6.51 For

Befehlsform: **For**(*Variable;Start;Test;Abstand*)

Dieser Befehl vereinfacht die Schleifenverarbeitung, indem er die nachfolgenden Befehle bis zum nächsten zugehörigen *EndFor*-Befehl so oft wiederholt, wie in den Parametern des Befehls angegeben wurde, sofern sie nicht vorher durch andere Bedingungen innerhalb der Schleife (z. B. If, Break) beendet wird. Die Werte für *Variable*, *Start*, *Test* und *Abstand* können Sie als fixe Werte oder als Variable angeben. Auch die Kombination beider Möglichkeiten ist erlaubt. Sofern Sie mit Variablen arbeiten, achten Sie bitte darauf, daß diese vor der Ausführung des Befehls auch gültige Werte enthalten. Ordnen Sie diese ggf. vor der Ausführung von *For* mit dem Befehl *Assign* zu. Werte können Sie natürlich auch z. B. durch die Befehle *GetNumber* oder *GetString* in die entsprechenden Variablen über die Tastatur eingeben oder innerhalb des Makros berechnen lassen.

Falsche Werte können zu unvorhersehbaren Ergebnissen führen (z. B. Endlosschleife).

Jede Befehlsfolge einer *For*-Anweisung muß zwingend mit einem *EndFor*-Befehl enden. Die zwischen *For* und *EndFor* definierten Befehle werden aufgrund der unter *For* angegebenen Werte wiederholt (= Schleife). Durch *EndFor* erfolgt ein Rücksprung zu dem zugehörigen *For*-Befehl (*For*-Schleifen können geschachtelt sein, d. h. innerhalb einer *For*-Struktur kann eine weitere *For*-Struktur enthalten sein, die wiederum eine *For*-Struktur beinhalten kann). Auch die Schachtelung in Verbindung mit *ForNext* oder *While* ist erlaubt. Die Ausführung wird wieder beim zugehörigen *For* fortgesetzt, wobei die Variablen dieses *For*-Befehls entsprechend verändert werden (siehe unten). Dieser Vorgang wird so lange wiederholt, bis die Schleife(n) so oft abgearbeitet wurden, wie unter *For* angegeben wurde. Die Verarbeitung wird dann mit dem Befehl fortgesetzt, der dem zugehörigen *EndFor*-Befehl folgt. Über *Break* kann die Schleife in Verbindung mit einer Bedingung auch vorher abgebrochen werden.

Parameter

Variable Beliebiger Variablenname. Diese Variable dient als Zähler. Sie enthält immer einen Wert, der im aktuellen Schleifendurchgang dem Makro für die Verarbeitung oder Steuerung zur Verfügung steht.

Start Numerischer oder alphanumerischer Ausdruck bzw. eine Maßeinheit. Bei der Prüfung der *For*-Bedingung wird mit dem Wert dieser Variablen begonnen. Für den Rest der Schleifenverarbeitung wird dieser Wert ignoriert.

Test Die *For*-Schleife wird beendet, wenn die hier angegebene Bedingung zutrifft. Erlaubt sind numerische Werte, Variable mit numerischem Inhalt sowie Vergleichsoperationen.

Abstand Über diese Variable wird das »Hoch-« oder »Runterzählen« geregelt. Erlaubt sind hier numerische Werte, Variable mit numerischem Inhalt sowie Vergleichsoperationen, die ein numerisches Ergebnis liefern.

Beispiel 1

```
For                 (Zähler;1;Zähler<16;Zähler+1)
   Type             ("Schleifendurchlauf: ")
   Type             (Zähler)
   HardReturn       ()
EndFor
Type                ("Ende der Schleife")
```

Nach Ausführung der Befehle wird folgender Text am Bildschirm angezeigt:

```
Schleifendurchlauf: 1
Schleifendurchlauf: 2
Schleifendurchlauf: 3
::::::
Schleifendurchlauf: 15
Ende der Schleife
```

Wie Sie aufgrund des Beispiels erkannt haben, wird der angegebene Text fünfzehnmal gedruckt. Der Befehl wird wie folgt gelesen:

Beginne bei dem unter *Start* angegebenen Wert (1) und führe die zwischen *For* und *EndFor* definierten Befehle so oft aus, bis in der Variablen *Zähler* der Wert erreicht wurde, der unter *Test* (= Zähler<16) angegeben wurde, indem bei jeder Wiederholung der Wert addiert wird, der unter *Abstand* (= Zähler+1) definiert wurde.

