Zeichenformatierung "Tiefstellen":

- **True:** Tiefstellen ein
- **False:** Tiefstellen aus
- **smoToggle** (nur beim Setzen): Aktueller Zustand wird ins Gegenteil gekehrt.
- **smoUndefined** (nur beim Lesen): Die Selektion ist teils tiefgestellt, teils nicht.

AllCaps (Eigenschaft)

Datentyp: **Long**

Liest oder setzt die Zeichenformatierung "Versalien" (Text in Großbuchstaben):

- **True:** Versalien ein
- **False:** Versalien aus
- **smoToggle** (nur beim Setzen): Aktueller Zustand wird ins Gegenteil gekehrt.
- **smoUndefined** (nur beim Lesen): Das Attribut ist bei einem Teil der Selektion gesetzt, bei einem anderen Teil nicht.

SmallCaps (Eigenschaft)

Datentyp: **Long**

Liest oder setzt die Zeichenformatierung "Kapitälchen":

- **True:** Kapitälchen ein
- **False:** Kapitälchen aus
- **smoToggle** (nur beim Setzen): Aktueller Zustand wird ins Gegenteil gekehrt.
- **smoUndefined** (nur beim Lesen): Die Selektion ist teils in Kapitälchen, teils nicht.

PreferredSmallCaps (Eigenschaft)

Datentyp: **Long**

Liest oder setzt die Zeichenformatierung "Kapitälchen", lässt Sie aber im Gegensatz zur Eigenschaft **SmallCaps** den Skalierungsfaktor frei wählen. Der Wert 0 bedeutet "keine Kapitälchen", alle anderen Werte stellen den prozentualen Skalierungsfaktor der Kapitälchen dar.

Beispiel:

```
' Formatiere den selektierten Text als Kapitälchen mit 75% Größe
tm.ActiveDocument.Selection.Font.PreferredSmallCaps = 75

' Schalte die Kapitälchen wieder aus
tm.ActiveDocument.Selection.Font.PreferredSmallCaps = 0
```

Blink (Eigenschaft)

Datentyp: **Long**

Liest oder setzt die Zeichenformatierung "Blinken":

- **True:** Blinken ein
- **False:** Blinken aus
- **smoToggle** (nur beim Setzen): Aktueller Zustand wird ins Gegenteil gekehrt.
- **smoUndefined** (nur beim Lesen): Die Selektion ist teils blinkend, teils nicht.

Color (Eigenschaft)

Datentyp: **Long** (SmoColor)

Liest oder setzt die Vordergrundfarbe der Schrift als "BGR"-Wert (Blau-Grün-Rot-Triplet). Sie können entweder einen beliebigen Wert angeben oder eine der vordefinierten BGR-Farbkonstanten verwenden.

Ist die Selektion in unterschiedlichen Farben formatiert, wird beim Auslesen **smoUndefined** zurückgeliefert.

ColorIndex (Eigenschaft)

Datentyp: **Long** (SmoColorIndex)

Liest oder setzt die Vordergrundfarbe der Schrift als Indexfarbe. "Indexfarben" sind die 16 Standardfarben von TextMaker, durchnummeriert von 0 für Schwarz bis 15 für Hellgrau. Sie dürfen ausschließlich einen der in der Tabelle der Indexfarben genannten Werte verwenden.

Ist die Selektion in unterschiedlichen Farben oder in einer anderen als einer der Indexfarben formatiert, wird beim Auslesen **smoUndefined** zurückgeliefert.

Hinweis: Wir empfehlen, stattdessen die Eigenschaft **Color** (siehe dort) zu verwenden, da Sie mit dieser nicht auf die 16 Standardfarben beschränkt sind, sondern mit beliebigen Farben der BGR-Palette arbeiten können.

BColor (Eigenschaft)

Datentyp: **Long** (SmoColor)

Liest oder setzt die Hintergrundfarbe der Schrift als "BGR"-Wert (Blau-Grün-Rot-Triplet). Sie können entweder einen beliebigen Wert angeben oder eine der vordefinierten BGR-Farbkonstanten verwenden.

Lesen Sie die Eigenschaft aus und die Selektion ist in unterschiedlichen Farben formatiert, wird **smoUndefined** zurückgeliefert.

BColorIndex (Eigenschaft)

Datentyp: **Long** (SmoColorIndex)

Liest oder setzt die Hintergrundfarbe der Schrift als Indexfarbe. "Indexfarben" sind die Standardfarben von TextMaker, durchnummeriert von -1 für Transparent bis 15 für Hellgrau. Sie dürfen ausschließlich einen der in der Tabelle der Indexfarben genannten Werte verwenden.

Ist die Selektion in unterschiedlichen Farben oder in einer anderen als einer der Indexfarben formatiert, wird beim Auslesen **smoUndefined** zurückgeliefert.

Hinweis: Wir empfehlen, stattdessen die Eigenschaft **BColor** (siehe dort) zu verwenden, da Sie mit dieser nicht auf diese Standardfarben beschränkt sind, sondern beliebige Farben der BGR-Palette wählen können.

Spacing (Eigenschaft)

Datentyp: **Long**

Liest oder setzt den Zeichenabstand. Der Standardwert ist 100 für normalen (100%) Zeichenabstand.

Lesen Sie die Eigenschaft aus und die Selektion ist in unterschiedlichen Zeichenabständen formatiert, wird **smoUndefined** zurückgeliefert.

Pitch (Eigenschaft)

Datentyp: **Long**

Liest oder setzt die Zeichenbreite. Der Standardwert ist 100 für normalbreite (100%) Zeichen.

Lesen Sie die Eigenschaft aus und die Selektion ist in unterschiedlichen Zeichenbreiten formatiert, wird **smoUndefined** zurückgeliefert.

Beachten Sie bitte, dass manche Drucker die Änderung der Zeichenbreite bei *druckerinternen* Schriften ignorieren.

Application (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das **Application**-Objekt.

Parent (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das übergeordnete Objekt, also ein Objekt vom Typ **Document**.

Paragraphs (Sammlung)

Zugriffspfade:
- Application → Documents → Item → **Paragraphs**
- Application → ActiveDocument → **Paragraphs**

1 Beschreibung

Paragraphs ist eine Sammlung aller Absätze eines Dokuments. Die einzelnen Elemente dieser Sammlung sind vom Typ **Paragraph**.

2 Zugriff auf die Sammlung

Jedes geöffnete Dokument besitzt genau eine Instanz der **Paragraphs**-Sammlung. Diese wird über **Document.Paragraphs** angesprochen:

```
' Anzahl der Absätze des aktuellen Dokuments anzeigen
MsgBox tm.ActiveDocument.Paragraphs.Count
```

3 Eigenschaften, Objekte, Sammlungen und Methoden

Eigenschaften:
- **Count** R/O

Objekte:
- ***Item*** → **Paragraph** (Defaultobjekt)
- **Application** → **Application**
- **Parent** → **Document**

Count (Eigenschaft, R/O)

Datentyp: **Long**

Liefert die Anzahl der **Paragraph**-Objekte des Dokuments – in anderen Worten: die Anzahl der Absätze des Dokuments.

Item (Zeiger auf Objekt)

Datentyp: **Object**

Liefert ein einzelnes **Paragraph**-Objekt, also einen einzelnen Absatz.

Welches Paragraph-Objekt Sie erhalten, hängt von dem Zahlenwert ab, den Sie an **Item** übergeben: 1 für den ersten Absatz des Dokuments, 2 für den zweiten etc.

Application (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das **Application**-Objekt.

Parent (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das übergeordnete Objekt, also ein Objekt vom Typ **Document**.

Paragraph (Objekt)

Zugriffspfade:
- Application → Documents → Item → Paragraphs → **Item**
- Application → ActiveDocument → Paragraphs → **Item**

1 Beschreibung

Ein **Paragraph**-Objekt repräsentiert einen einzelnen Absatz des Dokuments und erlaubt es Ihnen, dessen Formatierung zu ändern.

Für jeden Absatz existiert ein eigenes **Paragraph**-Objekt. Fügen Sie einem Dokument Absätze hinzu oder löschen diese, werden die zugehörigen **Paragraph**-Objekte dynamisch angelegt bzw. entfernt.

2 Zugriff auf das Objekt

Die einzelnen **Paragraph**-Objekte können ausschließlich durch Aufzählung der Elemente der **Paragraphs**-Sammlung angesprochen werden. Von dieser Sammlung besitzt jedes Dokument genau eine Instanz.

Ein Beispiel:

```
' Den ersten Absatz des Dokuments auf "Blocksatz" stellen
tm.ActiveDocument.Paragraphs.Item(1).Alignment = tmAlignParagraphJustify

' Dasselbe mit Hilfe eines Hilfsobjekts
Dim absatz as Object
Set absatz = tm.ActiveDocument.Paragraphs.Item(1)
absatz.Alignment = tmAlignParagraphJustify
Set absatz = Nothing ' Hilfsobjekt wieder entfernen
```

3 Eigenschaften, Objekte, Sammlungen und Methoden

Eigenschaften:

- **BorderBounds**
- **FirstLineIndent**
- **LeftIndent**
- **RightIndent**
- **LineSpacingRule**
- **LineSpacing**
- **PreferredLineSpacing**
- **SpaceBefore**
- **SpaceAfter**
- **Alignment**
- **Hyphenation**
- **OutlineLevel**
- **PageBreakBefore**
- **ColumnBreakBefore**
- **KeepWithNext**
- **KeepTogether**
- **WidowControl**
- **BorderClearance**

Objekte:

- **Shading → Shading**
- **DropCap → DropCap**
- **Range → Range**
- **Application → Application**
- **Parent → Paragraphs**

Sammlungen:

- **Borders → Borders**

BorderBounds (Eigenschaft)

Datentyp: **Long** (TmBorderBounds)

Liest oder setzt die Einstellung, welchen Abstand die Absatzumrandung vom Absatz einhält. Mögliche Werte:

```
tmBoundsPage          = 0 ' Umrandung erstreckt sich zu den Seitenrändern
tmBoundsIndents       = 1 ' Umrandung erstreckt sich zu den Absatzrändern
tmBoundsText          = 2 ' Umrandung erstreckt sich um den Text des Absatzes
```

FirstLineIndent (Eigenschaft)

Datentyp: **Single**

Liest oder setzt den Erstzeileneinzug des Absatzes in Punkt (1 Punkt entspricht 1/72 Zoll).

LeftIndent (Eigenschaft)

Datentyp: **Single**

Liest oder setzt den linken Einzug des Absatzes in Punkt (1 Punkt entspricht 1/72 Zoll).

RightIndent (Eigenschaft)

Datentyp: **Single**

Liest oder setzt den rechten Einzug des Absatzes in Punkt (1 Punkt entspricht 1/72 Zoll).

LineSpacingRule (Eigenschaft)

Datentyp: **Long** (TmLineSpacing)

Liest oder setzt, auf welche Art der Zeilenabstand des Absatzes durchgeführt wird. Mögliche Werte:

```
tmLineSpaceAuto          = 0 ' Automatisch (in Prozent)
tmLineSpaceExactly       = 1 ' Genau (in Punkt)
tmLineSpaceAtLeast       = 2 ' Mindestens (in Punkt)
```

LineSpacing (Eigenschaft)

Datentyp: **Single**

Liest oder setzt den Zeilenabstand des Absatzes.

Im Gegensatz zur Eigenschaft **PreferredLineSpacing** (siehe dort) wird hier der Zeilenabstandsmodus (siehe **LineSpacingRule**) ignoriert – es wird immer der Zeilenabstand in Punkt übergeben, normiert auf eine Standardschriftgröße von 12 Punkt.

Mit anderen Worten: Egal, ob der Zeilenabstand auf "Automatisch 100%", auf "Genau 12 pt" oder auf "Mindestens 12 Punkt" steht, liefert diese Eigenschaft immer das Ergebnis 12.

PreferredLineSpacing (Eigenschaft)

Datentyp: **Single**

Liest oder setzt den Zeilenabstand des Absatzes.

Diese Eigenschaft liefert und erwartet Werte abhängig vom gewählten Zeilenabstandsmodus (siehe **LineSpacingRule**):

- **tmLineSpaceAuto:** Die Werte sind Prozentwerte. So steht 100 für 100% (einzeiligen) Zeilenabstand.
- **tmLineSpaceExactly:** Die Werte sind absolute Beträge in Punkt.
- **tmLineSpaceAtLeast:** Die Werte sind absolute Beträge in Punkt.

Beispiel:

```
' Setze den Zeilenabstand auf "Automatisch 150%"
tm.ActiveDocument.Paragraphs(1).LineSpacingRule = LineSpacingAuto
tm.ActiveDocument.Paragraphs(1).PreferredLineSpacing = 150
```

SpaceBefore (Eigenschaft)

Datentyp: **Single**

Liest oder setzt den oberen Abstand des Absatzes in Punkt (1 Punkt entspricht 1/72 Zoll).

SpaceAfter (Eigenschaft)

Datentyp: **Single**

Liest oder setzt den unteren Abstand des Absatzes in Punkt (1 Punkt entspricht 1/72 Zoll).

Alignment (Eigenschaft)

Datentyp: **Long** (TmParagraphAlignment)

Liest oder setzt die Ausrichtung des Absatzes. Mögliche Werte:

```
tmAlignParagraphLeft       = 0 ' linksbündig
tmAlignParagraphRight      = 1 ' rechtsbündig
tmAlignParagraphCenter     = 2 ' zentriert
tmAlignParagraphJustify    = 3 ' Blocksatz
```

Hyphenation (Eigenschaft)

Datentyp: **Long** (TmHyphenation)

Liest oder setzt den Silbentrennungsmodus des Absatzes. Mögliche Werte:

```
tmHypenationNone           = 0 ' keine Silbentrennung
tmHypenationAlways         = 1 ' Silbentrennung wo immer möglich
tmHypenationEvery2Lines    = 2 ' 2-Zeilen-Trennung
tmHypenationEvery3Lines    = 3 ' 3-Zeilen-Trennung
```

OutlineLevel (Eigenschaft)

Datentyp: **Long** (TmOutlineLevel)

Liest oder setzt die Gliederungsebene des Absatzes. Mögliche Werte:

```
tmOutlineLevelBodyText     = 0 ' Textkörper
tmOutlineLevel1            = 1 ' Ebene 1
tmOutlineLevel2            = 2 ' Ebene 2
tmOutlineLevel3            = 3 ' Ebene 3
tmOutlineLevel4            = 4 ' Ebene 4
tmOutlineLevel5            = 5 ' Ebene 5
tmOutlineLevel6            = 6 ' Ebene 6
tmOutlineLevel7            = 7 ' Ebene 7
tmOutlineLevel8            = 8 ' Ebene 8
tmOutlineLevel9            = 9 ' Ebene 9
```

PageBreakBefore (Eigenschaft)

Datentyp: **Boolean**

Liest oder setzt die Eigenschaft "Seitenumbruch" des Absatzes (**True** oder **False**).

ColumnBreakBefore (Eigenschaft)

Datentyp: **Boolean**

Liest oder setzt die Eigenschaft "Spaltenumbruch" des Absatzes (**True** oder **False**).

KeepWithNext (Eigenschaft)

Datentyp: **Boolean**

Liest oder setzt die Eigenschaft "Absätze zusammenhalten" des Absatzes (**True** oder **False**).

KeepTogether (Eigenschaft)

Datentyp: **Boolean**

Liest oder setzt die Eigenschaft "Kein Umbruch im Absatz" des Absatzes (**True** oder **False**).

WidowControl (Eigenschaft)

Datentyp: **Boolean**

Liest oder setzt die Eigenschaft "Schusterjungen vermeiden" des Absatzes (**True** oder **False**).

BorderClearance (Eigenschaft)

Liest oder setzt den Abstand, den die Absatzumrandung vom Text einhalten soll. Jede der vier Seiten kann individuell angesprochen werden.

Syntax 1 (Wert setzen):

```
BorderClearance(Index) = n
```

Syntax 2 (Wert auslesen):

```
n = BorderClearance(Index)
```

Parameter:

Index (Typ: **Long** bzw. **TmBorderClearance**) gibt an, welche Seite des Absatzes angesprochen werden soll:

```
tmBorderClearanceLeft     = 1
tmBorderClearanceRight    = 2
tmBorderClearanceTop      = 3
tmBorderClearanceBottom   = 4
```

n (Typ: **Single**) legt den Abstand in Punkt fest.

Rückgabetyp:

Single

Beispiele:

```
' Im ersten Absatz den linken Abstand zur Umrandung auf 5 pt setzen
tm.ActiveDocument.Paragraphs(1).BorderClearance(tmBorderClearanceLeft) = 5

' Den linken Abstand des ersten Absatzes zur Umrandung ermitteln
MsgBox tm.ActiveDocument.Paragraphs(1).BorderClearance(tmBorderClearanceLeft)
```

Shading (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das **Shading**-Objekt, das die Schattierung des Absatzes beschreibt.

DropCap (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das **DropCap**-Objekt, das den Initialbuchstaben des Absatzes beschreibt.

Range (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das **Range**-Objekt, das die Start- und Endposition des Absatzes, gerechnet in Zeichen ab dem Dokumentanfang, beschreibt.

Application (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das **Application**-Objekt.

Parent (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das übergeordnete Objekt, also ein Objekt vom Typ **Paragraphs**.

Borders (Zeiger auf Sammlung)

Datentyp: **Object**

Liefert die **Borders**-Sammlung, die die fünf Umrandungslinien des Absatzes repräsentiert. Sie können mit Hilfe dieser Sammlung die Linieneinstellungen (Dicke, Farbe etc.) auslesen und verändern.

Range (Objekt)

Zugriffspfade:
- Application → Documents → Item → Paragraphs → Item → **Range**
- Application → ActiveDocument → Paragraphs → Item → **Range**

1 Beschreibung

Das **Range**-Objekt ist ein Tochterobjekt des **Paragraph**-Objekts und liefert Ihnen die Start- und Endposition des Absatzes, gerechnet als Zahl der Zeichen ab dem Dokumentanfang.

2 Zugriff auf das Objekt

Für jedes **Paragraph**-Objekt existiert genau ein **Range**-Objekt. Dieses **Range**-Objekt können Sie ausschließlich über den Objektzeiger **Range** im zugehörigen **Paragraph**-Objekt ansprechen:

```
' Zeige die Endposition des ersten Absatzes des aktuellen Dokuments an
```

```
MsgBox tm.ActiveDocument.Paragraphs.Item(1).Range.End
```

3 Eigenschaften, Objekte, Sammlungen und Methoden

Eigenschaften:
- **Start** R/O
- **End** R/O

Objekte:
- **Application → Application**
- **Parent → Paragraph**

Start (Eigenschaft, R/O)

Datentyp: **Long**

Liefert die Startposition des Absatzes, gerechnet in Zeichen ab dem Dokumentanfang.

End (Eigenschaft, R/O)

Datentyp: **Long**

Liefert die Endposition des Absatzes, gerechnet in Zeichen ab dem Dokumentanfang.

Ein Beispiel für **Start** und **End**:

Wenn der erste Absatz eines Dokuments aus dem Text "Erster Absatz" besteht, trifft folgendes zu:

- **tm.ActiveDocument.Paragraphs.Item(1).Range.Start** liefert den Wert 0 ("nulltes Zeichen ab Dokumentanfang").
- **tm.ActiveDocument.Paragraphs.Item(1).Range.End** liefert 14.

Sie können dann diese Werte verwenden, um den Absatz oder Teile von ihm zu selektieren:

```
' Selektiere die ersten zwei Zeichen des ersten Absatzes
tm.ActiveDocument.Selection.SetRange 0, 1

' Selektiere den gesamten Absatz
With tm.ActiveDocument
  .Selection.SetRange .Paragraphs(1).Range.Start, .Paragraphs(1).Range.End
End With
```

Sie können auch beispielsweise folgendermaßen die ersten vier Absätze eines Dokuments selektieren:

```
With tm.ActiveDocument
  .Selection.SetRange .Paragraphs(1).Range.Start, .Paragraphs(4).Range.End
End With
```

Application (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das **Application**-Objekt.

Parent (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das übergeordnete Objekt, also ein Objekt vom Typ **Paragraph**.

DropCap (Objekt)

Zugriffspfade:
- Application → Documents → Item → Paragraphs → Item → **DropCap**
- Application → ActiveDocument → Paragraphs → Item → **DropCap**

1 Beschreibung

Das **DropCap**-Objekt beschreibt den Initialbuchstaben eines Absatzes. Es ist ein Tochterobjekt von **Paragraph** und lässt Sie alle Eigenschaften des Initialbuchstaben lesen und setzen.

2 Zugriff auf das Objekt

Jeder Absatz besitzt genau eine Instanz des **DropCap**-Objekts. Diese wird über den Objektzeiger **DropCap** im **Paragraph**-Objekt angesprochen:

```
' Initialbuchstaben für den ersten Absatz einschalten
tm.ActiveDocument.Paragraphs(1).DropCap.Position = tmDropNormal

' ... und die Schrift des Initialbuchstaben ändern
tm.ActiveDocument.Paragraphs(1).DropCap.FontName = "Arial"
```

3 Eigenschaften, Objekte, Sammlungen und Methoden

Eigenschaften:
- **FontName**
- **Size**
- **Position**
- **LeftMargin**
- **RightMargin**
- **TopMargin**
- **BottomMargin**

Objekte:
- **Application** → **Application**
- **Parent** → **Paragraph**

FontName (Eigenschaft)

Datentyp: **String**

Liest oder setzt den Namen der Schriftart für den Initialbuchstaben.

Size (Eigenschaft)

Datentyp: **Single**

Liest oder setzt die Schriftgröße des Initialbuchstaben in Punkt.

Position (Eigenschaft)

Datentyp: **Long** (TmDropPosition)

Liest oder setzt die Art, in der der Initialbuchstabe positioniert wird. Mögliche Werte:

```
tmDropNone        = 0 ' Kein Initialbuchstabe
tmDropNormal      = 1 ' Im Absatz
tmDropMargin      = 2 ' Links vom Absatz
tmDropBaseLine    = 3 ' Auf der Basislinie
```

LeftMargin (Eigenschaft)

Datentyp: **Single**

Liest oder setzt den linken Rand des Initialbuchstaben in Punkt (1 Punkt entspricht 1/72 Zoll).

RightMargin (Eigenschaft)

Datentyp: **Single**

Liest oder setzt den rechten Rand des Initialbuchstaben in Punkt (1 Punkt entspricht 1/72 Zoll).

TopMargin (Eigenschaft)

Datentyp: **Single**

Liest oder setzt den oberen Rand des Initialbuchstaben in Punkt (1 Punkt entspricht 1/72 Zoll).

BottomMargin (Eigenschaft)

Datentyp: **Single**

Liest oder setzt den unteren Rand des Initialbuchstaben in Punkt (1 Punkt entspricht 1/72 Zoll).

Application (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das **Application**-Objekt

Parent (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das übergeordnete Objekt, also ein Objekt vom Typ **Paragraph**.

Tables (Sammlung)

Zugriffspfade:
- Application → Documents → Item → **Tables**
- Application → ActiveDocument → **Tables**

1 Beschreibung

Tables ist eine Sammlung aller Tabellen eines Dokuments. Die einzelnen Elemente dieser Sammlung sind vom Typ **Table**.

2 Zugriff auf die Sammlung

Jedes geöffnete Dokument besitzt genau eine Instanz der **Tables**-Sammlung. Diese wird über **Document.Tables** angesprochen:

```
' Anzahl der Tabellen des aktiven Dokuments anzeigen
MsgBox tm.ActiveDocument.Tables.Count
```

3 Eigenschaften, Objekte, Sammlungen und Methoden

Eigenschaften:
- **Count** R/O

Objekte:
- ***Item*** → **Table** (Defaultobjekt)
- **Application** → **Application**
- **Parent** → **Document**

Methoden:
- **Add**

Count (Eigenschaft, R/O)

Datentyp: **Long**

Liefert die Anzahl der **Table**-Objekte des Dokuments – in anderen Worten: die Anzahl der Tabellen im Dokument.

Item (Zeiger auf Objekt)

Datentyp: **Object**

Liefert ein einzelnes **Table**-Objekt, also eine einzelne Tabelle.

Welches Table-Objekt Sie erhalten, hängt von dem Parameter ab, den Sie an **Item** übergeben. Dies kann entweder der numerische Index oder der Name der gewünschten Tabelle sein. Beispiele:

```
' Die Zahl der Zeilen in der ersten Tabelle anzeigen
MsgBox tm.Tables.Item(1).Rows.Count

' Die Zahl der Zeilen in der Tabelle "Tabelle1" anzeigen
MsgBox tm.Tables.Item("Tabelle1").Rows.Count
```

Application (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das **Application**-Objekt.

Parent (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das übergeordnete Objekt, also ein Objekt vom Typ **Document**.

Add (Methode)

Fügt dem Dokument an der aktuellen Selektion eine neue Tabelle hinzu.

Syntax:

```
Add NumRows, NumColumns
```

Parameter:

NumRows (Typ: **Long**) legt die Zahl der Zeilen der neuen Tabelle fest. Geben Sie 0 oder einen Wert kleiner als Null an, wird der Standardwert 3 verwendet.

NumColumns (Typ: **Long**) legt die Zahl der Spalten der neuen Tabelle fest. Geben Sie 0 oder einen Wert kleiner als Null an, wird der Standardwert 3 verwendet.

Rückgabetyp:

Object (ein **Table**-Objekt, das die neue Tabelle repräsentiert)

Beispiele:

```
' Dem Dokument eine 3*3-Tabelle hinzufügen
tm.ActiveDocument.Tables.Add 3, 3

' dito, aber mit der Tabelle als Objekt direkt weiterarbeiten
Dim newTable as Object
Set newTable = tm.ActiveDocument.Tables.Add(3, 3)
MsgBox newTable.Rows.Count ' Zahl der Tabellenzeilen anzeigen
```

Table (Objekt)

Zugriffspfade:
- Application → Documents → Item → Tables → **Item**
- Application → ActiveDocument → Tables → **Item**

1 Beschreibung

Ein **Table**-Objekt repräsentiert eine einzelne Tabelle des Dokuments und erlaubt es Ihnen, deren Formatierung zu ändern.

Für jede Tabelle existiert ein eigenes **Table**-Objekt. Fügen Sie einem Dokument Tabellen hinzu oder löschen diese, werden die zugehörigen **Table**-Objekte dynamisch angelegt bzw. entfernt.

2 Zugriff auf das Objekt

Die einzelnen **Table**-Objekte können ausschließlich durch Aufzählung der Elemente der **Tables**-Sammlung angesprochen werden. Von dieser Sammlung besitzt jedes Dokument genau eine Instanz.

Ein Beispiel:

```
' Die erste Tabelle des Dokuments in Text wandeln
tm.ActiveDocument.Tables.Item(1).ConvertToText
```

3 Eigenschaften, Objekte, Sammlungen und Methoden

Objekte:
- **Shading → Shading**
- **Cell → Cell**
- **Application → Application**

- **Parent → Tables**

Sammlungen:
- **Rows → Rows**
- **Borders → Borders**

Methoden:
- **ConvertToText**

Shading (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das zur Tabelle gehörende **Shading**-Objekt, das die Schattierung der gesamten Tabelle repräsentiert.

Cell (Zeiger auf Objekt)

Datentyp: **Object**

Liefert ein **Cell**-Objekt zurück, das die durch Zeile und Spalte bezeichnete Tabellenzelle repräsentiert.

Syntax:

```
Cell(Row, Column)
```

Parameter:

Row (Typ: **Long**) bezeichnet die Zeile der gewünschten Zelle innerhalb der Tabelle.

Column (Typ: **Long**) bezeichnet die Spalte der gewünschten Zelle innerhalb der Tabelle.

Beispiele:

```
' Zelle B3 der ersten Tabelle auf "vertikal zentrieren" stellen
With tm.ActiveDocument
  .Tables(1).Cell(2,3).Format.VerticalAlignment = tmCellVerticalAlignmentCenter
End With

' Dasselbe, nur umständlicher durch den Umweg über die Rows-Sammlung
With tm.ActiveDocument
  .Tables(1).Rows(2).Cells(3).Format.VerticalAlignment = tmCellVerticalAlignmentCenter
End With
```

Application (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das **Application**-Objekt

Parent (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das übergeordnete Objekt, also ein Objekt vom Typ **Tables**.

Rows (Zeiger auf Sammlung)

Datentyp: **Object**

Liefert die zur Tabelle gehörende **Rows**-Sammlung. Über diese können Sie die einzelnen Zeilen der Tabelle aufzählen, um deren Formatierung abzufragen oder zu ändern.

Borders (Zeiger auf Sammlung)

Datentyp: **Object**

Liefert die **Borders**-Sammlung, die die sechs Umrandungslinien der Tabelle repräsentiert. Sie können mit Hilfe dieser Sammlung die Linieneinstellungen (Dicke, Farbe etc.) auslesen und verändern.

ConvertToText (Methode)

Konvertiert die Tabelle in Text.

Syntax:

```
ConvertToText [Separator]
```

Parameter:

Separator (optional; Typ: entweder **String** oder **Long** bzw. **TmTableFieldSeparator**) gibt das Zeichen an, das TextMaker zum Trennen der Spalten einfügt. Geben Sie entweder ein beliebiges Zeichen an oder eine der folgenden symbolischen Konstanten:

```
tmSeparateByCommas        = 0 ' Spaltentrennung durch Komma
tmSeparateByParagraphs    = 1 ' Spaltentrennung durch Absatzende
tmSeparateByTabs          = 2 ' Spaltentrennung durch Tabulator
tmSeparateBySemicolons    = 3 ' Spaltentrennung durch Strichpunkt
```

Wenn Sie den Parameter weglassen, wird **tmSeparateByTabs** verwendet.

Rückgabetyp:

Object (ein **Range**-Objekt, das den konvertierten Text repräsentiert)

Beispiel:

```
' Die erste Tabelle des Dokuments in Fließtext wandeln
tm.ActiveDocument.Tables.Item(1).ConvertToText tmSeparateByTabs
```

Rows (Sammlung)

Zugriffspfade:
- Application → Documents → Item → Tables → Item → **Rows**
- Application → ActiveDocument → Tables → Item → **Rows**

1 Beschreibung

Rows ist eine Sammlung aller Tabellenzeilen einer Tabelle. Die einzelnen Elemente dieser Sammlung sind vom Typ **Row**.

2 Zugriff auf die Sammlung

Jede Tabelle besitzt genau eine Instanz der **Rows**-Sammlung. Diese wird über den Objektzeiger **Rows** der Tabelle angesprochen:

```
' Zahl der Tabellenzeilen der ersten Tabelle des Dokuments anzeigen
MsgBox tm.ActiveDocument.Tables(1).Rows.Count
```

3 Eigenschaften, Objekte, Sammlungen und Methoden

Eigenschaften:
- **Count** R/O

Objekte:
- ***Item*** → **Row** (Defaultobjekt)
- **Application** → **Application**
- **Parent** → **Table**

Count (Eigenschaft, R/O)

Datentyp: **Long**

Liefert die Anzahl der **Row**-Objekte der Tabelle – in anderen Worten: die Anzahl der Zeilen in der Tabelle.

Item (Zeiger auf Objekt)

Datentyp: **Object**

Liefert ein einzelnes **Row**-Objekt, also eine einzelne Tabellenzeile.

Welches Row-Objekt Sie erhalten, hängt von dem Zahlenwert ab, den Sie an **Item** übergeben: 1 für die erste Zeile der Tabelle, 2 für die zweite etc.

Application (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das **Application**-Objekt.

Parent (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das übergeordnete Objekt, also ein Objekt vom Typ **Table**.

Row (Objekt)

Zugriffspfade:
- Application → Documents → Item → Tables → Item → Rows → **Item**
- Application → ActiveDocument → Tables → Item → Rows → **Item**

1 Beschreibung

Ein **Row**-Objekt repräsentiert eine einzelne Tabellenzeile einer Tabelle und erlaubt es Ihnen, die Formatierung dieser Tabellenzeile zu ändern.

Für jede Tabellenzeile existiert ein eigenes **Row**-Objekt. Fügen Sie einer Tabelle Zeilen hinzu oder löschen diese, werden die zugehörigen **Row**-Objekte dynamisch angelegt bzw. entfernt.

2 Zugriff auf das Objekt

Die einzelnen **Row**-Objekte können ausschließlich durch Aufzählung der Elemente der **Rows**-Sammlung angesprochen werden. Von dieser Sammlung besitzt jede Tabelle genau eine Instanz.

Ein Beispiel:

```
' Die Höhe der zweiten Zeile der ersten Tabelle des Dokuments anzeigen
MsgBox tm.ActiveDocument.Tables(1).Rows.Item(2).Height
```

3 Eigenschaften, Objekte, Sammlungen und Methoden

Eigenschaften:
- **Height**
- **HeightRule**
- **KeepTogether**
- **BreakPageAtRow**
- **AllowBreakInRow**
- **RepeatAsHeaderRow**

Objekte:
- **Shading → Shading**
- **Application → Application**
- **Parent → Rows**

Sammlungen:
- **Cells → Cells**
- **Borders → Borders**

Height (Eigenschaft)

Datentyp: **Single**

Liest oder setzt die Höhe der durch **Row** repräsentierten Tabellenzeile in Punkt (1 Punkt entspricht 1/72 Zoll).

Zwei Besonderheiten gibt es zu beachten, wenn die **HeightRule**-Eigenschaft (siehe unten) der Tabellenzeile auf "Automatisch" gestellt ist:

- Beim Lesen der Eigenschaft wird der Wert **SmoUndefined** (9.999.999) zurückgeliefert.
- Beim Setzen der Eigenschaft wird die Methode zur Festlegung der Zeilenhöhe (**HeightRule**) auf "Mindestens" geändert.

HeightRule (Eigenschaft)

Datentyp: **Long** (TmRowHeightRule)

Liest oder setzt die Methode zur Festlegung der Höhe der durch **Row** repräsentierten Tabellenzeile. Mögliche Werte:

```
tmRowHeightAuto          = 0 ' Zeilenhöhe "automatisch"
tmRowHeightExact         = 1 ' Zeilenhöhe "exakt"
tmRowHeightAtLeast       = 2 ' Zeilenhöhe "mindestens"
```

KeepTogether (Eigenschaft)

Datentyp: **Boolean**

Liest oder setzt die Eigenschaft "Mit der nächsten Zeile zusammenhalten".

Bei **True** wird verhindert, dass TextMaker zwischen dieser Tabellenzeile und der nächsten einen automatischen Seitenumbruch einfügt. Der Umbruch wird dann bereits *vor* der aktuellen Tabellenzeile durchgeführt.

BreakPageAtRow (Eigenschaft)

Datentyp: **Boolean**

Liest oder setzt die Eigenschaft "Seitenumbruch vor dieser Zeile". Bei **True** fügt TextMaker vor dieser Tabellenzeile einen Seitenumbruch ein.

AllowBreakInRow (Eigenschaft)

Datentyp: **Boolean**

Liest oder setzt die Eigenschaft "Seitenumbruch in Zeile erlauben".

Bei **True** ist TextMaker berechtigt, bei Bedarf auch *innerhalb* der Zeile einen Seitenumbruch durchzuführen. Bei **False** wird die gesamte Tabellenzeile auf die nächste Seite übernommen.

RepeatAsHeaderRow (Eigenschaft)

Datentyp: **Boolean**

Liest oder setzt die Eigenschaft "Diese Zeile als Kopfzeile wiederholen". Diese Eigenschaft ist nur für die erste Zeile einer Tabelle verfügbar.

Bei **True** wiederholt TextMaker diese Zeile auf jeder neuen Seite, wenn sich die Tabelle über zwei oder mehr Seiten erstreckt. Dies ist nützlich, um Tabellenüberschriften auf jeder Seite erscheinen zu lassen.

Shading (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das zur **Row** gehörende **Shading**-Objekt, das die Schattierung der gesamten Tabellenzeile repräsentiert.

Application (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das **Application**-Objekt.

Parent (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das übergeordnete Objekt, also ein Objekt vom Typ **Rows**.

Cells (Zeiger auf Sammlung)

Datentyp: **Object**

Liefert die zur Tabelle gehörende **Cells**-Sammlung, die alle Zellen der Tabellenzeile enthält.

Borders (Zeiger auf Sammlung)

Datentyp: **Object**

Liefert die **Borders**-Sammlung, die die fünf Umrandungslinien der Tabellenzeile repräsentiert. Sie können mit Hilfe dieser Sammlung die Linieneinstellungen (Dicke, Farbe etc.) auslesen und verändern.

Cells (Sammlung)

Zugriffspfade:
- Application → Documents → Item → Tables → Item → Rows → Item → **Cells**
- Application → ActiveDocument → Tables → Item → Rows → Item → **Cells**

1 Beschreibung

Cells ist eine Sammlung aller Tabellenzellen einer einzelnen Tabellenzeile. Die einzelnen Elemente dieser Sammlung sind vom Typ **Cell**.

2 Zugriff auf die Sammlung

Jede Zeile einer Tabelle besitzt genau eine Instanz der **Cells**-Sammlung. Diese wird über den Objektzeiger **Cells** der Tabellenzeile angesprochen:

```
' Zahl der Zellen in der zweiten Zeile der ersten Tabelle anzeigen
MsgBox tm.ActiveDocument.Tables(1).Rows(2).Cells.Count
```

3 Eigenschaften, Objekte, Sammlungen und Methoden

Eigenschaften:
- **Count** R/O

Objekte:
- ***Item*** → **Cell** (Defaultobjekt)
- **Application** → **Application**
- **Parent** → **Row**

Count (Eigenschaft, R/O)

Datentyp: **Long**

Liefert die Anzahl der **Cell**-Objekte der Tabellenzeile – in anderen Worten: die Anzahl der Zeilen in der betreffenden Tabellenzeile.

Item (Zeiger auf Objekt)

Datentyp: **Object**

Liefert ein einzelnes **Cell**-Objekt, also eine einzelne Tabellenzelle.

Welches Cell-Objekt Sie erhalten, hängt von dem Zahlenwert ab, den Sie an **Item** übergeben: 1 für die erste Zelle der Tabellenzeile, 2 für die zweite etc.

Application (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das **Application**-Objekt.

Parent (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das übergeordnete Objekt, also ein Objekt vom Typ **Table**.

Cell (Objekt)

Zugriffspfade:
- Application → Documents → Item → Tables → Item → **Cell(x, y)** → **Item**
- Application → ActiveDocument → Tables → Item → **Cell(x, y)** → **Item**
- Application → Documents → Item → Tables → Item → Rows → Item → Cells → **Item**
- Application → ActiveDocument → Tables → Item → Rows → Item → Cells → **Item**

1 Beschreibung

Ein **Cell**-Objekt repräsentiert eine einzelne Zelle einer Tabellenzeile und erlaubt es Ihnen, die Formatierung dieser Tabellenzelle zu ermitteln und zu ändern.

Für jede Zelle existiert ein eigenes **Cell**-Objekt. Fügen Sie einer Tabellenzeile Zellen hinzu oder löschen diese, werden die zugehörigen **Cell**-Objekte dynamisch angelegt bzw. entfernt.

2 Zugriff auf das Objekt

Die einzelnen **Cell**-Objekte können ausschließlich durch Aufzählung der Elemente der **Cells**-Sammlung angesprochen werden. Von dieser Sammlung besitzt jede Zeile einer Tabelle genau eine Instanz.

Ein Beispiel:

```
' Breite der fünften Zelle in der zweiten Zeile der ersten Tabelle auf 25 setzen
tm.ActiveDocument.Tables(1).Rows(2).Cells(5).PreferredWidth = 25
```

3 Eigenschaften, Objekte, Sammlungen und Methoden

Eigenschaften:
- **PreferredWidthType**
- **PreferredWidth**
- **Width**
- **VerticalAlignment**
- **Orientation**
- **LockText**
- **LeftPadding**
- **RightPadding**
- **TopPadding**
- **BottomPadding**

Objekte:
- **Shading → Shading**
- **Application → Application**
- **Parent → Row**

Sammlungen:
- **Borders → Borders**

PreferredWidthType (Eigenschaft)

Datentyp: **Long** (TmPreferredWidthType)

Liest oder setzt den Breitenmodus der Zelle. Mögliche Werte:

```
tmPreferredWidthPoints     = 0 ' Breite in Punkt
tmPreferredWidthPercent    = 1 ' Breite in Prozent
tmPreferredWidthAuto       = 2 ' Breite automatisch
```

PreferredWidth (Eigenschaft)

Datentyp: **Single**

Liest oder setzt die Breite der Zelle. Ob der Wert in Punkt oder Prozent zu verstehen ist, hängt vom Breitenmodus der Zelle ab (siehe **PreferredWidthType** weiter oben).

Beispiel:

```
' Erste Zelle auf eine Breite von 25 Prozent setzen
tm.ActiveDocument.Tables(1).Rows(1).Cells(1).PreferredWidthType =
   tmPreferredWidthPercent
tm.ActiveDocument.Tables(1).Rows(1).Cells(1).PreferredWidth = 25

' Zweite Zelle auf 3,5 cm Breite setzen
tm.ActiveDocument.Tables(1).Rows(1).Cells(2).PreferredWidthType = tmPreferredWidthPoints
tm.ActiveDocument.Tables(1).Rows(1).Cells(1).PreferredWidth =
   tm.CentimetersToPoints(3.5)
```

Width (Eigenschaft)

Datentyp: **Single**

Liest oder setzt die Breite der Zelle in Punkt (1 Punkt entspricht 1/72 Zoll).

Im Gegensatz zur Eigenschaft **PreferredWidth** (siehe dort) wird hier ignoriert, ob die Zelle eine absolute, eine prozentuale oder eine automatische Breite besitzt – es wird immer die Breite in Punkt geliefert.

VerticalAlignment (Eigenschaft)

Datentyp: **Long** (TmCellVerticalAlignment)

Liest oder setzt die vertikale Ausrichtung des Textes innerhalb der Zelle. Mögliche Werte:

```
tmCellVerticalAlignmentTop           = 0 ' oben
tmCellVerticalAlignmentCenter        = 1 ' zentriert
tmCellVerticalAlignmentBottom        = 2 ' unten
tmCellVerticalAlignmentJustify       = 3 ' vertikaler Blocksatz
```

Orientation (Eigenschaft)

Datentyp: **Long**

Liest oder setzt für die Druckrichtung der Zelle. Mögliche Werte: 0, 90, 180 und -90, entsprechend den jeweiligen Drehwinkeln.

Hinweis: Der Wert 270 wird automatisch in -90 gewandelt.

LockText (Eigenschaft)

Datentyp: **Boolean**

Liest oder setzt die Eigenschaft "Text sperren" für die Zelle (**True** oder **False**). Beachten Sie, dass TextMaker die Zelle nur bei eingeschaltetem Formularmodus für Texteingaben sperrt.

LeftPadding (Eigenschaft)

Datentyp: **Single**

Liest oder setzt den linken Rand des Textes innerhalb der Zelle. Die Maßeinheit ist Punkt (1 Punkt entspricht 1/72 Zoll).

RightPadding (Eigenschaft)

Datentyp: **Single**

Liest oder setzt den rechten Rand des Textes innerhalb der Zelle. Die Maßeinheit ist Punkt (1 Punkt entspricht 1/72 Zoll).

TopPadding (Eigenschaft)

Datentyp: **Single**

Liest oder setzt den oberen Rand des Textes innerhalb der Zelle. Die Maßeinheit ist Punkt (1 Punkt entspricht 1/72 Zoll).

BottomPadding (Eigenschaft)

Datentyp: **Single**

Liest oder setzt den unteren Rand des Textes innerhalb der Zelle. Die Maßeinheit ist Punkt (1 Punkt entspricht 1/72 Zoll).

Shading (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das **Shading**-Objekt, mit dem Sie auf die Schattierung der Tabellenzelle zugreifen können.

Application (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das **Application**-Objekt.

Parent (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das übergeordnete Objekt, also ein Objekt vom Typ **Row**.

Borders (Zeiger auf Sammlung)

Datentyp: **Object**

Liefert eine **Borders**-Sammlung, die die vier Umrandungslinien der Tabellenzelle repräsentiert. Sie können mit Hilfe dieser Sammlung die Linieneinstellungen (Dicke, Farbe etc.) auslesen und verändern.

Borders (Sammlung)

Zugriffspfade für Umrandungen von Absätzen:

- Application → Documents → Item → Paragraphs → Item → **Borders**

- Application → ActiveDocument → Paragraphs → Item → **Borders**

Zugriffspfade für Umrandungen von Tabellen:
- Application → Documents → Item → Tables → Item → **Borders**
- Application → ActiveDocument → Tables → Item → **Borders**

Zugriffspfade für Umrandungen von Tabellenzeilen:
- Application → Documents → Item → Tables → Item → Rows → Item → **Borders**
- Application → ActiveDocument → Tables → Item → Rows → Item → **Borders**

Zugriffspfade für Umrandungen von Tabellenzellen:
- Application → Documents → Item → Tables → Item → Cell(x, y) → **Borders**
- Application → ActiveDocument → Tables → Item → Cell(x, y) → **Borders**
- Application → Documents → Item → Tables → Item → Rows → Item → Cells → Item → **Borders**
- Application → ActiveDocument → Tables → Item → Rows → Item → Cells → Item → **Borders**

1 Beschreibung

Borders ist die Sammlung aller Umrandungslinien (links, rechts, oben, unten etc.) von Absätzen, Tabellen, Tabellenzeilen und Tabellenzellen. Dementsprechend ist es ein Tochterobjekt von entweder **Paragraph**, **Table**, **Row** oder **Cell**.

Die einzelnen Elemente dieser Sammlung sind vom Typ **Border**.

2 Zugriff auf das Objekt

Jeder Absatz, jede Tabelle, jede Tabellenzeile und jede Tabellenzelle besitzt genau eine Instanz der **Borders**-Sammlung. Diese wird über den Objektzeiger **Borders** im jeweiligen Objekt angesprochen. Hierbei übergeben Sie als Parameter die Nummer der Umrandungslinie, die Sie ansprechen möchten:

```
tmBorderTop           = -1 ' Obere Umrandungslinie
tmBorderLeft          = -2 ' Linke Umrandungslinie
tmBorderBottom        = -3 ' Untere Umrandungslinie
tmBorderRight         = -4 ' Rechte Umrandungslinie
tmBorderHorizontal    = -5 ' Horizontale Gitterlinie (nur bei Tabellen)
tmBorderVertical      = -6 ' Vertikale Gitterlinie (nur Tabellen, Tabellenzeilen)
tmBorderBetween       = -7 ' Umrandungslinie zwischen Absätzen (nur Absätze)
```

Beispiele:

```
' Linke Umrandung des ersten Absatzes ändern
tm.ActiveDocument.Paragraphs(1).Borders(tmBorderLeft).Type = tmLineStyleSingle

' Obere Umrandung der ersten Tabelle ändern
tm.ActiveDocument.Tables(1).Borders(tmBorderTop).Type = tmLineStyleDouble

' Vertikale Gitterlinien der der zweiten Zeile der ersten Tabelle ändern
tm.ActiveDocument.Tables(1).Rows(2).Borders(tmBorderVertical).Color = smoColorRed

' Untere Umrandung der dritten Zelle der zweiten Zeile der ersten Tabelle ändern
tm.ActiveDocument.Tables(1).Rows(2).Cells(3).Borders(tmBorderBottom).Type =
   tmLineStyleDouble
```

3 Eigenschaften, Objekte, Sammlungen und Methoden

Eigenschaften:
- **Count** R/O

Objekte:
- ***Item*** → **Border** (Defaultobjekt)
- **Application** → **Application**

- **Parent** → **Paragraph**, **Table**, **Row** oder **Cell**

Count (Eigenschaft, R/O)

Datentyp: **Long**

Liefert die Anzahl der **Border**-Objekte in der Sammlung, also die Zahl der möglichen Umrandungslinien des zugehörigen Objekts:

- Als Tochterobjekt eines **Paragraph**-Objekts liefert **Count** den Wert **5**, da ein Absatz fünf Umrandungslinien (links, rechts, oben, unten sowie zwischen Absätzen) kennt.
- Als Tochterobjekt eines **Table**-Objekts liefert **Count** den Wert **6**, da eine Tabelle sechs Umrandungslinien (links, rechts, oben, horizontales Gitter, vertikales Gitter) kennt.
- Als Tochterobjekt eines **Row**-Objekts liefert **Count** den Wert **5**, da eine Tabellenzeile fünf Umrandungslinien (links, rechts, oben, unten und vertikales Gitter) kennt.
- Als Tochterobjekt eines **Cell**-Objekts liefert **Count** den Wert **4**, da eine Tabellenzelle vier Umrandungslinien (links, rechts, oben und unten) kennt.

Item (Zeiger auf Objekt)

Datentyp: **Object**

Liefert ein einzelnes **Border**-Objekt, mit dem Sie eine einzelne Umrandungslinie ansprechen können, um deren Eigenschaften (etwa Farbe und Dicke) auszulesen oder zu setzen.

Welches Border-Objekt Sie erhalten, hängt von dem Zahlenwert ab, den Sie an **Item** übergeben. Die folgende Tabelle zeigt die erlaubten Werte:

```
tmBorderTop          = -1 ' Obere Umrandungslinie
tmBorderLeft         = -2 ' Linke Umrandungslinie
tmBorderBottom       = -3 ' Untere Umrandungslinie
tmBorderRight        = -4 ' Rechte Umrandungslinie
tmBorderHorizontal   = -5 ' Horizontale Gitterlinie (nur bei Tabellen)
tmBorderVertical     = -6 ' Vertikale Gitterlinie (nur Tabellen, -zeilen)
tmBorderBetween      = -7 ' Umrandungslinie zwischen Absätzen (nur Absätze)
```

Application (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das **Application**-Objekt

Parent (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das übergeordnete Objekt, also ein Objekt vom Typ **Paragraph**, **Table**, **Row** oder **Cell**.

Beispiel für die Anwendung der Borders-Sammlung

```
Sub Main
  Dim tm as Object

  Set tm = CreateObject("TextMaker.Application")
  tm.Visible = True
```

```
  With tm.ActiveDocument.Paragraphs.Item(1)
     .Borders(tmBorderLeft).Type    = tmLineStyleSingle
     .Borders(tmBorderLeft).Thick1  = 4
     .Borders(tmBorderLeft).Color   = smoColorBlue

     .Borders(tmBorderRight).Type   = tmLineStyleDouble
     .Borders(tmBorderRight).Thick1 = 1
     .Borders(tmBorderRight).Thick2 = 1
     .Borders(tmBorderRight).Color  = smoColorRed
  End With

  Set tm = Nothing
End Sub
```

Border (Objekt)

Zugriffspfade für Umrandungen von Absätzen:
- Application → Documents → Item → Paragraphs → Item → Borders → **Item**
- Application → ActiveDocument → Paragraphs → Item → Borders → **Item**

Zugriffspfade für Umrandungen von Tabellen:
- Application → Documents → Item → Tables → Item → Borders → **Item**
- Application → ActiveDocument → Tables → Item → Borders → **Item**

Zugriffspfade für Umrandungen von Tabellenzeilen:
- Application → Documents → Item → Tables → Item → Rows → Item → Borders → **Item**
- Application → ActiveDocument → Tables → Item → Rows → Item → Borders → **Item**

Zugriffspfade für Umrandungen von Tabellenzellen:
- Application → Documents → Item → Tables → Item → Cell(x, y) → Borders → **Item**
- Application → ActiveDocument → Tables → Item → Cell(x, y) → Borders → **Item**
- Application → Documents → Item → Tables → Item → Rows → Item → Cells → Item → Borders → **Item**
- Application → ActiveDocument → Tables → Item → Rows → Item → Cells → Item → Borders → **Item**

1 Beschreibung

Ein **Border**-Objekt repräsentiert jeweils eine der Umrandungslinien von Absätzen, Tabellen, Tabellenzeilen beziehungsweise Tabellenzellen (z.B. die obere, untere, linke oder rechte Linie). Es lässt Sie die Liniendicke, Farbe etc. dieser Umrandungslinie auslesen und setzen.

2 Zugriff auf das Objekt

Die einzelnen **Border**-Objekte können ausschließlich über die **Borders**-Sammlung eines Absatzes, einer Tabelle, einer Tabellenzeile oder einer Tabellenzelle angesprochen werden. Hierbei übergeben Sie der **Borders**-Sammlung als Parameter die Nummer der Umrandungslinie, die Sie ansprechen möchten:

```
tmBorderTop          = -1 ' Obere Umrandungslinie
tmBorderLeft         = -2 ' Linke Umrandungslinie
tmBorderBottom       = -3 ' Untere Umrandungslinie
tmBorderRight        = -4 ' Rechte Umrandungslinie
tmBorderHorizontal   = -5 ' Horizontale Gitterlinie (nur bei Tabellen)
tmBorderVertical     = -6 ' Vertikale Gitterlinie (nur Tabellen, Tabellenzeilen)
tmBorderBetween      = -7 ' Umrandungslinie zwischen Absätzen (nur Absätze)
```

Einige Beispiele:

```
' Linke Umrandung des ersten Absatzes ändern
tm.ActiveDocument.Paragraphs(1).Borders(tmBorderLeft).Type = tmLineStyleSingle

' Obere Umrandung der ersten Tabelle ändern
tm.ActiveDocument.Tables(1).Borders(tmBorderTop).Type = tmLineStyleDouble
```

```
' Vertikale Gitterlinien der der zweiten Zeile der ersten Tabelle ändern
tm.ActiveDocument.Tables(1).Rows(2).Borders(tmBorderVertical).Color = smoColorRed

' Untere Umrandung der dritten Zelle der zweiten Zeile der ersten Tabelle ändern
tm.ActiveDocument.Tables(1).Rows(2).Cells(3).Borders(tmBorderBottom).Type =
   tmLineStyleDouble
```

3 Eigenschaften, Objekte, Sammlungen und Methoden

Eigenschaften:
- **Type**
- **Thick1**
- **Thick2**
- **Separation**
- **Color**
- **ColorIndex**

Objekte:
- **Application → Application**
- **Parent → Borders**

Type (Eigenschaft)

Datentyp: **Long** (TmLineStyle)

Liest oder setzt den Typ der Umrandungslinie. Mögliche Werte:

```
tmLineStyleNone      = 0 ' Keine Umrandung
tmLineStyleSingle    = 1 ' Einfache Umrandung
tmLineStyleDouble    = 2 ' Doppelte Umrandung
```

Thick1 (Eigenschaft)

Datentyp: **Single**

Liest oder setzt die Dicke der ersten Umrandungslinie in Punkt (1 Punkt entspricht 1/72 Zoll).

Thick2 (Eigenschaft)

Datentyp: **Single**

Liest oder setzt die Dicke der zweiten Umrandungslinie in Punkt (1 Punkt entspricht 1/72 Zoll).

Diese Eigenschaft wird nur verwendet, wenn der Typ der Umrandung auf **tmLineStyleDouble** steht.

Separation (Eigenschaft)

Datentyp: **Single**

Liest oder setzt den Abstand zwischen den beiden Umrandungslinien in Punkt (1 Punkt entspricht 1/72 Zoll).

Diese Eigenschaft wird nur verwendet, wenn der Typ der Umrandung auf **tmLineStyleDouble** steht.

Color (Eigenschaft)

Datentyp: **Long** (SmoColor)

Liest oder setzt die Farbe der Umrandungslinie(n) als "BGR"-Wert (Blau-Grün-Rot-Triplet). Sie können entweder einen beliebigen Wert angeben oder eine der vordefinierten BGR-Farbkonstanten verwenden.

ColorIndex (Eigenschaft)

Datentyp: **Long** (SmoColorIndex)

Liest oder setzt die Farbe der Umrandungslinie(n) als Indexfarbe. "Indexfarben" sind die Standardfarben von TextMaker, durchnummeriert von 0 für Schwarz bis 15 für Hellgrau. Sie dürfen ausschließlich einen der in der Tabelle der Indexfarben genannten Werte verwenden.

Hinweis: Wir empfehlen, stattdessen die Eigenschaft **Color** (siehe oben) zu verwenden, da Sie mit dieser nicht auf die 16 Standardfarben beschränkt sind, sondern mit beliebigen Farben der BGR-Palette arbeiten können.

Application (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das **Application**-Objekt

Parent (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das übergeordnete Objekt, also **Borders**.

Shading (Objekt)

Zugriffspfade für Schattierungen von Absätzen:
- Application → Documents → Item → Paragraphs → Item → **Shading**
- Application → ActiveDocument → Paragraphs → Item → **Shading**

Zugriffspfade für Schattierungen von Tabellen:
- Application → Documents → Item → Tables → Item → **Shading**
- Application → ActiveDocument → Tables → Item → **Shading**

Zugriffspfade für Schattierungen von Tabellenzeilen:
- Application → Documents → Item → Tables → Item → Rows → Item → **Shading**
- Application → ActiveDocument → Tables → Item → Rows → Item → **Shading**

Zugriffspfade für Schattierungen von Tabellenzellen:
- Application → Documents → Item → Tables → Item → Cell(x, y) → **Shading**
- Application → ActiveDocument → Tables → Item → Cell(x, y) → **Shading**
- Application → Documents → Item → Tables → Item → Rows → Item → Cells → Item → **Shading**
- Application → ActiveDocument → Tables → Item → Rows → Item → Cells → Item → **Shading**

1 Beschreibung

Das **Shading**-Objekt beschreibt die Schattierung von Absätzen, Tabellen, Tabellenzeilen und Tabellenzellen. Es ist ein Tochterobjekt von **Paragraph**, von **Table**, von **Row** oder von **Cell**.

2 Zugriff auf das Objekt

Jeder Absatz, jede Tabelle, jede Tabellenzeile und jede Tabellenzelle besitzt genau eine Instanz des **Shading**-Objekts. Diese wird über den Objektzeiger **Shading** im jeweiligen Objekt angesprochen:

```
' Schattierung des ersten Absatzes ändern
tm.ActiveDocument.Paragraphs(1).Shading.Texture = smoPatternHalftone
```

```
' Schattierung der ersten Tabelle ändern
tm.ActiveDocument.Tables(1).Shading.Texture = smoPatternHalftone

' Schattierung der zweiten Zeile der ersten Tabelle ändern
tm.ActiveDocument.Tables(1).Rows(2).Shading.Texture = smoPatternHalftone

' Schattierung der dritten Zelle der zweiten Zeile der ersten Tabelle ändern
tm.ActiveDocument.Tables(1).Rows(2).Cells(3).Shading.Texture = smoPatternHalftone
```

3 Eigenschaften, Objekte, Sammlungen und Methoden

Eigenschaften:
- **Texture**
- **Intensity**
- **ForegroundPatternColor**
- **ForegroundPatternColorIndex**
- **BackgroundPatternColor**
- **BackgroundPatternColorIndex**

Objekte:
- **Application** → **Application**
- **Parent** → **Paragraph**, **Table**, **Row** oder **Cell**

Texture (Eigenschaft)

Datentyp: **Long** (SmoShadePatterns)

Liest oder setzt das Füllmuster der Schattierung. Mögliche Werte:

```
smoPatternNone                  = 0
smoPatternHalftone              = 1
smoPatternRightDiagCoarse       = 2
smoPatternLeftDiagCoarse        = 3
smoPatternHashDiagCoarse        = 4
smoPatternVertCoarse            = 5
smoPatternHorzCoarse            = 6
smoPatternHashCoarse            = 7
smoPatternRightDiagFine         = 8
smoPatternLeftDiagFine          = 9
smoPatternHashDiagFine          = 10
smoPatternVertFine              = 11
smoPatternHorzFine              = 12
smoPatternHashFine              = 13
```

Um eine *Schattierung* hinzuzufügen, setzen Sie **Texture** auf **smoPatternHalftone** und geben die gewünschte Stärke der Schattierung bei **Intensity** an.

Um ein *Muster* hinzuzufügen, setzen Sie **Texture** auf einen Wert zwischen **smoPatternRightDiagCoarse** und **smoPatternHashFine**.

Um eine Schattierung oder ein Muster wieder zu *entfernen*, setzen Sie **Texture** auf **smoPatternNone**.

Intensity (Eigenschaft)

Datentyp: **Long**

Liest oder setzt die Stärke der Schattierung. Die möglichen Werte liegen zwischen 0 und 100, entsprechend 0 bis 100 Prozent.

Dieser Wert darf nur gesetzt oder gelesen werden, wenn mittels **Texture**-Eigenschaft eine Schattierung angewählt wurde (**Texture** auf **smoPatternHalftone** gesetzt). Ist ein Muster gewählt (**Texture** enthält einen beliebigen anderen Wert), führt der Zugriff auf **Intensity** zu einem Fehler.

ForegroundPatternColor (Eigenschaft)

Datentyp: **Long** (SmoColor)

Liest oder setzt die Vordergrundfarbe der Schattierung oder des Musters als "BGR"-Wert (Blau-Grün-Rot-Triplet). Sie können entweder einen beliebigen Wert angeben oder eine der vordefinierten BGR-Farbkonstanten verwenden.

ForegroundPatternColorIndex (Eigenschaft)

Datentyp: **Long** (SmoColorIndex)

Liest oder setzt die Vordergrundfarbe der Schattierung oder des Musters als Indexfarbe. "Indexfarben" sind die 16 Standardfarben von TextMaker, durchnummeriert von 0 für Schwarz bis 15 für Hellgrau. Sie dürfen ausschließlich einen der in der Tabelle der Indexfarben genannten Werte verwenden.

Wir empfehlen, stattdessen die Eigenschaft **ForegroundPatternColor** (siehe dort) zu verwenden, da Sie mit dieser nicht auf die 16 Standardfarben beschränkt sind, sondern mit beliebigen Farben der BGR-Palette arbeiten können.

BackgroundPatternColor (Eigenschaft)

Datentyp: **Long** (SmoColor)

Liest oder setzt die Hintergrundfarbe der Schattierung oder des Musters als "BGR"-Wert (Blau-Grün-Rot-Triplet). Sie können entweder einen beliebigen Wert angeben oder eine der vordefinierten BGR-Farbkonstanten verwenden.

BackgroundPatternColorIndex (Eigenschaft)

Datentyp: **Long** (SmoColorIndex)

Liest oder setzt die Hintergrundfarbe der Schattierung oder des Musters als Indexfarbe. "Indexfarben" sind die Standardfarben von TextMaker, durchnummeriert von 0 für Schwarz bis 15 für Hellgrau. Sie dürfen ausschließlich einen der in der Tabelle der Indexfarben genannten Werte verwenden.

Wir empfehlen, stattdessen die Eigenschaft **ForegroundPatternColor** (siehe dort) zu verwenden, da Sie mit dieser nicht auf die 16 Standardfarben beschränkt sind, sondern mit beliebigen Farben der BGR-Palette arbeiten können.

