6 Makro-Programmierbefehle

In diesem Kapitel werden die verfügbaren Makro-Programmierbefehle im einzelnen beschrieben. WordPerfect für Windows verwendet für die Makro-Programmierbefehle einen eigenen Formalismus, der von dem früherer WordPerfect-Versionen (bis DOS 5.1) und von dem der Mischbefehle abweicht. Die Bedeutung ist jedoch weitgehend geblieben. Eine Zusammenstellung (Referenzliste) aller Befehle finden Sie in Kapitel »14 Makro- und Mischbefehle«. Kopieren Sie sich diese Liste für Ihre tägliche Arbeit. Sie brauchen dann nicht ständig im Handbuch nachzulesen, wenn Ihnen die Syntax eines Befehls gerade nicht geläufig ist. Dieses Kapitel ist zusätzlich auf der Diskette vorhanden, so daß Sie die Referenzliste auch drucken können (beachten Sie hierzu bitte die Hinweise in Kapitel »14 Makro- und Mischbefehle«). Bevor Sie mit den hier beschriebenen Befehlen arbeiten, sollte Ihnen bekannt sein, wie man Makros erstellt. Informieren Sie sich bitte in den vorausgegangenen Kapiteln.

ALT-Makros sind Makros, die in früheren WordPerfect-Versionen durch Drücken von Alt in Verbindung mit einer Buchstabentaste (mit Ausnahme der Umlaute und ß) definiert bzw. abgerufen wurden. Der Vorteil bestand darin, das Makro durch einen einfachen Tastendruck zur Ausführung aufzurufen. Das Drücken von [*Makro*] und die Eingabe eines Makronamens ist darum entfallen.

In dieser WordPerfect-Version ist die Verwendung von ALT-Makros nicht mehr erlaubt, da Alt in Verbindung mit einem Buchstaben unter Windows für die Menüauswahl verwendet wird. Makros können Sie mit dieser Version u. a. auch über Strg oder Strg-⇧ in Verbindung mit einer Buchstabentaste abrufen. Zuvor müssen Sie diese Makros in den Standardeinstellungen (Tastatur) einer Tastenkombination zuordnen.

Zum schnelleren Auffinden sind die nachfolgenden Makro-Programmierbefehle in alphabetischer Reihenfolge geordnet. Die Bedeutung der Makro-Programmierbefehle wird durch kleine Beispiele veranschaulicht. Es können hier natürlich nicht alle Befehle in einem Makro erklärt werden, um das Zusammenspiel der Befehle zu demonstrieren. Dieses Makro wäre einerseits zu umfangreich bzw. zu unübersichtlich, und andererseits wären hierdurch keine großen Lerneffekte zu erzielen. Größere Makros finden Sie in Kapitel »9 Fallstudien«.

Bei der Anwendung der Makro-Programmierbefehle sind zum besseren Verständnis Vorkenntnisse einer höheren Programmiersprache von Vorteil. Dies bezieht sich insbesondere auf die Befehle *Call*, *Case*, *Case Call*, *Else*, *For*, *ForEach*, *ForNext*, *If*, *Go*, *Repeat* und *While*, die in Verbindung mit Schleifenverarbeitung und Unterprogramm-Technik benutzt werden, und die unmittelbar damit verbundenen Befehle.

Wie Ihnen sicher bekannt ist, können Sie auch in Mischdateien sog. Mischbefehle integrieren, um umfangreiche Mischvorgänge zu vereinfachen bzw. komfortabler zu gestalten. Die meisten dieser Mischbefehle sind in der Handhabung und Wirkungsweise mit den Makro-Programmierbefehlen identisch. Die nachfolgende Beschreibung der Makro-Programmierbefehle können Sie darum weitestgehend sinngemäß bei den Mischbefehlen anwenden.

Mischbefehle Mischbefehle werden über die Funktion [*Tools, Mischen*] in eine Mischdatei (Daten- und/oder Formulardatei) eingefügt. Sind Variable erforderlich, müssen diese unmittelbar dem Befehl folgen. Mischbefehle können im Gegensatz zu Programmierbefehlen nicht zeichenweise über die Tastatur eingegeben werden. Eine Beschreibung und die Handhabung der zusätzlichen Mischbefehle finden Sie in Kapitel »7 Mischbefehle«.

Programmierbefehle Diese Befehle werden über den Makrobefehlsmanager oder manuell (Eingabe des Befehls über die Tastatur) in ein Makro eingefügt. Diese Befehle können auch zeichenweise über die Tastatur eingegeben werden.

Bei allen Befehlen, bei denen ein Boxentitel angeben werden kann (z. B. *Prompt*, *GetString* usw.), müssen Sie folgendes beachten:

- Wenn Sie keinen Titel eingeben und anstelle des Titels"" (keine Leerstelle) definieren, wird als Boxentitel »PerfectScript« angezeigt.
- Wenn Sie keinen Titel eingeben und anstelle des Titels" " definieren, wird eine leere Titelzeile angezeigt.

Beachten Sie bitte folgende Hinweise bei der Verwendung von Variablen, Parametern und Ausdrücken:

Parameter	Bedeutung
Abfrage	Bedienungs-/Fehlerhinweis, der in einem Dialogfeld angezeigt wird.
Ausdruck	Ausdruck, Befehl, String, Variable, Zahl oder eine Kombination aus diesen.
Dateiname	Eindeutiger Dateiname, evtl. mit Laufwerk und Pfad.
Feld	Eindeutiger Feldname oder Feldnummer einer Datendatei.
Label	Eindeutiger Name eines Labels.
Kommentar	Beschreibung oder Hinweise zu den Makro- oder Mischbefehlen (Dokumentation von Makros oder Mischvorgängen).
Makroname	Eindeutiger Makroname, evtl. mit Laufwerk und Pfad.
Meldung	Bedienungs-/Fehlerhinweis, der in einem Dialogfeld angezeigt wird.
Titel	Titel eines Dialogfeldes.
Unterausdruck	Ausdruck, Befehl, String, Variable, Zahl oder eine Kombination aus diesen.
Variable	Eindeutiger Variablenname.
Für die zuvor genannten Parameter dürfen keine reservierten Wörter (Mischbefehle oder Makro-Programmierbefehle) verwendet werden!	

Unter *Befehlsform* wird bei den nachfolgenden Beschreibungen die Syntax des jeweiligen Befehls angegeben. Verschiedene Befehle, die bereits unter den WordPerfect-Windows-Versionen 5.1/5.2 vorhanden waren, wurden in der Syntax geändert. Die meisten Befehle können jedoch unverändert übernommen werden. Die alte Version der Befehle, die auch hier noch verwendet werden kann, wird immer an zweiter Stelle angegeben, da beim Konvertieren älterer Makros diese Befehle nicht in die neue Syntax umgesetzt werden (z. B. *NumStr*, *Fraction*, *Integer* usw.). Um Makro-Programmierbefehle, Mischbefehle und produktspezifische Befehle auch optisch sofort unterscheiden zu können, werden in dem Original-WordPerfect-Handbuch die Makro-Programmierbefehle und die Mischbefehle generell in Großbuchstaben angegeben (z. B. *DIALOGADDCHECKBOX*). Weil längere Wörter in dieser Schreibweise aber schlecht zu lesen sind, werden sie nachfolgend in Groß-/Kleinbuchstaben angegeben, z. B. *DialogAddCheckBox*.

6.1 AppActivate

Befehlsform: **AppActivate**(*Titel*)

Dieser Befehl tritt immer in Verbindung mit *AppExecute* und *AppLocate* auf. *AppActivate* holt ein geöffnetes Fenster (= gestartetes Programm) in den Vordergrund und macht es zum aktiven Fenster.

Parameter

Titel Hier ist die Variable anzugeben, die unter *AppLocate* definiert wurde. Es handelt sich hierbei um den Titel in der Titelleiste eines gestarteten Programms. Wenn Sie den genauen Titel nicht kennen, starten Sie das betreffende Programm probehalber, um den genauen Titel der Titelleiste zu entnehmen.

Beispiel

```
Application          (A1;"WordPerfect«;Default;"DE")
WinVar=              AppLocate("Rechner")
If                   (WinVar <> 0)
  AppActivate        (WinVar)
Else
  AppExecute         ("C:\WINDOWS\CALC.EXE")
EndIF
```

In diesem Beispiel soll mit dem Windows-Programm *Rechner* gearbeitet werden. Über *AppLocate* wird der Titel des Fensters für das über *AppExecute* zu startende Programm angegeben. Der Befehl *AppActivate* bringt das Fenster mit dem unter *AppLocate* angegebenen Titel in den Vordergrund, sofern es vorhanden ist (*WinVar* enthält in diesem Fall einen Wert ungleich 0). Dieses Feld wird in der *If*-Bedingung überprüft. Ist das angegebene Fenster nicht vorhanden, wird das unter *AppExecute* angegebene Programm gestartet, ansonsten wird das Fenster in den Vordergrund geholt.

Achtung: Unter *AppLocate* muß der genaue Name des Fenstertitels angegeben werden, sonst trifft die *If*-Bedingung nicht zu und das Programm wird jedesmal erneut gestartet, so daß mehrere Fenster desselben Programms vorhanden sind. In dem folgenden Beispiel wird *AppActivate* nicht ausgeführt, weil der Fenstername nach dem Starten nicht *Paintbrush*, sonder *Paintbrush – (unbenannt)* heißt. Um *AppActivate* trotzdem ausführen zu können, ohne das Programm ein weiteres Mal zu starten, müssen Sie unter *AppLocate* entweder den genauen Titel, wie zuvor genannt, eingeben oder *PaintBrush **.

```
Application          (A1;"WordPerfect";Default;"DE")
//WinVar=            AppLocate("Paintbrush")Wird nicht ordnungsgemäß ausgeführt.
WinVar=              AppLocate("Paintbrush *")
If                   (WinVar <> 0)
  AppActivate        (WinVar)
 Else
  AppExecute         ("C:\WINDOWS\PBRUSH.EXE")
EndIF
```

Weitere Hinweise siehe: AppClose, AppExecute, AppExecuteExt, AppLocate, AppShow

6.2 AppClose

Befehlsform: **AppClose**(*Titel*)

Dieser Befehl tritt immer in Verbindung mit *AppExecute* und *AppLocate* auf. *AppClose* schließt ein geöffnetes Fenster.

Parameter

Titel Hier ist die Variable anzugeben, die unter *AppLocate* definiert wurde. Es handelt sich hierbei um den Titel in der Titelleiste eines gestarteten Programms. Wenn Sie den genauen Titel nicht kennen, starten Sie das betreffende Programm probehalber, um den genauen Titel der Titelleiste zu entnehmen. Verwenden Sie ggf. einen Stern als Wildcard. Achten Sie auf Groß-/Kleinschreibung.

Ein Beispiel finden Sie unter *AppShow.*

Weitere Hinweise siehe: AppActivate, AppExecute, AppExecuteExt, AppLocate, AppShow

6.3 AppExecute

Befehlsform: **AppExecute**(*Befehlszeile*)

AppExecute startet das unter *Befehlszeile* angegebene Programm. Das kann ein Windows-, ein DOS-Programm oder eine Batchdatei sein. Sofern das Programm/die Batchdatei nicht in einem Suchpfad liegt, müssen die Laufwerksbezeichnung und der volle Pfadname angegeben werden. Ist das angegebene Programm/die Batchdatei nicht vorhanden, erfolgt bei der Ausführung des Befehls eine Fehlermeldung.

Parameter

Befehlszeile Name des zu startenden Programms oder einer Batchdatei.

Beispiel: Siehe unter AppActivate.

Weitere Hinweise siehe: AppLocate, AppExecuteExt

6.4 AppExecuteExt

Befehlsform: *Ergebnis*=**AppExecuteExt**(*Befehlszeile;Fenstertyp*)

Hier handelt es sich um eine erweitere Ausführung des Befehls **AppExecute**. **AppExecuteExt** startet das unter *Befehlszeile* angegebene Programm. Das kann ein Windows-, ein DOS-Programm oder eine Batchdatei sein. Sofern das Programm/die Batchdatei nicht in einem Suchpfad liegen, müssen die Laufwerksbezeichnung und der volle Pfadname angegeben werden. Ist das angegebene Programm/die Batchdatei nicht vorhanden, erfolgt bei der Ausführung des Befehls keine Fehlermeldung, das Makro wird auch nicht abgebrochen. Nach der Ausführung des Befehls wird in *Ergebnis* ein Wert zurückgegeben, der zu überprüfen ist.

Parameter

Ergebnis Nach der Ausführung des Befehls wird von Windows in diese Variable ein Wert zurückgegeben, der zu überprüfen ist. Werte kleiner als 32 bedeuten eine Fehlermeldung (Programm kann z. B. nicht ausgeführt werden). Sie müssen dann entscheiden, wie das Makro weiterarbeiten soll. Entnehmen Sie bitte Ihrem Windows-Handbuch, was die einzelnen Werte bedeuten.

Befehlszeile Geben Sie in Hochkommata den Namen des zu startenden Programms bzw. der Batchdatei ein (evtl. vollqualifiziert).

Fenstertyp Geben Sie an, wie das Fenster der gestarteten Anwendung angezeigt werden soll:

Current State!	Fenster an aktueller Position und Größe anzeigen (aktiv).
CurrentStateNonActivate!	Fenster in aktuellem Status anzeigen. Das derzeit aktive Fenster bleibt aktiv.
Default!	Standardeinstellung der angegebenen Anwendung.
Hide!	Fenster im Hintergrund ablegen.
Maximize!	Fenster in voller Größe anzeigen (aktiv),
MaximizeNonActivate!	Fenster auf Symbolgröße verkleinern (inaktiv).
Minimize!	Fenster auf Symbolgröße verkleinern (aktiv).
MinimizeActivateTopLevel!	Fenster auf Symbolgröße verkleinern und nächstes Fenster aktivieren.
Normal!	Fenster in normaler Größe anzeigen.
RecentStateNonActivate!	Fenster in bisheriger Größe/Position anzeigen. Das derzeit aktive Fenster bleibt aktiv.
Restore!	Fenster aktivieren und in Originalgröße anzeigen.

Beispiel

```
Application          (A1;"WordPerfect";Default;"DE")
WinVar=              AppLocate("Paintbrush *")
If                   (WinVar <> 0)
  AppActivate        (WinVar)
Else
 Ergebnis=           AppExecuteExt("C:\WINDOWS\PBRUSH.EXE";3)
EndIF
If                   (Ergebnis < 32)
  Prompt             ("Fehler";"Paintbrush konnte nicht ausgeführt werden";;;)
  Pause
  EndPrompt
EndIf
```

In diesem Beispiel wird versucht, das Programm *Paintbrush* zu starten und das Fenster in voller Größe anzuzeigen. Nach der Ausführung des Befehls wird in *Ergebnis* ein Wert zurückgegeben, der zu überprüfen ist. Das Programm wurde dann erfolgreich gestartet, wenn dieser Wert nicht kleiner als 32 ist.

Weitere Hinweise siehe: AppActivate, AppLocate, AppExecute

6.5 Application

Befehlsform: **Application**(*ProduktAbkürz;ProduktKennung;DEFAULT;Sprache*)

Makros können Sie in verschiedenen WordPerfect-Produkten aufzeichnen oder ausführen. Mit diesem Befehl wird u. a. angegeben, mit welchem Programm gearbeitet wird (siehe unten). Der folgende Befehl wird jedem WordPerfect-Makro bei der Aufzeichnung automatisch vorangestellt:

Application(A1;"WordPerfect";Default;"DE")

Erstellen Sie ein neues Makro im Dokumentfenster ohne die Funktion [*Tools, Makro, Aufzeichnen*], müssen Sie selbst dafür Sorge tragen, daß dieser Befehl am Beginn des Makros erscheint. Erstellen Sie sich am besten eine Datei, die nur diesen einen Eintrag enthält. Wenn Sie ein neues Makro direkt im Dokumentfenster erstellen, kopieren Sie einfach diese Datei an den Beginn des neuen Makros, oder definieren Sie sich diese Zeile als eine Abkürzung unter [*Einfügen, Abkürzungen, Erstellen*]. Im Gegensatz zu anderen Makrobefehlen ist dieser Befehl immer ab der Stelle im Makro aktiv, an der er definiert wurde. Wurde er z. B. in eine *If*-Schleife eingebunden, wird er auch dann ausgeführt, wenn die *If*-Bedingung nicht zutrifft. D. h. ab der eingefügten Stelle gelten die neuen Parameter immer so lange, bis sie durch denselben Befehl mit anderen Parametern widerrufen werden.

Parameter

ProduktAbkürz Zweistellige, eindeutige Abkürzung des Produkts, in dem die produktspezifischen Befehle ausgeführt werden sollen, z. B. *A1* für WordPerfect 7.

ProduktKennung Kennung des betreffenden Programms, z. B. »WordPerfect«.

Default Dieser Parameter ist wahlfrei. Wenn er verwendet wird, wird die Produktabkürzung als Default verwendet. Alle Produktbefehle werden dann so interpretiert, als würde ihnen die zweistellige Produktabkürzung vorausgehen.

Sprache Zweistelliger Sprachenschlüssel der zugehörigen WordPerfect-Version, bei der deutschen Version also DE. Dieser Sprachenschlüssel wird auch in den zugehörigen Makro-Systemdateien verwendet.

Wird dieser Befehl nicht verwendet, müssen Sie innerhalb von WordPerfect jedem produktspezifischen Befehl die Produktabkürzung *A1.* voranstellen.

Beispiel

```
Application          (A1;"WordPerfect";Default;"DE")
Type                 ("Beispiel für Application mit Default")
HardReturn           ()
Quit
```

Weitere Hinweise siehe: NewDefault, EndApp

6.6 AppLocate

Befehlsform: *Variable*=**AppLocate**(*Titel*)

Dieser Befehl tritt immer in Verbindung mit *AppActivate* auf. *AppLocate* prüft alle Titel der geöffneten Programmfenster mit dem unter *Titel* angegebenen Text. Wird ein Fenster mit einem identischen Titel gefunden, wird in *Variable* der Wert 0 gespeichert. Über *AppActivate* kann dieses Fenster dann aktiviert werden. Ist der genaue Name eines Fenstertitels nicht bekannt, können Sie anstelle der unbekannten Zeichen einen Stern * eingeben. Wird **nur** ein Stern verwendet, wird das derzeit aktive Fenster zugeordnet. Der Befehl kann auch direkt in Verbindung mit Befehlen wie *Assign*, *If*, *Type* usw. verwendet werden (siehe Kapitel »5.17 Änderungen gegenüber WPWin 5.1/5.2«).

Parameter

Titel Es handelt sich hierbei um den Titel in der Titelleiste eines aktivierten Programms. Wenn Sie den genauen Titel nicht kennen, starten Sie das betreffende Programm probehalber, um den genauen Titel der Titelleiste zu entnehmen. Andernfalls können Sie anstelle der unbekannten Zeichen einen Stern * verwenden. Besonders bei Programmen, in denen Daten bearbeitet und gespeichert werden können wie z. B. *Write*, *PBrush* usw., ist die Titelzeile wegen des verwendeten Dateinamens ständigen Änderungen unterworfen.

Beispiel: Siehe unter AppActivate.

Weitere Hinweise siehe: AppActivate, AppExecute, AppExecuteExt

6.7 AppShow

Befehlsform: **AppShow**(*Titel;Status*)

Dieser Befehl tritt immer in Verbindung mit *AppExecute* und *AppLocate* auf. *AppShow* definiert einen bestimmten Anzeigestatus, in dem das betreffende Fenster (=Anwendung) dargestellt werden soll.

Parameter

Titel Hier ist die Variable anzugeben, die unter *AppLocate* definiert wurde. Es handelt sich hierbei um den Titel in der Titelleiste eines gestarteten Programms. Verwenden Sie ggf. einen Stern als Wildcard. Achten Sie auf Groß-/Kleinschreibung.

Status Geben Sie den jeweils gewünschten Parameter an (Beschreibung siehe unter *AppExecuteExt*):

CurrentState!	MinimizeNoActivated!
CurrentStateNoActivate!	MinimizeActivateTopLevel!
Default!	Normal!
Hide!	RecentStateNoActivate!
Maximize!	Restore!
Minimize!	

Beispiel

```
Application             (A1;"WordPerfect«;Default;"DE")
DialogDefine            ("DO1";190;120;200;150;OK! | Cancel!;"AppShow/AppClose-Beispiel")
DialogAddPushButton     ("DO1";"Taste1";110;35;80;14;;"Rechner aufrufen")
DialogAddPushButton     ("DO1";"Taste2";110;55;80;14;;"Rechner ausblenden")
DialogAddPushButton     ("DO1";"Taste3";110;75;80;14;;"Rechner schließen")
DialogLoad              ("DO1";"WordPerfect")
RegionShowWindow        ("DO1.Taste2";Hide!)
RegionShowWindow        ("DO1.Taste3";Hide!)
Aufruf=                 1
DialogShow              ("DO1";"WordPerfect";Anzeigen)
CallbackWait
Quit

Label                   (Anzeigen)
If                      (Anzeigen(5)=274)
                        DialogDestroy("DO1")
                        CallbackResume
                        Return
EndIf

Switch                  (Anzeigen(3))
  CaseOf  "OKBttn":     MessageBox(x;"OK-Taste";"OK wurde gedrückt")
  CaseOf "CancelBttn":  DialogDestroy("DO1")
                        CallbackResume
                        Return
  CaseOf  "Taste1":     AppExecute ("c:\windows\calc")
                        RegionShowWindow("DO1.Taste2";Show!)
                        RegionShowWindow("DO1.Taste3";Show!)
  CaseOf  "Taste2":     If (Aufruf=1)
                          AppShow("Rechner";Hide!)
                          Aufruf=0
                        Else
                         AppShow("Rechner";Restore!)
                         Aufruf=1
                        EndIf
  CaseOf  "Taste3":     AppClose("Rechner")
```

```
                    RegionShowWindow("DO1.Taste2";Hide!)
                    RegionShowWindow("DO1.Taste3";Hide!)
EndSwitch
Return
```

Dieses Beispiel finden Sie auf der CD-ROM unter dem Dateinamen APPSHOW.WCM.

Weitere Hinweise siehe: AppActivate, AppClose, AppExecute, AppExecuteExt, AppLocate

6.8 Assert

Befehlsform:	**Assert**(*Bedingung*) oder **Assert**(*num. Wert*)

Über diesen Befehl können verschiedene Abbruchbedingungen erzeugt werden, die Einfluß auf die Programmsteuerung nehmen können.

Parameter

Bedingung Hier können die Parameter *CancelCondition!*, *ErrorCondition!* oder *NotFoundCondition!* verwendet werden. Die Bedeutung entnehmen Sie bitte den nachfolgenden Beschreibungen. Die zuvor genannten Parameter können auch als numerische Werte abgeprüft werden:

CancelCondition! = 1
ErrorCondition! =2
NotFoundCondition! =3

CancelCondition!

Dieser Befehl verhält sich ähnlich dem Befehl *Return*(CancelCondition!), sofern dieser Befehl nicht in Verbindung mit *Case* oder *Case Call* verwendet wurde, d. h. es wird ein Abbruch simuliert. Nach dem Setzen der Abbruchbedingung *Assert*(*CancelCondition!*) wird der Befehl ausgeführt, der zuvor in Verbindung mit *OnCancel* definiert wurde. Die Ausführung ist nur dann möglich, wenn zuvor nicht der Befehl *Cancel*(*Off!*) ausgeführt wurde. Es erfolgt kein Rücksprung zu einem möglichen aufrufenden Befehl oder Makro (siehe auch *ReturnCancel*).

Beispiel

```
Application         (A1;"WordPerfect";Default;"DE")
Assign              (Zähler;0)
Display             (State:On!)
OnCancel            (Abbruch)
Label               (Anfang)
Assign              (Zähler;Zähler+1)
If                  (Zähler=10)
Assert              (CancelCondition!)
EndIf
```

```
Drucken=            NumStr(Zähler)
Type                (Drucken)
HardReturn          ()
Go                  (Anfang)
Label               (Abbruch)
Prompt              ("Abbruch";"Das Makro wurde beim 10. Durchlauf abgebrochen";;;)
Pause
EndPrompt
Quit
```

Dieses Makro addiert eine 1 zu dem Inhalt der Variablen *Zähler* und zeigt das jeweilige Ergebnis schrittweise am Bildschirm an. Die Schleife wird so lange wiederholt, bis die Variable *Zähler* den Wert 10 enthält. Durch *Assert*(*CancelCondition!*) wird dann eine Abbruchbedingung gesetzt. Danach tritt die *OnCancel*-Bedingung in Kraft, und das Makro verzweigt zu dem Label (*Abbruch*). Hier wird eine Meldung am Bildschirm angezeigt, die auf den Abbruch hinweist. Nach dem Klicken auf [*OK*] oder auf [*Abbrechen*] wird das Makro beendet.

Dieses Beispiel finden Sie auf der CD-ROM unter dem Dateinamen ASSERTCA.WCM.

ErrorCondition!

Dieser Befehl verhält sich ähnlich dem Befehl ***Return***(*ErrorCondition!*), sofern dieser Befehl nicht in Verbindung mit *Case* oder *Case* Call verwendet wurde, d. h. es wird eine Fehlerbedingung gesetzt. Nach dem Setzen von *Assert*(*ErrorCondition!*) (= Fehlerbedingung) wird der Befehl ausgeführt, der zuvor in Verbindung mit *OnError* definiert wurde. Die Ausführung ist nur dann möglich, wenn zuvor nicht der Befehl *Error*(*Off!*) ausgeführt wurde. Es erfolgt kein Rücksprung zu einem evtl. aufrufenden Befehl oder Makro (siehe auch *ReturnError*).

Beispiel

```
Application         (A1;"WordPerfect";Default;"DE")
Assign              (Zähler;0)
Display             (On!)
OnError             (Abbruch)
Label               (Anfang)
Assign              (Zähler;Zähler+1)
If                  (Zähler=10)
Assert              (ErrorCondition!)
EndIf
Type                (Zähler)
HardReturn          ()
Go                  (Anfang)
Label               (Abbruch)
Prompt              ("Abbruch";"Das Makro wurde beim 10. Durchlauf abgebrochen";;;)
Pause
EndPrompt
Quit
```

Dieses Makro addiert eine 1 zu dem Inhalt der Variablen *Zähler* und zeigt das Ergebnis schrittweise am Bildschirm an. Die Schleife wird so lange wiederholt, bis die Variable *Zähler* den Wert 10 enthält. Durch *Assert* (*ErrorCondition!*) tritt dann die *OnError*-Bedingung in Kraft,

das Makro verzweigt zu dem Label (*Abbruch*). Hier wird eine Meldung am Bildschirm angezeigt, die auf den Abbruch hinweist. Nach dem Klicken auf [*OK*] oder auf [*Abbrechen*] wird das Makro beendet.

Dieses Beispiel finden Sie auf der CD-ROM unter dem Dateinamen ASSERTER.WCM.

NotFoundCondition!

Dieser Befehl verhält sich ähnlich dem Befehl *Return*(*NotFoundCondition!*), sofern dieser Befehl nicht in Verbindung mit *Case* oder *Case Call* verwendet wurde, d. h. es wird eine »Nicht gefunden«-Bedingung gesetzt. Nach dem Setzen von *Assert*(*NotFoundCondition!*) wird der Befehl ausgeführt, der zuvor in Verbindung mit *OnNotFound* definiert wurde. Es erfolgt kein Rücksprung zu einem evtl. aufrufenden Befehl oder Makro. Beachten Sie bitte auch die Informationen zu *Return*(*NotFoundCondition!*).

Beispiel

```
Application          (A1;"WordPerfect";Default;"DE")
OnNotFound           (Abbruch)
Label                (Anfang)
GetString            (Eingabe;"Bitte beliebigen Text eingeben. Zum Erzeugen der
                     AssertNotFound-Bedingung ""Ende"" eingeben";"Befehl
                     ""AssertNotFound""  testen")
If                   (Eingabe ="Ende")
Assert               (NotFoundCondition!)
EndIf
Meldung=             ("Eingabe = "+Eingabe)
Prompt               ("Befehl  ""AssertNotFound""  testen";Meldung;;;)
Pause
EndPrompt
Go                   (Anfang)
Label                (Abbruch)
Prompt               ("Abbruch«;"Das Makro wurde abgebrochen";;;)
Pause
EndPrompt
Quit
```

Dieses Makro verlangt über *GetString* eine beliebige Eingabe, die danach am Bildschirm über *Prompt* angezeigt wird. Die Schleife wird so lange wiederholt, bis Sie *Ende* eingeben. Durch *Assert*(*NotFoundCondition!*) tritt dann die *OnNotFound*-Bedingung in Kraft (es wird eine Nicht-Gefunden-Bedingung simuliert), das Makro verzweigt zu dem Label (*Abbruch*). Hier wird eine Meldung am Bildschirm angezeigt, die auf den Abbruch hinweist. Nach dem Klikken auf [*OK*] oder auf [*Abbrechen*] wird das Makro beendet. Dieses Beispiel finden Sie auf der CD-ROM unter dem Dateinamen ASSERTNF.WCM.

Weitere Hinweise siehe: CancelOff, Condition, ErrorOff, OnCancel, OnCondition, OnError, OnNotFound, ReturnCancel, ReturnError, ReturnNotFound

6.9 Assign

Befehlsform: **Assign**(*Variable;Ausdruck*) oder
Variable=*Ausdruck*

Zuordnung eines Wertes zu einer Variablen bzw. Definition einer Variablen. Eine Variable ist ein Speicherfeld, das Daten aufnehmen, speichern und wieder abgeben kann. Sie können beliebig viele Variable definieren, wobei Sie für die Namensgebung alle Zeichen verwenden können. Handelt es sich bei dem Variableninhalt um eine Formel, wird nach der Ausführung das Ergebnis in dieser Variablen gespeichert. Andernfalls wird der eingegebene (zugeordnete) Wert als Zeichenfolge behandelt. Den in einer Variablen gespeicherten Wert können Sie beliebig oft verändern bzw. abrufen. Die Makros bleiben dadurch flexibel, weil anstelle von fixen Werten (= Konstanten) immer mit den aktuellen Variableninhalten gearbeitet werden kann. Beachten Sie bitte auch Kapitel »5.17 Änderungen gegenüber WPWin 5.1/5.2«.

Einer Variablen können Sie

- fixe numerische Werte zuweisen.
- Werte zuweisen, die aufgrund einer erfolgten Berechnung, z. B. einer Formel, ermittelt wurden.
- Texte zuweisen.

Achtung: Löschen Sie am Beginn des Makros die Inhalte aller Variablen, oder weisen Sie ihnen bestimmte Anfangswerte zu (= Initialisierung). Achten Sie besonders dann auf Variableninhalte, wenn sie zu Beginn oder innerhalb von Schleifen angesprochen werden. Setzen Sie den Inhalt ggf. auf bestimmte Ausgangswerte. Durch die Zuweisung eines neuen Wertes wird der bisherige Inhalt der Variablen durch den neuen Wert überschrieben. Der bisherige Wert geht dadurch verloren.

Parameter

Variable Beliebiger Variablenname. Die Verwendung eines reservierten Wortes (siehe Kapitel »5.11 Reservierte Wörter«) ist hier nicht erlaubt. Der Variablenname darf nicht in Hochkommata eingeschlossen sein. Es wird keine Unterscheidung zwischen Groß- und Kleinbuchstaben getroffen. Die Variablennamen *Eingabe*, *eingabe* oder *EiNgAbE* sind darum identisch.

Ausdruck Angaben einer oder mehrerer Variablen, fixer Text, numerische Werte oder eine Kombination aus diesen Angaben, maximal 512 Zeichen lang. Wird dieser Befehl als Mischbefehl verwendet, dürfen alphanumerische Konstanten nicht in Hochkommata eingeschlossen werden.

Bei diesem Befehl stehen Ihnen zwei Anwendungsmöglichkeiten zur Verfügung, die dasselbe Ergebnis erzielen:

```
Assign (Text2;"WordPerfect für Windows")
Assign (Text1;"WordPerfect")              oder
Assign (Text1;Text2)                      oder

Text1= "WordPerfect"                      oder
Text1= Text2
```

Beispiel

1. Assign (Zähler;200) oder
   Zähler= 200

   Der Variablen *Zähler* wird der fixe Wert 200 zugewiesen. Numerische Werte dürfen nicht in Hochkommata eingeschlossen werden. In diesem Fall kann die Variable *Zähler* in Rechenoperationen verwendet werden.

2. Assign (Zähler;"200") oder
   Zähler= "200"

   Der Variablen *Zähler* wird der String »200« zugewiesen. Diese Variable kann nicht in Rechenoperationen verwendet werden, da sie keinen numerischen Inhalt hat.

3. Assign (Zähler;-200,50) oder
   Zähler= -200,50

   Der Variablen *Zähler* wird der fixe Wert -200,50 (= negativ) zugewiesen.

4. Assign (Zähler;Zwischensumme) oder
   Zähler= Zwischensumme

   Der Variablen *Zähler* wird der Inhalt der Variablen *Zwischensumme* zugewiesen. Variablennamen dürfen nicht in Hochkommata angegeben werden, sonst wird nicht der Inhalt der betreffenden Variablen zugeordnet, sondern die zwischen den Hochkommata stehenden Zeichen:

   Assign (Zähler;"Zwischensumme") oder
   Zähler= "Zwischensumme"

   In diesem Beispiel wird der Variablen *Zähler* der Text (= String) "Zwischensumme" zugeordnet.

5. Assign (Text1;"WordPerfect") oder
   Text1= »WordPerfect«

   Der Variablen *Text1* wird der fixe Text *WordPerfect* zugewiesen. Fixe Texte oder Zeichen müssen in Hochkommata angegeben werden, ansonsten wird der hinter dem Semikolon stehende Begriff als Variablenname interpretiert.

6. Assign (Text2;"") oder
   Text2= ""

   Der Inhalt der Variablen *Text2* wird gelöscht. Der bisherige Inhalt der Variablen ist dann nicht mehr verfügbar.

7. Assign (Zahl1;10)
   Assign (Zahl2;60)
   Assign (Zahl2;Zahl1*Zahl2)
   Type NumStr(Zahl2)

   In den ersten beiden Befehlen werden den Variablen *Zahl1* und *Zahl2* die Werte 10 bzw. 60 zugewiesen (= fixe Werte). In dem dritten Befehl wird das Produkt von *Zahl1* und *Zahl2* ermittelt (Multiplikation von *Zahl1* mit *Zahl2*) und in *Zahl2* gespeichert (= errechneter Wert). Der ursprüngliche Wert von *Zahl2* geht dadurch unwiederbringlich verloren, d. h. er wird durch die neue Zuweisung überschrieben. Über den Befehl *NumStr* wird das Ergebnis zum Drucken in einen String konvertiert (sonst kann es nicht gedruckt werden). Der Befehl *Type* fügt in Verbindung mit *NumStr* das Ergebnis (in diesem Fall 600) im Dokument auf Cursorposition ein.

8. Assign (Zahl1;Zahl1+50) oder
   Zahl1= Zahl1+50

   Zum Inhalt der Variablen *Zahl1* wird der fixe Wert 50 addiert.

9. Geldwert= 100
   Assign (Text1;"Die Rückerstattung beträgt"+Geldwert+",00 DM.")

   Nach der Ausführung des *Assign*-Befehls enthält die Variable *Text1* die Kombination der beiden in Hochkommata eingeschlossenen Texte, ergänzt um den Inhalt der Variablen *Geldwert*. Enthält diese Variable den Inhalt 100, entsteht folgendes Ergebnis:

   Die Rückerstattung beträgt 100,00 DM.

   Dieser Text könnte jetzt z. B. über *Prompt* am Bildschirm angezeigt werden.

10. Umsatz[Werk;Abteilung;Gruppe]=StückPreis * Menge

    Die Variable *Umsatz* ist ein Element eines dreidimensionalen Arrays, das über die Indizes *Werk*, *Abteilung* und *Gruppe* angesprochen wird (siehe Befehl Declare).

    Ergebnis= (Monate[Jahr;Monat] * 12)

    Das Ergebnis errechnet sich aus dem Array-Element *Monate* in Verbindung mit den Indizes *Jahr* und *Monat*, multipliziert mit 12.

Beachten Sie bitte auch die mathematischen Operatoren in Kapitel »5.10 Mathematische Funktionen«.

6.10 BIF-Befehle

Bei einer lokalen Installation wird die persönliche BIF-Datei WPCSET.BIF erzeugt. In dieser Datei werden alle Programm-Einstellungen (z. B. Veränderungen in den Standardeinstellungen) gespeichert. Bei jedem Starten von WordPerfect und beim Ändern von Standardeinstellungen wird auf diese Datei zurückgegriffen. Beim Installieren von WordPerfect in einem Netzwerk wurde bei der Version 6.1 nach dem Setzen der Standardeinstellungen (Vorgaben) die gemeinsame BIF-Datei WPCNET.BIF erzeugt, durch die gewährleistet war, daß für alle

Netzwerkbenutzer eine einheitliche Benutzerumgebung zur Verfügung stand. Bei WordPerfect 7 gibt es bei der Netzwerk-Installation keine BIF-Dateien mehr. Die Datei für die zentrale Netzwerkeinstellung heißt NAMADMIN.IP und die des Benutzers einer Arbeitsstation NAM.IP (IP = Installation Profile).

Manuelle Änderungen an dieser BIF-Datei sind nicht erforderlich, da sie aufgrund der in WordPerfect vorgenommenen Modifikationen automatisch geändert wird. Über die hier verfügbaren Befehle können Sie anstelle des BIF-Editors direkt von WordPerfect-Makros aus BIF-Dateien verändern. Die Befehle finden Sie mit Hilfe des Makrobefehlsmanagers unter [*Typ*, *WP MacroFacility*].

Hinweis: Da diese Befehle von den wenigsten Lesern benötigt werden, wurde wegen begrenzten Platzes in diesem Buch auf eine detaillierte Beschreibung zugunsten wichtigerer Befehle verzichtet. Schlagen Sie bitte bei Bedarf im Makro-Online-Handbuch nach, wenn Sie diese Befehle benötigen. Beachten Sie auch die Beispiele.

6.11 Beep

Befehlsform: Beep

Ertönen eines akustischen Signals (z. B. für Makro-Ende, Aufforderung zur Dateneingabe, Anzeigen einer Meldung). Verwenden Sie diesen Befehl, um den Bediener akustisch auf wichtige Informationen oder auf Fehler hinzuweisen. Die Anwendung kann auch dann sinnvoll sein, um das Makroende bei sehr lange laufenden Makros anzuzeigen. Der Bediener ist dadurch nicht gezwungen, ständig den Bildschirm zu beobachten.