Der Befehl wird wie folgt ausgeführt:

1. For prüft, ob die Bedingungen noch zutreffen.

   Wenn ja, weiter bei Punkt 2 (dies ist in unserem Beispiel dann der Fall, wenn *Zähler* einen Wert von 1 – 15 (einschließlich) enthält).

   Wenn nein, weiter mit den Befehlen, die hinter dem Befehl *EndFor* folgen (hier: Wenn *Zähler* den Wert 16 enthält). Dies kann z. B. der Rücksprung zu einem aufrufenden Befehl sein, wenn es sich hier um eine Unterroutine handelt.

2. Der Text »Schleifendurchlauf« wird gedruckt, gefolgt von dem Inhalt der Variablen *Zähler*.

3. *Zähler* wird um den unter »Abstand« angegebenen Wert erhöht (hier: 1). Nach dem ersten Durchlauf enthält *Zähler* den Wert 2, nach dem zweiten Durchlauf den Wert 3 usw.

4. *EndFor* veranlaßt einen Rücksprung nach 1.

Würde unter *Abstand* anstelle des Wertes 1 der Wert 2 stehen, würde *Zähler* bei jedem Schleifendurchlauf um 2 erhöht. Der Ausdruck würde dann wie folgt aussehen:

```
Schleifendurchlauf: 1
Schleifendurchlauf: 3
Schleifendurchlauf: 5
:::::
Schleifendurchlauf: 15
```

Beispiel 2

Aufgrund der folgenden *For*-Anweisung wird das kleine Einmaleins von 5 errechnet und das Ergebnis wie folgt ausgedruckt:

```
1         *                   5          =          5
2         *                   5          =          10
:
10        *                   5          =          50
```

```
1         Assign              (Multiplikator;5)
2         For                 (Zähler;1;Zähler<11;Zähler+1)
3           Ergebnis=         (Zähler*Multiplikator)
4           Type              (Zähler)
            Tab               ()
            Type              (" * ")
            Tab               ()
            Type              (Multiplikator)
            Tab               ()
            Type              (" = ")
            Tab               ()
            Type              (Ergebnis)
            HardReturn        ()
5         EndFor
6         Type                ("Ende der Berechnung")
```

Vor dem Berechnen wird der Variablen *Multiplikator* der Wert 5 zugeordnet. Die Variable *Zähler* braucht vor der Ausführung in diesem Fall nicht initialisiert zu werden, da beim erstmaligen Ausführen des *For*-Befehls dies automatisch geschieht.

1 Der Variablen *Multiplikator* wird der Wert 5 zugewiesen. Wenn Sie hier eine andere Zahl angeben, wird das kleine Einmaleins dieser Zahl gedruckt, d. h. wenn Sie diese Variable mit 12 initialisieren, wird das Einmaleins von 12 errechnet.

2 Die Schleife wird 10 mal ausgeführt (Start = 1, Stop = 10, Abstand = 1). Der aktuelle Schleifendurchlauf wird in *Zähler* gespeichert.

3 Errechnen des Ergebnisses einer Zeile. Die Variable *Ergebnis* speichert das Ergebnis, *Zähler* dient hier als Multiplikant, die Variable *Multiplikator* als Multiplikator.

4 Drucken einer Zeile, nachdem das Ergebnis ermittelt wurde. Durch die Befehle *Type* wird fixer Text (*, =) und die Variableninhalte auf einer Tabulatorposition gedruckt.