Application (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das **Application**-Objekt

Parent (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das übergeordnete Objekt, also ein Objekt vom Typ **Paragraph**, **Table**, **Row** oder **Cell**.

Beispiel für die Anwendung des Shading-Objekts

```
Sub Main
  Dim tm as Object
```

```
  Set tm = CreateObject("TextMaker.Application")
  tm.Visible = True

  With tm.ActiveDocument.Paragraphs.Item(1)
     .Shading.Texture = smoPatternHorzFine
     .Shading.BackgroundPatternColor = smoColorAqua
  End With

  Set tm = Nothing
End Sub
```

FormFields (Sammlung)

Zugriffspfade:
- Application → Documents → Item → **FormFields**
- Application → ActiveDocument → **FormFields**

1 Beschreibung

FormFields ist eine Sammlung aller Formularobjekte (Textfelder, Kontrollkästchen und Auswahllisten) eines Dokuments. Die einzelnen Elemente dieser Sammlung sind vom Typ **FormField**.

2 Zugriff auf die Sammlung

Jedes geöffnete Dokument besitzt genau eine Instanz der **FormFields**-Sammlung. Diese wird über **Document.FormFields** angesprochen:

```
' Anzahl der Formularfelder im aktiven Dokument anzeigen
MsgBox tm.ActiveDocument.FormFields.Count
```

3 Eigenschaften, Objekte, Sammlungen und Methoden

Eigenschaften:
- **Count** R/O
- **DisplayFieldNames**
- **Shaded**

Objekte:
- ***Item*** → **FormField** (Defaultobjekt)
- **Application** → **Application**
- **Parent** → **Document**

Count (Eigenschaft, R/O)

Datentyp: **Long**

Liefert die Anzahl der **FormField**-Objekte des Dokuments – in anderen Worten: die Anzahl der Formularobjekte im Dokument.

DisplayFieldNames (Eigenschaft)

Datentyp: **Boolean**

Liest oder setzt die Einstellung "Nur Feldnamen anzeigen" des betreffenden Dokuments (**True** oder **False**).

Shaded (Eigenschaft)

Datentyp: **Boolean**

Liest oder setzt die Einstellung "Formularfelder schattieren" des betreffenden Dokuments (**True** oder **False**).

Item (Zeiger auf Objekt)

Datentyp: **Object**

Liefert ein einzelnes **FormField**-Objekt, also ein einzelnes Formularobjekt.

Welches FormField-Objekt Sie erhalten, hängt von dem Parameter ab, den Sie an **Item** übergeben. Dies kann entweder der numerische Index oder der Name des gewünschten Formularobjekts sein. Beispiele:

```
' Den numerischen Typ des ersten Formularfeld des Dokuments anzeigen
MsgBox tm.ActiveDocument.FormFields(1).Type

' Den numerischen Typ des Formularfelds mit dem Namen "DropDown1" anzeigen
MsgBox tm.ActiveDocument.FormFields("DropDown1").Type
```

Application (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das **Application**-Objekt.

Parent (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das übergeordnete Objekt, also ein Objekt vom Typ **Document**.

FormField (Objekt)

Zugriffspfade:
- Application → Documents → Item → FormFields → **Item**
- Application → ActiveDocument → FormFields → **Item**

1 Beschreibung

Ein **FormField**-Objekt repräsentiert ein einzelnes Formularobjekt des Dokuments und erlaubt es Ihnen, den Wert abzufragen, den es zurückliefert, und seine Formatierung zu ändern.

Bei einem Formularobjekt kann es sich entweder um ein Textfeld, ein Kontrollkästchen oder eine Auswahlliste handeln.

Für jedes Formularobjekt existiert ein eigenes **FormField**-Objekt. Fügen Sie einem Dokument Formularobjekte hinzu oder löschen diese, werden die zugehörigen **FormField**-Objekte dynamisch angelegt bzw. entfernt.

2 Zugriff auf das Objekt

Die einzelnen **FormField**-Objekte können ausschließlich durch Aufzählung der Elemente der **FormFields**-Sammlung angesprochen werden. Von dieser Sammlung besitzt jedes Dokument genau eine Instanz.

Ein Beispiel:

```
' Den Namen des ersten Formularobjekts im Dokument anzeigen
```

```
MsgBox tm.ActiveDocument.FormFields(1).Name
```

Textfelder, Kontrollkästchen und Auswahllisten besitzen *allgemeine* Eigenschaften und *typspezifische*. Der Zugriff auf die jeweilige Art von Eigenschaften erfolgt auf unterschiedliche Weise:

- Eigenschaften, die bei allen Formularobjekten *gleichermaßen* verfügbar sind (zum Beispiel, ob sie sichtbar sind), finden Sie direkt im **FormField**-Objekt. Diese Eigenschaften werden im Folgenden dokumentiert.
- Eigenschaften hingegen, die *typspezifisch* sind (beispielsweise besitzen nur Auswahllisten eine Auflistung aller ihrer Elemente), finden Sie in den Tochterobjekten **TextInput**, **CheckBox** und **DropDown**. Diese Eigenschaften werden beim jeweiligen Tochterobjekt dokumentiert.

3 Eigenschaften, Objekte, Sammlungen und Methoden

Eigenschaften:
- **Name**
- **Visible**
- **Printable**
- **Locked**
- **Tabstop**
- **Type** R/O
- **Result** R/O

Objekte:
- **TextInput → TextInput**
- **CheckBox → CheckBox**
- **DropDown → DropDown**
- **Application → Application**
- **Parent → FormFields**

Name (Eigenschaft)

Datentyp: **String**

Liest oder setzt den Namen des Objekts. Entspricht der Option **Name** im Dialogfenster des TextMaker-Befehls **Objekt > Eigenschaften**.

Visible (Eigenschaft)

Datentyp: **Boolean**

Liest oder setzt die Eigenschaft "Sichtbar" des Objekts (**True** oder **False**). Entspricht der Option "Sichtbar" im Dialogfenster des TextMaker-Befehls **Objekt > Eigenschaften**.

Printable (Eigenschaft)

Datentyp: **Boolean**

Liest oder setzt die Eigenschaft "Drucken" des Objekts (**True** oder **False**). Entspricht der Option "Drucken" im Dialogfenster des TextMaker-Befehls **Objekt > Eigenschaften**.

Locked (Eigenschaft)

Datentyp: **Boolean**

Liest oder setzt die Einstellung "Sperren" des Objekts (**True** oder **False**). Entspricht der Option "Sperren" im Dialogfenster des TextMaker-Befehls **Objekt > Eigenschaften**.

Tabstop (Eigenschaft)

Datentyp: **Boolean**

Liest oder setzt, ob das Objekt einen Tabstopp hat (**True** oder **False**). Entspricht der Option **Tabstopp** im Dialogfenster des TextMaker-Befehls **Objekt > Eigenschaften**.

Type (Eigenschaft, R/O)

Datentyp: **Long** (TmFieldType)

Liefert den Typ des Objekts als Zahlenwert. Mögliche Werte:

```
tmFieldFormTextInput                = 1  ' Textfeld
tmFieldFormCheckBox                 = 10 ' Kontrolkästchen
tmFieldFormDropDown                 = 11 ' Auswahlliste
```

Result (Eigenschaft, R/O)

Datentyp: **String**

Liefert das Ergebnis, das das Objekt derzeit liefert:

- Bei **CheckBox:** der Text des Kontrollkästchens, falls diese angekreuzt ist; ansonsten eine leere Zeichenkette
- Bei **DropDown:** der aktuell gewählte Eintrag als Text
- Bei **TextInput:** der Inhalt

TextInput (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das **TextInput**-Objekt, das Sie auf die textfeldspezifischen Eigenschaften des Formularobjekts zugreifen lässt.

Anmerkung: Nur wenn die Eigenschaft **TextInput.Valid** den Wert **True** liefert, handelt es sich bei dem Formularobjekt tatsächlich um ein Textfeld oder einen Textrahmen.

CheckBox (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das **CheckBox**-Objekt, das Sie auf die kontrollkästchenspezifischen Eigenschaften des Formularobjekts zugreifen lässt.

Anmerkung: Nur wenn die Eigenschaft **CheckBox.Valid** den Wert **True** liefert, handelt es sich bei dem Formularobjekt tatsächlich um ein Kontrollkästchen.

DropDown (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das **DropDown**-Objekt, das Sie auf die auswahllistenspezifischen Eigenschaften des Formularobjekts zugreifen lässt.

Anmerkung: Nur wenn die Eigenschaft **DropDown.Valid** den Wert **True** liefert, handelt es sich bei dem Formularobjekt tatsächlich um eine Auswahlliste.

Application (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das **Application**-Objekt.

Parent (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das übergeordnete Objekt, also ein Objekt vom Typ **FormFields**.

TextInput (Objekt)

Zugriffspfade:
- Application → Documents → Item → FormFields → Item → **TextInput**
- Application → ActiveDocument → FormFields → Item → **TextInput**

1 Beschreibung

Ein **TextInput**-Objekt repräsentiert ein einzelnes Formularobjekt vom Typ **TextInput** und erlaubt es Ihnen, dessen Wert abzufragen und zu ändern.

Ein **TextInput**-Objekt kann eine von drei verschiedenen Objektarten sein:

- ein Textfeld (erzeugt mit **Objekt** > **Neues Formularobjekt** > **Textfeld**)
- ein Textrahmen (erzeugt mit **Objekt** > **Neuer Textrahmen**)
- eine Zeichnung, bei der mit **Text hinzufügen** Text hinzugefügt wurde

TextInput ist ein Tochterobjekt von **FormField**.

2 Zugriff auf das Objekt

Das **TextInput**-Objekt lässt sich ausschließlich über sein übergeordnetes **FormField**-Objekt ansprechen.

Nur dann, wenn die Eigenschaft **TextInput.Valid** den Wert **True** liefert, handelt es sich wirklich um ein Textfeld – und nicht etwa ein Kontrollkästchen oder eine Auswahlliste.

Ein Beispiel:

```
' Den Typ des ersten Formularobjekts prüfen.
' Falls es ein TextInput-Objekt ist, dessen Text ausgegeben.

If tm.ActiveDocument.FormFields(1).TextInput.Valid Then
  MsgBox tm.ActiveDocument.FormFields(1).TextInput.Text
End If
```

3 Eigenschaften, Objekte, Sammlungen und Methoden

Eigenschaften:
- ***Text*** (Defaulteigenschaft)
- **Valid** R/O
- **LockText**

Objekte:
- **Application** → **Application**
- **Parent** → **FormField**

Text (Eigenschaft)

Datentyp: **String**

Liest oder setzt den Inhalt des Textfelds.

Valid (Eigenschaft, R/O)

Datentyp: **Boolean**

Liefert **False**, wenn das Objekt kein **TextInput**-Objekt ist.

LockText (Eigenschaft)

Datentyp: **Boolean**

Liest oder setzt die Einstellung "Text sperren" des Textfelds (**True** oder **False**). Entspricht der Option "Text sperren" im Dialogfenster des TextMaker-Befehls **Objekt > Eigenschaften**.

Beachten Sie, dass TextMaker das Textfeld nur bei eingeschaltetem Formularmodus für Texteingaben sperrt.

Application (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das **Application**-Objekt.

Parent (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das übergeordnete Objekt, also ein Objekt vom Typ **FormField**.

CheckBox (Objekt)

Zugriffspfade:
- Application → Documents → Item → FormFields → Item → **CheckBox**
- Application → ActiveDocument → FormFields → Item → **CheckBox**

1 Beschreibung

Ein **CheckBox**-Objekt repräsentiert ein einzelnes Formularobjekt vom Typ **Checkbox** (Kontrollkästchen) und erlaubt es Ihnen, dessen Wert abzufragen und zu ändern.

CheckBox ist ein Tochterobjekt von **FormField**.

2 Zugriff auf das Objekt

Das **CheckBox**-Objekt lässt sich ausschließlich über sein übergeordnetes **FormField**-Objekt ansprechen.

Nur dann, wenn die Eigenschaft **CheckBox.Valid** den Wert **True** liefert, handelt es sich wirklich um ein Kontrollkästchen – und nicht etwa ein Textfeld oder eine Auswahlliste.

Ein Beispiel:

```
' Den Typ des ersten Formularobjekts prüfen. Wenn es ein
' CheckBox-Objekt ist, dessen Wert (True oder False) ausgegeben.
```

```
If tm.ActiveDocument.FormFields(1).CheckBox.Valid Then
  MsgBox tm.ActiveDocument.FormFields(1).CheckBox.Value
End If
```

3 Eigenschaften, Objekte, Sammlungen und Methoden

Eigenschaften:
- ***Value*** (Defaulteigenschaft)
- **Text**
- **Valid** R/O

Objekte:
- **Application → Application**
- **Parent → FormField**

Value (Eigenschaft)

Datentyp: **Boolean**

Liest oder setzt, ob das Kontrollkästchen angekreuzt ist oder nicht (**True** oder **False**).

Text (Eigenschaft)

Datentyp: **String**

Liest oder setzt den Text des Kontrollkästchens.

Valid (Eigenschaft, R/O)

Datentyp: **Boolean**

Liefert **False**, wenn das Objekt kein **CheckBox**-Objekt ist.

Application (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das **Application**-Objekt.

Parent (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das übergeordnete Objekt, also ein Objekt vom Typ **FormField**.

DropDown (Objekt)

Zugriffspfade:
- Application → Documents → Item → FormFields → Item → **DropDown**
- Application → ActiveDocument → FormFields → Item → **DropDown**

1 Beschreibung

Ein **DropDown**-Objekt repräsentiert ein einzelnes Formularobjekt vom Typ **DropDown** (Auswahlliste) und erlaubt es Ihnen, dessen Wert abzufragen und zu ändern.

DropDown ist ein Tochterobjekt von **FormField**.

2 Zugriff auf das Objekt

Das **DropDown**-Objekt lässt sich ausschließlich über sein übergeordnetes **FormField**-Objekt ansprechen.

Nur dann, wenn die Eigenschaft **DropDown.Valid** den Wert **True** liefert, handelt es sich wirklich um eine Auswahlliste – und nicht etwa ein Textfeld oder ein Kontrollkästchen.

Ein Beispiel:

```
' Den Typ des ersten Formularobjekts prüfen. Wenn es ein
' DropDown-Objekt ist, die Nummer des ausgewählten Eintrags anzeigen

If tm.ActiveDocument.FormFields(1).DropDown.Valid Then
  MsgBox tm.ActiveDocument.FormFields(1).DropDown.Value
End If
```

3 Eigenschaften, Objekte, Sammlungen und Methoden

Eigenschaften:
- ***Value*** (Defaulteigenschaft)
- **Valid** R/O
- **ListEntries**

Objekte:
- **Application** → **Application**
- **Parent** → **FormField**

Value (Eigenschaft)

Datentyp: **Long**

Liest oder setzt die Nummer des gewählten Listeneintrags.

Valid (Eigenschaft, R/O)

Datentyp: **Boolean**

Liefert **False**, wenn das Objekt kein **DropDown**-Objekt ist.

ListEntries (Zeiger auf Sammlung)

Datentyp: **Object**

Liefert die **ListEntries**-Sammlung mit allen Einträgen der Auswahlliste. Über diese Sammlung können Sie die Einträge der Auswahlliste lesen und bearbeiten (bestehende löschen und neue hinzufügen).

Application (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das **Application**-Objekt.

Parent (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das übergeordnete Objekt, also ein Objekt vom Typ **FormField**.

ListEntries (Sammlung)

Zugriffspfade:
- Application → Documents → Item → FormFields → Item → DropDown → **ListEntries**
- Application → ActiveDocument → FormFields → Item → DropDown → **ListEntries**

1 Beschreibung

ListEntries ist eine Sammlung aller Listeneinträge eines **DropDown**-Objekts. Hiermit können Sie die einzelnen Einträge einer Auswahlliste ansehen und bearbeiten.

Die einzelnen Elemente dieser Sammlung sind vom Typ **ListEntry**.

2 Zugriff auf die Sammlung

Jedes **DropDown**-Formularobjekt besitzt genau eine Instanz der **ListEntries**-Sammlung. Diese wird über **DropDown.ListEntries** angesprochen:

```
' Anzahl der Listeneinträge des ersten Formularelements anzeigen
' (sofern es sich tatsächlich um eine Auswahlliste handelt)

If tm.ActiveDocument.FormFields(1).DropDown.Valid Then
  MsgBox tm.ActiveDocument.FormFields(1).DropDown.ListEntries.Count
End If
```

3 Eigenschaften, Objekte, Sammlungen und Methoden

Eigenschaften:
- **Count** R/O

Objekte:
- ***Item*** → **ListEntry** (Defaultobjekt)
- **Application** → **Application**
- **Parent** → **DropDown**

Methoden:
- **Add**
- **Clear**

Count (Eigenschaft, R/O)

Datentyp: **Long**

Liefert die Anzahl der **ListEntry**-Objekte in der Sammlung – in anderen Worten: die Anzahl der Einträge in der Auswahlliste.

Item (Zeiger auf Objekt)

Datentyp: **Object**

Liefert ein einzelnes **ListEntry**-Objekt, also einen einzelnen Listeneintrag in der Auswahlliste.

Welches ListEntry-Objekt Sie erhalten, hängt von dem Parameter ab, den Sie an **Item** übergeben. Dies kann entweder der numerische Index oder der Name des gewünschten Listeneintrags sein. Beispiele:

```
' Den ersten Listeneintrag anzeigen
MsgBox tm.FormFields(1).DropDown.ListEntries.Item(1).Name

' Den Listeneintrag mit dem Text "Test" anzeigen
MsgBox tm.FormFields(1).DropDown.ListEntries.Item("Test").Name
```

Application (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das **Application**-Objekt.

Parent (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das übergeordnete Objekt, also ein Objekt vom Typ **DropDown**.

Add (Methode)

Fügt der Auswahlliste einen weiteren Eintrag hinzu.

Syntax:

```
Add Name
```

Parameter:

Name (Typ: **String**) gibt die hinzuzufügende Zeichenkette an.

Rückgabetyp:

Object (ein **ListEntry**-Objekt, das den neuen Eintrag repräsentiert)

Beispiel:

```
' Dem 1. Formularfeld des Dokuments (einem Dropdown) einen Eintrag hinzufügen
tm.ActiveDocument.FormFields(1).DropDown.ListEntries.Add "Grün"

' Dito, aber mit Nutzung des Rückgabewerts (Klammern beachten!)
Dim entry as Object
Set entry = tm.ActiveDocument.FormFields(1).DropDown.ListEntries.Add("Grün")
```

Clear (Methode)

Entfernt alle Einträge aus der Auswahlliste.

Syntax:

```
Clear
```

Parameter:

keine

Rückgabetyp:

keiner

Beispiel:

```
' Aus dem ersten Formularfeld des Dokuments alle Einträge entfernen
tm.ActiveDocument.FormFields(1).DropDown.ListEntries.Clear
```

ListEntry (Objekt)

Zugriffspfade:
- Application → Documents → Item → FormFields → Item → DropDown → ListEntries → **Item**
- Application → ActiveDocument → FormFields → Item → DropDown → ListEntries → **Item**

1 Beschreibung

Ein **ListEntry**-Objekt repräsentiert einen einzelnen Eintrag in einer Auswahlliste (einem Formularobjekt) und erlaubt es Ihnen, diesen auszulesen, zu verändern und zu löschen.

Für jeden Eintrag in einer Auswahlliste existiert ein eigenes **ListEntry**-Objekt. Fügen Sie der Auswahlliste Einträge hinzu oder löschen diese, werden die zugehörigen **ListEntry**-Objekte dynamisch angelegt bzw. entfernt.

2 Zugriff auf das Objekt

Die einzelnen **ListEntry**-Objekte können ausschließlich durch Aufzählung der Elemente der **ListEntries**-Sammlung angesprochen werden. Von dieser Sammlung besitzt jede Auswahlliste genau eine Instanz.

Ein Beispiel:

```
' Aus dem 1. Formularfeld des Dokuments (einem Dropdown) einen Eintrag zeigen
MsgBox tm.ActiveDocument.FormFields(1).DropDown.ListEntries.Item(1).Name
```

3 Eigenschaften, Objekte, Sammlungen und Methoden

Eigenschaften:
- ***Name*** (Defaulteigenschaft)

Objekte:
- **Application → Application**
- **Parent → ListEntries**

Methoden:
- **Delete**

Name (Eigenschaft)

Datentyp: **String**

Liest oder setzt den Inhalt des **ListEntry**-Objekts – in anderen Worten: den Inhalt des jeweiligen Listeneintrags.

Beispiele:

```
' Den ersten Listeneintrag anzeigen
MsgBox tm.ActiveDocument.FormFields(1).DropDown.ListEntries.Item(1).Name

' Den ersten Listeneintrag auf einen neuen Wert setzen
```

```
tm.ActiveDocument.FormFields(1).DropDown.ListEntries.Item(1).Name = "Grün"
```

Hinweis: Sie können auf diese Weise nur den Text von *bereits existierenden* Listeneinträgen ersetzen. Wollen Sie hingegen der Liste *neue* Einträge hinzufügen, benutzen Sie dafür die Methode **Add** in der **ListEntries**-Sammlung.

Application (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das **Application**-Objekt.

Parent (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das übergeordnete Objekt, also ein Objekt vom Typ **ListEntries**.

Delete (Methode)

Löscht das **ListEntry**-Objekt aus der übergeordneten **ListEntries**-Sammlung.

Syntax:

Delete

Parameter:

keine

Rückgabetyp:

keiner

Beispiel:

```
' Den ersten Listeneintrag löschen
tm.ActiveDocument.FormFields(1).DropDown.ListEntries.Item(1).Delete
```

Windows (Sammlung)

Zugriffspfad: Application → **Windows**

1 Beschreibung

Die Sammlung **Windows** enthält alle geöffneten Dokumentfenster. Die einzelnen Elemente dieser Sammlung sind vom Typ **Window**.

2 Zugriff auf die Sammlung

Es existiert während der gesamten Laufzeit von TextMaker genau eine Instanz der **Windows**-Sammlung. Diese wird über **Application.Windows** angesprochen:

```
' Die Anzahl der offenen Dokumentfenster anzeigen
MsgBox tm.Application.Windows.Count

' Den Namen des ersten geöffneten Dokumentfensters anzeigen
MsgBox tm.Application.Windows(1).Name
```

3 Eigenschaften, Objekte, Sammlungen und Methoden

Eigenschaften:
- **Count** R/O

Objekte:
- ***Item*** → **Window** (Defaultobjekt)
- **Application** → **Application**
- **Parent** → **Application**

Count (Eigenschaft, R/O)

Datentyp: **Long**

Liefert die Anzahl der **Window**-Objekte in TextMaker – in anderen Worten: die Anzahl der geöffneten Dokumentfenster.

Item (Zeiger auf Objekt)

Datentyp: **Object**

Liefert ein einzelnes **Window**-Objekt, also ein einzelnes Dokumentfenster.

Welches Window-Objekt Sie erhalten, hängt von dem Parameter ab, den Sie an **Item** übergeben. Dies kann entweder der numerische Index oder der Dateiname des gewünschten Dokumentfensters sein. Beispiele:

```
' Den Namen des ersten Dokumentfensters anzeigen
MsgBox tm.Application.Windows.Item(1).FullName

' Den Namen des Dokumentfensters "Test.tmd" anzeigen (sofern gerade geöffnet)
MsgBox tm.Application.Windows.Item("Test.tmd").FullName

' Sie können auch den kompletten Namen mit Pfadangabe verwenden
MsgBox tm.Application.Windows.Item("c:\Dokumente\Test.tmd").FullName
```

Application (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das **Application**-Objekt.

Parent (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das übergeordnete Objekt, also **Application**.

Window (Objekt)

Zugriffspfade:
- Application → Windows → **Item**
- Application → **ActiveWindow**
- Application → Documents → Item → **ActiveWindow**
- Application → ActiveDocument → **ActiveWindow**

1 Beschreibung

Ein **Window**-Objekt repräsentiert ein einzelnes in TextMaker geöffnetes Dokumentfenster.

Für jedes Dokumentfenster existiert ein eigenes **Window**-Objekt. Öffnen oder schließen Sie Dokumentfenster, werden die zugehörigen **Window**-Objekte dynamisch angelegt bzw. entfernt.

2 Zugriff auf das Objekt

Die einzelnen **Window**-Objekte können auf folgenden Wegen angesprochen werden:

- Alle zu einem Zeitpunkt geöffneten Dokumentfenster werden in der Sammlung **Application.Windows** (Typ: **Windows**) verwaltet:

```
' Die Namen aller geöffneten Dokumentfenster anzeigen
For i = 1 To tm.Application.Windows.Count
  MsgBox tm.Application.Windows.Item(i).Name
Next i
```

- Das aktive Dokumentfenster erhalten Sie über **Application.ActiveWindow**:

```
' Den Namen des aktuellen Dokumentfensters anzeigen
MsgBox tm.Application.ActiveWindow.Name
```

- **Window** ist der **Parent** des **View**-Objekts:

```
' Den Namen des aktuellen Dokuments über einen Umweg anzeigen
MsgBox tm.Application.ActiveWindow.View.Parent.Name
```

- Das Objekt **Document** enthält einen Objektzeiger auf das ihm zugehörige Dokumentfenster:

```
' Über das aktive Dokument an das aktive Dokumentfenster kommen
MsgBox tm.Application.ActiveDocument.ActiveWindow.Name
```

3 Eigenschaften, Objekte, Sammlungen und Methoden

Eigenschaften:
- **FullName** R/O
- ***Name*** R/O
- **Path** R/O
- **Left**
- **Top**
- **Width**
- **Height**
- **WindowState**
- **DisplayHorizontalRuler**
- **DisplayVerticalRuler**
- **DisplayRulers**
- **DisplayHorizontalScrollBar**
- **DisplayVerticalScrollBar**

Objekte:
- **Document → Document**
- **View → View**
- **Application → Application**
- **Parent → Windows**

Methoden:
- **Activate**
- **Close**

FullName (Eigenschaft, R/O)

Datentyp: **String**

Liefert den Pfad und Namen des in dem Fenster geöffneten Dokuments (z.B. c:\Briefe\Müller.tmd).

Name (Eigenschaft, R/O)

Datentyp: **String**

Liefert den Namen des in dem Fenster geöffneten Dokuments (z.B. Müller.tmd).

Path (Eigenschaft, R/O)

Datentyp: **String**

Liefert den Pfad des in dem Fenster geöffneten Dokuments (z.B. c:\Briefe).

Left (Eigenschaft)

Datentyp: **Long**

Liest oder setzt die X-Koordinate der Fensterposition. Die Maßeinheit sind Bildschirmpixel.

Top (Eigenschaft)

Datentyp: **Long**

Liest oder setzt die Y-Koordinate der Fensterposition. Die Maßeinheit sind Bildschirmpixel.

Width (Eigenschaft)

Datentyp: **Long**

Liest oder setzt die Breite des Dokumentfensters. Die Maßeinheit sind Bildschirmpixel.

Height (Eigenschaft)

Datentyp: **Long**

Liest oder setzt die Höhe des Dokumentfensters. Die Maßeinheit sind Bildschirmpixel.

WindowState (Eigenschaft)

Datentyp: **Long** (SmoWindowState)

Liest oder setzt die Fensterdarstellung des Dokumentfensters. Mögliche Werte:

```
smoWindowStateNormal   = 1 ' normal
smoWindowStateMinimize = 2 ' minimiert
smoWindowStateMaximize = 3 ' maximiert
```

DisplayHorizontalRuler (Eigenschaft)

Datentyp: **Boolean**

Liest oder setzt die Einstellung, ob im Dokumentfenster ein horizontales Lineal angezeigt werden soll (**True** oder **False**).

DisplayVerticalRuler (Eigenschaft)

Datentyp: **Boolean**

Liest oder setzt die Einstellung, ob im Dokumentfenster ein vertikales Lineal angezeigt werden soll (**True** oder **False**).

DisplayRulers (Eigenschaft)

Datentyp: **Boolean**

Liest oder setzt die Einstellung, ob im Dokumentfenster sowohl das horizontale als auch das vertikale Lineal angezeigt werden sollen (**True** oder **False**).

DisplayHorizontalScrollBar (Eigenschaft)

Datentyp: **Boolean**

Liest oder setzt die Einstellung, ob im Dokumentfenster ein horizontaler Rollbalken angezeigt werden soll (**True** oder **False**).

DisplayVerticalScrollBar (Eigenschaft)

Datentyp: **Boolean**

Liest oder setzt die Einstellung, ob im Dokumentfenster ein vertikaler Rollbalken angezeigt werden soll (**True** oder **False**).

Document (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das diesem Dokumentfenster zugeordnete **Document**-Objekt. Mit diesem können Sie zahlreiche Einstellungen Ihres Dokuments lesen und setzen.

View (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das **View**-Objekt des Dokumentfensters. Mit diesem können Sie diverse Einstellungen zur Bildschirmdarstellung lesen und setzen.

Application (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das **Application**-Objekt.

Parent (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das übergeordnete Objekt, also ein Objekt vom Typ **Windows**.

Activate (Methode)

Bringt das Dokumentfenster in den Vordergrund (sofern **Visible** für das Dokument **True** ist) und setzt den Fokus auf das Dokumentfenster.

Syntax:

```
Activate
```

Parameter:

keine

Rückgabetyp:

keiner

Beispiel:

```
' Das erste Dokumentfenster aktivieren
tm.Windows(1).Activate
```

Close (Methode)

Schließt das Dokumentfenster.

Syntax:

```
Close [SaveChanges]
```

Parameter:

SaveChanges (optional; Typ: **Long** bzw. **SmoSaveOptions**) gibt an, ob das im Fenster geöffnete Dokument gespeichert werden soll, sofern es seit dem letzten Speichern verändert wurde. Lassen Sie den Parameter weg, wird stattdessen gegebenenfalls der Benutzer gefragt. Mögliche Werte für **SaveChanges**:

```
smoDoNotSaveChanges = 0       ' Nicht fragen, nicht speichern
smoPromptToSaveChanges = 1    ' Den Benutzer fragen
smoSaveChanges = 2            ' Ohne Rückfrage speichern
```

Rückgabetyp:

keiner

Beispiel:

```
' Das aktuelle Fenster schließen, ohne es zu speichern
tm.ActiveWindow.Close smoDoNotSaveChanges
```

View (Objekt)

Zugriffspfade:
- Application → Windows → Item → **View**
- Application → ActiveWindow → **View**
- Application → Documents → Item → ActiveWindow → **View**
- Application → ActiveDocument → ActiveWindow → **View**

1 Beschreibung

Das **View**-Objekt enthält zahlreiche Einstellungen für die Bildschirmdarstellung. Es ist ein Tochterobjekt von **Window**.

Hinweis: Die im **View**-Objekt verfügbaren Einstellungen sind allesamt *dokumentfensterspezifisch* – jedes Dokumentfenster besitzt also seine eigenen Einstellungen. *Globale* Einstellungen (für alle Dokumente gleichermaßen gültig) finden Sie in den Objekten **Application** und **Options**.

2 Zugriff auf das Objekt

Jedes Dokumentfenster besitzt genau eine Instanz des **View**-Objekts. Diese wird über den Objektzeiger **View** im **Window**-Objekt angesprochen:

```
' Im aktuellen Fenster alle Sonderzeichen (Tabs etc.) anzeigen
tm.ActiveWindow.View.ShowAll = True
```

3 Eigenschaften, Objekte, Sammlungen und Methoden

Eigenschaften:

- **Type**
- **Mode**
- **FieldShading**
- **HighlightComments**
- **RevisionsBalloonSide**
- **RevisionsBalloonWidth**
- **CommentsPaneAutoShow**
- **ShowHiddenText**
- **PrintHiddenText**
- **ShowParagraphs**
- **ShowSpaces**
- **ShowTabs**
- **ShowAll**
- **ShowBookmarks**
- **ShowTextBoundaries**
- **WrapToWindow**

Objekte:

- **Zoom → Zoom**
- **Application → Application**
- **Parent → Window**

Type (Eigenschaft)

Datentyp: **Long** (TmViewType)

Liest oder setzt die Ansichtsart des Dokumentfensters. Mögliche Werte:

```
tmPrintView          = 0 ' Ansicht > Normal
tmMasterView         = 1 ' Ansicht > Masterseiten
tmNormalView         = 2 ' Ansicht > Konzept
tmOutlineView        = 3 ' Ansicht > Gliederung
```

Mode (Eigenschaft)

Datentyp: **Long** (TmViewMode)

Liest oder setzt den Modus des Dokumentfensters. Mögliche Werte:

```
tmViewModeText       = 0 ' Editiermodus
tmViewModeObject     = 1 ' Objektmodus
```

Wenn Sie diese Eigenschaft auf **tmViewModeObject** setzen, während die Ansicht des Dokumentfensters (siehe oben) auf **tmNormalView** (**Ansicht > Konzept**) oder **tmOutlineView** (**Ansicht > Gliederung**) steht, schaltet TextMaker automatisch auf **tmPrintView** um, da der Objektmodus in den genannten Ansichten nicht verfügbar ist.

FieldShading (Eigenschaft)

Datentyp: **Long** (TmFieldShading)

Liest oder setzt die Eigenschaft "Felder schattieren" im Dialogfenster des TextMaker-Befehls **Datei > Eigenschaften > Ansicht**. Mögliche Werte:

```
tmFieldShadingNever   = 0 ' Felder nicht grau hinterlegen
tmFieldShadingAlways  = 1 ' Felder grau hinterlegen
```

HighlightComments (Eigenschaft)

Datentyp: **Boolean**

Liest oder setzt die Eigenschaft des Dokumentfensters, dass Kommentare im Text farblich hervorgehoben werden (**True** oder **False**).

RevisionsBalloonSide (Eigenschaft)

Datentyp: **Long** (TmRevisionsBalloonMargin)

Liest oder setzt die Position, an der Kommentare im Dokumentfenster erscheinen. Mögliche Werte:

```
tmRightMargin    = 0 ' rechts
tmLeftMargin     = 1 ' links
tmOuterMargin    = 2 ' außen
tmInnerMargin    = 3 ' innen
```

RevisionsBalloonWidth (Eigenschaft)

Datentyp: **Long**

Liest oder setzt die Breite des Kommentarfelds im Dokumentfenster. Maßeinheit ist Punkt (1 Punkt entspricht 1/72 Zoll).

CommentsPaneAutoShow (Eigenschaft)

Datentyp: **Boolean**

Liest oder setzt die Einstellung des Dokumentfensters, ob das Kommentarfeld automatisch angezeigt werden soll (**True** oder **False**).

ShowHiddenText (Eigenschaft)

Datentyp: **Boolean**

Liest oder setzt die Einstellung des Dokumentfensters, ob verborgener Text angezeigt wird oder nicht (**True** oder **False**).

PrintHiddenText (Eigenschaft)

Datentyp: **Boolean**

Liest oder setzt die Einstellung des Dokumentfensters, ob verborgener Text ausgedruckt wird oder nicht (**True** oder **False**).

ShowParagraphs (Eigenschaft)

Datentyp: **Boolean**

Liest oder setzt die Einstellung des Dokumentfensters, ob Absatzendemarken (¶) angezeigt werden oder nicht (**True** oder **False**).

ShowSpaces (Eigenschaft)

Datentyp: **Boolean**

Liest oder setzt die Einstellung des Dokumentfensters, ob Leerzeichen durch einen kleinen Punkt (·) repräsentiert werden oder nicht (**True** oder **False**).

ShowTabs (Eigenschaft)

Datentyp: **Boolean**

Liest oder setzt die Einstellung des Dokumentfensters, ob Tabulatoren durch einen Pfeil (→) repräsentiert werden oder nicht (**True** oder **False**).

ShowAll (Eigenschaft)

Datentyp: **Boolean**

Liest oder setzt die Einstellung des Dokumentfensters, ob alle nicht-druckbaren Zeichen (Absatzkenner, Tabstopps, Leerzeichen) angezeigt werden oder nicht (**True** oder **False**).

ShowBookmarks (Eigenschaft)

Datentyp: **Boolean**

Liest oder setzt die Einstellung des Dokumentfensters, ob Textmarken angezeigt werden oder nicht (**True** oder **False**).

ShowTextBoundaries (Eigenschaft)

Datentyp: **Boolean**

Liest oder setzt die Einstellung des Dokumentfensters, ob die Seitenränder durch eine gepunktete Linie symbolisiert werden oder nicht (**True** oder **False**).

WrapToWindow (Eigenschaft)

Datentyp: **Boolean**

Liest oder setzt die Einstellung des Dokumentfensters, ob der Text an den Fensterrändern umbrochen wird oder nicht (**True** oder **False**).

Zoom (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das **Zoom**-Objekt, das Einstellungen zur Vergrößerungsstufe des Dokumentfensters enthält.

Application (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das **Application**-Objekt.

Parent (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das übergeordnete Objekt, also ein Objekt vom Typ **Window**.

Zoom (Objekt)

Zugriffspfade:
- Application → Windows → Item → View → **Zoom**
- Application → ActiveWindow → View → **Zoom**
- Application → Documents → Item → ActiveWindow → View → **Zoom**
- Application → ActiveDocument → ActiveWindow → View → **Zoom**

1 Beschreibung

Das **Zoom**-Objekt enthält die Einstellungen der Vergrößerungsstufe eines Dokumentfensters. Es ist ein Tochterobjekt von **View**.

2 Zugriff auf das Objekt

Jedes Dokumentfenster besitzt genau eine Instanz des **View**-Objekts, und dieses wiederum genau eine Instanz des **Zoom**-Objekts. Letzteres wird über den Objektzeiger **Zoom** im **View**-Objekt angesprochen:

```
' Dokumentfenster auf 140% Vergrößerung setzen
tm.ActiveWindow.View.Zoom.Percentage = 200
```

3 Eigenschaften, Objekte, Sammlungen und Methoden

Eigenschaften:
- **Percentage**

Objekte:
- **Application → Application**
- **Parent → View**

Percentage (Eigenschaft)

Datentyp: **Long**

Liest oder setzt die Vergrößerungsstufe des Dokumentfensters als Prozentwert.

Beispiel:

```
' Dokumentfenster auf 140% Vergrößerung setzen
tm.ActiveWindow.View.Zoom.Percentage = 140
```

Application (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das **Application**-Objekt.

Parent (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das übergeordnete Objekt, also ein Objekt vom Typ **View**.

RecentFiles (Sammlung)

Zugriffspfad: Application → **RecentFiles**

1 Beschreibung

RecentFiles ist eine Sammlung der im Menü **Datei** angezeigten zuletzt geöffneten Dateien. Die einzelnen Elemente dieser Sammlung sind vom Typ **RecentFile**.

2 Zugriff auf die Sammlung

Es existiert während der gesamten Laufzeit von TextMaker genau eine Instanz der **RecentFiles**-Sammlung. Diese wird über **Application.RecentFiles** angesprochen:

```
' Zeige den Namen der ersten Datei im Dateimenü an
MsgBox tm.Application.RecentFiles.Item(1).Name

' Öffne die erste Datei im Dateimenü
tm.Application.RecentFiles.Item(1).Open
```

3 Eigenschaften, Objekte, Sammlungen und Methoden

Eigenschaften:
- **Count** R/O
- **Maximum**

Objekte:
- ***Item*** → **RecentFile** (Defaultobjekt)
- **Application** → **Application**
- **Parent** → **Application**

Methoden:
- **Add**

Count (Eigenschaft, R/O)

Datentyp: **Long**

Liefert die Anzahl der **RecentFile**-Objekte in TextMaker – in anderen Worten: die Anzahl der im Dateimenü verzeichneten zuletzt geöffneten Dateien.

Maximum (Eigenschaft, R/O)

Datentyp: **Long**

Liest oder setzt die Einstellung "Einträge im Datei-Menü" – in anderen Worten: wie viele zuletzt geöffnete Dateien das Dateimenü maximal aufnehmen kann.

Der Wert darf zwischen 0 und 9 liegen.

Item (Zeiger auf Objekt)

Datentyp: **Object**

Liefert ein einzelnes **RecentFile**-Objekt, also einen einzelnen Dateieintrag im Dateimenü.

Welches RecentFile-Objekt Sie erhalten, hängt von dem Zahlenwert ab, den Sie an **Item** übergeben: 1 für die erste der zuletzt geöffneten Dateien, 2 für die zweite etc.

Application (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das **Application**-Objekt.

Parent (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das übergeordnete Objekt, also **Application**.

Add (Methode)

Fügt der Liste der zuletzt geöffneten Dateien ein Dokument hinzu.

Syntax:

```
Add Document, [FileFormat]
```

Parameter:

Document ist eine Zeichenkette mit dem Pfad und Dateinamen des hinzuzufügenden Dokuments.

FileFormat (optional; Typ: **Long** bzw. **TmSaveFormat**) gibt das Dateiformat des hinzuzufügenden Dokuments an. Mögliche Werte:

```
tmFormatDocument         =  0 ' Dokument, ist der Standardwert
tmFormatTemplate         =  1 ' Dokumentvorlage
tmFormatWinWord97        =  2 ' Microsoft Word für Windows 97 und 2000
tmFormatOpenDocument     =  3 ' OpenDocument, OpenOffice.org, StarOffice
tmFormatRTF              =  4 ' Rich Text Format
tmFormatPocketWordPPC    =  5 ' Pocket Word auf dem Pocket PC
tmFormatPocketWordHPC    =  6 ' Pocket Word auf dem Handheld PC (Windows CE)
tmFormatPlainTextAnsi    =  7 ' Textdatei mit Windows-Zeichensatz
tmFormatPlainTextDOS     =  8 ' Textdatei mit DOS-Zeichensatz
tmFormatPlainTextUnicode =  9 ' Textdatei mit Unicode-Zeichensatz
tmFormatPlainTextUTF8    = 10 ' Textdatei mit UTF8-Zeichensatz
tmFormatHTML             = 12 ' HTML
tmFormatWinWord6         = 13 ' Microsoft Word für Windows 6.0
tmFormatPlainTextUnix    = 14 ' Textdatei für UNIX, Linux, FreeBSD
tmFormatWinWordXP        = 15 ' Microsoft Word für Windows XP und 2003
```

Wenn Sie diesen Parameter weglassen, wird **tmFormatDocument** angenommen.

Unabhängig vom übergebenen Parameter **FileFormat** versucht TextMaker stets, das Dateiformat selbst zu erkennen, und ignoriert offensichtlich falsche Angaben.

Rückgabetyp:

Object (ein **RecentFile**-Objekt, das das hinzugefügte Dokument repräsentiert)

Beispiel:

```
' Die Datei Test.rtf dem Dateimenü hinzufügen
tm.Application.RecentFiles.Add "Test.rtf", tmFormatRTF

' Dito, aber mit Auswertung des Rückgabewerts (Klammern beachten!)
Dim fileObj as Object
Set fileObj = tm.Application.RecentFiles.Add("Test.rtf", tmFormatRTF)
MsgBox fileObj.Name
```

RecentFile (Objekt)

Zugriffspfad: Application → RecentFiles → **Item**

1 Beschreibung

Ein **RecentFile**-Objekt repräsentiert eine einzelne der zuletzt geöffneten Dateien und lässt Sie deren Eigenschaften abfragen und sie erneut öffnen

Für jede der zuletzt geöffneten Datei existiert ein eigenes **RecentFile**-Objekt. Durch das Öffnen und Schließen von Dokumenten ändert sich die Liste dieser Dateien im Dateimenü; die zugehörigen **RecentFile**-Objekte werden von TextMaker dynamisch angelegt bzw. entfernt.

2 Zugriff auf das Objekt

Die einzelnen **RecentFile**-Objekte können ausschließlich durch Aufzählung der Elemente der **RecentFiles**-Sammlung angesprochen werden. Diese erreichen Sie über **Application.RecentFiles**:

```
' Den Namen der ersten Datei im Dateimenü anzeigen
MsgBox tm.Application.RecentFiles.Item(1).Name
```

3 Eigenschaften, Objekte, Sammlungen und Methoden

Eigenschaften:
- **FullName** R/O
- ***Name*** R/O (Defaulteigenschaft)
- **Path** R/O

Objekte:
- **Application → Application**
- **Parent → RecentFiles**

Methoden:
- **Open**

FullName (Eigenschaft, R/O)

Datentyp: **String**

Liefert den Pfad und Namen des Dokuments im Dateimenü (z.B. c:\Briefe\Müller.tmd).

Name (Eigenschaft, R/O)

Datentyp: **String**

Liefert den Namen des Dokuments (z.B. Müller.tmd).

Path (Eigenschaft, R/O)

Datentyp: **String**

Liefert den Pfad des Dokuments (z.B. c:\Briefe).

Application (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das **Application**-Objekt.

Parent (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das übergeordnete Objekt, also **Application**.

Open (Methode)

Öffnet das betreffende Dokument und liefert es als **Document**-Objekt zurück.

Syntax:

Open

Parameter:

keine

Rückgabetyp:

Document

Beispiel:

```
' Das erste Dokument aus dem Dateimenü öffnen
tm.Application.RecentFiles(1).Open
```

FontNames (Sammlung)

Zugriffspfad: Application → **FontNames**

1 Beschreibung

FontNames ist eine Sammlung aller in Windows installierten Schriftarten. Die einzelnen Elemente dieser Sammlung sind vom Typ **FontName**.

2 Zugriff auf die Sammlung

Es existiert während der gesamten Laufzeit von TextMaker genau eine Instanz der **FontNames**-Sammlung. Diese wird über **Application.FontNames** angesprochen:

```
' Den Namen der ersten installierten Schrift anzeigen
MsgBox tm.Application.FontNames.Item(1).Name

' Dasselbe, nur kürzer durch die Nutzung von Defaulteigenschaften:
MsgBox tm.FontNames(1)
```

3 Eigenschaften, Objekte, Sammlungen und Methoden

Eigenschaften:
- **Count** R/O

Objekte:
- ***Item*** → **FontName** (Defaultobjekt)
- **Application** → **Application**
- **Parent** → **Application**

Count (Eigenschaft, R/O)

Datentyp: **Long**

Liefert die Anzahl der **FontName**-Objekte in TextMaker – in anderen Worten: die Anzahl der im System installierten Schriften.

Item (Zeiger auf Objekt)

Datentyp: **Object**

Liefert ein einzelnes **FontName**-Objekt, also eine einzelne installierte Schriftart.

Welches FontName-Objekt Sie erhalten, hängt von dem Zahlenwert ab, den Sie an **Item** übergeben: 1 für die erste installierte Schrift, 2 für die zweite etc.

Application (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das **Application**-Objekt.

Parent (Zeiger auf Objekt)

Liefert das übergeordnete Objekt, also **Application**.

FontName (Objekt)

Zugriffspfad: Application → FontNames → **Item**

1 Beschreibung

Ein **FontName**-Objekt repräsentiert eine einzelne der in Windows installierten Schriftarten. Für jede installierte Schriftart existiert ein eigenes **FontName**-Objekt.

2 Zugriff auf das Objekt

Die einzelnen **FontName**-Objekte können ausschließlich durch Aufzählung der Elemente der **FontNames**-Sammlung angesprochen werden. Diese erreichen Sie über **Application.FontNames**:

```
' Den Namen der ersten installierten Schrift anzeigen
MsgBox tm.Application.FontNames.Item(1).Name

' Dasselbe, nur kürzer durch die Nutzung von Defaulteigenschaften:
MsgBox tm.FontNames(1)
```

3 Eigenschaften, Objekte, Sammlungen und Methoden

Eigenschaften:
- ***Name*** R/O (Defaulteigenschaft)
- **Charset**

Objekte:
- **Application → Application**
- **Parent → FontNames**

Name (Eigenschaft, R/O)

Datentyp: **String**

Liefert den Namen der betreffenden Schrift.

Charset (Eigenschaft, R/O)

Datentyp: **Long** (SmoCharset)

Liefert den Zeichensatz der betreffenden Schrift. Mögliche Werte:

```
smoAnsiCharset   = 0 ' normaler Zeichensatz
smoSymbolCharset = 2 ' Symbolschrift
```

Application (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das **Application**-Objekt.

Parent (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das übergeordnete Objekt, also **FontNames**.

BasicMaker und PlanMaker

Ganz ähnlich wie TextMaker lässt sich auch die Tabellenkalkulation PlanMaker mit BasicMaker programmieren. Dieses Kapitel enthält alle Informationen zur PlanMaker-Programmierung. Es ist in folgende Abschnitte gegliedert:

- **Programmierung von PlanMaker**

  Dieser Abschnitt enthält grundlegende Informationen zur Programmierung von PlanMaker mit BasicMaker.

- **Objektstruktur von PlanMaker**

  In diesem Abschnitt werden alle von PlanMaker zur Verfügung gestellten Objekte aufgelistet.

Programmierung von PlanMaker

Die Programmierung der Textverarbeitung *TextMaker* und die der Tabellenkalkulation *PlanMaker* unterscheiden sich nur darin, dass einige Schlüsselwörter andere Namen haben (zum Beispiel *Plan*Maker.Application statt *Text*Maker.Application). Wenn Sie also den Abschnitt "Programmierung von TextMaker" bereits kennen, werden Sie feststellen, dass dieser hier nahezu identisch ist.

PlanMaker stellt aber natürlich andere Objekte zur Verfügung als TextMaker. Eine Auflistung der exponierten Objekte finden Sie im nächsten Abschnitt "Objektstruktur von PlanMaker".

Um PlanMaker mit BasicMaker zu programmieren, verwenden Sie in erster Linie *OLE Automation-Befehle*. Allgemeine Informationen zu diesem Thema erhalten Sie im Abschnitt "OLE Automation".

Prinzipiell ist folgendermaßen vorzugehen (Details folgen im Anschluss):

1. Deklarieren Sie eine Variable vom Typ **Object**:

   ```
   Dim pm as Object
   ```

2. Stellen Sie über OLE Automation eine Verbindung zu PlanMaker her (PlanMaker wird dazu nötigenfalls automatisch gestartet):

   ```
   Set pm = CreateObject("PlanMaker.Application")
   ```

3. Setzen Sie die Eigenschaft **Application.Visible** auf **True**, damit PlanMaker sichtbar wird:

   ```
   pm.Application.Visible = True
   ```

4. Jetzt können Sie PlanMaker programmieren, indem Sie "Eigenschaften" von PlanMaker auslesen und abändern und die von PlanMaker bereitgestellten "Methoden" anwenden.

5. Wird das PlanMaker-Objekt nicht mehr benötigt, sollten Sie die Verbindung zu PlanMaker trennen:

   ```
   Set pm = Nothing
   ```

Soweit in aller Kürze. Auf den nächsten Seiten folgen ausführlichere Informationen zur Programmierung von PlanMaker. Eine Aufstellung aller PlanMaker-Objekte und der darauf anwendbaren Eigenschaften und Methoden finden Sie anschließend im Abschnitt "Objektstruktur von PlanMaker".

Verbindung zu PlanMaker herstellen

Wenn Sie PlanMaker mit BasicMaker steuern wollen, müssen Sie zuerst über eine OLE Automation eine Verbindung zu PlanMaker herstellen. Dazu ist eine Variable vom Typ **Object** zu deklarieren, der anschließend mit dem Befehl **CreateObject** das Objekt "PlanMaker.Application" zugewiesen wird:

```
Dim pm as Object
Set pm = CreateObject("PlanMaker.Application")
```

Wenn PlanMaker bereits läuft, wird dadurch lediglich eine Verbindung zu ihm aufgebaut; wenn PlanMaker noch nicht gestartet ist, wird er automatisch gestartet.

Die Objektvariable "pm" enthält nun eine Referenz auf PlanMaker.

Wichtig: PlanMaker sichtbar machen

Bitte beachten Sie: Wenn Sie PlanMaker wie gerade beschrieben starten, ist das Programmfenster standardmäßig *unsichtbar*. Soll PlanMaker sichtbar gemacht werden, muss die Eigenschaft **Visible** auf **True** gesetzt werden. Der vollständige Aufruf von PlanMaker sollte also lauten:

```
Dim pm as Object
Set pm = CreateObject("PlanMaker.Application")
pm.Application.Visible = True
```

Das Objekt "Application"

Das *grundlegende* Objekt, das PlanMaker für die Programmierung exponiert, ist **Application**. Alle anderen Objekte – wie zum Beispiel die derzeit geöffneten Arbeitsmappen oder sämtliche Programmoptionen – "hängen" am **Application**-Objekt.

Das **Application**-Objekt enthält einerseits eigene Eigenschaften (zum Beispiel **Application.Left** für die X-Koordinate des Programmfensters) und Methoden (wie **Application.Quit** zum Beenden von PlanMaker), andererseits enthält es Zeiger auf andere Objekte wie **Application.Options**, die wiederum eigene Eigenschaften und Methoden enthalten, und Zeiger auf Sammlungen ("Collections") wie **Workbooks** (die Liste der gerade geöffneten Dokumente).

Schreibweisen

Wie Sie aus dem vorherigen Abschnitt schon ersehen können, ist für den Zugriff auf die bereitgestellten Eigenschaften, Methoden usw. die bei OLE-Automation übliche Punktnotation zu verwenden.

Mit **Application.Left** wird beispielsweise die Eigenschaft **Left** des Objekts **Application** angesprochen. **Application.Workbooks.Add** bezeichnet die Methode **Add** des Objekts **Workbooks**, das wiederum ein Objekt von **Application** ist.

Eigenschaften (Properties) von PlanMaker auslesen und ändern

Wurde die Verbindung zu PlanMaker hergestellt, können Sie das Programm "fernsteuern". Dazu gibt es, wie im Abschnitt "OLE Automation" beschrieben, *Eigenschaften (Properties)* und *Methoden (Methods)*.

Beschäftigen wir uns zunächst mit den *Eigenschaften*. Als Eigenschaften bezeichnet man Optionen und Einstellungen, die abgefragt und teilweise verändert werden können.

Möchten Sie zum Beispiel den Programmnamen von PlanMaker ermitteln, verwenden Sie die Eigenschaft **Name** des Objekts **Application**:

```
MsgBox "Der Name der Applikation ist: " & pm.Application.Name
```

Bei **Application.Name** handelt es sich um eine Eigenschaft, die nur gelesen werden kann. Andere Eigenschaften lassen sich sowohl auslesen als auch von einem BasicMaker-Script aus abändern. So sind die Koordinaten des PlanMaker-Programmfensters in den Eigenschaften **Left, Top**, **Width** und **Height** des Application-Objekts abgelegt. Sie können sie wieder auslesen:

```
MsgBox "Der linke Fensterrand liegt bei: " & pm.Application.Left
```

Sie können diese Eigenschaft aber auch verändern:

```
pm.Application.Left = 200
```

PlanMaker reagiert sofort und verschiebt den linken Fensterrand auf dem Bildschirm an die Pixelposition 200. Sie können Lesen und Schreiben von Eigenschaften auch mischen, etwa:

```
pm.Application.Left = pm.Application.Left + 100
```

Hier wird der aktuelle linke Rand ausgelesen, um 100 erhöht und als neuer linker Rand an PlanMaker übergeben. Auch hier reagiert PlanMaker sofort und schiebt seinen linken Fensterrand um 100 Pixel nach rechts.

Es gibt eine große Anzahl von Eigenschaften des **Application**-Objekts. Eine Auflistung finden Sie im Abschnitt "Objektstruktur von PlanMaker".

Methoden (Methods) von PlanMaker verwenden

Neben Eigenschaften gibt es *Methoden*. Methoden sind Befehle, die PlanMaker anweisen, etwas Bestimmtes zu tun.

So können Sie zum Beispiel mit **Application.Quit** PlanMaker anweisen, sich zu beenden; mit **Application.Activate** erzwingen Sie, dass das PlanMaker-Programmfenster in den Vordergrund kommt, wenn es gegenwärtig von Fenstern anderer Programme überdeckt wird:

```
pm.Application.Activate
```

Unterschied zwischen Funktions- und Prozedurmethoden

Es gibt zwei Arten von Methoden: solche, die einen Wert an das Basic-Programm zurückliefern und solche ohne Rückgabewert. Erstere bezeichnen wir – in Anlehnung an andere Programmiersprachen – als "Funktionsmethoden" oder einfach "Funktionen", letztere als "Prozedurmethoden" oder "Prozeduren".

Diese Unterscheidung mag Ihnen vielleicht übertrieben feinsinnig erscheinen, sie ist es aber nicht, weil sie Auswirkungen auf die Schreibweise der Befehle hat.

Solange Sie eine Methode ohne Parameter aufrufen, gibt es keinen syntaktischen Unterschied:

Aufruf als Prozedur:

```
pm.Workbooks.Add ' Ein Dokument zu den offenen Dokumenten hinzufügen
```

Aufruf als Funktion:

```
Dim newDoc as Object
Set newDoc = pm.Workbooks.Add ' jetzt mit dem Workbook-Objekt als Rückgabewert
```

Bei Methoden *mit* Parametern sind aber unterschiedliche Schreibweisen erforderlich:

Aufruf als Prozedur:

```
pm.Application.RecentFiles.Add "Test.pmd"
```

Aufruf als Funktion:

```
Dim x as Object
Set x = pm.Application.RecentFiles.Add("Test.pmd") ' jetzt mit Rückgabewert
```

Sie sehen: Beim Aufruf als Prozedur dürfen Sie die Parameter *nicht* mit Klammern umgeben, beim Aufruf als Funktion *müssen* Sie es.

Zeiger auf andere Objekte verwenden

Eine dritte Gruppe von Elementen des **Application**-Objekts sind *Zeiger auf andere Objekte*.

Stellen Sie sich hier bitte nichts großartig Kompliziertes vor. Es ist lediglich unübersichtlich, alle Eigenschaften und Methoden von PlanMaker unmittelbar an das Application-Objekt zu hängen, da die Objektstruktur dadurch sehr unübersichtlich würde. Deshalb sind bestimmte Reihen von Eigenschaften und Methoden zu logischen Gruppen zusammengefasst. So kennt PlanMaker beispielsweise das Objekt **Options**, mit dem Sie viele grundlegende Programmeinstellungen (solche, die Sie meist in **Weiteres > Einstellungen** finden) auslesen und verändern können:

```
pm.Application.Options.CreateBackup = True
```

```
MsgBox "Überschreibe- oder Einfügemodus eingeschaltet? " &
pm.Application.Options.Overtype
```

Sammlungen verwenden

Die vierte Gruppe von Elementen des **Application**-Objekts sind Zeiger auf *Sammlungen* ("Collections").

Sammlungen enthalten, wie der Name schon sagt, eine Ansammlung von gleichartigen Objekten. Es gibt zum Beispiel eine Sammlung **Application.Workbooks**, die alle geöffneten Dokumente enthält und eine Sammlung **Application.RecentFiles** mit allen Dateien, die im Datei-Menü aufgelistet werden.

Es existieren zwei standardisierte Arten, um auf Sammlungen zuzugreifen, und PlanMaker unterstützt beide. Die simplere Art ist die Eigenschaft **Item**, die in jeder Sammlung vorhanden ist:

```
' Namen des ersten geöffneten Dokuments ausgeben:
MsgBox pm.Application.Workbooks.Item(1).Name

' Schließt das (geöffnete) Dokument "Test.pmd":
pm.Application.Workbooks.Item("Test.pmd").Close
```

Wollen Sie beispielsweise alle geöffneten Dokumente auflisten, lassen Sie sich zuerst mit der standardisierten Eigenschaft **Count** die Zahl der offenen Dokumente geben und greifen dann sukzessive auf die einzelnen Elemente, also Dokumente, zu:

```
'Gibt die Namen aller geöffneten Dokumente aus:
For i=1 To pm.Application.Workbooks.Count
   MsgBox pm.Application.Workbooks.Item(i).Name
Next i
```

Jede Sammlung besitzt also per Definition die Eigenschaft **Count**, welche die Zahl der Einträge in der Sammlung ermittelt, und die Eigenschaft **Item**, mit der Sie gezielt an einen Eintrag in der Sammlung herankommen.

Item akzeptiert als Argument stets die Nummer des gewünschten Eintrags. Soweit es sinnvoll ist, akzeptiert **Item** als Argument auch andere Argumente, zum Beispiel Dateinamen. Sie haben dies bereits weiter oben gesehen, als wir **Item** einmal eine Zahl übergeben haben und einmal einen Dateinamen.

Zu den meisten Sammlungen gibt es einen passenden Objekttyp für deren einzelne Elemente. Bei der Sammlung **Windows** ist ein beispielsweise ein einzelner Eintrag, der von **Item** zurückgeliefert wird, vom Typ **Window** – man beachte den Singular! Ein Element der **Workbooks**-Sammlung heißt **Workbook**, ein Element der **RecentFiles**-Sammlung eben **RecentFile**.

Eleganter Zugriff auf Sammlungen: For Each ... Next

Eine elegantere Methode, hintereinander auf alle Einträge einer Sammlung zuzugreifen, sei hier ebenfalls beschrieben: BasicMaker unterstützt auch die **For Each**-Anweisung:

```
' Namen aller geöffneten Dokumente ausgeben

Dim x As Object

For Each x In pm.Application.Workbooks
   MsgBox x.Name
Next x
```

Das ist gleichbedeutend mit der oben vorgestellten Schreibweise:

```
For i=1 To pm.Application.Workbooks.Count
   MsgBox pm.Application.Workbooks.Item(i).Name
Next i
```

Eigene Eigenschaften und Methoden von Sammlungen

Sammlungen besitzen neben **Item** und **Count** gegebenenfalls eigene Eigenschaften und Methoden, mit denen die jeweilige Sammlung verwaltet werden kann. Möchten Sie beispielsweise in PlanMaker ein leeres Dokument anlegen, so bedeutet dieser Vorgang für BasicMaker, dass Sie der **Workbooks**-Sammlung einen neuen Eintrag hinzufügen:

```
pm.Application.Workbooks.Add   ' leeres Dokument anlegen
```

Tipps für die Vereinfachung von Schreibweisen

Wenn Sie sich nun langsam wundern, ob wirklich so viel Tipparbeit nötig ist, um ein einzelnes Dokument anzusprechen, können wir Sie beruhigen: ist es nicht! Es gibt diverse Abkürzungen, die Ihnen viel Zeit ersparen.

Verwendung der With-Anweisung

Die erste Abkürzung ist, dass Sie zum Zugriff auf *mehrere* Eigenschaften eines Objekts die **With**-Anweisung verwenden können.