Beispiel

1. Beep
   GetString (KenBu;"Bitte Kennung eingeben";"Änderungsdienst";;1)

   Nach dem Ertönen des akustischen Signals können Sie ein Zeichen eingeben. Es wird in der Variablen *KenBu* gespeichert.

2. Beep
   GetString (Text1;"Bitte Vor- und Nachname eingeben.";"Adresse erfassen")

   Nach dem Ertönen des akustischen Signals können Sie einen beliebigen Text eingeben.

3. Beep Beep
   Prompt ("Fehler";"Falsche Kennung eingegeben";;;)

   Nach dem zweimaligen Ertönen des akustischen Signals wird die unter *Prompt* angegebene Meldung in einem Dialogfeld angezeigt.

6.12 Break

Befehlsform: Break

Mit diesem Befehl kann eine Schleife vorzeitig beendet werden. *Break* wird in Verbindung mit den Befehlen *For*, *ForEach*, *ForNext*, *Repeat*, *Switch* und *While* verwendet. Nach der Ausführung dieses Befehls werden die restlichen Befehle der abgebrochenen Schleife übersprungen und mit den Befehlen, die unmittelbar dem zugehörigen *End...*-Befehl folgen (z. B. EndFor bei einer ForNext-Schleife), weitergearbeitet. Sind mehrere Schleifen geschachtelt, wird nur die aktuelle Schleife abgebrochen.

Beispiel

```
Application          (A1; "WordPerfect"; Default; "DE")
Anfang=              1
Ende=                1000
Abstand=             1
Meldung1=            "Das Makro wurde vollständig ausgeführt."
GetNumber            (Abbruch;"Das Makro erstellt eine Zahlenreihe von 1 - 1000. Geben Sie an,
                     bei welcher Zahl das Makro enden soll.";"Befehl ""BREAK"" testen")
ForNext              (Zähler;Anfang; Ende; Abstand)
   If                (Zähler=Abbruch)
      Meldung1=      "Das Makro wurde bei dem eingegebenen Wert "+Zähler+" abgebrochen."
      Break
   EndIf
   Meldung=          "Das Makro zählt bis zu dem angegebenen Wert "+Abbruch+": "+Zähler
   Prompt            ("Befehl ""BREAK"" testen";Meldung;;;)
EndFor
Prompt               ("Befehl ""BREAK"" testen";Meldung1;;;)
Wait                 (50)
EndPrompt
```

In diesem Beispiel wird eine Zahlenreihe von 1–1000 erstellt. Über ein Dialogfeld wird eine Zahl eingegeben, bei der das Makro abgebrochen werden soll. Wird eine Zahl größer als 1000 eingegeben, endet das Makro ordnungsgemäß.

Weitere Hinweise siehe: For, ForEach, ForNext, Repeat, Switch, While

6.13 Call

Befehlsform: **Call**(*Label*) oder
Call *Label* (*Parameter*)

Ruft eine Unterroutine, eine Procedure oder eine Function desselben Makros auf. Die Makro-Ausführung wird an der aktuellen Stelle unterbrochen, um den unter *Label* angegebenen Ma-

kroteil (= Unterroutine) auszuführen. Die Unterroutine muß mit *Return* oder *EndProc* bzw. *EndFunc* enden. Nach deren Ausführung wird mit dem Befehl fortgefahren, der dem aufrufenden *Call*-Befehl folgt.

Bei der Verwendung der zweiten Varianten können Sie Variable an die Unterroutinen *Procedure* und *Function* übergeben (siehe Beispiel 3).

Verwenden Sie Unterroutinen dann,

- wenn ein und dieselbe Befehlsfolge innerhalb des Makros an verschiedenen Stellen erforderlich ist, also mehrmals vorkommt. Anstelle vieler identischer Definitionen an unterschiedlichen Stellen wird eine Programmroutine (= Befehlsfolge, Unterroutine) einmal erstellt und entsprechend oft aufgerufen.
- wenn Sie ein sehr großes Makro erstellt haben. Fassen Sie hier logisch zusammengehörende Befehle zu einer oder mehreren Unterroutinen zusammen, und rufen Sie diese an entsprechender Stelle mit *Call* auf. Das Makro wird dadurch wesentlich übersichtlicher.

In den aufgerufenen Unterroutinen können Sie (wenn erforderlich) Bedingungen setzen und aufgrund dieser die Unterroutine **vorzeitig** in Verbindung mit einer Fehlerbedingung verlassen (= Abbruch):

gesetzte Bedingung:	**wird abgeprüft durch:**
Return	keine Prüfung, nur Rücksprung
Return(CancelCondition!)	OnCancel
Return(ErrorCondition!)	OnError
Return(NotFoundCondition!)	OnNotFound

Nach dem Rücksprung aus der Unterroutine können Sie den Grund des Abbruchs überprüfen (siehe oben) und im Makro entsprechend reagieren. Der Befehl, der die gesetzte Bedingung abprüft (z. B. *OnCancel*), muß im Makro vor dem Call-Aufruf ausgeführt worden sein.

Parameter

Label Der hier angegebene Label-Name muß dem Namen einer aufzurufenden Unterroutine entsprechen. Die Unterroutine muß sich innerhalb desselben Makros befinden. Der Name kann mit dem Zeichen @ enden. Er darf nicht in Hochkommata angegeben werden. Beachten Sie bitte auch die Hinweise bei dem Befehl **Label**.

Beispiel

```
1.        Label               (Anfang)
            ::::::
          Label               (Abfrage2)
             Menu             (Auswahl;Digit;;;{"Drucker";"Bildschirm"})
             Case Call        (Auswahl;{......}...)
             Go               (Anfang)
          Label(Ab2-Gerät)
             If               (Auswahl=1)
             Call             (Drucker)
                    Else
             Call             (Bildschirm)
```

```
    EndIf
Return
Label                (Abfrage3)
  ::::::
    Go
Label                (Drucker)
  ::::::
Return
Label                (Bildschirm)
  ::::::
Return
```

Das Beispiel zeigt Ausschnitte eines größeren Makros. Unter dem Label »Abfrage2« wird folgendes Dialogfeld eingeblendet:

1. Drucker
2. Bildschirm

Erlaubte Eingaben sind 1 oder 2. Die eingegebene Ziffer wird durch den Befehl *Menu* in der Variablen *Auswahl* gespeichert. Unter dem Label *Ab2-Gerät* werden aufgrund der Menüauswahl unterschiedliche Unterroutinen aufgerufen. Aufgrund der eingegebenen Kennziffer wird eine der Call-Anweisungen ausgeführt. Wurde 1 eingegeben, wird die Unterroutine (*Drucker*) aufgerufen. Die darin definierten Befehle werden abgearbeitet. Danach erfolgt ein Rücksprung zu dem Befehl, der dem aufrufenden *Call*-Befehl folgt. Der Rest der *If*-Bedingung trifft nicht mehr zu (wird also nicht mehr ausgeführt), durch den *Return*-Befehl erfolgt wiederum ein Rücksprung in die Routine (*Abfrage2*).

2.
```
Label                (Anfang)
OnError              (Fehler)
Call                 (Prüfen)
Call                 (Drucken)
:::::
Label                (Prüfen)
If                   (Zähler=5)
Return               (ErrorCondition!)
EndIf
Assign               ::::
Call                 ::::
Prompt
::::::
Return
Label                (Fehler)
Prompt               ("Fehler";"Falscher Wert";;;)
Go                   (Anfang)
```

Wenn in der aufgerufenen Unterroutine (*Prüfen*) die Variable *Zähler* den Wert 5 enthält, wird die Bedingung *Return*(*ErrorCondition!*) gesetzt. Die Unterroutine wird an dieser Stelle verlassen. Die restlichen Befehle der Unterroutine werden nicht ausgeführt. Nach dem Rücksprung wird aufgrund des *OnError*-Befehls und der gesetzten Bedingung nach »Fehler« verzweigt.

```
3.          Call                Verarbeiten (Zeit;Gehalt)
            :::::
            Quit
            Procedure           Verarbeiten (Zeit;Gehalt)
            ::::
            EndProc
```

In diesem Beispiel wird die Procedure *Verarbeiten* aufgerufen und die in Klammern angegebenen Variablen übergeben. Sinngemäß wäre auch die Handhabung in Verbindung mit *Function*.

Weitere Hinweise siehe: Case Call, Run, Function, Procedure

6.14 CallbackResume / CallbackWait

Befehlsform:	CallbackResume CallbackWait

In Verbindung mit einer Callback-Routine kann über CallbackWait eine Makro-Pause erzwungen werden, um die Callback-Routine auszuführen. Über CallbackResume wird die Pause beendet. Mit diesen beiden Befehlen wird die Handhabung der Callback-Routine, die bei früheren Versionen z. B. über eine While-Schleife gesteuert wurde, vereinfacht.

Beispiel

1. Mit While-Schleife

```
:::::
DialogShow          ("Dialogfeld1";"WordPerfect";Anzeigen)
Schleife=           1
While               (Schleife = 1)
EndWhile
Quit
Label               (Anzeigen)
If                  (Anzeigen[5]=274)              // Siehe Abschnitt »6.37.32 DialogShow"
   DialogDestroy    ("Dialogfeld1")
   Schleife=        0
   Return
EndIf
If                  (Anzeigen[3]="OKSchaltfl"
   DialogDestroy    ("Dialogfeld1")
   Schleife=        0
   Return
EndIf
Return
```

2. Mit CallbackWait und CallbackResume

```
:::::
DialogShow          ("Dialogfeld1";"WordPerfect";Anzeigen)
```

```
CallbackWait
Quit
Label               (Anzeigen)
If                  (Anzeigen[5]=274)              // Siehe Abschnitt "6.37.32
                                                   // DialogShow"
   DialogDestroy    ("Dialogfeld1")
   CallbackResume
   Return
EndIf
If                  (Anzeigen[3]="OKSchaltfl"
   DialogDestroy    ("Dialogfeld1")
   CallbackResume
   Return
EndIf
Return
```

Weitere Beispiele finden Sie unter AppShow und DialogAddBitmap.

Beachten Sie bitte auch in der Makro-Online-Hilfe die umfangreichen Parameter und Beschreibungen zu »Callback Data Discription«.

6.15 Cancel

Befehlsform: *Variable*=**Cancel**(*Status*) oder **Cancel**(*Status*)

Durch Drücken von [Esc], [Strg-Untbr] oder durch Setzen der Bedingung *Assert*(*CancelCondition!*) bzw. *Return*(*CancelCondition!*) kann eine Makroausführung abgebrochen werden, d. h. es wird eine Abbruchbedingung gesetzt. Mit dem *Cancel*-Befehl können Sie festlegen, wie eine Abbruchbedingung behandelt werden soll. Die Standardvorgabe ist *Cancel*(*On!*).

Parameter

Variable Diese Variable enthält den Status des zuletzt ausgeführten *Cancel*-Befehls, der hier bei Bedarf abgefragt werden kann, um im Makro bestimmte Funktionen auszuführen. Ist die Variable leer, wurde der *Cancel*-Befehl vorher nicht ausgeführt. Die Variable kann auch weggelassen werden.

Status Über die beiden Parameter *On!* und *Off!* kann die weitere Makro-Verarbeitung gesteuert werden.

Begrenzen Sie nur bestimmte Routinen innerhalb des Makros durch die Befehle *Cancel*(*Off!*) und *Cancel*(*On!*), damit Sie für die restlichen Routinen [*Abbrechen*] oder [Esc] wie gewohnt nutzen können.

Achtung: Innerhalb der Routinen, die durch *Cancel*(*Off!*) und *Cancel*(*On!*) eingegrenzt sind, können Sie das Makro (wenn erforderlich) dann trotzdem durch die Tastenkombination [Strg-Untbr] abbrechen. Sollte dies dennoch nicht funktionieren, müssen Sie in der Datei CONFIG.SYS den Befehl »BREAK=ON« ergänzen (standardmäßig wird »BREAK=OFF« angenommen).

Cancel(Off!)

Die normale Funktion von [*Abbrechen*] ist unterdrückt. In diesem Fall können Sie also die Makroausführung nicht durch Klicken auf [*Abbrechen*] oder durch Drücken von Esc beenden. Wenn Sie diesen Befehl nicht verwenden, wird standardmäßig *Cancel*(*On!*) angenommen, d. h. durch Klicken auf [*Abbrechen*] oder durch Drücken von Esc wird das Makro beendet.

Cancel(On!)

Die normale Funktion von [*Abbrechen*] wird nach einem vorausgegangenen *Cancel*(*Off!*) wieder aktiviert. Ist das der Fall können Sie die Makroausführung wieder durch Klicken auf [*Abbrechen*] oder Drücken von Esc beenden. Zwischen drei verschiedenen Abbrucharten müssen Sie unterscheiden:

Esc Durch Drücken von Esc wird ein Makro abgebrochen. Es wird eine Meldung eingeblendet, die auf den Abbruch hinweist.

Abbrechen Wird während der Makroausführung ein Dialogfeld eingeblendet, das eine Taste [*Abbrechen*] beinhaltet, können Sie durch Klicken auf dieser Taste das Makro abbrechen. Ob das Makro dann wirklich abgebrochen wird, hängt davon ab, ob *OnCancel* verwendet wurde und welche Befehle danach ausgeführt werden. Wenn Sie *Cancel*(*Off!*) im Makro verwenden, werden die Abbruchbefehle ignoriert.

ReturnCancel/Assert(..)
Während des Makroablaufs werden die Makro-Programmierbefehle *Return*(*CancelCondition!*) oder *Assert*(*CancelCondition!*) ausgeführt. Auch hier hängt die weitere Verarbeitung davon ab, ob *OnCancel* verwendet wurde und welche Befehle danach ausgeführt werden.

Beispiel: Siehe unter »OnCancel«

Weitere Hinweise siehe: OnCancel

6.16 Case

Befehlsform: **Case**(*Test*;{*Case1*;*Label1*;..*Case*n;*Label*n};[*Default Label*])

Über diesen Befehl wird aufgrund der unter *Test* angegebenen Bedingung entschieden, zu welchem Label verzweigt wird. Die Angabe in *Test* wird mit jeder *Case*-Bedingung verglichen. Trifft die Angabe bei einer Case-Bedingung zu, erfolgt eine Verzweigung zu dem zugehörigen Label (jeder Case-Bedingung muß zwingend ein Label-Parameter folgen). Nach der Abarbeitung der Befehlsfolge des betreffenden Labels erfolgt dann keine Rückkehr zu dem *Case*-Befehl. Trifft keine Bedingung zu, wird zu dem Label verzweigt, der als *Default Label* angegeben wurde. Es sind mehrere Bedingungen erlaubt.

Parameter

Test Hier wird in den meisten Fällen eine Variable definiert sein, der zuvor ein bestimmter Wert zugeordnet wurde. Die Verwendung eines Ausdrucks ist aber auch möglich.

Casen Dieser Parameter enthält die zu prüfende Bedingung, die aus Zahlen, Wörtern oder Ausdrücken bestehen kann. Zwischen den beiden geschweiften Klammern können Sie beliebig viele Bedingungen setzen. Jeder Bedingung muß ein Label zugeordnet sein. *Casen* und *Labeln* müssen immer paarweise auftreten.

Labeln Jedem Case-Parameter muß ein zugehöriger Label-Parameter folgen. Zu dem jeweils zugehörigen Label wird verzweigt, wenn die unmittelbar vorhergehende Case-Bedingung zutrifft. Beachten Sie bitte auch die Hinweise bei dem Befehl *Label*.

Default Label Trifft keine der unter ***Case***n angegebenen Bedingungen zu, wird zu dem hier angegebenen Label verzweigt. Beachten Sie bitte auch die Hinweise bei dem Befehl *Label*.

Beispiel

```
1.      Label               (Anfang)
        :::::
        Label               (Abfrage)
        Menu                (Auswahl;Digit;;;{"Zugang";"Änderung";"Abgang"})
        Case                (Auswahl;{1;Neuaufnahme;2;Änderung;3;Löschung;} Fehler)
        Label               (Neuaufnahme)
        :::::
        Go                  (Abfrage)
        Label               (Änderung)
        ::::::
        Go                  (Abfrage)
        Label               (Löschung)
        ::::::
```

```
        Go                  (Abfrage)
        Label               (Fehler)
        Prompt              ....
        Go                  (Abfrage)
```

Bei diesem Makro wird aufgrund eines eingegebenen Zeichens zu dem zugehörigen Label verzweigt. Wurde z. B. 1 eingetippt, verzweigt das Makro zu Neuaufnahme. Wird ein Zeichen eingetippt, das nicht den Bedingungen entspricht, wird eine Verzweigung zu dem *Default*-Label durchgeführt, sofern vorhanden.

- Der Befehl *Menu* bewirkt die Anzeige eines Auswahlmenüs:

  1 Zugang
  2 Änderung
  3 Abgang

  Nach der Anzeige des Menüs ist die Eingabe einer Ziffer möglich (1–3). Sie können auch mit der Maus auf dem gewünschten Menüpunkt klicken.

- Mit der *Case*-Struktur wird einerseits die Richtigkeit des zuvor eingegebenen Wertes überprüft und andererseits zu einer bestimmten Unterroutine verzweigt, wenn der zuvor eingegebene Wert den Bedingungen entspricht. Zugelassen sind in diesem Beispiel nur die Werte 1, 2 oder 3. Trifft eine der Bedingungen zu, wird zu dem zugehörigen Label verzweigt. In unserem Beispiel werden je nach Eingabe vier unterschiedliche Label angesprungen (drei Label über die Menüauswahl, der vierte Label ist der *Default Label*).

- Trifft keine der Bedingungen zu, wird zu dem *Default*-Label (= Fehler) gesprungen.

Der Übersichtlichkeit wegen wurden in den Unterroutinen keine Befehle angezeigt. Die Reihenfolge der Unterroutinen ist hier unerheblich.

Die Unterroutinen, zu denen aufgrund zutreffender Bedingungen verzweigt wird, müssen in der Regel mit einem *Go*-Befehl enden. Wenn Sie dies nicht beachten, treten unweigerlich Fehler auf, weil durch *Case* kein Rücksprung zum aufrufenden Befehl erfolgt.

Die Unterroutinen enden nicht mit einem *Go*-Befehl:

```
2.      Label               (Neuaufnahme)
        ::::::
        Label               (Änderung)
        ::::::
        Label               (Löschung)
        ::::::
        Label               (Anzeigen)
        ::::::
```

Wird 1 ausgewählt — verzweigt WordPerfect zu dem Label Neuaufnahme. Die danach folgenden Befehle werden ausgeführt und anschließend alle Befehle, die den Labeln »Veränderung«, »Löschung« und »Anzeigen« folgen, sofern eine dieser Routinen nicht mit *Go* verlassen wird.

Wird 2 ausgewählt — verzweigt WordPerfect zu dem Label Änderung. Die danach folgenden Befehle werden ausgeführt und anschließend alle Befehle, die dem Label »Löschung« und »Anzeigen« folgen, sofern eine dieser Routinen nicht mit *Go* verlassen wird.

Wird 3 ausgewählt — verzweigt WordPerfect zu dem Label Löschung. Die danach folgenden Befehle werden ausgeführt und anschließend alle Befehle, die dem Label »Anzeigen« folgen, sofern eine dieser Routinen nicht mit *Go* verlassen wird.

Weitere Hinweise siehe: Case Call, CaseOf, Label

6.17 Case Call

Befehlsform: **Case Call**(*Test*;{*Case1*;*Label1*;..*Case*n;*Label*n};[*Default Label*])

Über diesen Befehl wird aufgrund der unter *Test* angegebenen Bedingung entschieden, welche Unterroutine aufgerufen wird. Die Angabe in *Test* wird mit jeder *Case*-Bedingung verglichen. Trifft die Angabe zu, wird die Unterroutine des zugehörigen Labels ausgeführt. Mehrere Bedingungen sind erlaubt. Nach der Ausführung der Unterroutine wird mit dem der *Case Call*-Struktur folgenden Befehl fortgefahren. Die aufgerufene Unterroutine muß mit *Return* enden. Trifft keine Bedingung zu, wird die unter *Default*-Label angegebene Routine ausgeführt (sofern angegeben). In diesem Fall wird die Verarbeitung ebenfalls mit dem der *Case Call*-Struktur folgenden Befehl fortgesetzt.

Parameter

Test — Hier wird in den meisten Fällen eine Variable definiert sein, der zuvor ein bestimmter Wert zugeordnet wurde. Die Verwendung eines Ausdrucks ist aber auch möglich.

Casen — Dieser Parameter enthält die zu prüfende Bedingung, die aus Zahlen, Wörtern oder Ausdrücken bestehen kann. Zwischen den beiden geschweiften Klammern können Sie mehrere Bedingungen setzen. Jeder Bedingung muß ein Label zugeordnet sein. *Casen* und *Labeln* müssen immer paarweise auftreten.

Labeln — Jedem Case-Parameter muß ein zugehöriger Label-Parameter folgen. Die jeweils zugehörige Unterroutine (= Label) wird aufgerufen, wenn die unmittelbar vorhergehende Case-Bedingung zutrifft. Jede Label-Routine innerhalb eines Makros kann ausgeführt werden. Sie muß mit *Return* enden. Beachten Sie bitte auch die Hinweise bei dem Befehl *Label*.

Default Label — Trifft keine der unter Casen angegebenen Bedingungen zu, wird der hier angegebene Label aufgerufen. Beachten Sie bitte auch die Hinweise bei dem Befehl *Label*.

In den aufgerufenen Unterroutinen können Sie (wenn erforderlich) Bedingungen setzen und aufgrund dieser die Unterroutine **vorzeitig** in Verbindung mit einer Fehlerbedingung verlassen (= Abbruch):

gesetzte Bedingung:	**wird abgeprüft durch:**
Return	keine Prüfung, nur Rücksprung
Return(CancelCondition!)	OnCancel
Return(ErrorCondition!)	OnError
Return(NotFoundCondition!)	OnNotFound

Nach dem Rücksprung aus der Unterroutine können Sie den Grund des Abbruchs überprüfen (siehe oben) und im Makro entsprechend reagieren. Der Befehl, der die gesetzte Bedingung abprüft (z. B. OnCancel), muß im Makro vor dem Case-Call-Aufruf ausgeführt worden sein.

Beispiel

```
1.        Label               (Anfang)
          :::::
          Label               (Abfrage)
            Menu              (Auswahl;Digit;;;{"Zugang";"Änderung";"Abgang"})
            Case Call         (Auswahl;{1;Neuaufnahme;2;Änderung;3;Löschung;} Fehler)
            Prompt            ::::
            Assign            ::::
            Call              ::::
          Label               (Neuaufnahme)
            ::::::
            Return            (Abfrage)
          Label               (Änderung)
            ::::::
            Return            (Abfrage)
          Label               (Löschung)
            ::::::
            Return            (Abfrage)
          Label               (Fehler)
            ::::::
            Return            (Abfrage)
```

Bei diesem Makro wird aufgrund eines eingegebenen Wertes die zugehörige Unterroutine aufgerufen. Wurde z. B. 1 eingetippt, werden alle Befehle zwischen *Label*(*Neuaufnahme*) und dem zugehörigen *Return* ausgeführt. Wird ein Zeichen eingetippt, das nicht den Bedingungen entspricht, wird aufgrund der Makrosteuerung eine Verzweigung zu dem *Default*-Label durchgeführt, sofern vorhanden.

- Der Befehl *Menu* bewirkt die Anzeige eines Auswahlmenüs:

  1 Zugang
  2 Änderung
  3 Abgang

- Nach der Anzeige des Menüs ist die Eingabe einer Ziffer möglich (1–3). Sie können auch mit der Maus auf dem gewünschten Menüpunkt klicken.
- Mit der *Case Call*-Struktur wird einerseits die Richtigkeit des zuvor eingegebenen Wertes überprüft und andererseits die zugehörige Unterroutine aufgerufen, wenn das zuvor eingegebene Zeichen den Bedingungen entspricht. Zugelassen sind in diesem Beispiel nur die Ziffern 1, 2 oder 3. Trifft eine der Bedingungen zu, wird zu dem zugehörigen Label verzweigt. In unserem Beispiel werden je nach Eingabe vier unterschiedliche Label angesprungen (drei Label über die Menüauswahl, der vierte Label ist der *Default Label*).
- Trifft keine der Bedingungen zu, wird zu der *Default* -Label ausgeführt.

Der Übersichtlichkeit wegen wurden in den Unterroutinen keine Befehle angezeigt. Die Reihenfolge der Unterroutinen ist hier unerheblich. Die Unterroutinen, die aufgrund zutreffender Bedingungen aufgerufen werden, müssen immer mit einem *Return*-Befehl enden. Wenn Sie dies nicht beachten, treten unweigerlich Fehler auf, weil dann alle Befehle bis zum nächsten *Return* ausgeführt werden. Bei den ersten beiden Unterroutinen des folgenden Beispiels wurden die *Return*-Befehle weggelassen:

```
2.      Label               (Neuaufnahme)
        :::::               (kein Return)
        Label               (Änderung)
        ::::::              (kein Return)
        Label               (Löschung)
        ::::::
        Return
```

Wird 1 ausgewählt — werden alle Befehle zwischen *Label*(*Neuaufnahme*) und dem zugehörigen *Return* ausgeführt, sofern innerhalb der nachfolgenden Unterroutinen nicht verschiedene Befehle übersprungen werden.

Wird 2 ausgewählt — werden alle Befehle zwischen *Label*(*Änderung*) und dem zugehörigen *Return* ausgeführt, sofern innerhalb der nachfolgenden Unterroutinen nicht verschiedene Befehle übersprungen werden.

Wird 3 ausgewählt — wird die Unterroutine *Label*(*Löschung*) ordnungsgemäß ausgeführt.

Weitere Hinweise siehe: Case, CaseOf, If, Label

6.18 CaseOf

Befehlsform: **CaseOf** (*Ausdruck;Ausdruck;...*)

Über diesen Befehl wird in Verbindung mit den unter *Switch* angegebenen Variablen oder Konstanten entschieden, welche Anweisungen ausgeführt werden sollen. Die Angabe in *Switch* wird mit den Parametern der *CaseOf*-Befehle verglichen. Der Vergleich erfolgt in der angegebenen Reihenfolge der *CaseOf*-Befehle. Es sind beliebig viele *CaseOf*-Befehle erlaubt. Trifft die Angabe bei einer *CaseOf*-Bedingung zu, werden die Anweisungen ausgeführt, die dem betreffenden *CaseOf*-Befehl folgen. Die *Switch*-Befehlsfolge wird verlassen, wenn entweder keine der Bedingungen zutrifft oder wenn die der zuerst gefundenen *CaseOf*-Anweisung folgenden Anweisungen ausgeführt wurden. In letzterem Fall werden die restlichen *CaseOf*-Befehle ignoriert, auch dann, wenn auch noch weitere zutreffen würden (Ausnahme: siehe Befehl *Continue*). Der *CaseOf*-Befehl tritt immer in Verbindung mit den Befehlen *Switch* und *Continue* auf, wobei *Continue* nur bei Bedarf verwendet wird. In Verbindung mit dem Befehl *Default* können Sie festlegen, wie weitergearbeitet werden soll, wenn keine der Bedingungen zutrifft.

Parameter

Ausdruck Variablenname(n) oder Konstante(n). Werden mehrere Konstanten oder Variable angegeben, müssen diese durch ein Semikolon getrennt werden (entspricht einer Oder-Bedingung). Zwischen Groß- und Kleinbuchstaben wird unterschieden. Beachten Sie bitte, das der letzte *Ausdruck* mit einem Doppelpunkt enden muß.

Beispiel

```
1         Application          (A1; "WordPerfect"; Default; "DE")
          Label                (Anfang)
          Taste1=
          Taste2=              0
          Taste3=              0
          DialogDefine         (100; 50; 50; 200; 100; OK! | Percent!;
                               "Beispielfür Switch/CaseOf")
          DialogAddRadioButton (100; 1000; 10; 10; 50; 15;"&Äpfel"; Taste1)
          DialogAddRadioButton (100; 1001; 10; 30; 50; 15; "&Bananen"; Taste2)
          DialogAddRadioButton (100; 1002; 10; 50; 50; 15; "&Orangen"; Taste3)
          DialogDisplay        (100; 1)
          Switch               (1)
            CaseOf             Taste1:   Prompt("Ergebnis";"Äpfel";;;)
            CaseOf             Taste2:   Prompt("Ergebnis";"Bananen";;;)
            CaseOf             Taste3:   Prompt("Ergebnis";"Orangen";;;)
          EndSwitch
          Wait                 (30)
          EndPrompt
          DialogDestroy        (100)
          Go                   (Anfang)
```

In diesem Beispiel wird ein Dialogfeld mit drei Optionsfeldern angezeigt. Über *Switch* in Verbindung mit *CaseOf* wird geprüft, welches der Optionsfelder aktiviert wurde. Beim Aktivieren eines Optionsfeldes wird der Inhalt der zugehörigen Variablen (Taste1, Taste2 oder Taste3) auf 1 gesetzt. Unter Switch wird darum zum Vergleich die Konstante 1 angegeben. Aufgrund der *CaseOf*-Befehle wird jetzt geprüft, welche der Variablen den Wert 1 enthält. Danach wird der zugehörige Befehl ausgeführt und die *Switch*-Struktur verlassen.

Achtung: Zwischen CaseOf und dem nachfolgenden Parameter darf zum Einrücken nicht F7 verwendet werden, sonst werden Kompilier-Fehler angezeigt, obwohl die Syntax stimmt. Verwenden Sie zum Einrücken ⇆.

```
2       Switch              (Sortierung)
          CaseOf            Eingabe:            Call (Verarbeitung1)
          CaseOf            "B";"b":            Call (Verarbeitung2)
                                                Call (Verarbeitung21)
                                                Type(AusgabeText)
          CaseOf            "C";"c";"D";"d":    Call (Verarbeitung3)
          Default:                              Call (Prüfen)
                            Go (Fehler)
        EndSwitch
```

Sortierung steht hier für eine einstellige Variable, die einen Groß- oder Kleinbuchstaben enthalten kann. Über *CaseOf* wird geprüft, welcher Buchstabe in *Sortierung* gespeichert ist. Konstanten müssen auch hier in Hochkommata angegeben werden, numerische Werte dürfen nicht in Hochkommata angegeben werden. *Eingabe* ist eine Variable. Trifft keine der drei Bedingungen zu, verzweigt das Makro über den Befehl *Default* zu einer Fehlerroutine. Nach der Rückkehr von einer durch *Call* aufgerufenen Unterroutine werden auch weitere vorhandene Befehle, die dem auslösenden *CaseOf*-Befehl folgen, ausgeführt (siehe zweite CaseOf-Anweisung), ansonsten wird mit dem Befehl weitergearbeitet, der *EndSwitch* folgt.

```
3       Switch              (Sortierung)
          CaseOf            "B";"b":            If        (Jahr = 1994)
                                                          Call (Verarbeitung)
                                                Else
                                                          Call (Umbuchung)
                                                EndIf
          CaseOf            "C";"c";"D";"d":    Call(Verarbeitung3)
          Default:          Go (Fehler)
        EndSwitch
```

Entspricht weitgehend dem Beispiel 2. Hier folgt dem *CaseOf* eine weitere Bedingung in Form einer *If*-Abfrage.

Weitere Hinweise siehe: Case, Case Call, Continue, Default, EndSwitch, Switch

6.19 Chain

Befehlsform: **Chain**(*Makroname*)

Ausführung (Verkettung) des angegebenen Makros, sobald das aktuelle Makro komplett abgearbeitet wurde. Der Aufruf richtet sich immer an ein eigenständiges Makro. Die Kontrolle wird an das aufgerufene Makro übergeben. Der Befehl kann, wenn erforderlich, eine vollständige Pfadangabe enthalten, wie z. B.

Chain("C:\WPWIN7\SONDMAK\VERKBED.WCM").

Die Dateinamen-Erweiterung muß angegeben werden. Der Befehl kann an einer beliebigen Stelle innerhalb eines Makros eingefügt werden. Der Übersichtlichkeit wegen sollten Sie ihn aber entweder am Anfang oder am Ende eines Makros definieren.

Sie können nur ein Makro je Makroebene über *Chain* verketten. Ist mehr als ein *Chain*-Befehl in einem Makro vorhanden, wird immer der letzte Befehl ausgeführt, wenn das Makro, das den *Chain*-Befehl enthält, beendet wird. Die aufgerufenen Makros müssen kompiliert sein, sonst erhalten Sie eine Fehlermeldung. Ein ausgeführter *Quit*-Befehl des aktuellen Makros bringt einen *Chain*-Befehl nicht zur Ausführung.

Parameter

Makroname Name des aufzurufenden Makros, ggf. unter Angabe des Laufwerksbuchstabens und Ordnernamens. Sie können auch den Namen einer Variablen angeben, die den Makronamen enthält.

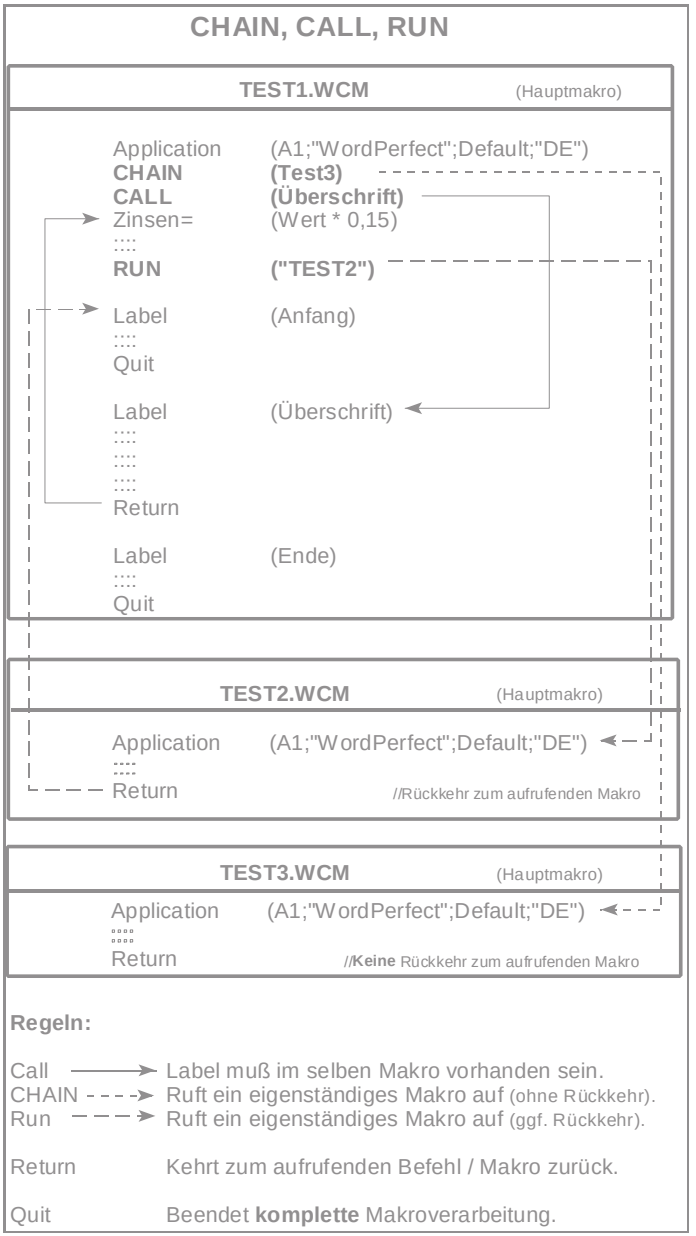
Beispiel

```
1   MakroDru= "C:\WPWIN7\MAKROS\DRUCK.WCM"
    Chain (MakroDru)

2   Chain ("C:\WPWIN7\MAKROS\DRUCK.WCM")
```

Nach Beendigung des aktuellen Makros wird das Makro DRUCK aufgerufen.

Weitere Hinweise siehe: Call, Run



Regeln:

Call ——→ Label muß im selben Makro vorhanden sein.
CHAIN - - -→ Ruft ein eigenständiges Makro auf (ohne Rückkehr).
Run — — —→ Ruft ein eigenständiges Makro auf (ggf. Rückkehr).

Return Kehrt zum aufrufenden Befehl / Makro zurück.

Quit Beendet **komplette** Makroverarbeitung.

6.20 CharLen

Befehlsform:	**Variable=CharLen**(*Ausdruck*) **CharLen**(*Variable;Ausdruck*)	(WPWin 5.2)

Prüfen Sie mit diesem Befehl die Anzahl der Zeichen eines eingegebenen Textes, einer Konstanten oder einer Dezimalzahl einschließlich der Steuerzeichen. Längenprüfungen sind dann wichtig, wenn z. B. verschiedene Datenfelder oder Variable nur eine bestimmte Anzahl von Zeichen enthalten dürfen. Leerzeichen werden mitgezählt. Das ermittelte Ergebnis der Längenberechnung wird als numerischer Wert in *Variable* gespeichert. Variante 1 kann auch direkt in Verbindung mit Befehlen wie *Assign*, *If*, *Type* usw. verwendet werden (siehe Kapitel »5.17 Änderungen gegenüber WPWin 5.1/5.2«).

Parameter

Ausdruck Beliebiger Variablenname bzw. Ausdruck, für den die Länge ermittelt werden soll.

Variable Beliebiger Variablenname zum Speichern der ermittelten Länge.

Beispiel

1

```
Application(A1;"WordPerfect";Default;"DE")
Text1=     "Anzahl in Bytes:  "
Label      (Anfang)
GetString  (Eingabe;"Bitte Text eingeben:";"Befehl  ""CharLen""
           testen")
Länge=     CharLen(Eingabe)
NumStr     (Anzeige;0;Länge)
Meldung=   Text1+Anzeige
Prompt     ("Befehl  ""CharLen""  testen";Meldung;;;)
Pause
EndPrompt
Go         (Anfang)
```

In diesem Beispiel wird die Länge eines über *GetString* eingegebenen Textes ermittelt, in der Variablen *Länge* gespeichert und danach am Bildschirm angezeigt.

2 *CharLen*(Länge;Eingabe)

Auch diese Variante des Befehls ist möglich. Hier wird die Anzahl der Zeichen der Variablen *Eingabe* in dem Feld *Länge* gespeichert.

Wenn Sie die Anzahl der Zeichen eines Ausdrucks oder einer Variablen ohne Steuerzeichen ermitteln möchten, müssen Sie den Befehl *StrLen* verwenden.

Weitere Hinweise siehe: CharPos, StrLen, SubChar

6.21 CharPos

Befehlsform: **Variable=CharPos**(*SuchenIn;Suchen*)
CharPos(*Variable;Suchen;SuchenIn*) (WPWin 5.2)

Mit diesem Befehl können Sie prüfen, ob der Inhalt von *Suchen* in *SuchenIn* enthalten ist. Wenn ja, wird in einer Variablen die Position, ab der der Ausdruck beginnt, in Form eines numerischen Wertes angegeben. Steuerzeichen und Leerzeichen werden mitgezählt. Variante 1 kann auch direkt in Verbindung mit Befehlen wie *Assign*, *If*, *Type* usw. verwendet werden (siehe Kapitel »5.17 Änderungen gegenüber WPWin 5.1/5.2«).