Durch den Befehl *HardReturn()* wird die gerade gedruckte Zeile beendet und der Cursor an den Beginn einer neuen Zeile gesetzt.

5 Rücksprung zum *For*-Befehl, solange die Schleife noch nicht vollständig abgearbeitet wurde.

6 Der Text »Ende der Berechnung« wird gedruckt, nachdem die *For*-Schleife beendet wurde.

Wie Sie das Einmaleins einer beliebigen Zahlenreihe drucken können, finden Sie bei der Beschreibung des Befehls *While*.

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

Weitere Hinweise siehe: Break, EndFor, Foreach, ForNext, Repeat, Until, While

6.52 ForEach

Befehlsform: **ForEach**(*Variable;{Ausdruck1;Ausdruck2;...}*)

Dieser Befehl vereinfacht die Schleifenverarbeitung, indem er die nachfolgenden Befehle bis zum nächsten zugehörigen *EndFor*-Befehl so oft wiederholt, wie *Ausdruck* vorhanden ist, sofern sie nicht vorher durch andere Bedingungen innerhalb der Schleife (z. B. If, Break) beendet wird. Wurden dort z. B. sechs *Ausdruck..* verwendet, wird die Schleife sechsmal wiederholt. *Ausdruck..* können fixe Werte sein oder Variable. Auch die Kombination beider Möglichkeiten ist erlaubt. Sofern Sie mit Variablen arbeiten, achten Sie bitte darauf, daß diese vor der Ausführung des Befehls gültige Werte enthalten. Geben Sie diese entweder über Tastatur ein, oder ordnen Sie die Werte vor der Ausführung von *ForEach* mit dem Befehl *Assign* zu. Der Befehl *ForEach* ist dem Befehl *For* ähnlich. Der Unterschied besteht darin, daß die Schleife so oft ausgeführt wird, wie *Ausdruck..* hinter *ForEach* angegeben sind.

Beim ersten Schleifendurchlauf wird *Ausdruck1* verarbeitet, beim zweiten Durchlauf *Ausdruck2* usw., bis alle definierten Ausdrücke der Reihe nach von links nach rechts abgearbeitet wurden.

Falsche Werte können zu unvorhersehbaren Ergebnissen führen. Jede Befehlsfolge einer *ForEach*-Anweisung muß zwingend mit einem *EndFor*-Befehl beendet werden. *EndFor* bildet den Abschluß einer Befehlsfolge, die mit *ForEach* beginnt. Die Ausführung dieses Befehls bewirkt den Rücksprung zu *ForEach*. Zwischen *ForEach* und *EndFor* können Sie fast alle Makrobefehle einschließen.

Parameter

Variable Beliebiger Variablenname. Dieser Variablen werden während der Ausführung der *ForEach*-Schleife der Reihe nach (von links nach rechts) die vorhandenen Ausdrücke zugewiesen, damit sie innerhalb der Schleife verarbeitet werden können.

Ausdruck.. Die zwischen *ForEach* und *EndFor* definierten Befehle werden in Abhängigkeit der Anzahl von *Ausdruck..* wiederholt. Sind z. B. 15 Ausdrücke vorhanden, wird die Schleife fünfzehnmal wiederholt. Als *Ausdruck..* können Sie numerische oder alphanumerische Werte sowie Variable verwenden.

Beispiel

```
Text1=                          "Ende der Schleife"
ForEach
(Drucktext;{"Fritz";"Franz";"Friedrich";"Hugo";"12345";Text1})
```

```
   Type                    (Drucktext)
   HardReturn              ()
EndFor
```

Nach der Ausführung dieses Befehls wird folgendes gedruckt:

```
Fritz
Franz
Friedrich
Hugo
12345
Ende der Schleife
```

Die in den geschweiften Klammern angegebenen, durch Hochkommata eingeschlossenen Konstanten, werden in der angegebenen Reihenfolge in die Variable *Drucktext* übertragen und anschließend gedruckt. Am Ende wird der Inhalt der Variablen *Text1* gedruckt.