Zunächst die herkömmliche Schreibweise:

```
pm.Application.Left = 100
pm.Application.Top = 50
pm.Application.Width = 500
pm.Application.Height = 300
msgbox pm.Application.Options.CreateBackup
```

Dieser Code sieht bei Verwendung der **With**-Anweisung wesentlich übersichtlicher aus:

```
With pm.Application
   .Left = 100
   .Top = 50
   .Width = 500
   .Height = 300
   msgbox .Options.CreateBackup
End With
```

Standardeigenschaften müssen nicht ausgeschrieben werden

Es geht in vielen Fällen noch einfacher: Jedes Objekt (zum Beispiel **Application** oder **Application.Workbooks**) besitzt unter seinen Eigenschaften jeweils eine Eigenschaft, die als *Standardeigenschaft* markiert ist. Das Praktische daran ist, dass Sie sich dadurch nochmals Tipparbeit ersparen können, denn die Standardeigenschaft kann einfach weglassen werden.

Die Standardeigenschaft von **Application** ist beispielsweise **Name**. Folgende beiden Befehle sind daher gleichbedeutend:

```
MsgBox pm.Application.Name   ' gibt den Namen von PlanMaker aus

MsgBox pm.Application        ' tut dasselbe
```

Typischerweise ist die am häufigsten benötigte Eigenschaft eines Objekts als Standardeigenschaft markiert. So ist sicherlich die am häufigsten benötigte Eigenschaft einer Sammlung die **Item**-Eigenschaft. Denn im Allgemeinen will man ja auf ein bestimmtes oder mehrere bestimmte Elemente einer Sammlung zugreifen. Folgende Anweisungen sind daher wieder gleichbedeutend:

```
MsgBox pm.Application.Workbooks.Item(1).Name

MsgBox pm.Application.Workbooks(1).Name
```

So wird das Ganze doch langsam übersichtlicher! Es kommt aber noch besser: **Name** ist die Standardeigenschaft eines einzelnen **Workbook**-Objekts (aufgepasst: "Workbook", nicht "Workbooks"!). Jedes **Item** der **Workbooks**-Sammlung ist vom Typ **Workbook**. Da also **Name** die Standardeigenschaft ist, können Sie **Name** wieder weglassen:

```
MsgBox pm.Application.Workbooks(1)
```

Immer noch nicht einfach genug? Also... **Application** ist die Standardeigenschaft von PlanMaker an sich. Lassen wir **Application** also einfach weg! Das sieht dann so aus:

```
MsgBox pm.Workbooks(1)
```

Mit diesem Grundwissen sind Sie nun gerüstet, um die Objektstruktur von PlanMaker zu verstehen und können sich dem nächsten Abschnitt widmen, der eine detaillierte Liste aller von PlanMaker bereitgestellten Objekte enthält.

Objektstruktur von PlanMaker

PlanMaker stellt BasicMaker und anderen OLE Automation-fähigen Programmiersprachen die im Folgenden aufgelisteten Objekte zur Verfügung.

Hinweise:

- Mit "R/O" gekennzeichnete Eigenschaften sind "Read Only" (also schreibgeschützt). Sie können zwar ausgelesen, aber nicht verändert werden.
- Die Default-Eigenschaft eines Objekts ist durch *Kursivschrift* gekennzeichnet.

Die folgende Tabelle führt alle in PlanMaker verfügbaren Objekte und Sammlungen auf.

Name	Typ	Beschreibung
Application	Objekt	"Wurzelobjekt" von PlanMaker
Options	Objekt	Globale Einstellungen
UserProperties	Sammlung	Sammlung aller Bestandteile der privaten und geschäftlichen Adresse
UserProperty	Objekt	Ein einzelner Bestandteil der Adresse
CommandBars	Sammlung	Sammlung aller Symbolleisten
CommandBar	Objekt	Eine einzelne Symbolleiste
AutoCorrect	Objekt	Automatische Textkorrektur und Textbausteine
AutoCorrectEntries	Sammlung	Sammlung aller Textbausteine
AutoCorrectEntry	Objekt	Ein einzelner Textbaustein
Workbooks	Sammlung	Sammlung aller geöffneten Dokumente (Arbeitsmappen)
Workbook	Objekt	Ein einzelnes geöffnetes Dokument
DocumentProperties	Sammlung	Sammlung aller Dokumenteigenschaften eines Dokuments
DocumentProperty	Objekt	Eine einzelne Dokumenteigenschaft
Sheets	Sammlung	Sammlung aller Arbeitsblätter eines Dokuments
Sheet	Objekt	Ein einzelnes Arbeitsblatt eines Dokuments
PageSetup	Objekt	Die Seiteneinstellungen eines Arbeitsblatts
Range	Objekt	Ein beliebiger Bereich von Zellen eines Arbeitsblatts
Rows	Sammlung	Sammlung aller Zeilen eines Arbeitsblatts oder Bereichs
Columns	Sammlung	Sammlung aller Spalten eines Arbeitsblatts oder Bereichs
FormatConditions	Sammlung	Sammlung aller bedingten Formatierungen eines Bereichs
FormatCondition	Objekt	Eine einzelne bedingte Formatierung eines Bereichs
NumberFormatting	Objekt	Die Zahlenformatierung eines Bereichs
Font	Objekt	Die Zeichenformatierung eines Bereichs oder einer bedingten Formatierung
Borders	Sammlung	Sammlung aller Umrandungslinien eines Bereichs oder einer bedingten Formatierung
Border	Objekt	Eine einzelne Umrandungslinie
Shading	Objekt	Die Schattierung eines Bereichs oder einer bedingten Formatierung
Validation	Objekt	Die Einstellungen zur Gültigkeitsprüfung eines Bereichs
AutoFilter	Objekt	Der AutoFilter eines Arbeitsblatts
Filters	Sammlung	Sammlung aller Spalten des AutoFilters
Filter	Objekt	Eine einzelne Spalte des AutoFilters

Name	Typ	Beschreibung
Windows	Sammlung	Sammlung aller geöffneten Dokumentfenster
Window	Objekt	Ein einzelnes Dokumentfenster
RecentFiles	Sammlung	Sammlung aller im Dateimenü aufgeführten zuletzt geöffneten Dateien
RecentFile	Objekt	Eine einzelne der zuletzt geöffneten Dateien
FontNames	Sammlung	Sammlung aller in Windows installierten Schriftarten
FontName	Objekt	Eine einzelne installierte Schriftart

Im Anschluss werden alle Objekte und Sammlungen im Detail beschrieben.

Application (Objekt)

Zugriffspfad: **Application**

1 Beschreibung

Application ist das "Wurzelobjekt" aller anderen Objekte in PlanMaker. Es ist das zentrale Steuerobjekt, über das die gesamte Kommunikation zwischen Ihrem Basic-Script und PlanMaker abgewickelt wird.

2 Zugriff auf das Objekt

Es existiert genau eine Instanz des **Application**-Objekts. Diese ist während der gesamten Laufzeit von PlanMaker verfügbar und wird direkt über die von **CreateObject** zurückgegebene Objektvariable angesprochen:

```
Set pm = CreateObject("PlanMaker.Application")
MsgBox pm.Application.Name
```

Da **Application** die Defaulteigenschaft von PlanMaker ist, kann es generell weggelassen werden:

```
Set pm = CreateObject("PlanMaker.Application")
MsgBox pm.Name ' gleichbedeutend mit pm.Application.Name
```

3 Eigenschaften, Objekte, Sammlungen und Methoden

Eigenschaften:

- **FullName** R/O
- ***Name*** R/O (Defaulteigenschaft)
- **Path** R/O
- **Build** R/O
- **Bits** R/O
- **Visible**
- **Caption** R/O
- **Left**
- **Top**
- **Width**
- **Height**
- **WindowState**
- **Calculation**
- **CalculateBeforeSave**
- **DisplayCommentIndicator**
- **EditDirectlyInCell**
- **MoveAfterReturn**
- **MoveAfterReturnDirection**

- **PromptForSummaryInfo**
- **WarningOnError**

Objekte:
- **Options → Options**
- **UserProperties → UserProperties**
- **CommandBars → CommandBars**
- **AutoCorrect → AutoCorrect**
- **ActiveWorkbook → Workbook**
- **ActiveSheet → Sheet**
- **ActiveWindow → Window**
- **ActiveCell → Range**
- **Selection → Range**
- **Range → Range**
- **Cells → Range**
- **Application → Application**

Sammlungen:
- **Workbooks → Workbooks**
- **Windows → Windows**
- **RecentFiles → RecentFiles**
- **FontNames → FontNames**
- **Columns → Columns**
- **Rows → Rows**

Methoden:
- **CentimetersToPoints**
- **MillimetersToPoints**
- **InchesToPoints**
- **PicasToPoints**
- **LinesToPoints**
- **Activate**
- **Calculate**
- **Quit**

FullName (Eigenschaft, R/O)

Datentyp: **String**

Liefert Namen und Pfad des Programms (z.B. "c:\Programme\SoftMaker Office\PlanMaker.exe").

Name (Eigenschaft, R/O)

Datentyp: **String**

Liefert den Namen des Programms (z.B. "PlanMaker")

Path (Eigenschaft, R/O)

Datentyp: **String**

Liefert den Pfad des Programms, zum Beispiel "c:\Programme\SoftMaker Office\".

Build (Eigenschaft, R/O)

Datentyp: **String**

Liefert die Revisionsnummer des Programms als Zeichenkette, zum Beispiel "320".

Bits (Eigenschaft, R/O)

Datentyp: **String**

Liefert eine Zeichenkette, die der Bit-Version des Programms entspricht: "16" bei der 16 Bit-Version, "32" bei der 32 Bit-Version von PlanMaker.

Visible (Eigenschaft)

Datentyp: **Boolean**

Liest oder setzt die Sichtbarkeit des Programmfensters:

```
pm.Application.Visible = True  ' PlanMaker wird sichtbar
pm.Application.Visible = False ' PlanMaker wird unsichtbar
```

Wichtig: Standardmäßig ist **Visible** auf **False** gesetzt – PlanMaker startet also unsichtbar, bis Sie ihn explizit sichtbar machen.

Caption (Eigenschaft, R/O)

Datentyp: **String**

Liefert eine Zeichenkette mit dem Inhalt der Titelleiste des Programmfensters (z.B. "PlanMaker - Meine Tabelle.pmd").

Left (Eigenschaft)

Datentyp: **Long**

Liest oder setzt die X-Koordinate (= linker Rand) des Programmfensters auf dem Bildschirm. Die Maßeinheit sind Bildschirmpixel.

Top (Eigenschaft)

Datentyp: **Long**

Liest oder setzt die Y-Koordinate (= oberer Rand) des Programmfensters auf dem Bildschirm. Die Maßeinheit sind Bildschirmpixel.

Width (Eigenschaft)

Datentyp: **Long**

Liest oder setzt die Breite des Programmfensters auf dem Bildschirm. Die Maßeinheit sind Bildschirmpixel.

Height (Eigenschaft)

Datentyp: **Long**

Liest oder setzt die Höhe des Programmfensters auf dem Bildschirm. Die Maßeinheit sind Bildschirmpixel.

WindowState (Eigenschaft)

Datentyp: **Long** (SmoWindowState)

Liest oder setzt die Fensterdarstellung des Programmfensters. Mögliche Werte:

```
smoWindowStateNormal   = 1 ' normal
smoWindowStateMinimize = 2 ' minimiert
smoWindowStateMaximize = 3 ' maximiert
```

Calculation (Eigenschaft)

Datentyp: **Long** (PmCalculation)

Liest oder setzt die Einstellung, ob Dokumente automatisch oder manuell neu berechnet werden. Mögliche Werte:

```
pmCalculationAutomatic   = 0 ' Berechnungen automatisch aktualisieren
pmCalculationManual      = 1 ' Berechnungen manuell aktualisieren
```

Hinweise:

- PlanMaker erlaubt es, diese Einstellung *pro Dokument* vorzunehmen, Excel hingegen nur programmweit. Diese Eigenschaft ist bei PlanMaker nur aus Kompatibilitätsgründen vorhanden. Vorzuziehen ist die gleichnamige Eigenschaft **Calculation** im **Workbook**-Objekt, da sich diese gezielt auf jeweils ein Dokument bezieht.
- Wenn Sie die Eigenschaft abfragen und sich die Einstellungen in den verschiedenen geöffneten Dokumenten unterscheiden, wird **smoUndefined** zurückgeliefert.

CalculateBeforeSave (Eigenschaft)

Datentyp: **Boolean**

Liest oder setzt die Einstellung, ob Dokumente vor dem Speichern neu berechnet werden.

Hinweise:

- Diese Eigenschaft hat nur dann eine Wirkung, wenn die Berechnungen manuell aktualisiert werden. Ist die Eigenschaft **Calculation** (siehe dort) auf **pmCalculationAutomatic** gesetzt, werden die Berechnungen ohnehin ständig auf dem aktuellsten Stand gehalten.
- PlanMaker erlaubt es, diese Einstellung *pro Dokument* vorzunehmen, Excel hingegen nur programmweit. Diese Eigenschaft ist bei PlanMaker nur aus Kompatibilitätsgründen vorhanden. Vorzuziehen ist die gleichnamige Eigenschaft **CalculateBeforeSave** im **Workbook**-Objekt, da sich diese gezielt auf jeweils ein Dokument bezieht.
- Wenn Sie die Eigenschaft abfragen und sich die Einstellungen in den verschiedenen geöffneten Dokumenten unterscheiden, wird **smoUndefined** zurückgeliefert.

DisplayCommentIndicator (Eigenschaft)

Datentyp: **Long** (PmCommentDisplayMode)

Liest oder setzt die Art und Weise, in der Kommentare angezeigt werden. Mögliche Werte:

```
pmNoIndicator          = 0 ' Weder Kommentare noch gelbes Dreieck
pmCommentIndicatorOnly = 1 ' Nur ein gelbes Hinweisdreieck in der Zelle
pmCommentOnly          = 2 ' Kommentare zeigen, aber kein Hinweisdreieck
pmCommentAndIndicator  = 3 ' Sowohl Kommentare als auch Dreieck zeigen
```

Hinweise:

- PlanMaker erlaubt es, diese Einstellung *pro Dokument* vorzunehmen, Excel hingegen nur programmweit. Diese Eigenschaft ist bei PlanMaker nur aus Kompatibilitätsgründen vorhanden. Vorzuziehen ist die gleichnamige Eigenschaft **DisplayCommentIndicator** im **Workbook**-Objekt, da sich diese gezielt auf jeweils ein Dokument bezieht.
- Wenn Sie die Eigenschaft abfragen und sich die Einstellungen in den verschiedenen geöffneten Dokumenten unterscheiden, wird **smoUndefined** zurückgeliefert.

EditDirectlyInCell (Eigenschaft)

Datentyp: **Boolean**

Liest oder setzt die Einstellung, ob Sie Zellen direkt im Arbeitsblatt editieren können oder nur in der Eingabeleiste oberhalb des Arbeitsblatts.

MoveAfterReturn (Eigenschaft)

Datentyp: **Boolean**

Liest oder setzt die Einstellung, ob PlanMaker nach dem Drücken der Wagenrücklauftaste den Zellenrahmen in eine andere Zelle verschiebt.

Wenn Sie diese Eigenschaft auf **True** setzen, wird automatisch die Eigenschaft **MoveAfterReturnDirection** (siehe dort) auf **pmDown** gesetzt. Sie können danach aber jede andere Bewegungsrichtung festlegen.

MoveAfterReturnDirection (Eigenschaft)

Datentyp: **Long** (PmDirection)

Liest oder setzt die Richtung, in der der Zellrahmen nach dem Drücken der Wagenrücklauftaste verschoben wird. Mögliche Werte:

```
pmDown    = 0 ' nach unten
pmUp      = 1 ' nach oben
pmToRight = 2 ' nach rechts
pmToLeft  = 3 ' nach links
```

PromptForSummaryInfo (Eigenschaft)

Datentyp: **Boolean**

Liest oder setzt die Einstellung "Beim Speichern nach Dokumentinfo fragen", die Sie bei PlanMaker in **Weiteres > Einstellungen > Dateien** finden.

WarningOnError (Eigenschaft)

Datentyp: **Boolean**

Liest oder setzt die Einstellung "Warnung bei fehlerhaften Formeln", die Sie bei PlanMaker in **Weiteres > Einstellungen > Bearbeiten** finden.

Options (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das **Options**-Objekt, mit dem Sie auf diverse globale Programmeinstellungen von PlanMaker zugreifen können.

UserProperties (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das **UserProperties**-Objekt, mit dem Sie auf den Namen und die Adresse des Anwenders zugreifen können.

CommandBars (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das **CommandBars**-Objekt, mit dem Sie auf die Symbolleisten von PlanMaker zugreifen können.

AutoCorrect (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das **AutoCorrect**-Objekt, mit dem Sie auf die Textbausteine von PlanMaker zugreifen können.

ActiveWorkbook (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das gerade aktive **Workbook**-Objekt, über das Sie auf das aktuelle Dokument zugreifen können.

ActiveSheet (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das gerade aktive **Sheet**-Objekt, über das Sie auf das aktive Arbeitsblatt des aktuellen Dokuments zugreifen können.

ActiveSheet ist eine Abkürzung für **ActiveWorkbook.ActiveSheet**. Die folgenden beiden Aufrufe sind daher gleichwertig:

```
MsgBox pm.Application.ActiveWorkbook.ActiveSheet
MsgBox pm.Application.ActiveSheet
```

ActiveWindow (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das gerade aktive **Window**-Objekt, über das Sie auf das aktuelle Dokumentfenster zugreifen können.

ActiveCell (Zeiger auf Objekt)

Datentyp: **Object**

Liefert ein **Range**-Objekt, das die im aktuellen Dokumentfenster aktive Zelle repräsentiert. Mit diesem Objekt können Sie die Formatierung und den Inhalt der Zelle lesen und bearbeiten.

ActiveCell ist eine Abkürzung für **ActiveWindow.ActiveCell**. Die folgenden beiden Aufrufe sind daher gleichwertig:

```
pm.Application.ActiveWindow.ActiveCell.Font.Size = 14
pm.Application.ActiveCell.Font.Size = 14
```

Bitte beachten Sie, dass **ActiveCell** auch dann nur eine *einzige Zelle* liefert, wenn im Arbeitsblatt ein Bereich selektiert ist. Denn der Zellrahmen kann innerhalb der Selektion an jeder beliebigen Stelle stehen: Ziehen Sie probehalber mit der Maus einen Bereich auf und drücken Sie dann wiederholt die Wagenrücklauftaste – Sie werden sehen, dass sich der Zellrahmen innerhalb der Selektion verschiebt.

Selection (Zeiger auf Objekt)

Datentyp: **Object**

Liefert ein **Range**-Objekt, das die selektierten Zellen im aktiven Arbeitsblatt des aktuellen Dokumentfensters repräsentiert.

Selection ist eine Abkürzung für **ActiveWorkbook.ActiveSheet.Selection**. Die folgenden beiden Aufrufe sind daher gleichwertig:

```
pm.Application.ActiveWorkbook.ActiveSheet.Selection.Font.Size = 14
pm.Application.Selection.Font.Size = 14
```

Range (Zeiger auf Objekt)

Datentyp: **Object**

Erzeugt anhand der übergebenen Parameter ein **Range**-Objekt, das sich auf das aktive Arbeitsblatt des aktuellen Dokuments bezieht, und liefert dieses zurück. Sie können mit diesem Objekt dann auf die Zellen im Bereich zugreifen und beispielsweise Werte auslesen und setzen.

Syntax 1:

```
obj = Range(Cell1)
```

Syntax 2:

```
obj = Range(Cell1, Cell2)
```

Parameter:

Cell1 (Typ: **String**) gibt entweder gemäß Syntax 1 einen Zellbereich an (dann muss **Cell2** weggelassen werden) oder gemäß Syntax 2 die linke obere Ecke eines Bereichs (dann gibt der Parameter **Cell2** die rechte untere Ecke des Bereichs an).

Cell2 (optional; Typ: **String**) darf nur verwendet werden, wenn **Cell1** eine einzelne Zelle referenziert, und gibt die rechte untere Ecke des Bereichs an).

Beispiele für Syntax 1:

```
Range("A1:B20")       ' Zellen A1 bis B20
Range("A1")           ' Nur Zelle A1
Range("A:A")          ' Gesamte Spalte A
Range("3:3")          ' Gesamte Zeile 3
Range("Sommer")       ' Benannter Bereich "Sommer"
```

Beispiel für Syntax 2:

```
Range("A1", "B20")    ' Zellen A1 bis B20
```

Range ist eine Abkürzung für **ActiveWorkbook.ActiveSheet.Range**. Die folgenden beiden Aufrufe sind daher gleichwertig:

```
pm.Application.ActiveWorkbook.ActiveSheet.Range("A1:B5").Value = 42
pm.Application.Range("A1:B5").Value = 42
```

Cells (Zeiger auf Objekt)

Datentyp: **Object**

Liefert ein **Range**-Objekt, das sämtliche Zellen des aktuellen Arbeitsblatts umfasst. Dies ist für zwei Anwendungsfälle nützlich:

- Sie können eine Operation (vorrangig Formatierungen) auf jede Zelle des Arbeitsblatts anwenden:

```
' Das aktuelle Arbeitsblatt komplett rot einfärben
pm.Cells.Shading.ForegroundPatternColor = smoColorRed
```

- Sie können einzelne Zellen über Schleifenvariablen adressieren, anstatt den Adressierungsstring (zum Beispiel "B5" für die zweite Spalte in der fünften Zeile) manuell zusammenzubauen. Hierzu benutzen Sie die Eigenschaft **Item** des durch den **Cells**-Zeiger adressierten **Range**-Objekts:

```
' Die ersten 5 * 10 Zellen des aktuellen Arbeitsblatts befüllen
Dim row, col as Integer
For row = 1 To 5
  For col = 1 to 10
     pm.Cells.Item(row, col).Value = 42
  Next col
Next row
```

Cells ist eine Abkürzung für **ActiveSheet.Cells**. Die folgenden beiden Aufrufe sind daher gleichwertig:

```
pm.Application.ActiveSheet.Cells(1, 1).Font.Size = 14
pm.Application.Cells(1, 1).Font.Size = 14
```

Application (Zeiger auf Objekt)

Liefert das **Application**-Objekt, also sich selbst. Dieser Objektzeiger ist eigentlich unnötig und nur der Vollständigkeit halber vorhanden.

Workbooks (Zeiger auf Sammlung)

Datentyp: **Object**

Liefert die **Workbooks**-Sammlung, eine Sammlung aller momentan geöffneten Dokumente.

Windows (Zeiger auf Sammlung)

Datentyp: **Object**

Liefert die **Windows**-Sammlung, eine Sammlung aller momentan geöffneten Dokumentfenster.

RecentFiles (Zeiger auf Sammlung)

Datentyp: **Object**

Liefert die **RecentFiles**-Sammlung, eine Sammlung der zuletzt geöffneten Dokumente.

FontNames (Zeiger auf Sammlung)

Datentyp: **Object**

Liefert die **FontNames**-Sammlung, eine Sammlung aller installierten Schriftarten.

Columns (Zeiger auf Sammlung)

Datentyp: **Object**

Liefert die **Columns**-Sammlung, eine Sammlung aller Spalten des aktuellen Arbeitsblatts.

Columns ist eine Abkürzung für **ActiveWorkbook.ActiveSheet.Columns**. Die folgenden beiden Aufrufe sind daher gleichwertig:

```
MsgBox pm.Application.ActiveWorkbook.ActiveSheet.Columns.Count
MsgBox pm.Application.Columns.Count
```

Rows (Zeiger auf Sammlung)

Datentyp: **Object**

Liefert die **Rows**-Sammlung, eine Sammlung aller Zeilen des aktuellen Arbeitsblatts.

Rows ist eine Abkürzung für **ActiveWorkbook.ActiveSheet.Rows**. Die folgenden beiden Aufrufe sind daher gleichwertig:

```
MsgBox pm.Application.ActiveWorkbook.ActiveSheet.Rows.Count
MsgBox pm.Application.Rows.Count
```

CentimetersToPoints (Methode)

Konvertiert den angegebenen Wert von Zentimetern (cm) in Punkt (pt). Diese Funktion ist nützlich, wenn Sie mit Zentimetern rechnen, eine PlanMaker-Funktion als Maßeinheit aber nur Punkt akzeptiert.

Syntax:

```
CentimetersToPoints(Centimeters)
```

Parameter:

Centimeters (Typ: **Single**) gibt den umzurechnenden Wert an.

Rückgabetyp:

Single

Beispiel:

```
' Den oberen Rand des aktiven Arbeitsblatts auf 3 Zentimeter setzen
pm.ActiveSheet.PageSetup.TopMargin = pm.Application.CentimetersToPoints(3)
```

MillimetersToPoints (Methode)

Konvertiert den angegebenen Wert von Millimetern (mm) in Punkt (pt). Diese Funktion ist nützlich, wenn Sie mit Millimetern rechnen, eine PlanMaker-Funktion als Maßeinheit aber nur Punkt akzeptiert.

Syntax:

```
MillimetersToPoints(Millimeters)
```

Parameter:

Millimeters (Typ: **Single**) gibt den umzurechnenden Wert an.

Rückgabetyp:

Single

Beispiel:

```
' Den oberen Rand des aktiven Arbeitsblatts auf 30 Millimeter setzen
pm.ActiveSheet.PageSetup.TopMargin = pm.Application.MillimetersToPoints(30)
```

InchesToPoints (Methode)

Konvertiert den angegebenen Wert von Zoll (in) in Punkt (pt). Diese Funktion ist nützlich, wenn Sie mit Zoll rechnen, eine PlanMaker-Funktion als Maßeinheit aber nur Punkt akzeptiert.

Syntax:

```
InchesToPoints(Inches)
```

Parameter:

Inches (Typ: **Single**) gibt den umzurechnenden Wert an.

Rückgabetyp:

Single

Beispiel:

```
' Den unteren Rand des aktiven Arbeitsblatts auf 1 Zoll setzen
pm.ActiveSheet.PageSetup.BottomMargin = pm.Application.InchesToPoints(1)
```

PicasToPoints (Methode)

Konvertiert den angegebenen Wert von Pica in Punkt (pt). Diese Funktion ist nützlich, wenn Sie mit Pica rechnen, eine PlanMaker-Funktion als Maßeinheit aber nur Punkt akzeptiert.

Syntax:

PicasToPoints(Picas)

Parameter:

Picas (Typ: **Single**) gibt den umzurechnenden Wert an.

Rückgabetyp:

Single

Beispiel:

```
' Den unteren Rand des aktiven Arbeitsblatts auf 6 Pica setzen
pm.ActiveSheet.PageSetup.BottomMargin = pm.Application.PicasToPoints(6)
```

LinesToPoints (Methode)

Identisch mit **PicasToPoints** (siehe dort).

Syntax:

LinesToPoints(Lines)

Parameter:

Lines (Typ: **Single**) gibt den umzurechnenden Wert an.

Rückgabetyp:

Single

Beispiel:

```
' Den unteren Rand des aktiven Arbeitsblatts auf 6 Pica setzen
pm.ActiveSheet.PageSetup.BottomMargin = pm.Application.LinesToPoints(6)
```

Activate (Methode)

Bringt das Programmfenster in den Vordergrund und setzt den Fokus darauf.

Syntax:

Activate

Parameter:

keine

Rückgabetyp:

keiner

Beispiel:

```
' PlanMaker in den Vordergrund holen
pm.Application.Activate
```

Hinweis: Damit die Funktion erfolgreich ausgeführt werden kann, muss **Application.Visible** = **True** sein.

Calculate (Methode)

Berechnet *alle* derzeit geöffneten Dokumente neu (ähnlich dem Menübefehl **Weiteres > Neu berechnen** in PlanMaker, nur dass der Menübefehl nur die *aktive* Arbeitsmappe neu berechnet).

Syntax:

Calculate

Parameter:

keine

Rückgabetyp:

keiner

Beispiel:

```
' Alle geöffneten Dokumente neu berechnen
pm.Application.Calculate
```

Quit (Methode)

Beendet das Programm.

Syntax:

Quit

Parameter:

keine

Rückgabetyp:

keiner

Beispiel:

```
' PlanMaker beenden
pm.Application.Quit
```

Sind noch ungespeicherte Dokumente geöffnet, wird der Benutzer gefragt, ob diese gespeichert werden sollen. Wenn Sie diese Frage vermeiden wollen, sollten Sie entweder alle offenen Dokumente von Ihrem Programm aus schließen oder bei dem Dokument die Eigenschaft **Saved** auf **True** setzen (siehe **Workbook**).

Options (Objekt)

Zugriffspfad: Application → **Options**

1 Beschreibung

Im **Options**-Objekt sind diverse globale Programmeinstellungen zusammengefasst, von denen Sie in PlanMaker die meisten im Dialogfenster **Weiteres > Einstellungen** finden.

2 Zugriff auf das Objekt

Es existiert während der gesamten Laufzeit von PlanMaker genau eine Instanz des **Options**-Objekts. Diese wird über **Application.Options** angesprochen:

```
Set pm = CreateObject("PlanMaker.Application")
pm.Application.Options.EnableSound = True
```

3 Eigenschaften, Objekte, Sammlungen und Methoden

Eigenschaften:

- **CheckSpellingAsYouType**
- **CreateBackup**
- **DefaultFilePath**
- **DefaultTemplatePath**
- **EnableSound**
- **Overtype**
- **SaveInterval**
- **SavePropertiesPrompt**

Objekte:

- **Application** → **Application**
- ***Parent*** → **Application** (Defaultobjekt)

CheckSpellingAsYouType (Eigenschaft)

Datentyp: **Boolean**

Liest oder setzt die Einstellung "Rechtschreibkorrektur während des Tippens" (**True** oder **False**).

CreateBackup (Eigenschaft)

Datentyp: **Boolean**

Liest oder setzt die Einstellung ".BAK-Dateien anlegen" (**True** oder **False**).

DefaultFilePath (Eigenschaft)

Datentyp: **String**

Liest oder setzt den Dateipfad, unter dem Dokumente standardmäßig gespeichert und geöffnet werden.

Dies ist lediglich eine temporäre Einstellung: Beim nächsten Aufruf von **Datei > Öffnen** oder **Datei > Speichern unter** erscheint der hier gewählte Pfad. Ändert der Benutzer diesen aber ab, wird ab diesem Zeitpunkt der vom Benutzer ausgewählte Pfad voreingestellt.

DefaultTemplatePath (Eigenschaft)

Datentyp: **String**

Liest oder setzt den Dateipfad, unter dem Dokumentvorlagen standardmäßig gespeichert werden.

Diese Einstellung wird dauerhaft gespeichert. Bei jedem Aufruf von **Datei > Neu** erscheinen die Dokumentvorlagen im hier angegebenen Pfad.

EnableSound (Eigenschaft)

Datentyp: **Boolean**

Liest oder setzt die Einstellung "Warnton bei Meldungen" (**True** oder **False**).

Overtype (Eigenschaft)

Datentyp: **Boolean**

Liest oder setzt den Überschreibe-/Einfügemodus (**True**=Überschreiben, **False**=Einfügen).

SaveInterval (Eigenschaft)

Datentyp: **Long**

Liest oder setzt die Einstellung "Automatisches Sichern alle *n* Minuten" (0=aus).

SavePropertiesPrompt (Eigenschaft)

Datentyp: **Boolean**

Liest oder setzt die Einstellung "Beim Speichern nach Dokumentinfo fragen" (**True** oder **False**).

Application (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das **Application**-Objekt.

Parent (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das übergeordnete Objekt, also **Application**.

UserProperties (Sammlung)

Zugriffspfad: Application → **UserProperties**

1 Beschreibung

Die Sammlung **UserProperties** enthält die Privat- und Geschäftsadresse des Benutzers (sofern er dies im Dialog des Befehls **Weiteres > Einstellungen**, Karteikarte **Allgemein**, eingetragen hat).

Die einzelnen Elemente dieser Sammlung sind vom Typ **UserProperty**.

2 Zugriff auf die Sammlung

Es existiert während der gesamten Laufzeit von PlanMaker genau eine Instanz der **UserProperties**-Sammlung. Diese wird über **Application.UserProperties** angesprochen:

```
' Zeige die erste UserProperty (den Nachnamen des Benutzers) an
MsgBox pm.Application.UserProperties.Item(1).Value
```

3 Eigenschaften, Objekte, Sammlungen und Methoden

Eigenschaften:
- **Count** R/O

Objekte:
- ***Item*** → **UserProperty** (Defaultobjekt)
- **Application** → **Application**
- **Parent** → **Application**

Count (Eigenschaft, R/O)

Datentyp: **Long**

Liefert die Anzahl der **UserProperty**-Objekte in der Sammlung, also die Zahl aller Adressbestandteile (Nachname, Vorname, Straße etc. – jeweils für Privat- und Geschäftsadresse).

Dieser Wert ist konstant 24, da es genau 24 derartige Elemente gibt.

Item (Zeiger auf Objekt)

Datentyp: **Object**

Liefert ein einzelnes **UserProperty**-Objekt, mit dem Sie einen einzelnen Adressbestandteil (Nachname, Vorname, Straße etc.) der privaten oder geschäftlichen Adresse des Benutzers lesen oder setzen können.

Welches UserProperty-Objekt Sie erhalten, hängt von dem Zahlenwert ab, den Sie an **Item** übergeben. Die folgende Tabelle zeigt die erlaubten Werte:

```
smoUserHomeAddressName              = 1  ' Nachname (privat)
smoUserHomeAddressFirstName         = 2  ' Vorname (privat)
smoUserHomeAddressStreet            = 3  ' Straße (privat)
smoUserHomeAddressZip               = 4  ' Postleitzahl (privat)
smoUserHomeAddressCity              = 5  ' Stadt (privat)
smoUserHomeAddressPhone1            = 6  ' Telefon (privat)
smoUserHomeAddressFax               = 7  ' Telefax (privat)
smoUserHomeAddressEmail             = 8  ' E-Mail-Adresse (privat)
smoUserHomeAddressPhone2            = 9  ' Mobiltelefon o.ä. (privat)
smoUserHomeAddressHomepage          = 10 ' Homepage (privat)

smoUserBusinessAddressName          = 11 ' Nachname (geschäftlich)
smoUserBusinessAddressFirstName     = 12 ' Vorname (geschäftlich)
smoUserBusinessAddressCompany       = 13 ' Firma (geschäftlich)
smoUserBusinessAddressDepartment    = 14 ' Abteilung (geschäftlich)
smoUserBusinessAddressStreet        = 15 ' Straße (geschäftlich)
smoUserBusinessAddressZip           = 16 ' Postleitzahl (geschäftlich)
smoUserBusinessAddressCity          = 17 ' Stadt (geschäftlich)
smoUserBusinessAddressPhone1        = 18 ' Telefon (geschäftlich)
smoUserBusinessAddressFax           = 19 ' Telefax (geschäftlich)
smoUserBusinessAddressEmail         = 20 ' E-Mail-Adresse (geschäftlich)
smoUserBusinessAddressPhone2        = 21 ' Mobiltelefon o.ä. (geschäftlich)
smoUserBusinessAddressHomepage      = 22 ' Homepage (geschäftlich)

smoUserHomeAddressInitials          = 23 ' Initialen des Benutzers (privat)
smoUserBusinessAddressInitials      = 24 ' Initialen des Benutzers (geschäftlich)
```

Beispiele:

```
' Den Nachnamen des Benutzers (privat) anzeigen
```

```
MsgBox pm.Application.UserProperties.Item(1).Value

' Die geschäftliche E-Mail-Adresse auf test@example.com ändern
With pm.Application
  .UserProperties.Item(smoUserBusinessAddressEmail).Value = "test@example.com"
End With
```

Application (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das **Application**-Objekt.

Parent (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das übergeordnete Objekt, also **Application**.

UserProperty (Objekt)

Zugriffspfad: Application → UserProperties → **Item**

1 Beschreibung

Ein **UserProperty**-Objekt repräsentiert einen einzelnen Teil (zum Beispiel Straße oder Postleitzahl) der vom Benutzer eingegebenen privaten und geschäftlichen Adresse.

Für jeden dieser Bestandteile existiert ein eigenes **UserProperty**-Objekt. Die Zahl dieser Objekte ist konstant, da Sie zwar die einzelnen Adressbestandteile bearbeiten, nicht aber neue anlegen können.

2 Zugriff auf das Objekt

Die einzelnen **UserProperty**-Objekte können ausschließlich durch Aufzählung der Elemente der Sammlung **Application.UserProperties** angesprochen werden. Der Typ dieser Sammlung ist **UserProperties**.

Beispiel:

```
' Den Inhalt des ersten Adressbestandteils (Nachname privat) anzeigen
MsgBox pm.Application.UserProperties.Item(1).Value
```

3 Eigenschaften, Objekte, Sammlungen und Methoden

Eigenschaften:
- ***Value*** (Defaulteigenschaft)

Objekte:
- **Application → Application**
- **Parent → UserProperties**

Value (Eigenschaft)

Datentyp: **String**

Liest oder setzt den Inhalt des Adressbestandteils. Das folgende Beispiel setzt den Firmennamen des Benutzers:

```
Sub Beispiel()
  Set pm = CreateObject("PlanMaker.Application")
  pm.UserProperties(smoUserBusinessAddressCompany).Value = "ACME Corporation"
End Sub
```

Application (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das **Application**-Objekt.

Parent (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das übergeordnete Objekt, also **UserProperties**.

CommandBars (Sammlung)

Zugriffspfad: Application → **CommandBars**

1 Beschreibung

Die Sammlung **CommandBars** enthält alle Symbolleisten von PlanMaker. Die einzelnen Elemente dieser Sammlung sind vom Typ **CommandBar**.

2 Zugriff auf die Sammlung

Es existiert während der gesamten Laufzeit von PlanMaker genau eine Instanz der **CommandBars**-Sammlung. Diese wird über **Application.CommandBars** angesprochen:

```
' Zeige den Namen der ersten Symbolleiste von PlanMaker an
MsgBox pm.Application.CommandBars.Item(1).Name

' Dasselbe einfacher durch Nutzung der Defaulteigenschaft
MsgBox pm.CommandBars(1)
```

3 Eigenschaften, Objekte, Sammlungen und Methoden

Eigenschaften:
- **Count** R/O
- **DisplayFonts**
- **DisplayTooltips**

Objekte:
- ***Item*** → **CommandBar** (Defaultobjekt)
- **Application** → **Application**
- **Parent** → **Application**

Count (Eigenschaft, R/O)

Datentyp: **Long**

Liefert die Anzahl der **CommandBar**-Objekte in der Sammlung, also die Zahl aller Symbolleisten von PlanMaker.

DisplayFonts (Eigenschaft)

Datentyp: **Boolean**

Liest oder setzt die Einstellung "Schriftenliste mit echten Schriften" (**True** oder **False**).

DisplayTooltips (Eigenschaft)

Datentyp: **Boolean**

Liest oder setzt die Einstellung von PlanMaker, ob QuickInfos (Tooltips) angezeigt werden, wenn die Maus über eine Schaltfläche in den Symbolleisten bewegt wird. Entspricht der Einstellung "QuickInfos" im Dialogfenster des Befehls **Weiteres > Einstellungen**.

Item (Zeiger auf Objekt)

Datentyp: **Object**

Liefert ein einzelnes **CommandBar**-Objekt, mit dem Sie auf eine einzelne Symbolleiste von PlanMaker zugreifen können.

Welches CommandBar-Objekt Sie erhalten, hängt von dem Parameter ab, den Sie an **Item** übergeben. Dies kann entweder der numerische Index oder der Name der gewünschten Symbolleiste sein. Beispiele:

```
' Mache die erste Symbolleiste unsichtbar
pm.Application.CommandBars.Item(1).Visible = False

' Mache die Formatleiste unsichtbar
pm.Application.CommandBars.Item("Format").Visible = False
```

Sie sollten aber Namen von Symbolleisten nicht fest in Ihr Programm eintragen, da sich diese Namen an die Sprache der gewählten Benutzeroberfläche von PlanMaker anpassen. Betreiben Sie PlanMaker beispielsweise in englischer Sprache, heißt die Formatleiste nicht "Format" sondern "Formatting".

Daher ist es empfehlenswerter, die folgenden symbolischen Konstanten für die Symbolleisten zu verwenden:

```
pmBarStatusShort     = 1 ' Statusleiste (ohne geöffnete Dokumente)
pmBarStandardShort   = 2 ' Funktionsleiste (ohne geöffnete Dokumente)
pmBarStatus          = 3 ' Statusleiste
pmBarStandard        = 4 ' Funktionsleiste
pmBarFormatting      = 5 ' Formatleiste
pmBarObjects         = 6 ' Objektleiste
pmBarEdit            = 7 ' Bearbeitungsleiste
pmBarOutliner        = 8 ' Gliederungsleiste
pmBarChart           = 9 ' Diagrammleiste
pmBarFormsEditing    = 10 ' Formularleiste
pmBarPicture         = 11 ' Grafikleiste
pmBarFullscreen      = 12 ' Vollbildleiste
```

Application (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das **Application**-Objekt.

Parent (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das übergeordnete Objekt, also **Application**.

CommandBar (Objekt)

Zugriffspfad: Application → CommandBars → **Item**

1 Beschreibung

Ein **CommandBar**-Objekt repräsentiert eine einzelne Symbolleiste von PlanMaker.

Für jede Symbolleiste existiert ein eigenes **CommandBar**-Objekt. Richten Sie neue Symbolleisten ein oder löschen diese, werden die zugehörigen **CommandBar**-Objekte dynamisch angelegt bzw. entfernt.

2 Zugriff auf das Objekt

Die einzelnen **CommandBar**-Objekte können ausschließlich durch Aufzählung der Elemente der Sammlung **Application.CommandBars** angesprochen werden. Der Typ dieser Sammlung ist **CommandBars**.

Beispiel:

```
' Zeige den Namen der ersten Symbolleiste von PlanMaker an
MsgBox pm.Application.CommandBars.Item(1).Name

' Dasselbe einfacher durch Nutzung der Defaulteigenschaft
MsgBox pm.CommandBars(1)
```

3 Eigenschaften, Objekte, Sammlungen und Methoden

Eigenschaften:
- ***Name*** (Defaulteigenschaft)
- **Visible**

Objekte:
- **Application → Application**
- **Parent → CommandBars**

Name (Eigenschaft)

Datentyp: **String**

Liest oder setzt den Namen der Symbolleiste.

Beispiel:

```
' Zeige den Namen der ersten Symbolleiste an
MsgBox pm.Application.CommandBars.Item(1).Name
```

Visible (Eigenschaft)

Datentyp: **Boolean**

Liest oder setzt die Sichtbarkeit der Symbolleiste. Das folgende Beispiel macht die Formatleiste unsichtbar:

```
Sub Beispiel()
  Set pm = CreateObject("PlanMaker.Application")
  pm.Application.CommandBars.Item("Format").Visible = False
End Sub
```

Application (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das **Application**-Objekt.

Parent (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das übergeordnete Objekt, also **CommandBars**.

AutoCorrect (Objekt)

Zugriffspfad: Application → **AutoCorrect**

1 Beschreibung

Das **AutoCorrect**-Objekt lässt Sie die Textbausteine von PlanMaker bearbeiten.

2 Zugriff auf das Objekt

Es existiert während der gesamten Laufzeit von PlanMaker genau eine Instanz des **AutoCorrect**-Objekts. Diese wird über **Application.AutoCorrect** angesprochen:

```
' Zahl der Textbausteine anzeigen
Set pm = CreateObject("PlanMaker.Application")
MsgBox pm.Application.AutoCorrect.Entries.Count
```

3 Eigenschaften, Objekte, Sammlungen und Methoden

Objekte:
- **Application** → **Application**
- **Parent** → **Application**

Sammlungen:
- **Entries** → **AutoCorrectEntries**

Application (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das **Application**-Objekt.

Parent (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das übergeordnete Objekt, also **Application**.

Entries (Zeiger auf Sammlung)

Datentyp: **Object**

Liefert die **AutoCorrectEntries**-Sammlung, die alle Textbausteine von PlanMaker enthält.

AutoCorrectEntries (Sammlung)

Zugriffspfad: Application → AutoCorrect → **Entries**

1 Beschreibung

Die Sammlung **AutoCorrectEntries** enthält alle in PlanMaker definierten Textbausteine. Die einzelnen Elemente dieser Sammlung sind vom Typ **AutoCorrectEntry**.

2 Zugriff auf die Sammlung

Es existiert während der gesamten Laufzeit von PlanMaker genau eine Instanz der **AutoCorrectEntries**-Sammlung. Diese wird über **Application.AutoCorrect.Entries** angesprochen:

```
Set pm = CreateObject("PlanMaker.Application")
pm.Application.AutoCorrect.Entries.Add "lax", "Los Angeles"
```

3 Eigenschaften, Objekte, Sammlungen und Methoden

Eigenschaften:
- **Count** R/O

Objekte:
- ***Item*** → **AutoCorrectEntry** (Defaultobjekt)
- **Application** → **Application**
- **Parent** → **AutoCorrect**

Methoden:
- **Add**

Count (Eigenschaft, R/O)

Datentyp: **Long**

Liefert die Anzahl der **AutoCorrectEntry**-Objekte in der Sammlung, also die Zahl der momentan definierten Textbausteine.

Item (Zeiger auf Objekt)

Datentyp: **Object**

Liefert ein einzelnes **AutoCorrectEntry**-Objekt, die Definition eines einzelnen Textbausteins.

Welches AutoCorrect-Objekt Sie erhalten, hängt von dem Parameter ab, den Sie an **Item** übergeben. Dies kann entweder der numerische Index oder der Name des gewünschten Textbausteins sein. Beispiele:

```
' Den Inhalt des ersten definierten Textbausteins anzeigen
MsgBox pm.Application.AutoCorrect.Entries.Item(1).Value

' Den Inhalt des Textbausteins mit dem Namen "mfg" anzeigen
MsgBox pm.Application.AutoCorrect.Entries.Item("mfg").Value
```

Application (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das **Application**-Objekt.

Parent (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das übergeordnete Objekt, also **AutoCorrect**.

Add (Methode)

Fügt einen neuen **AutoCorrectEntry**-Eintrag hinzu.

Syntax:

```
Add Name, Value
```

Parameter:

Name (Typ: **String**): Der gewünschte Name für den neuen Textbaustein. Wenn der Name leer ist oder bereits existiert, schlägt der Aufruf der Methode fehl.

Value (Typ: **String**): Der gewünschte Text für den neuen Textbaustein. Wenn die übergebene Zeichenkette leer ist, schlägt der Aufruf der Methode fehl.

Rückgabetyp:

Object (ein **AutoCorrectEntry**-Objekt, das den neuen Textbaustein repräsentiert)

Beispiel:

```
' Den Baustein "lax" mit dem Inhalt "Los Angeles" belegen
pm.Application.AutoCorrect.Entries.Add "lax", "Los Angeles"
```

AutoCorrectEntry (Objekt)

Zugriffspfad: Application → AutoCorrect → Entries → **Item**

1 Beschreibung

Ein **AutoCorrectEntry**-Objekt repräsentiert einen einzelnen Textbaustein in PlanMaker, zum Beispiel "mfg" für "Mit freundlichen Grüßen".

Für jeden Textbaustein existiert ein eigenes **AutoCorrectEntry**-Objekt. Legen Sie Textbausteine an oder löschen diese, werden die zugehörigen **AutoCorrectEntry**-Objekte dynamisch angelegt bzw. entfernt.

2 Zugriff auf das Objekt

Die einzelnen **AutoCorrectEntry**-Objekte können ausschließlich durch Aufzählung der Elemente der Sammlung **Application.AutoCorrect.Entries** angesprochen werden. Der Typ dieser Sammlung ist **AutoCorrectEntries**.

Beispiel:

```
' Den Namen des ersten Textbausteins anzeigen
MsgBox pm.Application.AutoCorrect.Entries.Item(1).Name
```

3 Eigenschaften, Objekte, Sammlungen und Methoden

Eigenschaften:
- ***Name*** (Defaulteigenschaft)
- **Value**

Objekte:
- **Application → Application**
- **Parent → AutoCorrectEntries**

Methoden:
- **Delete**

Name (Eigenschaft)

Datentyp: **String**

Liest oder setzt den Namen des Textbausteins (z.B. "mfg").

Value (Eigenschaft)

Datentyp: **String**

Liest oder setzt den Inhalt des Textbausteins (z.B. "Mit freundlichen Grüßen").

Application (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das **Application**-Objekt.

Parent (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das übergeordnete Objekt, also **AutoCorrectEntries**.

Delete (Methode)

Löscht das aktuelle **AutoCorrectEntry**-Objekt aus der **AutoCorrectEntries**-Sammlung.

Syntax:

```
Delete
```

Parameter:

keine

Rückgabetyp:

keiner

Beispiele:

```
' Den ersten Textbaustein löschen
pm.Application.AutoCorrect.Entries.Item(1).Delete

' Den Textbaustein mit dem Namen "mfg" löschen
```

```
pm.Application.AutoCorrect.Entries.Item("mfg").Delete
```

Workbooks (Sammlung)

Zugriffspfad: Application → **Workbooks**

1 Beschreibung

Die Sammlung **Workbooks** enthält alle geöffneten Dokumente. Die einzelnen Elemente dieser Sammlung sind vom Typ **Workbook**.

2 Zugriff auf die Sammlung

Es existiert während der gesamten Laufzeit von PlanMaker genau eine Instanz der **Workbooks**-Sammlung. Diese wird über **Application.Workbooks** angesprochen:

```
' Die Anzahl der offenen Dokumente anzeigen
MsgBox pm.Application.Workbooks.Count

' Den Namen des ersten geöffneten Dokuments anzeigen
MsgBox pm.Application.Workbooks(1).Name
```

3 Eigenschaften, Objekte, Sammlungen und Methoden

Eigenschaften:
- **Count** R/O

Objekte:
- ***Item*** → **Workbook** (Defaultobjekt)
- **Application** → **Application**
- **Parent** → **Application**

Methoden:
- **Add**
- **Open**
- **Close**

Count (Eigenschaft, R/O)

Datentyp: **Long**

Liefert die Anzahl der **Workbook**-Objekte in der Sammlung, also die Zahl der momentan geöffneten Dokumente.

Item (Zeiger auf Objekt)

Datentyp: **Object**

Liefert ein einzelnes **Workbook**-Objekt, also ein einzelnes Dokument.

Welches Workbook-Objekt Sie erhalten, hängt von dem Parameter ab, den Sie an **Item** übergeben. Dies kann entweder der numerische Index oder der Dateiname des gewünschten Dokuments sein. Beispiele:

```
' Den Namen des ersten Dokuments anzeigen
MsgBox pm.Application.Workbooks.Item(1).FullName
```

```
' Den Namen des Dokuments "Test.tmd" anzeigen (sofern gerade geöffnet)
MsgBox pm.Application.Workbooks.Item("Test.pmd").FullName

' Sie können auch den kompletten Namen mit Pfadangabe verwenden
MsgBox pm.Application.Workbooks.Item("c:\Dokumente\Test.pmd").FullName
```

Application (Zeiger auf Objekt)

Liefert das **Application**-Objekt.

Parent (Zeiger auf Objekt)

Liefert das übergeordnete Objekt, also **Application**.

Add (Methode)

Legt ein neues leeres Dokument an, wahlweise basierend auf der Standarddokumentvorlage **Normal.pmv** oder einer anderen von Ihnen gewählten Dokumentvorlage.

Syntax:

```
Add [Template]
```

Parameter:

Template (optional; Typ: **String**): Der Pfad und Dateiname der Dokumentvorlage, auf der das Dokument basieren soll. Wird dieser Parameter nicht angegeben, basiert das Dokument auf der Standardvorlage **Normal.pmv**.

Lassen Sie den Pfad weg oder geben nur einen relativen Pfad an, wird automatisch der Standardvorlagenpfad von PlanMaker vorangestellt. Lassen Sie die Dateierweiterung **.pmv** weg, wird sie automatisch angehängt.

Rückgabetyp:

Object (ein **Workbook**-Objekt, das das neue Dokument repräsentiert)

Beispiel:

```
Sub Sample()
  Dim pm as Object
  Dim newDoc as Object

  Set pm = CreateObject("PlanMaker.Application")
  pm.Visible = True
  Set newDoc = pm.Workbooks.Add
  MsgBox newDoc.Name
End Sub
```

Mit dem von **Add** zurückgegebenen **Document** können Sie arbeiten wie mit jedem anderen Dokument. Sie können aber auch den Rückgabewert von **Add** ignorieren und sich das neue Dokument beispielsweise über **ActiveWorkbook** holen.

Open (Methode)

Öffnet ein bestehendes Dokument.

Syntax:

```
Open FileName, [ReadOnly], [Format], [Password], [WritePassword], [Delimiter],
[TextMarker]
```

Parameter:

FileName (Typ: **String**): Pfad und Name des zu öffnenden Dokuments beziehungsweise der zu öffnenden Dokumentvorlage

ReadOnly (optional; Typ: **Boolean**): Gibt an, ob das Dokument nur zum Lesen geöffnet werden soll.

Format (optional; Typ: **Long** bzw. **PmSaveFormat**): Dateiformat des zu öffnenden Dokuments. Mögliche Werte:

```
pmFormatDocument          =  0 ' Dokument, ist Default
pmFormatTemplate          =  1 ' Dokumentvorlage
pmFormatExcel97           =  2 ' Excel 97/2000/XP
pmFormatExcel5            =  3 ' Excel 5.0/7.0
pmFormatExcelTemplate     =  4 ' Excel-Vorlage
pmFormatSYLK              =  6 ' Sylk
pmFormatRTF               =  7 ' Rich Text Format
pmFormatTextMaker         =  7 ' TextMaker (= RTF)
pmFormatHTML              =  8 ' HTML
pmFormatdBaseDOS          =  9 ' dBASE-Datenbank im DOS-Zeichensatz
pmFormatdBaseAnsi         = 10 ' dBASE-Datenbank im Windows-Zeichensatz
pmFormatDIF               = 11 ' Textdatei mit Windows-Zeichensatz
pmFormatPlainTextAnsi     = 12 ' Textdatei mit Windows-Zeichensatz
pmFormatPlainTextDOS      = 13 ' Textdatei mit DOS-Zeichensatz
pmFormatPlainTextUnix     = 14 ' Textdatei mit ANSI-Zeichensatz für UNIX, Linux und
FreeBSD
pmFormatPlainTextUnicode = 15 ' Textdatei mit Unicode-Zeichensatz
pmFormatdBaseUnicode      = 18 ' dBASE-Datenbank mit Unicode-Zeichensatz
pmFormatPlainTextUTF8     = 20 ' Textdatei mit UTF8-Zeichensatz
```

Wenn Sie diesen Parameter weglassen, wird **pmFormatDocument** angenommen.

Unabhängig vom übergebenen Parameter **FileFormat** versucht PlanMaker stets, das Dateiformat selbst zu erkennen, und ignoriert offensichtlich falsche Angaben.

Password (optional; Typ: **String**): Gibt bei kennwortgeschützten Dokumenten das Lesekennwort an. Lassen Sie diesen Parameter bei einem kennwortgeschützten Dokument weg, wird der Benutzer nach dem Lesekennwort gefragt.

WritePassword (optional; Typ: **String**): Gibt bei kennwortgeschützten Dokumenten das Schreibkennwort an. Lassen Sie diesen Parameter bei einem kennwortgeschützten Dokument weg, wird der Benutzer nach dem Schreibkennwort gefragt.

Delimiter (optional; Typ: **String**): Gibt bei den Textdatei-Formaten das Trennzeichen an, zum Beispiel Komma oder Strichpunkt. Wenn Sie den Parameter weglassen, wird der Tabulator als Trennzeichen verwendet.

TextMarker (optional; Typ: **Long** bzw. **PmImportTextMarker**): Gibt bei den Textdatei-Formaten an, mit welchem Zeichen Textfelder umgeben sind. Mögliche Werte:

```
pmImportTextMarkerNone          = 0 ' Text ist nicht speziell markiert
pmImportTextMarkerApostrophe    = 1 ' Apostrophe
pmImportTextMarkerQmark         = 2 ' Anführungszeichen
```

Rückgabetyp:

Object (ein **Workbook**-Objekt, das das geöffnete Dokument repräsentiert)

Beispiele:

```
' Ein Dokument öffnen
pm.Workbooks.Open "c:\doks\test.pmd"

' Ein Dokument nur zum Lesen öffnen
pm.Documents.Open "c:\doks\Test.pmd", True
```

Close (Methode)

Schließt alle momentan geöffneten Dokumente.

Syntax:

```
Close [SaveChanges]
```

Parameter:

SaveChanges (optional; Typ: **Long** bzw. **SmoSaveOptions**) gibt an, ob die seit dem letzten Speichern veränderten Dokumente gespeichert werden sollen oder nicht. Lassen Sie den Parameter weg, wird stattdessen gegebenenfalls der Benutzer gefragt. Mögliche Werte für **SaveChanges**:

```
smoDoNotSaveChanges = 0       ' Nicht fragen, nicht speichern
smoPromptToSaveChanges = 1    ' Den Benutzer fragen
smoSaveChanges = 2            ' Ohne Rückfrage speichern
```

Rückgabetyp:

keiner

Beispiel:

```
' Alle offenen Dokumente schließen, ohne sie zu speichern
pm.Workbooks.Close smoDoNotSaveChanges
```

Workbook (Objekt)

Zugriffspfade:
- Application → Workbooks → **Item**
- Application → **ActiveWorkbook**
- Application → Windows → Item → **Workbook**
- Application → ActiveWindow → **Workbook**

1 Beschreibung

Ein **Workbook**-Objekt repräsentiert ein einzelnes in PlanMaker geöffnetes Dokument.

Für jedes Dokument existiert ein eigenes **Workbook**-Objekt. Öffnen oder schließen Sie Dokumente, werden die zugehörigen **Workbook**-Objekte dynamisch angelegt bzw. entfernt.

2 Zugriff auf das Objekt

Die einzelnen **Workbook**-Objekte können auf folgenden Wegen angesprochen werden:

- Alle zu einem Zeitpunkt geöffneten Dokumente werden in der Sammlung **Application.Workbooks** (Typ: **Workbooks**) verwaltet:

```
' Die Namen aller geöffneten Dokumente anzeigen
For i = 1 To pm.Application.Workbooks.Count
  MsgBox pm.Application.Workbooks.Item(i).Name
Next i
```

- Das aktive Dokument erhalten Sie über **Application.ActiveWorkbook**:

```
' Den Namen des aktuellen Dokuments anzeigen
MsgBox pm.Application.ActiveWorkbook.Name
```

- **Workbook** ist der **Parent** des **Sheets**-Objekts, einer Sammlung aller Arbeitsblätter eines Dokuments:

```
' Den Namen des aktuellen Dokuments über einen Umweg anzeigen
MsgBox pm.Application.ActiveWorkbook.Sheets.Parent.Name
```

- Das **Window**-Objekt enthält einen Objektzeiger auf das ihm zugehörige Dokument:

```
' Über das aktive Dokumentfenster an das aktive Dokument kommen
MsgBox pm.Application.ActiveWindow.Workbook.Name
```

3 Eigenschaften, Objekte, Sammlungen und Methoden

Eigenschaften:
- ***Name*** (Defaulteigenschaft) R/O
- **FullName** R/O
- **Path** R/O
- **Saved**
- **ReadOnly**
- **EnableCaretMovement**
- **ManualApply**
- **ScreenUpdate**
- **Calculation**
- **CalculateBeforeSave**
- **CalculateBeforeCopying**
- **CalculateBeforePrinting**
- **DisplayCommentIndicator**
- **FixedDecimal**
- **FixedDecimalPlaces**
- **Iteration**
- **MaxIteration**
- **MaxChange**
- **ShowGuideLinesForTextFrames**
- **ShowHiddenObjects**
- **RoundFinalResults**
- **RoundIntermediateResults**

Objekte:
- **ActiveSheet → Sheet**
- **ActiveWindow → Window**
- **BuiltInDocumentProperties → DocumentProperties**
- **Application → Application**
- **Parent → Workbooks**

Sammlungen:
- **Sheets → Sheets**

Methoden:
- **Activate**
- **Calculate**
- **Close**
- **Save**
- **SaveAs**
- **PrintOut**

Name (Eigenschaft, R/O)

Datentyp: **String**

Liefert den Namen des Dokuments (z.B. Müller.pmd).

FullName (Eigenschaft, R/O)

Datentyp: **String**

Liefert den Pfad und Namen des Dokuments (z.B. c:\Kalkulation\Müller.pmd).

Path (Eigenschaft, R/O)

Datentyp: **String**

Liefert den Pfad des Dokuments (z.B. c:\Kalkulation).

Saved (Eigenschaft)

Datentyp: **Boolean**

Liest oder setzt die **Saved**-Eigenschaft des Dokuments. Diese bezeichnet, ob ein Dokument nach seiner letzten Speicherung verändert wurde:

- Wenn **Saved** auf **True** steht, wurde das Dokument seit dem letzten Speichern nicht mehr verändert.
- Wenn **Saved** auf **False** steht, wurde das Dokument seit dem letzten Speichern verändert. Der Anwender wird beim Schließen des Dokuments gefragt, ob es gespeichert werden soll.

ReadOnly (Eigenschaft)

Datentyp: **Boolean**

Liest oder setzt die **ReadOnly**-Eigenschaft des Dokuments.

Wenn diese Eigenschaft **True** ist, ist das Dokument gegen Änderungen durch den Benutzer geschützt. Er kann dann nichts mehr editieren, löschen oder einfügen.

Setzen Sie diese Eigenschaft auf **True**, wird automatisch die Eigenschaft **EnableCaretMovement** (siehe dort) auf **False** gesetzt. Dadurch kann dann im Dokument die Schreibmarke nicht mehr versetzt werden. Sie können aber **EnableCaretMovement** auch wieder auf **True** setzen, sodass dieses wieder möglich ist.

EnableCaretMovement (Eigenschaft)

Datentyp: **Boolean**

Liest oder setzt die **EnableCaretMovement**-Eigenschaft des Dokuments. Diese Eigenschaft ist nur sinnvoll in Kombination mit der **ReadOnly**-Eigenschaft (siehe dort).

Wenn **EnableCaretMovement True** ist, kann die Schreibmarke im (schreibgeschützten) Dokument frei bewegt werden. Wird die Eigenschaft auf **False** gesetzt, ist das Versetzen der Schreibmarke nicht mehr möglich.

ManualApply (Eigenschaft)

Datentyp: **Boolean**

Liest oder setzt die Einstellung, ob sich von Ihrem Basic-Programm gemachte Formatierungsänderungen sofort auswirken oder nicht.

Standardmäßig ist die Eigenschaft auf **False** gesetzt, wodurch sich Formatierungsanweisungen wie **Range.Font.Size = 12** sofort auswirken.