Parameter

Variable Beliebiger Variablenname zum Speichern der ermittelten Position.

Suchen Zu suchender Text (String). Hier kann es sich um eine Konstante oder um eine Variable handeln.

SuchenIn Text (String), in dem gesucht wird. Hier kann es sich um eine Konstante oder um eine Variable handeln.

Beispiel

```
1.      Text1=              "Text beginnt ab Position:  "
        Label               (Anfang)
        GetString           (Suchen;"Bitte den zu suchenden Text eingeben:"!;
                            "Befehl  ""CharPos""  testen")
        GetString           (Suchenin;"Zu suchen in:!";"Befehl  !""CharPos""  testen")
        Beginn=             CharPos(SuchenIn;Suchen)
        NumStr              (Anzeige;0;Beginn)
        Meldung=            Text1+Anzeige
        Prompt              ("Befehl  ""CharPos""  testen";Meldung;;;)
        Pause
        EndPrompt
        Go                  (Anfang)
```

In diesem Beispiel wird die Länge eines über *GetString* eingegebenen Textes ermittelt und danach am Bildschirm angezeigt.

2. Beginn= CharPos(»WordPerfect für Windows«;"Windows«)

   Ist *Windows* in *WordPerfect für Windows* enthalten, wird in der Variablen *Beginn* der Wert 17 gespeichert. Ist *Suchen* nicht in *SuchenIn* enthalten, wird der Variablen *Beginn* der Wert 0 zugewiesen.

3. Beginn= CharPos(»WordPerfect für Windows«;Feld1)

   Ist der Inhalt der Variablen *Feld1* in *WordPerfect für Windows* enthalten, wird in der Variablen *Beginn* der Wert gespeichert, bei dem der Begriff beginnt.

4. Beginn= CharPos(Feld2;Feld1)

Ist der Inhalt der Variablen *Feld1* in der Variablen *Feld2* enthalten, wird in der Variablen *Beginn* der Wert gespeichert, bei der der Begriff beginnt.

Wenn Sie den Beginn bei einer Position ohne Berücksichtigung der Steuerzeichen ermitteln möchten, müssen Sie den Befehl *StrPos* verwenden.

Weitere Hinweise siehe: CharLen, StrPos, SubChar

6.22 Coach-Befehle

Die Hilfefunktion [*?, Hilfethemen, Zeig mir wie, Hilfestellung*] ist besonders für Anfänger von Vorteil, die selten benötigte Funktionen noch nicht so richtig beherrschen. Nach der Auswahl dieses Menüpunkts wird ein Dialogfeld mit den zur Verfügung stehenden Themen angezeigt. Für jede vorhandene Trainer-Datei (.WCH-Datei) wird nach dem Aktivieren des Optionsfeldes [*Hilfestellung*] ein Eintrag in der Liste angezeigt. Doppelklicken Sie auf dem gewünschten Eintrag. WordPerfect spielt jetzt den Lotsen für Sie. Folgen Sie den angezeigten Informationen, um die ausgewählte Funktion auszuführen bzw. um diese zu erlernen. Dabei gibt es zwei Möglichkeiten. Einerseits kann WordPerfect die erforderlichen Schritte für Sie ausführen, andererseits können Sie die Schritte selbst ausführen, und WordPerfect kontrolliert Ihre Vorgehensweise, ggf. werden Hinweise zur Bedienung angezeigt. Zu dem markierten Eintrag erscheint unterhalb der Liste eine kurze Information. Ist kein Hilfetext vorhanden, wird der Dateiname der betreffenden Coach-Datei angezeigt.

Die Lektionen zeigen Dialogfelder an (Prompting), auf die Sie reagieren müssen, sowie Hilfetexte in Verbindung mit Tips und Tricks. Während der Ausführung kann WordPerfect für Sie die erforderlichen Funktionen starten (Animate), Daten eingeben und Mausklicks simulieren. Wird eine bestimmte Eingabe oder ein bestimmter Tastendruck erwartet (Filtering), läuft das Makro nur dann weiter, wenn Sie entsprechend reagiert haben (geprüft werden hierbei Mausklicks, Tastatureingaben oder Befehle). Andernfalls erfolgt ein Fehlerhinweis. Über die zur Verfügung stehenden Coach-Befehle können Sie in Verbindung mit Makros weitere Trainerdateien erstellen. Der nachfolgenden Kurzanweisung können Sie entnehmen, wie solche Trainer-Dateien erstellt werden könnten:

1. Tastatureingabe und Mausbedienung unterbrechen, bis sie für die Benutzung freigegeben werden.
2. Filter definieren. Der Übersichtlichkeit wegen sollten alle Filter am Makroanfang definiert werden.
3. Anzeigen von Hinweisen, damit der Benutzer die Anwendung bedienen kann.
4. Filter aktivieren.
5. Wenn Benutzereingaben erforderlich sind, diese entgegennehmen, prüfen, entsprechend reagieren oder bei Fehlbedienung einen Fehlerhinweis anzeigen.
6. Bei Bedarf Hinweise an den Benutzer ausgeben, um ihm bei seiner Arbeit behilflich zu sein.
7. Nach Beendigung der Anwendung Filter deaktivieren.

8. Filter im Speicher löschen.
9. Tastatureingabe und Mausbedienung wieder aktivieren und Anwendung beenden.
10. Speichern Sie die Datei in dem WordPerfect-Ordner Makros mit der Dateinamenerweiterung .WCH.

Die Schritte 2 – 8 sind so oft zu wiederholen, wie in der Anwendung erforderlich. Die Anzahl und die Art der Befehle hängt von dem Umfang der betreffenden Anwendung ab.

Hinweis: Da diese Befehle von den wenigsten Lesern benötigt werden, wurde wegen begrenzten Platzes in diesem Buch auf eine detaillierte Beschreibung zugunsten wichtigerer Befehle verzichtet. Sollten Sie diese Befehle benötigten, schauen Sie wegen deren Handhabung bitte in dem Ordner C:\COREL \OFFICE7\SHARED\HELP7 in den Dateien .WCH nach. Das sind die von WordPerfect erstellten Trainer-Dateien, die alle diese Befehle enthalten. Nutzen Sie diese Beispiele als Grundlage für die Erstellung eigener Trainer-Dateien. Die Befehle selbst finden Sie in dem Makrobefehlsmanager unter [*Typ, PerfectScript-DE*].

6.23 Condition

Befehlsform: *Variable*=**Condition**(*Bedingung*;*Status*)

Über diesen Befehl wird festgelegt, wie ein Makro auf die Befehle *Cancel*, *Error*, *NotFound* oder auf benutzerdefinierte Unterbrechungen reagieren soll.

Parameter

Variable Speicherung des Statuswertes des zuletzt ausgeführten *Condition*-Befehls.

0	Status der angegebenen Bedingung war Off!
1	Status der angegebenen Bedingung war On!

Bedingung Angabe, welche Bedingung mit welchem Status versehen werden soll:

CancelCondition!	Abbruchbedingung
ErrorCondition!	Fehlerbedingung
NotFoundCondition!	Nicht-Gefunden-Bedingung
ExitCondition!	Ende
UserDefinedCondition!	Benutzerdefinierte Bedingung (siehe »OnCondition«).

Status Setzen der gewünschten Bedingung. Standardmäßig wird *On!* vorgegeben. Beim Weglassen wird der aktuelle Status ohne Änderung zurückgegeben.

Off!	Die angegebene Bedingung wird ignoriert.
On!	Makroausführung wird abgebrochen, wenn OnCondition nicht verwendet wird, andernfalls wird die Makroausführung bei dem unter OnCondition angegebenen Label fortsetzt.

Ein Beispiel finden Sie unter OnCondition.

Weitere Hinweise siehe: OnCondition, OnCondition Call

6.24 Constant

Befehlsform: **Constant**(*Variable;Variable;...*)

Hier können Variable definiert werden, deren Inhalt während der Makroausführung nicht geändert werden kann. Benutzen Sie diesen Befehl zum Definieren von Variablen, die z. B. Dateinamen oder Mehrwert-Steuersätze enthalten, um diese Inhalte während der Makroausführung nicht versehentlich zu überschreiben.

Parameter

Variable Festlegung der Variablen, die während der Makroausführung nicht geändert werden dürfen. Die Variable können einzeln oder in einem Befehl definiert werden.

Beispiel

Die Inhalte der in Klammern angegebenen Variablen können während der Makroausführung nicht verändert werden.

```
Constant        (Dateiname="c:\autoexec.bat";Schleife=1;MwSt=15;Ort="Darmstadt")
Constant        (MwSt=15)
Constant        (Plz="64293")
Constant        Ort="Darmstadt"
```

6.25 Continue

Befehlsform: *Continue*

Über diesen Befehl wird in Verbindung mit *Switch* und *CaseOf* die Makro-Ausführung gesteuert. Nach der Ausführung eines *CaseOf*-Befehls, dem ein *Continue*-Befehl folgt, wird mit den Anweisungen des dem *Continue*-Befehls unmittelbar folgenden *CaseOf*-Befehls fortgefahren, gleichgültig, ob dessen Bedingung zutrifft oder nicht.

Beispiel

```
Application          (A1; "WordPerfect"; Default; "DE")
Label                (Anfang)
Feld1=               0
Feld2=               0
Feld3=               0
Feld4=               0
DialogDefine         (100; 50; 50; 200; 110; OK! | Percent!; "Beispiel für CONTINUE")
DialogAddCheckBox    (100; 2000; 10; 10; 75; 15; "Kontrollfeld 1"; Feld1)
DialogAddCheckBox    (100; 2001; 10; 30; 75; 15; "Kontrollfeld 2"; Feld2)
DialogAddCheckBox    (100; 2002; 10; 50; 75; 15; "Kontrollfeld 3"; Feld3)
DialogAddCheckBox    (100; 2003; 10; 70; 75; 15; "Kontrollfeld 4"; Feld4)
DialogDisplay        (100; 2000)
DialogDestroy        (100)
Meldung=             ""
Switch               (1)
   CaseOf            Feld1:  Meldung="Kontrollfeld 1 wurde gewählt.  "
   CaseOf            Feld2:  Meldung=Meldung+"Kontrollfeld 2 wurde gewählt. "
     Continue
   CaseOf            Feld3:  Meldung=Meldung+"Kontrollfeld 3 wurde gewählt."
   CaseOf            Feld4:  Meldung=Meldung+"Kontrollfeld 4 wurde gewählt."
EndSwitch
Prompt               ("Ergebnis";Meldung;;;)
Wait                 (30)
EndPrompt
Go                   (Anfang)
```

In diesem Beispiel werden aufgrund aktivierter/deaktivierter Kontrollfelder bestimmte Meldungen angezeigt (dient nur zum Test!). Die Meldung für *Feld3* wird nur angezeigt, wenn dieses Feld aktiviert wurde oder wenn *Feld2* aktiviert wurde. Auch wenn alle Kontrollfelder aktiviert wurden, wird immer nur die Meldung für *Feld1* angezeigt, weil die erste *CaseOf*-Bedingung bereits zutrifft.

Weitere Hinweise siehe: CaseOf, Default, Switch

6.26 CToN

Befehlsform: *Variable*=**CToN**(*Zeichen*)

Konvertieren des angegebenen Zeichens in einen numerischen Wert (**C**haracter **To N**umber). Mit Hilfe dieses Befehls können Sie den Zeichensatz und die Zeichennummer des angegebenen Zeichens innerhalb des betreffenden WordPerfect-Zeichensatzes ermitteln. WordPerfect bietet Ihnen 15 verschiedene Zeichensätze an. Der errechnete Wert besteht aus einer Dezimalzahl. Wenn Sie z. B. die Taste a drücken, wird der Wert 0,97 angezeigt, d. h. das kleine a ist das Zeichen mit der Nummer 97 im Zeichensatz 0.

Parameter

Variable Enthält das konvertierte Zeichen.

Zeichen Enthält das zu konvertierende Zeichen.

Beispiel

```
1         GetString            (Eingabe;"Bitte Zeichen eingeben.";"Befehl ""CToN""
                                testen";1)
          Rechnen=             Eingabe
2         Tastenwert=          CtoN(Rechnen)
3         Zeichensatznummer=   (Tastenwert  DIV  256)
4         Zeichennummer=       (Tastenwert  MOD 256)
5         Meldung=             "Ermitteltes Zeichen = "+Eingabe+", Tastenwert:
                               "+Tastenwert+", Zeichensatznummer: "+Zeichensatznummer+",
                               Zeichennummer: "+Zeichennummer
6         Prompt               ("Befehl  ""CToN""  testen";Meldung;;;)
          Wait                 (40)
          EndPrompt
```

In dem gezeigten Beispiel soll für ein eingegebenes Zeichen dessen Zeichensatz- und Zeichennummer ermittelt werden.

1 Anzeige einer Dialogbox, Übernahme des eingetippten Zeichens in die Variable Eingabe.

2 Ermitteln des Tastenwertes.

3 Ermitteln der Zeichensatznummer.

4 Ermitteln der Zeichennummer innerhalb eines Zeichensatzes.

5 Meldung generieren.

6 Meldung anzeigen.

Sie finden ein Beispiel für diesen Befehl auf der Diskette unter dem Dateinamen CTON.WCM.

Weitere Hinweise siehe: NtoC

6.27 Date-Befehle

Mit den nachfolgend beschriebenen Befehlen können Sie Datumsangaben in verschiedenen Variationen dem System entnehmen, in einer Variablen speichern und in dem Makro weiterverwenden.

DateAndTime Variable=DateAndTime(Tag;Monat;Jahr;Stunde;Minute;Sekunde;Hundertstelskunde)

Die in Klammern angegebenen Parameter liefern numerische Werte.

DateDay Variable=DateDay(TagZahl)

Ermitteln des Tages als numerischer Wert.

DateMonth Variable=DateMonth(MonatZahl)

Ermittelt den Monat als numerischen Wert (1 = Januar, 2 = Februar usw.).

DateMonthName Variable=DateMonthName(MonatName;Typ)

Ermittelt den Monatsnamen. Unter »Typ« können Sie folgende Parameter angeben:

Short!	Monatsname wird mit drei Buchstaben abgekürzt.
Long!	Monatsname wird ausgeschrieben.

DateString Variable=DateString(Datum;Type;Format)

Ermitteln des Datums als String.

Datum	Numerischer Wert.
Type	
Short!	Datum wird in Ziffern angezeigt.
Long!	Datum wird ausgeschrieben.
Format	Festlegung des Datumsformats. Wird der Parameter weggelassen, wird das aktuelle Systemformat verwendet.

Unter »Format« können folgende Parameter verwendet werden. Achten Sie auf die Groß-/Kleinschreibung, da falsch geschriebene Parameter als Text angezeigt werden.

Tagesnummer	d	Zahlen ohne führende Null bei einstelligen Werten.
	dd	Zahlen mit führender Null bei einstelligen Werten.
Wochentag	ddd	Tagesname in dreistelliger Abkürzung.
	dddd	Tagesname ausgeschrieben.
Monat	M	Zahlen ohne führende Null bei einstelligen Werten.
	MM	Zahlen mit führender Null bei einstelligen Werten.

	MMM	Monatsname in dreistelliger Abkürzung.
	MMMM	Monatsname ausgeschrieben.
Jahr	y	letzte Stelle der Jahreszahl
	yy	Die beiden letzten Stellen der Jahreszahl
	yyyy	Jahreszahl vierstellig.
Period/Era	gg	Wird ignoriert, wenn Era-String nicht verwendet wurde.

DateWeekday Variable=DateWeekday(TagName)

Der Wochentag wird als Ziffer ermittelt, Sonntag = 0, Montag = 1 usw.

DateWeekdayName Variable=DateWeekdayName(TagName;Typ)

Der Wochentag wird als Name ermittelt.

Short!	Tagesname wird mit drei Buchstaben abgekürzt.
Long!	Tagesname wird ausgeschrieben.

Beispiel:

Datum=	DateString(;;"d.MM.yyyy")
Ergibt:	29.07.1996
Datum=	DateString(;;"dddd, d.MM.yyyy")
Ergibt:	Montag, 29.07.1996
Datum=	DateString(;;"dddd, d.MMMM.yyyy")
Ergibt:	Montag, 29.Juli.1996
DatumDef=	DateAndTime(28;10;1996)
Datum=	DateString(DatumDef;;"dddd, d.MMMM.yyyy")
Ergibt:	Montag, 28.Oktober.1996

Weitere Hinweise siehe: Time-Befehle

6.28 DDEExecute

Befehlsform: **DDEExecute**(*PgmKennung;Befehl*)

DDEExecute übergibt einen Befehl zur Ausführung an das unter *PgmKennung* angegebene Programm. Bevor dieser Befehl ausgeführt werden kann, muß über *DDEInitiate* eine Verbindung aufgebaut worden sein. Mit der Bedienung des betreffenden Programms sollten Sie vertraut sein, um die gewünschten Befehle in der richtigen Form eingeben zu können.

Beispiel

PgmKennung Hier ist derselbe Name anzugeben, der bei dem zugehörigen *DDEInitiate* unter *PgmKennung* für das betreffende Programm verwendet wurde.

Befehl Befehl, der in dem angegebenen Programm ausgeführt werden soll. Der Befehl kann als Variable oder als Konstante übergeben werden. Konstanten müssen durch Hochkommata eingeschlossen sein. Treten innerhalb einer Konstanten weitere Hochkommata auf, müssen diese doppelt angegeben werden (siehe Beispiel).

Der in WordPerfect angegebene Befehl "[Öffnen(""C:\VERKAUF.XLS"")]" wird in dem betreffenden Programm als [Öffnen("C:\VERKAUF.XLS")] interpretiert. Schlagen Sie wegen der Verwendung der Befehle bitte im Handbuch oder der Online-Hilfe des betreffenden Programms nach, da jedes Programm eigene Befehle verwendet, die in der Handhabung unterschiedlich sind.

Beispiel

```
Application                (A1;"WordPerfect";Default;"DE")
PgmKenn=                   DDEInitiate("Excel";"Datei")
DDEExecute                 (PgmKenn;"[Öffnen(""C:\VERKAUF.XLS"")]")
DDETerminate               (PgmKenn)
```

Weitere Hinweise siehe: DDEExecuteExt

6.29 DDEExecuteExt

Befehlsform: *Ergebnis*=**DDEExecuteExt**(*PgmKennung;Befehl;Timeout;Label*)

DDEExecuteExt arbeitet ähnlich wie *DDEExecute*, wobei erweiterte DDE-Zugriffe auf andere Programme möglich sind. Wurde eine Verbindung mit anderen Programmen aufgebaut, versucht *DDEExecuteExt* den Befehl in dem angegebenen Programm auszuführen. Die Variable *Ergebnis* enthält das Ergebnis der Ausführung. War die Ausführung erfolgreich, wird ein numerischer Werte ungleich 0 zurückgegeben. *PgmKennung* enthält eine Ganzzahl, die von einer früheren Ausführung in Verbindung mit *DDEInitiate* zurückgegeben wurde.

Wenn der Wert von *Timeout* kleiner oder gleich 0 ist, wird der Befehl parallel ausgeführt und *DDEExecuteExt* kehrt sofort zurück. Für Befehle, die nicht parallel ausgeführt werden sollen, kann über *Label* eine Unterroutine aufgerufen werden, wenn der Befehl ausgeführt wurde. Ist kein Label angegeben, wird die Meldung, daß der Befehl ausgeführt wurde, ignoriert. Ein schwerwiegender Fehler kann auftreten, wenn *PgmKennung* keinen gültigen Wert enthält, wenn der Befehlsparameter keinen gültigen Ausdruck enthält oder wenn ein ungültiger Label angegeben wurde.

Parameter

Ergebnis Ergebnis der ausgeführten Transaktion. Die Ausführung war dann erfolgreich, wenn ein anderer Wert als 0 zurückgegeben wird.

PgmKennung Enthält eine Ganzzahl, die von einer früheren Ausführung von *DDEInitiate* zurückgegeben wurde.

Befehl Befehl, der in dem angegebenen Programm ausgeführt werden soll. Der Befehl kann als Variable oder als Konstante übergeben werden. Konstanten müssen durch Hochkommata eingeschlossen sein. Treten innerhalb einer Konstanten weitere Hochkommata auf, müssen diese doppelt angegeben werden.

Timeout Numerischer Wert in Millisekunden. Wartezeit bis das angegebene Programm sich meldet, um den angegebenen Befehl auszuführen.

Label Angabe eines Labels einer Unterroutine, die nach der Ausführung des Befehls aufgerufen werden soll.

Beispiel

```
Ergebnis=DDEExecuteExt(PgmKenn;"[Öffnen(""C:\EXCEL\STATISTIK.XLS"")]";0; Verarbeiten)
```

Beachten Sie bitte auch das WordPerfect-Beispiel im Online-Makrohandbuch unter *DDEExecuteExt*.

Weitere Hinweise siehe: DDEExecute

6.30 DDEInitiate

Befehlsform: *PgmKennung*=**DDEInitiate**(*ServiceName;MenüPunkt*)

Über diesen Befehl können Sie eine Verbindung mit einem DDE-kompatiblen Programm aufbauen. Nach dem Aufbau der Verbindung können dann über *DDEPoke* Daten übergeben oder über *DDEExecute* Befehle an das betreffende Programm übergeben werden. Weiterhin können umgekehrt z. B. über *DDERequest* Daten von diesem Programm übernommen werden. Über *DDETerminate* können bestimmte oder über *DDETerminateAll* alle Verbindungen gelöst werden.

Parameter

PgmKennung Jede Verbindung muß mit einem eindeutigen Namen gekennzeichnet werden. Das kann ein String sein, der allerdings keine Leerstellen enthalten darf. Wurde eine Verbindung erfolgreich hergestellt, wird in dieser Variablen eine Meldung gespeichert. Konnte keine Verbindung hergestellt werden, wird 0 zurückgegeben.

ServiceName Dieser alphanumerische Ausdruck muß den Namen des Programms enthalten zu dem eine Verbindung aufgebaut werden soll. Die Dateinamen-Erweiterung (.COM, EXE usw.) darf nicht angegeben werden (z. B. EXCEL, PBRUSH, WRITE).

MenüPunkt Angabe eines Menüpunkts des betreffenden Programms (programmspezifisch, siehe Dokumentation des betreffenden Programms). Erlaubt ist eine WordPerfect-Variable oder ein String.

Beispiel

```
Application          (A1;"WordPerfect";Default;"DE")
PgmKenn=             DDEInitiate("FAXMNG";"Transmit")
GetString            (FAXNum;"Faxnummer eingeben";"WinFAX";20)
DDEPoke              (PgmKenn;"FAX Nummer";FAXNum)
DDETerminate         (PgmKenn)
DruckenDokument      ()
```

In diesem Beispiel (es wurde dem Online-Handbuch entnommen) wird zu dem Programm *FAXMNG* eine DDE-Verbindung hergestellt. Durch *GetString* wird die Eingabe einer maximal 20stelligen Faxnummer verlangt. Der Befehl *DDEPoke* übergibt die eingegebene Faxnummer *FAXNum* dem Menüpunkt *Transmit* des Programms *FAXMNG*.

Weitere Hinweise siehe: DDEExecute, DDEPoke

6.31 DDEPoke

Befehlsform: **DDEPoke**(*PgmKennung;Variable;Daten*)

Über diesen Befehl können Sie Daten in eine Variable des betreffenden Programms übergeben, vorausgesetzt, daß zuvor eine Verbindung über *DDEInitiate* mit dem Programm hergestellt wurde.

Parameter

PgmKennung Hier ist derselbe Name anzugeben, der bei dem zugehörigen *DDEInitiate* unter *PgmKennung* für das betreffende Programm verwendet wurde.

Variable Name einer Variablen in dem unter *PgmKennung* angegebenen Programm. Hier kann ein Variablenname oder eine Konstante verwendet werden. Schlagen Sie wegen der Verwendung der Variablen bitte im Handbuch oder der Online-Hilfe des betreffenden Programms nach, da jedes Programm eigene Variable verwendet, die in der Handhabung unterschiedlich sind.

Daten Zu übergebenden Daten (WordPerfect-Variable oder String).

Beispiel: Siehe unter DDEInitiate.

Weitere Hinweise siehe: DDEInitiate

6.32 DDERequest

Befehlsform: *WPVariable*=**DDERequest**(*PgmKennung;PgmVariable*)

Dieser Befehl prüft den Inhalt einer Variablen eines verknüpften Programms und speichert den Variableninhalt in einer WordPerfect-Variablen. Bevor *DDERequest* verwendet werden kann, muß eine Verbindung über *DDEInitiate* aufgebaut worden sein. Nach der Ausführung von *DDERequest* können Sie über *OnDDEAdvice* **Call** prüfen, ob Veränderungen vorgenommen wurden.

Parameter

WPVariable Name einer WordPerfect-Variablen in der die Veränderung gespeichert wird.

PgmKennung Hier ist derselbe Name anzugeben, der bei dem zugehörigen *DDEInitiate* unter *PgmKennung* für das betreffende Programm verwendet wurde.

PgmVariable Name einer Variablen in dem unter *PgmKennung* angegebenen Programm. Hier kann ein Variablenname oder eine Konstante verwendet werden. Schlagen Sie wegen der Verwendung der Variablen bitte im Handbuch oder der Online-Hilfe des betreffenden Programms nach, da jedes Programm eigene Variable verwendet, die in der Handhabung unterschiedlich sind.

Beispiel

```
Application          (A1;"WordPerfect";Default;"DE")
PgmKenn=             DDEInitiate("Excel";"System")
Übergabe=            DDERequest(PgmKenn;"Bemerkungen")
Type                 ("Daten aus Excel: ")
Type                 (Übergabe)
HardReturn           ()
DDETerminate         (PgmKenn)
```

In diesem Programm (siehe Online-Handbuch) werden aus Excel in die Variable *Übergabe* Daten übergeben, die in WordPerfect durch den Befehl *Type* auf Cursorposition übernommen werden.

Weitere Hinweise siehe: DDEInitiate, OnDDEAdvice Call

6.33 DDETerminate / DDETerminateAll

Befehlsform: **DDETerminate**(*PgmKennung*)
DDETerminateAll

DDETerminate und *DDETerminateAll* beenden eine Verbindung zwischen WordPerfect und anderen Programmen, wobei *DDETerminate* nur das als Parameter angegebene Programm beendet, *DDETerminateAll* dagegen alle Verbindungen.

Parameter

PgmKennung Hier ist derselbe Name anzugeben, der bei dem zugehörigen *DDEInitiate* unter *PgmKennung* für das betreffende Programm verwendet wurde.

Beispiel: Siehe auch *DDEInitiate*.

DETerminate (PgmKenn)
Die angegebene Verbindung wird aufgelöst.

DETerminateAll Alle Verbindungen werden aufgelöst.

Weitere Hinweise siehe: DDEInitiate

6.34 Declare

Befehlsform: **Declare**(*Variable1;Variable2;...*)
Declare(*Array-Name*[*Index*])
Declare(*Array-Name*[*Index1;Index2;...Index10*])

Diesen Befehl können Sie unterschiedlich verwenden. Einerseits können Sie hiermit lokale Variable definieren und andererseits sog. Arrays. Durch Arrays können mehrere Variable desselben Typs unter einem Namen zusammengefaßt werden. Maximal sind zehn Dimensionen und maximal 32.767 Elemente je Dimension möglich. Die maximale Größe ist abhängig von dem RAM des verwendeten Rechners. Jede einzelne Variable wird als Element bezeichnet. Alle Elemente werden mit demselben Namen angesprochen (z. B. *Umsatz*). Die Unterscheidung der einzelnen Elemente findet durch den in eckigen Klammern angegebenen Wert (= Index) statt, wobei dieser immer eine positive Ganzzahl größer als 0 sein muß. Der höchste Indexwert einer Dimension darf nicht größer sein als der unter *Declare* angegebene Maximalwert, ansonsten erfolgt eine Fehlermeldung. Der Index kann eine numerische Konstante sein oder eine Variable, die eine Ganzzahl in dem gültigen Bereich des betreffenden Indexes enthält (zwischen 1 und dem Maximalwert einschließlich). Arrays werden meistens über Schleifen, mehrdimensionale Arrays über geschachtelte Schleifen abgearbeitet, wie z. B. mit den Befehlen *For*, *ForNext*, *Repeat* usw.

Parameter

Variable Beliebige lokale Variable oder die Definition eines Arrays.

Array-Name Beliebiger Name, über den die Elemente angesprochen werden können. Alle Elemente eines Arrays müssen mit demselben Namen versehen sein. Werden mehrere Arrays definiert, müssen diese jeweils unterschiedliche Elementnamen aufweisen. Das jeweilige Element kann während der Makro-Ausführung nur dann angesprochen werden, wenn es Daten enthält.

Index Elemente eines Arrays können nur in Verbindung mit einem Index angesprochen werden, wobei jede Dimension einen eigenen Index benutzt. Erlaubt sind nur numerische Ganzzahlen größer als 0 und kleiner oder gleich dem angegebenen Maximalwert. Der Index ist immer in eckigen Klammern anzugeben. Bei mehrdimensionalen Arrays sind so viel Indizes zu verwenden, wie Dimensionen vorhanden sind. Anstelle der Ganzzahl kann auch eine Variable verwendet werden, deren Inhalt entweder fix definiert oder erst zur Ausführungszeit ermittelt werden kann. In letzterem Fall wird die Größe des Arrays dadurch variabel gehalten. Hierbei darf sich der Declare-Befehl nicht in einer Schleife befinden, da er nur einmal ausgeführt werden darf. Jede weitere Ausführung führt zu einer Fehlermeldung.

Ungültige Indizes oder Elemente, die keine Daten enthalten, führen während der Ausführungszeit des Makros zu einem Fehler, wenn sie angesprochen werden. Überprüfen Sie darum mit geeigneten Makroroutinen, ob die zu verwendenden Indizes auch erlaubt sind. Elemente, die keine Daten enthalten, werden wie nicht definierte Variable behandelt und führen ebenfalls zu einer Fehlermeldung. Um sicher zu gehen, können Sie zusätzlich über eine Makroroutine am Makroanfang alle Elemente eines Arrays entweder mit Leerstellen oder mit Nullen füllen:

```
Declare             Umsatz[20;15]
ForNext             (Z1;1;20)
  ForNext           (Z2;1;15)
    Umsatz[Z1;Z2]=  "  "                Füllen mit Leerstellen.
  // Umsatz[Z1;Z2]= 0                   Füllen mit Nullen.
  EndFor
EndFor
```

In diesem Beispiel wird ein zweidimensionales Array mit Leerstellen gefüllt. Dadurch wird sichergestellt, daß zur Ausführungszeit auch alle Elemente angesprochen werden können, ohne einen Makroabsturz zu verursachen. Soll mit den Elementinhalten gerechnet werden, müssen Sie die betreffenden Elemente mit Nullen füllen.

6.34.1 Eindimensionale Arrays

Eindimensionale Arrays sind vergleichbar mit einer Tabelle, die aus einer Spalte und mehreren Reihen besteht. Jede Reihe (= Feld) wird als ein Element der gedachten Tabelle bezeichnet, und bei 1 beginnend bis *n* durchnumeriert. Einfacher gesagt: Hier handelt es sich um Variable, die Daten einer bestimmten Gruppe speichern, die dieselbe Bedeutung haben, wobei der Datentyp unterschiedlich sein kann. Jedem Element können vor Beginn einer Makroausführung Werte zugeordnet werden (ähnlich dem *Assign*-Befehl). Dies können numerische oder alphanumerische Daten sein (siehe Beispiele 1 und 2). Werden die Inhalte der Elemente erst zur Ausführungszeit ermittelt, braucht bei der Definition des Arrays nur ein Elementname und die maximale Anzahl der Elemente angegeben zu werden (siehe Beispiel 3).

Beispiel 1

```
        Application         (A1; "WordPerfect"; Default; "DE")
        OnCancel            (Abbruch)
1       Declare             (Monate[12])
          Monate(1)=        "Januar"
          Monate(2)=        "Februar"
          Monate(3)=        "März"
          Monate(4)=        "April"
          Monate(5)=        "Mai"
          Monate(6)=        "Juni"
          Monate(7)=        "Juli"
          Monate(8)=        "August"
          Monate(9)=        "September"
          Monate[10]=       "Oktober"
          Monate[11]=       "November"
          Monate[12]=       "Dezember"
2       Label               (Anfang)
3       GetNumber           (Monat;"Bitte Nummer eines Monats eingeben (z. B.
                            5  für den Monat Mai.)";"Befehl ""Declare""
                            testen.")
4       If                  ((Monat<1) or (Monat>12))
          Prompt            ("Fehler";"Geben Sie einen Wert von 1 - 12 ein.";;;)
          Wait              (30)
          EndPrompt
          Go                (Anfang)
        EndIf
5       Meldung=            ("Sie haben den Monat "+Monate[Monat]+
                            ausgewählt.")
6       Prompt              ("Monatsname ermitteln";Meldung;;;)
        Wait                (30)
        EndPrompt
        Go                  (Anfang)
7       Label               (Abbruch)
        Quit
```

In diesem Beispiel wird aufgrund der eingegebenen Monatsnummer der zugehörige Monatsname angezeigt.

1 Unter *Declare* wird der Name des eindimensionalen Arrays festgelegt (hier: Monate). Zusätzlich ist die Anzahl der einzelnen Elemente (= Monate pro Jahr) festzulegen (hier: 12). Achten Sie darauf, daß dieser Wert in eckigen Klammern [ ] angegeben wird.

2 Beginn einer Programm-Schleife.

3 Anzeigen einer Meldung und Eingabe einer Zahl von 1 – 12.

4 Prüfen der eingegebenen Zahl. Da nur zwölf Monate vorhanden sind, darf nur eine Zahl von 1 – 12 eingegeben werden. Alle anderen Zahlen sind als Fehler abzuweisen. Werden Dezimalzahlen eingegeben, so wird für den Index nur der Wert vor dem Dezimalkomma bzw. dem Dezimalpunkt akzeptiert. Wird diese Prüfung nicht durchgeführt, würde das Makro beim Eingeben einer unzulässigen Zahl abbrechen und eine Fehlermeldung anzeigen.

5 Generieren der anzuzeigenden Meldung. Der Monatsname wird aufgrund des Indexes *Monat* aus dem Array *Monate* entnommen. Für die Ermittlung ist nur eine Definition erforderlich. Wenn kein Array verwendet würde, könnte die Abfrage z. B. über *If*-Befehle erfolgen:

```
If (Monat=1) Meldung=("Sie haben den Monat Januar ausgewählt.") Else
If (Monat=2) Meldung=("Sie haben den Monat Februar ausgewählt.") Else
If (Monat=3) Meldung=("Sie haben den Monat März ausgewählt.") Else
::::
If (Monat=12) Meldung=("Sie haben den Monat Dezember ausgewählt.") EndIf .....
Prompt.....
```

Je größer in solchen Fällen ein Array ist, umso mehr Schreibarbeit wäre erforderlich.

Beispiel 2

```
          Application          (A1; "WordPerfect"; Default; "DE")
1         Declare              (Umsatz[12])
             Umsatz(1)=        1000
             Umsatz(2)=        1000
             Umsatz(3)=        1000
             Umsatz(4)=        1000
             Umsatz(5)=        1000
             Umsatz(6)=        1000
             Umsatz(7)=        1000
             Umsatz(8)=        1000
             Umsatz(9)=        1000
             Umsatz[10]=       1000
             Umsatz[11]=       1000
             Umsatz[12]=       1000
2         Summe=               0
3         ForNext              (Monat; 1; 12; 1)
4         Summe=               (Summe+Umsatz[Monat])
5         EndFor
```

In diesem Beispiel sollen die Umsätze der zwölf Monate eines Geschäftsjahrs addiert werden. Das Ergebnis wird in der Variablen *Summe* gespeichert. Hier wurden feste Daten vorgegeben. Die Umsätze der einzelnen Monate könnten z. B. auch über eine Datendatei eingelesen und monatlich kumuliert werden.

1 Unter *Declare* wird der Name des eindimensionalen Arrays festgelegt (hier: Umsatz). Zusätzlich ist die Anzahl der einzelnen Elemente (= Umsatz je Monat) festzulegen (hier: 12). Achten Sie darauf, daß dieser Wert in eckigen Klammern [ ] angegeben wird.

2 Definition der Variablen *Summe*. Diese Variable wird mit 0 (Null) initialisiert (Löschen eines vielleicht vorhandenen Wertes), damit vor Beginn des ersten Durchlaufs keine anderen Werte vorhanden sind.

3 In der *ForNext*-Schleife werden die einzelnen Elemente des betreffenden Arrays, in diesem Fall Umsatz, elementweise addiert. Im ersten Durchlauf wird das erste Element addiert, im zweiten Durchlauf das zweite Element usw. Die Schleife wird solange wiederholt, bis alle zwölf Monate addiert wurden.

4 Dieser Befehl wird je Durchlauf einmal ausgeführt. Vor der ersten Ausführung enthält die Variable *Summe* den Wert 0 (siehe Zeile 2). Nach der Addition des ersten Elements enthält *Summe* den Wert 1000, nach der Addition des zweiten Elements den Wert 2000. Wurden alle Elemente addiert, beträgt der Wert in *Summe* 12000. Das jeweils aktuelle Element wird durch die Variable *Monat* adressiert. Enthält *Monat* den Wert 4, wird dadurch das vierte Element angesprochen.

Ohne die Verwendung von Arrays müßten Sie die Summe wie folgt bilden, wenn die Variablen Januar – Dezember als Speicherfelder verwendet würden:

```
Summe=          (Summe+Januar)
Summe=          (Summe+Februar)
Summe=          (Summe+März)
::::
Summe=          (Summe+Dezember)
```

Hier wären so viele Additionen vorzunehmen, wie zu addierende Variable vorhanden sind. Stellen Sie sich vor, Sie müßten tägliche Summen bilden. Dann wären 365 Additionen erforderlichen, bei der Verwendung von Arrays nur eine innerhalb einer Schleife.