Weitere Hinweise siehe: Break, EndFor, ForNext, Until, While

6.53 ForNext

Befehlsform: *ForNext*(*Variable;Start;Ende;*[*Intervall*])

Dieser Befehl vereinfacht die Schleifenverarbeitung, indem er die nachfolgenden Befehle bis zum nächsten zugehörigen *EndFor*-Befehl so oft wiederholt, wie in den Parametern des Befehls angegeben wurde, sofern sie nicht vorher durch andere Bedingungen innerhalb der Schleife (z. B. If, Break) beendet wird. Die Werte für *Variable*, *Start*, *Ende* und *Intervall* können Sie als fixe Werte oder als Variable angeben. Auch die Kombination beider Möglichkeiten ist erlaubt. Sofern Sie mit Variablen arbeiten, achten Sie bitte darauf, daß diese vor der Ausführung des Befehls auch gültige Werte enthalten. Ordnen Sie diese ggf. vor der Ausführung von *ForNext* mit dem Befehl *Assign* zu. Werte können Sie natürlich auch z. B. durch die Befehle *GetNumber* oder *GetString* in die entsprechenden Variablen über die Tastatur eingeben oder durch Rechenoperationen errechnen lassen. Die Unterschiede zu *For* bestehen in der Behandlung der Variablen *Ende* und *Intervall*.

Falsche Werte können zu unvorhersehbaren Ergebnissen führen (z. B. Endlosschleife).

Jede Befehlsfolge einer *ForNext*-Anweisung muß zwingend mit einem *EndFor*-Befehl enden. Die Ausführung dieses Befehls bewirkt den Rücksprung zu *ForNext*. Die zwischen *ForNext* und *EndFor* definierten Befehle werden aufgrund der unter *ForNext* angegebenen Werte wiederholt (= Schleife). Durch *EndFor* erfolgt ein Rücksprung zu dem zugehörigen *ForNext*-Befehl (*ForNext*-Schleifen können geschachtelt sein, d. h. innerhalb einer *ForNext*-Struktur kann eine weitere *ForNext*-Struktur enthalten sein, die wiederum eine *ForNext*-Struktur beinhalten kann). Die Ausführung wird wieder beim zugehörigen *ForNext* fortgesetzt, wobei die Variablen dieses *ForNext*-Befehls entsprechend verändert werden (siehe unten). Dieser Vorgang wird so lange wiederholt, bis die Schleife so oft abgearbeitet wurde, wie unter *ForNext*

angegeben wurde. Die Verarbeitung wird mit dem Befehl, der dem zugehörigen *EndFor*-Befehl folgt, fortgesetzt. Über *Break* kann die Schleife in Verbindung mit einer Bedingung auch vorher abgebrochen werden.

Parameter

Variable Beliebiger Variablenname. Diese Variable dient als Zähler. Sie enthält immer einen Wert, der im aktuellen Schleifendurchgang dem Makro für die Verarbeitung oder Steuerung zur Verfügung steht.

Start Numerischer oder alphanumerischer Ausdruck bzw. eine Maßeinheit. Bei der Abprüfung der *ForNext*-Bedingung wird mit dem Wert dieser Variablen begonnen. Für den Rest der Schleifenverarbeitung wird dieser Wert ignoriert.

Ende Die *ForNext*-Schleife wird beendet, wenn der hier angegebene Wert erreicht wird. Erlaubt sind numerische Werte und Variable mit numerischem Inhalt. Im Gegensatz zu *For* sind hier keine Vergleichsoperationen erlaubt (führen zu falschen Ergebnissen).

Intervall Über diese Variable wird das »Hoch-« oder »Runterzählen« geregelt. Erlaubt sind hier numerische Werte und Variable mit numerischem Inhalt. Im Gegensatz zu *For* sind hier keine Vergleichsoperationen erlaubt (führen zu falschen Ergebnissen). Wird dieser Parameter weggelassen, wird 1 angenommen.