Wenn Sie aber eine große Anzahl von Formatierungen anbringen möchten, können Sie **ManualApply** auf **True** setzen. Dann sammelt PlanMaker alle Formatierungsbefehle, bis Sie die Methode **Range.ApplyFormatting** (siehe dort) aufrufen. Das bringt einen Geschwindigkeitsvorteil.

ScreenUpdate (Eigenschaft)

Datentyp: **Boolean**

Liest oder setzt die Einstellung, ob PlanMaker nach jeder Änderung den Bildschirm aktualisieren sollte.

Wenn Sie diese Eigenschaft auf **False** setzen und dann den Inhalt oder die Formatierung von Zellen ändern, wird dies solange nicht angezeigt, bis Sie diese wieder auf **True** setzen. Dies kann Geschwindigkeitsvorteile bewirken, wenn Sie viele Zellen auf einmal abändern.

Calculation (Eigenschaft)

Datentyp: **Long** (PmCalculation)

Liest oder setzt die Einstellung, ob das Dokument automatisch oder manuell neu berechnet wird. Mögliche Werte:

```
pmCalculationAutomatic   = 0 ' Berechnungen automatisch aktualisieren
pmCalculationManual      = 1 ' Berechnungen manuell aktualisieren
```

CalculateBeforeSave (Eigenschaft)

Datentyp: **Boolean**

Liest oder setzt die Einstellung, ob das Dokument vor dem Speichern neu berechnet wird.

Diese Eigenschaft hat nur dann eine Wirkung, wenn das Dokument manuell berechnet wird. Ist die Eigenschaft **Calculation** (siehe dort) auf **pmCalculationAutomatic** gesetzt, werden alle Berechnungen ohnehin ständig aktualisiert.

CalculateBeforeCopying (Eigenschaft)

Datentyp: **Boolean**

Liest oder setzt die Einstellung, ob das Dokument vor dem Kopieren oder Ausschneiden von Zellen neu berechnet wird.

Diese Eigenschaft hat nur dann eine Wirkung, wenn das Dokument manuell berechnet wird. Ist die Eigenschaft **Calculation** (siehe dort) auf **pmCalculationAutomatic** gesetzt, werden alle Berechnungen ohnehin ständig aktualisiert.

CalculateBeforePrinting (Eigenschaft)

Datentyp: **Boolean**

Liest oder setzt die Einstellung, ob das Dokument vor dem Drucken neu berechnet wird.

Diese Eigenschaft hat nur dann eine Wirkung, wenn das Dokument manuell berechnet wird. Ist die Eigenschaft **Calculation** (siehe dort) auf **pmCalculationAutomatic** gesetzt, werden alle Berechnungen ohnehin ständig aktualisiert.

DisplayCommentIndicator (Eigenschaft)

Datentyp: **Long** (PmCommentDisplayMode)

Liest oder setzt die Art und Weise, in der Kommentare angezeigt werden. Mögliche Werte:

```
pmNoIndicator          = 0 ' Weder Kommentare noch gelbes Dreieck
pmCommentIndicatorOnly = 1 ' Nur ein gelbes Hinweisdreieck in der Zelle
pmCommentOnly          = 2 ' Kommentare zeigen, aber kein Hinweisdreieck
pmCommentAndIndicator  = 3 ' Sowohl Kommentare als auch Dreieck zeigen
```

FixedDecimal (Eigenschaft)

Datentyp: **Boolean**

Liest oder setzt die Einstellung, ob der Dezimaltrenner nach der Eingabe von Zahlen automatisch verschoben werden soll.

Um *wie viele* Stellen der Dezimaltrenner verschoben wird, legt die Eigenschaft **FixedDecimalPlaces** (siehe dort) fest.

Beispiel:

```
' Dezimaltrenner nach Eingabe 2 Stellen nach links verschieben
pm.ActiveWorkbook.FixedDecimal = True
```

```
pm.ActiveWorkbook.FixedDecimalPlaces = 2 ' aus 4235 wird 42,35

' Dezimaltrenner nach Eingabe 2 Stellen nach rechts verschieben
pm.ActiveWorkbook.FixedDecimal = True
pm.ActiveWorkbook.FixedDecimalPlaces = -2 ' aus 42 wird 4200
```

FixedDecimalPlaces (Eigenschaft)

Datentyp: **Boolean**

Liest oder setzt die Einstellung, um wie viele Stellen der Dezimaltrenner nach der Eingabe von Zahlen automatisch verschoben werden soll.

Die Eigenschaft **FixedDecimal** (siehe dort) muss auf **True** stehen, damit die Verschiebung ausgeführt wird.

Iteration (Eigenschaft)

Datentyp: **Boolean**

Liest oder setzt die Einstellung "Iterationen durchführen" im Dialog **Datei > Eigenschaften**, Karteikarte **Berechnen**.

Wenn Sie diese Eigenschaft einschalten, sollten Sie auch bei **MaxChange** und **MaxIteration** (siehe dort) Werte eintragen.

MaxIteration (Eigenschaft)

Datentyp: **Long**

Liest oder setzt die Einstellung "Maximale Anzahl" (bei Iterationen) im Dialog **Datei > Eigenschaften**, Karteikarte **Berechnen**. Nur wirksam, wenn **Iteration** (siehe dort) auf **True** steht.

MaxChange (Eigenschaft)

Datentyp: **Long**

Liest oder setzt die Einstellung "Maximale Änderung" (bei Iterationen) im Dialog **Datei > Eigenschaften**, Karteikarte **Berechnen**. Nur wirksam, wenn **Iteration** (siehe dort) auf **True** steht.

ShowGuideLinesForTextFrames (Eigenschaft)

Datentyp: **Boolean**

Liest oder setzt die Einstellung "Textrahmen-Hilfslinien" im Dialog **Datei > Eigenschaften**, Karteikarte **Optionen**.

ShowHiddenObjects (Eigenschaft)

Datentyp: **Boolean**

Liest oder setzt die Einstellung "Verborgene Objekte anzeigen" im Dialog **Datei > Eigenschaften**, Karteikarte **Optionen**.

RoundFinalResults (Eigenschaft)

Datentyp: **Boolean**

Liest oder setzt die Einstellung "Endergebnisse runden" im Dialog **Datei > Eigenschaften**, Karteikarte **Berechnen**.

RoundIntermediateResults (Eigenschaft)

Datentyp: **Boolean**

Liest oder setzt die Einstellung "Zwischenergebnisse runden" im Dialog **Datei > Eigenschaften**, Karteikarte **Berechnen**.

ActiveSheet (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das gerade aktive **Sheet**-Objekt, über das Sie auf das aktuelle Arbeitsblatt zugreifen können.

ActiveWindow (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das gerade aktive **Window**-Objekt, über das Sie auf das aktuelle Dokumentfenster zugreifen können.

BuiltInDocumentProperties (Zeiger auf Objekt)

Datentyp: **Object**

Liefert die **DocumentProperties**-Sammlung, die Sie auf die Dokumentinfos (Titel, Thema, Autor etc.) des Dokuments zugreifen lässt.

Application (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das **Application**-Objekt.

Parent (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das übergeordnete Objekt, also **Workbooks**.

Sheets (Zeiger auf Sammlung

Datentyp: **Object**

Liefert die **Sheets**-Sammlung, eine Sammlung aller Arbeitsblätter des Dokuments.

Activate (Methode)

Bringt das Dokumentfenster in den Vordergrund (sofern **Visible** für das Dokument True ist) und setzt den Fokus auf das Dokumentfenster.

Syntax:

```
Activate
```

Parameter:

keine

Rückgabetyp:

keiner

Beispiel:

```
' Das erste Dokument der Workbooks-Sammlung in den Vordergrund bringen
pm.Workbooks(1).Activate
```

Calculate (Methode)

Berechnet das Dokument neu (entspricht dem Befehl **Weiteres > Neu berechnen** in PlanMaker).

Syntax:

```
Calculate
```

Parameter:

keine

Rückgabetyp:

keiner

Beispiel:

```
' Das erste Dokument der Workbooks-Sammlung neu berechnen
pm.Workbooks(1).Calculate
```

Close (Methode)

Schließt das Dokument.

Syntax:

```
Close [SaveChanges]
```

Parameter:

SaveChanges (optional; Typ: **Long** bzw. **SmoSaveOptions**) gibt an, ob das Dokument gespeichert werden soll oder nicht. Lassen Sie den Parameter weg, wird stattdessen der Benutzer gefragt – jedoch nur dann, wenn das Dokument seit der letzten Speicherung verändert wurde. Mögliche Werte für **SaveChanges**:

```
smoDoNotSaveChanges = 0       ' Nicht fragen, nicht speichern
smoPromptToSaveChanges = 1    ' Den Benutzer fragen
smoSaveChanges = 2            ' Ohne Rückfrage speichern
```

Rückgabetyp:

keiner

Beispiel:

```
' Das aktive Dokument schließen, ohne es zu speichern
pm.ActiveWorkbook.Close smoDoNotSaveChanges
```

Save (Methode)

Speichert das Dokument.

Syntax:

```
Save
```

Parameter:

keine

Rückgabetyp:

keiner

Beispiel:

```
' Das aktive Dokument speichern
pm.ActiveWorkbook.Save
```

SaveAs (Methode)

Speichert das Dokument unter einem anderen Namen und/oder in einem anderen Pfad.

Syntax:

```
SaveAs FileName, [FileFormat], [Delimiter], [TextMarker]
```

Parameter:

FileName (Typ: **String**): Pfad und Dateiname, unter dem das Dokument gespeichert werden soll

FileFormat (optional; Typ: **Long** bzw. **PmSaveFormat**) bestimmt das Dateiformat. Dieser Parameter kann folgende Werte annehmen (links die symbolische Konstante, rechts der entsprechende numerische Wert):

```
pmFormatDocument          =  0 ' Dokument, ist Default
pmFormatTemplate          =  1 ' Dokumentvorlage
pmFormatExcel97           =  2 ' Excel 97/2000/XP
pmFormatExcel5            =  3 ' Excel 5.0/7.0
pmFormatExcelTemplate     =  4 ' Excel-Vorlage
pmFormatSYLK              =  6 ' Sylk
pmFormatRTF               =  7 ' Rich Text Format
pmFormatTextMaker         =  7 ' TextMaker (= RTF)
pmFormatHTML              =  8 ' HTML
pmFormatdBaseDOS          =  9 ' dBASE-Datenbank im DOS-Zeichensatz
pmFormatdBaseAnsi         = 10 ' dBASE-Datenbank im Windows-Zeichensatz
pmFormatDIF               = 11 ' Textdatei mit Windows-Zeichensatz
pmFormatPlainTextAnsi     = 12 ' Textdatei mit Windows-Zeichensatz
pmFormatPlainTextDOS      = 13 ' Textdatei mit DOS-Zeichensatz
pmFormatPlainTextUnix     = 14 ' Textdatei mit ANSI-Zeichensatz für UNIX, Linux und
FreeBSD
pmFormatPlainTextUnicode  = 15 ' Textdatei mit Unicode-Zeichensatz
pmFormatdBaseUnicode      = 18 ' dBASE-Datenbank mit Unicode-Zeichensatz
pmFormatPlainTextUTF8     = 20 ' Textdatei mit UTF8-Zeichensatz
```

Wenn Sie diesen Parameter weglassen, wird **pmFormatDocument** angenommen.

Delimiter (optional; Typ: **String**): Gibt bei den Textdatei-Formaten das Trennzeichen an, zum Beispiel Komma oder Strichpunkt. Wenn Sie den Parameter weglassen, wird der Tabulator als Trennzeichen verwendet.

TextMarker (optional; Typ: **Long** bzw. **PmImportTextMarker**): Gibt bei den Textdatei-Formaten an, mit welchem Zeichen Textfelder umgeben werden sollen. Mögliche Werte:

```
pmImportTextMarkerNone          = 0 ' Text wird nicht speziell markiert
pmImportTextMarkerApostrophe    = 1 ' Apostrophe
pmImportTextMarkerQmark         = 2 ' Anführungszeichen
```

Rückgabetyp:

keiner

Beispiel:

```
' Das aktuelle Dokument unter neuem Namen im Excel 97-Format speichern
pm.ActiveWorkbook.SaveAs "c:\doks\test.xls", pmFormatExcel97
```

PrintOut (Methode)

Druckt das Dokument aus.

Syntax:

```
PrintOut [From], [To]
```

Parameter:

From (optional; Typ: **Long**) gibt an, ab welcher Seite gedruckt werden soll. Lassen Sie diesen Parameter weg, wird ab der ersten Seite gedruckt.

To (optional; Typ: **Long**) gibt an, bis zu welcher Seite gedruckt werden soll. Lassen Sie diesen Parameter weg, wird bis zur letzten Seite gedruckt.

Rückgabetyp:

keiner

Beispiel:

```
' Das aktuelle Dokument ausdrucken
pm.ActiveWorkbook.PrintOut
```

DocumentProperties (Sammlung)

Zugriffspfade:
- Application → Workbooks → Item → **DocumentProperties**
- Application → ActiveWorkbook → **DocumentProperties**

1 Beschreibung

Die Sammlung **DocumentProperties** enthält alle Dokumenteigenschaften eines Dokuments. Dazu gehören zum Beispiel der Titel, der Autor, die Anzahl der mit Inhalt gefüllten Zellen usw.

Die einzelnen Elemente dieser Sammlung sind vom Typ **DocumentProperty**.

2 Zugriff auf die Sammlung

Jedes geöffnete Dokument besitzt genau eine **DocumentProperties**-Sammlung. Diese wird über **Workbook.BuiltInDocumentProperties** angesprochen:

```
' Den Titel des aktiven Dokuments auf "Meine Kalkulation" setzen
pm.ActiveWorkbook.BuiltInDocumentProperties(smoPropertyTitle) = "Meine Kalkulation"

' Die Anzahl der Diagramme im aktiven Dokument ausgeben
MsgBox pm.ActiveWorkbook.BuiltInDocumentProperties("Number of charts")
```

3 Eigenschaften, Objekte, Sammlungen und Methoden

Eigenschaften:
- **Count** R/O

Objekte:
- ***Item*** → **DocumentProperty** (Defaultobjekt)
- **Application** → **Application**
- **Parent** → **Workbook**

Count (Eigenschaft, R/O)

Datentyp: **Long**

Liefert die Anzahl der **DocumentProperty**-Objekte in der Sammlung, also die Zahl der Dokumenteigenschaften eines Dokuments. Der Wert ist unveränderlich, da alle PlanMaker-Dokumente dieselbe Zahl von Dokumenteigenschaften besitzen.

Item (Zeiger auf Objekt)

Datentyp: **Object**

Liefert ein einzelnes **DocumentProperty**-Objekt, also eine einzelne Dokumenteigenschaft.

Welches DocumentProperty-Objekt Sie erhalten, hängt von dem Parameter ab, den Sie an **Item** übergeben. Dies kann entweder der numerische Index oder der Name der gewünschten Dokumenteigenschaft sein.

Die folgende Tabelle enthält sowohl die erlaubten Zahlenwerte als auch die zugehörigen Namen:

```
smoPropertyTitle                =  1 ' "Title"
smoPropertySubject              =  2 ' "Subject"
smoPropertyAuthor               =  3 ' "Author"
smoPropertyKeywords             =  4 ' "Keywords"
smoPropertyComments             =  5 ' "Comments"
smoPropertyAppName              =  6 ' "Application name"
smoPropertyTimeLastPrinted      =  7 ' "Last print date"
smoPropertyTimeCreated          =  8 ' "Creation date"
smoPropertyTimeLastSaved        =  9 ' "Last save time"
smoPropertyKeystrokes           = 10 ' - (bei PlanMaker nicht verfügbar)
smoPropertyCharacters           = 11 ' - (bei PlanMaker nicht verfügbar)
smoPropertyWords                = 12 ' - (bei PlanMaker nicht verfügbar)
smoPropertySentences            = 13 ' - (bei PlanMaker nicht verfügbar)
smoPropertyParas                = 14 ' - (bei PlanMaker nicht verfügbar)
smoPropertyChapters             = 15 ' - (bei PlanMaker nicht verfügbar)
smoPropertySections             = 16 ' - (bei PlanMaker nicht verfügbar)
smoPropertyLines                = 17 ' - (bei PlanMaker nicht verfügbar)
smoPropertyPages                = 18 ' "Number of pages"
smoPropertyCells                = 19 ' "Number of cells"
smoPropertyTextCells            = 20 ' "Number of cells with text"
smoPropertyNumericCells         = 21 ' "Number of cells with numbers"
smoPropertyFormulaCells         = 22 ' "Number of cells with formulas"
smoPropertyNotes                = 23 ' "Number of comments"
smoPropertySheets               = 24 ' "Number of worksheets"
smoPropertyCharts               = 25 ' "Number of charts"
smoPropertyPictures             = 26 ' "Number of pictures"
smoPropertyOLEObjects           = 27 ' "Number of OLE objects"
smoPropertyDrawings             = 28 ' "Number of drawings"
smoPropertyTextFrames           = 29 ' "Number of text frames"
smoPropertyTables               = 30 ' - (bei PlanMaker nicht verfügbar)
smoPropertyFootnotes            = 31 ' - (bei PlanMaker nicht verfügbar)
smoPropertyAvgWordLength        = 32 ' - (bei PlanMaker nicht verfügbar)
smoPropertyAvgCharactersSentence = 33 ' - (bei PlanMaker nicht verfügbar)
smoPropertyAvgWordsSentence     = 34 ' - (bei PlanMaker nicht verfügbar)
```

Diese Liste führt *alle* Dokumenteigenschaften auf, die in SoftMaker Office verfügbar sind, auch solche, die es bei PlanMaker nicht gibt. Diese sind mit "bei PlanMaker nicht verfügbar" gekennzeichnet.

Application (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das **Application**-Objekt.

Parent (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das übergeordnete Objekt, also ein Objekt vom Typ **Workbook**.

DocumentProperty (Objekt)

Zugriffspfade:
- Application → Workbooks → Item → BuiltInDocumentProperties → **Item**
- Application → ActiveWorkbook → BuiltInDocumentProperties → **Item**

1 Beschreibung

Ein **DocumentProperty**-Objekt repräsentiert eine einzelne Dokumenteigenschaft eines Dokuments, etwa den Titel, den Autor oder die Zahl der mit Inhalt gefüllten Zellen.

2 Zugriff auf das Objekt

Die einzelnen **DocumentProperty**-Objekte können ausschließlich durch Aufzählung der Elemente von Sammlungen des Typs **DocumentProperties** angesprochen werden.

Für jedes geöffnete Dokument existiert genau eine Instanz dieser **DocumentProperties**-Sammlung, nämlich **BuiltInDocumentProperties** im Workbook-Objekt:

```
' Den Titel des aktiven Dokuments auf "Meine Kalkulation" setzen
pm.ActiveWorkbook.BuiltInDocumentProperties.Item(smoPropertyTitle) = "Meine Kalkulation"
```

3 Eigenschaften, Objekte, Sammlungen und Methoden

Eigenschaften:
- **Name** R/O
- ***Value*** (Defaulteigenschaft)
- **Valid**
- **Type**

Objekte:
- **Application → Application**
- **Parent → BuiltInDocumentProperties**

Name (Eigenschaft, R/O)

Datentyp: **String**

Liefert den Namen der Dokumenteigenschaft. Beispiele:

```
' Den Namen der Dokumenteigenschaft smoPropertyTitle ausgeben, also "Title"
MsgBox pm.ActiveWorkbook.BuiltInDocumentProperties.Item(smoPropertyTitle).Name

' Den Namen der Dokumenteigenschaft "Author" ausgeben, also "Author"
MsgBox pm.ActiveWorkbook.BuiltInDocumentProperties.Item("Author").Name
```

Value (Eigenschaft)

Datentyp: **String**

Liest oder setzt den Inhalt der Dokumenteigenschaft.

Das folgende Beispiel setzt die Dokumenteigenschaft "Titel" über die numerische Konstante **smoPropertyTitle** und liest sie gleich wieder über die Stringkonstante "Title" aus:

```
Sub Beispiel()
  Dim pm as Object

  Set pm = CreateObject("PlanMaker.Application")
  pm.Workbooks.Add ' Neues leeres Dokument hinzufügen

  With pm.ActiveWorkbook

    ' Neuen Titel setzen (mit Hilfe der numerischen Konstante smoPropertyTitle)
    .BuiltInDocumentProperties.Item(smoPropertyTitle).Value = "Neuer Titel"

    ' Genau diese Eigenschaft wieder auslesen (diesmal über den String)
    MsgBox .BuiltInDocumentProperties.Item("Title").Value

  End With
End Sub
```

Da **Item** das Defaultobjekt von **DocumentProperties** ist und **Value** die Defaulteigenschaft von **DocumentProperty**, lässt sich dieses Beispiel übersichtlicher wie folgt schreiben:

```
Sub Beispiel()
  Dim pm as Object

  Set pm = CreateObject("PlanMaker.Application")
  pm.Workbooks.Add ' Neues leeres Dokument hinzufügen

  With pm.ActiveWorkbook

    ' Neuen Titel setzen (mit Hilfe der numerischen Konstante smoPropertyTitle)
    .BuiltInDocumentProperties(smoPropertyTitle) = "Neuer Titel"

    ' Genau diese Eigenschaft wieder auslesen (diesmal über den String)
    MsgBox .BuiltInDocumentProperties("Title")

  End With
End Sub
```

Valid (Eigenschaft, R/O)

Datentyp: **Boolean**

Liefert **True** zurück, wenn die Dokumenteigenschaft bei PlanMaker verfügbar ist.

Hintergrund: Die Liste der möglichen Dokumenteigenschaften enthält auch solche, die nur bei TextMaker verfügbar sind (zum Beispiel **smoPropertyChapters**, "Number of chapters"). Sie dürfen bei PlanMaker nur diejenigen Dokumenteigenschaften abfragen, die PlanMaker kennt – sonst wird ein Leerwert zurückgegeben (VT_EMPTY).

Die Eigenschaft **Valid** lässt Sie vor der Abfrage prüfen, ob die jeweilige Dokumenteigenschaft bei PlanMaker vorhanden ist. Beispiel:

```
Sub Test
  Dim pm as Object
  Dim i as Integer

  Set pm = CreateObject("PlanMaker.Application")

  pm.Visible = True
  pm.Workbooks.Add ' leeres Dokument hinzufügen

  With pm.ActiveWorkbook
   For i = 1 to .BuiltInDocumentProperties.Count
```

```
    If .BuiltInDocumentProperties(i).Valid then
       print i, .BuiltInDocumentProperties(i).Name, "=", _
          .BuiltInDocumentProperties(i).Value
    Else
       print i, "Nicht bei PlanMaker verfügbar"
    End If
   Next i
  End With

End Sub
```

Type (Eigenschaft, R/O)

Datentyp: **Long** (SmoDocProperties)

Liefert den Datentyp der Dokumenteigenschaft. Damit Sie eine Dokumenteigenschaft richtig auswerten können, müssen Sie ihren Typ wissen. Beispielsweise ist **Title** (smoPropertyTitle) ein String, **Creation Date** (smoPropertyTimeCreated) hingegen ein Datum. Mögliche Werte:

```
smoPropertyTypeBoolean   = 0 ' Boolean
smoPropertyTypeDate      = 1 ' Datum
smoPropertyTypeFloat     = 2 ' Fließkommawert
smoPropertyTypeNumber    = 3 ' Ganzzahl
smoPropertyTypeString    = 4 ' Zeichenkette
```

Application (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das **Application**-Objekt.

Parent (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das übergeordnete Objekt, also **BuiltInDocumentProperties**.

Sheets (Sammlung)

Zugriffspfade:
- Application → Workbooks → Item → **Sheets**
- Application → ActiveWorkbook → **Sheets**

1 Beschreibung

Die Sammlung **Sheets** enthält alle Arbeitsblätter (*sheet* = englisch für "Blatt") eines Dokuments. Die einzelnen Elemente dieser Sammlung sind vom Typ **Sheet**.

2 Zugriff auf die Sammlung

Jedes geöffnete Dokument besitzt genau eine Instanz der **Sheets**-Sammlung. Diese wird über **Workbook.Sheets** angesprochen:

```
' Anzahl der Arbeitsblätter des aktiven Dokuments anzeigen
MsgBox pm.ActiveWorkbook.Sheets.Count
```

Eigenschaften:
- **Count** R/O

Objekte:
- ***Item*** → **Sheet**
- **Application** → **Application**
- **Parent** → **Workbook**

Methoden:
- **Add**

Count (Eigenschaft, R/O)

Datentyp: **Long**

Liefert die Anzahl der **Sheet**-Objekte des Dokuments – in anderen Worten: die Anzahl der Arbeitsblätter des Dokuments.

Item (Zeiger auf Objekt)

Datentyp: **Object**

Liefert ein einzelnes **Sheet**-Objekt, also ein einzelnes Arbeitsblatt.

Welches Sheet-Objekt Sie erhalten, hängt von dem Parameter ab, den Sie an **Item** übergeben. Dies kann entweder der numerische Index oder der Name des gewünschten Arbeitsblatts sein:

```
' Zeige den Namen des ersten Arbeitsblatts
MsgBox pm.Application.ActiveWorkbook.Sheets.Item(1).Name

' Zeige den Namen des Arbeitsblatts mit dem Namen "Einnahmen"
MsgBox pm.Application.ActiveWorkbook.Sheets.Item("Einnahmen").Name
```

Application (Zeiger auf Objekt)

Liefert das **Application**-Objekt.

Parent (Zeiger auf Objekt)

Liefert das übergeordnete Objekt, also ein Objekt vom Typ **Workbook**.

Add (Methode)

Fügt dem Dokument ein neues leeres Arbeitsblatt hinzu und gibt das **Sheet**-Objekt zurück, das das neue Arbeitsblatt repräsentiert.

Syntax:

```
Add [Name]
```

Parameter:

Name (optional; Typ: **String**): Name für das neue Arbeitsblatt. Lassen Sie den Parameter weg, wird der Name automatisch generiert ("Tabelle1", "Tabelle2", "Tabelle3" etc.)

Rückgabetyp:

Object

Beispiel:

```
Sub Sample()
  Dim pm as Object
  Dim newDoc as Object
  Dim newSheet as Object

  Set pm = CreateObject("PlanMaker.Application")
  pm.Visible = True

  ' Ein Dokument hinzufügen
  Set newDoc = pm.Workbooks.Add

  ' Dem Dokument ein Arbeitsblatt hinzufügen
  Set newSheet = newDoc.Sheets.Add("MySheet")

  ' Namen des neuen Arbeitsblatts anzeigen
  MsgBox newSheet.Name
End Sub
```

Mit dem von **Add** zurückgegebenen **Sheet** können Sie arbeiten wie mit jedem anderen Arbeitsblatt. Sie können aber auch den Rückgabewert von **Add** ignorieren und sich das neue Arbeitsblatt beispielsweise über **ActiveSheet** holen.

Sheet (Objekt)

Zugriffspfade:
- Application → Workbooks → Item → Sheets → **Item**
- Application → Workbooks → **ActiveSheet**
- Application → ActiveWorkbook → **ActiveSheet**
- Application → **ActiveSheet**

1 Beschreibung

Ein **Sheet**-Objekt repräsentiert ein einzelnes Arbeitsblatt eines in PlanMaker geöffneten Dokuments.

Für jedes Arbeitsblatt eines Dokuments existiert ein eigenes **Sheet**-Objekt. Fügen Sie dem Dokument Arbeitsblätter hinzu oder löschen diese, werden die zugehörigen **Sheet**-Objekte dynamisch angelegt bzw. entfernt.

2 Zugriff auf das Objekt

Die einzelnen **Sheet**-Objekte können auf folgenden Wegen angesprochen werden:

- Alle zu einem Dokument gehörenden Arbeitsblätter werden in der Sammlung **Workbook.Sheets** (Typ: **Sheets**) verwaltet:

```
' Die Namen aller Arbeitsblätter des aktiven Dokuments anzeigen
For i = 1 To pm.Application.ActiveWorkbook.Sheets.Count
  MsgBox pm.Application.ActiveWorkbook.Sheets.Item(i).Name
Next i
```

- Das aktive Arbeitsblatt eines Dokuments erhalten Sie über **Workbook.ActiveSheet**:

```
' Den Namen des aktuellen Arbeitsblatts anzeigen
MsgBox pm.Application.Workbooks(1).ActiveSheet.Name
```

- Das aktive Arbeitsblatt des aktiven Dokuments erhalten Sie über **Application.ActiveSheet**:

```
' Den Namen des aktuellen Arbeitsblatts des aktiven Dokuments anzeigen
MsgBox pm.Application.ActiveSheet.Name
```

- **Sheet** ist der **Parent** diverser Objekte, die daran angebunden sind, zum Beispiel **Range** oder **AutoFilter**:

```
' Den Namen des aktuellen Arbeitsblatts über einen Umweg anzeigen
MsgBox pm.Application.ActiveSheet.Range("A1:B20").Parent.Name
```

3 Eigenschaften, Objekte, Sammlungen und Methoden

Eigenschaften:
- ***Name*** (Defaulteigenschaft)
- **Index** R/O
- **Hidden**
- **PageBreaks**
- **DisplayRowHeadings**
- **DisplayColumnHeadings**
- **AutoFilterMode**

Objekte:
- **PageSetup → PageSetup**
- **Selection → Range**
- **Rows → Rows**
- **Columns → Columns**
- **Cells → Range**
- **Range → Range**
- **AutoFilter → AutoFilter**
- **Application → Application**
- **Parent → Sheets**

Methoden:
- **Activate**
- **Calculate**
- **Delete**
- **Move**
- **Select**
- **ShowAllData**

Name (Eigenschaft)

Datentyp: **String**

Liest oder setzt den Namen des Arbeitsblatts.

Index (Eigenschaft, R/O)

Datentyp: **Long**

Liefert die Position des Arbeitsblatts innerhalb der anderen Arbeitsblätter (siehe auch **Move**).

Hidden (Eigenschaft)

Datentyp: **Boolean**

Liest oder setzt die Einstellung, ob das Arbeitsblatt verborgen ist. Entspricht den Befehlen **Tabelle > Blatt > Einblenden** und **Ausblenden** in PlanMaker.

PageBreaks (Eigenschaft)

Datentyp: **Boolean**

Liest oder setzt die Einstellung, ob im Arbeitsblatt Seitenumbrüche angezeigt werden. Entspricht dem Befehl **Tabelle > Eigenschaften > Seitenumbrüche** in PlanMaker.

DisplayRowHeadings (Eigenschaft)

Datentyp: **Boolean**

Liest oder setzt die Einstellung, ob im Arbeitsblatt Zeilenköpfe angezeigt werden. Entspricht der Einstellung **Zeilenköpfe** im Dialogfenster des Befehls **Tabelle > Eigenschaften > Ansicht**.

DisplayColumnHeadings (Eigenschaft)

Datentyp: **Boolean**

Liest oder setzt die Einstellung, ob im Arbeitsblatt Spaltenköpfe angezeigt werden. Entspricht der Einstellung **Spaltenköpfe** im Dialogfenster des Befehls **Tabelle > Eigenschaften > Ansicht**.

DisplayGridlines (Eigenschaft)

Datentyp: **Boolean**

Liest oder setzt die Einstellung, ob im Arbeitsblatt Gitternetzlinien zwischen den Zellen angezeigt werden. Entspricht der Einstellung **Gitternetzlinien** im Dialogfenster des Befehls **Tabelle > Eigenschaften**.

GridlineColor (Eigenschaft)

Datentyp: **Long** (SmoColor)

Liest oder setzt die Farbe der Gitternetzlinien als "BGR"-Wert (Blau-Grün-Rot-Triplet). Sie können entweder einen beliebigen Wert angeben oder eine der vordefinierten BGR-Farbkonstanten verwenden.

GridlineColorIndex (Eigenschaft)

Datentyp: **Long** (SmoColorIndex)

Liest oder setzt die Farbe der Gitternetzlinien als Indexfarbe. "Indexfarben" sind die Standardfarben von PlanMaker, durchnummeriert von -1 für Automatisch bis 15 für Hellgrau. Sie dürfen ausschließlich einen der in der Tabelle der Indexfarben genannten Werte verwenden.

Wir empfehlen, stattdessen die Eigenschaft **GridlineColor** (siehe dort) zu verwenden, da Sie mit dieser nicht auf die Standardfarben beschränkt sind, sondern mit beliebigen Farben der BGR-Palette arbeiten können.

AutoFilterMode (Eigenschaft)

Liest oder setzt die Einstellung, ob Dropdown-Pfeile bei dem aktiven AutoFilter angezeigt werden.

Lesen können Sie diese Einstellung jederzeit. Beim *Setzen* ist jedoch zu beachten, dass Sie sie lediglich auf **False** setzen können, um die Dropdown-Pfeile zu *verbergen*. Um Dropdown-Pfeile *anzuzeigen*, müssen Sie stattdessen die Methode **AutoFilter** im **Range**-Objekt aufrufen.

PageSetup (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das **PageSetup**-Objekt, das Sie auf die Seitenformatierung (Papierformat, Ränder etc.) des Arbeitsblatts zugreifen lässt.

Selection (Zeiger auf Objekt)

Datentyp: **Object**

Liefert ein **Range**-Objekt, das die gerade selektierten (markierten) Zellen des Arbeitsblatts repräsentiert. Sie können damit unter anderem deren Inhalte und Formatierungen auslesen und verändern.

Wenn im Arbeitsblatt nichts selektiert ist, repräsentiert das **Range**-Objekt diejenige Zelle, die den Zellrahmen enthält.

Rows (Zeiger auf Objekt)

Datentyp: **Object**

Liefert die **Rows**-Sammlung, eine Sammlung aller Zeilen des Arbeitsblatts.

Die einzelnen Elemente dieser Sammlung sind **Range**-Objekte, Sie können daher auf diese alle Eigenschaften und Methoden von Bereichen anwenden.

Beispiel:

```
' Alle Zellen in Zeile 10 auf die Schriftart Courier New setzen
pm.ActiveSheet.Rows(10).Font.Name = "Courier New"
```

Columns (Zeiger auf Objekt)

Datentyp: **Object**

Liefert die **Columns**-Sammlung, eine Sammlung aller Spalten des Arbeitsblatts.

Die einzelnen Elemente dieser Sammlung sind **Range**-Objekte, Sie können daher auf diese alle Eigenschaften und Methoden von Bereichen anwenden.

Beispiel:

```
' Alle Zellen in Spalte C (= dritte Spalte) auf Courier New setzen
pm.ActiveSheet.Columns(3).Font.Name = "Courier New"
```

Cells (Zeiger auf Objekt)

Datentyp: **Object**

Liefert ein **Range**-Objekt, das alle Zellen im Arbeitsblatt umfasst. Das ist für zwei Anwendungsfälle nützlich:

- Sie können eine Operation (vorrangig Formatierungen) auf jede Zelle des Arbeitsblatts anwenden:

```
' Das gesamte Arbeitsblatt rot einfärben
pm.ActiveSheet.Cells.Shading.ForegroundPatternColor = smoColorRed
```

- Sie können einzelne Zellen über Schleifenvariablen adressieren, anstatt den Adressierungsstring (zum Beispiel "B5" für die zweite Spalte in der fünften Zeile) manuell zusammenzubauen. Hierzu benutzen Sie die Eigenschaft **Item** des durch den **Cells**-Zeiger adressierten **Range**-Objekts:

```
' Die ersten 5 * 10 Zellen des Arbeitsblatts befüllen
Dim row, col as Integer
For row = 1 To 5
  For col = 1 to 10
    pm.ActiveSheet.Cells.Item(row, col).Value = 42
  Next col
```

```
Next row
```

Range (Zeiger auf Objekt)

Datentyp: **Object**

Liefert ein zu den übergebenen Parametern passendes **Range**-Objekt. Sie können mit diesem Objekt auf die Zellen im Bereich zugreifen und beispielsweise Werte auslesen und setzen.

Syntax 1:

```
obj = Range(Cell1)
```

Syntax 2:

```
obj = Range(Cell1, Cell2)
```

Parameter:

Cell1 (Typ: **String**) gibt entweder gemäß Syntax 1 einen Zellbereich an (dann muss **Cell2** weggelassen werden) oder gemäß Syntax 2 die linke obere Ecke eines Bereichs (dann gibt der Parameter **Cell2** die rechte untere Ecke des Bereichs an).

Cell2 (optional; Typ: **String**) darf nur verwendet werden, wenn **Cell1** eine einzelne Zelle referenziert, und gibt die rechte untere Ecke des Bereichs an).

Beispiele für Syntax 1:

```
Range("A1:B20")       ' Zellen A1 bis B20
Range("A1")           ' Nur Zelle A1
Range("A:A")          ' Gesamte Spalte A
Range("3:3")          ' Gesamte Zeile 3
Range("Sommer")       ' Benannter Bereich "Sommer"
```

Beispiel für Syntax 2:

```
Range("A1", "B20")    ' Zellen A1 bis B20
```

Beispiel:

```
' Zellen A1 bis B20 des aktuellen Arbeitsblatts selektieren
pm.ActiveSheet.Range("A1:B20").Select
```

AutoFilter (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das **AutoFilter**-Objekt, das Sie auf den AutoFilter des Arbeitsblatts zugreifen lässt.

Application (Zeiger auf Objekt)

Liefert das **Application**-Objekt.

Parent (Zeiger auf Objekt)

Liefert das übergeordnete Objekt, also **Sheets**.

Activate (Methode)

Macht das Arbeitsblatt zum aktuellen Arbeitsblatt.

Syntax:

```
Activate
```

Parameter:

keine

Rückgabetyp:

keiner

Beispiel:

```
' Das erste Blatt des aktuellen Dokuments in den Vordergrund bringen
pm.ActiveWorkbook.Sheets(1).Activate
```

Calculate (Methode)

Berechnet das Arbeitsblatt neu (ähnlich dem Menübefehl **Weiteres > Neu berechnen** in PlanMaker, nur dass der Menübefehl *alle* Arbeitsblätter einer Arbeitsmappe neu berechnet).

Syntax:

```
Calculate
```

Parameter:

keine

Rückgabetyp:

keiner

Beispiel:

```
' Das erste Arbeitsblatt neu berechnen
pm.ActiveWorkbook.Sheets(1).Calculate
```

Delete (Methode)

Löscht das Arbeitsblatt aus dem Dokument.

Syntax:

```
Delete
```

Parameter:

keine

Rückgabetyp:

keiner

Beispiel:

```
' Das erste Blatt des aktuellen Dokuments löschen
pm.ActiveWorkbook.Sheets(1).Delete
```

Move (Methode)

Verschiebt die Position des Arbeitsblatts innerhalb der anderen Arbeitsblätter.

Syntax:

```
Move Index
```

Parameter:

Index (Typ: **Long**) gibt die Zielposition an.

Rückgabetyp:

keiner

Beispiel:

```
' Das aktuelle Arbeitsblatt an die dritte Position verschieben
pm.ActiveSheet.Move 3
```

Select (Methode)

Selektiert alle Zellen des Arbeitsblatts (entspricht dem Befehl **Bearbeiten > Alles markieren** in PlanMaker).

Syntax:

Select

Parameter:

keine

Rückgabetyp:

keiner

Beispiel:

```
' Alle Zellen im aktuellen Arbeitsblatt selektieren
pm.ActiveWorkbook.Select
```

ShowAllData (Methode)

Hebt die durch einen aktiven AutoFilter verursachte Filterung auf und blendet alle ausgeblendeten Zeilen wieder ein. Entspricht der Filterung "(Alle)" im Menü, das erscheint, wenn Sie auf den Dropdown-Pfeil des AutoFilters klicken.

PageSetup (Objekt)

Zugriffspfade:
- Application → Workbooks → Item → Sheets → Item → **PageSetup**
- Application → Workbooks → ActiveSheet → **PageSetup**
- Application → ActiveWorkbook → ActiveSheet → **PageSetup**
- Application → ActiveSheet → **PageSetup**

1 Beschreibung

Das **PageSetup**-Objekt enthält die Seiteneinstellungen des **Sheet**-Objekts, zu dem es gehört. Sie können damit das Papierformat, Seitengröße und -ränder sowie die Druckrichtung eines einzelnen Arbeitsblatts ermitteln und verändern.

2 Zugriff auf das Objekt

Jedes Arbeitsblatt eines Dokuments besitzt genau eine Instanz des **PageSetup**-Objekts. Diese wird über **Sheet.PageSetup** angesprochen:

```
' Den linken Blattrand auf 2 cm setzen
pm.ActiveSheet.PageSetup.LeftMargin = pm.CentimetersToPoints(2)
```

Hinweis: Sie können für die einzelnen Arbeitsblätter eines Dokuments unterschiedliche Seiteneinstellungen vergeben.

3 Eigenschaften, Objekte, Sammlungen und Methoden

Eigenschaften:

- **LeftMargin**
- **RightMargin**
- **TopMargin**
- **BottomMargin**
- **HeaderMargin**
- **FooterMargin**
- **PageHeight**
- **PageWidth**
- **Orientation**
- **PaperSize**
- **PrintComments**
- **CenterHorizontally**
- **CenterVertically**
- **Zoom**
- **FirstPageNumber**
- **PrintGridlines**
- **PrintHeadings**
- **Order**
- **PrintArea**
- **PrintTitleRows**
- **PrintTitleColumns**

Objekte:

- **Application → Application**
- **Parent → Sheet**

LeftMargin (Eigenschaft)

Datentyp: **Single**

Liest oder setzt den linken Seitenrand des Arbeitsblatts in Punkt (1 Punkt entspricht 1/72 Zoll).

RightMargin (Eigenschaft)

Datentyp: **Single**

Liest oder setzt den rechten Seitenrand des Arbeitsblatts in Punkt (1 Punkt entspricht 1/72 Zoll).

TopMargin (Eigenschaft)

Datentyp: **Single**

Liest oder setzt den oberen Seitenrand des Arbeitsblatts in Punkt (1 Punkt entspricht 1/72 Zoll).

BottomMargin (Eigenschaft)

Datentyp: **Single**

Liest oder setzt den unteren Seitenrand des Arbeitsblatts in Punkt (1 Punkt entspricht 1/72 Zoll).

HeaderMargin (Eigenschaft)

Datentyp: **Single**

Liest oder setzt den Abstand der Kopfzeile zur oberen Blattkante in Punkt (1 Punkt entspricht 1/72 Zoll).

FooterMargin (Eigenschaft)

Datentyp: **Single**

Liest oder setzt den Abstand der Fußzeile zur unteren Blattkante in Punkt (1 Punkt entspricht 1/72 Zoll).

PageHeight (Eigenschaft)

Datentyp: **Single**

Liest oder setzt die Seitenhöhe des Arbeitsblatts in Punkt (1 Punkt entspricht 1/72 Zoll).

Wenn Sie diese Eigenschaft setzen, ändert sich die **PaperSize**-Eigenschaft (siehe unten) automatisch auf das passende Papierformat.

PageWidth (Eigenschaft)

Datentyp: **Single**

Liest oder setzt die Seitenbreite des Arbeitsblatts in Punkt (1 Punkt entspricht 1/72 Zoll).

Wenn Sie diese Eigenschaft setzen, ändert sich die **PaperSize**-Eigenschaft (siehe unten) automatisch auf das passende Papierformat.

Orientation (Eigenschaft)

Datentyp: **Long** (SmoOrientation)

Liest oder setzt die Ausrichtung des Arbeitsblatts. Folgende Konstanten sind erlaubt:

```
smoOrientLandscape  = 0 ' Querformat
smoOrientPortrait   = 1 ' Hochformat
```

PaperSize (Eigenschaft)

Datentyp: **Long** (SmoPaperSize)

Liest oder setzt die Papiergröße des Arbeitsblatts. Folgende Konstanten sind erlaubt:

```
smoPaperCustom                = -1
smoPaperLetter                = 1
smoPaperLetterSmall           = 2
smoPaperTabloid               = 3
smoPaperLedger                = 4
smoPaperLegal                 = 5
smoPaperStatement             = 6
smoPaperExecutive             = 7
smoPaperA3                    = 8
smoPaperA4                    = 9
smoPaperA4Small               = 10
smoPaperA5                    = 11
smoPaperB4                    = 12
smoPaperB5                    = 13
smoPaperFolio                 = 14
```

```
smoPaperQuarto                = 15
smoPaper10x14                 = 16
smoPaper11x17                 = 17
smoPaperNote                  = 18
smoPaperEnvelope9             = 19
smoPaperEnvelope10            = 20
smoPaperEnvelope11            = 21
smoPaperEnvelope12            = 22
smoPaperEnvelope14            = 23
smoPaperCSheet                = 24
smoPaperDSheet                = 25
smoPaperESheet                = 26
smoPaperEnvelopeDL            = 27
smoPaperEnvelopeC5            = 28
smoPaperEnvelopeC3            = 29
smoPaperEnvelopeC4            = 30
smoPaperEnvelopeC6            = 31
smoPaperEnvelopeC65           = 32
smoPaperEnvelopeB4            = 33
smoPaperEnvelopeB5            = 34
smoPaperEnvelopeB6            = 35
smoPaperEnvelopeItaly         = 36
smoPaperEnvelopeMonarch       = 37
smoPaperEnvelopePersonal      = 38
smoPaperFanfoldUS             = 39
smoPaperFanfoldStdGerman      = 40
smoPaperFanfoldLegalGerman    = 41
```

PrintComments

Datentyp: **Long** (PmPrintLocation)

Liest oder setzt die Einstellung, ob im Arbeitsblatt enthaltene Kommentare auch ausgedruckt werden. Entspricht der Einstellung "Kommentare" im Dialogfenster des Befehls **Datei > Seite einrichten > Optionen**. Folgende Konstanten sind erlaubt:

```
pmPrintNoComments    = 0 ' Keine Kommentare drucken
pmPrintInPlace       = 1 ' Kommentare ausdrucken
```

CenterHorizontally

Datentyp: **Boolean**

Liest oder setzt die Einstellung, ob das Arbeitsblatt horizontal zentriert ausgedruckt wird. Entspricht der Einstellung "Horizontal zentrieren" im Dialogfenster des Befehls **Datei > Seite einrichten > Optionen**.

CenterVertically

Datentyp: **Boolean**

Liest oder setzt die Einstellung, ob das Arbeitsblatt vertikal zentriert ausgedruckt wird. Entspricht der Einstellung "Vertikal zentrieren" im Dialogfenster des Befehls **Datei > Seite einrichten > Optionen**.

Zoom

Datentyp: **Long**

Liest oder setzt die Vergrößerungsstufe, mit der das Arbeitsblatt ausgedruckt wird. Entspricht der Einstellung "Skalierung" im Dialogfenster des Befehls **Datei > Seite einrichten > Optionen**.

FirstPageNumber

Datentyp: **Long**

Liest oder setzt die anfängliche Seitennummer, die beim Drucken verwendet wird. Sie können auch **pmAutomatic** übergeben, damit die erste gedruckte Seite die Seitennummer 1 erhält. Entspricht der Einstellung "Seitennummer" im Dialogfenster des Befehls **Datei > Seite einrichten > Optionen**.

PrintGridlines

Datentyp: **Boolean**

Liest oder setzt die Einstellung, ob auf dem Arbeitsblatt Gitternetzlinien ausgedruckt werden. Entspricht der Einstellung "Gitternetz" im Dialogfenster des Befehls **Datei > Seite einrichten > Optionen**.

PrintHeadings

Datentyp: **Boolean**

Liest oder setzt die Einstellung, ob auf dem Arbeitsblatt Zeilen- und Spaltenköpfe ausgedruckt werden. Entspricht der Einstellung "Zeilen- und Spaltenköpfe" im Dialogfenster des Befehls **Datei > Seite einrichten > Optionen**.

Order

Datentyp: **Long** (PmOrder)

Liest oder setzt die Reihenfolge, in der mehrseitige Arbeitsblätter ausgedruckt werden. Mögliche Werte:

```
pmOverThenDown    = 0 ' Von links nach rechts
pmDownThenOver    = 1 ' Von oben nach unten
```

Entspricht der Einstellung "Druckrichtung" im Dialogfenster des Befehls **Datei > Seite einrichten > Optionen**.

PrintArea

Datentyp: **String**

Liest oder setzt den Druckbereich des Arbeitsblatts, analog zum Befehl **Datei > Druckbereich > Druckbereich festlegen**.

Erhalten Sie einen leeren String, ist kein Druckbereich definiert. Übergeben Sie einen leeren String, wird ein bestehender Druckbereich entfernt.

PrintTitleRows

Datentyp: **String**

Liest oder setzt die Wiederholungszeilen des Arbeitsblatts, analog zur Einstellung "Wiederholungszeilen" im Dialogfenster des Befehls **Datei > Seite einrichten > Optionen**.

Beispiel:

```
' Zeilen 2 bis 5 des aktuellen Arbeitsblatts zu Wiederholungszeilen machen
pm.ActiveSheet.PageSetup.PrintTitleRows = "2:5"
```

PrintTitleColumns

Datentyp: **String**

Liest oder setzt die Wiederholungsspalten des Arbeitsblatts, analog zur Einstellung "Wiederholungsspalten" im Dialogfenster des Befehls **Datei** > **Seite einrichten** > **Optionen**.

Beispiel:

```
' Spalten A bis C des aktuellen Arbeitsblatts zu Wiederholungsspalten machen
pm.ActiveSheet.PageSetup.PrintTitleColumns = "A:C"
```

Application (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das **Application**-Objekt.

Parent (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das übergeordnete Objekt, also ein Objekt vom Typ **Sheet**.

Range (Objekt)

Zugriffspfade (für beliebige Zellbereiche):
- Application → Workbooks → Item → Sheets → Item → **Range**
- Application → Workbooks → ActiveSheet → **Range**
- Application → ActiveWorkbook → ActiveSheet → **Range**
- Application → ActiveSheet → **Range**
- Application → **Range**

Zugriffspfade (für ganze Tabellenzeilen):
- Application → Workbooks → Item → Sheets → Item → Rows → **Item**
- Application → Workbooks → ActiveSheet → Rows → **Item**
- Application → ActiveWorkbook → ActiveSheet → Rows → **Item**
- Application → ActiveSheet → Rows → **Item**
- Application → Rows → **Item**

Zugriffspfade (für ganze Tabellenspalten):
- Application → Workbooks → Item → Sheets → Item → Columns → **Item**
- Application → Workbooks → ActiveSheet → Columns → **Item**
- Application → ActiveWorkbook → ActiveSheet → Columns → **Item**
- Application → ActiveSheet → Columns → **Item**
- Application → Columns → **Item**

Zugriffspfade (für die momentan selektierten Zellen):
- Application → Workbooks → Item → Sheets → Item → **Selection**
- Application → Workbooks → ActiveSheet → **Selection**
- Application → ActiveWorkbook → ActiveSheet → **Selection**
- Application → ActiveSheet → **Selection**
- Application → **Selection**

1 Beschreibung

Range bezeichnet einen bestimmten Zellbereich in einem Arbeitsblatt (**Sheet**). Dieser Bereich kann beliebig viele Zellen umfassen, von einer einzigen Zelle bis zum gesamten Arbeitsblatt.

Mit einem **Range**-Objekt können Sie unter anderem die Inhalte und Formatierungen der Zellen im repräsentierten Bereich auslesen und ändern, aber auch den Zellbereich ausschneiden, in die Zwischenablage kopieren etc.

2 Zugriff auf das Objekt

Es gibt mehrere Möglichkeiten, ein **Range**-Objekt zu erhalten:

1. Sie sprechen das **Range**-Objekt direkt an, unter Benennung der Start- und Endzelle. Beispiel:

```
' In Zelle C10 einen Kommentar einfügen
pm.ActiveSheet.Range("C10").Comment = "Ein Kommentar"
```

2. Die Eigenschaft **Sheet.Selection** gibt ein **Range**-Objekt zurück, das die aktuelle Selektion repräsentiert, also die gegenwärtig markierten Zellen. Beispiel:

```
' Die aktuelle Selektion in der Schrift "Courier New" formatieren
pm.ActiveSheet.Selection.Font.Name = "Courier New"
```

3. Die Sammlung **Rows** gibt **Range**-Objekte zurück, die jeweils eine komplette Zeile des Arbeitsblatts repräsentieren. Sie können auf die **Rows**-Sammlung über **Sheet.Rows** zugreifen. Beispiel:

```
' Die Zeile 2 des Arbeitsblatts ausblenden
pm.ActiveSheet.Rows(2).Hidden = True
```

4. Die Sammlung **Columns** gibt **Range**-Objekte zurück, die jeweils eine komplette Spalte des Arbeitsblatts repräsentieren. Sie können auf die **Columns**-Sammlung über **Sheet.Columns** zugreifen. Beispiel:

```
' Die Spalte C (= dritte Spalte) des Arbeitsblatts ausblenden
pm.ActiveSheet.Columns(3).Hidden = True
```

Ganz egal, wie Sie das **Range**-Objekt erhalten, Sie können alle im Folgenden beschriebenen Eigenschaften und Methoden auf das Objekt anwenden.

3 Eigenschaften, Objekte, Sammlungen und Methoden

Eigenschaften:

- ***Item*** (Defaulteigenschaft)
- **Row** R/O
- **Column** R/O
- **Name**
- **Formula**
- **Value**
- **Value2**
- **HorizontalAlignment**
- **VerticalAlignment**
- **WrapText**
- **LeftPadding**
- **RightPadding**
- **TopPadding**
- **BottomPadding**
- **MergeCells**
- **Orientation**
- **VerticalText**
- **PageBreakCol**
- **PageBreakRow**
- **Comment**
- **Locked**
- **FormulaHidden**
- **CellHidden**
- **Nonprintable**
- **Hidden**
- **RowHeight**
- **ColumnWidth**

Objekte:

- **Cells → Range**
- **Range → Range**
- **Workbook → Workbook**
- **Sheet → Sheet**
- **NumberFormatting → NumberFormatting**
- **Font → Font**
- **Shading → Shading**
- **Validation → Validation**
- **Application → Application**
- **Parent → Sheet**

Sammlungen:

- **Borders → Borders**
- **FormatConditions → FormatConditions**

Methoden:

- **AutoFit**
- **ApplyFormatting**
- **Select**
- **Copy**
- **Cut**
- **Paste**
- **Insert**
- **Delete**
- **Clear**
- **ClearContents**
- **ClearFormats**
- **ClearConditionalFormatting**
- **ClearComments**
- **ClearInputValidation**
- **AutoFilter**

Item (Eigenschaft, R/O)

Datentyp: **Object**

Liefert ein **Range**-Objekt, das aus einer beliebigen einzelnen Zelle des aufrufenden **Range**-Objekts besteht. Dies lässt Sie jede einzelne Zelle des aufrufenden **Range**-Objekts individuell ansprechen.

Syntax:

```
Item(RowIndex, ColumnIndex)
```

Parameter:

RowIndex (Typ: **Long**) gibt die Zeilennummer der gewünschten Zelle an (relativ zum Anfang des Bereichs).

ColumnIndex (optional; Typ: **Long**) gibt die Spaltennummer der gewünschten Zelle an (relativ zum Anfang des Bereichs).

Beispiele:

```
' Die erste Zelle des Range-Objekts mit dem Wert 42 füllen
pm.ActiveSheet.Range("B5:B10").Item(1, 1).Value = 42

' Kürzer, da Item die Defaulteigenschaft des Range-Objekts ist
pm.ActiveSheet.Range("B5:B10")(1, 1).Value = 42

' Die erste Zelle der aktuellen Selektion umformatieren
pm.ActiveSheet.Selection.Item(1, 1).Font.Size = 24

' Wiederum kürzer durch Nutzung der Defaulteigenschaft
```

```
pm.ActiveSheet.Selection(1, 1).Font.Size = 24
```

Row (Eigenschaft, R/O)

Datentyp: **Long**

Liefert die Zeilennummer der obersten Zeile innerhalb des Bereichs.

Bei Mehrfachselektionen wird der Wert für den zuerst selektierten Bereich geliefert.

Column (Eigenschaft, R/O)

Datentyp: **Long**

Liefert die Spaltennummer der linken äußeren Spalte innerhalb des Bereichs.

Bei Mehrfachselektionen wird der Wert für den zuerst selektierten Bereich geliefert.

Name (Eigenschaft)

Datentyp: **String**

Liest oder setzt den Namen des Bereichs. Analog zum PlanMaker-Befehl **Tabelle > Namen** können Sie damit benannte Bereiche einrichten und auslesen.

Formula (Eigenschaft)

Datentyp: **String**

Liest oder setzt die Formel in den Zellen des Bereichs.

Beispiel:

```
' In Zellen A1, A2, B1 und B2 dieselbe Formel eintragen
pm.ActiveSheet.Range("A1:B2").Formula = "=ZEICHEN(64)"
```

Hinweis: Wenn die Formel nicht mit "=" oder "+" beginnt, wird sie als Literalwert (Zahl, String oder Datum) eingetragen.

Value (Eigenschaft)

Datentyp: **String**

Liest oder setzt den Wert in den Zellen des Bereichs. Datumsangaben werden dabei als *Zeichenkette* behandelt (siehe auch Value2-Eigenschaft weiter unten).

Beispiel:

```
' In Zellen A1, A2, B1 und B2 den Wert 42 eintragen
pm.ActiveSheet.Range("A1:B2").Value = 42
```

Value2 (Eigenschaft)

Datentyp: **String**

Liest oder setzt den Wert in den Zellen des Bereichs. Datumsangaben werden dabei als *Zahl* behandelt.

Unterschied zwischen Formula, Value und Value2

Mit allen drei Eigenschaften – **Formula**, **Value** und **Value2** – können Sie den Inhalt einer Zelle auslesen und ändern. Der Unterschied:

- **Formula** liefert, sofern eine Zelle eine *Rechenformel* enthält, den Formeltext zurück, zum Beispiel "=ABS(A1)".
- **Value** und **Value2** liefern stets das *Ergebnis* der Berechnung, mit einem Unterschied beim Auslesen von *Datumswerten*: Während **Value** hier eine Zeichenkette zurückliefert, erhalten Sie bei **Value2** die serielle Datumszahl.

HorizontalAlignment (Eigenschaft)

Datentyp: **Long** (PmHAlign)

Liest oder setzt die horizontale Ausrichtung der Zellen des Bereichs. Mögliche Werte:

```
pmHAlignGeneral                = 0 ' Standard
pmHAlignLeft                   = 1 ' Linksbündig
pmHAlignRight                  = 2 ' Rechtsbündig
pmHAlignCenter                 = 3 ' Zentriert
pmHAlignJustify                = 4 ' Blocksatz
pmHAlignCenterAcrossSelection  = 5 ' Zentriert über Spalten
```

VerticalAlignment (Eigenschaft)

Datentyp: **Long** (PmVAlign)

Liest oder setzt die vertikale Ausrichtung der Zellen des Bereichs. Mögliche Werte:

```
pmVAlignTop      = 0 ' Oben
pmVAlignCenter   = 1 ' Zentriert
pmVAlignBottom   = 2 ' Unten
pmVAlignJustify  = 3 ' Vertikaler Blocksatz
```

WrapText (Eigenschaft)

Datentyp: **Long**

Liest oder setzt die Einstellung "Zeilenumbruch" der Zellen des Bereichs, analog zur Option **Zeilenumbruch** im Dialogfenster des PlanMaker-Befehls **Format > Zelle**.

LeftPadding (Eigenschaft)

Datentyp: **Single**

Liest oder setzt den linken Innenrand der Zellen des Bereichs in Punkt (1 Punkt entspricht 1/72 Zoll).

RightPadding (Eigenschaft)

Datentyp: **Single**

Liest oder setzt den rechten Innenrand der Zellen des Bereichs in Punkt (1 Punkt entspricht 1/72 Zoll).

TopPadding (Eigenschaft)

Datentyp: **Single**

Liest oder setzt den oberen Innenrand der selektierten Zellen in Punkt (1 Punkt entspricht 1/72 Zoll).

BottomPadding (Eigenschaft)

Datentyp: **Single**

Liest oder setzt den unteren Innenrand der Zellen des Bereichs in Punkt (1 Punkt entspricht 1/72 Zoll).

MergeCells (Eigenschaft)

Datentyp: **Long**

Liest oder setzt die Einstellung "Zellen verbinden" im Dialogfenster des PlanMaker-Befehls **Format > Zelle**. Alle Zellen des Bereichs werden zu einer großen Zelle verbunden (**True**), oder die Zellverbindung wird wieder aufgehoben (**False**).

Orientation (Eigenschaft)

Datentyp: **Long**

Liest oder setzt für alle Zellen des Bereichs die Druckrichtung. Mögliche Werte: 0, 90, 180 und -90, entsprechend den jeweiligen Drehwinkeln.

Hinweis: Der Wert 270 wird automatisch in -90 gewandelt.

VerticalText (Eigenschaft)

Datentyp: **Long**

Liest oder setzt die Einstellung "Vertikaler Text" im Dialogfenster des PlanMaker-Befehls **Format > Zelle**.

PageBreakCol (Eigenschaft)

Datentyp: **Boolean**

Liest oder setzt die Einstellung, ob links von dem Bereich ein Seitenumbruch durchgeführt wird.

Setzen Sie diese Eigenschaft auf **True**, wird zwischen dem Bereich und der Spalte links davon ein vertikaler Seitenumbruch durchgeführt. Setzen Sie sie auf **False**, wird dieser wieder entfernt.

Entspricht dem Befehl **Einfügen > Seitenumbruch > Vor Spalte einfügen**.

PageBreakRow (Eigenschaft)

Datentyp: **Boolean**

Liest oder setzt die Einstellung, ob oberhalb des Bereichs ein Seitenumbruch durchgeführt wird.

Setzen Sie diese Eigenschaft auf **True**, wird oberhalb des Bereichs ein horizontaler Seitenumbruch durchgeführt. Setzen Sie sie auf **False**, wird dieser wieder entfernt.

Entspricht dem Befehl **Einfügen > Seitenumbruch > Vor Zeile einfügen**.

Comment (Eigenschaft)

Datentyp: **String**

Liest oder setzt den Kommentar der Zellen im Bereich. Beim Auslesen gilt: Unterscheiden sich die Kommentare oder sind keine hinterlegt, wird ein leerer String zurückgegeben.

Entspricht den Kommentaren, die in PlanMaker mit dem Befehl **Einfügen > Kommentar** angelegt und bearbeitet werden können.

Locked (Eigenschaft)

Datentyp: **Long**

Liest oder setzt die Einstellung "Zelle schützen", entsprechend der gleichnamigen Option im Dialog des PlanMaker-Befehls **Format > Zelle > Schutz**.

FormulaHidden (Eigenschaft)

Datentyp: **Long**

Liest oder setzt die Einstellung "Formel nicht anzeigen", entsprechend der gleichnamigen Option im Dialog des PlanMaker-Befehls **Format > Zelle > Schutz**.