Beispiel 3

```
        Application     (A1;"WordPerfect";Default;"DE")
1       Declare         (Texte[10])
2       ForNext         (Index;1;10)
3          GetString    (Texte[Index];"Bitte beliebigen Text eingeben, max. 20
                        Zeichen:";"Befehl ""Declare"" testen, 10 Eingaben
                        werden verlangt.";20)
        EndFor
4       FileNew         ()
5       ForNext         (Index;1;10)
6          Type         ("Array-Element ("+Index+") enthält: ")
7          Type         (Texte[Index])
           HardReturn   ()
        EndFor
```

In diesem Beispiel wird ein Array mit zehn Elementen definiert, denen zu diesem Zeitpunkt noch keine Inhalte zugewiesen wurden. Über eine *ForNext*-Schleife werden zehn Eingaben verlangt, die in den Elementen 1 – 10 gespeichert werden. Im Anschluß daran werden die eingegebenen Daten gedruckt. Verwenden Sie diese Art der Definition, wenn die Inhalte der Ele-

mente entweder über die Tastatur eingegeben oder während der Makroausführung aktuell ermittelt werden, wie z. B. Ergebnisse von Monatsabschlüssen usw.

1 Definition eines eindimensionalen Arrays mit 10 Elementen. Die Elemente werden alle mit dem Namen *Texte* bezeichnet. Die Unterscheidung findet durch den Zusatz [*Index*] statt, der Werte von 1 – 10 annehmen darf.

2 Beginn einer *ForNext*-Schleife zur Steuerung der Dateneingabe. Dadurch wird *GetString* zehnmal ausgeführt.

3 Die jeweils eingegebenen Zeichen werden in den Elementen *Texte* in Verbindung mit dem Zähler *Index* gespeichert. Enthält *Index* den Wert 1, wird die Eingabe in dem ersten Element gespeichert, enthält *Index* den Wert 7, wird die Eingabe in dem siebten Element gespeichert usw.

4 Öffnen eines leeren Dokumentfensters.

5 Beginn einer *ForNext*-Schleife zur Steuerung der Datenausgabe. Dadurch werden die Inhalte der zehn Array-Elemente der Reihe nach in das Dokumentfenster geschrieben.

6 Schreiben eines konstanten Textes und der jeweils aktuellen Elementnummer.

7 Schreiben des jeweils aktuellen Elementinhalts, gefolgt von einer Zeilenschaltung.

Dadurch wird folgender Text gedruckt:

```
Array-Element (1) enthält: Eingaben
Array-Element (2) enthält: Eingaben
::::
Array-Element (10) enthält: Eingaben
```

Anstelle von Eingaben erscheint die jeweils unter Punkt 3 gespeicherte Dateneingabe.

6.34.2 Mehrdimensionale Arrays

Mehrdimensionale Arrays sind vergleichbar mit einer Tabelle, die z. B. aus einer Spalte mit maximal 10 Reihen bestehen kann. Jede Reihe kann wiederum in weitere Reihen untergliedert werden, die ihrerseits wieder weiter untergliedert werden können usw. Pro Reihe können max. 32.767 Felder definiert werden. Jedes dadurch entstehende Feld wird als ein Element der gedachten Tabelle bezeichnet. Einfacher gesagt: Hier handelt es sich um Variable, die Daten mehrerer Gruppen speichert, wobei die Daten einer Gruppe die dieselbe Bedeutung haben. Der Datentyp kann dabei unterschiedlich sein.

Der abgebildeten Grafik können Sie den Aufbau eines mehrdimensionalen Arrays entnehmen. Es handelt sich hier um ein dreidimensionales Array:

1. Die erste Dimension umfaßt die Werke 1 – 4.
2. Die zweite Dimension umfaßt die Abteilungen eines der vier Werke.
3. Die dritte Dimension umfaßt die Gruppen einer Abteilung des zugehörigen Werks.

Soll nun ein Gruppen-Element angesprochen werden, sind hierfür drei Indizes anzugeben: Einen für das Werk, einen für die Abteilung und einen für die Gruppe selbst. Um dieses Array zu definieren, müßte folgender Declare-Befehl angegeben werden:

```
Declare Gruppen [4;2;2]
```

- Der erste Wert legt die Anzahl der Werke fest (= 4).
- Der zweite Wert legt die Anzahl der Abteilungen innerhalb eines Werkes fest (= 2).
- Der dritte Wert legt die Anzahl der Gruppen innerhalb einer Abteilung fest (=2).

Um z. B. den Umsatz der zweiten Gruppe in der ersten Abteilung von Werk drei zum Gesamtumsatz zu addieren, müßte hier folgender Befehl angegeben werden:

```
Gesamtumsatz=       (Gesamtumsatz + Gruppen[3;1;2])
```

Mehrdimensionale Arrays

Konzern							
Werk 1		Werk 2		Werk 3		Werk 4	
Abteilung 1	Abteilung 2	Abteilung 1	Abteilung 2	Abteilung 1	Abteilung 2	Abteilung 1	Abteilung 2
Gruppe 1 Gruppe 2	Gruppe 1 Gruppe 2	Gruppe 1 Gruppe 2	Gruppe 1 Gruppe 2	Gruppe 1 Gruppe 2	Gruppe 1 Gruppe 2	Gruppe 1 Gruppe 2	Gruppe 1 Gruppe 2

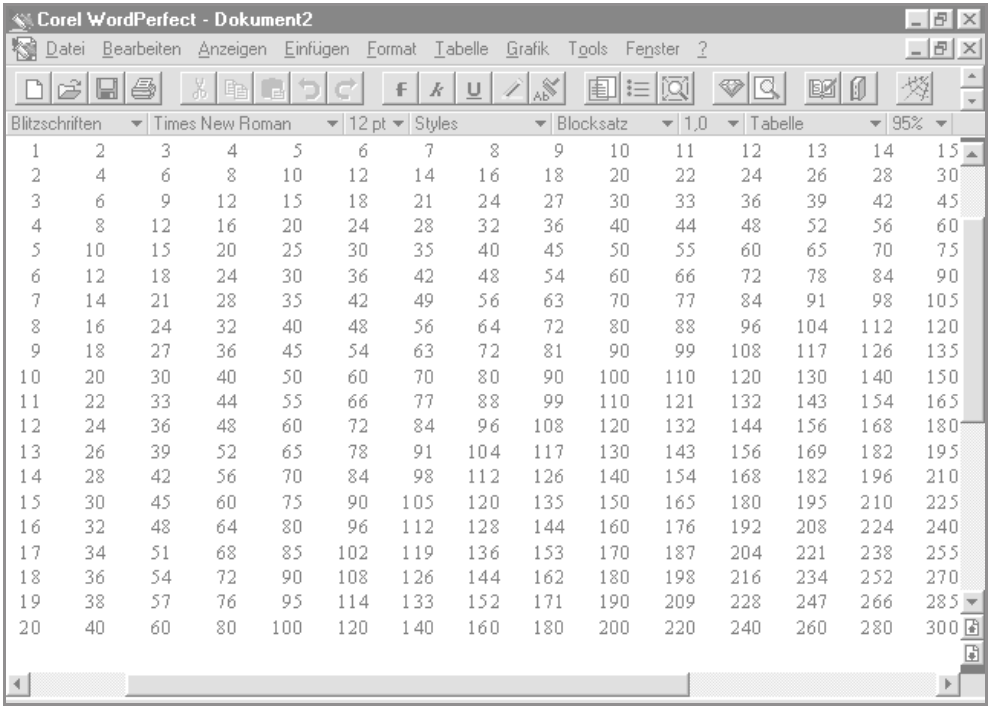
Jedem Element können vor Beginn einer Makroausführung Werte zugeordnet werden (ähnlich dem *Assign*-Befehl). Dies können numerische oder alphanumerische Daten sein:

```
Element[1;3]=        150
Element[5;3;7]=      "Das ist ein Test"
Element[2;6]=        Umsatz-Steuer
```

Anstelle der Ganzzahl können natürlich auch Variablen verwendet werden, die die betreffenden Elementnummern enthalten:

```
Element[Monat;Tag]=          150*Steuersatz
Element[Werk;Abtlg;Gruppe]=  "Das ist ein Test"
Element[Jahr;Monat]=         Umsatz-Steuer
```

Beispiel 4



1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
2	4	6	8	10	12	14	16	18	20	22	24	26	28	30
3	6	9	12	15	18	21	24	27	30	33	36	39	42	45
4	8	12	16	20	24	28	32	36	40	44	48	52	56	60
5	10	15	20	25	30	35	40	45	50	55	60	65	70	75
6	12	18	24	30	36	42	48	54	60	66	72	78	84	90
7	14	21	28	35	42	49	56	63	70	77	84	91	98	105
8	16	24	32	40	48	56	64	72	80	88	96	104	112	120
9	18	27	36	45	54	63	72	81	90	99	108	117	126	135
10	20	30	40	50	60	70	80	90	100	110	120	130	140	150
11	22	33	44	55	66	77	88	99	110	121	132	143	154	165
12	24	36	48	60	72	84	96	108	120	132	144	156	168	180
13	26	39	52	65	78	91	104	117	130	143	156	169	182	195
14	28	42	56	70	84	98	112	126	140	154	168	182	196	210
15	30	45	60	75	90	105	120	135	150	165	180	195	210	225
16	32	48	64	80	96	112	128	144	160	176	192	208	224	240
17	34	51	68	85	102	119	136	153	170	187	204	221	238	255
18	36	54	72	90	108	126	144	162	180	198	216	234	252	270
19	38	57	76	95	114	133	152	171	190	209	228	247	266	285
20	40	60	80	100	120	140	160	180	200	220	240	260	280	300

Hier wird ein zweidimensionales Array definiert, um z. B. über die berechnete Matrix das Ergebnis einer Multiplikation abzulesen. Das Ergebnis der Multiplikation ergibt sich aus der Position des adressierten Elements. Ist z. B. das Ergebnis der Multiplikation von 7 * 8 zu ermitteln (nicht zu errechnen!), muß über die beiden Indizes das betreffende Element ermittelt werden. In dem angezeigten Beispiel befindet sich das Ergebnis im Schnittpunkt von Spalte 7 und Reihe 8, also 56.

Um Platz zu sparen und um die Anzeige übersichtlich zu halten, werden hier nur die wichtigen Komponenten für die Abarbeitung des Arrays abgedruckt. Das komplette Makro finden Sie auf der CD-ROM unter dem Dateinamen DECLARE4.WCM. Nach der Berechnung wird ein Dialogfeld zur Eingabe von zwei zu multiplizierenden Zahlen angezeigt. Die Eingabewerte dürfen sich nur innerhalb des zulässigen Bereichs befinden (siehe unten). Das Ergebnis der Multiplikation wird nicht errechnet, sondern dem über die beiden Indizes adressierten Array-Element der Matrix entnommen.

```
1       Declare                 EinmalEins[20;15]
        ::::
2       ForNext                 (Z1;1;20)
3         ForNext               (Z2;1;15)
4           EinmalEins[Z1;Z2]=  (Z1 * Z2)
5         EndFor
6       EndFor
7       ::::                    Dialogfeld für Eingabe anzeigen.
8       Ergebnis=               NumStr(Einmaleins[Multiplikand;Multiplikator])
9       Prompt                  ("Ergebnis";Ergebnis;;;)
        ::::
10      Label                   (Drucken)
        FileNew                 ()
        Display                 (On!)

11      PaperSizeSelect         (Name: "A4 (Querformat)";
                                29.699c; 20.9995c;
                                Standard!)
12      ForNext                 (Z1;1;20)
13        ForNext               (Z2;1;15)
14        Meldung=              "Bitte warten, Array wird gedruckt, Zähler1:
                                "+NumStr(Z1)+",  Zähler2:  "+NumStr(Z2)
15        Prompt                ("Zweidimensionale Arrays";Meldung;;;)
16        TabRight              (Typ:Normal!)
17                              (NumStr(EinmalEins[Z1;Z2]))
18        If                    (Z2 = 15)
            HardReturn          ()
          EndIf
19      EndFor
20      EndFor
        EndPrompt
        ::::
```

1 Definition des zweidimensionalen Arrays *EinmalEins*. Insgesamt werden durch diese Definition 300 Array-Elemente festgelegt (20 * 15). Wie Sie der erstellten Matrix entnehmen können, besteht das Array aufgrund dieser Definition aus 20 Reihen (= erste Dimension), die jeweils in 15 Felder (= zweite Dimension) untergliedert sind.

2 Äußere *ForNext*-Schleife. Diese Schleife wird so lange durchlaufen, bis alle 20 Reihen erstellt wurden.

3 Innere *ForNext*-Schleife. Über diese Schleife werden die 15 Elemente je Reihe erstellt. Nach der Erstellung des letzten Elements wird der Zähler Z1 in der äußeren Schleife um 1 erhöht, und danach in der inneren Schleife mit Zähler Z2 wieder bei 1 begonnen.

4 Berechnen des Wertes des aktuellen Elements. Z1 (= Zähler für erste Dimension) und Z2 (= Zähler für zweite Dimension) sind die beiden Indizes, über die ein Element adressiert wird. Enthält z. B. Z1 den Wert 7 und Z2 den Wert 12, wird in Reihe sieben das zwölfte Element angesprochen. Durch Z1 * Z2 wird das Ergebnis ermittelt (= 84) und in dem Element gespeichert.

5-6 Schleifenende der inneren und äußeren Schleife.

7 Aufbau und Anzeige eines Dialogfeldes zur Eingabe zweier Zahlen, sowie Überprüfung, ob die Zahlen sich innerhalb des erlaubten Bereichs befinden: Bei der ersten Dimension von 1 – 20, bei der zweiten Dimension von 1 – 15.

8 Ermitteln des Ergebnisses. In den Feldern Multiplikand und Multiplikator wurden über das Dialogfeld zwei Zahlen eingegeben, die hier als Indizes für das Array verwendet werden (Z1 entspricht Multiplikand, Z2 entspricht Multiplikator). Dem dadurch adressierten Element wird das Ergebnis entnommen und in der Variablen *Ergebnis* gespeichert.

9 Anzeige des Ergebnisses.

10 Das unter Punkt 7 angezeigte Dialogfeld enthält eine Taste [Drucken], über die der Matrix gedruckt werden kann. Beim Klicken auf der Taste wird diese Unterroutine aufgerufen. Bei Bedarf können Sie die erstellte Matrix drucken.

11 Auswahl der Papiergröße A4-Quer, um eine Reihe komplett in einer Zeile zu drucken.

12 Äußere *ForNext*-Schleife. Diese Schleife wird so lange durchlaufen, bis alle 20 Reihen geschrieben wurden.

13 Innere *ForNext*-Schleife. Über diese Schleife werden die 15 Elemente je Reihe geschrieben. Nach dem Schreiben des letzten Elements wird der Zähler Z1 in der äußeren Schleife um 1 erhöht und danach in der inneren Schleife mit Zähler Z2 wieder bei 1 begonnen.

14-15 Meldung generieren und einblenden, die den Stand des Erstellens anzeigt.

16 Sprung auf den nächsten Tabstopp und rechtsbündige Ausrichtung.

17 Schreiben des Inhalts des aktuellen Elements.

18 Nach dem Schreiben des letzten Elements einer Reihe muß ein Zeilenumbruch erfolgen, damit der Ausdruck der nächsten Reihe wieder am Zeilenanfang beginnt.

19-20 Schleifenende der inneren und äußeren Schleife.

Die jeweiligen Werte können Sie beim Ablauf des Makros in der angezeigten Meldung verfolgen (siehe Zeile 14). Ergänzen Sie in dem Makro ggf. den Befehl *Speed*, um die Anzeige zu verlangsamen.

Diese Beispiele finden Sie auf der CD-ROM unter den Dateinamen DECLARE1.WCM DECLARE2.WCM, DECLARE3.WCM und DECLARE4.WCM.

Weitere Hinweise siehe: Dimensions, For, ForNext, Repeat, While

6.35 Default

Befehlsform: **Default:**(*Anweisungen*)

Dieser Befehl tritt immer in Verbindung mit dem *Switch*-Befehl auf. Die Verwendung ist wahlfrei. Über den Befehl *Default* können Sie festlegen, wie weitergearbeitet werden soll, wenn keine der *CaseOf*-Bedingungen zutrifft. *Default* muß, wenn er verwendet wird, immer dem letzten *CaseOf*-Befehl folgen. Benutzen Sie diesen Befehl, um z. B. bei nicht zutreffenden Bedingungen zu einer Fehlerroutine zu verzweigen oder um ggf. in solchen Fällen ein Makro zu beenden. Zwischen *Default* und *EndSwitch* können einer oder mehrere Befehle folgen.

Beispiel: Siehe unter CaseOf.

Weitere Hinweise siehe: Continue, CaseOf, Switch

6.36 DefaultUnits

Befehlsform: **DefaultUnits**(*Maßeinheiten*)

Hier können Sie Maßeinheiten vorgeben, die Sie z. B. beim Einstellen von Rändern oder beim Setzen von Tabulatoren innerhalb des Makros verwenden möchten. Folgende Maßeinheiten sind möglich (die Ausdrücke müssen wie angegeben, verwendet werden, Variable anstelle der Maßeinheit sind nicht erlaubt):

Centimeter!	Inches!Millimeters!Points!
WPUnits!	WP1200ths!None!

Die Angaben dürfen nicht durch Hochkommata eingeschlossen werden. Eine Maßeinheit ist so lange gültig, bis sie durch einen erneuten *DefaultUnits*-Befehl wieder überschrieben wird.

Parameter

Maßeinheiten Hier können Sie die zuvor genannten Maßeinheiten eingeben. Die danach eingegebenen Werte, wie z. B. Ränder, werden in der angegebenen Maßeinheit interpretiert, sofern bei der Eingabe von Werten keine Maßeinheiten angegeben werden.

Beispiel: **DefaultUnits**(Centimeter!).
Alle Maßeinheiten, die Sie eingeben, werden nach der Ausführung dieses Befehls als Zentimeter interpretiert. Möchten Sie vorübergehend eine andere Maßeinheit verwenden, können Sie diese auch direkt mit dem Wert eingeben, z. B. 2,5p. Obwohl Zentimeter vorgegeben sind, wird diese Eingabe als Punkte interpretiert.

6.37 Dialog

Allgemeine Hinweise

Über die nachfolgend beschriebenen Dialog-Befehle können Sie eigene Dialogfelder zur Verwendung in Ihren Makros definieren. Die Makro-Ausführung erhält dadurch ein professionelles Design, und die Bedienung wird erleichtert (z. B. Eingabemasken für Datenerfassung). Für alte Hasen: Die Befehle und die Befehlssyntax selbst (von der Großschreibung der Befehle mal abgesehen) sind identisch mit der Version 5.2. Beachten Sie bitte auch die Hinweise bei dem Befehl *DialogDefine* bzgl. der Variablennamen anstelle numerischer Konstanten und der Verwendung von Parameternamen. Eine einfachere Art und Weise zum Erstellen von Dialogfeldern finden Sie in Kapitel »11 Dialogeditor«. Für die Definition und die Verwendung solcher Dialogfelder müssen Sie folgende Reihenfolge einhalten:

1. Definieren eines Dialogfeldes.
2. Einbindung der benötigten Objekte (Linien, Tasten, Felder usw.).
3. Anzeigen eines Dialogfeldes.
4. Eingabe-Informationen oder gedrückte Tasten für Auswertung übernehmen.
5. Dialogfeld wieder ausblenden und im Speicher löschen.

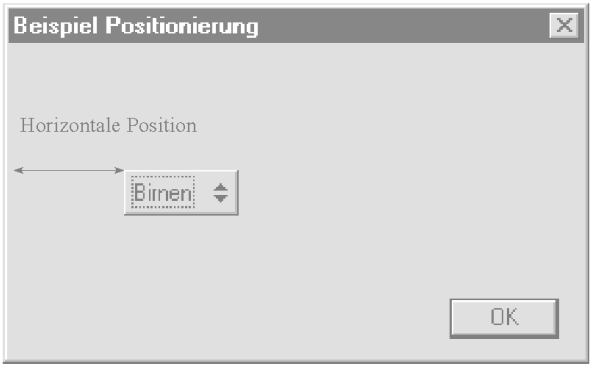
Selbstverständlich können auch die orginalen WordPerfect-Dialogfelder verwendet werden, die Sie unter den produktspezifischen Befehlen finden (mit der Endung ...Dlg). Wählen Sie zum Einfügen eigener Dialogbefehle [*Tools, Makro, Makroleiste, Befehle*]. Bei der Beschreibung der nachfolgenden Dialog-Befehle werden Parameter, die bei allen Befehlen identisch sind, hier im Anschluß beschrieben. Schlagen Sie bitte hier nach, wenn bei dem betreffenden Dialog-Befehl die Beschreibung eines Parameters fehlt:

DialogId Zur Unterscheidung verschiedener Dialogfelder erhält jedes Dialogfeld eine eigene Kennung (= DialogId), die bei allen zugehörigen Dialog..-Befehlen zur Identifikation angegeben werden muß. Sie können hierfür eine numerische oder eine alphanumerische Bezeichnung verwenden.

ControlId Nummer oder Name eines definierten Objekts (Item) innerhalb eines Dialogfeldes. Durch Angabe der ControlId können Sie z. B. in Verbindung mit dem Befehl *DialogDisplay* den Cursor in einem bestimmten Objekt (z. B. Textfeld für die Dateneingabe) positionieren. Maximal 1.000 Objekte sind erlaubt. Als ControlId können Sie Namen verwenden oder Zahlen. Namen haben den Vorteil, daß sie aussagefähig sind, was bei Zahlen wiederum nicht der Fall ist. Unterschiedliche Zahlen haben den Vorteil, daß Sie sich nicht jedesmal neue Namen ausdenken müssen. Welche Art der Namensgebung Sie verwenden, bleibt darum Ihnen überlassen.

Die nachfolgenden Parameter können als Ganzzahlen, als Variable mit ganzzahligem Inhalt, oder als Variable mit Offset definiert werden (siehe Abschnitt »6.37.25 DialogDefine«).

HPos Horizontale Position eines Objekts (Taste, Optionsfeld, Kontrollfeld usw.) innerhalb eines Dialogfeldes, gemessen in Dialogeinheiten vom linken Rand des Dialogfeldes bis zum linken Rand des Objekts.

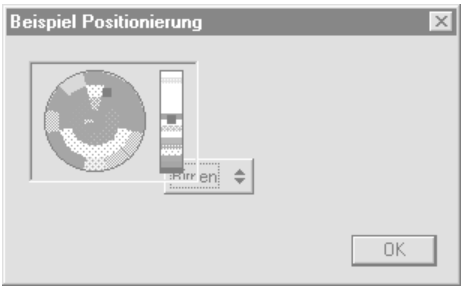


VPos Vertikale Position eines Objekts (Taste, Optionsfeld, Kontrollfeld usw.) innerhalb eines Dialogfeldes, gemessen in Dialogeinheiten vom oberen Rand des Dialogfeldes bis zum oberen Rand des Objekts.

Breite Breite des gesamten Objekts. Wird die Breite zu

kurz bemessen, werden Texte, die evtl. angezeigt werden sollen, rechtsbündig abgeschnitten, Objekte werden überblendet.

lang bemessen, können andere Elemente des Dialogfelds rechts daneben überschrieben werden, Objekte werden überblendet.



Höhe Höhe des gesamten Objekts. Wird die Höhe zu

niedrig bemessen, werden Texte, die angezeigt werden sollen oben bzw. unten abgeschnitten, Objekte werden überblendet.

hoch bemessen, können andere Elemente des Dialogfeldes überschrieben werden, Objekte werden überblendet.

Style Unter Style sind die Parameter (einer oder mehrere) der gewünschten Optionen einzutragen. Die verfügbaren Parameter müssen Sie der jeweiligen Beschreibung entnehmen. Mehrere Parameter sind durch | zu trennen.

Makros, die mit WordPerfect-Versionen bis 6.1 erstellt wurden, enthalten anstelle der Parameter-Namen Zahlen, wobei mehrere Zahlen durch + verbunden sind. Möchten Sie z. B. Styles verwenden, die mit den Zahlen 1, 16, 64 und 128 gekennzeichnet sind, müssen Sie diese wie folgt ohne Leerzeichen eintragen: 1+16+64+128. Auch die Summe dieser Zahlen kann als einziger Parameter verwendet werden, in diesem Beispiel also 209. Diese Makros laufen auch unter Version 7.

Dialogeinheiten Für jedes Dialogfeld müssen Sie für die horizontale und die vertikale Position, sowie für die Breite und Höhe, Angaben zur Größengestaltung und Positionierung des Dialogfeldes und der darin enthaltenen Objekte (z. B. Tasten, Kontrollfelder, Optionsfelder usw.) vornehmen. Die Maßeinheit, mit der hier gerechnet wird, nennt sich Dialogeinheit (Dialog Units). Die Einheiten sind für die vertikale und horizontale Richtung unterschiedlich:

Vertikal Eine Dialogeinheit = ein Achtel der Zeichenhöhe.

Horizontal Eine Dialogeinheit = ein Viertel der Zeichenbreite.

Probieren Sie vor der Erstellung eines Dialogfeldes diese Möglichkeiten aus.

Ein Dialogfeld sollten Sie objektweise aufbauen, d. h. nicht alle Objekte eines Dialogfeldes auf einmal definieren und dann testen. Halten Sie folgende Reihenfolge ein:

1. Dialogfelder sollten Sie bei der Definition unabhängig von einem bestimmten Makro erstellen und testen. D. h. Sie erstellen ein Makro, das nur die Definitionen des Dialogfeldes enthält. Erfahrungsgemäß müssen Sie dieses Makro zum Testen in der Anfangszeit sehr oft aufrufen. Zur Arbeitserleichterung sollten Sie darum den Makroaufruf über eine Makrotaste der Tastenleiste vornehmen. Erst wenn das Dialogfeld einwandfrei dargestellt wird und fehlerfrei arbeitet, sollten Sie es in das Makro einfügen, in dem es benötigt wird. Bei dieser Art der Erstellung haben Sie kürzere Verarbeitungszeiten (für Kompilierung und Test!) und zusätzlich bleibt das Makro klein und übersichtlich. Wenn Sie zu einem späteren Zeitpunkt bei einem größeren Makro ein Dialogfeld umfangreich ändern oder ergänzen müssen, gliedern Sie dieses aus dem Makro aus, und erstellen Sie daraus ein eigenes Makro, das nur die Definition dieses Dialogfeldes enthält. Kompilieren und Test laufen dann schneller (siehe oben).

2. Definieren Sie über *DialogDefine* ein leeres Dialogfeld, in dem Sie Breite und Höhe des Feldes festlegen. Wenn Sie sich hier verrechnet haben, können Sie die Maße nachträglich jederzeit ändern. Zusätzlich können Sie hier angeben, ob eine [*OK*]- und/oder eine [*Abbrechen*]-Taste erscheinen soll. Objekte können nur innerhalb des Bereichs definiert werden, der unter *DialogDefine* durch die Breite und Höhe des Dialogfeldes festgelegt wurde. Objekte, die nicht oder nur teilweise in dem Dialogfeld erscheinen, sind entweder zu groß bemessen oder auf einer falschen horizontalen bzw. vertikalen Position definiert. Ändern Sie dann ggf. die Größe des gesamten Dialogfeldes.

3. Definieren Sie der Reihe nach über die Befehle *DialogAdd...* die im Dialogfeld benötigten Objekte wie z. B. Tasten, Linien, Felder, Text usw. Damit ist gemeint, daß Sie nicht alle Objekte auf einmal definieren, sondern nach folgendem Schema vorgehen:

   - Ein Objekt definieren.
   - Über *DialogDislpay* das definierte Objekt anzeigen lassen. Speichern Sie das Makro, und führen Sie es anschließend aus.
   - Kontrollieren, ob das Objekt in der richtigen Größe an der gewünschten Position erscheint.
   - Nehmen Sie Änderungen vor, wenn Position oder Größe nicht Ihren Wünschen entsprechen. Bei größeren Dialogfeldern sollten Sie immer nur eine Änderung vornehmen und deren Auswirkung sofort kontrollieren. Bei mehreren gleichzeitigen Änderungen können Sie (zumindest in der Anfangszeit) evtl. nicht genau feststellen, welche der Änderungen sich wie auswirkt.

   Wiederholen Sie die zuvor genannten Punkte so lange, bis das Objekt in der entsprechenden Größe auf der gewünschten Position erscheint. Werden bei der Anzeige mehrerer Objekte Teile eines Objekts überschrieben oder abgeschnitten, müssen Sie die Breite bzw. die Höhe oder die horizontale bzw. vertikale Position des Objekts überprüfen:

Objekt wird überlagert

Das vorausgehende Objekt wurde zu groß definiert, so daß Teile eines oder mehrerer nachfolgender Objekte nicht angezeigt werden (siehe auch oben).

Objekt wird abgeschnitten
Das Objekt wurde zu klein definiert, um z. B. den vorgegebenen Text anzuzeigen (siehe auch oben).

4. Definieren Sie das nächste Objekt (sofern benötigt), in dem Sie die unter Punkt **3** genannten Punkte wiederholen.
5. Ergänzen Sie die logischen Befehle zur Steuerung des Makros. Hier wird über die üblichen Makro-Programmierbefehle wie z. B. *If* gesteuert, was zu tun ist, wenn auf bestimmten Optionsfeldern oder auf bestimmten Tasten geklickt wurde.
6. Definieren Sie Dialogfelder bei größeren Makros immer in Unterroutinen, damit das Definieren und Testen einfacher wird (leicht auszugliedern, siehe Punkt 1).

Erstellen Sie auf diese Art das gesamte Dialogfeld, indem Sie Befehl für Befehl ergänzen. Ein Musterbeispiel eines Dialogfeldes mit allen Dialogbefehlen finden Sie auf der CD-ROM unter dem Dateinamen DIALOG.WCM. Weitere Beispiele können Sie den bei der Installation von WordPerfect kopierten Makros entnehmen. Der Übersichtlichkeit wegen werden in den nachfolgenden Beispielen zu den Befehlen in erster Linie die jeweils beschriebenen Dialogbefehle verwendet, um Ihnen eine leichtere Einarbeitung zu gewährleisten. Sie können bei Ihrer praktischen Arbeit natürlich in einem Dialogfeld mehrere Dialogbefehle kombinieren und somit auch solche Dialogfelder erstellen, die Sie von WordPerfect her gewöhnt sind. Die in den folgenden Beispielen angezeigten Makros finden Sie auf der CD-ROM. Die ersten drei Stellen des Makronamens lauten immer DIA..., ergänzt um einen Teil des jeweiligen Befehls, so daß die Makronamen leicht zuzuordnen sind.

Beispiel:

```
DIACHECK.WCM    =    DialogAddCheckBox
DIACOLOR.WCM    =    DialogAddColorWheel
DIAFRAME.WCM    =    DialogAddFrame
DIARADIO.WCM    =    DialogAddRadioButton
```

In den meisten Makros ist noch eine Schleife eingebaut, damit Sie beim Testen unterschiedliche Eingabe- bzw. Auswahlmöglichkeiten ausprobieren können. Die von Ihnen ausgewählten Funktionen bzw. gedrückten Tasten werden zur Kontrolle über *Prompt* angezeigt, damit Sie das korrekte Arbeiten der Makros testen können. Die Schleifen sind nur in den Makros enthalten. In den hier aufgelisteten Beispielen wurden sie der Übersichtlichkeit wegen weggelassen.

Durch Drücken auf Esc werden diese Makros abgebrochen. Wenn Sie »einfache« Menüs verwenden wollen, können Sie diese über den Makro-Programmierbefehl *Menu* erstellen. Weitere Informationen entnehmen Sie bitte dem Abschnitt »6.76 Menu«. In dem auf CD-ROM befindlichen Makro DIALOG.WCM wurden zum Testen einige Möglichkeiten zusammengefaßt.

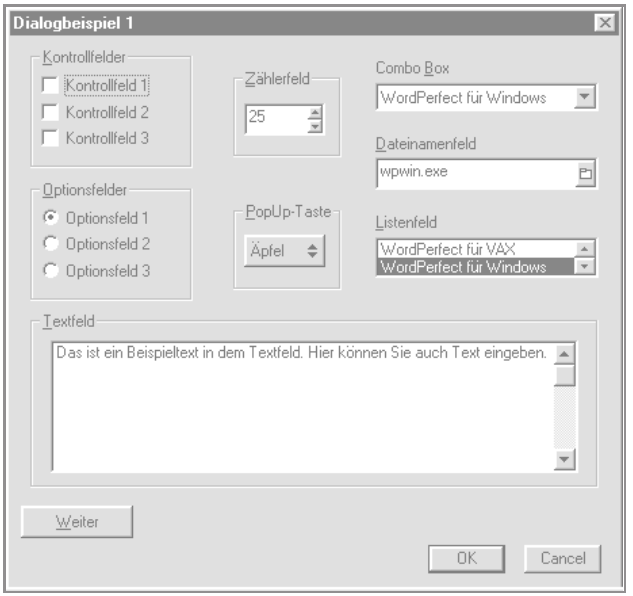


Bild 6.1: DIALOG.WCM, 1. Seite

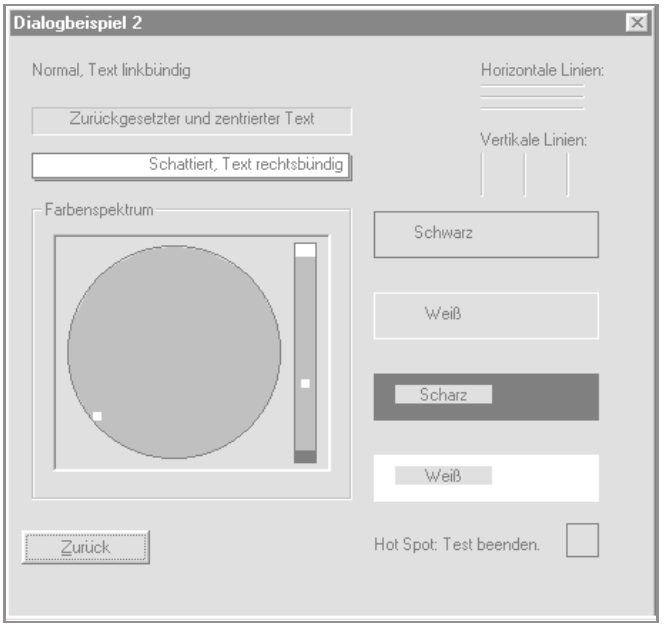


Bild 6.2: DIALOG.WCM, 2. Seite

6.37.1 DialogAddBitmap

Befehlsform:	**DialogAddBitmap** (*DialogId;ControlId;HPos;VPos;Breite;Höhe;Style;Dateiname;Name;Farbe*)

Dem Dialogfeld wird eine Bitmap-Grafik zugeordnet. Wurde ein Dialogfeld mit dem Dialogeditor erstellt, kann nach der Anzeige des Dialogfelds eine Bitmap ergänzt werden.

Parameter

Style Wie soll die Grafik angezeigt werden?

SizeCtlToBmp!	Das Element wird der Bitmapgröße angepaßt.
SizeBmpToCtl!	Die Bitmapgröße wird dem Element angepaßt.
Transparent!	Transparente Anzeige.

Dateiname Vollständiger Dateiname der Bitmapgrafik oder eine DLL-Datei.

Name Name der Bitmapdatei in einer DLL-Datei z. B. "#1".

Farbe Jede beliebige Farbe zwischen 0 und 16.777.215 (RGB-Farben), wobei 0 der Farbe »Schwarz« entspricht.

Beispiel

```
Application           (A1;"WordPerfect";Default;"DE")
OnError Call          (Fehler)
DialogDefine          ("DO1";50;50;200;100;OK! | Cancel!;"Makro-Beispiel")
DialogAddPushButton   ("DO1";"Taste1";110;35;80;14;;"Bitmap 1 anzeigen")
DialogAddPushButton   ("DO1";"Taste2";110;55;80;14;;"Bitmap 2 anzeigen")
DialogShow            ("DO1";"WordPerfect";Anzeigen)
CallbackWait
Quit

Label                 (Anzeigen)
If                    (Anzeigen(5)=274)
  DialogDestroy       ("DO1")
  CallbackResume
  Return
EndIf

Switch                (Anzeigen(3))
  CaseOf              "OKBttn":  MessageBox(x;"OK-Taste";"OK wurde gedrückt")
  CaseOf              "CancelBttn":DialogDestroy("DO1")
  CallbackResume
  Return
  CaseOf"Taste1":     DialogAddBitMap("DO1";"BMP";10;10;50;50;SizeBmpToCtl!;
                      "c:\win95\Wellen.bmp")
  CaseOf              "Taste2":  RegionSetBitMapFileName("DO1.BMP";"c:\win95\kreise.bmp")
  EndSwitch
Return
```

```
Label                (Fehler)
MessageBox           (x;"Fehler";"Sie müssen zuerst Bitmap 1 anzeigen lassen")
Return
```

Dieses Beispiel finden Sie auf der CD-ROM unter dem Dateinamen DIABITMP.WCM.

6.37.2 DialogAddCheckBox

Befehlsform: **DialogAddCheckBox**(*DialogId;ControlId;HPos;VPos;Breite;Höhe;Text;Variable;Style*)

Erstellen eines Kontrollfeldes mit zugehörigem Text. Den anzuzeigenden Text müssen Sie unter dem Parameter *Text* angeben. Nach dem Klicken in einem Kontrollfeld wird dieses je nach dem derzeitigen Zustand aktiviert oder deaktiviert, d. h. es kann ein- oder ausgeschaltet werden:

```
Kontrollfeld aktiv:   x
Kontrollfeld inaktiv: o
```

In dem unter *Variable* definierten Datenfeld wird nach dem Klicken auf [*OK*] der zu diesem Zeitpunkt aktuelle Zustand des Kontrollfeldes gespeichert (siehe unten). Wird dieser Variablen vor dem Einblenden des Dialogfeldes der Wert 1 zugeordnet, wird das zugehörige Optionsfeld beim erstmaligen Aufruf aktiviert.

Parameter

Text Text, der hinter dem Kontrollfeld erscheinen soll.

Variable In dieser Variablen wird nach dem Klicken auf [*OK*] der aktuelle Zustand des zugehörigen Kontrollfeldes gespeichert. Wenn das Dialogfeld über [*OK*] verlassen wird, enthält diese Variable entweder 0 oder 1. Wurde das Kontrollfeld aktiviert, wird 1 gesetzt, ansonsten 0.

Style Folgende Parameter stehen zur Verfügung:

CheckboxAuto! Wechselt zwischen markiert und nicht markiert.

Checkbox!	Nur ein Status möglich: Markiert oder nicht markiert.
Checkbox3State!	Erlaubter Status: Markiert, nicht markiert oder grau.
CheckboxAuto3State!	Wechselt zwischen markiert, nicht markiert und grau.
TextOnLeft!	Text wird links der Checkbox angezeigt.