Beispiel 1

```
ForNext              (Zähler;1;15;1)
   Type              ("Schleifendurchlauf: ")
   Type              (Zähler)
   HardReturn        ()
EndFor
Type                 ("Ende der Schleife")
```

Nach Ausführung der Befehle wird folgender Text am Bildschirm angezeigt:

```
Schleifendurchlauf: 1
Schleifendurchlauf: 2
Schleifendurchlauf: 3
::::::
Schleifendurchlauf: 15
Ende der Schleife
```

Wie Sie aufgrund des Beispiels erkannt haben, wird der angegebene Text fünfzehnmal gedruckt. Der Befehl wird wie folgt gelesen:

Beginne bei dem unter »Start« angegebenen Wert (1) und führe die zwischen *ForNext* und *EndFor* definierten Befehle so oft aus, bis in der Variablen *Zähler* der Wert erreicht wurde, der in der Variablen *Ende* (= 15) angegeben wurde, indem bei jeder Wiederholung der Wert addiert wird, der unter *Intervall* (= 1) definiert wurde.

Der Befehl wird wie folgt ausgeführt:

1. *ForNext* prüft, ob die Bedingung noch zutrifft.

   Wenn ja, weiter bei Punkt 2. Dies ist in unserem Beispiel dann der Fall, wenn *Ende* einen Wert von 1 – 15 (einschließlich) enthält.

   Wenn nein, weiter mit den Befehlen, die hinter dem zugehörigen Befehl *EndFor* folgen (hier: Wenn *Zähler* den Wert 16 enthält). Dies kann z. B. der Rücksprung zu einem aufrufenden Befehl sein, wenn es sich hier um eine Unterroutine handelt.

2. Der Text »Schleifendurchlauf« wird gedruckt, gefolgt von dem Inhalt der Variablen *Zähler*.

3. *Zähler* wird um den unter *Intervall* angegebenen Wert erhöht (hier: 1). Nach dem ersten Durchlauf enthält *Zähler* den Wert 2, nach dem zweiten Durchlauf den Wert 3 usw.

4. *EndFor* veranlaßt einen Rücksprung nach 1 (zugehöriger *ForNext*-Befehl).

Würde unter *Intervall* anstelle des Wertes 1 der Wert 2 stehen, würde *Zähler* bei jedem Schleifendurchlauf um 2 erhöht. Der Ausdruck würde dann wie folgt aussehen:

```
Schleifendurchlauf: 1
Schleifendurchlauf: 3
Schleifendurchlauf: 5
Schleifendurchlauf: 7
:::::
Schleifendurchlauf: 15
```

Beispiel 2

Aufgrund der folgenden *ForNext*-Anweisung wird das kleine Einmaleins von 5 errechnet, und das Ergebnis wie folgt ausgedruckt:

```
1         *                5          =          5
2         *                5          =          10
3         *                5          =          15
10        *                5          =          50
          Application      (A1; "WordPerfect"; Default; "DE")
          FileNew          ()
          Display          (On!)
          TabSet           (Relative!; {Position: 4.5403c; TabstoppTyp: TabRight!;
                           Position: 5.5393c; TabstoppTyp: TabLeft!; Position: 7.5396c;
                           TabstoppTyp: TabRights!; Position: 8.5408c; TabstoppTyp:
                           TabLeft!; Position: 10.5389c; TabstoppTyp: TabLeft!})

1         Multiplikator=   5
1         Anfang=          1
1         Ende=            10
1         Intervall=       1
2         ForNext          (Zähler; Anfang; Ende; Intervall)
3           Ergebnis=      (Zähler*Multiplikator)
            Tab            ()
            Type           (Zähler)
```

```
            Tab                 ()
            Type                ("*")
            Tab                 ()
            Type                (Multiplikator)
            Tab                 ()
            Type                ("=")
            Tab                 ()
            Type                (Ergebnis)
            HardReturn          ()
5         EndFor
          HardReturn            ()
6         Type                  ("***   Ende der Berechnung   ***")
```

Vor dem Berechnen wird der Variablen Multiplikator der Wert 5 zugeordnet. Die Variable Zähler braucht vor der Ausführung in diesem Fall nicht initialisiert zu werden, da beim erstmaligen Ausführen von ForNext dies automatisch geschieht.