CellHidden (Eigenschaft)

Datentyp: **Long**

Liest oder setzt die Einstellung "Zelle nicht anzeigen", entsprechend der gleichnamigen Option im Dialog des PlanMaker-Befehls **Format > Zelle > Schutz**.

Nonprintable (Eigenschaft)

Datentyp: **Long**

Liest oder setzt die Einstellung "Zelle nicht ausdrucken", entsprechend der gleichnamigen Option im Dialog des PlanMaker-Befehls **Format > Zelle > Schutz**.

Hidden (Eigenschaft)

Datentyp: **Long**

Liest oder setzt die Einstellung, ob komplette Spalten oder Zeilen ausgeblendet sind, analog zu den PlanMaker-Befehlen **Tabelle > Spalte > Ausblenden** und **Tabelle > Zeile > Ausblenden**.

Der Bereich muss stets eine oder mehrere *ganze* Zeilen oder Spalten bezeichnen. Im Folgenden einige Beispiele:

- Spalte A referenzieren Sie mit **A:A**.
- Die Spalten A bis C referenzieren Sie **A:C**.
- Zeile 3 referenzieren Sie mit **3:3**.
- Die Zeilen 3 bis 7 referenzieren Sie mit **3:7**.

Beispiele:

```
' Spalte A ausblenden
pm.ActiveSheet.Range("A:A").Hidden = True

' Spalten A, B und C ausblenden
pm.ActiveSheet.Range("A:C").Hidden = True

' Zeile 3 ausblenden
pm.ActiveSheet.Range("3:3").Hidden = True

' Zeilen 3 bis 7 ausblenden
pm.ActiveSheet.Range("3:7").Hidden = True
```

Alternativ können Sie ganze Zeilen auch über die **Rows**-Sammlung und ganze Spalten über die **Columns**-Sammlung adressieren:

```
' Spalte A (= 1. Spalte) ausblenden
pm.ActiveSheet.Columns(1).Hidden = True

' Zeile 3 ausblenden
pm.ActiveSheet.Columns(3).Hidden = True
```

RowHeight (Eigenschaft)

Datentyp: **Long**

Liest oder setzt die Höhe der Zeile in Punkt (1 Punkt entspricht 1/72 Zoll).

Damit diese Eigenschaft gesetzt oder gelesen werden kann, muss der Bereich eine oder mehrere *ganze Zeilen* umfassen. Beachten Sie hierzu die Hinweise bei der Eigenschaft **Hidden**.

ColumnWidth (Eigenschaft)

Datentyp: **Long**

Liest oder setzt die Breite der Spalte in Punkt (1 Punkt entspricht 1/72 Zoll).

Damit diese Eigenschaft gesetzt oder gelesen werden kann, muss der Bereich eine oder mehrere *ganze Spalten* umfassen. Beachten Sie hierzu die Hinweise bei der Eigenschaft **Hidden**.

Cells (Zeiger auf Objekt)

Datentyp: **Object**

Liefert ein **Range**-Objekt, dessen Elemente genau denen des Quellbereichs entsprechen. Dadurch können Sie die einzelnen Zellen eines Bereichs durch Schleifenvariablen adressieren. Beispiel:

```
' Alle Zellen des Bereichs mit Werten füllen
Dim row, col as Integer
Dim rng as Object

Set rng = pm.ActiveSheet.Range("A1:F50")
For row = 1 To rng.Rows.Count
  For col = 1 to rng.Columns.Count
     rng.Cells.Item(row, col).Value = 42
  Next col
Next row
```

Range (Zeiger auf Objekt)

Datentyp: **Object**

Liefert ein zu den übergebenen Parametern passendes **Range**-Objekt. Sie können auf diese Weise einen "Unterbereich" eines Bereichs konstruieren, um darin beispielsweise Werte auslesen und setzen.

Hinweis: Die Zelladressierung hat hierbei *relativ* zu erfolgen. Die Zelle B2 von einem Bereich bezeichnet also nicht die Zelle mit den absoluten Koordinaten B2, sondern die Zelle, die sich in der zweiten Zeile und Spalte des *Bereichs* befindet (siehe Beispiel).

Syntax 1:

```
obj = Range(Cell1)
```

Syntax 2:

```
obj = Range(Cell1, Cell2)
```

Parameter:

Cell1 (Typ: **String**) gibt entweder gemäß Syntax 1 einen Zellbereich an (dann muss **Cell2** weggelassen werden) oder gemäß Syntax 2 die linke obere Ecke eines Bereichs (dann gibt der Parameter **Cell2** die rechte untere Ecke des Bereichs an).

Cell2 (optional; Typ: **String**) darf nur verwendet werden, wenn **Cell1** eine einzelne Zelle referenziert, und gibt die rechte untere Ecke des Bereichs an).

Beispiele für Syntax 1:

```
Range("A1:B20")       ' Zellen A1 bis B20
Range("A1")           ' Nur Zelle A1
Range("A:A")          ' Gesamte Spalte A
Range("3:3")          ' Gesamte Zeile 3
Range("Sommer")       ' Benannter Bereich "Sommer"
```

Beispiel für Syntax 2:

```
Range("A1", "B20")    ' Zellen A1 bis B20
```

Beispiel:

```
' Selektiert die Zelle D4
pm.ActiveSheet.Range("B2:F20").Range("C3:C3").Select
```

Workbook (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das **Workbook**-Objekt, über das Sie auf die zum Bereich gehörende Arbeitsmappe (= Dokument) zugreifen können.

Sheet (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das **Sheet**-Objekt, über das Sie auf das zum Bereich gehörende Arbeitsblatt zugreifen können.

NumberFormatting (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das **NumberFormatting**-Objekt, über das Sie auf die Zahlenformatierung der Zellen des Bereichs zugreifen können.

Font (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das **Font**-Objekt, das Sie auf die Zeichenformatierung der Zellen des Bereichs zugreifen lässt.

Shading (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das **Shading**-Objekt, das Sie auf die Schattierung der Zellen des Bereichs zugreifen lässt.

Validation (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das **Validation**-Objekt, das Sie auf die Gültigkeitsprüfung im Bereich zugreifen lässt.

Application (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das **Application**-Objekt.

Parent (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das übergeordnete Objekt, also ein Objekt vom Typ **Sheet**.

Borders (Zeiger auf Sammlung)

Datentyp: **Object**

Liefert eine **Borders**-Sammlung, die die vier Umrandungslinien der Zellen des Bereichs repräsentiert. Sie können mit Hilfe dieser Sammlung die Linieneinstellungen (Dicke, Farbe etc.) auslesen und verändern.

FormatConditions (Zeiger auf Sammlung)

Datentyp: **Object**

Liefert die **FormatConditions**-Sammlung, eine Sammlung aller bedingten Formatierungen des Bereichs.

AutoFit (Methode)

Setzt die Zeile(n) beziehungsweise Spalte(n) in dem Bereich auf optimale Höhe bzw. Breite. Entspricht den Befehlen **Tabelle > Zeile > Optimale Höhe** und **Tabelle > Spalte > Optimale Breite**.

Der Bereich muss *ganze* Zeilen oder Spalten umfassen.

Syntax:

```
AutoFit
```

Parameter:

keine

Rückgabetyp:

keiner

Beispiele:

```
' Spalte A auf optimale Breite setzen
pm.ActiveSheet.Range("A:A").AutoFit

' Spalten A, B und C auf optimale Breite setzen
pm.ActiveSheet.Range("A:C").AutoFit

' Zeile 3 auf optimale Breite setzen
pm.ActiveSheet.Range("3:3").AutoFit

' Zeilen 3 bis 7 auf optimale Breite setzen
pm.ActiveSheet.Range("3:7").AutoFit

' Spalte A (= 1. Spalte) auf optimale Breite setzen
```

```
pm.ActiveSheet.Columns(1).AutoFit

' Zeile 3 auf optimale Breite setzen
pm.ActiveSheet.Rows(3).AutoFit
```

ApplyFormatting (Methode)

PlanMaker führt normalerweise alle Formatierungsanweisungen, die Sie an ihn übermitteln, sofort aus.

Wenn Sie allerdings an einem Zellbereich mehrere Formatierungen direkt hintereinander anbringen möchten, können Sie die Ausführung deutlich beschleunigen, indem Sie die Arbeitsblatteigenschaft **ManualApply** (siehe **Workbook**-Objekt) auf **True** setzen.

Dann sind Sie selbst dafür zuständig, PlanMaker den Abschluss der Formatierung mitzuteilen. Dies geschieht dadurch, dass Sie die Änderungen an den Formatierungen in eine **With**-Struktur einbetten und den Abschluss mit der **ApplyFormatting**-Methode auslösen (siehe Beispiel).

Syntax:

ApplyFormatting

Parameter:

keine

Rückgabetyp:

keiner

Beispiel mit automatischer Formatierung (der Normalfall):

```
Sub Main
  Dim pm as Object

  Set pm = CreateObject("PlanMaker.Application")
  pm.Visible = True

  With pm.ActiveSheet.Range("A1:C3")
     .Font.Name = "Arial"
     .Font.Size = 14
     .Font.Bold = TRUE
     .NumberFormatting.Type = pmNumberPercentage
     .NumberFormatting.Digits = 2
  End With

  Set pm = Nothing
End Sub
```

Beispiel mit manueller Formatierung:

```
Sub Main
  Dim pm as Object

  Set pm = CreateObject("PlanMaker.Application")
  pm.Visible = True

  pm.ActiveWorkbook.ManualApply = True
  With pm.ActiveSheet.Range("A1:C3")
     .Font.Name = "Arial"
     .Font.Size = 14
     .Font.Bold = TRUE
     .NumberFormatting.Type = pmNumberPercentage
     .NumberFormatting.Digits = 2
     .ApplyFormatting
  End With
  pm.ActiveWorkbook.ManualApply = False
```

```
  Set pm = Nothing
End Sub
```

Select (Methode)

Selektiert den per **Range**-Anweisung festgelegten Bereich.

Syntax:

```
Select [Add]
```

Parameter:

Add (optional; Typ: **Boolean**): Wenn dieser Wert **False** ist oder nicht angegeben wird, ersetzt die neue Selektion eine bestehende alte. Ansonsten wird die neue Selektion zur alten hinzugefügt.

Rückgabetyp:

keiner

Beispiele:

```
' Den Bereich B2:D4 selektieren
pm.ActiveSheet.Range("B2:D4").Select

' Die aktuelle Selektion um den Bereich F6:F10 erweitern
pm.ActiveSheet.Range("F6:F10").Select True
```

Deselektieren: Wenn Sie möchten, dass im Dokument nichts mehr selektiert sein soll, dann selektieren Sie einfach einen Range, der aus nur *einer* Zelle besteht:

```
' Den Zellrahmen in Zelle A1 setzen (ohne diese zu selektieren)
pm.ActiveSheet.Range("A1").Select
```

Copy (Methode)

Kopiert die Zellen des Bereichs in die Zwischenablage.

Syntax:

```
Copy
```

Parameter:

keine

Rückgabetyp:

keiner

Cut (Methode)

Schneidet die Zellen des Bereichs in die Zwischenablage aus.

Syntax:

```
Cut
```

Parameter:

keine

Rückgabetyp:

keiner

Paste (Methode)

Fügt den Inhalt der Zwischenablage in den Bereich ein. Enthält der Bereich mehr als eine Zelle, wird der Inhalt der Zwischenablage so beschnitten/erweitert, dass er genau in den Bereich passt.

Syntax:

```
Paste
```

Parameter:

keine

Rückgabetyp:

keiner

Insert (Methode)

Fügt einen leeren Zellbereich in der Größe des mit **Range** festgelegten Bereichs ein.

PlanMaker verhält sich dabei so, als ob Sie den Bereich selektiert und dann den Befehl **Tabelle > Zellen einfügen** aufgerufen hätten.

Syntax:

```
Insert [Shift]
```

Parameter:

Shift (optional; Typ: **Long** bzw. **PmInsertShiftDirection**): Gibt an, in welche Richtung die vorhandenen Zellen dabei ausweichen sollen. Mögliche Werte:

```
pmShiftDown = 0 ' Nach unten
pmShiftToRight = 1 ' Nach rechts
```

Wenn Sie den Parameter nicht angeben, wird **pmShiftDown** angenommen.

Rückgabetyp:

keiner

Delete (Methode)

Entfernt alle Zellen in dem durch den **Range** festgelegten Bereich. Die restlichen Zellen der Tabelle rücken dabei nach, um die Lücke zu füllen.

PlanMaker verhält sich dabei so, als ob Sie den Bereich selektiert und dann den Befehl **Tabelle > Zellen löschen** aufgerufen hätten.

Syntax:

```
Delete [Shift]
```

Parameter:

Shift (optional; Typ: **Long** bzw. **PmDeleteShiftDirection**): Gibt an, in welche Richtung die vorhandenen Zellen dabei nachrücken sollen. Mögliche Werte:

```
pmShiftUp      = 0 ' Nach oben
pmShiftToLeft  = 1 ' Nach links
```

Wenn Sie den Parameter nicht angeben, wird **pmShiftUp** angenommen.

Rückgabetyp:

keiner

Clear (Methode)

Löscht in dem durch den **Range** festgelegten Bereich alle Zellinhalte und -formatierungen.

Syntax:

```
Clear
```

Parameter:

keine

Rückgabetyp:

keiner

ClearContents (Methode)

Löscht in dem durch den **Range** festgelegten Bereich alle Zellinhalte. Die Formatierungen bleiben erhalten.

Syntax:

```
ClearContents
```

Parameter:

keine

Rückgabetyp:

keiner

ClearFormats (Methode)

Löscht in dem durch den **Range** festgelegten Bereich alle Zellformatierungen (jedoch nicht die bedingten Formatierungen). Die Zellinhalte bleiben erhalten.

Syntax:

```
ClearFormats
```

Parameter:

keine

Rückgabetyp:

keiner

ClearConditionalFormatting (Methode)

Löscht in dem durch den **Range** festgelegten Bereich alle bedingten Formatierungen. Die Zellinhalte bleiben erhalten.

Syntax:

```
ClearConditionalFormatting
```

Parameter:

keine

Rückgabetyp:

keiner

ClearComments (Methode)

Löscht in dem durch den **Range** festgelegten Bereich alle Kommentare.

Syntax:

```
ClearComments
```

Parameter:

keine

Rückgabetyp:

keiner

ClearInputValidation (Methode)

Löscht in dem durch den **Range** festgelegten Bereich alle Gültigkeitsprüfungen.

Syntax:

```
ClearInputValidation
```

Parameter:

keine

Rückgabetyp:

keiner

AutoFilter (Methode)

Aktiviert, deaktiviert oder konfiguriert einen AutoFilter für den Bereich.

Syntax:

```
AutoFilter [Field], [Criteria1], [Operator], [Criteria2], [VisibleDropDown]
```

Parameter:

Hinweis: Wenn Sie *keinen* der Parameter angeben, wird der bestehende AutoFilter für den angegebenen Zellbereich abgeschaltet (siehe Beispiele unten).

Field (optional; Typ: **Long**) gibt die Nummer der Spalte innerhalb des AutoFilter-Bereichs an, nach der Sie filtern möchten. Wenn Sie den Parameter weglassen, wird 1 (= erste Spalte) angenommen.

Criteria1 (optional; Typ: **Variant**) gibt den Filterbegriff an – zum Beispiel "Rot", wenn Sie nach dem Text "Rot" filtern möchten, oder ">3", um nach Werten größer als Drei zu filtern. Ausnahme: Bei den Operatoren **pmTop10Items**, **pmTop10Percent**, **pmBottom10Items** und **pmBottom10Percent** geben Sie hier an, wie viele Zeilen Sie sehen möchten. Wenn Sie **Criteria1** weglassen, werden alle Zeilen gezeigt.

Operator (optional; Typ: **Long** bzw. **PmAutoFilterOperator**) legt den Typ der Filterung fest:

```
pmAll              = 0 ' Alle Zeilen anzeigen (= nicht filtern)
pmAnd              = 1 ' Criteria1 und Criteria2 müssen zutreffen
pmBottom10Items    = 2 ' Nur die n Zeilen mit den niedrigsten Werten*
pmBottom10Percent  = 3 ' Nur die n Prozent mit den niedrigsten Werten*
pmOr               = 4 ' Criteria1 oder Criteria2 muss zutreffen
pmTop10Items       = 5 ' Nur die n Zeilen mit den höchsten Werten*
pmTop10Percent     = 6 ' Nur die n Prozent mit den höchsten Werten*
pmBlank            = 7 ' Nur leere Zeilen anzeigen
pmNonblank         = 8 ' Nur nicht-leere Zeilen anzeigen
```

* Den Wert für "n" übergeben Sie bei diesen Operatoren im Parameter **Criteria1**.

Criteria2 (optional; Typ: **Variant**) erlaubt es Ihnen, einen zweiten Filterbegriff anzugeben. Dies ist nur möglich bei den Operatoren **pmAnd** und **pmOr**.

VisibleDropDown (optional; Typ: **Boolean**) lässt Sie festlegen, ob Dropdown-Pfeile für den Filter angezeigt werden (**True**) oder nicht (**False**). Falls Sie den Parameter weglassen, wird **True** angenommen.

Rückgabetyp:

keiner

Beispiele:

pm.ActiveSheet.Range("A1:D10").AutoFilter 1, pmTop10Items, 5 weist PlanMaker an, nur noch die 5 größten Werte der Spalte A1 anzuzeigen.

Wenn Sie keinerlei Parameter angeben, schaltet der Aufruf dieser Methode einen bestehenden AutoFilter für den angegebenen Zellbereich wieder ab. Beispiel:

pm.ActiveSheet.Range("A1:D10").AutoFilter schaltet obigen AutoFilter wieder ab.

Rows (Sammlung)

Zugriffspfade für die Zeilen von Arbeitsblättern:
- Application → Workbooks → Item → Sheets → Item → **Rows**
- Application → Workbooks → Item → ActiveSheet → **Rows**
- Application → ActiveWorkbook → ActiveSheet → **Rows**
- Application → ActiveSheet → **Rows**
- Application → **Rows**

Zugriffspfade für die Zeilen beliebiger Bereiche:
- Application → Workbooks → Item → Sheets → Item → Range → **Rows**
- Application → Workbooks → ActiveSheet → Range → **Rows**
- Application → ActiveWorkbook → ActiveSheet → Range → **Rows**
- Application → ActiveSheet → Range → **Rows**
- Application → Range → **Rows**

Zugriffspfade für die Zeilen ganzer Tabellenspalten:
- Application → Workbooks → Item → Sheets → Item → Columns → Item → **Rows**
- Application → Workbooks → ActiveSheet → Columns → Item → **Rows**
- Application → ActiveWorkbook → ActiveSheet → Columns → Item → **Rows**
- Application → ActiveSheet → Columns → Item → **Rows**
- Application → Columns → Item → **Rows**

Zugriffspfade für die Zeilen der selektierten Zellen:
- Application → Workbooks → Item → Sheets → Item → Selection → **Rows**
- Application → Workbooks → ActiveSheet → Selection → **Rows**
- Application → ActiveWorkbook → ActiveSheet → Selection → **Rows**
- Application → ActiveSheet → Selection → **Rows**
- Application → Selection → **Rows**

1 Beschreibung

Rows ist die Sammlung aller Zeilen eines Arbeitsblatts oder Bereichs. Die einzelnen Elemente dieser Sammlung sind vom Typ **Range**, wodurch alle Eigenschaften und Methoden von Bereichen auf sie angewandt werden können.

2 Zugriff auf das Objekt

Rows kann ein Tochterobjekt zweier Objekte sein:

- Als Tochterobjekt eines **Sheet**-Objekts repräsentiert es alle Zeilen dieses Arbeitsblatts.
- Als Tochterobjekt eines **Range**-Objekts repräsentiert es alle Zellen dieses Bereichs.

Beispiele für **Rows** als Tochterobjekt eines Sheet-Objekts:

```
' Die Zahl der Zeilen des Arbeitsblatts anzeigen
```

```
MsgBox pm.ActiveSheet.Rows.Count

' Die erste Zeile des Arbeitsblatts auf Fettdruck setzen
pm.ActiveSheet.Rows(1).Font.Bold = True
```

Beispiele für **Rows** als Tochterobjekt eines Range-Objekts:

```
' Die Zahl der Zeilen des angegebenen Bereichs anzeigen
MsgBox pm.ActiveSheet.Range("A1:F50").Rows.Count

' Die erste Zeile des angegebenen Bereichs auf Fettdruck setzen
pm.ActiveSheet.Range("A1:F50").Rows(1).Font.Bold = True
```

3 Eigenschaften, Objekte, Sammlungen und Methoden

Eigenschaften:
- **Count** R/O

Objekte:
- ***Item*** → **Range** (Defaultobjekt)
- **Application** → **Application**
- **Parent** → **Sheet** oder **Range**

Count (Eigenschaft, R/O)

Datentyp: **Long**

Liefert die Anzahl der **Range**-Objekte in der **Rows**-Sammlung – in anderen Worten: die Anzahl der Zeilen im Arbeitsblatt oder Bereich.

Item (Zeiger auf Objekt)

Datentyp: **Object**

Liefert ein einzelnes **Range**-Objekt, also einen Bereich, der eine einzelne Zeile umfasst.

Welches Range-Objekt Sie erhalten, hängt von dem Zahlenwert ab, den Sie an **Item** übergeben: 1 für die erste Zeile, 2 für die zweite und 3 für die dritte.

Beispiel:

```
' Schrift in der zweiten Zeile des Arbeitsblatts auf Courier New setzen
pm.ActiveSheet.Rows.Item(2).Font.Name = "Courier New"
```

Application (Zeiger auf Objekt)

Liefert das **Application**-Objekt.

Parent (Zeiger auf Objekt)

Liefert das übergeordnete Objekt, also ein Objekt, das entweder vom Typ **Sheet** oder vom Typ **Range** ist.

Columns (Sammlung)

Zugriffspfade für die Spalten von Arbeitsblättern:

- Application → Workbooks → Item → Sheets → Item → **Columns**
- Application → Workbooks → Item → ActiveSheet → **Columns**
- Application → ActiveWorkbook → ActiveSheet → **Columns**
- Application → ActiveSheet → **Columns**
- Application → **Columns**

Zugriffspfade für die Spalten beliebiger Bereiche:
- Application → Workbooks → Item → Sheets → Item → Range → **Columns**
- Application → Workbooks → ActiveSheet → Range → **Columns**
- Application → ActiveWorkbook → ActiveSheet → Range → **Columns**
- Application → ActiveSheet → Range → **Columns**
- Application → Range → **Columns**

Zugriffspfade für die Spalten ganzer Tabellenzeilen:
- Application → Workbooks → Item → Sheets → Item → Rows → Item → **Columns**
- Application → Workbooks → ActiveSheet → Rows → Item → **Columns**
- Application → ActiveWorkbook → ActiveSheet → Rows→ Item → **Columns**
- Application → ActiveSheet → Rows→ Item → **Columns**
- Application → Rows→ Item → **Columns**

Zugriffspfade für die Spalten der selektierten Zellen:
- Application → Workbooks → Item → Sheets → Item → Selection → **Columns**
- Application → Workbooks → ActiveSheet → Selection → **Columns**
- Application → ActiveWorkbook → ActiveSheet → Selection → **Columns**
- Application → ActiveSheet → Selection → **Columns**
- Application → Selection → **Columns**

1 Beschreibung

Columns ist die Sammlung aller Spalten eines Arbeitsblatts oder Bereichs. Die einzelnen Elemente dieser Sammlung sind vom Typ **Range**, wodurch alle Eigenschaften und Methoden von Bereichen auf sie angewandt werden können.

2 Zugriff auf das Objekt

Columns kann ein Tochterobjekt zweier Objekte sein:

- Als Tochterobjekt eines **Sheet**-Objekts repräsentiert es alle Spalten dieses Arbeitsblatts.
- Als Tochterobjekt eines **Range**-Objekts repräsentiert es alle Spalten dieses Bereichs.

Beispiele für **Columns** als Tochterobjekt eines Sheet-Objekts:

```
' Die Zahl der Spalten des Arbeitsblatts anzeigen
MsgBox pm.ActiveSheet.Columns.Count

' Die erste Spalte des Arbeitsblatts auf Fettdruck setzen
pm.ActiveSheet.Columns(1).Font.Bold = True
```

Beispiele für **Columns** als Tochterobjekt eines Range-Objekts:

```
' Die Zahl der Spalten des angegebenen Bereichs anzeigen
MsgBox pm.ActiveSheet.Range("A1:F50").Columns.Count

' Die erste Spalte des angegebenen Bereichs auf Fettdruck setzen
pm.ActiveSheet.Range("A1:F50").Columns(1).Font.Bold = True
```

3 Eigenschaften, Objekte, Sammlungen und Methoden

Eigenschaften:
- **Count** R/O

Objekte:
- ***Item*** → **Range** (Defaultobjekt)
- **Application** → **Application**
- **Parent** → **Sheet** oder **Range**

Count (Eigenschaft, R/O)

Datentyp: **Long**

Liefert die Anzahl der **Range**-Objekte in der **Columns**-Sammlung – in anderen Worten: die Anzahl der Spalten im Arbeitsblatt oder Bereich.

Item (Zeiger auf Objekt)

Datentyp: **Object**

Liefert ein einzelnes **Range**-Objekt, also einen Bereich, der eine einzelne Spalte umfasst.

Welches Range-Objekt Sie erhalten, hängt von dem Zahlenwert ab, den Sie an **Item** übergeben: 1 für die erste Spalte, 2 für die zweite und 3 für die dritte.

Beispiel:

```
' Schrift in der zweiten Spalte (Spalte B) auf Courier New setzen
pm.ActiveSheet.Columns.Item(2).Font.Name = "Courier New"
```

Application (Zeiger auf Objekt)

Liefert das **Application**-Objekt.

Parent (Zeiger auf Objekt)

Liefert das übergeordnete Objekt, also ein Objekt, das entweder vom Typ **Sheet** oder vom Typ **Range** ist.

FormatConditions (Sammlung)

Zugriffspfade:
- Application → Workbooks → Item → Sheets → Item → Range → **FormatConditions**
- Application → Workbooks → ActiveSheet → Range → **FormatConditions**
- Application → ActiveWorkbook → ActiveSheet → Range → **FormatConditions**
- Application → ActiveSheet → Range → **FormatConditions**
- Application → Range → **FormatConditions**

Statt "Range" können Sie auch alle anderen Objekte und Eigenschaften angeben, die ein **Range**-Objekt zurückgeben: **ActiveCell**, **Selection**, **Rows(n)**, **Columns(n)** und **Cells(x, y)**. Beispiele für diese Zugriffspfade finden Sie beim **Range**-Objekt.

1 Beschreibung

Die Sammlung **FormatConditions** enthält alle bedingten Formatierungen eines **Range**-Objekts. Die einzelnen Elemente dieser Sammlung sind vom Typ **FormatCondition**.

2 Zugriff auf die Sammlung

Es existiert für jedes **Range**-Objekt genau eine Instanz der **FormatConditions**-Sammlung. Diese wird über **Range.FormatConditions** angesprochen:

```
' Die Anzahl der bedingten Formatierungen eines Range anzeigen
MsgBox pm.ActiveSheet.Range("A1:B3").FormatConditions.Count
```

3 Eigenschaften, Objekte, Sammlungen und Methoden

Eigenschaften:
- **Count** R/O

Objekte:
- ***Item*** → **FormatCondition** (Defaultobjekt)
- **Application** → **Application**
- **Parent** → **Range**

Methoden:
- **Add**
- **Delete**

Count (Eigenschaft, R/O)

Datentyp: **Long**

Liefert die Anzahl der **FormatCondition**-Objekte im **Range** – in anderen Worten: die Anzahl der bedingten Formatierungen in diesem Bereich.

Eine Zelle kann bis zu drei bedingte Formatierungen besitzen.

Item (Zeiger auf Objekt)

Datentyp: **Object**

Liefert ein einzelnes **FormatCondition**-Objekt, also eine einzelne bedingte Formatierung.

Welches FormatCondition-Objekt Sie erhalten, hängt von dem Zahlenwert ab, den Sie an **Item** übergeben: 1 für die erste bedingte Formatierung im Bereich, 2 für die zweite und 3 für die dritte.

Beispiel:

```
' Die Formel der zweiten bedingten Formatierung in Zelle A1 anzeigen
MsgBox pm.ActiveSheet.Range("A1").FormatConditions.Item(2).Formula1
```

Application (Zeiger auf Objekt)

Liefert das **Application**-Objekt.

Parent (Zeiger auf Objekt)

Liefert das übergeordnete Objekt, also **Range**.

Add (Methode)

Eine neue bedingte Formatierung hinzufügen.

Syntax:

```
Add Type, [Operator], [Formula1], [Formula2]
```

Parameter:

Type (Typ: **Long** bzw. **PmFormatConditionType**) legt den Typ der bedingten Formatierung fest. Mögliche Werte:

```
pmCellValue      = 0 ' Prüfung anhand des Zellwerts
pmExpression     = 1 ' Prüfung anhand einer Formel
```

Operator (optional; Typ: **Long** bzw. **PmFormatConditionOperator**) legt den Vergleichsoperator der bedingten Formatierung fest. Mögliche Werte:

```
pmBetween        = 0 ' zwischen
pmNotBetween     = 1 ' nicht zwischen
pmEqual          = 2 ' gleich
pmNotEqual       = 3 ' ungleich
pmGreater        = 4 ' größer als
pmLess           = 5 ' kleiner als
pmGreaterEqual   = 6 ' größer oder gleich
pmLessEqual      = 7 ' kleiner oder gleich
```

Formula1 (optional; Typ: **Variant**) legt entweder einen Zahlenwert fest oder einen String, der eine Zahl, einen Zellbezug oder eine Formel enthält. Bei **pmBetween** und **pmNotBetween** bestimmen Sie damit das Minimum, bei allen anderen Operatoren den Wert.

Formula2 (optional; Typ: **Variant**) legt entweder einen Zahlenwert fest oder einen String, der eine Zahl, einen Zellbezug oder eine Formel enthält. Er gibt das Maximum an und darf nur bei **pmBetween** und **pmNotBetween** verwendet werden.

Rückgabetyp:

Object (ein **FormatCondition**-Objekt, das die neue bedingte Formatierung repräsentiert)

Übersicht über die Kombinationsmöglichkeiten der Parameter:

Type	Operator	Formula1	Formula2
pmCellValue	Alle oben genannten	Enthält bei **pmBetween** und **pmNotBetween** das Minimum, bei allen anderen Operatoren den Wert.	Darf nur bei **pmBetween** und **pmNotBetween** verwendet werden und enthält dann das Maximum.
pmExpression	(Nicht verwendet)	Muss einen Ausdruck enthalten, der **True** liefert, wenn die Bedingung erfüllt ist, ansonsten **False**.	(Nicht verwendet)

Beispiel:

```
' Eine bedingte Formatierung (Werte von 1 bis 50) zur Zelle A1 hinzufügen
Dim newCondition as Object
Set newCondition = pm.ActiveSheet.Range("A1").FormatConditions.Add(pmCellValue,
   pmBetween, 1, 50)

' Diese bedingte Formatierung soll die Schriftfarbe auf Rot ändern
newCondition.Font.Color = smoColorRed
```

Delete (Methode)

Entfernt alle bedingten Formatierungen aus dem Bereich.

Syntax:

```
Delete
```

Parameter:

keine

Rückgabetyp:

keiner

Beispiel:

```
' Alle bedingten Formatierungen aus den Zellen A1 und A2 entfernen
pm.ActiveSheet.Range("A1:A2").FormatConditions.Delete
```

FormatCondition (Objekt)

Zugriffspfade:
- Application → Workbooks → Item → Sheets → Item → Range → FormatConditions → **Item**
- Application → Workbooks → ActiveSheet → Range → FormatConditions → **Item**
- Application → ActiveWorkbook → ActiveSheet → Range → FormatConditions → **Item**
- Application → ActiveSheet → Range → FormatConditions → **Item**
- Application → ActiveSheet → Range → FormatConditions → **Item**

Statt "Range" können Sie auch alle anderen Objekte und Eigenschaften angeben, die ein **Range**-Objekt zurückgeben: **ActiveCell**, **Selection**, **Rows(n)**, **Columns(n)** und **Cells(x, y)**. Beispiele für diese Zugriffspfade finden Sie beim **Range**-Objekt.

1 Beschreibung

Ein **FormatCondition**-Objekt repräsentiert eine einzelne bedingte Formatierung.

Bringen Sie an einem Bereich bedingte Formatierungen an oder entfernen diese, werden die zugehörigen **FormatCondition**-Objekte dynamisch angelegt bzw. entfernt.

2 Zugriff auf das Objekt

Die einzelnen **FormatCondition**-Objekte können ausschließlich durch Aufzählung der Elemente von Sammlungen des Typs **FormatConditions** angesprochen werden.

Für jeden Bereich existiert genau eine Instanz dieser **FormatConditions**-Sammlung. Sie erreichen diese über **Range.FormatConditions**:

```
' Die Formel der zweiten bedingten Formatierung in Zelle A1 anzeigen
MsgBox pm.ActiveSheet.Range("A1").FormatConditions.Item(2).Formula1
```

3 Eigenschaften, Objekte, Sammlungen und Methoden

- **Type** R/O
- **Operator** R/O
- **Formula1** R/O
- **Formula2** R/O

Objekte:
- **Font** → **Font**
- **Shading** → **Shading**
- **Application** → **Application**
- **Parent** → **FormatConditions**

Sammlungen:
- **Borders** → **Borders**

Methoden:
- **Modify**
- **Delete**

Type (Eigenschaft, R/O)

Datentyp: **Long** (PmFormatConditionType)

Liefert den Typ der bedingten Formatierung. Mögliche Werte:

```
pmCellValue    = 0 ' Prüfung anhand des Zellwerts
pmExpression   = 1 ' Prüfung anhand einer Formel
```

Operator (Eigenschaft, R/O)

Datentyp: **Long** (PmFormatConditionOperator)

Liefert den Vergleichsoperator der bedingten Formatierung. Mögliche Werte:

```
pmBetween         = 0 ' zwischen
pmNotBetween      = 1 ' nicht zwischen
pmEqual           = 2 ' gleich
pmNotEqual        = 3 ' ungleich
pmGreater         = 4 ' größer als
pmLess            = 5 ' kleiner als
pmGreaterEqual    = 6 ' größer oder gleich
pmLessEqual       = 7 ' kleiner oder gleich
```

Formula1 (Eigenschaft, R/O)

Datentyp: **String**

Liefert, sofern der Typ **pmCellValue** ist und der Operator **pmBetween** oder **pmNotBetween**, das Minimum, gegen das verglichen wird.

Liefert, sofern der Typ **pmCellValue** ist und der Operator ein anderer als **pmBetween** oder **pmNotBetween**, den Wert, gegen den verglichen wird.

Liefert, sofern der Typ **pmExpression** ist, die Rechenformel, gegen die verglichen wird.

Formula2 (Eigenschaft, R/O)

Datentyp: **String**

Liefert, sofern der Typ **pmCellValue** ist und der Operator **pmBetween** oder **pmNotBetween**, das Maximum, gegen das verglichen wird.

Font (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das **Font**-Objekt, das Sie auf die Zeichenformatierung zugreifen lässt, die von der bedingten Formatierung ausgelöst wird.

Hinweis: Diejenigen Zeichenformatierungen, die nicht *explizit* bei der bedingten Formatierung gesetzt wurden, werden als "undefiniert" (also **smoUndefined** bzw. Leerstring) zurückgeliefert. Das hat den Grund, dass bedingte Formatierungen immer *additiv* sind: Sie könnten zum Beispiel die Schrift "Arial 10" fest formatiert haben, und durch die bedingte Formatierung soll nur die Schriftfarbe "Rot" hinzugefügt werden – dann ist in **FormatCondition.Font** auch tatsächlich nur die Schriftfarbe gesetzt.

Shading (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das **Shading**-Objekt, das Sie auf die Schattierung zugreifen lässt, die von der bedingten Formatierung ausgelöst wird.

Bitte beachten Sie den Hinweis beim Objektzeiger **Font** (siehe oben) zu undefinierten Formatierungen.

Application (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das **Application**-Objekt.

Parent (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das übergeordnete Objekt, also ein Objekt vom Typ **FormatConditions**.

Borders (Zeiger auf Sammlung)

Datentyp: **Object**

Liefert eine **Borders**-Sammlung, die die vier Umrandungslinien von Zellen repräsentiert, die der bedingten Formatierung unterliegen. Sie können mit Hilfe dieses Objekts die Linieneinstellungen (Dicke, Farbe etc.) auslesen und verändern.

Bitte beachten Sie den Hinweis beim Objektzeiger **Font** (siehe oben) zu undefinierten Formatierungen.

Modify (Methode)

Die bestehende bedingte Formatierung modifizieren.

Syntax:

```
Modify Type, [Operator], [Formula1], [Formula2]
```

Parameter:

Type (Typ: **Long** bzw. **PmFormatConditionType**) legt den Typ der bedingten Formatierung fest. Mögliche Werte:

```
pmCellValue      = 0 ' Prüfung anhand des Zellwerts
pmExpression     = 1 ' Prüfung anhand einer Formel
```

Operator (optional; Typ: **Long** bzw. **PmFormatConditionOperator**) legt den Vergleichsoperator der bedingten Formatierung fest. Mögliche Werte:

```
pmBetween        = 0 ' zwischen
pmNotBetween     = 1 ' nicht zwischen
pmEqual          = 2 ' gleich
pmNotEqual       = 3 ' ungleich
pmGreater        = 4 ' größer als
pmLess           = 5 ' kleiner als
pmGreaterEqual   = 6 ' größer oder gleich
pmLessEqual      = 7 ' kleiner oder gleich
```

Formula1 (optional; Typ: **Variant**) legt entweder einen Zahlenwert fest oder einen String, der eine Zahl, einen Zellbezug oder eine Formel enthält. Bei **pmBetween** und **pmNotBetween** legen Sie damit das Minimum fest, bei allen anderen Operatoren den Wert.

Formula2 (optional; Typ: **Variant**) legt entweder einen Zahlenwert fest oder einen String, der eine Zahl, einen Zellbezug oder eine Formel enthält. Darf nur bei **pmBetween** und **pmNotBetween** angegeben werden.

Rückgabetyp:

keiner

Übersicht über die Kombinationsmöglichkeiten der Parameter:

Type	Operator	Formula1	Formula2
pmCellValue	Alle oben genannten	Enthält bei **pmBetween** und **pmNotBetween** das Minimum, bei allen anderen Operatoren den Wert.	Darf nur bei **pmBetween** und **pmNotBetween** verwendet werden und enthält dann das Maximum.
pmExpression	(Nicht verwendet)	Muss einen Ausdruck enthalten, der **True** liefert, wenn die Bedingung erfüllt ist, ansonsten **False**.	(Nicht verwendet)

Delete (Methode)

Entfernt die bedingte Formatierung aus dem Bereich.

Syntax:

```
Delete
```

Parameter:

keine

Rückgabetyp:

keiner

Beispiel:

```
' Die erste bedingte Formatierung aus den Zellen A1 und A2 entfernen
pm.ActiveSheet.Range("A1:A2").FormatConditions.Item(1).Delete
```

NumberFormatting (Objekt)

Zugriffspfade:

- Application → Workbooks → Item → Sheets → Item → Range → **NumberFormatting**
- Application → Workbooks → ActiveSheet → Range → **NumberFormatting**
- Application → ActiveWorkbook → ActiveSheet → Range → **NumberFormatting**
- Application → ActiveSheet → Range → **NumberFormatting**

Statt "Range" können Sie auch alle anderen Objekte und Eigenschaften angeben, die ein **Range**-Objekt zurückgeben: **ActiveCell**, **Selection**, **Rows(n)**, **Columns(n)** und **Cells(x, y)**. Beispiele für diese Zugriffspfade finden Sie beim **Range**-Objekt.

1 Beschreibung

Mit dem **NumberFormatting**-Objekt können Sie das Zahlenformat eines Bereichs auslesen und ändern (entsprechend den Optionen im Dialog des PlanMaker-Befehls **Format** > **Zelle**, Karteikarte **Zahlenformat**).

2 Zugriff auf das Objekt

NumberFormatting ist ein Tochterobjekt des **Range**-Objekts – zu jedem **Range**-Objekt existiert genau *ein* **NumberFormatting**-Objekt.

Eigenschaften:
- ***Type*** (Defaulteigenschaft)
- **DateFormat**
- **CustomFormat**
- **Currency**
- **Accounting**
- **Digits**
- **NegativeRed**
- **SuppressMinus**
- **SuppressZeros**
- **ThousandsSeparator**

Objekte:
- **Application → Application**
- **Parent → Range**

Type (Eigenschaft)

Datentyp: **Long** (PmNumberFormatting)

Liest oder setzt das Zahlenformat der Zellen des Bereichs. Mögliche Werte:

```
pmNumberGeneral      = 0 ' Standard
pmNumberDecimal      = 1 ' Zahl
pmNumberScientific   = 2 ' Wissenschaftlich
pmNumberFraction     = 3 ' Bruch (für den Nenner siehe Digits-Eigenschaft)
pmNumberDate         = 4 ' Datum/Uhrzeit (siehe Hinweis)
pmNumberPercentage   = 5 ' Prozent
pmNumberCurrency     = 6 ' Währung (siehe Hinweis)
pmNumberBoolean      = 7 ' Wahrheitswert
pmNumberCustom       = 8 ' Benutzerdefiniert (siehe Hinweis)
pmNumberAccounting   = 10 ' Buchhaltung (siehe Hinweis)
```

Hinweis: Die Formate **pmNumberDate**, **pmNumberCurrency**, **pmNumberAccounting** und **pmNumberCustom** dürfen Sie nur auslesen, nicht aber setzen. Zum Setzen verwenden Sie die Eigenschaften **DateFormat**, **Currency**, **Accounting** und **CustomFormat** (siehe unten).

DateFormat (Eigenschaft)

Datentyp: **String**

Liest oder setzt das Datums-/Zeitformat der Zellen des Bereichs.

Beispiel:

```
' Zelle A1 als Datum formatieren
pm.ActiveSheet.Range("A1").NumberFormatting.DateFormat = "TT.MM.JJJJ"
```

Informationen zu den verfügbaren Formatcodes finden Sie in der Online-Hilfe von PlanMaker, Stichwort "Benutzerdefinierte Zahlenformate".

Hinweis: Die Buchstabenkennungen für die Bestandteile eines Datumsformats sind sprachspezifisch. Läuft PlanMaker mit deutscher Benutzeroberfläche, ist TT.MM.JJJJ ein gültiges Datumsformat. Bei englischer Benutzeroberfläche müssten Sie stattdessen DD.MM.YYYY angeben, bei französischer Oberfläche JJ.MM.AAAA etc.

Wenn Sie den in einer Zelle verwendeten Datumsstring *auslesen* möchten, müssen Sie zuerst prüfen, ob eine Zelle überhaupt als Datum formatiert ist – ansonsten schlägt die Eigenschaft fehl:

```
' Den Datumsstring in Zelle A1 anzeigen
With pm.ActiveSheet.Range("A1")
```

```
  If .NumberFormatting.Type = pmNumberDate Then
     MsgBox .NumberFormatting.DateFormat
  Else
     MsgBox "Zelle A1 ist nicht als Datum formatiert."
  End If
End With
```

CustomFormat (Eigenschaft)

Datentyp: **String**

Liest oder setzt die benutzerdefinierte Formatierung der Zellen des Bereichs.

Beispiel:

```
' Zelle A1 in einem benutzerdefinieten Format formatieren
pm.ActiveSheet.Range("A1").NumberFormatting.CustomFormat = "000000"
```

Currency (Eigenschaft)

Datentyp: **String**

Liest oder setzt das Währungsformat der Zellen des Bereichs.

Sie übergeben der Eigenschaft den ISO-Kenner der gewünschten Währung. Beim Auslesen erhalten Sie analog den ISO-Kenner zurück. Einige gebräuchliche ISO-Kenner:

```
EUR   Euro
USD   US Dollar
CAD   Kanadische Dollar
AUD   Australische Dollar
JPY   Japanische Yen
RUB   Russische Rubel
BEF   Belgische Francs
CHF   Schweizer Franken
DEM   Deutsche Mark
ESP   Spanische Peseten
FRF   Französische Francs
LUF   Luxemburgische Francs
NLG   Niederländische Gulden
PTE   Portugiesische Escudos
```

Eine Liste aller ISO-Kenner (von denen PlanMaker viele, jedoch nicht alle unterstützt) finden Sie im Internet unter http://en.wikipedia.org/wiki/ISO_4217

Beispiel:

```
' Zelle A1 in der Währung "Euro" formatieren
pm.ActiveSheet.Range("A1").NumberFormatting.Currency = "EUR"
```

Wenn Sie den in einer Zelle verwendeten Währungsstring *auslesen* möchten, müssen Sie zuerst prüfen, ob eine Zelle überhaupt als Währung formatiert ist – ansonsten schlägt die Eigenschaft fehl:

```
' Den Währungsstring in Zelle A1 anzeigen
With pm.ActiveSheet.Range("A1")
  If .NumberFormatting.Type = pmNumberCurrency Then
     MsgBox .NumberFormatting.Currency
  Else
     MsgBox "Zelle A1 ist nicht als Währung formatiert."
  End If
End With
```

Accounting (Eigenschaft)

Datentyp: **String**

Liest oder setzt das Buchhaltungsformat für die Zellen des Bereichs.

Genau wie bei der Eigenschaft **Currency** (siehe dort) übergeben Sie der Eigenschaft den ISO-Kenner der gewünschten Währung. Beim Auslesen erhalten Sie analog den ISO-Kenner zurück.

Beispiel:

```
' Zelle A1 im Buchhaltungsformat mit der Währung "Euro" formatieren
pm.ActiveSheet.Range("A1").NumberFormatting.Accounting = "EUR"
```

Wenn Sie den in einer Zelle verwendeten Buchhaltungsstring *auslesen* möchten, müssen Sie zuerst prüfen, ob eine Zelle überhaupt im Buchhaltungsformat formatiert ist – ansonsten schlägt die Eigenschaft fehl:

```
' Den Währungsstring in Zelle A1 anzeigen (Format: Buchhaltung)
With pm.ActiveSheet.Range("A1")
  If .NumberFormatting.Type = pmNumberAccounting Then
    MsgBox .NumberFormatting.Accounting
  Else
    MsgBox "Zelle A1 ist nicht im Format Buchhaltung formatiert."
  End If
End With
```

Digits (Eigenschaft)

Datentyp: **Long**

Liest oder setzt in den Zellen des Bereichs die Zahl der anzuzeigenden Nachkommastellen.

Diese Eigenschaft kann bei folgenden Zahlenformaten eingesetzt werden:

- Zahl (**pmNumberDecimal**)
- Wissenschaftlich (**pmNumberScientific**)
- Prozent (**pmNumberPercentage**)
- Währung (**pmNumberCurrency**)
- Buchhaltung (**pmNumberAccounting**)

Beispiel:

```
' Zelle A1 mit 4 Nachkommastellen formatieren
pm.ActiveSheet.Range("A1").NumberFormatting.Digits = 4
```

Auch beim Zahlenformat "Bruch" (**pmNumberFraction**) können Sie die Eigenschaft einsetzen, hier legt sie aber den *Nenner* des Bruchs fest:

```
' Zelle A1 als Bruch mit Nenner 8 formatieren
With pm.ActiveSheet.Range("A1")
  .NumberFormatting.Type = pmNumberFraction
  .NumberFormatting.Digits = 8
End With
```

Beim Zahlenformat "Bruch" darf **Digits** zwischen 0 und 1000 liegen, bei allen anderen Zahlenformaten zwischen 0 und 15.

NegativeRed (Eigenschaft)

Datentyp: **Boolean**

Liest oder setzt in den Zellen des Bereichs die Einstellung "Negative Werte in Rot", entsprechend der gleichnamigen Option im Dialogfenster des Befehls **Format > Zelle**.

SuppressMinus (Eigenschaft)

Datentyp: **Boolean**

Liest oder setzt in den Zellen des Bereichs die Einstellung "Minuszeichen unterdrücken", entsprechend der gleichnamigen Option im Dialogfenster des Befehls **Format > Zelle**.

SuppressZeros (Eigenschaft)

Datentyp: **Boolean**

Liest oder setzt in den Zellen des Bereichs die Einstellung "Null nicht anzeigen", entsprechend der gleichnamigen Option im Dialogfenster des Befehls **Format > Zelle**.

ThousandsSeparator (Eigenschaft)

Datentyp: **Boolean**

Liest oder setzt in den Zellen des Bereichs die Einstellung "Tausendertrennzeichen", entsprechend der gleichnamigen Option im Dialogfenster des Befehls **Format > Zelle**.

Application (Zeiger auf Objekt)

Liefert das **Application**-Objekt.

Parent (Zeiger auf Objekt)

Liefert das übergeordnete Objekt, also ein Objekt vom Typ **Range**.

Beispiel für das NumberFormatting-Objekt

Im nachfolgenden Beispiel wird der Bereich A1 bis C3 als Prozentwerte mit 2 Nachkommastellen formatiert.

```
Sub Main
  Dim pm as Object

  Set pm = CreateObject("PlanMaker.Application")
  pm.Visible = True

  With pm.ActiveSheet.Range("A1:C3")
     .NumberFormatting.Type = pmNumberPercentage
     .NumberFormatting.Digits = 2
  End With

  Set pm = Nothing
End Sub
```

Font (Objekt)

Zugriffspfade für die direkte Formatierung:
- Application → Workbooks → Item → Sheets → Item → Range → **Font**
- Application → Workbooks → ActiveSheet → Range → **Font**
- Application → ActiveWorkbook → ActiveSheet → Range → **Font**
- Application → ActiveSheet → Range → **Font**

Zugriffspfade für die bedingte Formatierung:
- Application → Workbooks → Item → Sheets → Item → Range → FormatConditions → Item→ **Font**
- Application → Workbooks → ActiveSheet → Range → FormatConditions → Item→ **Font**
- Application → ActiveWorkbook → ActiveSheet → Range → FormatConditions → Item→ **Font**
- Application → ActiveSheet → Range → FormatConditions → Item→ **Font**

Statt "Range" können Sie auch alle anderen Objekte und Eigenschaften angeben, die ein **Range**-Objekt zurückgeben: **ActiveCell**, **Selection**, **Rows(n)**, **Columns(n)** und **Cells(x, y)**. Beispiele für diese Zugriffspfade finden Sie beim **Range**-Objekt.

1 Beschreibung

Das **Font**-Objekt beschreibt die Zeichenformatierung (Schriftart, Textfarbe, Unterstreichen etc.) von Zellen.

2 Zugriff auf das Objekt

Font kann ein Tochterobjekt zweier Objekte sein:

- Als Tochterobjekt eines **Range**-Objekts repräsentiert es die Zeichenformatierung der *Zellen* in diesem Bereich, entsprechend dem PlanMaker-Befehl **Format** > **Zeichen**.
- Als Tochterobjekt eines **FormatCondition**-Objekts repräsentiert es die Zeichenformatierung, die durch die *bedingte Formatierung* ausgelöst wird.

Beispiele:

```
' Schrift von Zelle A1 anzeigen
MsgBox pm.ActiveSheet.Range("A1").Font.Name

' Schrift der ersten bedingten Formatierung von A1 anzeigen
MsgBox pm.ActiveSheet.FormatConditions(1).Font.Name
```

3 Eigenschaften, Objekte, Sammlungen und Methoden

Eigenschaften:
- ***Name*** (Defaulteigenschaft)
- **Size**
- **Bold**
- **Italic**
- **Underline**
- **StrikeThrough**
- **Superscript**
- **Subscript**
- **AllCaps**
- **SmallCaps**
- **PreferredSmallCaps**
- **Blink**
- **Color**
- **ColorIndex**
- **BColor**
- **BColorIndex**
- **Spacing**
- **Pitch**

Objekte:
- **Application → Application**
- **Parent → Range** oder **FormatCondition**

Name (Eigenschaft)

Datentyp: **String**

Liest oder setzt den Namen der Schriftart (als Zeichenkette).

Falls innerhalb der Zellen mehrere Schriftarten vorkommen, wird eine leere Zeichenkette zurückgeliefert.

Size (Eigenschaft)

Datentyp: **Single**

Liest oder setzt die Schriftgröße in Punkt (pt).

Falls innerhalb der Zellen mehrere Schriftgrößen vorkommen, wird **smoUndefined** (9.999.999) zurückgeliefert.

Beispiel:

```
' Setze die Schriftgröße der selektierten Zellen auf 10,3 pt
pm.ActiveSheet.Selection.Font.Size = 10.3
```

Bold (Eigenschaft)

Datentyp: **Long**

Liest oder setzt die Zeichenformatierung "Fettdruck":

- **True:** Fettdruck ein
- **False:** Fettdruck aus
- **smoUndefined** (nur beim Lesen): Die Zellen sind teils fett, teils nicht.

Italic (Eigenschaft)

Datentyp: **Long**

Liest oder setzt die Zeichenformatierung "Kursivschrift":

- **True:** Kursivschrift ein
- **False:** Kursivschrift aus
- **smoUndefined** (nur beim Lesen): Die Zellen sind teils kursiv, teils nicht.

Underline (Eigenschaft)

Datentyp: **Long** (PmUnderline)

Liest oder setzt die Zeichenformatierung "Unterstreichen". Folgende Werte sind zulässig:

```
pmUnderlineNone         = 0 ' aus
pmUnderlineSingle       = 1 ' einfach durchgehend
pmUnderlineDouble       = 2 ' doppelt durchgehend
pmUnderlineWords        = 3 ' einfach wortweise
```

```
pmUnderlineWordsDouble = 4 ' doppelt wortweise
```

Lesen Sie die Eigenschaft aus und die Zellen sind teils unterstrichen, teils nicht, wird **smoUndefined** zurückgeliefert.

StrikeThrough (Eigenschaft)

Datentyp: **Long**

Liest oder setzt die Zeichenformatierung "Durchstreichen":

- **True:** Durchstreichen ein
- **False:** Durchstreichen aus
- **smoUndefined** (nur beim Lesen): Die Zellen sind teils durchgestrichen, teils nicht.

Superscript (Eigenschaft)

Datentyp: **Long**

Liest oder setzt die Zeichenformatierung "Hochstellen":

- **True:** Hochstellen ein
- **False:** Hochstellen aus
- **smoUndefined** (nur beim Lesen): Die Zellen sind teils hochgestellt, teils nicht.

Subscript (Eigenschaft)

Datentyp: **Long**

Liest oder setzt die Zeichenformatierung "Tiefstellen":

- **True:** Tiefstellen ein
- **False:** Tiefstellen aus
- **smoUndefined** (nur beim Lesen): Die Zellen sind teils tiefgestellt, teils nicht.

AllCaps (Eigenschaft)

Datentyp: **Long**

Liest oder setzt die Zeichenformatierung "Versalien" (Text in Großbuchstaben):

- **True:** Versalien ein
- **False:** Versalien aus
- **smoUndefined** (nur beim Lesen): Das Attribut ist bei einem Teil der Zellen gesetzt, bei einem anderen Teil nicht.

SmallCaps (Eigenschaft)

Datentyp: **Long**

Liest oder setzt die Zeichenformatierung "Kapitälchen":

- **True:** Kapitälchen ein
- **False:** Kapitälchen aus
- **smoUndefined** (nur beim Lesen): Die Zellen sind teils in Kapitälchen, teils nicht.

PreferredSmallCaps (Eigenschaft)

Datentyp: **Long**

Liest oder setzt die Zeichenformatierung "Kapitälchen", lässt Sie aber im Gegensatz zur Eigenschaft **SmallCaps** den Skalierungsfaktor frei wählen. Der Wert 0 bedeutet "keine Kapitälchen", alle anderen Werte stellen den prozentualen Skalierungsfaktor der Kapitälchen dar.

Beispiel:

```
' Formatiere die aktive Zelle als Kapitälchen mit 75% Größe
tm.ActiveCell.Font.PreferredSmallCaps = 75

' Schalte die Kapitälchen wieder aus
tm.ActiveCell.Font.PreferredSmallCaps = 0
```

Blink (Eigenschaft)

Datentyp: **Long**

Liest oder setzt die Zeichenformatierung "Blinken":

- **True:** Blinken ein
- **False:** Blinken aus
- **smoUndefined** (nur beim Lesen): Die Zellen sind teils blinkend, teils nicht.

Color (Eigenschaft)

Datentyp: **Long** (SmoColor)

Liest oder setzt die Vordergrundfarbe der Schrift als "BGR"-Wert (Blau-Grün-Rot-Triplet). Sie können entweder einen beliebigen Wert angeben oder eine der vordefinierten BGR-Farbkonstanten verwenden.

Falls die Zellen in unterschiedlichen Farben formatiert sind, wird beim Auslesen **smoUndefined** zurückgeliefert.

ColorIndex (Eigenschaft)

Datentyp: **Long** (SmoColorIndex)

Liest oder setzt die Vordergrundfarbe der Schrift als Indexfarbe. "Indexfarben" sind die Standardfarben von PlanMaker, durchnummeriert von 0 für Schwarz bis 15 für Hellgrau. Sie dürfen ausschließlich einen der in der Tabelle der Indexfarben genannten Werte verwenden.

Falls die Zellen in unterschiedlichen Farben oder in einer anderen als einer der Indexfarben formatiert sind, wird beim Auslesen **smoUndefined** zurückgeliefert.

Wir empfehlen, stattdessen die Eigenschaft **Color** (siehe dort) zu verwenden, da Sie mit dieser nicht auf die 16 Standardfarben beschränkt sind, sondern mit beliebigen Farben der BGR-Palette arbeiten können.

BColor (Eigenschaft)

Datentyp: **Long** (SmoColor)

Liest oder setzt die Hintergrundfarbe der Schrift als "BGR"-Wert (Blau-Grün-Rot-Triplet). Sie können entweder einen beliebigen Wert angeben oder eine der vordefinierten BGR-Farbkonstanten verwenden.

Lesen Sie die Eigenschaft aus und die Zellen sind in unterschiedlichen Farben formatiert, wird **smoUndefined** zurückgeliefert.

BColorIndex (Eigenschaft)

Datentyp: **Long** (SmoColorIndex)

Liest oder setzt die Hintergrundfarbe der Schrift als Indexfarbe. "Indexfarben" sind die Standardfarben von PlanMaker, durchnummeriert von -1 für Transparent bis 15 für Hellgrau. Sie dürfen ausschließlich einen der in der Tabelle der Indexfarben genannten Werte verwenden.

Falls die Zellen in unterschiedlichen Farben oder in einer anderen als einer der Indexfarben formatiert sind, wird beim Auslesen **smoUndefined** zurückgeliefert.

Wir empfehlen, stattdessen die Eigenschaft **BColor** (siehe dort) zu verwenden, da Sie mit dieser nicht auf diese Standardfarben beschränkt sind, sondern beliebige Farben der BGR-Palette wählen können.

Spacing (Eigenschaft)

Datentyp: **Long**

Liest oder setzt den Zeichenabstand. Der Standardwert ist 100 für einen normalen Zeichenabstand (100%).

Lesen Sie die Eigenschaft aus und die Zellen sind in unterschiedlichen Zeichenabständen formatiert, wird **smoUndefined** zurückgeliefert.

Pitch (Eigenschaft)

Datentyp: **Long**

Liest oder setzt die Zeichenbreite. Der Standardwert ist 100 für normalbreite Zeichen (100%).

Lesen Sie die Eigenschaft aus und die Zellen sind in unterschiedlichen Zeichenbreiten formatiert, wird **smoUndefined** zurückgeliefert.

Beachten Sie bitte, dass manche Drucker die Änderung der Zeichenbreite bei *druckerinternen* Schriften ignorieren.

Application (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das **Application**-Objekt.

Parent (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das übergeordnete Objekt, also ein Objekt, das entweder vom Typ **Range** oder vom Typ **FormatCondition** ist.

Beispiel für das Font-Objekt

Im nachfolgenden Beispiel wird der Bereich A1 bis C3 auf die Schrift Times New Roman fett mit 24 Punkt gesetzt.

```
Sub Main
  Dim pm as Object

  Set pm = CreateObject("PlanMaker.Application")
  pm.Visible = True

  With pm.ActiveSheet.Range("A1:C3")
     .Font.Name = "Times New Roman"
     .Font.Size = 24
     .Font.Bold = True
  End With
```

```
  Set pm = Nothing
End Sub
```

Borders (Sammlung)

Zugriffspfade für die direkte Formatierung:
- Application → Workbooks → Item → Sheets → Item → Range → **Borders**
- Application → Workbooks → ActiveSheet → Range → **Borders**
- Application → ActiveWorkbook → ActiveSheet → Range → **Borders**
- Application → ActiveSheet → Range → **Borders**

Zugriffspfade für die bedingte Formatierung:
- Application → Workbooks → Item → Sheets → Item → Range → FormatConditions → Item→ **Borders**
- Application → Workbooks → ActiveSheet → Range → FormatConditions → Item→ **Borders**
- Application → ActiveWorkbook → ActiveSheet → Range → FormatConditions → Item→ **Borders**
- Application → ActiveSheet → Range → FormatConditions → Item→ **Borders**

Statt **Range** können Sie auch alle anderen Objekte und Eigenschaften angeben, die ein **Range**-Objekt zurückgeben: **ActiveCell**, **Selection**, **Rows(n)**, **Columns(n)** und **Cells(x, y)**. Beispiele für diese Zugriffspfade finden Sie beim **Range**-Objekt.

1 Beschreibung

Borders ist eine Sammlung, die die vier Umrandungslinien (links, rechts, oben und unten) von Zellen repräsentiert. Sie erlaubt Ihnen, die Formatierung von jeweils einer der Umrandungslinien (zum Beispiel Liniendicke und -farbe) auszulesen und zu ändern.

Die einzelnen Elemente der **Borders**-Sammlung sind vom Typ **Border**.

Geben Sie als Parameter von **Borders** an, *welche* Linie verändert werden soll. Zulässig sind folgende Werte:

```
pmBorderTop = -1 ' Linie oberhalb der Zellen
pmBorderLeft  = -2 ' Linie links der Zellen
pmBorderBottom = -3 ' Linie unterhalb der Zellen
pmBorderRight = -4 ' Linie rechts der Zellen
pmBorderHorizontal  = -5 ' Horizontale Gitternetzlinien
pmBorderVertical  = -6 ' Vertikale Gitternetzlinien
```

Beispiel:

```
' Die Farbe der linken Linie von Zelle A1 auf Rot setzen
pm.ActiveSheet.Range("A1").Borders(pmBorderLeft).Color = smoColorRed
```

2 Zugriff auf das Objekt

Borders kann ein Tochterobjekt zweier Objekte sein:

- Als Tochterobjekt eines **Range**-Objekts repräsentiert es die Umrandungen der *Zellen* in diesem Bereich, entsprechend dem PlanMaker-Befehl **Format** > **Umrandung**.
- Als Tochterobjekt eines **FormatCondition**-Objekts repräsentiert es die Umrandungen, die durch die *bedingte Formatierung* ausgelöst werden.