Beispiel: Die folgenden Anweisungen erzeugen das angezeigte Dialogfeld.

```
Application          (A1;"WordPerfect";Default;"DE")
Feld1=               0
Feld2=               0
DialogDefine         (100; 50; 50; 200; 100; OK! | Cancel! | Percent!;
                     "Beispiel für DialogAddCheckBox")
DialogAddCheckBox    (100; 2000; 10; 10; 75; 15; "Kontrollfeld 1"; Feld1)
DialogAddCheckBox    (100; 2001; 10; 30; 75; 15; "Kontrollfeld 2"; Feld2)
DialogDisplay        (100; 2000)
DialogDestroy        (100)
```

Werden mehrere Kontrollfelder definiert, können Sie über die jeweils zugehörige Variable *Feld1* und *Feld2* angeben, welches Kontrollfeld bei der Anzeige des Dialogfeldes aktiviert sein soll. Diese Felder müssen vor der Ausführung des Befehls als Variable definiert worden sein. Sie können mehrere Kontrollfelder gleichzeitig aktivieren. Diese werden dann wie üblich mit einem diagonalen Kreuz gekennzeichnet. Mit der Tabulatortaste können Sie vorwärts von Kontrollfeld zu Kontrollfeld springen, durch Drücken von [Umschalten-Tab] können Sie rückwärts springen. Zum Aktivieren oder Deaktivieren eines Feldes klicken Sie auf das entsprechende Kontrollfeld. In dem gezeigten Beispiel sind unmittelbar nach dem erstmaligen Aufruf des Makros die beiden Felder deaktiviert (*Feld1* und *Feld2* enthalten den Wert 0).

Weitere Hinweise siehe: DialogAddRadioButton

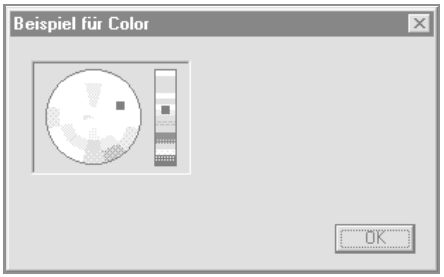
6.37.3 DialogAddColorWheel

Befehlsform: **DialogAddColorWheel**(*DialogId;ControlId;HPos;VPos;Breite;Höhe; Variable*)

Diese Funktion wird meistens zur Auswahl von Farben zum Drucken verwendet. Es wird ein Farbenspektrum angezeigt, das aus zwei Teilen besteht:

1. Durch Verschieben des weiß/schwarzen Quadrates in dem kreisförmigen Farbenspektrum können Sie eine Grobauswahl einer Farbe treffen. Ziehen Sie dieses Quadrat mit der Maus auf den gewünschten Farbton. Befindet sich das Quadrat am äußeren Rand des Kreises, werden die Farben intensiver angezeigt als bei Positionen, die mehr in der Mitte des Kreises liegen.

2. Durch Verschieben des weiß/schwarzen Quadrates innerhalb der vertikalen Farbenleiste können Sie Schattierungen der zuvor gewählten Farbe erzeugen. Beim Verschieben des Quadrates nach oben werden die Farben heller, bei der Verschiebung nach unten dunkler.



Nach der Auswahl der Farbe und dem Klicken auf [*OK*] wird unter *Variable* die ausgewählte Farbe in Form eines numerischen Wertes gespeichert, der sich von 0 (schwarz) bis 16.777.215 (weiß) erstreckt. Wenn Sie vor der Anzeige des Dialogfeldes dieser Variablen einen numerischen Wert zuordnen, wird ein Farbenspektrum in der entsprechenden Schattierung angezeigt.

Parameter

Variable Dieser Variablen wird beim Verlassen des Dialogfeldes die ausgewählte Farbe in Form eines numerischen Wertes zugeordnet. Dieser Wert ergibt sich aus der Kombination der RGB-Intensität (Rot/Grün/Blau) der gewählten Schattierung, die mit folgender Formel errechnet wird:

Rot	=Variable % 256
Grün	=Variable / 256 % 256
Blau	=Variable / 256 / 256

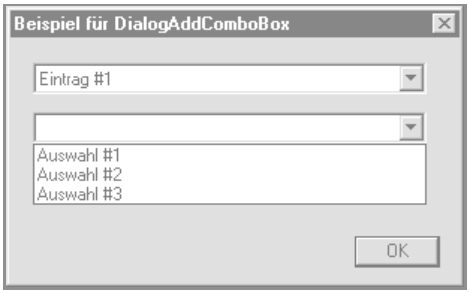
Beispiel: Die folgenden Anweisungen erzeugen das angezeigte Dialogfeld.

```
Application          (A1;"WordPerfect";Default;"DE")
Farbe=               8000000000
DialogDefine         (100; 50; 50; 200; 100; OK! | Cancel! | Percent!; "Beispiel für Color")
DialogAddColorWheel  (100; 1001; 10; 10; 75; 50; Farbe)
DialogDisplay        (100; 1001)
DialogDestroy        (100)
Blau=                Farbe/256/256
Grün=                Farbe/256%256
Rot=                 Farbe%256
```

Das Farbenspektrum im Kreis, sowie das in dem vertikalen Balken, können Sie unterschiedlich groß anzeigen. Die Breite und Höhe richten sich nach den bei *DialogAddColorWheel* definierten Größen.

6.37.4 DialogAddComboBox

Befehlsform: **DialogAddComboBox**
(*DialogId;ControlId;HPos;VPos;Breite;Höhe;Style;Variable;DatenLimit*)



Über diesen Befehl wird ein Textfeld angezeigt, das einen oder mehrere Einträge enthalten kann. Nach der Anzeige des Dialogfeldes wird das Textfeld mit einem Eintrag angezeigt, der unter *Variable* angegeben wurde. Enthält diese Variable vor der Anzeige keinen Eintrag, bleibt das Feld bei der Anzeige leer, so daß Sie eine beliebige Eingabe vornehmen können.

Für die Anzeige in diesem Feld können Sie mehrere Einträge definieren (in dem gezeigten Beispiel jeweils drei), die nach dem Klicken auf der Taste rechts des Feldes in einer Liste angezeigt werden. Jeder Eintrag entspricht einem Menüpunkt, der ausgewählt werden kann. Klikken Sie zur Auswahl auf dem gewünschten Eintrag. Dieser wird in *Variable* gespeichert und kann im Makro abgeprüft werden, um entsprechend zu reagieren. Ist *Variable* leer, wurde kein Eintrag ausgewählt.

Parameter

Style Die nachfolgend genannten Optionen sind verfügbar. Möchten Sie mehrere der Optionen kombinieren, müssen Sie die Kennziffern in der angegebenen Reihenfolge durch Pluszeichen (+) verbinden:

Droplist! Wurden für das Textfeld mehrere Einträge definiert, wird nach dem Klicken auf der Pfeiltaste rechts des Textfeldes eine Liste aller Einträge angezeigt. Jeden Eintrag können Sie in dem Textfeld verändern. Nach dem Klicken auf [*OK*] wird der ausgewählte Eintrag in *Variable* gespeichert. Zum schnellen Suchen können Sie die ersten Zeichen eingeben, mit denen der gewünschte Eintrag beginnt. Der Cursor wird auf diesem Eintrag positioniert. Klicken Sie zur Auswahl des Eintrags auf der Pfeiltaste rechts des Textfeldes. Es können Daten in der unter *DatenLimit* angegebenen Länge eingegeben werden.

	DropDown!	Wie unter 0, Liste wird jedoch erst nach dem Klikken auf der Taste rechts des Textfeldes geöffnet.
	Simple!	Anzeigen einer geöffneten Listenbox.
	Sort!	Die Listeneinträge werden in alphabetischer Reihenfolge angezeigt. In diesem Fall wird Simple! als Standardvorgabe verwendet.
	WPChars!	Zeichen der WordPerfect-Zeichensätze sind erlaubt.

DatenLimit Begrenzung der einzugebenden Daten. Hier können Sie festlegen, wieviel Zeichen eingegeben werden dürfen, wenn dies über Style erlaubt ist.

Beispiel: Die folgenden Anweisungen erzeugen das abgebildete Dialogfeld.

```
Application         (A1;"WordPerfect";Default;"DE")
Feld1=              "Eintrag #1"
Feld2=              ""
DialogDefine        (100; 50; 50; 200; 100; OK! | Cancel! | Percent!;
                    "Beispiel für DialogAddComboBox")
DialogAddComboBox   (100; 2000; 10; 10; 175; 50; ; Feld1; 10)
DialogAddListItem   (100; 2000; "Eintrag #1")
DialogAddListItem   (100; 2000; "Eintrag #2")
DialogAddListItem   (100; 2000; "Eintrag #3")
DialogAddComboBox   (100; 2001; 10; 30; 175; 50; ; Feld2; 0)
DialogAddListItem   (100; 2001; "Auswahl #1")
DialogAddListItem   (100; 2001; "Auswahl #2")
DialogAddListItem   (100; 2001; "Auswahl #3")
DialogDisplay       (100; 2001)
DialogDestroy       (100)
```

Nach der Anzeige des Dialogfeldes erscheint in dem Textfeld nur der erste Listeneintrag (der Eintrag, der unter *Feld1* vorgegeben wurde). Erst nach dem Klicken auf der Taste rechts des Textfeldes werden die restlichen Einträge, die jeweils durch einen eigenen Befehl *DialogAddListItem* definiert werden müssen, in einer Liste unterhalb des Textfeldes angezeigt. Die Einträge erscheinen in der Reihenfolge, wie sie im Makro definiert wurden.

Achten Sie darauf, daß die ControlIDs von *DialogAddComboBox* und die in den zugehörigen Objekten *DialogAddListItem* identisch sein müssen (hier: 2000 bzw. 2001).

6.37.5 DialogAddControl / DialogHandle

Befehlsform: **DialogAddControl**
(*DialogId;ControlId;HPos;VPos;Breite;Höhe;Klasse;Style;Fenster;Variable;Instanz*)

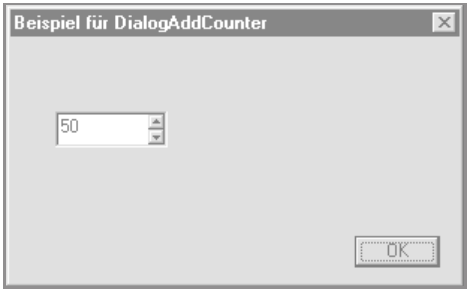
Durch *DialogAddControl* wird ein benutzerdefiniertes Element (Control) in dem Dialogfeld ergänzt. Die Befehle *DialogAddControl* und *DialogHandle* können wegen diverser Parameter nur

in Verbindung mit dem »GroupWise Software Develepors's Kit« sinnvoll erklärt werden. Schlagen Sie darum bitte in diesen Unterlagen nach, sofern sie Ihnen zur Verfügung stehen (müssen zusätzlich gekauft werden). Leider stand mir beim Erstellen dieser Ergänzung dieses Kit noch nicht zur Verfügung, so daß keine weitere Angaben folgen können. Wenn Sie diese Befehle trotzdem verwenden möchten, schauen Sie sich in dem Makro-Online-Handbuch das zugehörige Beispiel an.

Weitere Hinweise siehe: DialogDefine, DialogShow, MacroDialogResult, DialogHandle

6.37.6 DialogAddCounter

Befehlsform: DialogAddCounter
(*DialogId;ControlId;HPos;VPos;Breite;Höhe;*
Format;Variable;Minimum;Maximum;Abstand)



Definition eines Zählerfeldes und zwei Zählertasten mit Pfeilen, die beim Klicken auf den Pfeiltasten die Auswahl bestimmter Werte zwischen *Minimum* und *Maximum* erlauben. Durch Klicken auf den Pfeiltasten wird der Wert des Zählerfeldes in dem Bereich zwischen *Minimum* und *Maximum* um den unter *Abstand* definierten Wert vermindert oder erhöht. Das Zählerfeld können Sie auch wie üblich manuell verändern. Dann werden *Minimum* und *Maximum* nicht berücksichtigt, d. h. Sie können beliebige Werte eingeben. Über *Format* können Sie Maßeinheiten auswählen. Dem Feld *Variable* können Sie vor der Anzeige einen Wert zuweisen.

Parameter

Style Folgende Parameter stehen zur Verfügung, von denen jeweils nur einer verwendet werden darf:

DisplayNormal!	(-2147483647 bis 2147483647)
DisplayWPU!	(-2147483647 bis 2147483647)
DisplayPoints!	(-4294967.0 bis 4294966.0)
DisplayCentimeters!	(-151518.0 bis 151519.0)
DisplayFixedPoint!	(-32767.0 bis 32767.0)
DisplayPercent!	(-2147483647 bis 2147483647)

DisplayInches!	(-59651.0 bis 59652.0)
DisplayInches2!	(-59651.0 bis 59652.0)
DisplayMillimeters!	(-1515513.0 bis 1515514.0)
DisplayI_Inches	(-59651.0 bis 59652.0)
AutoHScroll!	
NoScroll!	
AutoValidated!	

Variable Der eingegebene Zählerwert wird in dieser Variablen gespeichert. Vor der Anzeige des Dialogfeldes können Sie dieser Variablen einen Wert zuweisen, der dann vorgeschlagen wird. Wird diesem Wert eine Maßeinheit zugewiesen, wird diese nach der Eingabe eines Wertes beibehalten.

Minimum Niedrigster Zählerwert, der über die Pfeiltasten ausgewählt werden kann.

Maximum Höchster Zählerwert, der über die Pfeiltasten ausgewählt werden kann.

Abstand Abstand zwischen zwei Werten, wenn auf den Pfeilen der Zählertaste geklickt wird.

Beispiel: Die folgenden Anweisungen erzeugen das angezeigte Dialogfeld.

```
Application         (A1;"WordPerfect";Default;"DE")
Wert=               50
DialogDefine        (100; 50; 50; 200; 100;  OK! | Cancel! | Percent!;
                    "Beispiel für DialogAddCounter")
DialogAddCounter    (100; 1000; 20; 30; 40; 15; DisplayNormal!; Wert; 0; 100; 10)
DialogDisplay       (100; 1)
DialogDestroy       (100)
```

Durch Klicken auf der oberen Pfeiltaste werden die Werte in dem Zählerfeld in dem angegebenen Abstand (10) erhöht bis maximal zur Obergrenze (100), beim Klicken auf der unteren Pfeiltaste werden die Werte in dem angegebenen Abstand vermindert, bis die Untergrenze (0) erreicht wird. Das Erreichen der Grenzen wird jeweils durch ein akustisches Signal angezeigt. Sie können manuell jeden beliebigen Wert eingeben, auch solche, die außerhalb der definierten Grenzen von *Minimum* und *Maximum* liegen. Sicherheitshalber müssen Sie darum im Makro prüfen, ob die eingegebenen Werte auch erlaubt sind (aus Datumsfeldern darf z. B. keine Eingabe 31.02.1996 übernommen werden).

6.37.7 DialogAddDate

Befehlsform: **DialogAddDate**
(*DialogId;ControlId;HPos;VPos;Breite;Höhe;Style;MakroVar;Jahr;Monat;Tag*)

Anzeigen eines Datums in einem Bereich des Dialogfelds. Wurde ein Dialogfeld mit dem Dialogeditor erstellt, kann über diesen Befehl nach dem Anzeigen des Dialogfelds das Datum ergänzt werden.

Parameter

Style Formale Prüfung des Datums.

- Validate! — Prüfen des eingegebenen Datums. Ist das Datum nicht korrekt, z. B. »31. Juni 1996«, erfolgt eine Fehlermeldung. Verwenden Sie diesen Parameter in jedem Fall, um falsche Datumseingaben zu vermeiden.

MakroVar Diese Variable enthält den im Makro gesetzten Wert, wenn das Dialogfeld/der Bereich angezeigt wird bzw. den eingegebenen Wert, der beim Schließen des Dialogfelds in diesem Datenfeld vorhanden ist.

Jahr Numerischer Wert der Jahresangabe.

Monat Monatsname ausgeschrieben (z. B. Januar!, Februar!, März! usw.).

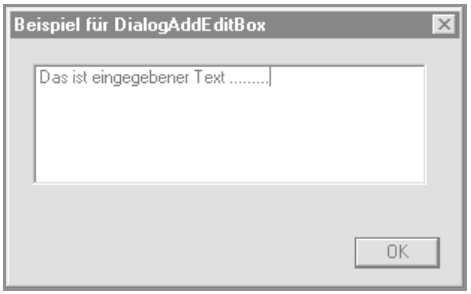
Tag Numerischer Wert der Tagesangabe.

Weitere Hinweise siehe: Date-Befehle, Time-Befehle

6.37.8 DialogAddEditBox

Befehlsform: **DialogAddEditBox**
(*DialogId;ControlId;HPos;VPos;Breite;Höhe;Style;Variable;DatenLimit*)

Definition eines Textfeldes, in dem der Benutzer Daten eingeben kann. Sie können einzeilige oder mehrzeilige Textfelder definieren. Der eingegebene Text wird in der vorgegebenen Länge in *Variable* gespeichert. Wird dieser Variablen vor dem Einblenden des Dialogfeldes ein Text zugeordnet, wird dieser Text in dem Textfeld angezeigt. Enthält das Textfeld beim Schließen des Dialogfeldes keinen Text, bleibt das Feld *Variable* leer.



Parameter

Style Die nachfolgend genannten Parameter können Sie auswählen:

Attributes!	Attribute sind erlaubt (Fett, unterstreichen usw.).
AutoHScroll!	Automatisch horizontal blättern.
AutoVScroll!	Automatisch vertikal blättern.
Center!	In der Mitte ausgerichteter Text, wenn er nicht in Verbindung mit WP-Zeichensätzen verwendet wird.
Hscroll!	Horizontale Bildlaufleiste anzeigen.
Left!	Linksbündig ausgerichteter Text.
Right!	Rechtsbündig ausgerichteter Text, wenn er nicht in Verbindung mit WP-Zeichensätzen verwendet wird.
LowerCase!	Nur Kleinbuchstaben sind erlaubt.
Multiline!	Mehrzeilige Dateneingabe möglich.
NoTabs!	Tabulatoren werden nicht in die Ausgabe übernommen.
NoWPChars!	Das Aufrufen von WordPerfect-Zeichensätzen ist nicht erlaubt.
Password!	Anstelle der eingegeben Zeichen werden * angezeigt.
QEMConvert!	ANSI-Zeichensatz in OEM-Zeichensatz konvertieren.
ReadOnly!	Keine Texteingabe in Box möglich.
SoftRet!	In der Ausgabe werden »Softreturns« (NZ) eingefügt.

UpperCase!	Nur Großbuchstaben sind erlaubt bzw. eingegebene Kleinbuchstaben werden in Großbuchstaben umgesetzt.
Vscroll!	Vertikale Bildlaufleiste anzeigen.
WantReturn!	↵ erzeugt in Verbindung mit Multiline! einen Zeilenumbruch.
WordWrap!	Automatischer Zeilenumbruch bei mehrzeiligen Feldern.
WPChars!	Verwendung von WordPerfect-Zeichensätzen ist erlaubt.

Variable Variable, die den eingegebenen Text aufnimmt. Text, der vor der Anzeige des Textfeldes in der Variablen gespeichert wurde, wird in dem Textfeld angezeigt.

DatenLimit Begrenzung der einzugebenden Daten. Hier können Sie festlegen, wieviel Zeichen maximal eingegeben werden dürfen.

Beispiel: Die folgenden Anweisungen erzeugen das angezeigte Dialogfeld.

```
Application         (A1;"WordPerfect";Default;"DE")
Daten=              ""
DialogDefine        (100; 50; 50; 200; 100;  OK! | Cancel! | Percent!; "Beispiel für
                                                          DialogAddEditBox")
DialogAddEditBox    (100; 1000; 10; 10; 175; 50; Chars!|Multiline|WordWrap; Daten; 100)
DialogDisplay       (100; 1000)
DialogDestroy       (100)
```

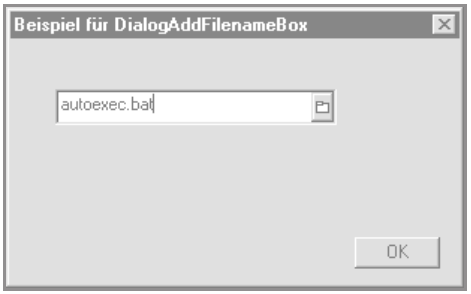
Beim Anzeigen des Dialogfeldes wird der unter *Daten* angegebene Text angezeigt (in diesem Beispiel ein leeres Textfeld). Mit den eingegebenen Daten können Sie in dem Makro dann weiterarbeiten. Verwenden Sie diese Art der Dateneingabe z. B. zur Erfassung von Adressendaten, die nach der Eingabe in einer Datendatei gespeichert werden.

6.37.9 DialogAddFilenameBox

Befehlsform: **DialogAddFilenameBox**(*DialogId;ControlId;HPos;VPos;Breite;Höhe; Style;Variable;StandVerz;Maske*)

Definieren eines Dialogfeldes zum Anzeigen bzw. Auswählen einer Datei. Der ausgewählte Dateiname wird in *Variable* einschließlich Laufwerksbezeichnung und Ordnername gespeichert. Wird dieser Variablen vor dem Einblenden des Dialogfeldes ein Dateiname zugeordnet, wird dieser Text in dem Textfeld beim erstmaligen Aufruf vorgeschlagen.

Über die Ordnertaste rechts des Textfeldes können Sie sich wie üblich einen Dateiordner anzeigen lassen und von dort aus weiterarbeiten. Auch die manuelle Eingabe eines Dateinamens in das Textfeld ist möglich. Wird das Textfeld gelöscht, enthält *Variable* nach dem Klikken auf [*OK*] keinen Eintrag.



Parameter

Style	Die nachfolgend genannten Optionen können Sie auswählen.	
	DirOnly!	Nur Ordner sind erlaubt
	FilesAndDirs!	Anzeige von Ordnern und Dateien.
	FileDoesntHave	
	ToExist!	Datei muß nicht vorhanden sein.
Variable	Variable, zur Speicherung der Auswahl.	
StandVerz	Angabe eines Standardordners, das nach dem Klicken auf der Ordnertaste angezeigt werden soll.	
DatMaske	Dateinamen-Maske der anzuzeigenden Dateien, z. B. *.*, *.txt usw.	

Beispiel: Die folgenden Anweisungen erzeugen das angezeigte Dialogfeld.

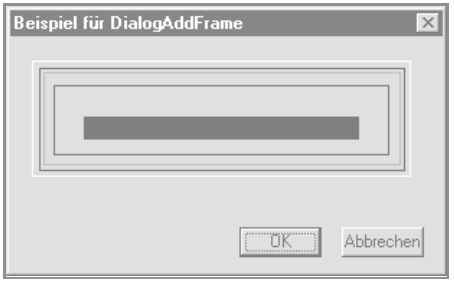
```
Application           (A1;"WordPerfect";Default;"DE")
Dateiname=            "autoexec.bat"
DialogDefine          (100; 50; 50; 200; 100; OK! | Cancel! | Percent!;
                      "Beispiel für DialogAddFileNameBox")
DialogAddFileNameBox (100; 1000; 20; 20; 125; 15; FilesAndDirs; Dateiname; "c:\"; "*.*")
DialogDisplay         (100; 1000)
DialogDestroy         (100)
```

Der unter *Dateiname* angegebene Dateiname wird bei der Anzeige des Dialogfeldes vorgeschlagen. Nach dem Klicken auf der Ordnertaste werden die Dateien angezeigt, die der Vorgabe in dem Befehl *DialogAddFileNameBox* entsprechen. In diesem Beispiel werden alle Dateien des Stammordners der Festplatte C:\ angezeigt. Diesen Befehl können Sie in Verbindung mit *DialogAddViewer* verwenden, um den Dateiinhalt einer ausgewählten Datei anzeigen zu lassen.

Weitere Hinweise siehe: DialogAddViewer

6.37.10 DialogAddFrame

Befehlsform: **DialogAddFrame**(*DialogId;ControlId;HPos;VPos;Breite;Höhe;Style*)



Definieren eines Rahmens, der innerhalb eines Dialogfeldes angezeigt werden kann, um z. B. größere Dialogfelder optisch zu untergliedern und dadurch übersichtlicher zu gestalten. Die Verwendung von Farben und Schattierungen ist möglich. Beachten Sie bitte auch den Befehl *DialogAddGroupBox*, wenn Sie Optionsfelder in einer Gruppe zusammenfassen wollen.

Parameter

Style Die nachfolgend genannten Optionen können Sie auswählen. Nicht alle Kombinationen sind erlaubt.

Black!	Schwarzer Rahmen.
Filled!	Ausgefüllter Rahmen.
Frame!	Rahmen anzeigen.
Gray!	Grauer Rahmen.
White!	Weißer Rahmen.

Beispiel: Die folgenden Anweisungen erzeugen das angezeigte Dialogfeld.

```
Application       (A1;"WordPerfect";Default;"DE")
DialogDefine      (100; 50; 50; 200; 100;  OK! | Cancel! | Percent!;
                  "Beispiel für DialogAddFrame")
DialogAddFrame    (100; 1000; 10; 10; 175; 50; Frame! | White!)
DialogAddFrame    (100; 1002; 13; 13; 169; 44; Black!)
DialogAddFrame    (100; 1001; 15; 15; 165; 40; Gray!)
DialogAddFrame    (100; 5555; 20; 20; 155; 30; Frame!)
DialogAddFrame    (100; 1009; 34; 34; 127; 10; Filled!)
DialogDisplay     (100; 1000)
DialogDestroy     (100)
```

6.37.11 DialogAddGroupBox

Befehlsform: **DialogAddGroupBox**(*DialogId;ControlId;HPos;VPos;Breite;Höhe;Titel*)

Definieren einer Gruppe, die innerhalb eines Dialogfeldes angezeigt werden kann, um z. B. größere Dialogfelder optisch zu untergliedern (bessere Lesbarkeit). Die Gruppe können Sie mit einem Gruppennamen versehen. Werden mehrere Optionsfelder innerhalb des Gruppenbereichs verwendet, kann immer nur ein Optionsfeld aktiviert werden.

Kontrollfelder können generell ohne Rücksicht auf andere Optionsfelder aktiviert oder deaktiviert werden.

Parameter

Titel Anzeigen eines Textes im linken Teil der obersten Linie des Gruppenbereichs (= Gruppenüberschrift).

Beispiel: Die folgenden Anweisungen erzeugen das angezeigte Dialogfeld.

```
Application          (A1;"WordPerfect";Default;"DE")
DialogDefine         (100; 50; 50; 200; 100;  OK! | Cancel! | Percent!;
                     "Beispiel für DialogAddGroupBox")
DialogAddGroupBox    (100; 1000; 10; 10; 175; 45; "Gruppenbereich")
DialogDisplay        (100; 1000)
DialogDestroy        (100)
```

Verwenden Sie diesen Befehl, wenn in einem Dialogfeld mehrere Optionsfelder definiert sind, die unterschiedlichen logischen Einheiten angehören. Fassen Sie jede logische Einheit, die aus mehreren Optionsfeldern bestehen kann, durch einem ***DialogAddGroupBox-Befehl*** zusammen. Sind keine Optionsfelder in dem Dialogfeld vorhanden, können Sie diese Funktion trotzdem zur optischen Untergliederung des Dialogfeldes verwenden. Das Dialogfeld sieht dann nicht so »nackt« aus.

Weitere Hinweise siehe: DialogAddRadioButton

6.37.12 DialogAddHLine

Befehlsform: **DialogAddHLine**(*DialogId;ControlId;HPos;VPos;Länge*)

Definieren einer horizontalen Linie, die innerhalb eines Dialogfeldes angezeigt werden kann, um z. B. größere Dialogfelder optisch zu untergliedern und dadurch übersichtlicher zu gestalten.

Parameter

Länge Länge der Linie in Dialogeinheiten.

Beispiel: Die folgenden Anweisungen erzeugen das angezeigte Dialogfeld.

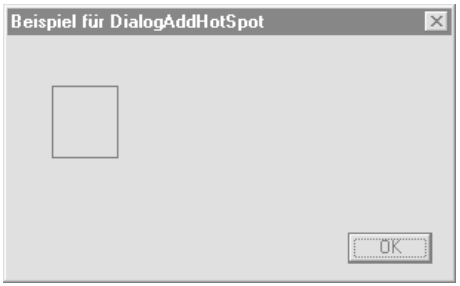
```
Application         (A1;"WordPerfect";Default;"DE")
DialogDefine        (100; 50; 50; 200; 100; OK! | Cancel! | Percent!;
                    "Beispiel für HLine und VLine")
DialogAddHLine      (100; 1000; 10; 10; 175)
DialogAddVLine      (100; 1001; 100; 15; 50)
DialogDisplay       (100; 1)
DialogDestroy       (100)
```

Weitere Hinweise siehe: DialogAddVLine

6.37.13 DialogAddHotSpot

Befehlsform: **DialogAddHotSpot**(*DialogId;ControlId;HPos;VPos;Breite;Höhe;Style*)

Definieren eines Feldes, das beim Anklicken durch einen einfachen oder durch einen Doppelklick das Dialogfeld ausblendet und ggf. eine damit verbundene Aktion ausführt. Dieses Feld können Sie über ein Icon legen, um durch Anklicken das Icon auszuwählen.



Parameter

Style Folgende Optionen können Sie wählen:

Click!	Einfacher Klick schließt das Dialogfeld.
DblClick!	Doppelklick schließt das Dialogfeld.

Beispiel: Die folgenden Anweisungen erzeugen das angezeigte Dialogfeld.

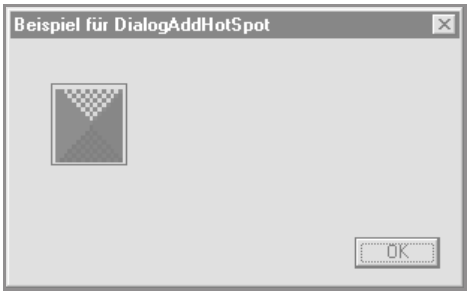
```
Application          (A1;"WordPerfect";Default;"DE")
DialogDefine         (100; 50; 50; 200; 100; OK! | Percent!; "Beispiel für HotSpot")
DialogAddHotSpot     (100; 1000; 20; 20; 30; 30; Click!)
DialogAddFrame       (100;1001;20;20;35;35;Frame!)
DialogDisplay        (100; 1000)
Hot=                 MacroDialogResult
                     // Die Makro-Variable MacroDialogResult muß vor dem Befehl DialogDestroy
                     // übergeben werden, sonst steht sie nicht mehr zur Verfügung.
DialogDestroy        (100)
If                   (Hot="1000")
  Meldung=           "HotSpot wurde angeklickt."
Else
  Meldung=           "HotSpot wurde nicht angeklickt."
EndIf
Prompt               ("Ergebnis";Meldung;;;)
EndPrompt
```

Weitere Hinweise siehe: DialogAddIcon.

6.37.14 DialogAddIcon

Befehlsform: DialogAddIcon(*DialogId;ControlId;HPos;VPos;Breite;Höhe; IconName;DLLVerb*)

Definieren (Auswählen) eines Icons, das in dem Dialogfeld angezeigt werden soll. Das Einfügen von Icons dient entweder der Auflockerung eines Dialogfeldes (reine optische Zwecke) oder der Auswahl einer Funktion (Steuerung eines Makros). Im letzteren Fall können Sie diesen Befehl in Verbindung mit *DialogAddHotSpot* verwenden.



Dadurch wird über das Anklicken des Icons eine Funktion ausgelöst. Das Icon muß in einer DLL-Datei oder einer Windows-EXE-Datei vorhanden sein. Der Name des Icons muß Ihnen bekannt sein.

Parameter

IconName Name des gewünschten Icons.

DLLVerb Angabe der DLL (Dynamic Link Library) oder der Windows-EXE-Datei, die das Icon enthält. Wenn Sie DLL-Dateien verwenden, können Sie den Makro-Programmierbefehl *DLLLoad* verwenden.

Beispiel: Die folgenden Anweisungen erzeugen das angezeigte Dialogfeld.

```
Application          (A1;"WordPerfect";Default;"DE")
DLLLoad              (DLLHandle;"C:\COREL\OFFICE7\SHARED\PFIT7\WPSH2ODE.DLL")
DialogDefine         (100; 50; 50; 200; 100; OK! | Percent!; "Beispiel für DialogAddIcon")
DialogAddHotSpot     (100; 1000; 20; 20; 30; 30; 1)
DialogAddFrame       (100; 1001;20;20;35;35;Frame!)
DialogAddIcon        (100; 1001; 25; 25; 30; 30; "hier ICON-Name eingeben"; DLLHandle)
DLLFree              (DLLHandle)
DialogDisplay        (100; 1000)
```

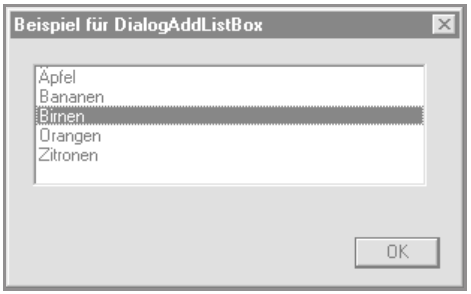
Weitere Hinweise siehe: DialogAddHotSpot.

6.37.15 DialogAddListBox

Befehlsform: **DialogAddListBox**(*DialogId;ControlId;HPos;VPos;Breite;Höhe;Style;Variable*)

Definieren eines Listenfeldes, das Einträge enthält, die mit DialogAddListItems definiert wurden. Der ausgewählte Eintrag wird in *Variable* gespeichert. Wird dieser Variablen vor dem Einblenden des Dialogfeldes ein String zugeordnet, so wird dieser String in dem Textfeld beim erstmaligen Aufruf vorgeschlagen.

Durch Eintippen eines oder mehrerer Anfangsbuchstaben eines Eintrags können Sie eine Suchfunktion starten.



Parameter

Style Die nachfolgend genannten Optionen können Sie auswählen:

Checkboxes!	Die Einträge sind mit Kontrollfeldern versehen. Benutzen Sie diese Option in Verbindung mit WP-Chars!.
ExtendedSelection!	Markieren eines Blockes von Einträgen. Klicken Sie auf dem ersten Eintrag, halten Sie [Umschalten] gedrückt, klicken Sie auf einem weiteren Eintrag.
MultiColumn!	Die Listeneinträge werden spaltenweise angezeigt.
MultipleSelection!	Mehrere Einträge können markiert werden. Durch erneutes Klicken auf einem markierten Eintrag wird die Markierung wieder gelöscht.
NameSearch!	Die Suche nach Einträgen ist erlaubt. Je mehr Zeichen Sie eingeben, um so näher kommen Sie dem gesuchten Eintrag.
Sorted!	Anzeige der Einträge in alphabetischer Reihenfolge. Beim Eintippen eines Zeichens wird der Eintrag markiert, der mit diesem Zeichen beginnt.
Unsorted!	Die Einträge werden in der definierten Reihenfolge angezeigt. Beim Eintippen eines Zeichens wird der Eintrag markiert, der mit diesem Zeichen beginnt.
WPChars!	Zeichen aus den WordPerfect-Zeichensätzen sind erlaubt.

Variable Der ausgewählte Listeneintrag wird in *Variable* gespeichert.

Beispiel: Die folgenden Anweisungen erzeugen das angezeigte Dialogfeld.

```
Application          (A1;"WordPerfect";Default;"DE")
Anzeige=             "Birnen"
DialogDefine         (100; 50; 50; 200; 100; OK! | Cancel! | Percent!;
                     "Beispiel für DialogAddListBox")
DialogAddListBox     (100; 1000; 10; 10; 175; 55; Sorted! | NameSearch!; Anzeige)
DialogAddListItem    (100; 1000; "Äpfel")
DialogAddListItem    (100; 1000; "Bananen")
DialogAddListItem    (100; 1000; "Birnen")
DialogAddListItem    (100; 1000; "Orangen")
DialogAddListItem    (100; 1000; "Zitronen")
DialogDisplay        (100; 1000)
DialogDestroy        (100)
```

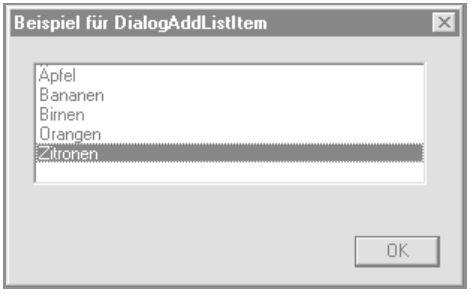
Nach der Anzeige des Dialogfeldes wird der unter *Variable* angegebene Text durch einen Cursorbalken markiert. Mit den Pfeiltasten können Sie diesen Balken zur Auswahl nach oben oder nach unten bewegen. Sie können einen Eintrag natürlich auch mit der Maus auswählen. Über den Style-Parameter können Sie festlegen, ob die Einträge in sortierter Reihenfolge erscheinen oder in der Reihenfolge, in der die Einträge unter *DialogAddListItem* definiert wurden.

Achten Sie darauf, daß die ControlIDs von *DialogAddListBox* und die in den zugehörigen Objekten *DialogAddListItem* identisch sein müssen (hier: 1000).

Weitere Hinweise siehe: DialogAddListItem

6.37.16 DialogAddListItem

Befehlsform: **DialogAddListItem**(*DialogId;ControlId;String*)



Definieren von Listeneinträgen, die in Verbindung mit den Befehlen *DialogAddComboBox*, *DialogAddListBox* und *DialogAddPopupButton* verwendet werden können. Schlagen Sie zu weiteren Informationen bitte bei diesen Befehlen nach. Jeder anzuzeigende Eintrag muß separat definiert werden.

Parameter

String Textstring der gewünschten Listeneinträge.

Beispiel: Die folgenden Anweisungen erzeugen das angezeigte Dialogfeld.

```
Application          (A1;"WordPerfect";Default;"DE")
Anzeige=             "Zitronen"
DialogDefine         (100; 50; 50; 200; 100; OK! | Cancel! | Percent!;
                     "Beispiel für DialogAddListItem")
DialogAddListBox     (100; 1000; 10; 10; 175; 55; 0; Anzeige)
DialogAddListItem    (100; 1000; "Äpfel")
DialogAddListItem    (100; 1000; "Bananen")
DialogAddListItem    (100; 1000; "Birnen")
DialogAddListItem    (100; 1000; "Orangen")
DialogAddListItem    (100; 1000; "Zitronen")
DialogDisplay        (100; 1000)
DialogDestroy        (100)
```

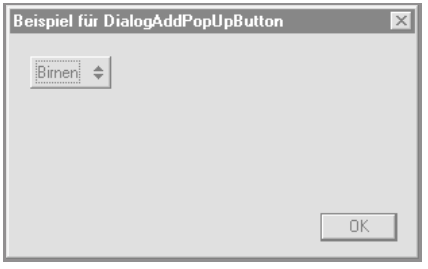
Nach der Anzeige des Dialogfeldes wird der unter *Variable* angegebene Text durch einen Cursorbalken markiert. Mit den Pfeiltasten können Sie diesen Balken zur Auswahl nach oben oder nach unten bewegen. Sie können einen Eintrag natürlich auch mit der Maus auswählen. Über den Style-Parameter können Sie festlegen, ob die Einträge in sortierter Reihenfolge erscheinen oder in der Reihenfolge, in der die Einträge unter *DialogAddListItem* definiert wurden.