1 Der Variablen Multiplikator wird der Wert 5 zugewiesen. Wenn Sie hier eine andere Zahl angeben, wird das kleine Einmaleins dieser Zahl gedruckt, d. h. wenn Sie diese Variable mit 12 initialisieren, wird das Einmaleins von 12 errechnet. Den Variablen Anfang, Ende und Intervall werden Anfangswerte zugewiesen.

2 Die Schleife wird 10mal ausgeführt (Anfang = 1, Ende = 10, Intervall = 1). Der aktuelle Schleifendurchlauf wird in Zähler gespeichert. Die Variable Intervall kann in diesem Beispiel auch entfallen, da standardmäßig 1 angenommen wird.

3 Errechnen des Ergebnisses einer Zeile. Die Variable Ergebnis speichert das Ergebnis, Zähler dient hier als Multiplikant, die Variable Multiplikator als Multiplikator.

4 Drucken einer Zeile, nachdem das Ergebnis ermittelt wurde. Durch die Befehle Type wird fixer Text (* bzw. =) und die Variableninhalte jeweils auf einer Tabulatorposition gedruckt.

  Durch den Befehl HardReturn() wird die gerade gedruckte Zeile beendet, und der Cursor an den Beginn einer neuen Zeile gesetzt.

5 Rücksprung zum ForNext-Befehl, solange die Schleife noch nicht vollständig abgearbeitet wurde.

6 Der Text »*** Ende der Berechnung ***« wird gedruckt, nachdem die ForNext-Schleife beendet wurde.

Wie Sie das Einmaleins einer beliebigen Zahlenreihe drucken können, finden Sie bei der Beschreibung des Befehls While.

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

Weitere Hinweise siehe: Break, EndFor, For, ForEach, Repeat, Until, While

6.54 Fraction

Befehlsform:	**Variable=Fraction**(NumAusdruck) **Fraction(Variable;NumAusdruck)** (WPWin 5.2)

Der Dezimalbruch der Variablen NumAusdruck (z. B. ein ermitteltes Rechenergebnis) wird einer Variablen zugeordnet. Wurde z. B. bei einer Rechenoperation das Ergebnis 2,50 ermittelt, wird der Wert hinter dem Komma (0,50) in der angegebenen Variablen gespeichert. Beachten Sie bitte auch Kapitel »5.17 Änderungen gegenüber WPWin 5.1/5.2«.

Parameter

Variable Beliebiger Variablenname. Enthält nach der Ausführung des Befehls den Dezimalbruch.

NumAusdruck Numerischer Wert oder Ausdruck, aus dem der Dezimalbruch zu ermitteln ist.

Beispiel

Die Befehle Fraction und Integer in dem folgenden Beispiel dienen nur zu Demonstrationszwecken. Die Mehrwertsteuer läßt sich natürlich mit dem folgenden Befehl auch einfacher berechnen:

```
                SteuerDMPF= Betrag*0.15   oder   SteuerDMPF= Betrag*0,15

                Application         (A1;"WordPerfect";Default;"DE")
                Labe                (Anfang)
1               GetNumber           (Betrag;"Betrag eingeben.";"MWSt-Berechnung")
2               SteuerDM=           Integer(Betrag*0.15)
3               SteuerPf=           Fraction(Betrag*0.15)
4               SteuerDMPF=         (SteuerDM+SteuerPf)
5               Prompt              ("Ermittelte Mehrwertsteuer:";SteuerDMPf;;;)
6               Wait                (30)
7               EndPrompt
8               Go                  (Anfang)
```

Dieses Makro ermittelt die Mehrwertsteuer (15%) aus einem eingegebenen Betrag.

1 Anzeigen eines Dialogfeldes zum Eingeben eines Wertes. Der eingegebene Wert wird in Betrag gespeichert. Trennen Sie Ganzzahl und Dezimalbruch durch einen Dezimalpunkt bzw. Dezimalkomma.