Beispiele:

```
' Zelle A1 unten umranden
pm.ActiveSheet.Range("A1").Borders(pmBorderBottom).Type = pmLineStyleSingle

' Die linke Umrandung in der ersten bedingten Formatierung in A1 anzeigen
MsgBox pm.ActiveSheet.Range("A1").FormatConditions(1).Borders(pmBorderLeft).Type
```

Eigenschaften:
- **Count** R/O

Objekte:
- ***Item*** → **Border** (Defaultobjekt)
- **Application** → **Application**
- **Parent** → **Range** oder **FormatCondition**

Count (Eigenschaft, R/O)

Datentyp: **Long**

Liefert die Anzahl der **Border**-Objekte in der Sammlung, also die Zahl der möglichen Umrandungslinien. Dieser Wert ist stets 4, da es vier Umrandungen (links, rechts, oben und unten) gibt.

Item (Zeiger auf Objekt)

Datentyp: **Object**

Liefert ein einzelnes **Border**-Objekt, mit dem Sie eine einzelne Umrandungslinie ansprechen können, um deren Eigenschaften (etwa Farbe und Dicke) auszulesen oder zu setzen.

Welches Border-Objekt Sie erhalten, hängt von dem Zahlenwert ab, den Sie an **Item** übergeben. Die folgende Tabelle zeigt die erlaubten Werte:

```
pmBorderTop = -1 ' Linie oberhalb der Zellen
pmBorderLeft  = -2 ' Linie links der Zellen
pmBorderBottom = -3 ' Linie unterhalb der Zellen
pmBorderRight = -4 ' Linie rechts der Zellen
pmBorderHorizontal  = -5 ' Horizontale Gitternetzlinien
pmBorderVertical  = -6 ' Vertikale Gitternetzlinien
```

Application (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das **Application**-Objekt.

Parent (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das übergeordnete Objekt, also ein Objekt, das entweder vom Typ **Range** oder vom Typ **FormatCondition** ist.

Beispiel für das Borders-Objekt

Im nachfolgenden Beispiel wird der Bereich B2 bis D4 links mit einer 4 pt dicken blauen Linie und rechts mit einer doppelten roten Linie (jeweils 1 pt dick) versehen.

```
Sub Main
  Dim pm as Object

  Set pm = CreateObject("PlanMaker.Application")
  pm.Visible = True

  With pm.ActiveSheet.Range("B2:D4")
```

```
   .Borders(pmBorderLeft).Type    = pmLineStyleSingle
   .Borders(pmBorderLeft).Thick1  = 4
   .Borders(pmBorderLeft).Color   = pmColorBlue
   .Borders(pmBorderRight).Type   = pmLineStyleDouble
   .Borders(pmBorderRight).Thick1 = 1
   .Borders(pmBorderRight).Thick2 = 1
   .Borders(pmBorderRight).Color  = smoColorRed
  End With

  Set pm = Nothing
End Sub
```

Border (Objekt)

Zugriffspfade für die direkte Formatierung:
- Application → Workbooks → Item → Sheets → Item → Range → Borders → **Item**
- Application → Workbooks → ActiveSheet → Range → Borders → **Item**
- Application → ActiveWorkbook → ActiveSheet → Range → Borders → **Item**
- Application → ActiveSheet → Range → Borders → **Item**

Zugriffspfade für die bedingte Formatierung:
- Application → Workbooks → Item → Sheets → Item → Range → FormatConditions → Item→ Borders → **Item**
- Application → Workbooks → ActiveSheet → Range → FormatConditions → Item→ Borders → **Item**
- Application → ActiveWorkbook → ActiveSheet → Range → FormatConditions → Item→ Borders → **Item**
- Application → ActiveSheet → Range → FormatConditions → Item→ Borders → **Item**

Statt **Range** können Sie auch alle anderen Objekte und Eigenschaften angeben, die ein **Range**-Objekt zurückgeben: **ActiveCell**, **Selection**, **Rows(n)**, **Columns(n)** und **Cells(x, y)**. Beispiele für diese Zugriffspfade finden Sie beim **Range**-Objekt.

1 Beschreibung

Ein **Border**-Objekt repräsentiert jeweils eine der Umrandungslinien von Zellen (z.B. die obere, untere, linke oder rechte Linie). Es lässt Sie die Liniendicke, Farbe etc. dieser Umrandungslinie auslesen und setzen.

2 Zugriff auf das Objekt

Die einzelnen **Border**-Objekte können ausschließlich über die **Borders**-Sammlung angesprochen werden. Diese kann ein Tochterobjekt folgender zwei Objekte sein:

- Als Tochterobjekt eines **Range**-Objekts repräsentiert **Borders.Item(n)** eine Umrandungslinie der *Zellen* in diesem Bereich, entsprechend dem PlanMaker-Befehl **Format > Umrandung**.
- Als Tochterobjekt eines **FormatCondition**-Objekts repräsentiert **Borders.Item(n)** eine Umrandungslinie, die durch die *bedingte Formatierung* ausgelöst wird.

Um festzulegen, welche der Linien einer **Borders**-Sammlung Sie ansprechen möchten (links, rechts, oben, unten etc.), übergeben Sie als Parameter die Nummer dieser Linie (oder die entsprechende Konstante) gemäß der folgenden Tabelle:

```
pmBorderTop = -1 ' Linie oberhalb der Zellen
pmBorderLeft  = -2 ' Linie links der Zellen
pmBorderBottom = -3 ' Linie unterhalb der Zellen
pmBorderRight = -4 ' Linie rechts der Zellen
pmBorderHorizontal  = -5 ' Horizontale Gitternetzlinien
pmBorderVertical  = -6 ' Vertikale Gitternetzlinien
```

Beispiele:

```
' Zelle A1 unten umranden
pm.ActiveSheet.Range("A1").Borders(pmBorderBottom).Type = pmLineStyleSingle

' Die linke Umrandung in der ersten bedingten Formatierung in A1 anzeigen
MsgBox pm.ActiveSheet.Range("A1").FormatConditions(1).Borders(pmBorderLeft).Type
```

Eigenschaften:
- ***Type*** (Defaulteigenschaft)
- **Thick1**
- **Thick2**
- **Separator**
- **Color**
- **ColorIndex**

Objekte:
- **Application → Application**
- **Parent → Borders**

Type (Eigenschaft)

Datentyp: **Long** (PmLineStyle)

Liest oder setzt den Typ der Umrandungslinie. Mögliche Werte:

```
pmLineStyleNone   = 0 ' Keine Linie
pmLineStyleSingle = 1 ' Einfache Linie
pmLineStyleDouble = 2 ' Doppelte Linie
```

Thick1 (Eigenschaft)

Datentyp: **Single**

Liest oder setzt die Dicke der ersten Umrandungslinie in Punkt (1 Punkt entspricht 1/72 Zoll).

Thick2 (Eigenschaft)

Datentyp: **Single**

Liest oder setzt die Dicke der zweiten Umrandungslinie in Punkt (1 Punkt entspricht 1/72 Zoll).

Diese Eigenschaft wird nur verwendet, wenn der Typ der Umrandung auf **pmLineStyleDouble** steht.

Thick1, **Thick2** und **Separator** dürfen zusammen nicht größer als 12 sein.

Separator (Eigenschaft)

Datentyp: **Single**

Liest oder setzt den Abstand zwischen den beiden Umrandungslinien in Punkt (1 Punkt entspricht 1/72 Zoll).

Diese Eigenschaft wird nur verwendet, wenn der Typ der Umrandung auf **pmLineStyleDouble** steht.

Thick1, **Thick2** und **Separator** dürfen zusammen nicht größer als 12 pt. sein.

Color (Eigenschaft)

Datentyp: **Long** (SmoColor)

Liest oder setzt die Farbe der Umrandungslinie(n) als "BGR"-Wert (Blau-Grün-Rot-Triplet). Sie können entweder einen beliebigen Wert angeben oder eine der vordefinierten BGR-Farbkonstanten verwenden.

ColorIndex (Eigenschaft)

Datentyp: **Long** (SmoColorIndex)

Liest oder setzt die Farbe der Umrandungslinie(n) als Indexfarbe. "Indexfarben" sind die Standardfarben von PlanMaker, durchnummeriert von 0 für Schwarz bis 15 für Hellgrau. Sie dürfen ausschließlich einen der in der Tabelle der Indexfarben genannten Werte verwenden.

Wir empfehlen, stattdessen die Eigenschaft **Color** (siehe dort) zu verwenden, da Sie mit dieser nicht auf die 16 Standardfarben beschränkt sind, sondern mit beliebigen Farben der BGR-Palette arbeiten können.

Application (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das **Application**-Objekt.

Parent (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das übergeordnete Objekt, also ein Objekt von Typ **Borders**.

Shading (Objekt)

Zugriffspfade für die direkte Formatierung:
- Application → Workbooks → Item → Sheets → Item → Range → **Shading**
- Application → Workbooks → ActiveSheet → Range → **Shading**
- Application → ActiveWorkbook → ActiveSheet → Range → **Shading**
- Application → ActiveSheet → Range → **Shading**

Zugriffspfade für die bedingte Formatierung:
- Application → Workbooks → Item → Sheets → Item → Range → FormatConditions → Item→ **Shading**
- Application → Workbooks → ActiveSheet → Range → FormatConditions → Item→ **Shading**
- Application → ActiveWorkbook → ActiveSheet → Range → FormatConditions → Item→ **Shading**
- Application → ActiveSheet → Range → FormatConditions → Item→ **Shading**

Statt "Range" können Sie auch alle anderen Objekte und Eigenschaften angeben, die ein **Range**-Objekt zurückgeben: **ActiveCell**, **Selection**, **Rows(n)**, **Columns(n)** und **Cells(x, y)**. Beispiele für diese Zugriffspfade finden Sie beim **Range**-Objekt.

1 Beschreibung

Das **Shading**-Objekt beschreibt die Schattierung von Zellen (entweder mit einem Raster oder einem Muster).

2 Zugriff auf das Objekt

Shading kann ein Tochterobjekt zweier Objekte sein:

- Als Tochterobjekt eines **Range**-Objekts repräsentiert es die Schattierung der *Zellen* in diesem Bereich, entsprechend dem PlanMaker-Befehl **Format** > **Zelle** > **Schattierung**.
- Als Tochterobjekt eines **FormatCondition**-Objekts repräsentiert es die Schattierung, die durch die *bedingte Formatierung* ausgelöst wird.

Beispiele:

```
' Die Musterung in Zelle A1 anzeigen
MsgBox pm.ActiveSheet.Range("A1").Shading.Texture

' Die Musterung in der ersten bedingten Formatierung in A1 anzeigen
```

```
MsgBox pm.ActiveSheet.Range("A1").FormatConditions(1).Shading.Texture
```

3 Eigenschaften, Objekte, Sammlungen und Methoden

Eigenschaften:
- **Texture**
- **Intensity**
- ***ForegroundPatternColor*** (Defaulteigenschaft)
- **ForegroundPatternColorIndex**
- **BackgroundPatternColor**
- **BackgroundPatternColorIndex**

Objekte:
- **Application** → **Application**
- **Parent** → **Range** oder **FormatCondition**

Texture (Eigenschaft)

Datentyp: **Long** (SmoShadePatterns)

Liest oder setzt die Art des Musters. Mögliche Werte:

```
smoPatternNone            = 0   (Kein Muster)
smoPatternHalftone        = 1   (Raster)
smoPatternRightDiagCoarse = 2
smoPatternLeftDiagCoarse  = 3
smoPatternHashDiagCoarse  = 4
smoPatternVertCoarse      = 5
smoPatternHorzCoarse      = 6
smoPatternHashCoarse      = 7
smoPatternRightDiagFine   = 8
smoPatternLeftDiagFine    = 9
smoPatternHashDiagFine    = 10
smoPatternVertFine        = 11
smoPatternHorzFine        = 12
smoPatternHashFine        = 13
```

Um eine *Schattierung* hinzuzufügen, setzen Sie **Texture** auf **smoPatternHalftone** und geben die gewünschte Stärke der Schattierung bei **Intensity** an.

Um ein *Muster* hinzuzufügen, setzen Sie **Texture** auf einen Wert zwischen **smoPatternRightDiagCoarse** und **smoPatternHashFine**.

Um eine Schattierung oder ein Muster wieder zu *entfernen*, setzen Sie **Texture** auf **smoPatternNone**.

Intensity (Eigenschaft)

Datentyp: **Long**

Liest oder setzt die Rasterstärke in Prozent.

Zulässig ist ein Wert zwischen 0 (nicht sichtbar) und 100 (volle Farbtönung).

Dieser Wert darf nur gesetzt oder gelesen werden, wenn mittels **Texture**-Eigenschaft eine Schattierung angewählt wurde (**Texture** auf **smoPatternHalftone** gesetzt). Ist ein Muster gewählt (**Texture** enthält einen beliebigen anderen Wert), führt der Zugriff auf **Intensity** zu einem Fehler.

ForegroundPatternColor (Eigenschaft)

Datentyp: **Long** (SmoColor)

Liest oder setzt die Vordergrundfarbe des Musters oder der Schattierung als "BGR"-Wert (Blau-Grün-Rot-Triplet). Sie können entweder einen beliebigen Wert angeben oder eine der vordefinierten BGR-Farbkonstanten verwenden.

ForegroundPatternColorIndex (Eigenschaft)

Datentyp: **Long** (SmoColorIndex)

Liest oder setzt die Vordergrundfarbe des Musters oder der Schattierung als Indexfarbe. "Indexfarben" sind die Standardfarben von PlanMaker, durchnummeriert von 0 für Schwarz bis 15 für Hellgrau. Sie dürfen ausschließlich einen der in der Tabelle der Indexfarben genannten Werte verwenden.

Wir empfehlen, stattdessen die Eigenschaft **ForegroundPatternColor** (siehe dort) zu verwenden, da Sie mit dieser nicht auf die 16 Standardfarben beschränkt sind, sondern mit beliebigen Farben der BGR-Palette arbeiten können.

BackgroundPatternColor (Eigenschaft)

Datentyp: **Long** (SmoColor)

Liest oder setzt die Hintergrundfarbe des Musters oder der Schattierung als "BGR"-Wert (Blau-Grün-Rot-Triplet). Sie können entweder einen beliebigen Wert angeben oder eine der vordefinierten BGR-Farbkonstanten verwenden.

BackgroundPatternColorIndex (Eigenschaft)

Datentyp: **Long** (SmoColorIndex)

Liest oder setzt die Hintergrundfarbe des Musters oder der Schattierung als Indexfarbe. "Indexfarben" sind die Standardfarben von PlanMaker, durchnummeriert von 0 für Schwarz bis 15 für Hellgrau. Sie dürfen ausschließlich einen der in der Tabelle der Indexfarben genannten Werte verwenden.

Wir empfehlen, stattdessen die Eigenschaft **BackgroundPatternColor** (siehe dort) zu verwenden, da Sie mit dieser nicht auf die 16 Standardfarben beschränkt sind, sondern beliebige Farben der BGR-Palette wählen können.

Application (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das **Application**-Objekt.

Parent (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das übergeordnete Objekt, also ein Objekt, das entweder vom Typ **Range** oder vom Typ **FormatCondition** ist.

Beispiel für das Shading-Objekt

Im nachfolgenden Beispiel wird der Bereich A1 bis C3 mit einem 50%-igen roten Raster versehen.

```
Sub Main
```

```
  Dim pm as Object

  Set pm = CreateObject("PlanMaker.Application")
  pm.Visible = True

  With pm.ActiveSheet.Range("A1:C3")
     .Shading.Intensity = 50
     .Shading.ForegroundPatternColor = smoColorRed
  End With

  Set pm = Nothing
End Sub
```

Validation (Objekt)

Zugriffspfade:
- Application → Workbooks → Item → Sheets → Item → Range → **Validation**
- Application → Workbooks → ActiveSheet → Range → **Validation**
- Application → ActiveWorkbook → ActiveSheet → Range → **Validation**
- Application → ActiveSheet → Range → **Validation**

Statt "Range" können Sie auch alle anderen Objekte und Eigenschaften angeben, die ein **Range**-Objekt zurückgeben: **ActiveCell**, **Selection**, **Rows(n)**, **Columns(n)** und **Cells(x, y)**. Beispiele für diese Zugriffspfade finden Sie beim **Range**-Objekt.

1 Beschreibung

Das **Validation**-Objekt repräsentiert die Gültigkeitsprüfung eines Bereichs (also eines **Range**-Objekts). In PlanMaker können solche Gültigkeitsprüfungen mit dem Befehl **Format > Gültigkeitsprüfung** eingerichtet werden.

2 Zugriff auf das Objekt

Es existiert für jedes **Range**-Objekt genau eine Instanz des **Validation**-Objekts. Diese wird über **Range.Validation** angesprochen:

```
' Die Eingabemeldung der Zelle A1 anzeigen (sofern zugewiesen)
MsgBox pm.ActiveSheet.Range("A1").Validation.InputMessage
```

3 Eigenschaften, Objekte, Sammlungen und Methoden

Eigenschaften:
- **Type** R/O
- **AlertStyle**
- **Value** R/O
- **ShowInput**
- **InputTitle**
- **InputMessage**
- **ShowError**
- **ErrorTitle**
- **ErrorMessage**
- **Operator** R/O
- **Formula1** R/O
- **Formula2** R/O
- **InCellDropDown**
- **IgnoreBlank**

Objekte:
- **Application → Application**
- **Parent → Range**

Methoden:

- **Add**
- **Modify**
- **Delete**

Type (Eigenschaft, R/O)

Datentyp: **Long** (PmDVType)

Liest oder setzt den Typ der Werte, die als "erlaubt" angesehen werden sollen. Mögliche Werte:

```
pmValidateInputOnly      = 0 ' Alle Arten von Werten erlauben
pmValidateWholeNumber    = 1 ' Nur ganze Zahlen erlauben
pmValidateDecimal        = 2 ' Nur Dezimalzahlen erlauben
pmValidateList           = 3 ' Nur feste Listeneinträge erlauben
pmValidateDate           = 4 ' Nur Datumswerte erlauben
pmValidateTime           = 5 ' Nur Zeitwerte erlauben
pmValidateTextLength     = 6 ' Zulässige Länge für Einträge begrenzen
pmValidateCustom         = 7 ' Benutzerdefinierte Prüfung
```

AlertStyle (Eigenschaft)

Datentyp: **Long** (PmDVAlertStyle)

Liest oder setzt den Stil der Gültigkeitswarnung.

```
pmValidAlertStop           = 0 ' Fehlermeldung
pmValidAlertWarning        = 1 ' Warnung
pmValidAlertInformation    = 2 ' Information
```

Value (Eigenschaft, R/O)

Datentyp: **Boolean**

Liefert **True**, wenn der Bereich gültige (= die Gültigkeitsprüfung bestehende) Daten enthält, ansonsten **False**.

ShowInput (Eigenschaft)

Datentyp: **Long**

Liest oder setzt die Einstellung, ob beim Aktivieren einer Zelle eine Eingabemeldung erscheinen soll. Entspricht der Einstellung "Eingabemeldung anzeigen, wenn die Zelle markiert ist" im Dialogfenster des Befehls **Format > Gültigkeitsprüfung > Eingabemeldung**.

InputTitle (Eigenschaft)

Datentyp: **String**

Liest oder setzt den Titel der Eingabemeldung, die erscheint, sobald eine Zelle mit Eingabeprüfung aktiviert wird. Entspricht dem Eingabefeld "Titel" im Dialogfenster des Befehls **Format > Gültigkeitsprüfung > Eingabemeldung**.

InputMessage (Eigenschaft)

Datentyp: **String**

Liest oder setzt den Text der Eingabemeldung, die erscheint, sobald eine Zelle mit Eingabeprüfung aktiviert wird. Entspricht dem Eingabefeld "Meldung" im Dialogfenster des Befehls **Format > Gültigkeitsprüfung > Eingabemeldung**.

ShowError (Eigenschaft)

Datentyp: **Long**

Liest oder setzt die Einstellung, ob bei Eingabe von Werten, die die Gültigkeitsprüfung nicht bestehen, eine Fehlermeldung erscheinen soll. Entspricht der Einstellung "Fehlermeldung anzeigen, wenn ungültige Daten eingegeben wurden" im Dialogfenster des Befehls **Format > Gültigkeitsprüfung > Fehlermeldung**.

ErrorTitle (Eigenschaft)

Datentyp: **String**

Liest oder setzt den Text der Fehlermeldung, die erscheint, sobald ein ungültiger Wert eingegeben wird. Entspricht dem Eingabefeld "Titel" im Dialogfenster des Befehls **Format > Gültigkeitsprüfung > Fehlermeldung**.

ErrorMessage (Eigenschaft)

Datentyp: **String**

Liest oder setzt den Text der Fehlermeldung, die erscheint, sobald ein ungültiger Wert eingegeben wird. Entspricht dem Eingabefeld "Meldung" im Dialogfenster des Befehls **Format > Gültigkeitsprüfung > Fehlermeldung**.

Operator (Eigenschaft, R/O)

Datentyp: **Long** (PmDVOperator)

Liest oder setzt den Vergleichsoperator, der bei der Gültigkeitsprüfung verwendet wird.

```
pmDVBetween         = 0 ' zwischen
pmDVNotBetween      = 1 ' nicht zwischen
pmDVEqual           = 2 ' gleich
pmDVNotEqual        = 3 ' ungleich
pmDVGreater         = 4 ' größer als
pmDVLess            = 5 ' kleiner als
pmDVGreaterEqual    = 6 ' größer oder gleich
pmDVLessEqual       = 7 ' kleiner oder gleich
```

Formula1 (Eigenschaft, R/O)

Datentyp: **String**

Liefert bei den Operatoren **pmDVBetween** und **pmDVNotBetween** das Minimum der Gültigkeitsprüfung, bei allen anderen Operatoren den Wert.

Formula2 (Eigenschaft, R/O)

Datentyp: **String**

Liefert bei den Operatoren **pmDVBetween** und **pmDVNotBetween** das Maximum der Gültigkeitsprüfung, bei allen anderen Operatoren ist der Rückgabewert leer.

InCellDropDown (Eigenschaft)

Datentyp: **Long**

Liest oder setzt die Einstellung, ob eine Auswahlliste mit den zulässigen Werten in der Zelle angezeigt werden soll. Nur anwendbar, wenn der Typ der Gültigkeitsprüfung (siehe **Type**-Eigenschaft weiter oben) auf "Feste Listeneinträge" (**pmValidateList**) gesetzt wurde.

Entspricht der Option "Auswahlliste verwenden" im Dialogfenster des Befehls **Format > Gültigkeitsprüfung**.

IgnoreBlank (Eigenschaft)

Datentyp: **Long**

Liest oder setzt die Einstellung, ob leere Zellen bei der Gültigkeitsprüfung ignoriert werden. Entspricht der Einstellung "Leere Zellen ignorieren" im Dialogfenster des Befehls **Format > Gültigkeitsprüfung**.

Application (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das **Application**-Objekt.

Parent (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das übergeordnete Objekt, also ein Objekt vom Typ **Range**.

Add (Methode)

Richtet in einem Bereich eine neue Gültigkeitsprüfung ein. Entspricht dem Befehl **Format > Gültigkeitsprüfung**.

Beachten Sie bitte, dass jede Zelle nur genau *eine* Gültigkeitsprüfung besitzen darf.

Syntax:

```
Add Type, [AlertStyle], [Operator], [Formula1], [Formula2]
```

Parameter:

Type (Typ: **Long** bzw. **PmDVType**) legt den Typ der Gültigkeitsprüfung fest. Mögliche Werte:

```
pmValidateInputOnly      = 0 ' Alle Werte *
pmValidateWholeNumber    = 1 ' Ganze Zahlen
pmValidateDecimal        = 2 ' Dezimalzahlen
pmValidateList           = 3 ' Feste Listeneinträge **
pmValidateDate           = 4 ' Datumswerte
pmValidateTime           = 5 ' Zeitwerte
pmValidateTextLength     = 6 ' Textlänge
pmValidateCustom         = 7 ' Benutzerdefiniert ***
```

* Bei dieser Einstellung werden alle Werte akzeptiert. Verwenden Sie diese Einstellung, wenn Sie lediglich eine Meldung anzeigen wollen, sobald der Benutzer die betroffene(n) Zelle(n) selektiert.

** Bei dieser Einstellung werden nur Werte aus einer Liste akzeptiert. Geben Sie bei **Formula1** den Bereich im Arbeitsblatt an, der die erlaubten Werte enthält. Wenn beispielsweise die Zellen C1 bis C3 die Werte "Hund", "Katze", "Maus" enthalten und Sie bei **Formula1** C1:C3 angeben, kann der Benutzer nur diese Werte angeben.

*** Bei dieser Einstellung müssen Sie bei **Formula1** einen Ausdruck angeben, der bei gültiger Dateneingabe **True**, ansonsten **False** liefert.

AlertStyle (Typ: **Long** bzw. **PmDVAlertStyle**) legt den Stil der Fehlermeldung für ungültige Werte fest:

```
pmValidAlertStop         = 0 ' Fehlermeldung
pmValidAlertWarning      = 1 ' Warnung
pmValidAlertInformation  = 2 ' Information
```

Operator (Typ: **Long** bzw. **PmDVOperator**) legt den Vergleichsoperator fest, der bei der Gültigkeitsprüfung verwendet werden soll:

```
pmDVBetween         = 0 ' zwischen
pmDVNotBetween      = 1 ' nicht zwischen
pmDVEqual           = 2 ' gleich
pmDVNotEqual        = 3 ' ungleich
pmDVGreater         = 4 ' größer als
pmDVLess            = 5 ' kleiner als
pmDVGreaterEqual    = 6 ' größer oder gleich
pmDVLessEqual       = 7 ' kleiner oder gleich
```

Formula1 (optional; Typ: **String**) legt einen String fest, der eine Zahl, einen Zellbezug oder eine Formel enthält. Bei **pmDVBetween** und **pmDVNotBetween** legen Sie damit das Minimum fest, bei allen anderen Operatoren den Wert.

Formula2 (optional; Typ: **String**) legt einen String fest, der eine Zahl, einen Zellbezug oder eine Formel enthält. Darf nur bei **pmDVBetween** und **pmDVNotBetween** angegeben werden.

Rückgabetyp:

keiner

Übersicht über die Kombinationsmöglichkeiten der Parameter:

Type	Operator	Formula1	Formula2
pmValidateInputOnly	(Nicht verwendet)	(Nicht verwendet)	(Nicht verwendet)
pmValidateWholeNumber, pmValidateDecimal, pmValidateDate, pmValidateTime, pmValidateTextLength	Alle oben genannten	Enthält bei **pmDVBetween** und **pmDVNotBetween** das Minimum, bei allen anderen Operatoren den Wert.	Darf nur bei **pmDVBetween** und **pmDVNotBetween** verwendet werden und enthält dann das Maximum.
pmValidateList	(Nicht verwendet)	Enthält entweder eine durch den System-Listentrenner (in Deutschland: Strichpunkte) getrennte Liste oder einen Verweis auf einen Zellbereich.	(Nicht verwendet)
pmValidateCustom	(Nicht verwendet)	Muss einen Ausdruck enthalten, der bei gültiger Dateneingabe **True** liefert, ansonsten **False**.	(Nicht verwendet)

Modify (Methode)

Modifiziert die Gültigkeitsprüfung für einen Bereich.

Syntax:

```
Modify [Type], [AlertStyle], [Operator], [Formula1], [Formula2]
```

Parameter:

Type (optional; Typ: **Long** bzw. **PmDVType**) legt den Typ der Gültigkeitsprüfung fest. Mögliche Werte:

```
pmValidateInputOnly         = 0 ' Alle Werte *
pmValidateWholeNumber       = 1 ' Ganze Zahlen
pmValidateDecimal           = 2 ' Dezimalzahlen
pmValidateList              = 3 ' Feste Listeneinträge **
pmValidateDate              = 4 ' Datumswerte
pmValidateTime              = 5 ' Zeitwerte
pmValidateTextLength        = 6 ' Textlänge
pmValidateCustom            = 7 ' Benutzerdefiniert ***
```

* Bei dieser Einstellung werden alle Werte akzeptiert. Verwenden Sie diese Einstellung, wenn Sie lediglich eine Meldung anzeigen wollen, sobald der Benutzer die betroffene(n) Zelle(n) selektiert.

** Bei dieser Einstellung werden nur Werte aus einer Liste akzeptiert. Geben Sie bei **Formula1** den Bereich im Arbeitsblatt an, der die erlaubten Werte enthält. Wenn beispielsweise die Zellen C1 bis C3 die Werte "Hund", "Katze", "Maus" enthalten und Sie bei **Formula1** C1:C3 angeben, kann der Benutzer nur diese Werte angeben.

*** Bei dieser Einstellung müssen Sie bei **Formula1** einen Ausdruck angeben, der bei gültiger Dateneingabe **True**, ansonsten **False** liefert.

AlertStyle (Typ: **Long** bzw. **PmDVAlertStyle**) legt den Stil der Gültigkeitswarnung fest:

```
pmValidAlertStop          = 0 ' Fehlermeldung
pmValidAlertWarning       = 1 ' Warnung
pmValidAlertInformation   = 2 ' Information
```

Operator (Typ: **Long** bzw. **PmDVOperator**)

```
pmDVBetween         = 0 ' zwischen
pmDVNotBetween      = 1 ' nicht zwischen
pmDVEqual           = 2 ' gleich
pmDVNotEqual        = 3 ' ungleich
pmDVGreater         = 4 ' größer als
pmDVLess            = 5 ' kleiner als
pmDVGreaterEqual    = 6 ' größer oder gleich
pmDVLessEqual       = 7 ' kleiner oder gleich
```

Formula1 (optional; Typ: **String**) legt einen String fest, der eine Zahl, einen Zellbezug oder eine Formel enthält. Bei **pmDVBetween** und **pmDVNotBetween** legen Sie damit das Minimum fest, bei allen anderen Operatoren den Wert.

Formula2 (optional; Typ: **String**) legt einen String fest, der eine Zahl, einen Zellbezug oder eine Formel enthält. Darf nur bei **pmDVBetween** und **pmDVNotBetween** angegeben werden.

Rückgabetyp:

keiner

Delete (Methode)

Entfernt die Gültigkeitsprüfung für einen Bereich.

Syntax:

```
Delete
```

Parameter:

keine

Rückgabetyp:

keiner

Beispiel:

```
' Gültigkeitsprüfung für die Zellen A1 und A2 entfernen
pm.Application.ActiveSheet.Range("A1:A2").Validation.Delete
```

AutoFilter (Objekt)

Zugriffspfade:
- Application → Workbooks → Item → Sheets → Item → **AutoFilter**
- Application → Workbooks → ActiveSheet → **AutoFilter**
- Application → ActiveWorkbook → ActiveSheet → **AutoFilter**
- Application → ActiveSheet → **AutoFilter**

1 Beschreibung

Das **AutoFilter**-Objekt lässt Sie auf den AutoFilter des Arbeitsblatts zugreifen. In PlanMaker können solche Filter mit dem Befehl **Tabelle > Filter > Autofilter** eingerichtet werden.

2 Zugriff auf das Objekt

Jedes Arbeitsblatt (**Sheet**) besitzt genau ein **AutoFilter**-Objekt. Dieses wird über **Sheet.AutoFilter** angesprochen:

```
' Die Zahl der Spalten anzeigen, die der AutoFilter umfasst
MsgBox pm.ActiveSheet.AutoFilter.Filters.Count
```

3 Eigenschaften, Objekte, Sammlungen und Methoden

Objekte:
- **Application → Application**
- **Parent → Sheet**

Sammlungen:
- **Filters → Filters**

Application (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das **Application**-Objekt.

Parent (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das übergeordnete Objekt, also ein Objekt vom Typ **Sheet**.

Filters (Zeiger auf Sammlung)

Datentyp: **Object**

Liefert die **Filters**-Sammlung, über die Sie auf die einzelnen Spalten des AutoFilters zugreifen können.

Filters (Sammlung)

Zugriffspfade:
- Application → Workbooks → Item → Sheets → Item → AutoFilter → **Filters**
- Application → Workbooks → ActiveSheet → AutoFilter → **Filters**
- Application → ActiveWorkbook → ActiveSheet → AutoFilter → **Filters**
- Application → ActiveSheet → AutoFilter → **Filters**

1 Beschreibung

Die Sammlung **Filters** enthält alle Spalten des aktiven AutoFilters.

Die einzelnen Elemente dieser Sammlung sind vom Typ **Filter**. Sie können mit den einzelnen **Filter**-Objekten gezielt die Selektionskriterien und Operatoren einzelner Spalten des AutoFilters abfragen.

2 Zugriff auf die Sammlung

Jeder AutoFilter besitzt genau eine **Filters**-Sammlung. Diese wird über **AutoFilter.Filters** angesprochen:

```
' Die Zahl der Spalten zeigen, die der AutoFilter umfasst
MsgBox pm.ActiveSheet.AutoFilter.Filters.Count
```

3 Eigenschaften, Objekte, Sammlungen und Methoden

Eigenschaften:
- **Count** R/O

Objekte:
- ***Item*** → **Filter** (Defaultobjekt)
- **Application** → **Application**
- **Parent** → **AutoFilter**

Count (Eigenschaft, R/O)

Datentyp: **Long**

Liefert die Anzahl der **Filter**-Objekte in der Sammlung, also die Zahl der Spalten, die der aktive AutoFilter enthält.

Item (Zeiger auf Objekt)

Datentyp: **Object**

Liefert ein einzelnes **Filter**-Objekt, also eine einzelne Spalte des AutoFilters.

Welches Filter-Objekt Sie erhalten, hängt von dem Zahlenwert ab, den Sie an **Item** übergeben: 1 für die erste Spalte, 2 für die zweite etc.

Application (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das **Application**-Objekt.

Parent (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das übergeordnete Objekt, also ein Objekt vom Typ **AutoFilter**.

Filter (Objekt)

Zugriffspfade:
- Application → Workbooks → Item → Sheets → Item → AutoFilter → Filters → **Item**
- Application → Workbooks → ActiveSheet → AutoFilter → Filters → **Item**
- Application → ActiveWorkbook → ActiveSheet → AutoFilter → Filters → **Item**
- Application → ActiveSheet → AutoFilter → Filters → **Item**

1 Beschreibung

Ein **Filter**-Objekt repräsentiert eine einzelne Spalte des aktiven AutoFilters. Sie können es benutzen, um die Selektionskriterien und Filterbedingungen der jeweiligen Spalte abzufragen.

2 Zugriff auf das Objekt

Die einzelnen **Filter**-Objekte können ausschließlich durch Aufzählung der Elemente von Sammlungen des Typs **Filters** angesprochen werden.

Für jeden AutoFilter existiert genau eine Instanz dieser **Filters**-Sammlung, nämlich **AutoFilter.Filters**:

```
' Das Selektionskriterium der ersten Spalte des AutoFilters anzeigen
MsgBox pm.ActiveSheet.AutoFilter.Filters.Item(1).Criteria1
```

Beachten Sie bitte, dass alle Eigenschaften des **Filter**-Objekts schreibgeschützt sind. Zum Festlegen neuer AutoFilter-Eigenschaften benutzen Sie die Methode **AutoFilter** im **Range**-Objekt.

3 Eigenschaften, Objekte, Sammlungen und Methoden

Eigenschaften:
- **Operator** R/O
- **Criteria1** R/O
- **Criteria2** R/O

Objekte:
- **Application** → **Application**
- **Parent** → **Filters**

Operator (Eigenschaft, R/O)

Datentyp: **Long** (PmAutoFilterOperator)

Liefert den Typ der Filterbedingung. Mögliche Werte:

```
pmAll             = 0 ' Alle Zeilen anzeigen (= nicht mehr filtern)
pmAnd             = 1 ' Criteria1 und Criteria2 müssen zutreffen
pmBottom10Items   = 2 ' Nur die n Zeilen mit den niedrigsten Werten*
pmBottom10Percent = 3 ' Nur die n Prozent mit den niedrigsten Werten*
pmOr              = 4 ' Criteria1 oder Criteria2 muss zutreffen
pmTop10Items      = 5 ' Nur die n Zeilen mit den höchsten Werten*
pmTop10Percent    = 6 ' Nur die n Prozent mit den höchsten Werten*
pmBlank           = 7 ' Nur leere Zeilen anzeigen
pmNonblank        = 8 ' Nur nicht-leere Zeilen anzeigen
pmCustom          = 9 ' Benutzerdefinierter Filter
```

* Bei diesen Typen enthält **Criteria1** den Wert für "n".

Criteria1 (Eigenschaft, R/O)

Datentyp: **String**

Liefert den Filterbegriff – zum Beispiel "Rot", wenn Sie nach "Rot" gefiltert haben.

Ausnahme: Bei den Operatoren **pmTop10Items**, **pmTop10Percent**, **pmBottom10Items** und **pmBottom10Percent** erhalten Sie hier den Wert, *wie viele* Zeilen angezeigt werden sollen.

Criteria2 (Eigenschaft, R/O)

Datentyp: **String**

Liefert den zweiten Filterbegriff. Voraussetzung ist, dass der Operator auf **pmAnd** oder **pmOr** gesetzt ist, weil nur dann zwei Filterbegriffe möglich sind.

Application (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das **Application**-Objekt.

Parent (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das übergeordnete Objekt, also **Filters**.

Windows (Sammlung)

Zugriffspfad: Application → **Windows**

1 Beschreibung

Die Sammlung **Windows** enthält alle geöffneten Dokumentfenster. Die einzelnen Elemente dieser Sammlung sind vom Typ **Window**.

2 Zugriff auf die Sammlung

Es existiert während der gesamten Laufzeit von PlanMaker genau eine Instanz der **Windows**-Sammlung. Diese wird über **Application.Windows** angesprochen:

```
' Die Anzahl der offenen Dokumentfenster anzeigen
MsgBox pm.Application.Windows.Count

' Den Namen des ersten geöffneten Dokumentfensters anzeigen
MsgBox pm.Application.Windows(1).Name
```

3 Eigenschaften, Objekte, Sammlungen und Methoden

Eigenschaften:
- **Count** R/O

Objekte:
- ***Item*** → **Window** (Defaultobjekt)
- **Application** → **Application**
- **Parent** → **Application**

Count (Eigenschaft, R/O)

Datentyp: **Long**

Liefert die Anzahl der **Window**-Objekte in PlanMaker – in anderen Worten: die Anzahl der geöffneten Dokumentfenster.

Item (Zeiger auf Objekt)

Datentyp: **Object**

Liefert ein einzelnes **Window**-Objekt, also ein einzelnes Dokumentfenster.

Welches Window-Objekt Sie erhalten, hängt von dem Parameter ab, den Sie an **Item** übergeben. Dies kann entweder der numerische Index oder der Dateiname des gewünschten Dokumentfensters sein. Beispiele:

```
' Den Namen des ersten Dokumentfensters anzeigen
MsgBox pm.Application.Windows.Item(1).FullName

' Den Namen des Dokumentfensters "Test.pmd" anzeigen (sofern gerade geöffnet)
MsgBox pm.Application.Windows.Item("Test.pmd").FullName

' Sie können auch den kompletten Namen mit Pfadangabe verwenden
MsgBox pm.Application.Windows.Item("c:\Dokumente\Test.pmd").FullName
```

Application (Zeiger auf Objekt)

Liefert das **Application**-Objekt.

Parent (Zeiger auf Objekt)

Liefert das übergeordnete Objekt, also **Application**.

Window (Objekt)

Zugriffspfade:
- Application → Windows → **Item**
- Application → **ActiveWindow**
- Application → Workbooks → Item → **ActiveWindow**
- Application → ActiveWorkbook → **ActiveWindow**

1 Beschreibung

Ein **Window**-Objekt repräsentiert ein einzelnes in PlanMaker geöffnetes Dokumentfenster.

Für jedes Dokumentfenster existiert ein eigenes **Window**-Objekt. Öffnen oder schließen Sie Dokumentfenster, werden die zugehörigen **Window**-Objekte dynamisch angelegt bzw. entfernt.

2 Zugriff auf das Objekt

Die einzelnen **Window**-Objekte können auf folgenden Wegen angesprochen werden:

- Alle zu einem Zeitpunkt geöffneten Dokumente werden in der Sammlung **Application.Windows** (Typ: **Windows**) verwaltet:

```
' Die Namen aller geöffneten Dokumentfenster anzeigen
For i = 1 To pm.Application.Windows.Count
  MsgBox pm.Application.Windows.Item(i).Name
Next i
```

- Das aktive Dokumentfenster erhalten Sie über **Application.ActiveWindow**:

```
' Den Namen des aktuellen Dokumentfensters anzeigen
MsgBox pm.Application.ActiveWindow.Name
```

- Das Objekt **Workbook** enthält einen Objektzeiger auf das ihm zugehörige Dokumentfenster:

```
' Über das aktive Dokument an das aktive Dokumentfenster kommen
MsgBox pm.Application.ActiveWorkbook.ActiveWindow.Name
```

3 Eigenschaften, Objekte, Sammlungen und Methoden

- **FullName** R/O
- ***Name*** R/O
- **Path** R/O
- **Left**
- **Top**
- **Width**
- **Height**
- **WindowState**
- **DisplayFormulas**
- **DisplayVerticalScrollBar**
- **DisplayHorizontalScrollBar**
- **DisplayWorkbookTabs**
- **DisplayHeadings**
- **Zoom**
- **DisplayGridlines**
- **GridlineColor**
- **GridlineColorIndex**

Objekte:
- **Workbook → Workbook**
- **ActiveCell → Range**
- **ActiveSheet → Sheet**
- **Application → Application**
- **Parent → Windows**

Methoden:
- **Activate**
- **Close**

FullName (Eigenschaft, R/O)

Datentyp: **String**

Liefert den Pfad und Namen des in dem Fenster geöffneten Dokuments (z.B. c:\Kalkulation\Müller.pmd).

Name (Eigenschaft, R/O)

Datentyp: **String**

Liefert den Namen des in dem Fenster geöffneten Dokuments (z.B. Müller.pmd).

Path (Eigenschaft, R/O)

Datentyp: **String**

Liefert den Pfad des in dem Fenster geöffneten Dokuments (z.B. c:\Kalkulation).

Left (Eigenschaft)

Datentyp: **Long**

Liest oder setzt die X-Koordinate der Fensterposition. Die Maßeinheit sind Bildschirmpixel.

Top (Eigenschaft)

Datentyp: **Long**

Liest oder setzt die Y-Koordinate der Fensterposition. Die Maßeinheit sind Bildschirmpixel.

Width (Eigenschaft)

Datentyp: **Long**

Liest oder setzt die Breite des Dokumentfensters. Die Maßeinheit sind Bildschirmpixel.

Height (Eigenschaft)

Datentyp: **Long**

Liest oder setzt die Höhe des Dokumentfensters. Die Maßeinheit sind Bildschirmpixel.

WindowState (Eigenschaft)

Datentyp: **Long** (SmoWindowState)

Liest oder setzt die Fensterdarstellung des Dokumentfensters. Mögliche Werte:

```
smoWindowStateNormal   = 1 ' normal
smoWindowStateMinimize = 2 ' minimiert
smoWindowStateMaximize = 3 ' maximiert
```

DisplayFormulas (Eigenschaft)

Datentyp: **Boolean**

Liest oder setzt die Einstellung, ob in den Zellen statt der Ergebnisse die zugrundeliegenden Formeln angezeigt werden. Entspricht dem Befehl **Ansicht > Formelanzeige**.

DisplayVerticalScrollBar (Eigenschaft)

Datentyp: **Boolean**

Liest oder setzt die Einstellung, ob am rechten Rand des Dokumentfensters der vertikale Rollbalken angezeigt wird. Entspricht der Einstellung "Vertikale Bildlaufleiste" im Dialogfenster des Befehls **Datei > Eigenschaften > Optionen > Fenster**.

DisplayHorizontalScrollBar (Eigenschaft)

Datentyp: **Boolean**

Liest oder setzt die Einstellung, ob am unteren Rand des Dokumentfensters der horizontale Rollbalken angezeigt wird. Entspricht der Einstellung "Horizontale Bildlaufleiste" im Dialogfenster des Befehls **Datei > Eigenschaften > Optionen > Fenster**.

DisplayWorkbookTabs (Eigenschaft)

Datentyp: **Boolean**

Liest oder setzt die Einstellung, ob am unteren Rand des Dokumentfensters das Arbeitsblattregister angezeigt wird. Entspricht der Einstellung "Arbeitsblattregister" im Dialogfenster des Befehls **Datei > Eigenschaften > Optionen > Fenster**.

DisplayHeadings (Eigenschaft)

Datentyp: **Boolean**

Liest oder setzt die Einstellung, ob Zeilen- und Spaltenköpfe angezeigt werden. Entspricht dem Befehl **Ansicht > Zeilen- & Spaltenköpfe**.

Hinweise:

- Diese Eigenschaft ist nur aus Kompatibilitätsgründen mit Excel vorhanden. Vorzuziehen sind die Eigenschaften **DisplayRowHeadings** und **DisplayColumnHeadings** im **Sheet**-Objekt, da diese erstens die Ein-/Abschaltung der Zeilen- und der Spaltenköpfe getrennt erlauben und zweitens für jedes Arbeitsblatt einzeln eingesetzt werden können.
- Wenn Sie die Eigenschaft abfragen und sich die Einstellungen in den verschiedenen Arbeitsblättern des Dokuments unterscheiden, wird **smoUndefined** zurückgeliefert.

Zoom (Eigenschaft)

Datentyp: **Long**

Liest oder setzt die Vergrößerungsstufe, in der das Dokumentfenster dargestellt wird. Erlaubt sind Werte von 50 bis 400 (Prozent).

Alternativ können Sie den Wert -1 (oder die Konstante `pmZoomFitToSelection`) übergeben, woraufhin die Vergrößerungsstufe an die Selektion angepasst wird.

Beispiel:

```
' Vergrößerung im aktuellen Fenster auf 120% setzen
pm.ActiveWindow.Zoom = 120
```

Hinweis: Diese Einstellung wirkt sich nur auf das derzeit aktive Arbeitsblatt aus. Wenn Sie die Vergrößerungsstufe anderer Arbeitsblätter ändern möchten, müssen Sie diese zuerst aktivieren.

DisplayGridlines (Eigenschaft)

Datentyp: **Boolean**

Liest oder setzt die Einstellung, ob im Dokumentfenster Gitternetzlinien zwischen den Zellen angezeigt werden. Entspricht der Einstellung "Gitternetzlinien" im Dialogfenster des Befehls **Tabelle > Eigenschaften** – mit dem Unterschied, dass hier die Gitternetzlinien *aller* Arbeitsblätter im Dokument betroffen sind.

Hinweise:

- Diese Eigenschaft ist nur aus Kompatibilitätsgründen mit Excel vorhanden. Vorzuziehen ist die gleichnamige Eigenschaft im **Sheet**-Objekt, da diese das Abfragen und Setzen der Eigenschaft für jedes Arbeitsblatt separat erlaubt.
- Wenn Sie die Eigenschaft abfragen und sich die Einstellungen in den verschiedenen Arbeitsblättern des Dokuments unterscheiden, wird **smoUndefined** zurückgeliefert.

GridlineColor (Eigenschaft)

Datentyp: **Long** (SmoColor)

Liest oder setzt die Farbe der Gitternetzlinien als "BGR"-Wert (Blau-Grün-Rot-Triplet). Sie können entweder einen beliebigen Wert angeben oder eine der vordefinierten BGR-Farbkonstanten verwenden.

Hinweise:

- Diese Eigenschaft ist nur aus Kompatibilitätsgründen mit Excel vorhanden. Vorzuziehen ist die gleichnamige Eigenschaft im **Sheet**-Objekt, da diese das Abfragen und Setzen der Eigenschaft für jedes Arbeitsblatt separat erlaubt.
- Wenn Sie die Eigenschaft abfragen und sich die Einstellungen in den verschiedenen Arbeitsblättern des Dokuments unterscheiden, wird **smoUndefined** zurückgeliefert.

GridlineColorIndex (Eigenschaft)

Datentyp: **Long** (SmoColorIndex)

Liest oder setzt die Farbe der Gitternetzlinien als Indexfarbe. "Indexfarben" sind die Standardfarben von PlanMaker, durchnummeriert von -1 für Automatisch bis 15 für Hellgrau. Sie dürfen ausschließlich einen der in der Tabelle der Indexfarben genannten Werte verwenden.

Hinweise:

- Diese Eigenschaft ist nur aus Kompatibilitätsgründen mit Excel vorhanden. Vorzuziehen ist die gleichnamige Eigenschaft im **Sheet**-Objekt, da diese das Abfragen und Setzen der Eigenschaft für jedes Arbeitsblatt separat erlaubt.
- Wenn Sie die Eigenschaft abfragen und sich die Einstellungen in den verschiedenen Arbeitsblättern des Dokuments unterscheiden, wird **smoUndefined** zurückgeliefert.

Workbook (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das diesem Dokumentfenster zugeordnete **Workbook**-Objekt. Mit diesem können Sie zahlreiche Einstellungen Ihres Dokuments lesen und setzen.

ActiveCell (Zeiger auf Objekt)

Datentyp: **Object**

Liefert ein **Range**-Objekt, das die in diesem Dokumentfenster aktive Zelle repräsentiert. Mit diesem Objekt können Sie die Formatierung und den Inhalt der Zelle lesen und bearbeiten.

Bitte beachten Sie, dass **ActiveCell** auch dann nur eine einzige Zelle liefert, wenn im Arbeitsblatt ein Bereich selektiert ist. Denn der Zellrahmen kann innerhalb der Selektion an jeder beliebigen Stelle stehen: Ziehen Sie probehalber mit der Maus einen Bereich auf und drücken Sie dann wiederholt die Wagenrücklauftaste – Sie werden sehen, dass sich der Zellrahmen innerhalb der Selektion verschiebt.

ActiveSheet (Zeiger auf Objekt)

Datentyp: **Object**

Liefert ein **Sheet**-Objekt, das das in diesem Dokumentfenster aktive Arbeitsblatt repräsentiert. Mit diesem Objekt können Sie die Einstellungen des Arbeitsblatts lesen und bearbeiten.

Application (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das **Application**-Objekt.

Parent (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das übergeordnete Objekt, also **Windows**.

Activate (Methode)

Bringt das Dokumentfenster in den Vordergrund (sofern **Visible** für das Dokument **True** ist) und setzt den Fokus auf das Dokumentfenster.

Syntax:

```
Activate
```

Parameter:

keine

Rückgabetyp:

keiner

Beispiel:

```
' Das erste Dokumentfenster aktivieren
pm.Windows(1).Activate
```

Close (Methode)

Schließt das Dokumentfenster.

Syntax:

```
Close [SaveChanges]
```

Parameter:

SaveChanges (optional; Typ: **Long** bzw. **SmoSaveOptions**) gibt an, ob das im Fenster geöffnete Dokument gespeichert werden soll, sofern es seit dem letzten Speichern verändert wurde. Lassen Sie den Parameter weg, wird stattdessen gegebenenfalls der Benutzer gefragt. Mögliche Werte für **SaveChanges**:

```
smoDoNotSaveChanges = 0       ' Nicht fragen, nicht speichern
smoPromptToSaveChanges = 1    ' Den Benutzer fragen
smoSaveChanges = 2            ' Ohne Rückfrage speichern
```

Rückgabetyp:

keiner

Beispiel:

```
' Das aktuelle Fenster schließen, ohne es zu speichern
pm.ActiveWindow.Close smoDoNotSaveChanges
```

RecentFiles (Sammlung)

Zugriffspfad: Application → **RecentFiles**

1 Beschreibung

RecentFiles ist eine Sammlung der im Menü **Datei** angezeigten zuletzt geöffneten Dateien. Die einzelnen Elemente dieser Sammlung sind vom Typ **RecentFile**.

2 Zugriff auf die Sammlung

Es existiert während der gesamten Laufzeit von PlanMaker genau eine Instanz der **RecentFiles**-Sammlung. Diese wird über **Application.RecentFiles** angesprochen:

```
' Zeige den Namen der ersten Datei im Dateimenü an
MsgBox pm.Application.RecentFiles.Item(1).Name

' Öffne die erste Datei im Dateimenü
pm.Application.RecentFiles.Item(1).Open
```

3 Eigenschaften, Objekte, Sammlungen und Methoden

Eigenschaften:
- **Count** R/O
- **Maximum**

Objekte:
- ***Item*** → **RecentFile** (Defaultobjekt)
- **Application** → **Application**
- **Parent** → **Application**

Methoden:
- **Add**

Count (Eigenschaft, R/O)

Datentyp: **Long**

Liefert die Anzahl der **RecentFile**-Objekte in PlanMaker – in anderen Worten: die Anzahl der im Dateimenü verzeichneten zuletzt geöffneten Dateien.

Maximum (Eigenschaft, R/O)

Datentyp: **Long**

Liest oder setzt die Einstellung "Einträge im Datei-Menü" – in anderen Worten: wie viele zuletzt geöffnete Dateien das Dateimenü maximal anzeigen kann.

Der Wert darf zwischen 0 und 9 liegen.

Item (Zeiger auf Objekt)

Datentyp: **Object**

Liefert ein einzelnes **RecentFile**-Objekt, also einen einzelnen Dateieintrag im Dateimenü.

Welches RecentFile-Objekt Sie erhalten, hängt von dem Zahlenwert ab, den Sie an **Item** übergeben: 1 für die erste der zuletzt geöffneten Dateien, 2 für die zweite etc.

Application (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das **Application**-Objekt.

Parent (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das übergeordnete Objekt, also **Application**.

Add (Methode)

Fügt der Liste der zuletzt geöffneten Dateien ein Dokument hinzu.

Syntax:

```
Add Document, [FileFormat]
```

Parameter:

Document ist eine Zeichenkette mit dem Pfad und Dateinamen des hinzuzufügenden Dokuments.

FileFormat (optional; Typ: **Long** bzw. **PmSaveFormat**) gibt das Dateiformat des hinzuzufügenden Dokuments an. Mögliche Werte:

```
pmFormatDocument          =  0 ' Dokument, ist Default
pmFormatTemplate          =  1 ' Dokumentvorlage
pmFormatExcel97           =  2 ' Excel 97/2000/XP
pmFormatExcel5            =  3 ' Excel 5.0/7.0
pmFormatExcelTemplate     =  4 ' Excel-Vorlage
pmFormatSYLK              =  6 ' Sylk
pmFormatRTF               =  7 ' Rich Text Format
pmFormatTextMaker         =  7 ' TextMaker (= RTF)
pmFormatHTML              =  8 ' HTML
pmFormatdBaseDOS          =  9 ' dBASE-Datenbank im DOS-Zeichensatz
pmFormatdBaseAnsi         = 10 ' dBASE-Datenbank im Windows-Zeichensatz
pmFormatDIF               = 11 ' Textdatei mit Windows-Zeichensatz
pmFormatPlainTextAnsi     = 12 ' Textdatei mit Windows-Zeichensatz
pmFormatPlainTextDOS      = 13 ' Textdatei mit DOS-Zeichensatz
pmFormatPlainTextUnix     = 14 ' Textdatei mit ANSI-Zeichensatz für UNIX, Linux und
FreeBSD
pmFormatPlainTextUnicode  = 15 ' Textdatei mit Unicode-Zeichensatz
pmFormatdBaseUnicode      = 18 ' dBASE-Datenbank mit Unicode-Zeichensatz
pmFormatPlainTextUTF8     = 20 ' Textdatei mit UTF8-Zeichensatz
```

Wenn Sie diesen Parameter weglassen, wird **pmFormatDocument** angenommen.

Unabhängig vom übergebenen Parameter **FileFormat** versucht PlanMaker stets, das Dateiformat selbst zu erkennen, und ignoriert offensichtlich falsche Angaben.

Rückgabetyp:

Object (ein **RecentFile**-Objekt, das das hinzugefügte Dokument repräsentiert)

Beispiel:

```
' Die Datei Test.pmd dem Dateimenü hinzufügen
pm.Application.RecentFiles.Add "Test.pmd"

' Dito, aber mit Auswertung des Rückgabewerts (Klammern beachten!)
Dim fileObj as Object
Set fileObj = pm.Application.RecentFiles.Add("Test.pmd")
MsgBox fileObj.Name
```

RecentFile (Objekt)

Zugriffspfad: Application → RecentFiles → **Item**

1 Beschreibung

Ein **RecentFile**-Objekt repräsentiert eine einzelne der zuletzt geöffneten Dateien und lässt Sie deren Eigenschaften abfragen und sie erneut öffnen

Für jede Datei im Dateimenü existiert ein eigenes **RecentFile**-Objekt. Durch das Öffnen und Schließen von Dokumenten ändert sich die Liste der Dateien im Dateimenü; die zugehörigen **RecentFile**-Objekte werden von PlanMaker dynamisch angelegt bzw. entfernt.

2 Zugriff auf das Objekt

Die einzelnen **RecentFile**-Objekte können ausschließlich durch Aufzählung der Elemente der **RecentFiles**-Sammlung angesprochen werden. Diese erreichen Sie über **Application.RecentFiles**:

```
' Den Namen der ersten Datei im Dateimenü anzeigen
MsgBox pm.Application.RecentFiles.Item(1).Name
```

3 Eigenschaften, Objekte, Sammlungen und Methoden

Eigenschaften:
- **FullName** R/O
- ***Name*** (Defaulteigenschaft) R/O
- **Path** R/O

Objekte:
- **Application → Application**
- **Parent → RecentFiles**

Methoden:
- **Open**

FullName (Eigenschaft, R/O)

Datentyp: **String**

Liefert den Pfad und Namen des Dokuments im Dateimenü (z.B. c:\Kalkulation\Müller.pmd).

Name (Eigenschaft, R/O)

Datentyp: **String**

Liefert den Namen des Dokuments (z.B. Müller.pmd).

Path (Eigenschaft, R/O)

Datentyp: **String**

Liefert den Pfad des Dokuments (z.B. c:\Kalkulation).

Application (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das **Application**-Objekt.

Parent (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das übergeordnete Objekt, also **RecentFiles**.

Open (Methode)

Öffnet das betreffende Dokument und liefert es als **Workbook**-Objekt zurück.

Syntax:

Open

Parameter:

keine

Rückgabetyp:

Workbook

Beispiel:

```
' Das erste Dokument aus dem Dateimenü öffnen
pm.Application.RecentFiles(1).Open
```

FontNames (Sammlung)

Zugriffspfad: Application → **FontNames**

1 Beschreibung

FontNames ist eine Sammlung aller in Windows installierten Schriftarten. Die einzelnen Elemente dieser Sammlung sind vom Typ **FontName**.

2 Zugriff auf die Sammlung

Es existiert während der gesamten Laufzeit von PlanMaker genau eine Instanz der **FontNames**-Sammlung. Diese wird über **Application.FontNames** angesprochen:

```
' Den Namen der ersten installierten Schrift anzeigen
MsgBox pm.Application.FontNames.Item(1).Name

' Dasselbe, nur kürzer durch die Nutzung von Defaulteigenschaften:
MsgBox pm.FontNames(1)
```

3 Eigenschaften, Objekte, Sammlungen und Methoden

Eigenschaften:

- **Count** R/O

Objekte:
- ***Item*** → **FontName** (Defaultobjekt)
- **Application** → **Application**
- **Parent** → **Application**

Count (Eigenschaft, R/O)

Datentyp: **Long**

Liefert die Anzahl der **FontName**-Objekte in PlanMaker – in anderen Worten: die Anzahl der im System installierten Schriften.

Item (Zeiger auf Objekt)

Datentyp: **Object**

Liefert ein einzelnes **FontName**-Objekt, also eine einzelne installierte Schriftart.

Welches FontName-Objekt Sie erhalten, hängt von dem Zahlenwert ab, den Sie an **Item** übergeben: 1 für die erste installierte Schrift, 2 für die zweite etc.

Application (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das **Application**-Objekt.

Parent (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das übergeordnete Objekt, also **Application**.

FontName (Objekt)

Zugriffspfad: Application → FontNames → **Item**

1 Beschreibung

Ein **FontName**-Objekt repräsentiert eine einzelne der in Windows installierten Schriftarten. Für jede installierte Schriftart existiert ein eigenes **FontName**-Objekt.

2 Zugriff auf das Objekt

Die einzelnen **FontName**-Objekte können ausschließlich durch Aufzählung der Elemente der **FontNames**-Sammlung angesprochen werden. Diese erreichen Sie über **Application.FontNames**:

```
' Den Namen der ersten installierten Schrift anzeigen
MsgBox pm.Application.FontNames.Item(1).Name

' Dasselbe, nur kürzer durch die Nutzung von Defaulteigenschaften:
MsgBox pm.FontNames(1)
```

3 Eigenschaften, Objekte, Sammlungen und Methoden

Eigenschaften:
- ***Name*** (Defaulteigenschaft) R/O
- **Charset** R/O

Objekte:
- **Application → Application**
- **Parent → FontNames**

Name (Eigenschaft, R/O)

Datentyp: **String**

Liefert den Namen der betreffenden Schrift.

Charset (Eigenschaft, R/O)

Datentyp: **Long** (SmoCharset)

Liefert den Zeichensatz der betreffenden Schrift. Mögliche Werte:

```
smoAnsiCharset   = 0 ' normaler Zeichensatz
smoSymbolCharset = 2 ' Symbolschrift
```

Application (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das **Application**-Objekt.

Parent (Zeiger auf Objekt)

Datentyp: **Object**

Liefert das übergeordnete Objekt, also **FontNames**.

Anweisungen und Funktionen von A-Z

In diesem Kapitel finden Sie eine Beschreibung aller in SoftMaker Basic verfügbaren Anweisungen und Funktionen.