Achten Sie darauf, daß die ControlIDs von *DialogAddListBox* und die in den zugehörigen Objekten *DialogAddListItem* identisch sein müssen (hier: 1000).

Weitere Hinweise siehe: DialogAddListBox, DialogAddPopupButton

6.37.17 DialogAddPopupButton

Befehlsform: **DialogAddPopupButton**(*DialogId;ControlId;HPos;VPos;Breite;Höhe; Variable*)



Definieren einer PopUp-Taste mit den Listeneinträgen, die unter *DialogAddListItem* angegeben wurden. Der ausgewählte Eintrag wird in *Variable* gespeichert. Wird dieser Variablen vor dem Einblenden des Dialogfeldes ein String zugeordnet, wird dieser String auf der PopUp-Taste beim erstmaligen Aufruf vorgeschlagen.

Nach dem Klicken auf dieser Taste werden die verfügbaren Einträge angezeigt. Ziehen Sie zur Auswahl den Mauszeiger auf den betreffenden Eintrag. Buchstaben, denen innerhalb der Einträge ein & vorausgeht, werden in der Anzeige unterstrichen und somit als Menü-Kennbuchstabe zur Auswahl angezeigt.

Parameter

Variable Der ausgewählte Listeneintrag wird in *Variable* als Zeichen-Ausdruck gespeichert.

Beispiel: Die folgenden Anweisungen erzeugen das angezeigte Dialogfeld.

```
Application          (A1;"WordPerfect";Default;"DE")
Anzeige=             "B&irnen"
DialogDefine         (100; 50; 50; 200; 100; OK! | Cancel! | Percent!;
                                        "Beispiel für DialogAddPopUpButton")
DialogAddPopUpButton (100; 1000; 10; 10; 40; 15; Anzeige)
DialogAddListItem    (100; 1000; "&Äpfel")
// Das Zeichen, vor dem ein & steht, wird unterstrichen angezeigt (= Menükennbuchstabe).
DialogAddListItem    (100; 1000; "&Bananen")
DialogAddListItem    (100; 1000; "&Birnen")
DialogAddListItem    (100; 1000; "&Orangen")
DialogAddListItem    (100; 1000; "&Zitronen")
DialogDisplay        (100; 1000)
DialogDestroy        (100)
```

Achten Sie darauf, daß die ControlIDs von *DialogAddPopUpButton* und die in den zugehörigen Objekten *DialogAddListItem* identisch sein müssen (hier: 1000).

Weitere Hinweise siehe: DialogAddListBox, DialogAddPushButton

6.37.18 DialogAddProgress

Befehlsform: **DialogAddProgress**(*DialogId;ControlId;HPos;VPos;Breite;Höhe*)

Anzeigen eines Laufzeitanzeigers, dem entnommen wird, wie weit eine bestimmte Verarbeitung fortgeschritten ist. Wurde ein Dialogfeld mit dem Dialogeditor erstellt, kann über diesen

Befehl nach dem Anzeigen des Dialogfelds ein Laufzeitanzeiger ergänzt werden. Laufzeitanzeiger kennen Sie z. B. von der Installation von Programmen: In einem Zeitstrahl wird prozentual angezeigt, wieviel der Arbeit bereits erledigt ist.

Weitere Informationen und Beispiel finden Sie in Abschnitt »6.100 RegionSetProgressPercent«.

Beispiel

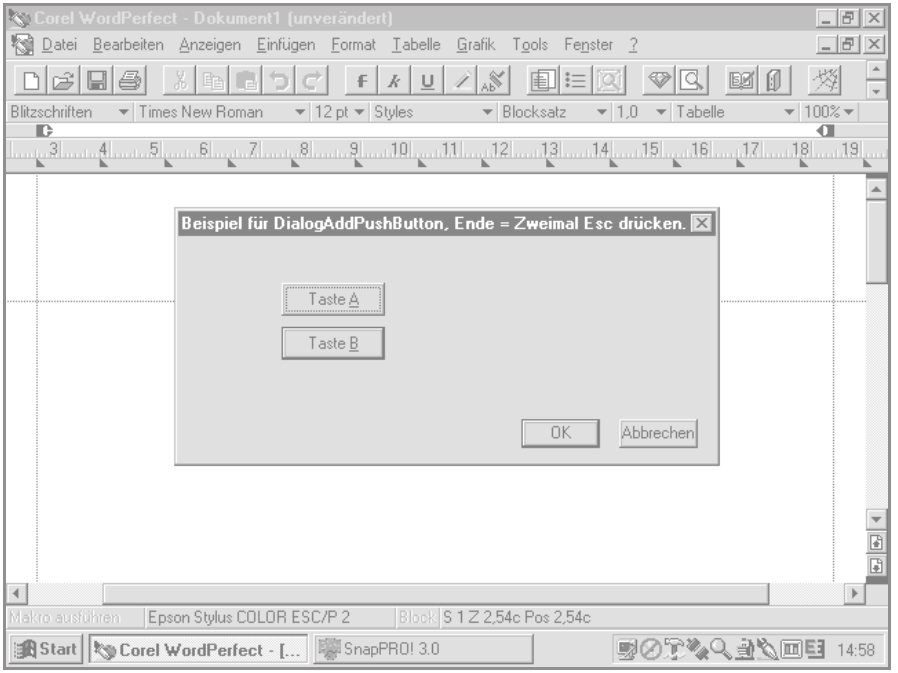
```
Application                    (A1;"WordPerfect";Default;"DE")
DialogDefine                   ("DO1";50;50;200;100;OK! | Cancel!;"AddProgress-Beispiel")
DialogAddPushButton            ("DO1";"Taste1";100;35;90;14;;"Statusanzeiger einblenden")
DialogAddPushButton            ("DO1";"Taste2";100;55;90;14;;"Statusanzeiger starten")
DialogSetProperties            ("DO1";"Arial";"15p")
DialogShow                     ("DO1";"WordPerfect";Anzeigen)
CallbackWait
Quit
Label                          (Anzeigen)
If                             (Anzeigen[5]=274)
                               DialogDestroy("DO1")
       CallbackResume
       Return
EndIf
Switch (Anzeigen[3])
       CaseOf "OKBttn":        MessageBox(x;"OK-Taste";" OK wurde gedrückt")
       CaseOf"CancelBttn":     DialogDestroy("DO1")
                               CallbackResume
                               Return
       CaseOf"Taste1":         DialogAddProgress("DO1";"Prog";10;10;180;15)
       CaseOf"Taste2":         ForNext (Prozent;1;100;1)
                                  RegionSetProgressPercent("DO1.Prog";Prozent)
                                  Wait (1)
                               EndFor
EndSwitch
Return
```

Dieses Beispiel finden Sie auf der CD-ROM unter dem Dateinamen DIAPROGR.WCM

Weitere Hinweise siehe: RegionSetProgressPercent

6.37.19 DialogAddPushButton

Befehlsform: **DialogAddPushButton**(*DialogId;ControlId;HPos;VPos;Breite;Höhe;Style;Text*)



Definieren einer beliebig großen Auswahltaste in Verbindung mit einem Hinweistext. Durch Klicken auf dieser Taste wird die Funktion ausgewählt, die auf der Taste angegeben ist. Ist die Taste zu klein für den anzuzeigenden Text, müssen Sie entweder den Text kürzen, oder die Maße der Taste verändern.

Parameter

Style Dialogfeld-Option.

CancelBttn! Taste mit der Aufschrift *Abbrechen*.

	DefaultBttn!	Taste wird aktiviert, d. h. beim Drücken von [Eingabe] wird die Funktion ausgewählt und das Dialogfeld geschlossen.
	HelpBttn!	Taste mit der Aufschrift *Hilfe*.
	NonDefaultBttn!	Taste ist nicht aktiviert.
	OKBttn!	Taste mit der Aufschrift *OK*.

Text Text, der auf einer Taste erscheinen soll.

Beispiel: Die folgenden Anweisungen erzeugen das angezeigte Dialogfeld.

```
Application          (A1; "WordPerfect"; Default; "DE")
DialogDefine         (100; 50; 50; 260; 100; OK! | Cancel! | Percent!;
                     "Beispiel für DialogAddPushButton, Ende = 2 x Esc drücken.")
DialogAddPushButton  (100; 0; 50; 20; 50; 15; NonDefaultBttn!; "Taste &A")
TasteA=              MacroDialogResult
DialogAddPushButton  (100; 0; 50; 40; 50; 15; DefaultBttn!; "Taste &B")
TasteB=              MacroDialogResult
DialogDisplay        (100; TasteA)
Switch               (MacroDialogResult)
  CaseOf  TasteA:    Prompt("Ergebnis";"Taste A wurde gedrückt.";;;)
  CaseOf  TasteB:    Prompt("Ergebnis";"Taste B wurde gedrückt.";;;)
  CaseOf  1:         Prompt("Ergebnis";"OK-Taste wurde gedrückt.";;;)
  CaseOf  2:         Prompt("Ergebnis";"Abbrechen-Taste wurde gedrückt.";;;)
EndSwitch
Wait                 (30)
EndPrompt
DialogDestroy        (100)
```

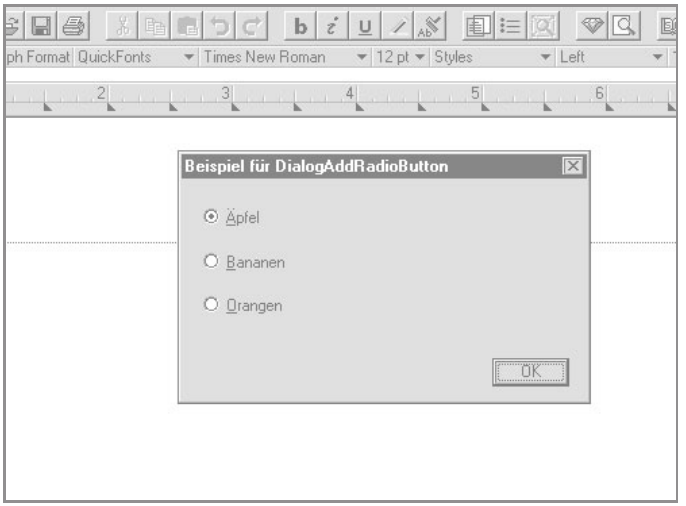
DialogDestroy darf erst dann erfolgen, wenn die Variable *MacroDialogResult* nicht mehr benötigt wird, es sei denn, der Inhalt von *MacroDialogResult* wurde vorher in eine andere Variable »gerettet«. Unter *Switch* ist dann der Name dieser Variablen anzugeben. Wenn Sie dieses Beispiel testen, beenden Sie den Test bitte durch zweimaliges Drücken von [Esc]. *MacroDialogResult* ist eine von WordPerfect fest vorgegebene Variable (kein Befehl!), in der gespeichert wird, welche Taste gedrückt wurde. In Verbindung mit *DialogDisplay* kann angegeben werden, welche Taste als Default-Taste angezeigt wird (diese enthält um den Text eine gestrichelte Linie und wird auch beim Drücken durch [Eingabe] ausgelöst).

Weitere Hinweise siehe: DialogAddPopupButton, MacroDialogResult.

6.37.20 DialogAddRadioButton

Befehlsform: **DialogAddRadioButton**(*DialogId;ControlId;HPos;VPos;Breite;Höhe;Text;Variable;Style*)

Erstellen eines Optionsfeldes mit zugehörigem Text. Den anzuzeigenden Text müssen Sie unter dem Parameter *Text* angeben. Nach dem Klicken in einem Optionsfeld wird dieses je nach



dem derzeitigen Zustand aktiviert oder deaktiviert, d. h. es kann ein- oder ausgeschaltet werden:

Option aktiv:	⊙
Option inaktiv:	○

In dem unter *Variable* definierten Datenfeld wird nach dem Klicken auf [*OK*] der zu diesem Zeitpunkt aktuelle Zustand des Optionsfeldes gespeichert (siehe unten). Wird dieser Variablen vor dem Einblenden des Dialogfeldes der Wert 1 zugeordnet, wird das zugehörige Optionsfeld beim erstmaligen Aufruf aktiviert. Wenn Sie mehrere Gruppen von Optionsfeldern in einem Dialogfeld verwenden wollen, müssen Sie diese Gruppen jeweils mit dem Befehl *DialogAddGroupBox* zusammenfassen, damit je Gruppe immer nur ein Optionsfeld aktiviert werden kann.

Parameter

Text Text, der hinter dem Optionsfeld erscheinen soll.

Variable In dieser Variablen wird nach dem Klicken auf [*OK*] der aktuelle Zustand des zugehörigen Optionsfeldes gespeichert. Wenn das Dialogfeld über [*OK*] verlassen wird, enthält diese Variable entweder 0 oder 1. Wurde das Optionsfeld aktiviert, wird 1 gesetzt, ansonsten 0.

Style Folgende Parameter stehen zur Verfügung:

Radio!	Optionsfeld ist nicht markiert (siehe auch "RegionSetCheck«).
RadioAuto!	Normales Optionsfeld.
RadioLeft!	Zugehöriger Text wird links vom Optionsfeld angezeigt.

Beispiel: Die folgenden Anweisungen erzeugen das angezeigte Dialogfeld.

```
Application           (A1;"WordPerfect";Default;"DE")
Taste1=               1
Taste2=               0
Taste3=               0
DialogDefine          (100; 50; 50; 200; 100; OK! | Cancel! | Percent!;
                      "Beispiel für DialogAddRadiobutton")
DialogAddRadioButton  (100; 1000; 10; 10; 50; 15; "&Äpfel"; Taste1)
DialogAddRadioButton  (100; 1001; 10; 30; 50; 15; "&Bananen"; Taste2)
DialogAddRadioButton  (100; 1002; 10; 50; 50; 15; "&Orangen"; Taste3)
DialogDisplay         (100; 1)
DialogDestroy         (100)
```

In diesem Makro wurde *Taste1* auf 1 gesetzt (aktiviert), so daß nach der Anzeige des Dialogfeldes das Optionsfeld *Äpfel* aktiviert wird. Werden mehrere Optionsfelder definiert, können Sie über die jeweils zugehörige Variable angeben, welches Optionsfeld bei der Anzeige des Dialogfeldes aktiviert sein soll. Bei Optionsfeldern innerhalb einer Gruppenbox, die mit *DialogAddGroupBox* definiert wurde, kann immer nur ein Feld markiert werden. Dieses wird dann wie üblich mit einem schwarzen Punkt gekennzeichnet. Lassen sich mehrere Felder gleichzeitig markieren, überlagern sich evtl. zwei oder mehrere Optionsfelder. Überprüfen Sie in diesem Fall die Definition des Dialogfeldes und der einzelnen Objekte.

Mit der Tabulatortaste können Sie vorwärts von Optionsfeld zu Optionsfeld springen, durch Drücken von [Umschalten-Tab] können Sie rückwärts springen. Zum Aktivieren oder Deaktivieren eines Feldes klicken Sie auf dem entsprechenden Optionsfeld.

Weitere Hinweise siehe: DialogAddRadioButton, DialogAddGroupBox

6.37.21 DialogAddScrollBar

Befehlsform: **DialogAddScrollBar**(*DialogId;ControlId;HPos;VPos;Breite;Höhe;Style;Variable;Minimum;Maximum;Abstand*)

Definieren einer horizontalen oder einer vertikalen Bildlaufleiste. Die ausgewählte Position des Bildlauffeldes wird in *Variable* gespeichert. Wird dieser Variablen vor dem Einblenden

des Dialogfeldes ein numerischer Wert zugeordnet, wird das Bildlauffeld auf der betreffenden Position der Bildlaufleiste beim erstmaligen Aufruf des Dialogfeldes angezeigt.

Parameter

Style Die nachfolgend genannten Optionen können Sie auswählen. Nicht alle Kombinationen sind möglich.

Left! / Top!	Das Bildlauffeld wird in Abhängigkeit der unter *VScroll/HScroll* getroffenen Auswahl links oder oben angezeigt. Ein Wert in *Variable* überschreibt diese Auswahl.
Right! / Bottom!	Das Bildlauffeld wird in Abhängigkeit der unter *VScroll/HScroll* getroffenen Auswahl rechts oder unten angezeigt. Ein Wert in *Variable* überschreibt diese Auswahl.
HScroll!	Bildlaufleiste wird horizontal angezeigt.
VScroll!	Bildlaufleiste wird vertikal angezeigt.

Variable Der ausgewählte Wert wird in *Variable* gespeichert.

Minimum Niedrigster Wert des Zählers.

Maximum Höchster Wert des Zählers.

Abstand Abstand für Zwischenwerte.

Beispiel: Die folgenden Anweisungen erzeugen das angezeigte Dialogfeld.

```
Application          (A1;"WordPerfect";Default;"DE")
BLeiste=             2
DialogDefine         (100; 50; 50; 200; 100;OK! |Cancel!|Percent!;
                     "Beispielfür DialogAddScrollbar")
DialogAddScrollBar   (100; 1000; 20; 20; 130; 15; Top! | Left! | HScroll!; BLeiste; 1; 16)
DialogDisplay        (100;1000)
DialogDestroy        (100)
```

In diesem Beispiel wird die Bildlaufleiste oben in horizontaler Position angezeigt. In dem Style-Parameter wurde diese Auswahl durch Top! | Left! | HScroll! getroffen. Das Bildlauffeld können Sie mit der Maus in beliebiger Richtung verschieben. In *Variable* wird nach dem Klikken auf [*OK*] die aktuelle Position des Bildlauffeldes gespeichert.

6.37.22 DialogAddText

Befehlsform: **DialogAddText**(*DialogId;ControlId;HPos;VPos;Breite;Höhe;Style;Text*)

Mit diesem Befehl können Sie einen beliebigen Text in einem Dialogfeld anzeigen. Dieser Text sollte in erster Linie als Bedienungshinweis oder als Erklärung verwendet werden. Er kann während der Anzeige nicht verändert werden.

Parameter

Style Die nachfolgend genannten Optionen können Sie auswählen. Nicht alle Kombinationen sind möglich.

Center!	Zentrierter Text, wenn er nicht in Verbindung mit WP-Zeichen verwendet wird.
Filename!	Pfad- und Dateiname werden in Abhängigkeit des Width-Parameters angezeigt.
Left!	Linksbündig ausgerichteter Text.
Multiline!	Textumbruch, wenn WPChars! verwendet wird.
NoPrefix!	Das Zeichen & wird angezeigt und wird nicht dazu benutzt, den unmittelbar folgenden Buchstaben als Menükennbuchstaben zu kennzeichnen.
RecessBox!	Text wird in einer Vertiefung angezeigt (3D-Effekt).
Right!	Rechtsbündig ausgerichteter Text, wenn er nicht in Verbindung mit WP-Zeichen verwendet wird.
ShadowBox!	Text wird in einer Box mit schattiertem Hintergrund angezeigt (3D-Effekt).
WPChars!	Zeichen der WordPerfect-Zeichensätze können verwendet werden.

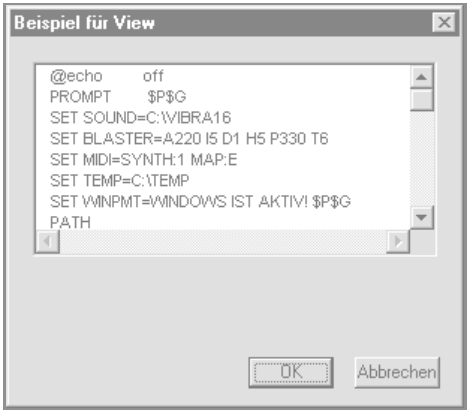
Text Text, der in dem Dialogfeld angezeigt werden soll.

Beispiel: Die folgenden Anweisungen erzeugen das angezeigte Dialogfeld.

```
Application         (A1;"WordPerfect";Default;"DE")
DialogDefine        (100; 50; 50; 200; 100; OK! | Cancel! | Percent!; "Beispiel für
                                                          DialogAddText")
DialogAddText       (100; 1001; 10; 10; 75; 15; RecessBox!; "Feld mit scheinbarer
                                                          Vertiefung")
DialogAddText       (100; 1002; 10; 30; 75; 15; ShadowBox!; "Feld mit Schattierung")
DialogDisplay       (100; 1)
DialogDestroy       (100)
```

6.37.23 DialogAddViewer

Befehlsform: **DialogAddViewer**(*DialogId;ControlId;HPos;VPos;Breite;Höhe; Dateiname*)



Mit diesem Befehl können Sie ein Dialogfeld zum Anzeigen von Dateiinhalten erstellen. Mit Hilfe von vertikalen und horizontalen Bildlaufleisten, die ggf. automatisch eingefügt werden, können Sie innerhalb des angezeigten Textes blättern. Unter *Variable* ist der Name der Datei anzugeben, die angezeigt werden soll.

Dies kann ein fixer Name sein oder eine Variable, der an anderer Stelle des Makros ein Dateiname zugeordnet wurde, z. B. über *DialogAddFileName*.

Parameter

Dateiname Name der anzuzeigenden Datei.

Beispiel: Die folgenden Anweisungen erzeugen das angezeigte Dialogfeld.

```
Application         (A1;"WordPerfect";Default;"DE")
DialogDefine        (100; 50; 50; 200; 100;  OK! | Cancel! | Percent!; "Beispiel für View")
```

```
DialogAddViewer      (100; 1000; 10; 10; 75; 50; "c:\autoexec.bat")
DialogDisplay        (100; 1)
DialogDestroy        (100)
```

Weitere Hinweise siehe: DialogAddFileName

6.37.24 DialogAddVLine

Befehlsform: **DialogAddVLine**(*DialogId;ControlId;HPos;VPos;Länge*)

Definieren einer vertikalen Linie, die innerhalb eines Dialogfeldes angezeigt werden kann, um z. B. größere Dialogfelder optisch zu untergliedern und dadurch übersichtlicher zu gestalten.

Parameter

Länge Länge der Linie in Dialogeinheiten.

Beispiel: Die folgenden Anweisungen erzeugen das angezeigte Dialogfeld.

```
Application          (A1;"WordPerfect";Default;"DE")
DialogDefine         (100; 50; 50; 200; 100; OK! | Cancel! | Percent!; "Beispiel für HLine und
                                                                                    VLine")
DialogAddHLine       (100; 1000; 10; 10; 175)
DialogAddVLine       (100; 1001; 100; 15; 50)
DialogDisplay        (100; 1)
DialogDestroy        (100)
```

In diesem Beispiel wird eine vertikale und eine horizontale Linie angezeigt.

Weitere Hinweise siehe: DialogAddHLine

6.37.25 DialogDefine

Befehlsform: **DialogDefine**(*DialogId;HPos;VPos;Breite;Höhe;Style;Titel*)

Mit *DialogDefine* wird einem zu definierenden Dialogfeld eine eindeutige Kennung (= DialogId) zugeordnet, die in allen zu diesem Dialogfeld gehörenden Dialog-Befehlen verwendet werden muß. Zur Benennung sind maximal 25 Zeichen erlaubt. Zusätzlich werden die Position, die Größe, die Funktion und die Gestaltung des Dialogfeldes definiert. Die Definition ist nur dann erforderlich, wenn Sie nicht mit dem Dialogeditor arbeiten wollen.

Mit dieser Definition wird ein leeres Dialogfeld (nur ein Rahmen, ein Titel für das Dialogfeld und evtl. bestimmte Tasten) definiert. In diesem Dialogfeld können Sie dann mit den anderen Dialog-Befehlen weitere Objekte wie Kontrollfelder, Optionsfelder, Texteingabefelder usw. ergänzen. In einem Makro können beliebig viele Dialogfelder definiert werden, zu einer Zeit kann aber immer nur ein Dialogfeld aktiviert werden. Über diesen Befehl wird das Dialogfeld nur im Speicher generiert. Damit es am Bildschirm erscheint, muß es über den Befehl *Dialog-Display* oder *DialogShow* angezeigt werden. Zum Ausblenden wird der Befehl *DialogDestroy* verwendet.

Die zum Ausblenden gedrückte Taste wird in der Variablen *MacroDialogResult* gespeichert. Weitere Informationen hierzu finden Sie unter Abschnitt »6.73 MacroDialogResult«.

Parameter

DialogId Name oder Nummer des aktuellen Dialogfeldes (numerischer Wert oder Zeichen-Ausdruck). Alle Dialog-Befehle, die Bezug auf dieses Dialogfeld nehmen, müssen dieselbe Kennung haben.

Hpos Horizontale Position des Dialogfeldes, gemessen in Dialogeinheiten oder in prozentualer Angabe der Bildschirmgröße, jeweils vom linken Rand des Bildschirms bis zum linken Rand des Dialogfeldes.

Vpos Vertikale Position des Dialogfeldes, gemessen in Dialogeinheiten oder in prozentualer Angabe der Bildschirmgröße, jeweils vom oberen Rand des Bildschirms bis zum oberen Rand des Dialogfeldes.

Style Die nachfolgend genannten Optionen können Sie auswählen.

	OK!	Anzeigen einer OK-Taste in dem Dialogfeld.
	Cancel!	Anzeigen einer Abbrechen-Taste in dem Dialogfeld.
	Percent!	Setzt eine Maßeinheit in prozentualer Größe des Bildschirms. Dieser Wert steht immer in Verbindung mit den Parameter *HPos* und *VPos*. Wenn diese Funktion verwendet wird, wird *HPos* und *VPos* auf jeweils 50% gesetzt, so daß das Dialogfeld in der Mitte des Bildschirms erscheint.
	NoFrame!	Dialogfeld wird ohne Rahmen angezeigt (hat keine Auswirkung in WIN 95).
	NoTitle!	Titelzeile wird nicht angezeigt.
	Sizeable!	Die Größe des Dialogfelds kann mit den üblichen Windows-Funktionen variiert werden.
	Modeless!	Das Dialogfeld braucht vor der Auswahl einer neuen Task nicht ausgeblendet zu werden. Wenn diese Funktion nicht verwendet wird, muß das Dialogfeld vor der Ausführung einer neuen Task ausgeblendet werden.
	Enter2HRtn!	Wird in einem mehrzeiligen Textfenster (DialogAddEditBox) [Eingabe] gedrückt, springt der Cursor an den Beginn einer neuen Zeile. Es wird dabei nicht die Standard-Schaltfläche aktiviert.
	NoCloseBox!	

Titel Text, der in der Titelleiste erscheinen soll.

Beispiel: Die folgenden Anweisungen erzeugen das angezeigte Dialogfeld.

```
Application         (A1;"WordPerfect";Default;"DE")
DialogDefine        (100; 50; 50; 200; 100; OK! | Cancel! | Percent!; "Beispiel für
                                                        DialogDefine")
DialogDisplay       (100; 1)
DialogDestroy       (100)
```

Anstelle der Ganzzahlen können auch Variable mit ganzzahligem Inhalt oder Variable mit Offset (siehe Breite) verwendet werden:

```
Application         (A1;"WordPerfect";Default;"DE")
HPos=               50
VPos=               50
Breite=             100                 //  Wird zusätzlich durch einen Offset ergänzt.
Höhe=               100
Style=              19                  //  = Summe der einzelnen Styles (bis Vers. 6.0a)
Text=               "Beispiel für DialogDefine"
DialogDefine        (100; HPos; VPos; Breite+50; Höhe; Style; Text)
DialogDisplay       (100; 1)
```

```
DialogDestroy        (100)
```

Die Dimensionen eines Dialogfeldes können hierdurch variabel gehalten werden, da die Werte während des Makroablaufs bei Bedarf geändert werden können. Dasselbe gilt natürlich auch für die zu definierenden Objekte. In diesem Beispiel wird für die Breite die Position 150 ermittelt.

Die Erstellung des Dialogfeldes über *DialogDefine* kostet Zeit, was sich besonders bei langsamen Computern bemerkbar macht. Wird ein Dialogfeld während eines Makroablaufs nicht verändert, braucht es nur einmal erstellt zu werden, sofern nicht unterschiedliche Dialogfelder mit demselben Namen (DialogId) verwendet werden. Erstellen Sie darum für die Definition des Dialogfeldes eine Unterroutine. Rufen Sie diese Unterroutine am Beginn des Makros zur Erstellung des Dialogfeldes auf. Soll das Dialogfeld später angezeigt werden, genügt an der betreffenden Stelle des Makros nur der Befehl *DialogDisplay*, um das Dialogfeld anzuzeigen. Die Zeitersparnis wirkt sich besonders dann aus, wenn das Anzeigen öfters erforderlich ist, z. B. innerhalb einer Schleife. In diesem Fall dürfen Sie hinter *DialogDisplay* kein *DialogDestroy* ausführen.

Weitere Hinweise siehe: DialogDestroy, DialogDisplay

6.37.26 DialogDestroy

Befehlsform: **DialogDestroy**(*DialogId*)

Mit *DialogDestroy* wird das Dialogfeld mit der angegebenen Kennung einschließlich aller zugehörigen Definitionen im Speicher gelöscht bzw. ausgeblendet. Verwenden Sie diesen Befehl auch, wenn Sie ein neues Dialogfeld unter demselben Namen definieren und anzeigen möchten, um das vorherige, gleichnamige zu löschen. Ansonsten wird eine Fehlermeldung angezeigt.

Parameter

DialogId Name oder Nummer des zu löschenden Dialogfeldes.

```
Application          (A1;"WordPerfect";Default;"DE")
DialogDefine         (100; 50; 50; 200; 100; OK! | Cancel! | Percent!; "Beispiel für
                                                                    DialogDestroy")
DialogDisplay        (100; 1)
DialogDestroy        (100)
```

In diesem Beispiel wird das Dialogfeld mit der DialogId 100 im Speicher gelöscht und am Bildschirm ausgeblendet.

Weitere Hinweise siehe: DialogDefine, DialogDisplay

6.37.27 DialogDisplay/DialogUndisplay

Befehlsform: **DialogDisplay**(*DialogId;ControlId;Label*)
DialogUndisplay(*DialogId;ControlId*)

Das unter *DialogId* angegebene Dialogfeld wird am Bildschirm angezeigt (= DialogDisplay) oder ausgeblendet (= DialogUndisplay). Achten Sie bitte darauf, daß *DialogDisplay* erst dann ausgeführt werden darf, wenn alle zugehörigen Objekte definiert wurden.

Parameter

DialogId Name oder Nummer des anzuzeigenden Dialogfeldes. Diese muß mit dem/der unter *DialogDefine* definierten Namen oder Nummer übereinstimmen.

ControlId Nummer oder Name eines definierten Objekts innerhalb eines Dialogfeldes. Hier können Sie außerdem steuern, ob bei der Anzeige eines Dialogfeldes die Taste [*OK*], die Taste [*Abbrechen*] oder eine benutzerdefinierte Taste als Default vorgeschlagen werden soll (sofern sie angezeigt werden):

1	Taste [*Abbrechen*] wird vorgeschlagen.
2	Taste [*OK*] wird vorgeschlagen.
Variable	Variable, die unmittelbar einem *DialogAddPushButton* in Verbindung mit *MacroDialogResult* folgt.

Label Angabe des Namens einer Unterroutine, die bei folgenden Aktionen aufgerufen werden soll:

- Beim Drücken von [Alt-F4].
- Beim Doppelklicken auf dem Systemmenüfeld.
- Bei der Auswahl von [*Schließen*] in dem Systemmenüfeld.
- Bei der Auswahl einer Befehlstaste, eines Kontrollfeldes, eines Optionsfeldes oder eines Hotspots.

Wird die Label-Option nicht verwendet, wird das Dialogfeld so lange angezeigt, bis z. B. auf einer OK-, einer Abbrechen-Taste oder auf einem Hotspot geklickt wird.

Die Verwendung eines Labels ist wahlfrei. Die über *Label* aufgerufene Unterroutine muß mit dem angegebenen Labelnamen beginnen und mit einem *Return* enden. Wird diese Option verwendet, wird das Dialogfeld so lange angezeigt, bis *DialogUndisplay* ausgeführt wird. Um das zu überprüfen, sollte unmittelbar hinter *DialogDisplay* eine *While*-Schleife verwendet werden, die eine Bedingung prüft, die in der unter *Label* angegebenen Unterroutine gesetzt wurde:

Beispiel:

```
DialogDisplay       (100;1;Prüfen)
While               (Feld1=0)
EndWhile
::::
DialogDestroy       (100)
Label               (Prüfen)
::::
Feld1=              1
DialogUndisplay     (100)
Return
```

Beim Aufrufen der Unterroutine wird automatisch ein Array erstellt, in dem Informationen über die vom Benutzer durchgeführten Aktionen gespeichert werden. Hier müssen jetzt aufgrund dieser Informationen Entscheidungen getroffen werden (z. B. Ausblenden des Dialogfeldes, wenn auf Abbrechen geklickt wurde). Die Elementnamen des Arrays werden aufgrund des unter *Label* angegebenen Namens generiert. Bei dem Labelnamen *Prüfen* heißen die Array-Elemente darum *Prüfen[1]*, *Prüfen[2]* usw. Die Elemente können folgende Eintragungen enthalten (anstelle von *Array* muß der unter Label angegebene Name der Unterroutine verwendet werden):

Array[1]=3 Dialogfeld-Typ.

Array[2]=DialogId Name oder Nummer des betreffenden Dialogfeldes (muß mit den Angaben von *DialogDisplay* übereinstimmen).

Array[3]=ControlId Name oder Nummer des Objekts, das den Aufruf der Unterroutine veranlaßte. Enthält dieses Element 1, betrifft das die OK-Taste und bei 2 die Abbrechen-Taste.

Array[4]=DialogHwnd
Hinweis auf das auslösende Dialogfeld.

Array[5]=Message Enthält eine Meldung aufgrund einer im Dialogfeld ausgeführten Aktion:

WM_SYSCOMMAND(274)
Wird verwendet, wenn [Alt-F4], ein Doppelklick auf dem Systemmenüfeld oder [*Schließen*] in dem Systemmenüfeld ausgeführt wurde.

WM_COMMAND(273)
Wird verwendet, wenn eine Befehlstaste, ein Kontrollfeld, ein Optionsfeld, oder ein Hotspot aktiviert wurde.

Array[6]=wParam Wird in Verbindung mit Array[5] verwendet. Benutzen Sie diesen Parameter nur dann, wenn Sie mit der Windows-Programmierung vertraut sind.

Array[7]=lParam Wird in Verbindung mit Array[5] verwendet. Benutzen Sie diesen Parameter nur dann, wenn Sie mit der Windows-Programmierung vertraut sind.

Beispiel 1

Die folgenden Anweisungen erzeugen das oben angezeigte Dialogfeld.

```
Application         (A1;"WordPerfect";Default;"DE")
DialogDefine        (100; 50; 50; 200; 100; OK! | Cancel! | Percent!;
                    "Beispiel für DialogDisplay")
DialogDisplay       (100; 1)
DialogDestroy       (100)
```

Über *DialogDefine* wird ein leeres Dialogfeld definiert, das Sie mit den restlichen Dialog-Befehlen mit Objekten wie Optionsfelder, Kontrollfelder, Tasten usw. füllen können. Dieser Befehl erstellt das Dialogfeld im Speicher des Computers. Über *DialogDisplay* wird das definierte Dialogfeld angezeigt.

Beispiel 2

Das folgende Makro wurde dem WordPerfect-Makrohandbuch (Online-Hilfe) entnommen. Es zeigt ein Dialogfeld so lange an, bis es über [Alt-F4], durch einen Doppelklick auf dem Systemmenüfeld, durch Auswählen von [*Schließen*] aus dem Systemmenüfeld, durch Klicken auf [*OK*] oder [*Abbrechen*] ausgeblendet wird. Beachten Sie bitte die ausgerückten Kommentare.

```
Application          (A1;WordPerfect;Default;"DE")
FileNew              ()                              // Nachrichten in leeres Dokument.
Display              (On!)                           // Makroablauf anzeigen.
WM_SYSCOMMAND=       274                             // Nachricht für Systembefehle.
WM_COMMAND=          273                             // Nachricht für Dialogobjekte.
EditVar=             ""                              // Variable initialisieren.
RadioVar=            ""
Done=                0
DialogDefine         (1000;50;50;150;155;OK! | Cancel! | Percent!;"Test Dialog")
DialogAddEditBox     (1000;100;10;10;100;13;1;EditVar;20)
DialogAddRadioButton (1000;101;10;30;100;10;"Bitte hier auswählen:";RadioVar)
DialogDisplay        (1000;100;Callback)             // Unterroutine heißt Callback.
While                (Done=0)                        // Warteschleife, bis Done=1.
EndWhile
Result=              MacroDialogResult
DialogDestroy        (1000)
Quit
Label                (Callback)                      // Dialog-Unterroutine
If                   ((Callback[5]=WM_SYSCOMMAND) or (Callback[3]=1))
                                                     // Versuch, das Dialogfeld mit eine
```

```
                                                    // Systembefehl oder durch Klicken
                                                    // auf [OK] zu schließen.
   DialogUnDisplay    (1000;1)                      // Dialogfeld ausblenden.
   Done=              1
   Return
Else                                                // Wenn nicht [OK] gewählt wurde.
    If                (Callback[3]=2)               // Wenn [Abbrechen] gewählt wurde.
       DialogUndisplay(1000;2                       // Dialogfeld ausblenden und Abbruch.
       Done=          1
       Return
     EndIf
   Type                ("Auswahl: "+Callback[3])// Andere als [OK] oder [Abbrechen].
   HardReturn          ()
EndIf
Return
```

Weitere Hinweise siehe: DialogDefine, DialogDestroy

6.37.28 DialogDismiss

Befehlsform: **DialogDismiss**(DialogId;ControlId)

Dialogfelder werden über den Befehl *DialogShow* am Bildschirm angezeigt. Nach der Anzeige eines Dialogfeldes können/müssen Sie Daten eingeben, Options- oder ggf. Kontrollfelder aktivieren/deaktivieren usw. Nach der Abarbeitung des Dialogfelds muß dieses wieder geschlossen und ausgeblendet werden. Die Art des Schließens hängt davon ab, ob der zugehörige *DialogShow*-Befehl in Verbindung mit einer Callback-Funktion verwendet wurde oder nicht:

Ohne Callback — Das angezeigte Dialogfeld wird nach dem Anklicken einer beliebigen Taste ausgeblendet. Wird hier mit *DialogDismiss* gearbeitet, wird bei dessen Ausführung eine Fehlermeldung angezeigt.