2 Ermitteln der Steuer (Wert vor dem Komma = Ganzzahl). Auch folgende Möglichkeit ist erlaubt: Integer(SteuerDM;Betrag*0.15).

3 Ermitteln der Steuer (Wert hinter dem Komma =Dezimalbruch). Auch folgende Möglichkeit ist erlaubt: Fraction(SteuerPF;Betrag*0.15).

4 Ganzzahl und Dezimalbruch addieren.

5 Text und Mehrwertsteuer in einem Dialogfeld anzeigen.

6 Dauer der Anzeige.

7 Dialogfeld wieder ausblenden.

8 Rücksprung an den Anfang des Makros (Schleife).

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

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

Weitere Hinweise siehe: Integer

6.55 FractionStr

Befehlsform: **Variable=FractionStr**(Variable;Nenner;Optionen)

Dieser Befehl wandelt einen numerischen Bruch in einen String um.

Parameter

Variable Variable enthält nach der Ausführung des Befehls den umgewandelten Wert.

Daten Enthält die umzuwandelnden Daten.

Nenner Der dem Nenner am nächsten liegende Bruch wird zurückgegeben. Wird der Parameter weggelassen bzw. ist er kleiner oder gleich 0, wird der am nächsten liegende Nenner verwendet.

Optionen Legt die Form des Bruches für Werte fest, die größer als 0 sind. Wird keine Option angegeben, wird ImproperFraction! angenommen.

ImproperFraction!	Zähler ist größer als der Nenner.
MixedFraction!	Eine Ganzzahl, gefolgt von einem echten Bruch (Zähler ist kleiner als der Nenner).

Beispiel

```
Ergebnis=  FractionStr(4,5;0;MixedFraction!)
Type       (Ergebnis)
Ergebnis=  FractionStr(4,5;8;MixedFraction!))
Type       Ergebnis
```

Als Ergebnis wird »4 1/2« bzw. »4 4/8« gedruckt.

6.56 Function

Befehlsform: **Function Name (**Parameter;Parameter;...)

Über diesen Befehl wird der Beginn einer Befehlsfolge (Unterroutine) definiert, die bei Bedarf aufgerufen werden kann. Der Aufruf ist von jeder beliebigen Stelle des aktuellen Makros aus möglich. In Verbindung mit Use können auch Function-Routinen anderer Makros aufgerufen werden. Der Unterschied zu anderen aufrufenden Befehlen besteht darin, daß entweder eine oder mehrere Variable an die Unterroutine weitergegeben, oder daß Ergebnisse an den aufrufenden Befehl zurückgegeben werden können. Die Makro-Ausführung wird an der aufrufenden Stelle unterbrochen, um die unter Name angegebene Unterroutine auszuführen. Die Unterroutine muß mit EndFunc oder mit Return enden, wobei über Return bei Bedarf der Wert einer Variablen zurückgegeben werden kann (es wird 0 zurückgegeben, wenn keine Ergebnisse zurückgemeldet werden sollen). Danach wird mit dem Befehl fortgefahren, der dem aufrufenden Befehl folgt. Function-Routinen werden im Gegensatz zu Label-Routinen nur dann ausgeführt, wenn sie aufgerufen wurden (wird eine solche Routine z. B. direkt am Makroanfang definiert, wird sie trotzdem erst dann ausgeführt, wenn sie aufgerufen wird). Um den Wert der übergebenen Variablen zu ändern, können Sie der betreffenden Variablen des aufrufenden Befehls und der zugehörigen Variablen in dem Function-Befehl ein & voranstellen. In diesem Fall wird die Adresse (Speicheradresse) der Variablen übergeben und nicht deren Inhalt.