Es gibt folgende Funktionen:

- **Kontrollstrukturen**

  Do ... Loop, End, Exit For, Exit Loop, For ... Next, Gosub, Goto, If ... Then ... Else, Return, Select Case, Stop, While ... Wend

- **Konvertierung**

  Asc, CDbl, Chr, CInt, CLng, CSng, CStr, Fix, Format, Hex, Int, Oct, Str, Val

- **Datum** und **Uhrzeit**

  Date, DateSerial, DateValue, Day, Hour, Minute, Month, Now, Second, Time, TimeSerial, TimeValue, Weekday, Year

- **Dialoge**

  Dialog, Dialogfunktion, DlgEnable, DlgText, DlgVisible

- **Dateioperationen**

  ChDir, ChDrive, Close, CurDir, EOF, FileCopy, FileLen, FreeFile, Input, Kill, Line Input #, MkDir, Open, Print #, RmDir, Seek, Write #

- **Arithmetik**

  Abs, Atn, Cos, Exp, Log, Rnd, Sgn, Sin, Sqr, Tan

- **Prozeduren**

  Call, Declare, Exit, Function ... End Function, Sub ... End Sub

- **Zeichenketten**

  Asc, Chr, InStr, LCase, Left, Len, LTrim, Mid, Right, RTrim, Space, Str, StrComp, String, Trim

- **Variablen** und **Konstanten**

  Const, Dim, IsDate, IsEmpty, IsNull, IsNumeric, Option Explicit, VarType

- **Arrays**

  Dim, Erase, LBound, Option Base, Option Explicit, Static, UBound

- **Applikationen** und **OLE**

  AppActivate, AppDataMaker, AppPlanMaker, AppSoftMakerPresentations, AppTextMaker, CreateObject, GetObject, SendKeys, Shell

- **Verschiedenes**

  Beep, Rem

Abs (Funktion)

Abs (*Num*)

Ermittelt den Absolutbetrag des numerischen Wertes *Num*, entfernt also das Vorzeichen. Wenn *Num* Null ist, liefert **Abs** den Wert Null.

Der Datentyp des Rückgabewerts entspricht dem des übergebenen Parameters *Num*. Ausnahme: wenn *Num* vom Typ Variant mit VarType 8 (String) ist und in eine Zahl konvertiert werden kann, ist das Ergebnis vom Typ Variant mit VarType 5 (Double).

Siehe auch: Sgn

Beispiel:

```
Sub Main

    Dim Msg, x, y

    x = InputBox("Geben Sie eine Zahl ein:")
    y = Abs(x)
    Msg = "Der Absolutwert von " & x & " beträgt: " & y
    MsgBox Msg

End Sub
```

AppActivate (Anweisung)

AppActivate *"Titel"*

Aktiviert eine bereits gestartete Applikation, bringt also das Applikationsfenster in den Vordergrund und gibt der Applikation den Fokus.

Die Zeichenkette *Titel* ist der Name der Applikation, wie er in der Titelleiste erscheint.

Siehe auch: SendKeys, **Shell**

Beispiel:

```
Sub Main
    X = Shell("Calc.exe", 1)        ' Rechner aufrufen
    For i = 1 To 5
        SendKeys i & "{+}", True    ' Tastendrücke senden
    Next i
    Msg = "Der Rechner wird jetzt wieder beendet."
    MsgBox Msg
    AppActivate "Rechner"           ' Fokus auf Rechner setzen
    SendKeys "%{F4}", True          ' Alt+F4 zum Beenden senden
End Sub
```

AppDataMaker (Funktion)

AppDataMaker ["*Kommandozeilenparameter*"]

Startet die Datenbank DataMaker.

Der Rückgabewert ist eine Task-ID, die das Programm identifiziert. Werte kleiner als 32 zeigen an, dass der Programmstart fehlgeschlagen ist.

Als Kommandozeilenparameter kann beispielsweise der Name der zu öffnenden Datei übergeben werden – zum Beispiel:

```
AppDataMaker "c:\Daten\Kunden.dbf"
```

Damit dieser Befehl nicht fehlschlägt, muss DataMaker in die Windows-Registry eingetragen sein. Ist dies nicht der Fall, genügt es, DataMaker einmal konventionell zu starten, wobei sich das Programm automatisch in die Registry einträgt.

Siehe auch: AppPlanMaker, **AppSoftMakerPresentations**, **AppTextMaker**, **CreateObject**, **GetObject**, **Shell**

AppPlanMaker (Funktion)

AppPlanMaker ["*Kommandozeilenparameter*"]

Startet die Tabellenkalkulation PlanMaker.

Der Rückgabewert ist eine Task-ID, die das Programm identifiziert. Werte kleiner als 32 zeigen an, dass der Programmstart fehlgeschlagen ist.

Als Kommandozeilenparameter kann beispielsweise der Name der zu öffnenden Datei übergeben werden – zum Beispiel:

```
AppPlanMaker "c:\Daten\Tabelle1.pmd"
```

Damit dieser Befehl nicht fehlschlägt, muss PlanMaker in die Windows-Registry eingetragen sein. Ist dies nicht der Fall, genügt es, PlanMaker einmal konventionell zu starten, wobei sich das Programm automatisch in die Registry einträgt.

Hinweis: Dieser Befehl startet lediglich die Applikation PlanMaker, baut dabei aber keine OLE Automation-Verbindung auf. Möchten Sie eine OLE Automation-Verbindung mit PlanMaker herstellen, können Sie nach dem **AppPlanMaker**-Aufruf die **GetObject**-Funktion verwenden. Alternativ können Sie von vornherein statt der **AppPlanMaker**-Funktion die **CreateObject**-Funktion verwenden. Hierbei wird ebenfalls PlanMaker gestartet, gleichzeitig aber auch eine OLE Automation-Verbindung hergestellt.

Siehe auch: AppDataMaker, **AppSoftMakerPresentations**, **AppTextMaker**, **CreateObject**, **GetObject**, **Shell**

AppSoftMakerPresentations (Funktion)

AppSoftMakerPresentations ["*Kommandozeilenparameter*"]

Startet das Präsentationsgrafikprogramm SoftMaker Presentations.

Der Rückgabewert ist eine Task-ID, die das Programm identifiziert. Werte kleiner als 32 zeigen an, dass der Programmstart fehlgeschlagen ist.

Als Kommandozeilenparameter kann beispielsweise der Name der zu öffnenden Datei übergeben werden – zum Beispiel:

```
AppSoftMakerPresentations "c:\daten\Präsentation1.prd"
```

Damit dieser Befehl nicht fehlschlägt, muss SoftMaker Presentations in die Windows-Registry eingetragen sein. Ist dies nicht der Fall, genügt es, SoftMaker Presentations einmal konventionell zu starten, wobei sich das Programm automatisch in die Registry einträgt.

Siehe auch: AppDataMaker, **AppTextMaker**, **CreateObject**, **GetObject**, **Shell**

AppTextMaker (Funktion)

AppTextMaker ["*Kommandozeilenparameter*"]

Startet die Textverarbeitung TextMaker.

Der Rückgabewert ist eine Task-ID, die das Programm identifiziert. Werte kleiner als 32 zeigen an, dass der Programmstart fehlgeschlagen ist.

Als Kommandozeilenparameter kann beispielsweise der Name der zu öffnenden Datei übergeben werden – zum Beispiel:

```
AppTextMaker "c:\Texte\Brief.tmd"
```

Damit dieser Befehl nicht fehlschlägt, muss TextMaker in die Windows-Registry eingetragen sein. Ist dies nicht der Fall, genügt es, TextMaker einmal konventionell zu starten, wobei sich das Programm automatisch in die Registry einträgt.

Hinweis: Dieser Befehl startet lediglich die Applikation TextMaker, baut dabei aber keine OLE Automation-Verbindung auf. Möchten Sie eine OLE Automation-Verbindung mit TextMaker herstellen, können Sie nach dem **AppTextMaker**-Aufruf die **GetObject**-Funktion verwenden. Alternativ können Sie von vornherein statt der **AppTextMaker**-Funktion die **CreateObject**-Funktion verwenden. Hierbei wird ebenfalls TextMaker gestartet, gleichzeitig aber auch eine OLE Automation-Verbindung hergestellt.

Siehe auch: AppDataMaker, **AppPlanMaker**, **AppSoftMakerPresentations**, **CreateObject**, **GetObject**, **Shell**

Asc (Funktion)

Asc(*Str*)

Ermittelt den Zeichencode (ANSI-Code) des ersten Zeichens einer Zeichenkette.

Das Ergebnis ist eine ganze Zahl zwischen 0 und 32767.

Siehe auch: Chr

Beispiel:

```
Sub Main
    Dim i, Msg
    For i = Asc("A") To Asc("Z")
        Msg = Msg & Chr(i)
    Next i
    MsgBox Msg
End Sub
```

Atn (Funktion)

Atn(*Num*)

Ermittelt den Arcustangens einer Zahl.

Das Ergebnis wird im Bogenmaß (Radiant) ermittelt.

Siehe auch: Cos, **Sin**, **Tan**

Beispiel:

```
Sub AtnExample
    Dim Msg, Pi          ' Variablen deklarieren
    Pi = 4 * Atn(1)      ' Pi berechnen
    Msg = "Pi = " & Str(Pi)
    MsgBox Msg           ' Ergebnis: "Pi = 3.1415..."
End Sub
```

Beep (Anweisung)

Beep

Gibt einen kurzen Ton aus.

Beispiel:

```
Sub Beep3x
```

```
    Dim i As Integer
    For i = 1 to 3
        Beep
    Next i
End Sub
```

Begin Dialog ... End Dialog (Anweisung)

Begin Dialog *DialogName* [*X, Y,*] *Breite, Höhe, Titel$* [*,.Dialogfunktion*]

    Dialogdefinition...

End Dialog

Dient zur Definition eines benutzerdefinierten Dialogfensters. Siehe Abschnitt "Dialogdefinition".

Allgemeine Informationen zum Erstellen benutzerdefinierter Dialogfenster finden Sie im Abschnitt "Dialogfenster".

Call (Anweisung)

Call *Name* [(*Parameter*)]

Oder:

Name [*Parameter*]

Ruft die **Sub**- oder **Function**-Prozedur oder DLL-Funktion *Name* auf.

Parameter ist eine mit Kommata getrennte Liste von Parametern, die an die Prozedur übergeben werden kann.

Das Schlüsselwort **Call** wird üblicherweise weggelassen. Wenn es benutzt wird, muss die Parameterliste in Klammern eingeschlossen werden, sonst dürfen *keine* Klammern verwendet werden.

Call *Name*(*Parameter1, Parameter2* ...) ist also gleichbedeutend mit *Name Parameter1, Parameter2* ...

Auch Funktionen können mit der **Call**-Anweisung aufgerufen wird; der Rückgabewert geht hierbei jedoch verloren.

Siehe auch: Declare, **Function**, **Sub**

Beispiel:

```
Sub Main
    Call Beep
End Sub
```

CDbl (Funktion)

CDbl(*Ausdruck*)

Wandelt einen Ausdruck in den Datentyp **Double** um. Der Parameter *Ausdruck* muss numerisch oder eine Zeichenkette sein.

Siehe auch: CInt, **CLng**, **CSng**, **CStr**

Beispiel:

```
Sub Main
    Dim y As Integer
    y = 25
    If VarType(y) = 2 Then
        Print y
        x = CDbl(y)
```

```
        Print x
    End If
End Sub
```

ChDir (Anweisung)

ChDir [*Laufwerk:*]*Verzeichnis*

Wechselt das aktuelle Laufwerk/Verzeichnis.

Laufwerk ist optional (Standardwert: aktuelles Laufwerk)

Verzeichnis ist der Name des Verzeichnisses auf dem angegebenen Laufwerk.

Der gesamte Pfadname darf maximal 255 Zeichenumfassen.

Siehe auch: CurDir, **ChDrive**, **MkDir**, **RmDir**

Beispiel:

```
Sub Main
    Dim Answer, Msg, NL
    NL = Chr(10)                    ' Chr(10)=Neue Zeile
    CurPath = CurDir()              ' Aktuellen Pfad ermitteln
    ChDir "\"
    Msg = "Es wurde in das Verzeichnis " & CurDir() & " gewechselt."
    Msg = Msg & NL & NL & "Klicken Sie auf OK, "
    Msg = Msg & "um in das vorherige Verzeichnis zurückzugelangen."
    Answer = MsgBox(Msg)
    ChDir CurPath                   ' Zurück ins alte Verzeichnis
    Msg = "Wir sind jetzt wieder im Verzeichnis " & CurPath & "."
    MsgBox Msg
End Sub
```

ChDrive (Anweisung)

ChDrive *Laufwerk*

Wechselt das aktuelle Laufwerk.

Laufwerk ist eine Zeichenkette mit dem Laufwerksbuchstaben.

Sollte *Laufwerk* mehr als einen Buchstaben enthalten, wird nur der erste Buchstabe verwendet.

Siehe auch: ChDir, **CurDir**, **MkDir**, **RmDir**

Beispiel:

```
Sub Main
    Dim Answer, Msg, NL
    NL = Chr(10)                    ' Chr(10)=Neue Zeile
    CurPath = CurDir()              ' Aktuellen Pfad ermitteln
    ChDrive "D"
    Msg = "Es wurde in das Verzeichnis " & CurDir() & " gewechselt. "
    Msg = Msg & NL & NL & "Klicken Sie auf OK, "
    Msg = Msg & "um in das vorherige Verzeichnis zurückzugelangen."
    Answer = MsgBox(Msg)
    ChDir CurPath                   ' Zurück ins urspr. Verzeichnis
    Msg = "Wir sind jetzt wieder im Verzeichnis " & CurPath & "."
    MsgBox Msg
End Sub
```

Chr (Funktion)

Chr(Num)

Liefert das dem angegebenen Zeichencode (ANSI-Code) entsprechende Zeichen.

Für *Num* wird eine Integer-Zahl zwischen 0 und 32767 erwartet.

Siehe auch: Asc

Beispiel:

```
Sub Main
    Dim i, Msg
    For i = Asc("A") To Asc("Z")
        Msg = Msg & Chr(i)
    Next i
    MsgBox Msg
End Sub
```

CInt (Funktion)

CInt(*Ausdruck*)

Wandelt einen Ausdruck in den Datentyp **Integer** um.

Der Parameter *Ausdruck* muss numerisch oder eine Zeichenkette, die eine Zahl enthält, sein.

Gültiger Wertebereich:

-32768 <= *Ausdruck* <= 32768

Siehe auch: CDbl, **CLng**, **CSng**, **CStr**

Beispiel:

```
Sub Main
    Dim y As Long
    y = 25
    x = CInt(y)
    Print x
End Sub
```

CLng (Funktion)

CLng(*Ausdruck*)

Wandelt einen Ausdruck in den Datentyp **Long** um.

Der Parameter *Ausdruck* muss numerisch oder eine Zeichenkette, die eine Zahl enthält, sein.

Gültiger Wertebereich:

-2147483648 <= *Ausdruck* <= 2147483648

Siehe auch: CDbl, **CInt**, **CSng**, **CStr**

Beispiel:

```
Sub Main
    Dim y As Integer
    y = 25
    If VarType(y) = 2 Then
        Print y
```

```
        x = CLng(y)
        Print x
    End If
End Sub
```

Close (Anweisung)

Close [[#]*Dateinummer*]

Schließt eine beziehungsweise alle offenen Dateien.

Dateinummer ist die in der **Open**-Anweisung vergebene Nummer. Wird keine Dateinummer angegeben, werden alle derzeit geöffneten Dateien geschlossen.

Siehe auch: Open

Beispiel:

```
Sub Make3Files
    Dim i, FNum, Fname
    For i = 1 To 3
        FNum = FreeFile                    ' Freien Dateizeiger holen
        FName = "TEST" & FNum
        Open FName For Output As Fnum      ' Datei öffnen
        Print #I, "Dies ist Test #" & i    ' In Datei schreiben
        Print #I, "Eine weitere Zeile"
    Next i
    Close                                  ' Alle Dateien schließen
End Sub
```

Const (Anweisung)

Const *Name = Ausdruck*

Definiert einen symbolischen Namen für eine Konstante.

Außerhalb von Prozeduren definierte Konstanten sind stets global.

Ein Typsuffix (z.B. % für **Integer**, siehe Abschnitt "Datentypen") kann an den Namen angehängt werden, um den Datentyp der Konstante festzulegen. Ansonsten ist der Typ je nach Wert **Long**, **Double** oder **String**.

Siehe auch: Abschnitt "Datentypen"

Beispiel:

```
Global Const GlobalConst = 142
Const MyConst = 122

Sub Main
    Dim Answer, Msg
    Const PI = 3.14159
    .
    .
    .
```

Cos (Funktion)

Cos(*Num*)

Ermittelt den Cosinus eines Winkels.

Der Winkel muss im Bogenmaß (Radiant) angegeben werden.

Siehe auch: Atn, **Sin**, **Tan**

Beispiel:

```
Sub Main
    pi = 4 * Atn(1)
    rad = 180 * (pi/180)
    x = Cos(rad)
    Print x
End Sub
```

CreateObject (Funktion)

CreateObject(*Klasse*)

Erzeugt ein OLE Automation-Objekt und liefert eine Referenz auf dieses Objekt zurück.

Für den Parameter *Klasse* wird folgende Syntax erwartet:

Applikation.Klasse

Applikation ist der Name der Applikation und *Klasse* der Typ des Objekts. *Klasse* ist dabei der Name, unter dem das Objekt in der Windows-Registry bekannt ist.

Ein Beispiel:

```
Set tm = CreateObject("TextMaker.Application")
```

Beim Aufruf wird die betreffende Applikation automatisch gestartet, sofern sie nicht bereits läuft.

Sobald das Objekt kreiert ist, kann auf die bereitgestellten Methoden und Eigenschaften mit der Punktnotation zugegriffen werden – zum Beispiel:

```
tm.Visible = True
```

Siehe auch: GetObject, **Set**, Abschnitt "OLE Automation"

CSng (Funktion)

CSng(*Ausdruck*)

Wandelt einen Ausdruck in den Datentyp **Single** um.

Siehe auch: CDbl, **CInt**, **CLng**, **CStr**

Beispiel:

```
Sub Main
    Dim y As Integer
    y = 25
    If  VarType(y) = 2 Then
        Print y
        x = CSng(y)
        Print x
    End If
End Sub
```

CStr (Funktion)

CStr(*Ausdruck*)

Wandelt einen Ausdruck in den Datentyp **String** um.

Im Gegensatz zur **Str**-Funktion wird der resultierenden Zeichenkette bei **CStr** kein Leerzeichen vorangestellt, wenn diese eine positive Zahl enthält.

Siehe auch: CDbl, **CInt**, **CLng**, **CSng**, **Str**

CurDir (Funktion)

CurDir(*Laufwerk*)

Ermittelt das aktuelle Verzeichnis auf dem angegebenen Laufwerk.

Laufwerk ist eine Zeichenkette mit dem Laufwerksbuchstaben.

Wird *Laufwerk* nicht angegeben, wird das aktuelle Laufwerk verwendet.

Siehe auch: ChDir, **ChDrive**, **MkDir**, **RmDir**

Beispiel:

```
Sub Main
    MsgBox "Das aktuelle Verzeichnis ist: " & CurDir()
End Sub
```

Date (Funktion)

Date [()]

Gibt das aktuelle Datum im kurzen Datumsformat zurück.

Das kurze Datumsformat (normalerweise TT.MM.JJ) kann über das Symbol zum Ändern der Regions- und Sprachoptionen in der Systemsteuerung verändert werden.

Das Ergebnis ist eine Variant-Variable vom VarType 8 (String).

Siehe auch: DateSerial, **DateValue**, **Day**, **Month**, **Now**, **Time**, **TimeSerial**, **TimeValue**, **Weekday**, **Year**

Beispiel:

```
Sub Main
   MsgBox "Heute ist der " & Date & "."
End Sub
```

DateSerial (Funktion)

DateSerial(*Jahr*, *Monat*, *Tag*)

Liefert eine Variant-Variable (Typ: Datum) mit dem Datum, das durch die Parameter *Jahr*, *Monat* und *Tag* festgelegt wird.

Siehe auch: DateValue, **Day**, **Month**, **Now**, **Time**, **TimeSerial**, **TimeValue**, **Weekday**, **Year**

Beispiel:

```
Sub Main
```

```
   Print DateSerial(2010,09,25)   ' liefert 25.09.2010
End Sub
```

DateValue (Funktion)

DateValue(*DatumsAusdruck*)

Liefert eine Variant-Variable (Typ: Datum) mit dem Datum, das durch den übergebenen Parameter *DatumsAusdruck* festgelegt wird. Dieser Parameter kann eine Zeichenkette oder ein beliebiger anderer Ausdruck sein, der ein Datum, eine Uhrzeit oder sowohl ein Datum als auch eine Uhrzeit repräsentiert.

Siehe auch: DateSerial, **Day**, **Month**, **Now**, **Time**, **TimeSerial**, **TimeValue**, **Weekday**, **Year**

Beispiel:

```
Sub Main
   Print DateValue("25. September 2010")   ' liefert 25.09.2010
End Sub
```

Day (Funktion)

Day(*Ausdruck*)

Liefert den Tag des angegebenen Datums als ganze Zahl.

Ausdruck ist ein numerischer oder Zeichenkettenausdruck, der ein Datum repräsentiert.

Siehe auch: Date, **Hour**, **Minute**, **Month**, **Now**, **Second**, **Time**, **Weekday**, **Year**

Beispiel:

```
Sub Main

    T1 = Now     ' Now = aktuelles Datum + Uhrzeit

    MsgBox T1

    MsgBox "Tag: " & Day(T1)
    MsgBox "Monat: " & Month(T1)
    MsgBox "Jahr: " & Year(T1)

    MsgBox "Stunde: " & Hour(T1)
    MsgBox "Minute: " & Minute(T1)
    MsgBox "Sekunde: " & Second(T1)

End Sub
```

Declare (Anweisung)

Declare Sub *Name* **Lib** *Libname$* [**Alias** *Aliasname$*] [(*Parameter*)]

Oder:

Declare Function *Name* **Lib** *Libname$* [**Alias** *Aliasname$*] [(*Parameter*)] [**As** *Typ*]

Deklariert eine Prozedur oder Funktion, die in einer Dynamic Link Library (DLL) enthalten ist.

Name ist der Name der Prozedur oder Funktion.

Libname ist der Name der DLL, in der sich die Prozedur oder Funktion befindet.

Aliasname ist der Name, unter dem die Prozedur oder Funktion von der DLL exportiert wird. Wird *Aliasname* nicht angegeben, ist er gleich *Name*. Ein Alias wird beispielsweise benötigt, wenn der Exportname ein reservierter Name in SoftMaker Basic ist oder Zeichen enthält, die in Namen nicht zulässig sind.

Parameter ist eine durch Kommata getrennte Liste von Parameterdeklarationen (siehe unten).

Typ spezifiziert den Datentyp (**String**, **Integer**, **Double**, **Long**, **Variant**). Alternativ kann der Typ auch durch einen Typsuffix (z.B. % für **Integer**) am Funktionsnamen angegeben werden (siehe Abschnitt "Datentypen").

Declare-Anweisungen dürfen nur außerhalb jeglicher Sub- oder Function-Deklarationen verwendet werden.

Parameterdeklaration

[**ByVal** | **ByRef**] *Variable* [**As** *Typ*]

Mit **ByVal** beziehungsweise **ByRef** (Standardwert) wird bestimmt, ob der Parameter als Wert oder als Referenz übergeben wird (siehe Abschnitt "Parameterübergabe ByRef oder ByVal").

Typ spezifiziert den Datentyp (**String**, **Integer**, **Double**, **Long**, **Variant**). Alternativ kann der Typ auch durch einen Typsuffix (z.B. % für **Integer**) angegeben werden (siehe Abschnitt "Datentypen").

Siehe auch: Call, Abschnitt "Aufruf von Funktionen in DLLs"

Dialog (Funktion)

Dialog(*Dlg*)

Zeigt ein benutzerdefiniertes Dialogfenster an.

Dlg ist der Name einer Dialogvariablen, die zuvor mit der **Dim**-Anweisung angelegt werden muss.

Der Rückgabewert ist der Index der Schaltfläche, die der Benutzer betätigt hat:

-1 **OK**

0 **Abbrechen**

> 0 Benutzerdefinierte Befehlsschaltfläche (1 für die erste, 2 für die zweite usw.)

Siehe auch: DlgEnable, **DlgText**, **DlgVisible**, Abschnitt "Dialogfenster"

Beispiel:

```
' Zeigt abhängig davon, welche Schaltfläche angeklickt wurde,
' unterschiedliche Informationen an.

Sub Main
    Dim MyList$(2)
    MyList(0) = "Banane"
    MyList(1) = "Orange"
    MyList(2) = "Apfel"

    Begin Dialog DialogName1 60, 60, 240, 184, "Test-Dialog"
        Text 10, 10, 28, 12, "Name:"
        TextBox 40, 10,50, 12, .joe
        ListBox 102, 10, 108, 16, MyList$(), .MyList1
        ComboBox 42, 30, 108, 42, MyList$(), .Combo1
        DropListBox 42, 76, 108, 36, MyList$(), .DropList1$
        OptionGroup .grp1
            OptionButton 42, 100, 48, 12, "Option&1"
            OptionButton 42, 110, 48, 12, "Option&2"
        OptionGroup .grp2
            OptionButton 42, 136, 48, 12, "Option&3"
            OptionButton 42, 146, 48, 12, "Option&4"
        GroupBox 132, 125, 70, 36, "Group"
        CheckBox 142, 100, 48, 12, "Check&A", .Check1
```

```
        CheckBox 142, 110, 48, 12, "Check&B", .Check2
        CheckBox 142, 136, 48, 12, "Check&C", .Check3
        CheckBox 142, 146, 48, 12, "Check&D", .Check4
        CancelButton 42, 168, 40, 12
        OKButton 90, 168, 40, 12
        PushButton 140, 168, 40, 12, "Schaltfläche1"
        PushButton 190, 168, 40, 12, "Schaltfläche2"
    End Dialog

    Dim Dlg1 As DialogName1
    Dlg1.joe  = "Hase"
    Dlg1.MyList1 = 1
    Dlg1.Combo1 = "Kiwi"
    Dlg1.DropList1 = 2
    Dlg1.grp2 = 1

    ' Dialog liefert -1 bei OK, 0 bei Abbrechen, # bei Schaltfläche1/2
    button = Dialog(Dlg1)
    If button = 0 Then Return
    MsgBox "Eingabefeld: "& Dlg1.joe
    MsgBox "Listenfeld: " & Dlg1.MyList1
    MsgBox Dlg1.Combo1
    MsgBox Dlg1.DropList1
    MsgBox "Gruppe1: " & Dlg1.grp1
    MsgBox "Gruppe2: " & Dlg1.grp2

    Begin Dialog DialogName2 60, 60, 160, 60, "Test-Dialog 2"
        Text 10, 10, 28, 12, "Name:"
        TextBox 42, 10, 108, 12, .fred
        OkButton 42, 44, 40, 12
    End Dialog

    If button = 2 Then
        Dim Dlg2 As DialogName2
        Dialog Dlg2
        MsgBox Dlg2.fred
    ElseIf button = 1 Then
        Dialog Dlg1
        MsgBox Dlg1.Combo1
    End If

End Sub
```

Dim (Anweisung)

Dim *Name* [(*Subscripts*)][**As** *Typ*] [, ...]

Legt Speicherplatz für eine Variable an und bestimmt deren Datentyp.

Name ist der Name der Variable.

Subscripts gibt Zahl und Größe der Dimensionen an, falls ein Array angelegt werden soll (siehe Abschnitt "Arrays"). Es wird dabei folgende Syntax erwartet:

> [Untergrenze **To**] Obergrenze [, [Untergrenze **To**] Obergrenze] ...
>
> Für Untergrenze und Obergrenze sind ganze Zahlen anzugeben, die den kleinsten und den größten zulässigen Wert für den Index des Arrays angeben und somit seine Größe bestimmen. Ist Untergrenze nicht angegeben, wird der mittels **Option Base** festgelegte Wert (0 oder 1) dafür genommen.
>
> Bei dynamischen Arrays (siehe **ReDim**-Anweisung) werden keine Grenzen angegeben:
>
> **Dim** a()

Typ spezifiziert den Datentyp (**Integer**, **Long**, **Single**, **Double**, **String**, **String****n*, **Variant, Object** oder ein benutzerdefinierter Typ). Alternativ kann der Typ auch durch einen Typsuffix (z.B. % für **Integer**) am Namen angegeben werden (siehe Abschnitt "Datentypen").

```
Dim Value As Integer
```

entspricht beispielsweise:

```
Dim Value%
```

Wird weder der Datentyp noch ein Typsuffix angegeben, wird eine Variant-Variable erzeugt.

Siehe auch: Option Base, **ReDim**, Abschnitt "Variablen"

Beispiel:

```
Sub Main
    Dim a As Integer   ' (alternativ: Dim a%)
    Dim b As Long
    Dim c As Single
    Dim d As Double
    Dim e As String
    Dim f As Variant   ' (alternativ: Dim f oder Dim f as Any)
    Dim g(10,10) As Integer   ' Array von Variablen
    .
    .
    .
```

DlgEnable (Anweisung)

DlgEnable "*Name*" [, *Zustand*]

Aktiviert oder deaktiviert ein Kontrollelement in einem benutzerdefinierten Dialog. Ein deaktiviertes Kontrollelement wird "ausgegraut" dargestellt. Es kann vom Anwender nicht verändert werden.

Diese Anweisung kann von einer Dialogfunktion aus aufgerufen werden.

Die Zeichenkette *Name* ist der Name des Kontrollelements im Dialogfeld.

Wenn *Zustand* = 0 ist, wird das Kontrollelement deaktiviert, bei allen anderen Werten aktiviert. Wird für *Zustand* nichts angegeben, wird der Zustand des Kontrollelements umgeschaltet.

Siehe auch: DlgText, **DlgVisible**, Abschnitt "Dialogfenster"

Beispiel:

```
If ControlID$ = "Chk1" Then
          DlgEnable "Group", 1
          DlgVisible "Chk2"
          DlgVisible "History"
End If
```

DlgText (Anweisung)

DlgText "*Name*", *Text*

Setzt den Text eines Kontrollelements in einem benutzerdefinierten Dialog.

Diese Anweisung kann von einer Dialogfunktion aus aufgerufen werden.

Die Zeichenkette *Name* ist der Name des Kontrollelements im Dialogfeld.

Die Zeichenkette *Text* ist der zu setzende Text.

Siehe auch: DlgEnable, **DlgVisible**, Abschnitt "Dialogfenster"

Beispiel:

```
If ControlID$ = "Chk2" Then
    DlgText "t1", "Öffnen"
End If
```

DlgVisible (Anweisung)

DlgVisible "*Name*", [*Wert*]

Versteckt ein Kontrollelement in einem benutzerdefinierten Dialog beziehungsweise macht er wieder sichtbar.

Diese Anweisung kann von einer Dialogfunktion aus aufgerufen werden.

Die Zeichenkette *Name* ist der Name des Kontrollelements im Dialogfeld.

Wenn *Wert* = 0 ist, wird das Kontrollelement versteckt, bei allen anderen Werten angezeigt. Wird für *Wert* nichts angegeben, wird das Kontrollelement versteckt, falls es momentan sichtbar ist, und umgekehrt.

Siehe auch: DlgEnable, **DlgText**, Abschnitt "Dialogfenster"

Beispiel:

```
If ControlID$ = "Chk1" Then
          DlgEnable "Group", 1
          DlgVisible "Chk2"
          DlgVisible "Öffnen"
End If
```

Do ... Loop (Anweisung)

Do [{**While**|**Until**} *Bedingung*]
   [*Anweisungen*]
   [**Exit Do**]
   [*Anweisungen*]
Loop

Oder:

Do
   [*Anweisungen*]
   [**Exit Do**]
   [*Anweisungen*]
Loop [{**While**|**Until**} *Bedingung*]

Führt eine Gruppe von Anweisungen wiederholt aus, solange eine Bedingung wahr ist (Do ... While) beziehungsweise bis eine Bedingung wahr wird (Do ... Until). Siehe auch Abschnitt "Kontrollstrukturen".

Siehe auch: While ... **Wend**, Abschnitt "Kontrollstrukturen"

Beispiel:

```
Sub Main
    Dim Value, Msg
    Do
        Value = InputBox("Geben Sie eine Zahl zwischen 5 und 10 ein.")
        If Value >= 5 And Value <= 10 Then
            Exit Do      ' Zahl OK -> Exit
        Else
            Beep         ' Zahl nicht OK -> nochmal von vorn
        End If
    Loop
End Sub
```

End (Anweisung)

End [{**Function**|**If**|**Sub**}]

Beendet ein Script oder einen Anweisungsblock.

Siehe auch: Exit, **Function**, **If ... Then ... Else**, **Select Case**, **Stop**, **Sub**

Beispiel:

In diesem Beispiel beendet die **End**-Anweisung in der Routine "Test" die Programmausführung.

```
Sub Main
    Dim Var1 as String
    Var1 = "Hallo"
    MsgBox "Test"
    Test Var1
    MsgBox Var1
End Sub

Sub Test(wvar1 as String)
    wvar1 = "Ende"
    MsgBox "Programmende wegen End-Anweisung"
    End
End Sub
```

EOF (Funktion)

EOF(*Dateinummer*)

Liefert **True**, wenn das Dateiende ("End Of File") erreicht wurde.

Dateinummer ist die in der **Open**-Anweisung vergebene Nummer der betreffenden Datei.

Siehe auch: Open

Beispiel:

```
' Jeweils 10 Zeichen aus einer Datei lesen und anzeigen.
' "Testfile" muss bereits existieren.

Sub Main
    Open "TESTFILE" For Input As #1   ' Datei öffnen
    Do While Not EOF(1)               ' Wiederhole bis Dateiende
        MyStr = Input(10, #1)         ' 10 Zeichen lesen
        MsgBox MyStr
    Loop
    Close #1                          ' Datei schließen
End Sub
```

Erase (Anweisung)

Erase *Arrayname* [, ...]

Initialisiert die Elemente eines Arrays neu.

Siehe auch: Dim

Beispiel:

```
Option Base 1

Sub Main
    Dim a(10) As Double
    Dim i As Integer
    For i = 1 to 3
        a(i) = 2 + i
    Next i
    Erase a
    Print a(1), a(2), a(3)   ' Ergebnis: 0 0 0
End Sub
```

Exit (Anweisung)

Exit {**Do**|**For**|**Function**|**Sub**}

Verlässt eine **Do**-Schleife, eine **For**-Schleife, eine Funktion oder eine Prozedur.

Siehe auch: End, **Stop**

Beispiel:

```
Sub Main
    Dim Value, Msg
    Do
        Value = InputBox("Geben Sie eine Zahl zwischen 5 und 10 ein.")
        If Value >= 5 And Value <= 10 Then
            Exit Do      ' Zahl OK -> Schleife verlassen
        Else
            Beep         ' Zahl nicht OK -> nochmal von vorn
        End If
    Loop
End Sub
```

Exp (Funktion)

Exp(*Zahl*)

Berechnet die Exponentialfunktion (e ^ *Zahl*).

Der Wert der Konstanten e (Eulersche Zahl) beträgt etwa 2,71828.

Siehe auch: Log

Beispiel:

```
' Exp(x)=e^x, also ist Exp(1)=e

Sub ExpExample
    Dim Msg, ValueOfE
    ValueOfE = Exp(1)
    Msg = "Der Wert von e beträgt " & ValueOfE
    MsgBox Msg
End Sub
```

FileCopy (Anweisung)

FileCopy *Quelldatei*, *Zieldatei*

Kopiert die Datei *Quelldatei* auf *Zieldatei*.

Die Parameter *Quelldatei* und *Zieldatei* müssen Zeichenketten mit den gewünschten eindeutigen Dateinamen sein. Platzhalter wie "*" oder "?" sind nicht zulässig.

FileLen (Funktion)

FileLen(*Dateiname*)

Liefert die Größe der übergebenen Datei in Bytes (als Long Integer).

Der Parameter *Dateiname* muss eine Zeichenkette mit dem gewünschten Dateinamen sein. Platzhalter wie "*" oder "?" sind nicht zulässig.

Fix (Funktion)

Fix(*Num*)

Ermittelt den ganzzahligen Anteil eines numerischen Ausdrucks.

Der Unterschied zur Funktion **Int** besteht in der Behandlung negativer Zahlen: Während **Int** immer die nächstkleinere ganze Zahl zurückgibt, entfernt **Fix** lediglich die Nachkommastellen (siehe Beispiel).

Siehe auch: Int

Beispiel:

```
Sub Main
    Print Int( 1.4)    ' -> 1
    Print Fix( 1.4)    ' -> 1
    Print Int(-1.4)    ' -> -2
    Print Fix(-1.4)    ' -> -1
End Sub
```

For Each ... Next (Anweisung)

```
For Each Element In Gruppe
   [Anweisungen]
   [Exit For]
   [Anweisungen]
Next [Element]
```

Führt eine Gruppe von Anweisungen für alle Elemente eines Feldes oder einer Sammlung aus.

Element ist eine Variable vom Typ **Variant** (bei Feldern) oder **Object** (bei Sammlungen), die nacheinander die Werte der einzelnen Elemente von *Gruppe* annimmt.

For Each ... **Next** kann nicht mit Feldern von benutzerdefinierten Typen angewendet werden.

Siehe auch: For ... Next, **Exit**, Abschnitt "Arrays", Abschnitt "Sammlungen verwenden"

Beispiel:

```
Sub Main
  Dim z(1 To 4) As Double
  z(1) = 1.11
  z(2) = 2.22
  z(3) = 3.33
  z(4) = 4.44
  For Each v In z
    Print v
  Next v
End Sub
```

For ... Next (Anweisung)

```
For Zähler = Startwert To Endwert [Step Schrittweite]
   [Anweisungen]
   [Exit For]
   [Anweisungen]
Next [Zähler]
```

Führt eine Gruppe von Anweisungen in einer Schleife aus.

Zähler ist die Laufvariable, die bei jedem Durchlauf der Schleife um die angegebene *Schrittweite* erhöht wird.

Startwert ist der Startwert für *Zähler*.

Endwert ist der Endwert.

Schrittweite ist die Schrittweite. Falls nicht angegeben, ist die Schrittweite 1.

Beim ersten Durchlauf der Schleife hat *Zähler* den Wert *Startwert*. Bei jedem weiteren Durchlauf wird *Schrittweite* dazuaddiert. Die Schleife wird beendet, wenn *Endwert* überschritten wird.

Siehe auch: For Each ... Next, **Exit**, Abschnitt "Kontrollstrukturen"

Beispiel:

```
Sub Main
    Dim x, y, z
    For x = 1 To 3
        For y = 1 To 3
            For z = 1 To 3
                Print z, y, x
            Next z
        Next y
    Next x
End Sub
```

Format (Funktion)

Format(*Ausdruck* [,*Format*])

Liefert eine Zeichenkette, die den übergebenen *Ausdruck* in einer frei wählbaren Formatierung enthält.

Das gewünschte Format geben Sie über die Zeichenkette *Format* an. Es stehen eine Reihe vordefinierter Formate zur Verfügung, die auf den nächsten Seiten aufgelistet sind. Über benutzerdefinierte Formate kann die Formatierung noch genauer spezifiziert werden.

Wenn der Parameter *Format* leer ist und *Ausdruck* numerisch ist, ist das Ergebnis von **Format** das gleiche wie das von **Str**, außer dass **Format** bei positiven Zahlen kein Leerzeichen voranstellt.

Für numerische Formate muss *Ausdruck* ein numerischer Ausdruck; für Zeichenkettenformate eine Zeichenkette sein.

Für Datum/Zeit-Formate muss *Ausdruck* eine Zeichenkette sein, wie sie beispielsweise von der Funktion **Now** geliefert wird.

Siehe auch: Str, Abschnitte "Numerische Formate der Format-Funktion", "Datums-/Zeitformate der Format-Funktion" und "Zeichenkettenformate der Format-Funktion"

Beispiel:

```
Sub Main
    MsgBox Format(Date, "long date")
    MsgBox Format(Date, "dd.mm.yy")
End Sub
```

Numerische Formate der Format-Funktion

Die folgende Tabelle listet die vordefinierten numerischen Formate für die **Format**-Funktion auf:

Formatname	Beschreibung
General Number	Ausgabe der unformatierten Zahl
Fixed	Ausgabe mit mindestens einer Stelle vor und genau zwei nach dem Komma
Standard	Ausgabe mit mindestens einer Stelle vor und genau zwei nach dem Komma; zusätzlich Tausender-Trennzeichen, falls die Zahl >= 1000 ist
Percent	Ausgabe mit mindestens einer Stelle vor und genau zwei nach dem Komma; zusätzlich wird die Zahl mit 100 multipliziert und mit einem Prozentzeichen (%) versehen

Scientific	Ausgabe mit mindestens einer Stelle vor und genau zwei nach dem Komma in wissenschaftlicher Notation (Exponentialschreibweise)
True/False	"False", wenn die Zahl Null ist, sonst "True"

Benutzerdefinierte numerische Formate

Benutzerdefinierte numerische Formate können aus den folgenden Zeichen zusammengesetzt werden:

Zeichen	Bedeutung
0	Ziffern-Platzhalter: Ausgabe einer Ziffer der Zahl oder Null. Wenn sich bei der zu formatierenden Zahl an der Stelle, an der in *Format* "0" vorkommt, eine Ziffer befindet, wird diese ausgegeben, sonst wird 0 ausgegeben. Wenn die zu formatierende Zahl links oder rechts vom Dezimaltrennzeichen weniger Stellen hat als in *Format* angegeben, werden führende oder angehängte Nullen angezeigt. Wenn die zu formatierende Zahl rechts vom Dezimaltrennzeichen mehr Stellen hat, als in *Format* angegeben, wird die Zahl auf die entsprechende Stellenzahl gerundet. Wenn die zu formatierende Zahl links von Dezimaltrennzeichen mehr Stellen hat als in *Format* angegeben, werden die zusätzlichen Ziffern immer angezeigt.
#	Ziffern-Platzhalter: Ausgabe einer Ziffer der Zahl beziehungsweise nichts. Wenn sich bei der zu formatierenden Zahl an der Stelle, an der in *Format* "#" vorkommt, eine Ziffer befindet, wird diese ausgegeben, sonst wird nichts ausgegeben.
.	Dezimaltrennzeichen
%	Prozentzeichen. Bewirkt die Ausgabe eines Prozentzeichens (%); weiterhin wird der Ausdruck mit 100 multipliziert.
,	Tausender-Trennzeichen. Sollte die Zahl >= 1000 sein, erscheint dieses Zeichen zwischen Tausendern und Hundertern.
E- E+ e- e+	Wissenschaftliches Format. Wenn *Format* mindestens einen Ziffernplatzhalter (**0** oder **#**) rechts von **E-**, **E+**, **e-** oder **e+** enthält, wird die Zahl in wissenschaftlichen Format formatiert, wobei zwischen Mantisse und Exponent ein E oder e eingefügt wird und die Zahl der Ziffernplatzhalter rechts die Zahl der Ziffern im Exponent bestimmt. Bei **E+**/**e+** wird der Exponent immer mit Vorzeichen ausgegeben, bei **E-**/**e-** nur bei negativem Exponent.
:	Zeit-Trennzeichen. Das ausgegebene Zeichen wird durch das in der Systemsteuerung eingestellte Zeitformat bestimmt.
/	Datums-Trennzeichen. Das ausgegebene Zeichen wird durch das in der Systemsteuerung eingestellte Datumsformat bestimmt.
- + $ ( ) Leerzeichen	Das angegebene Zeichen wird ausgegeben. Um ein anderes Zeichen auszugeben, muss diesem ein umgekehrter Schrägstrich \ vorangestellt werden oder das/die Zeichen mit Anführungsstrichen umgeben werden.
\	Das nachfolgende Zeichen wird ausgegeben. Der umgekehrte Schrägstrich selbst wird nicht ausgegeben. Um einen umgekehrten Schrägstrich auszugeben, muss dieser doppelt geschrieben werden (\\). Hinweis: Anführungszeichen können in Formatstrings generell nicht verwendet werden; auch \" führt zu einer Fehlermeldung.
"*Text*"	Die Zeichenkette innerhalb der Anführungszeichen wird ausgegeben. Die Anführungszeichen selbst werden nicht ausgegeben.
*	Bestimmt das nachfolgende Zeichen als Füllzeichen. Leerstellen werden mit diesem Zeichen ausgefüllt.

Benutzerdefinierte numerische Formate können dabei aus bis zu vier Abschnitten bestehen:

Abschnitte	Ergebnis
1 Abschnitt	Dieses Format gilt für alle Werte.
2 Abschnitte	Das im ersten Abschnitt angegebene Format gilt für positive Werte und den Wert 0, das im zweiten Abschnitt für negative Werte.
3 Abschnitte	Das erste Format gilt für positive, das zweite für negative Werte und das dritte für den Wert 0.

4 Abschnitte	Das erste Format gilt für positive, das zweite für negative Werte, das dritte für den Wert 0 und das vierte für Null-Werte (siehe **IsNull**-Funktion).

Wird einer dieser Abschnitte nicht angegeben, wird das Format für positive Zahlen übernommen.

Die einzelnen Abschnitte sind durch Strichpunkte zu trennen.

Beispiele

Die folgende Tabelle zeigt einige Beispiele. Links ist der Format-Ausdruck angegeben, rechts die Ergebnisse bei den Zahlen 3, -3, 0.3 und bei NULL-Werten.

Format	3	-3	0.3	NULL
(leer)	3	-3	0.3	
"0"	3	-3	0	
"0.00"	3,00	-3,00	0,30	
"#,##0"	3	-3	0	
"$#,##0;($#,##0)"	$3	($3)	$0	
"$#,##0.00;($#,##0.00)"	$3,00	($3,00)	$0,30	
"0%"	300%	-300%	30%	
"0.00%"	300,00%	-300,00%	30,00%	
"0.00E+00"	3,00E+00	-3,00E+00	3,00E-01	
"0.00E-00"	3,00E00	-3,00E00	3,00E-01	

Datums-/Zeitformate der Format-Funktion

Datums- und Zeitangaben sind nichts anderes als Fließkommazahlen. Die Stellen vor dem Komma geben das Datum an, die Stellen hinter dem Komma die Zeit. Hat die Zahl keine Nachkommastellen, enthält sie nur ein Datum. Hat sie keine Vorkommastellen, enthält sie nur eine Uhrzeit.

Datum- und Zeitangaben können mit vordefinierten und benutzerdefinierten Formatanweisungen formatiert werden.

Die folgende Tabelle listet die vordefinierten Datum/Zeit-Formate für die **Format**-Funktion auf:

Formatname	Beschreibung
General Date	Gibt das Datum und/oder Uhrzeit unformatiert aus (also normalerweise im kurzen Datumsformat).
Short Date	Gibt das Datum im kurzen Datumsformat aus.
Medium Date	Gibt das Datum mit auf drei Buchstaben gekürztem Monatsnamen aus.
Long Date	Gibt das Datum im langen Datumsformat aus.
Short Time	Gibt die Uhrzeit im kurzen Uhrzeitformat aus.
Medium Time	Gibt die Uhrzeit im 12-Stunden-Format aus (hh:mm AM|PM).
Long Time	Gibt die Uhrzeit im langen Uhrzeitformat aus.

Benutzerdefinierte Datums- und Zeitformate

Benutzerdefinierte Formate können aus den nachfolgenden Zeichen zusammengesetzt werden.

Wichtig: Achten Sie genau auf die Groß-/Kleinschreibung.

Zeichen	Bedeutung
c	Liefert das komplette Datum im kurzen Datumsformat und die komplette Uhrzeit im Format hh:nn:ss
d	Liefert den Tag als Zahl (1-31)
dd	Liefert den Tag als zweistellige Zahl (01-31)
ddd	Liefert den Wochentag auf drei Buchstaben gekürzt (Son-Sam)
dddd	Liefert den Wochentag (Sonntag-Samstag)
ddddd	Liefert das komplette Datum im kurzen Datumsformat
dddddd	Liefert das komplette Datum im langen Datumsformat
w	Liefert den Wochentag als Zahl (1-7), 1=Sonntag, 2=Montag, ... 7=Samstag
m	Liefert den Monat als Zahl (1-12)
mm	Liefert den Monat als zweistellige Zahl (01-12)
mmm	Liefert den Monatsnamen auf drei Buchstaben gekürzt Abkürzung (Jan-Dez)
mmmm	Liefert den Monatsnamen (Januar-Dezember)
q	Liefert die Quartalsnummer (1-4)
yy	Liefert das Jahr als zweistellige Zahl mit führender Null (00-99)
yyyy	Liefert das Jahr als drei- bis vierstellige Zahl (100-9999)
h	Liefert die Stunde als Zahl (0-23)
hh	Liefert die Stunde als zweistellige Zahl (00-23)
n	Liefert die Minute als Zahl (0-59)
nn	Liefert die Minute als zweistellige Zahl (00-59)
s	Liefert die Sekunde als Zahl (0-59)
ss	Liefert die Sekunde als zweistellige Zahl (00-59)
AM/PM	12-Stunden-Format mit AM bzw. PM verwenden
am/pm	12-Stunden-Format mit am bzw. pm verwenden
A/P	12-Stunden-Format mit A bzw. P verwenden
a/p	12-Stunden-Format mit a bzw. p verwenden

Beispiele

Nachfolgend einige Beispiele:

Format	Ausgabe am 26.2.2010, 18:45:15
"d.m.yy"	26.2.10
"d. mmmm yyyy"	26. Februar 2010
"hh:nn AM/PM"	06:45 PM
"hh:nn:ss"	18:45:15

Zeichenkettenformate der Format-Funktion

Bei Zeichenketten können mit der **Format**-Funktion benutzerdefinierte Formate aus den folgenden Zeichen zusammengesetzt werden:

Zeichen	Bedeutung
@	Gibt ein Zeichen oder ein Leerzeichen aus. Die Ausgabe erfolgt normalerweise rechtsbündig (siehe jedoch !-Zeichen).
&	Gibt ein Zeichen oder nichts aus.
<	Alle Zeichen als Kleinbuchstaben ausgeben.
>	Alle Zeichen als Großbuchstaben ausgeben.
!	Das Ausrufezeichen schaltet auf linksbündige Ausgabe.

FreeFile (Funktion)

FreeFile [()]

Gibt die Nummer des nächsten freien Dateizeigers zurück. Das Ergebnis ist eine ganze Zahl zwischen 1 und 255.

Dateizeiger werden zum Öffnen von Dateien benötigt (siehe **Open**-Anweisung).

Siehe auch: Open

Beispiel:

```
Sub Main
   A = FreeFile
   Open "TESTFILE" For Output As #A
   Write #A, "Test"
   Close #A
   Kill "TESTFILE"
End Sub
```

Function (Anweisung)

Function *Name* [(*Argumentliste*)] [**As** *Typ*]
   [*Anweisungen*]
   Name **=** *Ausdruck*
   [*Anweisungen*]
   Name **=** *Ausdruck*
End Function

Leitet die Definition einer benutzerdefinierten Funktion ein.

Name ist der Name der Funktion.

Argumentliste ist eine durch Kommata getrennte Liste von Parameterdeklarationen (siehe unten).

Typ spezifiziert den Datentyp (**String**, **Integer**, **Double**, **Long**, **Variant**). Alternativ kann der Typ auch durch einen Typsuffix (z.B. % für **Integer**) am Funktionsnamen angegeben werden (siehe Abschnitt "Datentypen").

Die Funktionsdefinition wird mit **End Function** beendet. Die Anweisung **Exit Function** kann dazu verwendet werden, eine Funktion vorzeitig verlassen.

Parameterdeklaration

[**ByVal** | **ByRef**] *Variable* [**As** *Typ*]

Mit **ByVal** beziehungsweise **ByRef** (Standardwert) wird bestimmt, ob der Parameter als Wert oder als Referenz übergeben wird (siehe Abschnitt "Parameterübergabe ByRef oder ByVal").

Typ spezifiziert den Datentyp (**String**, **Integer**, **Double**, **Long**, **Variant**). Alternativ kann der Typ auch durch einen Typsuffix (z.B. % für **Integer**) angegeben werden (siehe Abschnitt "Datentypen").

Siehe auch: Dim, **End**, **Exit**, **Sub**

Beispiel:

```
Sub Main
   For I% = 1 to 10
   Print GetColor2(I%)
   Next I
End Sub

Function GetColor2(c%) As Long
   GetColor2 = c% * 25
   If c% > 2 Then
      GetColor2 = 255           ' 0x0000FF - Rot
   End If
   If c% > 5 Then
      GetColor2 = 65280         ' 0x00FF00 - Grün
   End If
   If c% > 8 Then
      GetColor2 = 16711680      ' 0xFF0000 - Blau
   End If
End Function
```

GetObject (Funktion)

GetObject(*Name* [,*Klasse*])

Liefert eine Referenz auf ein OLE-Objekt, das bereits erzeugt worden ist.

Name ist der Name einer Datei, die das Objekt enthält. Wenn *Name* leer ist, muss *Klasse* angegeben werden.

Klasse ist der Name, unter dem das Objekt in der Windows-Registry bekannt ist.

Siehe auch: CreateObject, Set, Abschnitt "OLE Automation"

Gosub ... Return (Anweisung)

```
Gosub Label
.
.
.
Label:
   Anweisung(en)
Return
```

Gosub springt zu einer durch das Sprungziel *Label* gekennzeichneten Stelle im Script; **Return** kehrt wieder zurück.

Das Sprungziel *Label* muss sich innerhalb der gleichen Subroutine oder Funktion wie der **Gosub**-Aufruf befinden.

Hinweis: **Gosub** ... **Return** wird nur aus Kompatibilitätsgründen zu älteren Basic-Varianten noch unterstützt. Es ist übersichtlicher, die **Sub**-Anweisung für Unterprogramme zu verwenden.

Siehe auch: Goto, **Sub**, Abschnitt "Kontrollstrukturen"

Beispiel:

```
Sub Main
    Print "Hauptprogramm"
    Gosub Abstecher
    Exit Sub

Abstecher:
```

```
    Print "Unterprogramm"
Return

End Sub
```

Goto (Anweisung)

Goto *Label*
.
.
.
Label:
Anweisungen

Unbedingter Sprung zum Sprungziel *Label*.

Das Sprungziel *Label* muss sich innerhalb der gleichen Subroutine oder Funktion wie der **Goto**-Aufruf befinden.

Hinweis: Diese Anweisung wird nur aus Kompatibilitätsgründen noch unterstützt. Der Gebrauch von **Goto**-Anweisungen macht den Programmcode schnell unübersichtlich. Sie sollten stattdessen die strukturierten Kontrollanweisungen (**Do** ... **Loop**, **For** ... **Next**, **If ... Then ... Else**, **Select Case**) verwenden.

Siehe auch: Gosub ... Return, **Sub**, Abschnitt "Kontrollstrukturen"

Beispiel:

```
Sub Main
    Dim x
    For x = 1 to 5
        Print x
        If x > 3 Then
            Goto Label1
        End If
    Next x

Label1:

    Print "Muss genügen!"

End Sub
```

Hex (Funktion)

Hex(*Num*)

Liefert eine Zeichenkette mit der hexadezimalen Darstellung der angegebenen Zahl.

Num kann ein beliebiger numerischer Ausdruck sein; er wird auf die nächste ganze Zahl gerundet.

Das Ergebnis kann maximal achtstellig werden.

Siehe auch: Oct

Beispiel:

```
Sub Main
    Dim Msg As String, x%
    x% = 1024
    Msg =Str(x%) & " dezimal entspricht "
    Msg = Msg & Hex(x%) & " hexadezimal."
    MsgBox Msg
End Sub
```

Hour (Funktion)

Hour(*Ausdruck*)

Liefert die Stunde der angegebenen Zeit als ganze Zahl.

Ausdruck ist ein numerischer oder Zeichenkettenausdruck, der eine Zeit repräsentiert.

Siehe auch: Date, **Day**, **Minute**, **Month**, **Now**, **Second**, **Time**, **Weekday**, **Year**

Beispiel:

```
Sub Main

    T1 = Now     ' Now = aktuelles Datum + Uhrzeit

    MsgBox T1

    MsgBox "Tag: " & Day(T1)
    MsgBox "Monat: " & Month(T1)
    MsgBox "Jahr: " & Year(T1)

    MsgBox "Stunde: " & Hour(T1)
    MsgBox "Minute: " & Minute(T1)
    MsgBox "Sekunde: " & Second(T1)

End Sub
```

If ... Then ... Else (Anweisung)

```
If Bedingung Then
   [Anweisungen]
[ElseIf Bedingung Then
   [Anweisungen]]...
[Else
   [Anweisungen]]
End If
```

Oder:

```
If Bedingung Then Anweisung [Else Anweisung]
```

Führt eine Gruppe von Anweisungen aus, wenn *Bedingung* wahr ist. Optional wird eine andere Gruppe von Anweisungen ausgeführt, wenn *Bedingung* falsch ist (siehe auch Abschnitt "Kontrollstrukturen").

Siehe auch: Select Case, Abschnitt "Kontrollstrukturen"

Beispiel:

```
Sub IfTest
    Dim Geschlecht as String

    Geschlecht = InputBox("Geben Sie Ihr Geschlecht ein (m oder w)")

    If Geschlecht = "m" Then
        MsgBox "Sie sind männlich."
    ElseIf Geschlecht = "w" Then
        MsgBox "Sie sind weiblich."
    Else
        MsgBox "Bitte geben Sie entweder m oder w ein!"
    End If

End Sub
```

Input (Funktion)

Input(*n*, [#]*Dateinummer*)

Liest eine Zeichenkette aus einer Datei.

n ist die Anzahl der zu lesenden Zeichen (Bytes).

Dateinummer ist die in der **Open**-Anweisung vergebene Nummer der betreffenden Datei.

Siehe auch: Line Input, **Open**, **Seek**

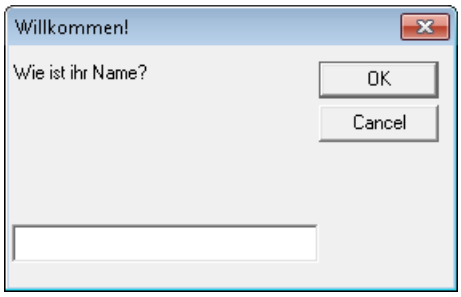
Beispiel:

```
Sub Main
    Open "TESTFILE" For Input As #1    ' Datei öffnen
    Do While Not EOF(1)                ' Schleife bis zum Dateiende
        MyStr = Input(10, #1)          ' 10 Zeichen lesen
        MsgBox MyStr
    Loop
    Close #1                           ' Datei schließen
End Sub
```

InputBox (Funktion)

InputBox(*Prompt$* [, [*Titel$*] [,[*Default$*] [,*X*, *Y*]]])

Zeigt ein Dialogfenster an, in dem der Anwender etwas eingeben kann. Das Resultat ist eine Zeichenkette mit der Benutzereingabe.



Prompt$ ist die im Dialog anzuzeigende Zeichenkette.

Die weiteren Parameter sind optional:

Titel$ ist die in der Titelleiste anzuzeigende Zeichenkette.

Default$ ist die Zeichenkette, mit dem das Eingabefeld vorbelegt wird.

X und *Y* sind die Bildschirmkoordinaten des Eingabedialogs in Bildschirmpixeln.

Siehe auch: Dialog

Beispiel:

```
Sub Main
    Title$ = "Willkommen!"
    Prompt$ = "Wie ist ihr Name?"
    Default$ = ""
    X% = 100
    Y% = 200
    N$ = InputBox(Prompt$, Title$, Default$, X%, Y%)
    MsgBox "Guten Tag, " & N$ & "!"
End Sub
```

InStr (Funktion)

InStr(*Start*, *String*, *Suchstring*)

Ermittelt die Position, an der die Zeichenkette *SuchString* in der Zeichenkette *String* zum ersten Mal vorkommt.

Start ist die Position, an der die Suche beginnen soll; geben Sie 1 an, um die ganze Zeichenkette zu durchsuchen. *Start* muss eine positive Integer-Zahl sein.

String ist die zu durchsuchende Zeichenkette.

Suchstring ist die Zeichenkette, nach der gesucht werden soll.

Siehe auch: Mid, **StrComp**

Beispiel:

```
Sub Main
    B$ = "SoftMaker Basic"
    A = InStr(2, B$, "Basic")
    MsgBox A
End Sub
```

Int (Funktion)

Int(*Num*)

Ermittelt den ganzzahligen Anteil eines numerischen Ausdrucks.

Der Unterschied zur Funktion **Fix** besteht in der Behandlung negativer Zahlen: Während **Int** immer die nächstkleinere ganze Zahl zurückgibt, entfernt **Fix** lediglich die Nachkommastellen (siehe Beispiel).

Siehe auch: Fix

Beispiel:

```
Sub Main
    Print Int( 1.4)    ' -> 1
    Print Fix( 1.4)    ' -> 1
    Print Int(-1.4)    ' -> -2
    Print Fix(-1.4)    ' -> -1
End Sub
```

IsDate (Funktion)

IsDate(*Variant*)

Prüft, ob die übergebene Variant-Variable in ein Datum umgewandelt werden kann.

Siehe auch: IsEmpty, **IsNull**, **IsNumeric**, **VarType**

IsEmpty (Funktion)

IsEmpty(*Variant*)

Prüft, ob die übergebene Variant-Variable initialisiert wurde.

Siehe auch: IsDate, **IsNull**, **IsNumeric**, **VarType**, Abschnitt "Besonderheiten beim Variant-Datentyp"

Beispiel:

```
Sub Main

    Dim x       ' Empty, da noch kein Wert zugewiesen
    MsgBox "IsEmpty(x): " & IsEmpty(x)

    x = 5       ' Nicht mehr Empty
    MsgBox "IsEmpty(x): " & IsEmpty(x)

    x = Empty   ' Wieder Empty
    MsgBox "IsEmpty(x): " & IsEmpty(x)

End Sub
```

IsNull (Funktion)

IsNull(*Variant*)

Prüft, ob die übergebene Variant-Variable den Wert "Null" trägt.

Der spezielle Wert "Null" zeigt an, dass die Variable keinen Wert enthält. Unterscheiden Sie diesen Wert vom numerischen Wert 0, von einer leeren Zeichenkette und vom speziellen Wert **Empty**, der anzeigt, dass eine Variable nicht initialisiert wurde.

Siehe auch: IsDate, **IsEmpty**, **IsNumeric**, **VarType**, Abschnitt "Besonderheiten beim Variant-Datentyp"

IsNumeric (Funktion)

IsNumeric(*Variant*)

Prüft, ob die übergebene Variant-Variable in eine Zahl umgewandelt werden kann.

Siehe auch: IsDate, **IsEmpty**, **IsNull**, **VarType**

Beispiel:

```
Sub Test
    Dim TestVar
    TestVar = InputBox("Geben Sie eine Zahl oder Text ein:")
    If IsNumeric(TestVar) Then
        MsgBox "Eingabe ist numerisch."
    Else
        MsgBox "Eingabe ist nicht numerisch."
    End If
End Sub
```

Kill (Anweisung)

Kill *Dateiname*

Löscht die angegebene(n) Datei(en).