Mit Callback — Das angezeigte Dialogfeld wird nach der Ausführung von *DialogDismiss* ausgeblendet. Das Schließen erfolgt über eine bestimmte Taste, die Sie in Verbindung mit dem Befehl *DialogDismiss* festlegen können.

Wurden Daten eingegeben, Options- oder Kontrollfelder aktiviert/deaktiviert, können nach dem Ausblenden die jeweils zugehörigen Variablen dementsprechend geprüft und verarbeitet werden, sofern das Dialogfeld nicht mit [*Abbrechen*] geschlossen wurde.

Parameter

DialogId — Name des auszublendenden Dialogfelds.

ControlId — Das Feld enthält den Wert oder den Namen einer Taste, nach deren Drücken das Dialogfeld ausgeblendet werden soll:

OKBttn oder **1** — Das Dialogfeld wird ausgeblendet. Vorgenommene Veränderungen in Textfeldern oder das Aktivieren/Deaktivieren von Options- bzw. Kontroll-

feldern können im Makro geprüft und weiterverarbeitet werden.

CancelBttn oder **2** Das Dialogfeld wird geschlossen. Alle vorgenommenen Änderungen usw. werden nicht übernommen.

Benutzer-Taste (oder ein Hot-Spot). Das Makro verhält sich wie bei *OKBttn* bzw. *1.*

Im Makro muß aufgrund der eingegebenen Daten bzw. markierten Felder in Verbindung mit der gedrückten Taste entsprechend reagiert werden. Welche Taste gedrückt wurde, wird in dem Feld *MacroDialogResult* gespeichert. Weitere Informationen hierzu entnehmen Sie bitte Abschnitt »6.73 MacroDialogResult«.

Beispiel: Siehe unter DialogShow

Weitere Hinweise siehe: DialogDestroy, DialogDisplay, DialogShow

6.37.29 DialogLoad

Befehlsform: *Ergebnis*=**DialogLoad**(*DialogId;Übergeordnet*)

Laden eines Dialogfeldes in den Speichern, ohne das Dialogfeld anzuzeigen.

Parameter

Variable Dient zur Kontrolle, ob das Laden erfolgreich ausgeführt wurde:

HWND	Windows-Handle, wenn erfolgreich geladen.
0	Laden wurde nicht erfolgreich durchgeführt.

DialogId Name des zu speichernden Dialogfelds.

Übergeordnet Name des übergeordneten Fensters wie z. B. *PerfectScript* oder *WordPerfect*.

Beispiel: *Weitere Hinweise siehe:* AppShow, DialogShow

6.37.30 DialogSave

Befehlsform: *Variable*=**DialogSave**(*DialogId;Name*)

Das angegebene Dialogfeld wird in dem aktuellen Makro gespeichert. Das ist z. B. dann sinnvoll, wenn Dialogfelder noch ohne den Dialogeditor erstellt wurden (z. B. unter WP 5.2).

Nach dem Speichern können sie im Dialogeditor bearbeitet werden. Löschen Sie dann die alten *DialogDefine-*, *DialogDisplay-* und *DialogAdd...*-Befehle, und verwenden Sie die neuen Definitionen.

Parameter

Variable	Dient zur Kontrolle, ob das Speichern erfolgreich ausgeführt wurde:	
	1	Speicherung erfolgreich durchgeführt.
	0	Speicherung nicht durchgeführt.
DialogId	Name des zu speichernden Dialogfelds.	
Name	Name des Dialogfelds, unter dem es gespeichert wird. Wird kein Name angegeben, wird das Dialogfeld unter dem aktuellen Dialogfeldnamen gespeichert. Über diesen Namen kann das Dialogfeld im Dialogeditor bearbeitet werden.	

Weitere Hinweise siehe: DoesDialogExist

6.37.31 DialogSetProperties

Befehlsform: **DialogSetProperties**(*DialogId;Schrift;Schriftgröße;ClassName; Hilfedatei;HilfeText*)

Dem unter DialogID angegebenen Dialogfeld können die nachfolgend aufgezählten Attribute zugeordnet werden.

Parameter

Schrift	Schriftart, in der der Text des Dialogfeld angezeigt werden soll.
Schriftgröße	Schriftgröße, z. B. 12p. Wird keine Maßeinheit (hier: p = Punkte) angegeben, wird die unter DefaultUnits angegebene Maßeinheit verwendet.
ClassName	Klassifizierung.
Hilfedate	Name einer mit dem Dialogfeld verknüpften Hilfedatei.
Hilfetext	Teil der Hilfedatei, der zusammen mit dem Dialogfeld angezeigt wird.
Beispiel:	DialogSetProperties ("D01";"Arial";"15p")

Das Dialogfeld »D01« wird in einer Arial-Schrift mit 15 Punkten angezeigt. Das vollständige Beispiel finden Sie unter dem Befehl DialogAddProgress.

6.37.32 DialogShow

Befehlsform: **DialogShow**(*DialogId;Übergeordnet;Label;Focus*)

Dialogfelder werden über den Befehl *DialogShow* am Bildschirm angezeigt. Dieser Befehl hat dieselbe Funktion wie der Befehl *DialogDisplay*. Sind Daten in diesem Dialogfeld anzuzeigen, müssen diese vorher den zugehörigen Variablen übergeben werden. Dasselbe gilt für bestimmte Options- oder Kontrollfelder, die zu aktivieren/deaktivieren sind. Welche Daten übergeben werden können/müssen, entnehmen Sie bitte der Beschreibung des jeweiligen Dialog-Befehls. Beachten Sie bitte auch Kapitel »11.6 Controls (Elemente)«.

Vor der Anzeige des Dialogfelds können/müssen zugehörige Variable mit den Daten versehen werden, die angezeigt werden sollen. Auch das Aktivieren/Deaktivieren von Kontroll- und Optionsfeldern ist möglich. Nach der Anzeige eines Dialogfelds können/müssen Sie Daten eingeben, Options- oder Kontrollfelder aktivieren/deaktivieren usw. Zu einer Zeit kann aber immer nur ein Dialogfeld aktiviert werden. Nach der Abarbeitung des Dialogfelds muß dieses wieder geschlossen und ausgeblendet werden. Das Schließen erfolgt über eine bestimmte Taste, die Sie in Verbindung mit dem Befehl *DialogDismiss* festlegen können. Der Befehl kann nur dann ausgeführt werden, wenn zuvor über *DialogShow* oder *DialogDisplay* ein Dialogfeld eingeblendet wurde. Nach der Ausführung von *DialogDismiss* wird das Dialogfeld ausgeblendet. Dieser Befehl entspricht weitgehend dem Befehl *DialogDestroy*. Wurden Daten eingegeben, Options- oder Kontrollfelder aktiviert/deaktiviert, können nach dem Ausblenden die jeweils zugehörigen Variablen dementsprechend geprüft und verarbeitet werden.

Parameter

Übergeordnet Werden Dialogfelder nach der bisherigen Methode (bis Version 6.0a) erstellt, kann diese Angabe entfallen. Die Angabe ist erforderlich, wenn Dialogfelder mit dem Dialogeditor erstellt werden. Hier ist der Name des übergeordneten Fensters anzugeben, wie z. B. »WordPerfect« wenn das Dialogfeld in WordPerfect angezeigt wird.

Label Angabe des Namens einer Unterroutine, die bei folgenden Aktionen aufgerufen werden soll (CallBack-Routine):

- Beim Drücken von [Alt-F4].
- Beim Doppelklicken auf dem Systemmenüfeld.
- Bei der Auswahl von [*Schließen*] in dem Systemmenüfeld.
- Bei der Auswahl einer Befehlstaste, eines Kontrollfeldes, eines Optionsfeldes, einesHotspots oder einer Bildlaufleiste.

Focus Festlegung der Standard-Schaltfläche.

Wird die *Label*-Option nicht verwendet, wird das Dialogfeld so lange angezeigt, bis z. B. auf einer OK-, einer Abbrechen-Taste oder auf einem Hotspot geklickt wird. Ansonsten reagiert das Makro sofort, ohne das Ausblenden des Dialogfelds abzuwarten.

Die Verwendung eines Callback-Labels ist wahlfrei. Die über ***Label*** aufgerufene Unterroutine muß mit dem angegebenen Labelnamen beginnen und mit einem ***Return*** enden. Wird diese Option verwendet, wird das Dialogfeld so lange angezeigt, bis *DialogUndisplay* oder *DialogDismiss* ausgeführt wird. Um das zu überprüfen sollte unmittelbar hinter *DialogDisplay* bzw. *DialogShow* eine *While*-Schleife oder der Befehl *CallbackWait* verwendet werden, die eine Bedingung prüft, die in der unter *Label* angegebenen Unterroutine gesetzt wurde:

Beispiel

```
DialogShow                    "Dialog1";"WordPerfect";Prüfen)
While                         (Feld1=0)
EndWhile
::::
DialogDismiss                 ("Dialog1";"CancelBttn")
Label                         (Prüfen)
::::
Feld1=                        1
DialogUndisplay               ("Dialog1")
Return
```

Beim Aufrufen der Unterroutine wird automatisch ein Array erstellt, in dem Informationen über die vom Benutzer durchgeführten Aktionen gespeichert werden. Hier müssen jetzt aufgrund dieser Informationen Entscheidungen getroffen werden (z. B. Ausblenden des Dialogfeldes, wenn auf [*Abbrechen*] geklickt wurde). Die Elementnamen des Arrays werden aufgrund des unter *Label* angegebenen Namens generiert. Bei dem Labelnamen *Prüfen* heißen die Array-Elemente darum *Prüfen[1]*, *Prüfen[2]* usw. Die Elemente können folgende Eintragungen enthalten (anstelle von ***Array*** muß der unter Label angegebene Name der Unterroutine verwendet werden):

Array[1]=3 Dialogfeld-Typ.

Array[2]=DialogId Name oder Nummer des betreffenden Dialogfeldes (muß mit den Angaben von *DialogShow* übereinstimmen).

Array[3]=ControlId Name oder Nummer des Elements, das den Aufruf der Unterroutine veranlaßte. Enthält dieses Element OKBttn, betrifft das die OK-Taste und bei CancelBttn die Abbrechen-Taste.

Array[4]=DialogHwnd
Hinweis auf das auslösende Dialogfeld.

Array[5]=Message Enthält eine Meldung aufgrund einer im Dialogfeld ausgeführten Aktion. Die Einträge haben folgende Bedeutung:

6	Das Dialogfeld wird aktiviert (WM_ACTIVATE).
273	Wird verwendet, wenn eine Befehlstaste, ein Kontrollfeld, ein Optionsfeld, oder ein Hotspot aktiviert wurde (WM_COMMAND).
274	Wird verwendet, wenn [Alt-F4], ein Doppelklick auf dem Systemmenüfeld oder [*Schließen*] in dem Systemmenüfeld ausgeführt wurde (WM_SYSCOMMAND).

	276	Der Benutzer klickt auf einer horizontalen Bildlaufleiste (WM_HSCROLL).
	277	Der Benutzer klickt auf einer vertikalen Bildlaufleiste (WM_VSCROLL).

Array[6]=wParam Wird in Verbindung mit Array[5] verwendet. Benutzen Sie diesen Parameter nur dann, wenn Sie mit der Windows-Programmierung vertraut sind.

Array[7]=lParam Wird in Verbindung mit Array[5] verwendet. Benutzen Sie diesen Parameter nur dann, wenn Sie mit der Windows-Programmierung vertraut sind.

wParam und *lParam* haben unterschiedliche Bedeutungen für jede Windows-Nachricht. Weitere Informationen hierzu entnehmen Sie bitte Ihrer Windows-Dokumentation.

Beispiel

```
Application          (A1;WordPerfect;Default;"DE")
FileNew              ()                              // Nachrichten in leeres Dokument.
Display              (On!)                           // Makroablauf anzeigen.
WM_SYSCOMMAND=       274                             // Nachricht für Systembefehle.
WM_COMMAND=          273                             // Nachricht für Dialogobjekte.
TextVar=             ""                              // Variable initialisieren.
OptionsVar=          ""
Fertig=              0
DialogDefine         (1000;50;50;150;155;OK! | Cancel! | Percent!;"Test Dialog")
DialogAddEditBox     (1000;100;10;10;100;13;1;TexttVar;20)
DialogAddRadioButton (1000;101;10;30;100;10;"Bitte hier auswählen:";OptionsVar)
                                                     //         Anstelle der Dialog...

Befehle hätte auch mit dem Dialogeditor ein Dialogfeld definiert werden können, das über
DialogShow angezeigt worden wäre. Dann wäre allerdings hier nicht sichtbar, welche Elemente
definiert worden wären.

DialogShow           (1000;"WordPerfect";Callback)// Unterroutine heißt Callback.
While                (Fertig=0)                      // Warteschleife, bis Fertig:=1.
EndWhile
Result=              MacroDialogResult
DialogDismiss        (1000;1)
Quit
Label                (Callback)                      // Dialog-Unterroutine
If                   ((Callback[5]=WM_SYSCOMMAND) or (Callback[3]=1))
                                                     // Versuch, das Dialogfeld mit einem
                                                     // Systembefehl oder durch Klicken
                                                     // auf [OK] zu schließen.
   DialogDismiss     (1000;1)                        // Dialogfeld ausblenden.
   Fertig=           1
   Return
Else                                                 // Wenn nicht [OK] gewählt wurde.
    If               (Callback[3]=2)                 // Wenn [Abbrechen] gewählt wurde.
       DialogDismiss (1000;2)                        // Dialogfeld ausblenden und Abbruch.
       Fertig=       1
```

```
        Return
      EndIf
   Type              ("Auswahl: "+Callback[3])// Andere als [OK] oder [Abbrechen].
   HardReturn        ()
EndIf
Return
```

Weitere Hinweise siehe: CallbackWait/CallbackResume, DialogDismiss, DialogDisplay

6.37.33 DoesDialogExist

Befehlsform: *Variable*=**DoesDialogExist**(*DialogId*)

Überprüfung, ob das unter »DialogID« angegebene Dialogfeld im aktuellen Makro existiert. Wenn nicht, wird die *Use*-Dateiliste des aktuellen Makros durchsucht (siehe auch unter *Use*).

Parameter

Variable Dient zur Prüfung, ob das Dialogfeld in dem aktuellen Makro oder in der *Use*-Dateiliste existiert:

0	Dialogfeld wurde nicht gefunden.
1	Dialogfeld wurde gefunden.

DialogId Name des zu suchenden Dialogfelds.

Weitere Hinweise siehe: DialogSave

6.38 Dimensions

Befehlsform: **Dimensions** (*Variable;NumWert*)

Über diesen Befehl können Sie sich Informationen über den Aufbau eines Arrays anzeigen lassen. Das Ergebnis zeigt an, aus wieviel Dimensionen ein Array besteht, wieviel Elemente insgesamt vorhanden sind, und wieviel Elemente einer bestimmten Dimension zugeordnet sind. Der Befehl wird besser verstanden, wenn Sie Grundkenntnisse im Umgang von Arrays haben. Schlagen Sie bei Bedarf bitte in Abschnitt »6.34 Declare« nach.

Parameter

Variable Name des Arrays, zu dem Informationen gewünscht werden.

NumWert Parameter zum Anzeigen diverser Informationen:

Keine Angabe	Gesamtzahl der Arrays.

0	Gesamtzahl der Elemente des Arrays.
Dimensions-Nr.	Gesamtzahl der Elemente der betr. Dimension.

Beispiel

```
Declare Gruppen [3;4;5]          Ergebnis:

D1=Dimensions  (Gruppen[])       D1=3      ArrayGruppe hat 3 Dimensionen.
D2=Dimensions  (Gruppen[]; 0)    D2=60     Insgesamt 60 Elemente (3 * 4 * 5).
D3=Dimensions  (Gruppen[]; 1)    D3=3      3 Elemente in Dimension 1.
D4=Dimensions  (Gruppen[]; 2)    D4=4      4 Elemente in Dimension 2.
D5=Dimensions  (Gruppen[]; 3)    D5=5      5 Elemente in Dimension 3.
D6=Dimensions  (Gruppen[]; 4)    D6=0      Keine 4. Dimension vorhanden.
D7=Dimensions  (c)               D7=0      Keine Array (runde Klammern!).
D8=Dimensions  (Firmen[])        D8=0      Array Firmen gibt es nicht.
```

Weitere Hinweise siehe: Declare

6.39 Directory-Befehle

Die nachfolgend beschriebenen Directory-Befehle helfen Ihnen bei der Handhabung von Ordnern, wie z. B. Erstellen, Löschen, Umbenennen usw. Bei der Beschreibung dieser Befehle werden Parameter, die bei allen Befehlen identisch sind, hier anschließend beschrieben.

Boolean Anstelle von ***Boolean*** kann z. B. eine Variable oder eine Bedingung verwendet werden. Dadurch erfolgt eine Rückmeldung, ob der Befehl erfolgreich ausgeführt wurde oder nicht.

False	Ausführung war nicht erfolgreich.
True	Ausführung war erfolgreich.

Schauen Sie sich bei Bedarf auch die Original-WordPerfect-Beispiele im Online-Makrohandbuch unter dem jeweiligen Befehl an.

Ausdruck Anzeigen einer Meldung. Der Befehl kann bei Bedarf weggelassen werden. Unter *Ausdruck* können folgende Anweisungen vorgenommen werden.

NoPrompts!	Keine Anzeige.
Prompts!	Anzeige einer Meldung.

String.. Angabe von alphanumerischen Konstanten oder einer Variablen, die den erforderlichen Eintrag (Ordnernamen) enthalten. Konstanten müssen durch Hochkommata eingeschlossen sein.

Da die Befehlssyntax einfach zu handhaben ist, und die Befehle eindeutig benannt sind, wird auf eine umfangreiche Beschreibung verzichtet. Den folgenden Beispielen können Sie entnehmen, wie die Directory-Befehle zu verwenden sind. Nach der Ausführung eines Befehls wird

in Verbindung mit *Boolean* ein Wert ermittelt, der Auskunft darüber gibt, ob die Ausführung erfolgreich war oder nicht (siehe oben).

Die Beschreibung erfolgt in alphabetischer Reihenfolge der englischen Befehle. Zum schnelleren Auffinden wird in der jeweiligen Überschriftszeile am rechten Rand die deutsche Übersetzung angezeigt.

Nachfolgend wird an drei Beispielen verdeutlicht, wie die Befehle zu verwenden sind. Alle drei Beispiele führen jeweils dieselbe Aktion aus. Beim ersten Beispiel wird eine Variable verwendet, deren Inhalt nach der Ausführung des Befehls geprüft werden muß. Das zweite Beispiel zeigt eine verkürzte Form der Aktion und der Prüfung. Im dritten Beispiel wird eine verneinende Abfrage verwendet.

In den Beispielen wird der Ordner C:\TEST in C\TEST111 umbenannt. Je nachdem, ob das Umbenennen erfolgreich war oder nicht, wird über *MessageBox* eine Meldung angezeigt.

Beispiel

```
Application          (A1; "WordPerfect"; Default; "DE")
AktionOK=            RenameDirectory(OldDirectoryName:"C:\TEST";
                     NewDirectoryName:"C:\TEST111")
If                   (AktionOK = True)
   MessageBox        (MVar;"Ordner umbenennen"; "Ordner umbenannt"; IconExclamation!)
Else
   Messagebox        (MVar;"Ordner umbenennen"; "Ordner nicht gefunden"; IconExclamation!)
EndIf
```

Wenn das Umbenennen erfolgreich war, enthält die Variable *AktionOK* den Inhalt *True*, wenn nicht *False*. Achten Sie auf das Gleichheitszeichen zwischen der Variablen und dem Befehl. Aufgrund dieses Inhalts wird in Verbindung mit der *If*-Abfrage die zutreffende Meldung angezeigt. *True* und *False* dürfen nicht durch Hochkommata eingeschlossen sein. Die Parameternamen *OldDirectoryName* und *NewDirectoryName* können auch weggelassen werden.

Beispiel

```
Application          (A1; "WordPerfect"; Default; "DE")
If                   (RenameDirectory("C:\TEST";"C:\TEST111") = True)
   MessageBox        (MVar;"Ordner umbenennen"; "Ordner umbenannt"; IconExclamation!)
Else
   Messagebox        (MVar;"Ordner umbenennen"; "Ordner nicht gefunden"; IconExclamation!)
EndIf
```

Bei dem zweiten Beispiel kann die Verwendung einer Variablen entfallen, da die Prüfung direkt in Verbindung mit dem *RenameDirectory*-Befehl erfolgt. Achten Sie hier insbesondere auf die Klammern. Die Reihenfolge ist bei mehreren Parametern in jedem Fall einzuhalten.

Beispiel

```
Application          (A1; "WordPerfect"; Default; "DE")
If                   (Not(RenameDirectory("C:\TEST";"C:\TEST111") = True))
   MessageBox        (MVar;"Ordner umbenennen"; "Ordner nicht gefunden"; IconExclamation!)
Else
   Messagebox        (MVar;"Ordner umbenennen"; "Ordner umbenannt"; IconExclamation!)
EndIf
```

In Verbindung mit *Not* sind die beiden Befehle *MessageBox* zu tauschen, da die Abfrage im Vergleich zu Beispiel 2 umgekehrt ist. Achten Sie auch hier insbesondere auf die Verwendung der Klammern!

Wenden Sie die hier gezeigten Definitionsmöglichkeiten sinngemäß auf die nachfolgend beschriebenen Befehle an. Beachten Sie bitte auch das Beispiel am Beginn von Abschnitt »6.50 File-Befehle«.

CreateDirectory

Erstellen eines neuen Ordners

Befehlsform: *Boolean* **CreateDirectory**(*OrdnerName;Meldung*)

Erstellen eines neuen Ordners. Ist bereits ein Ordner desselben Namens vorhanden, erfolgt eine Fehlermeldung.

Beispiel

```
Application          (A1; "WordPerfect"; Default; "DE")
If                   (CreateDirectory(DirectoryName:"C:\TEST")= True)
MessageBox           (MVar;"Ordner erstellen"; "Ordner erstellt"; IconExclamation!)
Else
Messagebox           (MVar;"Ordner erstellen"; "Ordner nicht erstellt"; IconExclamation!)
EndIf
```

DeleteDirectory

Löschen eines Ordners

Befehlsform: *Boolean* **DeleteDirectory**(*OrdnerName;Meldung*)

Löschen eines Ordners. Ist der zu löschende Ordner nicht vorhanden, erfolgt eine Fehlermeldung.

Beispiel

```
Application          (A1; "WordPerfect"; Default; "DE")
If                   (DeleteDirectory(DirectoryName:"C:\TEST") = True)
MessageBox           (MVar;"Ordner löschen"; "Ordner gelöscht"; IconExclamation!)
Else
Messagebox           (MVar;"Ordner löschen"; "Ordner nicht gefunden"; IconExclamation!)
EndIf
```

DoesDirectoryExist

Ist Ordner vorhanden?

Befehlsform: *Boolean* **DoesDirectoryExist**(*OrdnerName*)

Prüfen, ob ein bestimmter Ordner vorhanden ist.

Beispiel

```
Application          (A1; "WordPerfect"; Default; "DE")
```

```
If                  (DoesDirectoryExist(DirectoryName:"C:\TEST") = True)
MessageBox          (MVar;"Ordner prüfen"; "Ordner gefunden"; IconExclamation!)
Else
Messagebox          (MVar;"Ordner prüfen"; "Ordner nicht gefunden"; IconExclamation!)
EndIf
```

GetCurrentDirectory

Aktuelle Ordner ermitteln

Befehlsform: *Boolean* **GetCurrentDirectory()**

In der Variablen *Ordner* wird der ermittelte Ordnername gespeichert. Achten Sie bitte bei der Weiterverwendung darauf, daß der Ordnername mit einem Backslash \ endet.

Beispiel

```
Application         (A1; "WordPerfect"; Default; "DE")
Ordnername=         GetCurrentDirectory()
MessageBox          (MVar; "Aktueller Ordner:"; Ordnername; IconExclamation!)
```

RenameDirectory

Ordner umbenennen

Befehlsform: *Boolean* **RenameDirectory**(*OrdnerAlt;OrdnerNeu;Meldung*)

Unter ***OrdnerAlt*** ist der bisherige Ordnername und unter ***OrdnerNeu*** der neue Ordnername einzugeben.

Beispiel

```
Application         (A1; "WordPerfect"; Default; "DE")
If                  (RenameDirectory(OldDirectoryName:"C:\TEST";
                    NewDirectoryName:"C:\TEST111") = True)
MessageBox          (MVar;"Ordner umbenennen"; "Ordner umbenannt"; IconExclamation!)
Else
Messagebox          (MVar;"Ordner umbenennen"; "Ordner nicht gefunden"; IconExclamation!)
EndIf
```

SetCurrentDirectory

Ordner wechseln

Befehlsform: *Boolean* **SetCurrentDirectory**(*OrdnerName*)

Nach der Ausführung des Befehls wird der unter ***OrdnerName*** angegebene Ordner zum Standardordner, sofern er vorhanden ist.

Beispiel

```
Application         (A1; "WordPerfect"; Default; "DE")
If                  (SetCurrentDirectory(DirectoryName:"C:\TEST") = True)
MessageBox          (MVar;"Ordner zuordnen"; "Ordner zugeordnet"; IconExclamation!)
Else
```

```
Messagebox          (MVar;"Ordner zuordnen"; "Ordner nicht gefunden"; IconExclamation!)
EndIf
```

6.40 Discard

Befehlsform: **Discard**(*Variable1;Variable2;...*)

Nicht mehr benötigte Variable können Sie mit diesem Befehl löschen. Gelöscht werden können Variable in lokalen, globalen und Persist-Tabellen. Nach dem Löschen ist die Variable nicht mehr verfügbar, denn es wird nicht nur der Inhalt gelöscht, sondern die Variable wird komplett entfernt. Es wird eine Fehlermeldung angezeigt, wenn in einem nachfolgenden Befehl auf diese Variable wieder Bezug genommen wird. Die Löschung erfolgt in der Reihenfolge lokale, globale und Persist-Variable. Es kann nicht kontrolliert werden, in welcher Tabelle eine Variable gelöscht wurde. Um die Variable in allen drei Tabellen zu löschen, sollten Sie die unter Beispiel 2 gezeigten Anweisungen verwenden.

Parameter

Variable Beliebige lokale, globale oder Persist-Variable.

Beispiel

```
1         Application          (A1; "WordPerfect"; Default; "DE")
          Test=                "Das ist ein Test, nach dieser Anzeige wird das Makro
                               automatisch abgebrochen!"
          Prompt               ("Befehl ""DISCARD"" testen";Test)
          Wait                 (40)
          EndPrompt
          Discard              (Test)
                               // Der folgende Prompt-Befehl wird
                               // nicht mehr ausgeführt, da die
                               // Variable TEST gelöscht wurde. Die
                               // Makroausführung wird mit einer
                               // Fehlermeldung abgebrochen.
          Prompt               ("Befehl ""DISCARD""
                               testen";Test;;;)
          Dieses Beispiel finden Sie auf der CD-ROM unter DISCARD.WCM.
2         While                (Exists(Test))
            Discard            (Test)
          EndWhile
```

In dieser Befehlsfolge wird die Variable *Test* in allen drei Tabellen gelöscht. Die Schleife wird max. dreimal wiederholt (so lange, bis die Bedingung zutrifft).

Weitere Hinweise siehe: Assign, Global, Local, Persist, PersistAll

6.41 DLL-Befehle

Die DLL-Befehle arbeiten direkt mit unter Windows erstellten Programmen (EXE.Dateien) oder mit DLL (**D**ynamic **L**ink **L**ibraries) zusammen, d. h. sie greifen auf Programme oder Dateien außerhalb von WordPerfect zu. Diesen Makrobefehl dürfen Sie erst dann verwenden, wenn Sie vollkommen mit dessen Funktion und mit der DLL-Verarbeitung unter Windows vertraut sind. Ist dies nicht der Fall, können z. B. bei Unachtsamkeit, Dateien zerstört werden oder der Computer wird u. U. neu gebootet.

```
DLLCall              (Link;Funktion;VonDLL:Datentyp;{NachDLL}))
DLLFree              (Link)
DLLLoad              (Link;DLL)
DLLCall Prototype    Name (ModulDateiname;Funktion;Datentyp;{Parameter;Parameter;...})
```

Da diese Befehle von den wenigsten Lesern benötigt werden, wurde wegen begrenzten Platzes in diesem Buch auf eine detaillierte Beschreibung zugunsten wichtigerer Befehle verzichtet. Schlagen Sie bitte bei Bedarf im Makro-Online-Handbuch nach oder sehen Sie sich die von Corel WordPerfect mitgelieferten Makros an.

6.42 Else

Befehlsform: Else

Dieser Befehl kann immer in Verbindung mit einer *If*-Struktur auftreten. Die nach *Else* folgenden Befehle werden nur dann ausgeführt, wenn die *If*...-Bedingung nicht zutrifft. Trifft eine *If*-Bedingung nicht zu, und ist kein *Else* vorhanden, werden die zwischen *If* und dem zugehörigen *EndIf* definierten Befehle nicht ausgeführt. Ausnahmen bestehen bei geschachtelten *If*-Bedingungen. In beiden Fällen wird mit den Befehlen weitergearbeitet, die dem *EndIf* folgen. Der Befehl kann auch in Verbindung mit *Case* oder *Case Call* ausgeführt werden, wenn dort die vorgegebenen Bedingungen nicht zutreffen.

Beispiel:

```
If                   (Nummer=5)
   Call              (Drucken)
  Else
   Call              (Speichern)
EndIf
Call                 (Laden)
```

Enthält die Variable *Nummer* den Wert 5, wird die Unterroutine *Drucken* aufgerufen, wenn nicht, die Unterroutine *Speichern*. Der Befehl *Call*(Laden) wird immer ausgeführt, gleichgültig, welchen Wert die Variable *Nummer* enthält, da er in keiner Beziehung zu dem *If*-Befehl steht.

Weitere Hinweise siehe: If, IfPlatForm, EndIf, EndIfPlatForm

6.43 EndApp

Befehlsform: **EndApp**(*Produkt-Abkürzung*)

Beendet die Anweisung *Application*. In Klammern muß die zweistellige Produktabkürzung angegeben werden (z. B. A1=WordPerfect). Die danach folgenden Makrobefehle des aktuellen Makros beziehen sich nicht mehr auf das zugeordnete Produkt. Alle produktspezifischen Anweisungen, die diesem Befehl folgen, und die keine Produkt-Abkürzung aufweisen, sind ungültig, sofern im Anschluß an *EndApp* nicht wieder ein Befehl *Application* mit einer neuen Produktabkürzung folgt.

Parameter

Produkt-Abkürzung
Zweistellige Abkürzung der betreffenden Produkt-Kennung.

Beispiel: EndApp(A1)

Weitere Hinweise siehe: Application

6.44 End...

Befehlsform: EndFor, EndFunc, EndIf, EndIfPlatForm, EndProc, EndPrompt, *EndSwitch*, EndWhile

Diese Befehle beenden eine Schleife bzw. eine Befehlsfolge.

EndFor Die Befehlsfolge einer *For*-Anweisung muß zwingend mit einem *EndFor*-Befehl beendet werden. Die dazwischen definierten Befehle werden aufgrund der unter *For* angegebenen Werte wiederholt (= Schleife). Die Ausführung dieses Befehls bewirkt den Rücksprung zu *For*. Die Ausführung wird wieder bei *For* fortgesetzt, wobei die Parameter von *For* entsprechend verändert werden. Dieser Vorgang wird so lange wiederholt, bis die Schleife so oft abgearbeitet wurde, wie unter *For* angegeben. Zwischen *For* und *EndFor* können Sie fast alle Makrobefehle einschließen. Gleiches gilt sinngemäß für den Befehl *ForEach*.

EndFunc Die Befehlsfolge einer *Function*-Struktur muß zwingend mit einem *EndFunc*-Befehl beendet werden.

EndIf Dieser Befehl markiert das Ende einer *If...*-Struktur. Jede *If...*-Struktur muß mit *EndIf* beendet werden. Werden mehrere *If*-Befehle geschachtelt, müssen ebenso viele *EndIf*-Befehle vorhanden sein, wie *If*-Befehle

definiert wurden. Wenn Sie dies nicht beachten, wird beim Kompilieren eine Fehlermeldung angezeigt.

Beispiel 1 (einfache If-Bedingung):

```
If        (Druck=5)        //Beginn der If-Struktur
  Call    (Drucken)        //Aufruf von Drucken, wenn Druck = 5
    Else
  Call    (Sichern)        //Aufruf von Sichern, wenn Druck nicht = 5
EndIf                      //Ende der IF-Struktur
Call      (Laden)          //Aufruf von Laden, wird immer ausgeführt
```

Beispiel 2 (geschachtelte If-Bedingung):

```
If        (KennAbkz="z")                    //Beginn der IF-Struktur
          Assign(KennTxt="Neuzugang") Else
  If      (KennAbkz="a")
          Assign(KennTxt="Abgang")    Else
    If    (KennAbkz="v")
          Assign(KennTxt="Veränderung")
Else
          Prompt("Fehler";"Falsches Änderungskennzeichen")
          Beep
          Wait(20)
          GO(Anfang)
EndIf EndIf EndIf                           //Ende der IF-Struktur
```

EndProc Beendet eine *Procedure*-Struktur.

EndPrompt Dieser Befehl tritt immer in Verbindung mit *Prompt* auf. Er schließt ein durch *Prompt* am Bildschirm angezeigtes Dialogfeld. Er ist ohne Wirkung, wenn zuvor kein ***Prompt***-Befehl ausgeführt wurde. Beispiele finden Sie unter *Prompt*.

EndSwitch Die Befehlsfolge einer *Switch*-Anweisung muß zwingend mit einem *EndSwitch*-Befehl beendet werden. Die dazwischen definierten Befehle werden aufgrund der unter *Switch* angegebenen Variablen in Verbindung mit dem Befehl *CaseOf* ausgeführt. Beispiele finden Sie unter *Switch*.

EndWhile Die Befehlsfolge einer *While*-Anweisung muß zwingend mit einem *EndWhile*-Befehl beendet werden. Die dazwischen definierten Befehle werden aufgrund der unter *While* angegebenen Werte wiederholt (= Schleife). Die Ausführung dieses Befehls bewirkt den Rücksprung zu *While*. Die Ausführung wird wieder bei *While* fortgesetzt. Dieser Vorgang wird so lange wiederholt, bis die Schleife so oft abgearbeitet wurde, wie unter *While* angegeben. Danach wird mit dem Befehl weitergearbeitet, der dem *EndWhile*-Befehl folgt.

Weitere Hinweise siehe: For, If, Prompt, Switch, While

6.45 EndIfPlatform

Befehlsform: **EndIfPlatform**([*PlatFormKennung*])

Dieser Befehl beendet eine **IFPlatForm**-Struktur.

Parameter

PlatFormKennung Die Verwendung ist wahlfrei. Hier können eine oder mehrere der unter *IfPlatForm* verwendeten Plattform-Kennungen angegeben werden.

Beispiel: Siehe unter IfPlatForm

Weitere Hinweise siehe: IfPlatform

6.46 EnvVariableGet / EnvVariableSet

Befehlsform: *Variable*=**EnvVariableGet**(*EnvVariable*;*EnvType*)
Variable=**EnvVariableSet**(*EnvVariable*;*Wert*;*EnvTyp*)

Setzen, löschen (Set) oder ermitteln (Get) einer Umgebungsvariablen, der entnommen werden kann, ob das Makro unter DOS oder unter NetWare ausgeführt wird.

Parameter

Variable Speicherung des ermittelten Systems.

EnvVariable Name der Environment-Variablen.

EnvType DOS! oder NetWare!

Wert Wert, der der Variablen zugeordnet werden soll. Ist dieser Parameter nicht angegeben, wird die Environment-Variable gelöscht.

6.47 Error

Befehlsform: *Variable*=**Error**(*Status*) oder **Error**(*Status*)

Über diesen Befehl können Sie festlegen, was beim Auftreten von Fehlern geschehen soll (z. B. wenn eine zu öffnende Datei nicht vorhanden ist).

Parameter

Variable Diese Variable enthält den Status des zuletzt ausgeführten *Error*-Befehls, der hier bei Bedarf abgefragt werden kann, um im Makro bestimmte Funktionen auszuführen. Ist die Variable leer, wurde der *Error*-Befehl vorher nicht ausgeführt. Die Variable kann auch weggelassen werden.

Status Folgende Statusangaben sind möglich:

On! Die während der Makro-Ausführung auftretenden Programm-Fehlermeldungen werden angezeigt. Die Makro-Verarbeitung wird bei dem Label fortgesetzt, der bei dem zuletzt ausgeführten *OnError*- oder *OnError Call*-Befehl angegeben wurde. Wurden *OnError* oder ***OnError Call*** in diesem Zusammenhang nicht verwendet, wird das Makro beendet. Standardmäßig wird *Error*(*On!*) vorgegeben, d. h. Sie brauchen diesen Befehl nur dann zu verwenden, wenn zuvor *Error*(*Off!*) ausgeführt wurde, oder wenn Sie die Standardvorgabe nicht verwenden wollen.

Off! Während der Makro-Ausführung werden auftretende Programm-Fehlermeldungen ignoriert, das Makro läuft weiter. Fehlermeldungen aufgrund falscher Makro-Programmierung (z. B. Variable nicht vorhanden) werden allerdings nicht ignoriert.

Beispiel:

```
Error               (On!)
OnError             (Fehler)
Call                (Öffnen)
::::
Label               (Öffnen)
DateiÖffnen         (Dateiname:Adressen.dat)
::::
Label               (Fehler)
Prompt              ("Fehler";"Datei konnte nicht geöffnet werden";;;)
::::
```

In diesem Beispiel wird vor dem *Call*-Befehl die Bedingung *OnError* gesetzt. Tritt beim Öffnen der Datei ADRESSEN.DAT ein Fehler auf (z. B. Datei nicht gefunden), wird zu dem Label verzweigt, der unter *OnError* angegeben wurde. Hier: Zu Label *Fehler*.

Beachten Sie bitte auch das Beispiel unter »OnCancel«, und wenden Sie es sinngemäß an.

Weitere Hinweise siehe: OnError, OnError Call

6.48 ErrorNumber

Befehlsform: *Variable*=**ErrorNumber**

In der angegebenen Variablen wird die ermittelte Fehlernummer einer Cancel-, Error oder Not-Found-Bedingung als numerischer Wert gespeichert. Aufgrund dieser Nummer kann geprüft werden, welche Bedingung zutraf.