In *Dateiname* sind auch Platzhalter wie "*" und "?" erlaubt. Folgendermaßen werden beispielsweise alle Dateien mit der Namenserweiterung "bak" entfernt:

```
Kill "*.bak"
```

Siehe auch: RmDir

Beispiel:

```
Const NumberOfFiles = 3
```

```
Sub Main
    Dim Msg                   ' Variablen deklarieren
    Call MakeFiles()          ' Dateien anlegen
    Msg = "Einige Testdateien wurden angelegt. "
    Msg = Msg & "Klicken Sie auf OK, werden sie wieder entfernt."
    MsgBox Msg
    For I = 1 To NumberOfFiles
        Kill "TEST" & I       ' Dateien löschen
    Next I
End Sub

Sub MakeFiles()
    Dim I, FNum, FName        ' Variablen deklarieren
    For I = 1 To NumberOfFiles
        FNum = FreeFile       ' Nächsten freien Dateizeiger holen
        FName = "TEST" & I
        Open FName For Output As Fnum       ' Datei öffnen
        Print #FNum, "This is test #" & I   ' In Datei schreiben
        Print #FNum, "Here is another ";  "line";  I
    Next I
    Close                     ' Alle Dateien wieder schließen
End Sub
```

LBound (Funktion)

LBound(*Array* [,*Dimension*])

Ermittelt den kleinsten Index der angegebenen Dimension eines Arrays.

Wird *Dimension* nicht angegeben, wird die erste Dimension des Feldes verwendet.

Siehe auch: Dim, **Option Base**, **ReDim**, **UBound**

Beispiel:

```
Option Base 1

Sub Main
    Dim a(10,20)
    Print "1. Dimension: " & LBound(a) & " bis " & UBound(a)
    Print "2. Dimension: " & LBound(a, 2) & " bis " & UBound(a, 2)
End Sub
```

LCase (Funktion)

LCase(*String*)

Wandelt eine Zeichenkette in Kleinbuchstaben um.

Siehe auch: UCase

Beispiel:

```
Sub Main
   MsgBox LCase("Think BIG!")   ' ergibt "think big!"
End Sub
```

Left (Funktion)

Left(*String*, *n*)

Liefert eine Zeichenkette, die aus den ersten *n* Zeichen der übergebenen Zeichenkette besteht.

Siehe auch: Len, **Mid**, **Right**

Beispiel:

```
Sub Main
    Dim LWord, Msg, RWord, SpcPos, UsrInp
    Msg = "Geben Sie zwei durch ein Leerzeichen "
    Msg = Msg & "getrennte Wörter ein."
    UsrInp = InputBox(Msg)
    SpcPos = InStr(1, UsrInp, " ")   ' Leerzeichen finden
    If SpcPos Then
        LWord = Left(UsrInp, SpcPos - 1)    ' Linkes Wort
        RWord = Right(UsrInp, Len(UsrInp) - SpcPos)   ' Rechtes Wort
        Msg = "Das erste Wort ist " & LWord & ","
        Msg = Msg & " das zweite ist " & RWord & "."
    Else
        Msg = "Das waren nicht 2 Wörter."
    End If
    MsgBox Msg
End Sub
```

Len (Funktion)

Len(*String*)

Ermittelt die Länge einer Zeichenkette.

Siehe auch: InStr

Beispiel:

```
Sub Main
    A$ = "BasicMaker"
    StrLen = Len(A$)     'Ergebnis: 10
    MsgBox StrLen
End Sub
```

Let (Anweisung)

[**Let**] *Variable = Ausdruck*

Weist einer Variablen einen Wert zu.

Das Schlüsselwort **Let** war nur in älteren Versionen von Basic nötig. Es wird heutzutage normalerweise weggelassen.

Beispiel:

```
Sub Main
    Dim Msg, Pi
    Let Pi = 4 * Atn(1)
    Msg = "Pi = " & Str(Pi)
    MsgBox Msg
End Sub
```

Line Input # (Anweisung)

Line Input [#]*Dateinummer, Name*

Liest eine Zeile von einer Datei in die String- oder Variant-Variable *Name*.

Dateinummer ist die in der **Open**-Anweisung vergebene Nummer. Die Datei muss mit **Open** zum Lesen geöffnet worden sein.

Die **Line Input**-Anweisung liest solange Zeichen aus der Datei, bis sie auf einen Zeilenvorschub oder die Kombination Wagenrücklauf + Zeilenvorschub trifft.

Siehe auch: Input, **Open**, **Seek**

Beispiel:

```
Sub Main
    Open "c:\autoexec.bat" For Input As #1   ' Datei öffnen
    While Not EOF(1)               ' Schleife bis Dateiende
        Line Input #1, TextLine    ' Zeile aus Datei einlesen
        Print TextLine             ' Zeile ausgeben
    Wend
    Close #1                       ' Datei schließen
End Sub
```

Log (Funktion)

Log(*Num*)

Berechnet den natürlichen Logarithmus einer Zahl.

Der Parameter *Num* muss größer als 0 sein.

Siehe auch: Exp

Beispiel:

```
Sub Main
    For I = 1 to 3
        Print Log(I)
    Next I
End Sub
```

Mid (Funktion)

Mid(*String*, *Start* [, *Länge*])

Liefert eine Teilzeichenkette von *String*. Sie beginnt bei Position *Start* und ist *Länge* Zeichen lang. Ist *Länge* leer, wird der komplette Rest der Zeichenkette geliefert.

Siehe auch: Len, **Left**, **Right**

Beispiel:

```
Sub Main
   MidTest = Mid("Kartoffelsalat", 8, 4)
   MsgBox MidTest   'Ergebnis: "elsa"
End Sub
```

Minute (Funktion)

Minute(*Ausdruck*)

Liefert die Minute der angegebenen Uhrzeit als ganze Zahl.

Ausdruck ist ein numerischer oder Zeichenkettenausdruck, der eine Zeit repräsentiert.

Siehe auch: Date, **Day**, **Hour**, **Month**, **Now**, **Second**, **Time**, **Weekday**, **Year**

Beispiel:

```
Sub Main

    T1 = Now     ' Now = aktuelles Datum + Uhrzeit

    MsgBox T1

    MsgBox "Tag: " & Day(T1)
    MsgBox "Monat: " & Month(T1)
    MsgBox "Jahr: " & Year(T1)

    MsgBox "Stunde: " & Hour(T1)
    MsgBox "Minute: " & Minute(T1)
    MsgBox "Sekunde: " & Second(T1)

End Sub
```

MkDir (Anweisung)

MkDir *Pfad*

Erzeugt ein neues Verzeichnis.

Der übergebene *Pfad* darf maximal 255 Zeichen umfassen.

Siehe auch: ChDir, **ChDrive**, **RmDir**

Beispiel:

```
Sub Main
   ChDir "c:\"
   MkDir "Test"
   MsgBox "Das Verzeichnis c:\Test wurde angelegt."
End Sub
```

Month (Funktion)

Month(*Ausdruck*)

Liefert den Monat des angegebenen Datums als ganze Zahl.

Ausdruck ist ein numerischer oder Zeichenkettenausdruck, der ein Datum repräsentiert.

Siehe auch: Date, **Day**, **Hour**, **Minute**, **Now**, **Second**, **Time**, **Weekday**, **Year**

Beispiel:

```
Sub Main

    T1 = Now     ' Now = aktuelles Datum + Uhrzeit

    MsgBox T1

    MsgBox "Tag: " & Day(T1)
    MsgBox "Monat: " & Month(T1)
    MsgBox "Jahr: " & Year(T1)

    MsgBox "Stunde: " & Hour(T1)
    MsgBox "Minute: " & Minute(T1)
    MsgBox "Sekunde: " & Second(T1)

End Sub
```

MsgBox (Funktion)

MsgBox(*Text* [, *Typ*] [, *Titel*])

Zeigt ein Meldungsfenster an.

Der Rückgabewert zeigt an, welche Schaltfläche betätigt wurde, um das Meldungsfenster zu verlassen (siehe unten).

Text ist die anzuzeigende Zeichenkette.

Der optionale Parameter *Typ* bestimmt, welche Schaltflächen und welches Symbol im Meldungsfenster angezeigt werden (siehe unten). Die Standardeinstellung ist: nur **OK**-Schaltfläche, kein Symbol.

Der optionale Parameter *Titel* bestimmt, welcher Text in der Titelleiste angezeigt wird (Standardwert: leer).

Siehe auch: Dialog, **InputBox**

Zulässige Werte für den Parameter "Typ":

Symbolische Konstante	Wert	Bedeutung
MB_OK	0	Nur **OK**-Schaltfläche anzeigen
MB_OKCANCEL	1	Schaltflächen **OK** und **Abbrechen** anzeigen
MB_ABORTRETRYIGNORE	2	Schaltflächen **Abbrechen**, **Wiederholen**, **Ignorieren** anzeigen
MB_YESNOCANCEL	3	Schaltflächen **Ja**, **Nein**, **Abbrechen** anzeigen
MB_YESNO	4	Schaltflächen **Ja** und **Nein** anzeigen
MB_RETRYCANCEL	5	Schaltflächen **Wiederholen** und **Abbrechen** anzeigen
MB_ICONSTOP	16	Zeigt ein Symbol für Fehlermeldungen an.
MB_ICONQUESTION	32	Zeigt ein Symbol für Abfragen an.
MB_ICONEXCLAMATION	48	Zeigt ein Symbol für Hinweismeldungen an.
MB_ICONINFORMATION	64	Zeigt ein Symbol für Informationsmeldungen an.
MB_DEFBUTTON1	0	Macht die erste Schaltfläche zur Standardschaltfläche.
MB_DEFBUTTON2	256	Macht die zweite Schaltfläche zur Standardschaltfläche.
MB_DEFBUTTON3	512	Macht die dritte Schaltfläche zur Standardschaltfläche.
MB_APPLMODAL	0	Das Meldungsfenster ist applikationsmodal. Die *aktuelle Anwendung* nimmt also so lange keine Eingaben an, bis der Anwender das Meldungsfenster geschlossen hat.
MB_SYSTEMMODAL	4096	Das Meldungsfenster ist systemmodal. Das *gesamte System* nimmt so lange keine Eingaben an, bis der Anwender das Meldungsfenster geschlossen hat (nur für kritische Fehler verwenden!).

Aus jeder der vier obigen Gruppen kann ein Wert ausgewählt werden. Kombinieren Sie die einzelnen Konstanten durch Addition.

Rückgabewerte der MsgBox-Funktion

Der Rückgabewert dieser Funktion gibt an, welche Schaltfläche zum Verlassen des Meldungsfensters betätigt wurde:

Symbolische Konstante	Wert	Bedeutung
IDOK	1	**OK**-Schaltfläche
IDCANCEL	2	**Abbrechen**-Schaltfläche, außer bei MB_ABORTRETRYIGNORE
IDABORT	3	**Abbrechen**-Schaltfläche bei MB_ABORTRETRYIGNORE

IDRETRY	4	**Wiederholen**-Schaltfläche
IDIGNORE	5	**Ignorieren**-Schaltfläche
IDYES	6	**Ja**-Schaltfläche
IDNO	7	**Nein**-Schaltfläche

Beispiel:

Dieses Beispiel benutzt **MsgBox**, um eine Sicherheitsabfrage anzuzeigen.

```
Sub Main
    Dim DgDef, Msg, Response, Title
    Title = "MsgBox-Beispiel"
    Msg = "Möchten Sie fortfahren?"
    DgDef = MB_YESNOCANCEL + MB_ICONQUESTION + MB_DEFBUTTON3

    Response = MsgBox(Msg, DgDef, Title)
    If Response = IDYES Then
        Msg = "Sie haben Ja gewählt."
    ElseIf Response = IDCANCEL Then
        Msg = "Sie haben Abbrechen gewählt."
    Else
        Msg = "Sie haben Nein gewählt."
    End If

    MsgBox Msg

End Sub
```

Name (Anweisung)

Name *AlterName* **As** *NeuerName*

Benennt die Datei *AlterName* in *NeuerName* um.

Jeder der beiden Parameter muss eine einzelne Datei bezeichnen. Platzhalter wie "*" und "?" sind nicht erlaubt.

Siehe auch: ChDir, **Kill**

Beispiel:

```
Sub Main
    Name "testfile" As "newtest"
End Sub
```

Now (Funktion)

Now[()]

Ermittelt die aktuelle Systemzeit (Datum und Uhrzeit).

Die **Now**-Funktion liefert ein Ergebnis vom Typ Variant, das Datum und Uhrzeit enthält. Die Vorkommastellen repräsentieren das Datum, die Nachkommastellen die Zeit.

Siehe auch: Date, **Day**, **Hour**, **Minute**, **Month**, **Second**, **Time**, **Weekday**, **Year**

Beispiel:

```
Sub Main

    T1 = Now      ' Now = aktuelles Datum + Uhrzeit

    MsgBox T1
```

```
    MsgBox "Tag: " & Day(T1)
    MsgBox "Monat: " & Month(T1)
    MsgBox "Jahr: " & Year(T1)

    MsgBox "Stunde: " & Hour(T1)
    MsgBox "Minute: " & Minute(T1)
    MsgBox "Sekunde: " & Second(T1)

End Sub
```

Oct (Funktion)

Oct(*Num*)

Liefert eine Zeichenkette mit der oktalen Darstellung der angegebenen Zahl.

Num kann ein beliebiger numerischer Ausdruck sein; er wird auf die nächste ganze Zahl gerundet.

Siehe auch: Hex

Beispiel:

```
Sub Main
    Dim Msg, Num
    Num = InputBox("Geben Sie eine Zahl ein.")
    Msg = Num & " dezimal entspricht "
    Msg = Msg & Oct(Num) & " oktal."
    MsgBox Msg
End Sub
```

On Error (Anweisung)

On Error Goto *Label*

Oder:

On Error Resume Next

Oder:

On Error Goto 0

Aktiviert eine Fehlerbehandlungsroutine zur Behandlung von Laufzeitfehlern:

- Bei **On Error Goto** *Label* wird im Falle eines Laufzeitfehlers beim angegebenen Sprungziel *Label* fortgefahren.
- Bei **On Error Resume Next** werden Laufzeitfehler einfach ignoriert. ***Vorsicht:*** Hierbei können im Falle eines Laufzeitfehlers unvorhersehbare Ergebnisse geliefert werden.
- Bei **On Error Goto 0** wird die Fehlerbehandlung deaktiviert – Laufzeitfehler führen dann wieder wie gewohnt zum Programmabbruch mit einer Fehlermeldung.

Eine **On Error**-Anweisung gilt nur innerhalb der Subroutine oder Funktion, in der sie sich befindet.

Wurde mit **On Error Goto** zu einem Label verzweigt, kann die Script-Ausführung von dort aus mit der **Resume**-Anweisung wieder aufgenommen werden. Das Script wird dann in der nächsten Zeile fortgesetzt.

Siehe auch: Resume

Beispiel:

In diesem Beispiel wird absichtlich ein Fehler ausgelöst, um die Fehlerbehandlungsroutine beim Label "Fehler" auszulösen. Darin wird der Anwender gefragt, ob die Ausführung des Scripts fortgesetzt werden soll. Bei "Ja" wird das Script mit **Resume Next** in der nächsten Zeile nach dem Laufzeitfehler fortgesetzt, bei "Nein" wird die Ausführung mit **Stop** beendet.

```
Sub Main
    On Error Goto Fehler
    Print 1/0     'Fehler (Division durch Null) auslösen
    MsgBox "Ende"
    Exit Sub

Fehler:           'Fehlerbehandlungsroutine
    Dim DgDef, Msg, Response, Title
    Title = "Fehler"
    Msg = "Es ist ein Laufzeitfehler aufgetreten. Möchten Sie fortfahren?"
    DgDef = MB_YESNO + MB_ICONEXCLAMATION

    Response = MsgBox(Msg, DgDef, Title)
    If Response = IDYES Then
        Resume Next
    Else
        Stop
    End If

End Sub
```

Zu Testzwecken können Laufzeitfehler mit dem Befehl **Err.Raise** künstlich ausgelöst werden.

Syntax: **`Err.Raise`** *Nummer*

Für *Nummer* ist die Fehlernummer anzugeben:

3: "RETURN ohne GOSUB"
5: "Ungültiger Funktionsaufruf"
6: "Überlauf"
7: "Speicher voll"
9: "Subskript außerhalb des gültigen Bereichs"
10: "Array hat feste Größe oder ist vorübergehend verriegelt"
11: "Division durch Null"
13: "Typen passen nicht zueinander"
14: "Stringspeicher voll"
16: "Ausdruck zu komplex"
17: "Operation kann nicht ausgeführt werden"
18: "Durch Benutzer abgebrochen"
20: "RESUME ohne Fehler"
28: "Stapelspeicher voll"
35: "SUB, FUNCTION oder PROPERTY nicht definiert"
47: "Zu viele DLL-Klienten"
48: "Fehler beim Laden der DLL"
49: "Fehlerhafter DLL-Aufruf"
51: "Interner Fehler"
52: "Ungültiger Dateiname oder ungültige Dateinummer"
53: "Datei nicht gefunden"
54: "Ungültiger Dateimodus"
55: "Diese Datei ist bereits geöffnet"
57: "Fehler bei Ein-/Ausgabe auf Gerät"
58: "Datei existiert bereits"
59: "Fehlerhafte Datensatzlänge"
60: "Datenträger voll"
62: "Versuch, hinter dem Dateiende zu lesen"
63: "Ungültige Satznummer"
67: "Zu viele Dateien geöffnet"
68: "Gerät nicht verfügbar"
70: "Zugriff verweigert"
71: "Datenträger nicht bereit"
74: "Umbenennen über Laufwerke hinweg nicht möglich"
75: "Fehler beim Zugriff auf Pfad oder Datei"
76: "Pfad nicht gefunden"
91: "Objektvariable oder WITH-Block-Variable nicht gesetzt"
92: "FOR-Schleife nicht initialisiert"
93: "Fehlerhafter Musterstring"
94: "Fehlerhafte Benutzung von NULL"

OLE Automation-Fehler

424: "Objekt benötigt"
429: "Der OLE-Automation-Server kann das Objekt nicht erzeugen"
430: "Klasse unterstützt keine OLE-Automation"
432: "Datei- oder Klassenname während OLE-Automation-Operation nicht gefunden"
438: "Das Objekt unterstützt diese Eigenschaft oder Methode nicht"
440: "OLE-Automation-Fehler"
443: "Das OLE-Automation-Objekt hat keinen voreingestellten Wert"
445: "Das Objekt unterstützt diese Aktion nicht"
446: "Das Objekt unterstützt keine benannten Parameter"
447: "Das Objekt unterstützt die aktuelle lokale Einstellung nicht"
448: "Benannten Parameter nicht gefunden"
449: "Parameter nicht optional"
450: "Zahl der Parameter ist falsch"
451: "Das Objekt ist keine Collection"

Diverse Fehler

444: "Methode ist in diesem Kontext nicht anwendbar"
452: "Ungültiger Ordinalwert"
453: "Funktion nicht gefunden"
480: "ByRef-Parameter hat falschen Typ"

Open (Anweisung)

Open *Dateiname* [**For** *Modus*] [**Access** *Zugriffsart*] **As** [#]*Dateinummer*

Öffnet eine Datei für Ein- und/oder Ausgabeoperationen.

Dateiname ist der Name der Datei.

Der optionale Parameter *Modus* kann einen der folgenden Werte annehmen:

Modus	Beschreibung
Input	Sequentielle Eingabe. Die Datei muss bereits existieren. *Zugriffsart* muss, sofern angegeben, auf **Read** gesetzt werden.
Output	Sequentielle Ausgabe. Die Datei wird dazu automatisch angelegt. Sofern eine Datei des angegebenen Namens bereits existiert, wird sie überschrieben. *Zugriffsart* muss, sofern angegeben, auf **Write** gesetzt werden.
Append	Sequentielle Ausgabe. Entspricht **Output**, allerdings wird hier der Dateizeiger an das Ende der Datei gesetzt, sodass alle folgenden Ausgabebefehle Daten an die existierende Datei *anhängen*.

Der optionale Parameter *Zugriffsart* schränkt die Art des Zugriffs auf die Datei ein:

Zugriffsart	Beschreibung
Read	Öffnet die Datei nur zum Lesen.
Write	Öffnet die Datei nur zum Schreiben.
Read Write	Öffnet die Datei zum Lesen und Schreiben.

Wenn die Datei nicht existiert, wird sie automatisch angelegt, sofern als Modus **Append** oder **Output** angegeben wurde; ansonsten schlägt **Open** fehl.

Wenn die Datei bereits von einem anderen Prozess geöffnet ist oder die gewünschte Art des Zugriffs nicht möglich ist, schlägt **Open** fehl.

Dateinummer ist eine ganze Zahl zwischen 1 und 255, mit der in nachfolgenden Zugriffsfunktionen die Datei identifiziert wird. Die Nummer des nächsten freien Dateizeigers kann mit **FreeFile** ermittelt werden.

Siehe auch: Close, **FreeFile**

Beispiel:

```
Sub Main

    Open "TESTFILE" For Output As #1    ' Datei anlegen
    userData1$ = InputBox("Geben Sie eine Zeile Text ein.")
    userData2$ = InputBox("Geben Sie eine weitere Zeile ein.")
    Write #1, userData1, userData2      ' Daten schreiben
    Close #1

    Open "TESTFILE" for Input As #2     ' Datei öffnen
    Print "Inhalt der Datei:"
    Do While Not EOF(2)
        Line Input #2, FileData         ' Zeile einlesen
        Print FileData
    Loop
    Close #2                            ' Datei schließen

    Kill "TESTFILE"                     ' Datei löschen

End Sub
```

Option Base (Anweisung)

Option Base {0|1}

Legt die Untergrenze für Feldindizes fest, wenn diese nicht angegeben werden. Zulässig sind ausschließlich die Werte 0 oder 1.

Wird **Option Base** nicht angegeben, ist die Untergrenze automatisch 0.

Diese Anweisung muss außerhalb jeder Prozedur und vor allen Felddefinitionen stehen.

Siehe auch: Dim, **LBound**, Abschnitt "Arrays"

Beispiel:

```
Option Base 1

Sub Main
    Dim A(20)
    Print "Die Untergrenze des Arrays ist: " & LBound(A) & "."
    Print "Die Obergrenze des Arrays ist: " & UBound(A) & "."
End Sub
```

Option Explicit (Anweisung)

Option Explicit

Bewirkt, dass die Verwendung nicht definierter Variablen als Syntaxfehler gemeldet wird.

Standardmäßig werden Variablen, die zwar benutzt, aber nicht mit **Dim** oder **Static** deklariert wurden, automatisch angelegt (als Variant-Variablen). Dies ist zwar praktisch, führt jedoch dazu, dass Tippfehler bei Variablennamen unbemerkt bleiben.

Nach der Anweisung **Option Explicit** führen unbekannte Variablennamen zu einer Fehlermeldung.

Beispiel:

```
Option Explicit

Sub Main
    Print y    'Fehler, weil y nicht deklariert wurde.
End Sub
```

Print (Anweisung)

Print *Ausdruck* [, ...]

Gibt Daten auf dem Bildschirm aus. Um diese einzusehen, rufen Sie in BasicMaker mit dem Befehl **Ansicht > Ausgabefenster** das Ausgabefenster auf.

Siehe auch: MsgBox, **Print #**

Beispiel:

```
Sub PrintExample
    Dim Pi
    Pi = 4 * Atn(1)      ' Pi berechnen
    Print Pi
End Sub
```

Print # (Anweisung)

Print #*Dateinummer*, [*Ausdruck*]

Gibt Daten in eine Datei aus.

Dateinummer ist die in der **Open**-Anweisung vergebene Nummer der Datei.

Ausdruck enthält die auszugebenden Zeichen.

Wenn *Ausdruck* weggelassen wird, wird eine Leerzeile ausgegeben. Beachten Sie, dass in diesem Fall trotzdem das Komma benötigt wird (z.B. Print #1,).

Formatierung der Ausgabe

Der auszugebende Ausdruck kann optional folgendermaßen formatiert werden:

[[{ **Spc(***n***)**|**Tab(***n***)** }] [*Ausdruck*] [{ **;**|**,** }]]

Spc(*n*) Gibt *n* Leerzeichen vor *Ausdruck* aus

Tab(*n*) Gibt *Ausdruck* in Spalte *n* aus

; Bewirkt, dass das nächste Zeichen unmittelbar anschließt

, Bewirkt, dass das nächste Zeichen am Beginn der nächsten Druckzone ausgegeben wird. Druckzonen beginnen alle 14 Spaltenpositionen.

Wenn weder **;** noch **,** angegeben werden, wird das nächste Zeichen in einer neuen Zeile ausgegeben.

Datums/Zeitwerte werden im kurzen Datums-/Zeitformat ausgegeben.

Der Wert **Empty** (Variant mit VarType 0) erzeugt eine leere Ausgabe.

Der Wert **Null** (Variant mit VarType 1) erzeugt die Ausgabe **#NULL#**.

Siehe auch: Open, **Print**, **Seek**, **Write #**

Beispiel:

Dieses Beispiel schreibt Daten in eine Testdatei und liest sie zurück.

```
Sub Main

    Dim FileData, Msg, NL
    NL = Chr(10)                    ' Chr(10)=Neue Zeile

    Open "TESTFILE" For Output As #1 ' Datei anlegen
    Print #1, "Dies ist ein Test der Print #-Anweisung"
    Print #1, "Zeile 2"
```

```
    Print #1, "Zeile 3"
    Close                           ' Alle Dateien schließen

    Open "TESTFILE" for Input As #2  ' Datei öffnen
    Do While Not EOF(2)
        Line Input #2, FileData      ' Zeile einlesen
        Msg = Msg & FileData & NL
        MsgBox Msg
    Loop
    Close                           ' Alle Dateien schließen

    Kill "TESTFILE"                 ' Datei löschen

End Sub
```

ReDim (Anweisung)

ReDim [**Preserve**] *Varname(Subscripts)* [**As** *Typ*] [, ...]

Mit Hilfe der **ReDim**-Anweisung lässt sich die Größe eines dynamischen Arrays festlegen oder verändern.

Der Inhalt des Arrays wird dabei gelöscht, sofern dem Variablennamen nicht **Preserve** vorangestellt und nur die Größe der letzten Dimension verändert wird.

Varname ist der Name der Array-Variable.

Subscripts gibt Zahl und Größe der Dimensionen an (siehe Abschnitt "Arrays")

Typ ist der Datentyp (siehe Abschnitt "Datentypen").

Dynamische Arrays

Um ein *dynamisches* Array zu erzeugen, muss dieses zunächst mittels der **Global**- oder **Dim**-Anweisung deklariert werden, wobei jedoch anstelle der sonst üblichen Spezifizierung der Anzahl und Größe der Dimensionen ein *leeres* Klammernpaar anzugeben ist.

Beispiel: `Dim A()`

Die Zahl und Größe der Dimensionen kann später mit dem *ersten* Aufruf der **ReDim**-Anweisung festgelegt werden.

Beispiel: `ReDim A(42, 42)`

Mit weiteren Aufrufen der **ReDim**-Anweisung kann die *Größe* der Dimensionen auch weiterhin beliebig oft verändert werden; die *Zahl* der Dimensionen und der *Typ* des Arrays können jedoch, einmal festgelegt, nachträglich nicht mehr geändert werden.

Hinweis: Der Inhalt des Arrays wird bei einer **ReDim**-Anweisung gelöscht.

Verwenden Sie in der Anweisung das Schlüsselwort **Preserve**, lässt sich nur die Größe der letzten Dimension verändern. Hat ein Array beispielsweise zwei Dimensionen, lässt sich lediglich die zweite Dimensionen vergrößern oder verkleinern. Vorteil: Der bisherige Inhalt des Arrays bleibt dabei erhalten.

Beispiel:

```
Dim B()
ReDim B(10, 10)
.
.
ReDim Preserve B(10, 20)
```

Siehe auch: Dim, **Option Base**, **Static**, Abschnitt "Arrays"

Rem (Anweisung)

Rem *Kommentar*

Oder:

'Kommentar

Markiert Kommentare. Kommentare werden bei der Ausführung des Scripts ignoriert.

Siehe auch: Abschnitt "Grundlegendes zur Syntax"

Beispiel:

```
Rem Dies ist ein Kommentar
' Dies auch
```

Resume (Anweisung)

Resume [0]

Oder:

Resume Next

Oder:

Resume *Label*

Beendet eine mittels der **On Error**-Anweisung festgelegte Fehlerbehandlungsroutine und fährt mit der Ausführung fort.

Siehe auch: On Error

Beispiel:

```
Sub Main
    On Error Goto Fehler
    Print 1/0     'Fehler (Division durch Null) auslösen
    MsgBox "Ende"
    Exit Sub

Fehler:           'Fehlerbehandlungsroutine
    Dim DgDef, Msg, Response, Title
    Title = "Fehler"
    Msg = "Es ist ein Laufzeitfehler aufgetreten. Möchten Sie fortfahren?"
    DgDef = MB_YESNO + MB_ICONEXCLAMATION

    Response = MsgBox(Msg, DgDef, Title)
    If Response = IDYES Then
        Resume Next
    Else
        Stop
    End If

End Sub
```

Right (Funktion)

Right(*String*, *n*)

Liefert eine Zeichenkette, die aus den letzten *n* Zeichen der Zeichenkette *String* besteht.

Siehe auch: Len, **Left**, **Mid**

Beispiel:

```
Sub Main
    Dim LWord, Msg, RWord, SpcPos, UsrInp
    Msg = "Geben Sie zwei durch ein Leerzeichen "
    Msg = Msg & "getrennte Wörter ein."
    UsrInp = InputBox(Msg)
    SpcPos = InStr(1, UsrInp, " ")   ' Leerzeichen finden
    If SpcPos Then
        LWord = Left(UsrInp, SpcPos - 1)    ' Linkes Wort
        RWord = Right(UsrInp, Len(UsrInp) - SpcPos)   ' Rechtes Wort
        Msg = "Das erste Wort ist " & LWord & ","
        Msg = Msg & " das zweite ist " & RWord & "."
    Else
        Msg = "Das waren nicht 2 Wörter."
    End If
    MsgBox Msg
End Sub
```

RmDir (Anweisung)

RmDir *Pfad*

Entfernt das angegebene Verzeichnis.

Als Parameter wird der Pfad in der Notation *Laufwerksbuchstabe*:*Verzeichnis* erwartet.

Siehe auch: ChDir, **ChDrive**, **CurDir**, **Kill**

Beispiel:

```
Sub Main
   Dim dirName As String
   dirName = "t1"
   MkDir dirName
   MkDir "t2"
   MsgBox "Verzeichnisse t1 und t2 erzeugt. Klicken Sie auf OK, um sie wieder zu
entfernen."
   RmDir "t1"
   RmDir "t2"
End Sub
```

Rnd (Funktion)

Rnd [()]

Erzeugt eine Zufallszahl zwischen 0 und 1.

Second (Funktion)

Second(*Ausdruck*)

Liefert die Sekunde der angegebenen Uhrzeit als ganze Zahl.

Ausdruck ist ein numerischer oder Zeichenkettenausdruck, der eine Zeit repräsentiert.

Siehe auch: Date, **Day**, **Hour**, **Minute**, **Month**, **Now**, **Time**, **Weekday**, **Year**

Beispiel:

```
Sub Main

    T1 = Now     ' Now = aktuelles Datum + Uhrzeit
```

```
    MsgBox T1

    MsgBox "Tag: " & Day(T1)
    MsgBox "Monat: " & Month(T1)
    MsgBox "Jahr: " & Year(T1)

    MsgBox "Stunde: " & Hour(T1)
    MsgBox "Minute: " & Minute(T1)
    MsgBox "Sekunde: " & Second(T1)

End Sub
```

Seek (Anweisung)

Seek [#]*Dateinummer*, *Position*

Setzt die Position des Dateizeigers in einer Datei. Diese Datei muss zuvor geöffnet worden sein.

Dateinummer ist die in der **Open**-Anweisung vergebene Nummer der Datei.

Position ist die Position innerhalb der Datei, an der die nächste Schreib- oder Leseoperation stattfinden soll (als Byte-Offset ab dem Dateianfang).

Siehe auch: Open

Beispiel:

```
Sub Main
    Open "TESTFILE" For Input As #1   ' Datei öffnen
    For i = 0 To 24 Step 3

        Seek #1, i                    ' Dateizeiger setzen
        MyChar = Input(1, #1)         ' Zeichen einlesen
        Print MyChar                  ' Zeichen ausgeben
    Next i
    Close #1                          ' Datei schließen
End Sub
```

Select Case (Anweisung)

Select Case *Ausdruck*

   [**Case** *Wert1*
      [*Anweisungen*]]

   [**Case** *Wert2*
      [*Anweisungen*]]
   .
   .
   .

   [**Case Else**
      [*Anweisungen*]]

End Select

Führt abhängig vom Wert des übergebenen Ausdrucks einen von mehreren Anweisungsblöcken aus (siehe auch Abschnitt "Kontrollstrukturen").

Eine **Select Case**-Struktur muss stets mit **End Select** abgeschlossen werden.

Siehe auch: If ... Then ... Else, Abschnitt "Kontrollstrukturen"

Beispiel:

```
Sub Main
```

```
   Zahl = InputBox("Geben Sie eine ganze Zahl zwischen 1 und 3 ein:")

   Select Case Val(Zahl)

      Case 1
      MsgBox "Sie haben eine Eins eingegeben."

      Case 2
      MsgBox "Sie haben eine Zwei eingegeben."

      Case 3
      MsgBox "Sie haben eine Drei eingegeben."

      Case Else
      MsgBox "Es sind nur ganze Zahlen zwischen 1 und 3 erlaubt!"

   End Select

End Sub
```

SendKeys (Anweisung)

SendKeys(*Tasten*, [*Warten*])

Simuliert das Drücken von Tasten.

Tasten ist eine Zeichenkette, die die zu betätigenden Tasten enthält.

Wenn der optionale Parameter *Warten* **True** ist, kehrt die Kontrolle erst wieder zurück, wenn die Verarbeitung der Tastendrücke abgeschlossen ist.

Um "gewöhnliche" Tasten zu übergeben, sind diese einfach einzutippen – zum Beispiel "Test". Spezialtasten wie die Enter- oder die Alt-Taste erreichen Sie folgendermaßen:

- Die Tasten + ^ ~ % ( ) [ ] { und } sind Spezialzeichen, die deshalb von geschweiften Klammern umgeben werden müssen – zum Beispiel: "{%}" oder {(}.
- Auch Sondertasten wie die Enter-Taste müssen von geschweiften Klammern umgeben werden – zum Beispiel {Enter}. Eine Liste finden Sie im nächsten Abschnitt "Tabelle der von Sendkeys unterstützten Sondertasten".
- Tastenkombinationen mit der Umschalt-, Alt- und Strg-Taste können über das Voranstellen von +, ^ beziehungsweise % erreicht werden:

  Umschalt+Enter: "+{Enter]"

  Alt+F4: "%{F4}"

  Strg+C: "^c" (*nicht* ^C!)

  Achten Sie hierbei auf die Groß-/Kleinschreibung: "^c" entspricht beispielsweise der Tastenkombination Strg+C, "^C" entspricht hingegen Strg+Umschalt+C.
- Sollen Anführungszeichen übergeben werden, sind diese doppelt einzugeben – zum Beispiel "Jörg ""Turbo"" Wontorra"
- Sequenzen einer Taste können durch Angabe der Taste und der Anzahl an Wiederholungen in geschweiften Klammern übergeben werden: "{a 10}" wiederholt 10x die Taste a, {Enter 2} 2x die Enter-Taste.
- Die Enter-Taste kann auch über das Kürzel ~ erzeugt werden. "ab~cd" ist gleichbedeutend mit "ab{Enter}cd"

Beispiel:

```
Sub Main
    X = Shell("Calc.exe", 1)        ' Rechner aufrufen
    For I = 1 To 5
        SendKeys I & "{+}", True    ' Tastendrücke senden
    Next I
    Msg = "Der Rechner wird jetzt wieder beendet."
    MsgBox Msg
```

```
    AppActivate "Rechner"           ' Fokus auf Rechner setzen
    SendKeys "%{F4}", True          ' Alt+F4 zum Beenden senden
End Sub
```

Tabelle der von Sendkeys unterstützten Sondertasten

Folgende Sondertasten können mit der **Sendkeys**-Anweisung verwendet werden:

Taste	Zu übergebende Zeichenkette
Escape	{Escape} oder {Esc}
Enter (Eingabetaste)	{Enter}
Umschalttaste	+ voranstellen (zum Beispiel +{F9} für Umschalt+F9)
Alt-Taste	% voranstellen (zum Beispiel %{F9} für Alt+F9)
Strg-Taste	^ voranstellen (zum Beispiel ^{F9} für Strg+F9)
Tabulator	{Tab}
Nach links	{Left}
Nach rechts	{Right}
Nach unten	{Down}
Nach oben	{Up}
Pos1	{Home}
Ende	{End}
Bild abwärts	{PageDn}
Bild aufwärts	{PageUp}
Rücktaste	{Backspace} oder {BS}
Entfernen	{Delete} oder {Del}
Einfügen	{Insert}
Druck	{PrtSc}
Strg-Untbr	{Break}
CapsLock (Feststelltaste)	{CapsLock}
NumLock	{NumLock}
0 auf 10er-Tastatur	{NumPad0}
.	
.	
9 auf 10er-Tastatur	{NumPad9}
/ auf 10er-Tastatur	{NumPad/}
* auf 10er-Tastatur	{NumPad*}
- auf 10er-Tastatur	{NumPad-}
+ auf 10er-Tastatur	{NumPad+}
. auf 10er-Tastatur	{NumPad.}
F1	{F1}
.	
.	

F12 {F12}

Set (Anweisung)

Set *Object* = [**New**] *ObjectExpression*

Beziehungsweise:

Set *Object* = **Nothing**

Die obere Schreibweise verknüpft eine Objektvariable mit einem OLE-Objekt; die untere löst die Verknüpfung auf.

Siehe auch: Dim, **Static**, Abschnitt "OLE Automation"

Sgn (Funktion)

Sgn(*Num*)

Ermittelt das Vorzeichen einer Zahl.

Mögliche Rückgabewerte:

- -1, wenn die Zahl < 0 ist
- 0, wenn die Zahl = 0 ist
- 1, wenn die Zahl > 0 ist

Siehe auch: Abs

Shell (Funktion)

Shell(*Appname [, Modus]*)

Startet ein Programm.

Der Rückgabewert ist eine Task-ID, die das gestartete Programm identifiziert. Werte kleiner als 32 zeigen an, dass der Programmstart fehlgeschlagen ist.

Appname ist der Dateiname der ausführbaren Datei. Der Name muss eine der Erweiterungen .PIF, .COM, .BAT oder .EXE besitzen.

Der optionale Parameter *Modus* gibt an, wie das Fenster des Programms geöffnet werden soll:

Wert	Bedeutung
1	Normal mit Fokus (Standardwert)
2	Minimiert mit Fokus
3	Maximiert mit Fokus
4	Normal ohne Fokus
6	Minimiert ohne Fokus

Siehe auch: AppDataMaker, **AppPlanMaker**, **AppTextMaker**, **CreateObject**, **GetObject**

Beispiel:

```
Sub Main
    X = Shell("Calc.exe", 1)        ' Rechner aufrufen
    If X < 32 Then
        MsgBox "Rechner konnte nicht gestartet werden"
        Stop
    End If

    For I = 1 To 5
        SendKeys I & "{+}", True    ' Tastendrücke senden
    Next I

    Msg = "Der Rechner wird jetzt wieder beendet."
    MsgBox Msg
    AppActivate "Rechner"           ' Fokus auf Rechner setzen
    SendKeys "%{F4}", True          ' Alt+F4 zum Beenden senden
End Sub
```

Sin (Funktion)

Sin(*Num*)

Ermittelt den Sinus eines Winkels.

Der Winkel muss im Bogenmaß (Radiant) angegeben werden.

Siehe auch: Atn, **Cos**, **Tan**

Beispiel:

```
Sub Main
    pi = 4 * Atn(1)
    rad = 90 * (pi/180)
    x = Sin(rad)
    Print x
End Sub
```

Space (Funktion)

Space(*n*)

Erzeugt eine Zeichenkette, die aus *n* Leerzeichen besteht.

Für *n* sind Werte zwischen 0 und 32767 zulässig.

Siehe auch: String

Beispiel:

```
Sub Main
    MsgBox "Ab..." & Space(20) & "...stand"
End Sub
```

Sqr (Funktion)

Sqr(*Num*)

Berechnet die Quadratwurzel einer Zahl.

Num darf nicht kleiner als 0 sein.

```
Sub Wurzel
    Dim Titel, Msg, Zahl
```

```
    Titel = "Berechnung der Quadratwurzel"
    Prompt = "Geben Sie eine positive Zahl ein:"
    Zahl = InputBox(Prompt, Titel)
    If Zahl < 0 Then
        Msg = "Die Wurzel von negativen Zahlen ist nicht definiert."
    Else
        Msg = "Die Wurzel von " & Zahl & " beträgt "
        Msg = Msg & Sqr(Zahl) & "."
    End If
    MsgBox Msg
End Sub
```

Static (Anweisung)

Static *Variable*

Legt Speicherplatz für eine Variable an und bestimmt deren Datentyp.

Im Unterschied zu Variablen, die mit **Dim** angelegt werden, behalten **Static**-Variablen ihren Wert während des gesamten Programmlaufs, auch wenn sie innerhalb einer Funktion deklariert worden sind.

Siehe auch: Dim, **Function**, **Sub**

Beispiel:

```
' Dieses Beispiel zeigt den Gebrauch von Static, um den Wert der
' Variablen i in der Prozedur Joe zu erhalten.

Sub Main
    For i = 1 to 2
        Joe 2
    Next i
End Sub

Sub Joe(j As Integer)
    Static i
    Print i
    i = i + 5
    Print i
End Sub
```

Stop (Anweisung)

Stop

Beendet die Ausführung des Scripts sofort.

Siehe auch: End

Beispiel:

```
Sub Main
    Dim x, y, z
    For x = 1 to 3
        For y = 1 to 3
            For z = 1 to 3
                Print z, y, x
            Next z
        Next y
        Stop
    Next x
End Sub
```

Str (Funktion)

Str(*Num*)

Wandelt einen numerischen Ausdruck in eine Zeichenkette um.

Wird eine Zahl übergeben, die positiv ist, beginnt die resultierende Zeichenkette mit einem Leerzeichen. Bei negativen Zahlen erscheint an dieser Stelle das Minuszeichen.

Siehe auch: CStr, **Format**, **Val**

Beispiel:

```
Sub Main
    Dim msg
    a = -1
    MsgBox "Zahl = " & Str(a)
    MsgBox "Abs(Zahl) =" & Str(Abs(a))
End Sub
```

StrComp (Funktion)

StrComp(*String1, String2* [, *IgnoreCase*])

Vergleicht zwei Zeichenketten.

Wenn Sie den Parameter *IgnoreCase* auf **True** setzen, wird die Groß-/Kleinschreibung ignoriert. Ist er **False** oder wird er weggelassen, wird die Groß-/Kleinschreibung beachtet.

Die Funktion liefert folgendes Ergebnis:

- 0, wenn die Zeichenketten gleich sind
- -1, wenn String1 < String2
- 1, wenn String1 > String2

String (Funktion)

String(*Num, Zeichen*)

Erzeugt eine Zeichenkette, die aus n Wiederholungen eines bestimmten Zeichens besteht.

Num ist die gewünschte Anzahl an Wiederholungen.

Zeichen ist das zu wiederholende Zeichen.

Siehe auch: Space

Beispiel:

```
Print String(80, "-")
```

Sub (Anweisung)

```
Sub Name [(Argumentliste)]
   Dim [Variable(n)]
   [Anweisungen]
   [Exit Sub]
End Sub
```

Leitet die Definition eines Unterprogramms ein.

Name ist der Name des Unterprogramms.

Argumentliste ist eine durch Kommata getrennte Liste von Parameterdeklarationen (siehe unten).

Mit **End Sub** wird das Ende der Definition gekennzeichnet.

Exit Sub kann verwendet werden, um die Abarbeitung des Unterprogramms vorzeitig abzubrechen.

Parameterdeklaration

[**ByVal** | **ByRef**] *Variable* [**As** *Typ*]

Mit **ByVal** beziehungsweise **ByRef** (Standardwert) wird bestimmt, ob der Parameter als Wert oder als Referenz übergeben wird (siehe Abschnitt "Parameterübergabe ByRef oder ByVal").

Typ spezifiziert den Datentyp (**String**, **Integer**, **Double**, **Long**, **Variant**). Alternativ kann der Typ auch durch einen Typsuffix (z.B. % für **Integer**) angegeben werden (siehe Abschnitt "Datentypen").

Siehe auch: Call, **Dim**, **Function**

Beispiel:

```
Sub Main
    Dim Var1 as String
    Var1 = "Hallo"
    MsgBox "Test"
    Test Var1
    MsgBox Var1
End Sub

Sub Test(wvar1 as String)
    wvar1 = "Tschüß"
End Sub
```

Tan (Funktion)

Tan(*Num*)

Ermittelt den Tangens eines Winkels.

Der Winkel muss im Bogenmaß (Radiant) angegeben werden.

Siehe auch: Atn, Cos, Sin

Beispiel:

```
Sub Main
    Dim Msg, Pi
    Pi = 4 * Atn(1)      ' Pi ermitteln
    x = Tan(Pi/4)
    MsgBox "Tan(Pi/4)=" & x
End Sub
```

Time (Funktion)

Time [()]

Liefert die aktuelle Systemuhrzeit im Format HH:MM:SS.

Das Trennzeichen (normalerweise ein Doppelpunkt) kann über das Symbol zum Ändern der Regions- und Sprachoptionen in der Systemsteuerung verändert werden.

Siehe auch: Date, **DateSerial**, **DateValue**, **Hour**, **Minute**, **Now**, **Second**, **TimeSerial**, **TimeValue**

Beispiel:

```
Sub Main

    T1 = Time

    MsgBox T1

    MsgBox "Stunde: " & Hour(T1)
    MsgBox "Minute: " & Minute(T1)
    MsgBox "Sekunde: " & Second(T1)

End Sub
```

TimeSerial (Funktion)

TimeSerial(*Stunde*, *Minute*, *Sekunde*)

Liefert eine serielle Zahl mit der Uhrzeit, die durch die Parameter *Stunde*, *Minute* und *Sekunde* festgelegt wird.

Siehe auch: DateSerial, **DateValue**, **Hour**, **Minute**, **Now**, **Second**, **Time**, **TimeValue**

Beispiel:

```
Sub Main
   Print TimeSerial(09,30,59)
End Sub
```

TimeValue (Funktion)

TimeValue(*ZeitAusdruck*)

Liefert eine serielle Zahl vom Typ Double mit der Uhrzeit, die durch den übergebenen Parameter *ZeitAusdruck* festgelegt wird. Dieser Parameter kann eine beliebige Zeichenkette sein, die eine Uhrzeit repräsentiert.

Siehe auch: DateSerial, **DateValue**, **Hour**, **Minute**, **Now**, **Second**, **Time**, **TimeSerial**

Beispiel:

```
Sub Main
   Print TimeValue("09:30:59")
End Sub
```

Trim, LTrim, RTrim (Funktion)

Entfernt Leerzeichen am Beginn oder Ende einer Zeichenkette.

LTrim(*String*) entfernt führende Leerzeichen.

RTrim(*String*) entfernt abschließende Leerzeichen.

Trim(*String*) entfernt führende und abschließende Leerzeichen.

Beispiel:

```
Sub Main

    MyString = "   <-Trim->   "

    TrimString = LTrim(MyString)          ' "<-Trim->   ".
```

```
    MsgBox "|" & TrimString & "|"

    TrimString = RTrim(MyString))         ' "   <-Trim->".
    MsgBox "|" & TrimString & "|"

    TrimString = LTrim(RTrim(MyString))   ' "<-Trim->".
    MsgBox "|" & TrimString & "|"

    TrimString = Trim(MyString)           ' "<-Trim->".
    MsgBox "|" & TrimString & "|"

End Sub
```

Type (Anweisung)

```
Type Typname
   Element As Typ
   Element As Typ
   Element As Typ
   .
   .
   .
End Type
```

Deklariert einen benutzerdefinierten Typ.

Typname ist der Name des neuen Typs.

Element ist der Name eines Elements dieses Typs.

Typ ist der Datentyp dieses Elements (**Integer**, **Long**, **Single**, **Double**, **String**, **String*n***, **Variant** oder ein benutzerdefinierter Typ).

Nach der Definition können Variablen des neuen Typs mit **Dim** x **As** *Typname* oder **Static** x **As** *Typname* angelegt werden.

Um auf ein Element zuzugreifen, benutzt man die Punktnotation: *Variable.Element*.

Type-Anweisungen dürfen nicht innerhalb von **Sub**- oder **Function**-Anweisungen verwendet werden.

Siehe auch: Dim, **Static**, **With**, Abschnitt "Datentypen"

Beispiel:

```
Type type1
    a As Integer
    d As Double
    s As String
End Type

Type type2
    a As String
    o As type1
End Type

Type type3
    b As Integer
    c As type2
End Type

Dim var2a As type2
Dim var2b As type2
Dim var1a As type1
Dim var3a as type3

Sub Test
    a = 5
    var1a.a = 7472
    var1a.d = 23.1415
    var1a.s = "TEST"
```

```
    var2a.a = "43 - dreiundvierzig"
    var2a.o.s = "Hi"
    var3a.c.o.s = "COS"
    var2b.a = "943 - neunhundertdreiundvierzig"
    var2b.o.s = "Yogi"
    MsgBox var1a.a
    MsgBox var1a.d
    MsgBox var1a.s
    MsgBox var2a.a
    MsgBox var2a.o.s
    MsgBox var2b.a
    MsgBox var2b.o.s
    MsgBox var3a.c.o.s
    MsgBox a
End Sub
```

UBound (Funktion)

UBound(*Arrayname*[, *Dimension*])

Ermittelt den größten Index der angegebenen Dimension eines Arrays.

Wird *Dimension* nicht angegeben, wird die erste Dimension des Feldes verwendet.

Siehe auch: Dim, **LBound**, **ReDim**

Beispiel:

```
Option Base 1

Sub Main
    Dim a(10, 20 To 40)
    Print "1. Dimension: " & LBound(a) & " bis " & UBound(a)
    Print "2. Dimension: " & LBound(a, 2) & " bis " & UBound(a, 2)
End Sub
```

UCase (Funktion)

UCase(*String*)

Wandelt eine Zeichenkette in Großbuchstaben um.

Siehe auch: LCase

Beispiel:

```
Sub Main
   MsgBox UCase("Think BIG!")    ' ergibt "THINK BIG!"
End Sub
```

Val (Funktion)

Val(*String*)

Wandelt eine Zeichenkette in eine Zahl um.

Der Inhalt der Zeichenkette wird nur bis zum ersten nicht-numerischen Zeichen berücksichtigt. Leerzeichen, Tabulator und Zeilenvorschub (Linefeed) werden ignoriert.

Beginnt die Zeichenkette nicht mit einer Zahl, ist das Ergebnis 0.

Val ("2") ergibt 2

Val ("2 Uhr") ergibt 2

Val ("2 Uhr 30") ergibt 2

Val ("xyz 2") ergibt 0

Siehe auch: Str

Beispiel:

```
Sub main
    Dim Msg
    Dim YourVal As Double
    YourVal = Val(InputBox$("Geben Sie eine Zahl ein."))
    Msg = "Die eingegebene Zahl war " & YourVal
    MsgBox Msg
End Sub
```

VarType (Funktion)

VarType(*Varname*)

Ermittelt den Datentyp einer Variant-Variable.

Die möglichen Rückgabewerte sind:

Typ	Rückgabewert
Empty	0
Null	1
Integer	2
Long	3
Single	4
Double	5
String	8

Siehe auch: IsDate, **IsEmpty**, **IsNull**, **IsNumeric**, Abschnitt "Besonderheiten beim Variant-Datentyp"

Beispiel:

```
If VarType(x) = 5 Then Print "Variable ist vom Typ Double"
```

Weekday (Funktion)

Weekday(*Ausdruck*)

Ermittelt den Wochentag des angegebenen Datums.

Das Ergebnis ist eine ganze Zahl zwischen 1 und 7, wobei 1=Sonntag, 2=Montag, ... 7=Samstag.

Ausdruck ist ein numerischer oder Zeichenkettenausdruck, der ein Datum repräsentiert.

Siehe auch: Date, **Day**, **Hour**, **Minute**, **Month**, **Now**, **Second**, **Time**, **Year**

Beispiel:

```
Sub Main
    Print Weekday(Date)
```

```
End Sub
```

While ... Wend (Anweisung)

```
While Bedingung
   [Anweisungen]
Wend
```

Wiederholt die Ausführung einer Gruppe von Anweisungen so lange, wie die angegebene Bedingung wahr ist (siehe auch Abschnitt "Kontrollstrukturen").

Siehe auch: Do ... Loop, Abschnitt "Kontrollstrukturen"

With (Anweisung)

```
With Object
   [Anweisungen]
End With
```

Führt eine Gruppe von Anweisungen für ein bestimmtes Objekt aus.

Die **With**-Anweisung erlaubt es, auf die Elemente eines Objekts zuzugreifen, ohne den Objektnamen jedes Mal angeben zu müssen.

With-Anweisungen können verschachtelt werden.

Siehe auch: While ... Wend, **Do ... Loop**, Abschnitt "Tipps für die Vereinfachung von Schreibweisen"

Beispiel:

```
Type type1
    a As Integer
    d As Double
    s As String
End Type

Type type2
    a As String
    o As type1
End Type

Dim var1a As type1
Dim var2a As type2

Sub Main

    With var1a
      .a = 65
      .d = 3.14
    End With
    With var2a
      .a = "Hallo"
      With .o
        .s = "Tschüß"
      End With
    End With
    var1a.s = "TEST"
    MsgBox var1a.a
    MsgBox var1a.d
    MsgBox var1a.s
    MsgBox var2a.a
    MsgBox var2a.o.s

End Sub
```

Write # (Anweisung)

Write #*Dateinummer*, [*Ausdruck*]

Schreibt Daten in eine Datei.

Die Datei muss vorher mittels einer **Open**-Anweisung im **Output**- oder **Append**-Modus geöffnet worden sein.

Dateinummer ist die in der **Open**-Anweisung vergebene Nummer für die Datei.

Ausdruck enthält ein oder mehrere Ausgabeelemente.

Wenn *Ausdruck* weggelassen wird, wird eine Leerzeile ausgegeben. Beachten Sie, dass in diesem Fall trotzdem ein Komma geschrieben werden muss.

Siehe auch: Open, **Seek**, **Print #**

Beispiel:

```
Sub Main

    Open "TESTFILE" For Output As #1    ' Datei anlegen
    userData1$ = InputBox("Geben Sie eine Zeile Text ein.")
    userData2$ = InputBox("Geben Sie eine weitere Zeile ein.")
    Write #1, userData1, userData2      ' Daten schreiben
    Close #1

    Open "TESTFILE" for Input As #2     ' Datei öffnen
    Print "Inhalt der Datei:"
    Do While Not EOF(2)
        Line Input #2, FileData         ' Zeile einlesen
        Print FileData
    Loop
    Close #2                            ' Datei schließen

    Kill "TESTFILE"                     ' Datei löschen

End Sub
```

Year (Funktion)

Year(*Ausdruck*)

Liefert das Jahr des angegebenen Datums.

Ausdruck ist ein numerischer oder Zeichenkettenausdruck, der ein Datum repräsentiert.

Das Ergebnis ist eine ganze Zahl zwischen 100 und 9999.

Siehe auch: Date, **Day**, **Hour**, **Minute**, **Month**, **Now**, **Second**, **Time**, **Weekday**

Beispiel:

```
Sub Main

    T1 = Now      ' Now = aktuelles Datum + Uhrzeit

    MsgBox T1

    MsgBox "Tag: " & Day(T1)
    MsgBox "Monat: " & Month(T1)
    MsgBox "Jahr: " & Year(T1)

    MsgBox "Stunde: " & Hour(T1)
    MsgBox "Minute: " & Minute(T1)
    MsgBox "Sekunde: " & Second(T1)

End Sub
```

Anhang

Im Anhang finden Sie Informationen zu folgenden Themen:

- **Farbkonstanten**

  Dieser Abschnitt enthält eine Liste aller vordefinierten Farbkonstanten.

Farbkonstanten

Es gibt eine Reihe von Eigenschaften in TextMaker und PlanMaker, mit denen man Farben setzen oder abfragen kann. Alle diese Eigenschaften sind in zwei verschiedenen Versionen verfügbar: einmal zum Arbeiten mit *BGR-Farbwerten* ("Blau-Grün-Rot") und einmal mit *Indexfarben* – bei letzteren sind die Standardfarben einfach durchnummeriert.

Zum Beispiel setzt **Selection.Font.Color** in TextMaker die Farbe des selektierten Textes auf den übergebenen BGR-Wert. Die Methode **Selection.Font.ColorIndex** hingegen erwartet eine Indexfarbe.

Auf den nächsten Seiten finden Sie eine Liste aller Farbkonstanten, die Sie in solchen Anweisungen verwenden können. Sie ist aufgeteilt auf zwei Bereiche:

- **Farbkonstanten für BGR-Farben**
- **Farbkonstanten für Indexfarben**

Farbkonstanten für BGR-Farben

Einige Eigenschaften in TextMaker und PlanMaker erwarten als Argument eine BGR-Farbe (blau/grün/rot). Sie können entweder einen beliebigen BGR-Wert angeben oder eine Konstante aus der folgenden Tabelle auswählen:

```
smoColorAutomatic            = -1 ' Automatisch (siehe unten)
smoColorTransparent          = -1 ' Transparent (siehe unten)
smoColorBlack                = &h0&
smoColorBlue                 = &hFF0000&
smoColorBrightGreen          = &h00FF00&
smoColorRed                  = &h0000FF&
smoColorYellow               = &h00FFFF&
smoColorTurquoise            = &hFFFF00&
smoColorViolet               = &h800080&
smoColorWhite                = &hFFFFFF&
smoColorIndigo               = &h993333&
smoColorOliveGreen           = &h003333&
smoColorPaleBlue             = &hFFCC99&
smoColorPlum                 = &h663399&
smoColorRose                 = &hCC99FF&
smoColorSeaGreen             = &h669933&
smoColorSkyBlue              = &hFFCC00&
smoColorTan                  = &h99CCFF&
smoColorTeal                 = &h808000&
smoColorAqua                 = &hCCCC33&
smoColorBlueGray             = &h996666&
smoColorBrown                = &h003399&
smoColorGold                 = &h00CCFF&
smoColorGreen                = &h008000&
smoColorLavender             = &hFF99CC&
smoColorLime                 = &h00CC99&
smoColorOrange               = &h0066FF&
smoColorPink                 = &hFF00FF&
smoColorLightBlue            = &hFF6633&
smoColorLightOrange          = &h0099FF&
smoColorLightYellow          = &h99FFFF&
smoColorLightGreen           = &hCCFFCC&
smoColorLightTurquoise       = &hFFFFCC&
```

```
smoColorDarkBlue              = &h800000&
smoColorDarkGreen             = &h003300&
smoColorDarkRed               = &h000080&
smoColorDarkTeal              = &h663300&
smoColorDarkYellow            = &h008080&
smoColorGray05                = &hF3F3F3&
smoColorGray10                = &hE6E6E6&
smoColorGray125               = &hE0E0E0&
smoColorGray15                = &hD9D9D9&
smoColorGray20                = &hCCCCCC&
smoColorGray25                = &hC0C0C0&
smoColorGray30                = &hB3B3B3&
smoColorGray35                = &hA6A6A6&
smoColorGray375               = &hA0A0A0&
smoColorGray40                = &h999999&
smoColorGray45                = &h8C8C8C&
smoColorGray50                = &h808080&
smoColorGray55                = &h737373&
smoColorGray60                = &h666666&
smoColorGray625               = &h606060&
smoColorGray65                = &h595959&
smoColorGray75                = &h404040&
smoColorGray85                = &h262626&
smoColorGray90                = &h191919&
smoColorGray70                = &h4C4C4C&
smoColorGray80                = &h333333&
smoColorGray875               = &h202020&
smoColorGray95                = &hC0C0C0&
```

Die Farben **smoColorAutomatic** und **smoColorTransparent** decken Spezialfälle ab und können *nicht* generell verwendet werden:

- Mit **smoColorAutomatic** können Sie in PlanMaker die Farbe des Arbeitsblattsgitters auf "Automatisch" setzen.
- Mit **smoColorTransparent** können Sie in TextMaker und PlanMaker die Hintergrundfarbe des Textes auf "Transparent" setzen.

Farbkonstanten für Indexfarben

Einige Eigenschaften in TextMaker und PlanMaker erwarten als Argument einen Farbindex. Sie dürfen dann **ausschließlich** einen der folgenden Werte angeben:

```
smoColorIndexAuto             = -1 ' Automatisch (siehe unten)
smoColorIndexTransparent      = -1 ' Transparent (siehe unten)
smoColorIndexBlack            =  0  ' Schwarz
smoColorIndexBlue             =  1  ' Blau
smoColorIndexCyan             =  2  ' Zyanblau
smoColorIndexGreen            =  3  ' Grün
smoColorIndexMagenta          =  4  ' Magenta
smoColorIndexRed              =  5  ' Rot
smoColorIndexYellow           =  6  ' Gelb
smoColorIndexWhite            =  7  ' Weiß
smoColorIndexDarkBlue         =  8  ' Dunkelblau
smoColorIndexDarkCyan         =  9  ' Dunkles Zyanblau
smoColorIndexDarkGreen        = 10 ' Dunkelgrün
smoColorIndexDarkMagenta      = 11 ' Dunkles Magenta
smoColorIndexDarkRed          = 12 ' Dunkelrot
smoColorIndexBrown            = 13 ' Braun
smoColorIndexDarkGray         = 14 ' Dunkelgrau
smoColorIndexLightGray        = 15 ' Hellgrau
```

Tipp: Die Eigenschaften mit BGR-Farben sind flexibler und sollten daher vorzugsweise benutzt werden.

Die Farben **smoColorIndexAuto** und **smoColorIndexTransparent** decken Spezialfälle ab und können *nicht* generell verwendet werden:

- Mit **smoColorIndexAuto** können Sie in PlanMaker die Farbe des Arbeitsblattsgitters auf "Automatisch" setzen.

- Mit **smoColorIndexTransparent** können Sie in TextMaker und PlanMaker die Hintergrundfarbe des Textes auf "Transparent" setzen.