Parameter

Variable: Speicherung der ermittelten Fehlernummer:

1 Cancel-Bedingung

2 Error-Bedingung

3 Not-Found-Bedingung

Weitere Hinweise siehe: Assert

6.49 Exists

Befehlsform: *Variable1*=**Exists**(*Variable2;Pool*) oder
Bedingung=**Exists**(*Variable2;Pool*)

Dieser Befehl kontrolliert in Verbindung mit anderen Befehlen wie z. B. *If*, ob die angegebene Variable einen Wert enthält bzw. ob diese vorhanden ist. Ist ein Wert vorhanden, trifft die Bedingung zu, andernfalls nicht. Die Programmierlogik entspricht den Beschreibungen des *If*-Befehls. Auch hier muß das Ende der Struktur durch *EndIf* abgeschlossen werden. Wurden die Befehle *Declare*, *Local*, *Global* oder *Persist* verwendet, so wird in einer Variablen angezeigt, in welcher Variablentabelle die Variable vorhanden ist.

Parameter

Variable1: Wird der Parameter *Pool* weggelassen, enthält diese Variable einen der unter *Pool* genannten Inhalte. Ist *Pool* angegeben, enthält diese Variable entweder *True* oder *False* (Variable ist oder ist nicht in einer der unter *Pool* genannten Tabelle enthalten).

Bedingung: Verwendung des Befehls in Verbindung mit anderen Befehlen, z. B. IF (siehe Beispiel 1).

Variable2: Zu prüfende Variable.

Pool: Folgende Parameter stehen zur Verfügung:

NotFound!	oder 0	Variable existiert nicht.

Local!	oder 1	Variable ist in der Tabelle der lokalen Variablen enthalten.
Global!	oder 2	Variable ist in der Tabelle der globalen Variablen enthalten.
Persist!	oder 3	Variable ist in der Tabelle der mit *Persist* zugeordneten Variblen vorhanden.

Wird dieser Parameter weggelassen, wird zurückgegeben, in welcher Tabelle die Variable vorhanden ist. Wenn der Parameter vorhanden ist, wird *True* oder *False* zurückgegeben.

Beispiel:

```
1         IF                  (Exists(Nachname))
             Type             (Nachname)
           Else
             Prompt           ("Fehler";"Nachname fehlt.";;;)
             Call             (Erfassen)
          EndIf
```

Mittels dieser Abfrage wird geprüft, ob die Variable *Nachname* Daten enthält, bzw. ob sie vorhanden ist. Wenn ja, wird der Name geschrieben, ansonsten erfolgt eine Fehlermeldung, und die Unterroutine *Erfassen* wird zur Erfassung des Nachnamens aufgerufen.

```
2         Vorhanden=          Exists(Nachname)
          If                  (Vorhanden>0)
             Type             (Nachname)
           Else
             Prompt           ("Fehler";"Nachname fehlt.";;;)
             Call             (Erfassen)
          EndIf
```

Auch hier wird geprüft, ob die Variable *Nachname* vorhanden ist. Aufgrund der Definition des *Exists*-Befehls wird der Variablen *Vorhanden* der Inhalt NotFound!, Local!, Global! oder Persist! zugeordnet. Enthält die Variable NotFound!, so ist die Variable *Nachname* nicht vorhanden oder sie enthält keine Daten. Beachten Sie bitte auch Kapitel »5.17 Änderungen gegenüber WPWin 5.1/5.2«.

Weitere Hinweise siehe: EndIf, If, Look

6.50 File-Befehle

Die nachfolgend beschriebenen File-Befehle helfen Ihnen bei der Handhabung von Dateien wie z. B. Öffnen, Löschen, Umbenennen usw. Bei der Beschreibung dieser Befehle werden Parameter, die bei allen Befehlen identisch sind, hier anschließend beschrieben.

Boolean Anstelle von *Boolean* kann z. B. eine Variable oder eine Bedingung verwendet werden. Dadurch erfolgt eine Rückmeldung, ob der Befehl erfolgreich ausgeführt wurde oder nicht.

False	Ausführung war nicht erfolgreich.
True	Ausführung war erfolgreich.

Beachten Sie wegen der unterschiedlichen Verwendung der Befehle die Hinweise am Beginn von Abschnitt »6.39 Directory-Befehle«, und wenden Sie diese sinngemäß auf die File-Befehle an. Schauen Sie sich bei Bedarf auch die Original-WordPerfect-Beispiele im Online-Makrohandbuch unter dem jeweiligen Befehl an.

Ausdruck Anzeigen einer Meldung. Der Befehl kann bei Bedarf weggelassen werden. Unter *Ausdruck* können folgende Anweisungen vorgenommen werden.

NoPrompts!	Keine Anzeige.
Prompts!	Anzeige einer Meldung.

String.. Angabe von alphanumerischen Konstanten oder einer Variablen, die den erforderlichen Eintrag (Ordnernamen) enthalten. Konstanten müssen durch Hochkommata eingeschlossen sein.

Da die Befehlssyntax einfach zu handhaben ist, und die Befehle eindeutig benannt sind, wird auf eine umfangreiche Beschreibung verzichtet. Den folgenden Beispielen können Sie entnehmen, wie die File-Befehle zu verwenden sind. Nach der Ausführung eines Befehls wird in Verbindung mit Boolean ein Wert ermittelt, der Auskunft darüber gibt, ob die Ausführung erfolgreich war oder nicht (siehe oben).

Die Beschreibung erfolgt in alphabetischer Reihenfolge der englischen Befehle. Zum schnelleren Auffinden wird in der jeweiligen Überschriftszeile am rechten Rand die deutsche Übersetzung angezeigt.

Das folgende Beispiel beinhaltet die meisten der beschriebenen Befehle. In einer Kommentarzeile wird jeweils auf den betreffenden Befehl hingewiesen. Dort finden Sie auch eine nähere Beschreibung. Das Beispiel wurde mit Absicht einfach und übersichtlich gehalten, um es leicht zu verstehen.

Beispiel:

```
Application                (A1; "WordPerfect"; Default; "DE")
// >>>>                    Laufwerk und Ordner eingeben.
Label                      (Anfang)
GetString                  (Laufwerk;Length=1;"Bitte Laufwerk eingeben: ";"Laufwerk")
GetString                  (Ordner;Length=8;"Bitte Test-Ordnername eingeben:";"Ordnername")
Ordner=                    (Laufwerk+":\"+Ordner)
//>>CreateDirectory<<      Ordner erstellen, wenn er noch nicht existiert.
If                         (Not(DoesDirectoryExist(Ordner) = True))
   If                      (CreateDirectory(Ordner) = True)
     MessageBox            (MVar;"Hinweis"; "Ordner  " + Ordner + " +erstellt.")
```

```
   Else
     MessageBox          (MVar;"Hinweis"; "Falscher Ordnername  " + Ordner)
     Go                  (Anfang)
   EndIf
Else
   MessageBox            (MVar;"Hinweis"; Ordner + "  existiert bereits.")
EndIf
//>>>>                    Dateiname eingeben.
GetString                (Datei;Length=8;"Bitte Dateiname eingeben:";"Dateiname")
//>>>>                    Laufwerk, Ordner und Dateiname zusammenfügen.
Dateiname=               (Ordner + "\" + Datei)
//>>>>                    Wieviel Sätze sollen geschrieben werden (max. 20).
Label                    (Erstellen)
GetNumber                (Anzahl;"Bitte Anzahl der zu schreibenden Sätze eingeben
                         (max. 20).";"Test")
If                       ((Anzahl>20) or (Anzahl<1))
   Go                    (Erstellen)
EndIf
//>>OpenFile<<           Datei öffnen, bestehende Daten werden gelöscht.
DatKennz1 =              OpenFile(Dateiname; WriteNew!;DenyNone!; AnsiText!)
If                       (DatKennz1 < 0)
  MessageBox             (MVar; "Hinweis"; "Fehler beim Öffnen."; IconExclamation!)
EndIf
//>>FileWrite<<          Schreiben von Sätzen aufgrund der eingegebenen Anzahl.
ForNext                  (Zähler; 1; Anzahl; 1)
   If                    (FileWrite(DatKennz1; "**Datensatz** " + Zähler;NewLine!) < 0)
     MessageBox          (MVar; "Hinweis"; "Fehler beim Schreiben.";IconExclamation!)
   EndIf
EndFor
//>>diverse<<            Diverse Meldungen anzeigen (Testen mehrerer Befehle).
MessageBox               (MVar;"Hinweis"; Anzahl + " Sätze wurden geschrieben.")
DateiGröße =             FileSize(Dateiname)
MessageBox               (MVar;"Dateigröße:"; DateiGröße)
DateiAttrib =            GetFileAttributes(Dateiname)
MessageBox               (MVar;"Dateiattribut:"; DateiAttrib)
DatumZeit=               GetFileDateAndTime(Dateiname)
MessageBox               (MVar;"Datum und Uhrzeit:"; DatumZeit)
//>>>>                    Lesen der Datei ab einem bestimmten Satz bis zum Dateiende.
Label                    (DateiLesen)
GetNumber                (AnzahlR;"Ab welchem Satz soll gelesen werden?";"Test")
If                       ((AnzahlR>20) or (AnzahlR<1) or (AnzahlR>Anzahl))
   Go                    (DateiLesen)
EndIf
//>>FilePosition<<       Positionszeiger am Dateianfang positionieren.
FilePosition             (DatKennz1; 0; FromBeginning!)
//>>FileRead<<           Positionszeiger zum gewünschten Satz bringen.
ForNext                  (Zähler; 1; AnzahlR-1; 1)
   If                    (FileRead(DatKennz1; EingabeBereich) < 0)
     MessageBox              (MVar; "Hinweis"; "Fehler beim Lesen."; IconExclamation!)
   EndIf
EndFor
//>>>>                    Sätze lesen und anzeigen.
```

```
ForNext                    (Zähler; 1; Anzahl; 1)
//>>FileIsEOF              Wenn Dateiende erreicht wird, ForNext-Schleife abbrechen.
   If                      (FileIsEOF(DatKennz1)=True)
      MessageBox           (MVar;"Hinweis";"Dateiende wurde erreicht.")
      Break
   EndIf
//>>FileRead<<             Nächsten Satz lesen.
   If                      (FileRead(DatKennz1; EingabeBereich) < 0)
      MessageBox           (MVar; "Hinweis"; "Fehler beim Lesen."; IconExclamation!)
   Else
      MessageBox           (MVar; "Eingabesatz:"; EingabeBereich)
   EndIf
EndFor
//>>FileClose<<            Schließen der Datei und Makro beenden.
If                         (Not(CloseFile(DatKennz1)) = True)
   MessageBox              (MVar; "Hinweis"; "Fehler beim Schließen.")
Else
   MessageBox              (MVar; "Hinweis"; "Datei wurde geschlossen.")
EndIf
Quit
```

Dieses Beispiel führt folgende Funktionen aus:

1. Aufgrund der unter *GestString* eingegebenen Laufwerks- und Ordnerangaben wird ein neuer Unterordner erstellt, sofern er nicht vorhanden ist.
2. Eingabe eines Dateinamens zum Speichern von Datensätzen.
3. Eingabe einer Zahl (Menge der zu speichernden Datensätze).
4. Öffnen einer Datei.
5. Schreiben der Datensätze (z. B. ***Datensatz** 1*). Der Text bleibt bei jedem Satz identisch, der Zähler wird erhöht.
6. Anzeigen von Dateigröße, Attributen, Datum und Uhrzeit.
7. Eingabe eines Zählers zum Anzeigen von Datensätzen.
8. Anzeigen der Datensätze ab einem bestimmten Satz.
9. Schließen der Datei.

CloseFile

Datei schließen

Befehlsform: *Boolean* **CloseFile**(*DateiKenn*)

Schließen einer Datei. Unter ***DateiKenn*** ist dieselbe Variable anzugeben, die als ***DateiKenn*** beim Öffnen der Datei mit *OpenFile* verwendet wurde. Werden mehrere Dateien geöffnet und gleichzeitig offen gehalten, ist für jede Datei eine eigene Variable zu definieren (siehe *OpenFile*). Eine Datei kann nur einmal geschlossen werden, d. h. ein *CloseFile*-Befehl auf eine bereits geschlossene Datei erzeugt eine Fehlermeldung. Beim Verlassen eines Makros werden alle nicht geschlossenen Dateien automatisch geschlossen.

Beispiel

```
//>>FileClose<<       Schließen der Datei und Makro beenden.
If                    (Not(CloseFile(DatKennz1)) = True)
  MessageBox          (MVar; "Hinweis"; "Fehler beim Schließen.")
Else
  MessageBox          (MVar; "Hinweis"; "Datei wurde geschlossen.")
EndIf
Quit
```

Für einen sicheren Makroablauf muß geprüft werden, ob die Datei auch geschlossen wurde, ansonsten ist eine Fehlermeldung auszugeben, und das Makro ggf. abzubrechen (muß der Programmierer von Fall zu Fall entscheiden). Dateien sollten nur so lange geöffnet bleiben, wie sie benötigt werden!

CopyFile

Datei kopieren

Befehlsform: *Boolean* **CopyFile**(*VonDatei;NachDatei;Meldung*)

Die unter *VonDatei* angegebene Datei wird unter dem Dateinamen von *NachDatei* kopiert. Achten Sie auf vollständige Dateinamen. Unter *Meldung* sollten Sie sicherheitshalber *Prompts!* angeben, damit eine Abfrage eingeblendet wird, wenn der unter *NachDatei* angegebene Dateiname schon existiert. Bestätigen Sie dann je nach Wunsch mit [*Ja*] oder [*Nein*].

Beispiel

```
Application           (A1; "WordPerfect"; Default; "DE")
If                    (CopyFile("C:\TEST\BEISPIEL.DAT";"C:\TEST\BEISPIEL.TXT") = True)
MessageBox            (Mvar;"Datei kopieren"; "Datei wurde kopiert"; IconExclamation!)
Else
Messagebox            (Mvar;"Datei kopieren"; "Fehler beim Kopieren"; IconExclamation!)
EndIf
```

DeleteFile

Datei löschen

Befehlsform: *Boolean* **DeleteFile**(*Dateiname;Meldung*)

Die unter ***Dateiname*** angegebene Datei wird gelöscht.

Beispiel

```
Application           (A1; "WordPerfect"; Default; "DE")
If                    (DeleteFile("C:\TEST\BEISPIEL.DAT";) = True)
MessageBox            (Mvar;"Datei löschen"; "Datei wurde gelöscht"; IconExclamation!)
Else
Messagebox            (Mvar;"Datei löschen"; "Fehler beim Löschen"; IconExclamation!)
EndIf
```

DoesFileExist

Ist Datei vorhanden?

Befehlsform: *Boolean* **DoesFileExist**(*Dateiname*)

Prüfen, ob eine Datei vorhanden ist. Geben Sie unter *Dateiname* den vollständigen Dateinamen ein.

Beispiel

```
Application          (A1; "WordPerfect"; Default; "DE")
If                   (DoesFileExist("C:\TEST\BEISPIEL.DAT";) = True)
MessageBox           (Mvar;"Ist Datei vorhanden?"; "Datei ist vorhanden"; IconExclamation!)
Else
Messagebox           (Mvar;"Ist Datei vorhanden?"; "Datei nicht vorhanden"; IconExclamation!)
EndIf
```

FileError

Dateifehler ermitteln

Befehlsform: *NumWert*=**FileError()**

Treten in Verbindung mit den Befehlen *CloseFile*, *FileIsEOF*, *FilePosition*, *FileRead*, *FileSize*, *FileTruncate*, *FileWrite* oder *OpenFile* Fehler auf, enthält ***NumWert*** einen positiven oder negativen Wert:

-1	Unbekannter Fehler.
0	Kein Fehler.
>0	Letzter Dateifehler.

Dieser bezieht sich immer auf den zuletzt ausgeführten Datei-Befehl. Wenn dieser Befehl verwendet wird, sollte er unmittelbar nach der Ausführung eines Dateibefehls benutzt werden. Bei einem Wert ungleich 0 muß die Verarbeitung im Makro mit einer entsprechenden Fehlermeldung abgebrochen werden (tritt z. B. beim Lesen einer Datei mit *READ!* auf, wenn die Datei nicht geöffnet wurde oder bereits geschlossen ist).

Beispiel

```
Application          (A1; "WordPerfect"; Default; "DE")
FehlNum=             FileError()
If                   (FehlNum = 0)
   MessageBox        (Mvar;"Dateifehler"; "Kein Fehler"; IconExclamation!) Else
   If                (Fehlnum = -1)
      Messagebox     (Mvar;"Dateifehler"; "Unbekannter Fehler"; IconExclamation!)Else
      If             (FehlNum > 0)
         Messagebox  (Mvar;"Dateifehler"; "Letzter Fehler"; IconExclamation!)Else
EndIf  EndIf EndIf
```

Die Meldungen haben Loword- und Hiword-Werte, wobei Loword einen von den WordPerfect Ein-/Ausgabe-Routinen erzeugten Fehlercode enthält (beschrieben unter »WordPerfect für Windows SDK«). Hiword enthält einen windowsspezifischen Fehler, der unter »Microsoft SDK« beschrieben wird.

FileFind

Datei suchen

Befehlsform: *String*=**FileFind**(*Dateiname;Attribute;Context*)

Suchen einer Datei nach bestimmten Suchkriterien, die unter ***String2*** angegeben wurde in Verbindung mit den unter **Attribute** und **Context** genannten Werten.

String Wurde die Datei gefunden, enthält diese Variable den vollständigen Dateinamen (Laufwerk, Ordner, Name). Die Variable bleibt leer, wenn die Datei nicht gefunden wurde.

Dateiname Angabe der zu suchenden Datei (vollständiger Dateiname). Die von DOS her bekannten Wildcard * und ? sind erlaubt.

Attribute Auswahl der DOS-Datei-Attribute. Der Parameter ist optional. Wird er weggelassen, wird automatisch *Normal!* angenommen.

Archived!	Directory!
Hidden!	Label!
Normal!	ReadOnly!
System!	

Für weitere Informationen zu diesen Parametern schlagen Sie bitte in Ihrem DOS-Handbuch nach.

Context Durch die Eingabe einer Ziffer wird die Suche einer Datei beeinflußt. Der Parameter ist optional. Wird er weggelassen, wird automatisch 0 angenommen. Wird 1 eingegeben, wird das erste Auftreten der Datei gemeldet.

Benutzen Sie unterschiedliche numerische Werte, um mehrfaches Suchen gleichzeitig auszuführen. Werden mehrere Suchvorgänge vor dem Makroende gestartet, kann jeder Suchvorgang durch die Verwendung geeigneter Werte wiederholt werden. Die weitere Suche wird dort fortgesetzt, wo die vorherige geendet hat.

Beispiel

```
Application     (A1; "WordPerfect"; Default; "DE")
DateiName=      FileFind(filename:"*.DOC";Attributes:Normal!;Context:1)
MessageBox      (Mvar;"Datei löschen"; "Name= " + Dateiname; IconExclamation!)
```

In diesem Beispiel werden Daten mit der Dateinamen-Erweiterung **DOC** gesucht.

FileIsEOF

Prüfen, ob Dateiende erreicht wurde

Befehlsform: *Boolean* **FileIsEOF**(*DateiKenn*)

Der Befehl prüft, ob das Dateiende erreicht wurde. Die Dateizugriffsart muß in Verbindung mit *OpenFile* als **Read!** (= *AccessMode*) definiert sein.

Beispiel

```
ForNext              (Zähler; 1; Anzahl; 1)
If                   (FileIsEOF(DatKennz1)=True)
   MessageBox        (MVar;"Hinweis";"Dateiende wurde erreicht.")
   Break
EndIf
If                   (FileRead(DatKennz1; EingabeBereich) < 0)
   MessageBox        (MVar; "Hinweis"; "Fehler beim Lesen."; IconExclamation!)
Else
   MessageBox        (MVar; "Eingabesatz:"; EingabeBereich)
EndIf
EndFor
```

In diesem Beispiel wird vor dem Lesen einer Datei geprüft, ob das Dateiende (EOF = **E**nd **O**f **F**ile) erreicht wurde. Wenn ja (vorausgegangener Read-Befehl hat den letzten Datensatz gelesen), wird ein Hinweis angezeigt und die *ForNext*-Schleife abgebrochen.

FileNameDialog

Öffnen-/Speichern-unter-Dialogfeld anzeigen

Befehlsform: *Variable*=**FileNameDialog**

(Style;Titel;TastenText;Ordner;Dateiname;Dateischema;RegDatKey)

Anzeigen des »Öffnen-« oder »Speichern unter«-Dialogfelds in Abhängigkeit der gewählten Parameter.

Variable Name der zu öffnenden oder zu speichernden Datei. Das Feld bleibt leer, wenn [Abbrechen] gedrückt wurde.

Style Festlegung des Dialogtyps zum Öffnen und Festlegung der zugeordneten Optionen. Mehrere Optionen können ausgewählt werden. Die Trennung der Optionen muß durch | erfolgen.

OpenDialog!	Anzeigen des Öffnen-Dialogfelds.
SaveAsDialog	Anzeigen des Speichern-unter-Dialogfelds
FileMustExist!	Die in Verbindung mit OpenDialog! angegebene Datei muß vorhanden sein.
PromptOnReplace!	Ist die in Verbindung mit SaveAsDialog! angegebene Datei vorhanden, wird ein Dialogfeld angezeigt, in dem Sie bestätigen müssen, ob die bestehende Datei überschrieben werden soll.

Titel Text für die Titelzeile des Dialogfelds. Beim Weglassen des Parameters oder der Verwendung von »« wird »Öffnen« bzw. »Speichern unter« angezeigt.

Tastentext Text für die Öffnen-Taste. Beim Weglassen des Parameters oder der Verwendung von »« wird »Öffnen« angezeigt.

Ordner Anzeigen eines Standardordners. Beim Weglassen des Parameters oder der Verwendung von »« wird der aktuelle Ordner angezeigt.

Dateiname Anzeigen eines Standarddateinamens. Beim Weglassen des Parameters oder der Verwendung von »« werden die unter *Dateischema* angegebenen Zeichen angezeigt.

Dateischema Anzeigen eines Dateischemas. Beim Weglassen des Parameters oder der Verwendung von »« wird *.* vorgeschlagen.

RegDatKey Name des Schlüssels (HKEY_CURRENT_USER\Software) der Registrationsdatenbank zur Anzeige einer Liste der maximal letzten fünfzehn Dateien, die in diesem Dialogfeld geöffnet wurden.

FilePosition

Setzen des Positionszeigers

Befehlsform: *NumWert1*=**FilePosition**(*DateiKenn*;*NeuPos*;*VonPos*)

Angabe der Position des Positionszeigers. Das ist ein Pointer (Zeiger), der angibt, ab welcher Position innerhalb der Datei gelesen oder geschrieben werden soll.

NumWert1 Bisherige Position des Positionszeigers (negativer Wert bedeutet einen Fehler).

DateiKenn Unter ***DateiKenn*** ist dieselbe Variable anzugeben, die als ***DateiKenn*** beim Öffenen der Datei mit *OpenFile* verwendet wurde.

NeuPos Anzahl (negativ oder positiv) der Zeichen, um die der Positionszeiger bewegt werden soll. Bei negativen Werten, wird der Positionszeiger rückwärts bewegt.

VonPos Bewegt den Positionsanzeiger relativ zu der angegebenen Position. Der Parameter ist optional. Wird er weggelassen, wird automatisch *From Beginning!* angenommen.

FromBeginning! Relativ zum Dateianfang.

FromCurrentPosition! Relativ zur aktuellen Positionsanzeige.

FromEnd! Relativ zum Dateiende.

Beispiel

FilePosition (DatKennz1; 0; FromBeginning!)

Die Datei wird von Anfang an bearbeitet (Lesen oder Schreiben).

FileRead — Datei lesen

Befehlsform: *Anzahl*=**FileRead**(*DateiKenn*;*Daten*)

Lesen einer Datei, die zuvor mit *OpenFile* göffnet worden sein muß. Normalerweise wird unmittelbar nach dem Öffnen und der Ausführung des ersten *FileRead* der erste Satz der Datei gelesen, und der Inhalt in *Variable* zum Verarbeiten zur Verfügung gestellt. Ein Satz ist dort zu Ende, wo das Betriebssystem ein Satzende erkennt. Wird der Positionsanzeiger nicht verändert, wird beim zweiten Lesen der zweite Satz, beim dritten Lesen der dritte Satz usw. gelesen. Das wird so lange fortgesetzt, bis das Dateiende erreicht wird.

Anzahl Anzahl der gelesenen Bytes (negativer Wert bedeutet einen Fehler).

DateiKenn Unter *DateiKenn* ist dieselbe Variable anzugeben, die als *DateiKenn* beim Öffenen der Datei mit *OpenFile* verwendet wurde.

Daten Vom Beginn des Positionszeigers an wird eine Zeile gelesen, in einen WordPerfect-String konvertiert und in dieser Variablen dem Makro zur Verfügung gestellt.

Beispiel

```
If(FileRead(DatKennz1; EingabeBereich) < 0)
   MessageBox        (MVar; "Hinweis"; "Fehler beim Lesen."; IconExclamation!)
Else
   MessageBox        (MVar; "Eingabesatz:"; EingabeBereich)

EndIf
```

Die unter *DatKennz1* geöffnete Datei wird gelesen. Die gelesenen Daten werden in *Daten* gespeichert und können im Makro weiterverarbeitet werden. In diesem Beispiel werden sie in einer Messagebox angezeigt. Treten Fehler auf, ist das Makro ggf. abzubrechen (muß von Fall zu Fall entschieden werden). Beachten Sie bitte auch die Dateiende-Erkennung (siehe Befehl *FileIsEOF*).

FileSize — Dateigröße in Bytes

Befehlsform: *Anzahl*=**FileSize**(*Dateiname*)

Anzeigen der Dateigröße (in Bytes). Es wird die Größe angezeigt, die DOS ermittelt hat. Sie können den Wert mit den Angaben im Dateimanager vergleichen. Bei DOS-Text-Dateien sind in diesem Wert auch Carriage-Return und Line-Feed enthalten, bei WordPerfect-Dateien der Header.

Anzahl Anzahl der gelesenen Bytes (negativer Wert bedeutet einen Fehler).

Dateiname Angabe des vollständigen Namens der betreffenden Datei.

Beispiel

```
DateiGröße =        FileSize(Dateiname)
MessageBox          (MVar;"Dateigröße:"; DateiGröße)
```

Die Dateigröße wird in der Variablen *Dateigröße* gespeichert und in der Messagebox angezeigt.

FileTruncate

Datei ab bestimmter Satzposition löschen

Befehlsform: *Anzahlt*=**FileTruncate**(*DateiKenn*)

Der Inhalt einer Datei kann ab einer bestimmten Position bis zum Dateiende gelöscht werden. Diese Position kann vorher mit *FilePosition* festgelegt werden. Nach der Ausführung enthält *Anzahl* die Anzahl der gelöschten Beytes.

Anzahl Anzahl der gelöschten Bytes (negativer Wert bedeutet einen Fehler).

FileWrite

Daten in Datei schreiben

Befehlsform: *Anzahl* =**FileWrite**(*DateiKenn*;*Daten*;*SatzEnde*;*Parameter*)

Schreiben von Daten in die unter *DateiKenn* geöffnete Datei. Wie eine Datei geschrieben wird, hängt von den beim Öffnen unter *AccessMode* verwendeten Parametern ab. Wird eine Datei neu erstellt, werden die Daten der Variabeln *Daten* (evtl. in Verbindung mit den Ergänzungen von ***Parameter***) in die Datei geschrieben. Jeder selbständige Datensatz muß mit dem Befehl *NewLine!* beendet werden.

Anzahl Anzahl der geschriebenen Bytes (negativer Wert bedeutet einen Fehler).

DateiKenn Unter ***DateiKenn*** ist dieselbe Variable anzugeben, die als ***DateiKenn*** beim Öffnen der Datei mit *OpenFile* verwendet wurde.

Daten Die enthaltenen Daten werden vor dem Schreiben in das beim Öffnen der Datei vorgegebene Dateiformat (*DataTyp*) konvertiert. Das Zeichen **^** gefolgt von einer Zahl wird durch den zugehörigen Text in *Parameter* ersetzt. Um ein einzelnes Zeichen **^** zu schreiben müssen zwei **^^** angegeben werden.

SatzEnde Einfügen eines Satz-Ende-Zeichens, wenn erforderlich. Der Parameter ist optional. Wird er weggelassen, wird automatisch *NewLine!* angenommen.

NewLine!	Satz-Ende-Zeichen wird gesetzt.
NoNewLine!	Satz-Ende-Zeichen wird nicht gesetzt. Dadurch kann durch mehrere *FileWrite*-Befehle ein Satz aufgebaut werden. Nach Fertigstellung des Satzes (= anhängen des letzten Satzteils/Feldes) muß ein *File-Write* in Verbindung mit *NewLine!* erfolgen.

Parameter In *Daten* können Daten eingefügt werden, wenn dort mit ^ in Verbindung mit einer Zahl von 0 bis 9 gearbeitet wurde. Die Parameter müssen durch die Zeichen { } eingeschlossen werden.

Beispiel

```
FileWrite       (DateiKenn;

                Data:"Wir fahren am ^0 mit ^1 um 17.00 Uhr nach ^2";
                NewLine!
                ParameterData:{"1. Juni 1996";"der Bahn";"Frankfurt"})
```

Geschrieben wird:

```
Wir fahren am 1. Juni 1996 mit der Bahn um 17.00 Uhr nach Frankfurt.
```

Hierbei werden die Platzhalter ^0, ^1 und ^2 in der angegebenen Reihenfolge durch die unter *Parameter* angegebenen Daten ersetzt. Die Konstanten in *Parameter* können auch durch Variable ersetzt werden, wodurch der Befehl flexibler genutzt werden kann.

Beispiel:

```
DatKennz1 =         OpenFile(Dateiname; WriteNew!;DenyNone!; AnsiText!)
If                  (DatKennz1 < 0)
   MessageBox       (MVar; "Hinweis"; "Fehler beim Öffnen."; IconExclamation!)
EndIf
ForNext             (Zähler; 1; Anzahl; 1)
If                  (FileWrite(DatKennz1; "**Datensatz** " + Zähler;NewLine!) < 0)
   MessageBox       (MVar; "Hinweis"; "Fehler beim Schreiben."; IconExclamation!)
EndIf
EndFor
```

Vor dem Schreiben wird die unter Dateiname angegebene Datei mit dem Parameter *WriteNew!* geöffnet. Dadurch sind die bisherigen Daten dieser Datei verloren. In der *ForNext*-Schleife werden so viele Sätze geschrieben, wie unter ***Anzahl*** angegeben wurde. Tritt hierbei ein Fehler auf, wird über *MessageBox* eine Fehlermeldung angezeigt.

GetFileAttributes — Dateiattribute ermitteln

Befehlsform: *Attribut*=**GetFileAttributes**(*Dateiname*)

Ermitteln der DOS-Dateiattribute

Attribut	Ermitteltes Dateiattribut (negativer Wert bedeutet einen Fehler).
Error!	ReadOnly!
Hidden!	System!
Label!	Normal!
Archived!	

Für weitere Informationen zu diesen Parametern schlagen Sie bitte in Ihrem DOS-Handbuch nach.

Dateiname Vollständiger Dateiname der betreffenden Datei.

Beispiel

```
DateiAttrib=        GetFileAttributes("C:\TEST\BEISPIEL.DAT")
MessageBox          (MVar;"Dateiattribut:"; DateiAttrib)
```

GetFileDateAndTime — Datum/Uhrzeit der Erstellung

Befehlsform: *DatumZeit*=**GetFileDateAndTime**(*Dateiname*)

Ermitteln von Datum und Uhrzeit der Dateierstellung.

DatumZeit Datum und Uhrzeit der Dateierstellung in Form von Sekunden seit dem 1. Januar 1980 (-1 bedeutet einen Fehler).

Dateiname Vollständiger Dateiname der betreffenden Datei.

Beispiel

```
DatumZeit=          GetFileDateAndTime(Dateiname)
MessageBox          (MVar;"Datum und Uhrzeit:"; DatumZeit)
```

OpenFile — Datei öffnen

Befehlsform: *DateiKenn*=**OpenFile**(*Dateiname*;*Zugriffsart*;*DateiStatus*;*Datentyp*)

Durch diesen Befehl wird eine Datei zum Lesen (= Eingabe) und/oder Schreiben (= Ausgabe) geöffnet. Eine Datei kann nur einmal geöffnet werden, d. h. ein *OpenFile*-Befehl auf eine bereits geöffnete Datei erzeugt eine Fehlermeldung. Unmittelbar nach dem Öffnen befindet sich der Positionsanzeiger vor dem ersten Datensatz.

DateiKenn Dieses Feld enthält nach der Ausführung des Befehls eine Dateikennung. Werden mehrere Dateien in einem Makro geöffnet und gleichzeitig offen gehalten, ist für jede Datei eine eigene Variable zu benennen, da diese Variable auch zum Lesen, Schreiben, Schließen usw. dieser Datei wieder benötigt wird (gilt für alle File-Befehle, die mit der Datei-Kennung *FileID* arbeiten). Sie darf darum nicht durch das Öffnen einer neuen Datei überschrieben werden, solange die bisherige Datei noch nicht durch *FileClose* geschlossen wurde. Ist ein negativer Wert enthalten, wurde beim Öffnen ein Fehler erkannt (FileError).

Dateiname Eingabe des vollständigen Dateinamens. Wurde ein Dateiname ohne Laufwerk- und Pfadangabe eingegeben, versucht WordPerfect die Datei in folgender Reihenfolge in den genannten Ordnern zu finden: aktueller Ordner, Windows-Ordner, Windows-System-Ordner, aktuelle Programm-Ordner, DOS-Ordner, NetzwerkOrdner (wenn gemapped).

Zugriffsart In Anhängigkeit der Zugriffsart kann eine Datei gelesen oder geschrieben werden. Der Parameter ist optional. Wird er weggelassen, wird automatisch *Read!* (= lesen) angenommen.

Append!	Daten werden an eine bestehende Datei angehängt.
Exists!	Öffnen und Schließen einer Datei, um zu testen, ob die angegebene Datei vorhanden ist. Es wird beim *Read!* oder *ReadWrite!* eine Fehlermeldung erzeugt, wenn die Datei nicht vorhanden ist.
Read!	Lesen einer Datei in Abhängigkeit von *FilePosition*.
ReadWrite!	Lesen und Schreiben einer Datei. Dadurch können gelesene Daten in Abhängigkeit von FilePosition wieder auf ihre bisherige Position zurückgeschrieben werden.
Write!	Schreiben von Daten in Abhängigkeit von FilePosition.
WriteNew!	Erstellen einer neuen Datei. Die bisherigen Daten dieser Datei gehen verloren.

DateiStatus Erlaubt oder unterbindet den gleichzeitigen Zugriff auf eine Datei von mehreren Makros. Der Parameter ist optional. Wird er weggelassen, wird automatisch *None!* (= keine) angenommen. Verwenden Sie diese Parameter besonders dann, wenn in einem Netzwerk mehrere Personen gleichzeitig auf eine Datei zugreifen wollen.

None!	Keine Zuordnung.
Compatibility!	Datei kann von mehreren Makros gleichzeitig geöffnet werden (auch mehrmals).
Exclusive!	Datei kann nur von einem Makro gelesen oder geschrieben werden.
DenyNone!	Andere Makros dürfen lesen und schreiben.
DenyRead!	Andere Makros dürfen nicht lesen.
DenyWrite!	Andere Makros dürfen nicht schreiben.

Datentyp In welcher Dateiform ist die Datei gespeichert. Der Parameter ist optional. Wird er weggelassen, wird automatisch *AnsiText!* angenommen.

AnsiText!	Datei im *Ansi*-Format.
OEMText!	Datei im *OEM*-Format.
WPTEXT!	Nur das Lesen oder Schreiben von Zeichen-Strings ist erlaubt. Es kann kein WordPerfect-Dokument damit erstellt werden.

Beispiel

```
DatKennz1 =          OpenFile(Dateiname; WriteNew!;DenyNone!; AnsiText!)
If(DatKennz1 < 0)
   MessageBox        (MVar; "Hinweis"; "Fehler beim Öffnen."; IconExclamation!)
EndIf
```

Die unter ***Dateiname*** angegebene Datei wird als Ausgabedatei geöffnet. Aufgrund von *WriteNew!* werden alle bisherigen Datensätze gelöscht.

RenameFile — Datei umbenennen

Befehlsform: *Boolean* **RenameFile**(*DateiAlt;DateiNeu;Meldung*)

Umbenennen oder Verschieben einer Datei.

DateiAlt Eingabe der umzubenennenden Datei (vollständiger Dateiname).

DateiNeu Eingabe des neuen Dateinamens (vollständiger Dateiname). Wird ein anderer Ordner angegeben, wird die Datei in diesen Ordner verschoben.

Meldung Anzeigen einer Meldung über *Prompts!*, wenn die umzubenennende Datei nicht existiert oder ein falscher Pfadname angegeben wurde.

Beispiel

```
If(RenameFile ("C:\BEISPIEL.DAT";"C:\BEISPIEL.TXT") = False)
   MessageBox(MVar; "Hinweis"; "Fehler beim Umbenennen."; IconExclamation!)
EndIf
```

SetFileAttributes

Dateiattribute setzen

Befehlsform: *Boolean* **SetFileAttributes**(*Dateiname;Attribute;Meldung*)

Setzen von Dateiattributen.

Dateiname Eingabe des vollständigen Dateinamens der betreffenden Datei.

Attribute Eingabe des gewünschten Attributs:

Archived!	Directory!
Hidden!	Label!
Normal!	ReadOnly!
System!	

Für weitere Informationen zu diesen Parametern schlagen Sie bitte in Ihrem DOS/Windows-Handbuch nach.

Meldung Anzeigen einer Meldung über *Prompts!*, wenn ein Attribut nicht existiert.

Beispiel

```
If                (Not(SetFileAttributes ("C:\BEISPIEL.TXT";ReadOnly!) = True))
   MessageBox     (MVar; "Hinweis"; "Fehler bei Datei-Attributen."; IconExclamation!)
EndIf
```

Die angegebene Datei wird mit dem Datei-Attribut *ReadOnly!* (= nur Lesen) versehen.

SetFileDateAndTime

Datum/Uhrzeit setzen

Befehlsform: *Boolean* **SetFileDateAndTime**(*Dateiname;DatumZeit*)

Ändern von Datum und Uhrzeit der Dateierstellung.

Dateiname Vollständiger Dateiname der betreffenden Datei.

DatumZeit Datum und Uhrzeit der Dateierstellung in Form von Sekunden seit dem 1. Januar 1980.

Beispiel

```
If                (SetFileDateAndTime(Dateiname;"511271000") = True)
   MessageBox     (MVar; "Hinweis"; "Datum/Uhrzeit wurden geändert."; IconExclamation!)
EndIf
```

Die unter ***Dateiname*** angegebene Datei wird mit Datum und Uhrzeit vom 25.3.1995, 12:02 versehen.