Auch Arrays können in Verbindung mit Function-Routinen verwendet werden. Die Handhabung ist mit der Beschreibung in den nachfolgenden Beispielen identisch. Achten Sie darauf, daß dabei die Angabe von Elementen in eckigen Klammern [ ] vorgenommen wird. Wird ein komplettes Array übergeben, muß sichergestellt sein, daß alle Elemente einen Wert enthalten, sonst erfolgt bei der Ausführung eine Fehlermeldung.

Parameter

Name Der Name ist frei wählbar, er muß jedoch mit einem Buchstaben beginnen. Verwenden Sie aussagefähige Namen, die auf die Funktion der Routine Bezug nehmen. Wird in der Routine eine Zinsberechnung durchgeführt, könnte man die Routine z. B. Zinsen nennen. Verwenden Sie keine zu langen Bezeichnungen (max. 30 Zeichen), denn diese Namen müssen Sie ggf. mehrmals verwenden (wenn Routine öfters benötigt wird), wobei das Eintippen dann mühselig sein kann (Fehlerquelle!). Enthält ein Makro mehrere Function-Routinen, müssen diese unterschiedliche Namen aufweisen. Kein Name darf doppelt vergeben werden, ansonsten erfolgt bei der Kompilierung eine Fehlermeldung.

Parameter Dieser Variablen wird von dem aufrufenden Befehl ein Wert übergeben, mit dem in der Function-Routine weitergearbeitet werden kann. Mehrere Variable sind jeweils durch ein Semikolon zu trennen. Wird der Variablen ein & vorangestellt, muß auch der zugehörigen Variablen unter Function ein & vorangestellt werden.

Beispiel

```
1.      Feld1=          "Karl"
        Feld2=          Aufruf(Feld1)               //(aufrufender Befehl)
        MessageBox      (Mvar;"Function-Befehl testen";Feld2)

        Function        Aufruf(Feld3)
        Feld3=          Feld3 + " Meier"
        Return          (Feld3)
        EndFunc

        Call            (Berechnen)
```

Beim Aufruf der Function-Routine wird der Inhalt von Feld1 (= Karl) übergeben und in Feld3 gespeichert. Innerhalb der Routine wird in Feld3 die Konstante »Meier« ergänzt, so daß danach in Feld3 der Inhalt »Karl Meier« vorhanden ist. Return gibt den generierten Inhalt von Feld3 an den aufrufenden Befehl (= Feld2) zurück. Nach der Ausführung von Return und EndProc enthält Feld2 darum den in der Function-Routine generierten Inhalt »Karl Meier«. Würde in diesem Fall Return weggelassen werden oder keinen Wert zurückgeben, würde in Feld2 0 gespeichert. Feld1 bleibt unverändert. Nach der Ausführung von EndProc wird der dem aufrufenden Befehl folgende Befehle (= MessageBox) ausgeführt und anschließend der Call-Befehl. Die Function-Routine wird übersprungen und nicht nochmals ausgeführt.

```
2.      Feld1=          "Karl"
        Feld2=          Aufruf(&Feld2)              //(aufrufender Befehl)
        MessageBox      (Mvar;"Function Prototype testen";Feld2)

        Function        Aufruf(&Feld3)
        Feld3=          Feld3 + " Meier"
        Return          (Feld3)
        EndFunc
```

Beim Aufruf der Function-Routine wird der Inhalt von Feld1 (= Karl) übergeben und in Feld3 gespeichert. Innerhalb der Routine wird in Feld3 die Konstante »Meier« ergänzt, so daß danach in Feld3 der Inhalt »Karl Meier« vorhanden ist. Return gibt den ermittelten Wert von Feld3 an den aufrufenden Befehl (= Feld2) zurück. Nach der Ausführung von Return und EndProc enthält Feld2 darum den in der Function-Routine generierten Inhalt »Karl Meier«.

Würde in diesem Fall Return weggelassen werden oder keinen Wert zurückgeben, würde in Feld2 0 gespeichert. Durch die Verwendung von & bei der Variablenübergabe enthält jetzt auch Feld1 den Inhalt »Karl Meier«.

Weitere Hinweise siehe: Procedure, Use

6.57 Function Prototype

Befehlsform: **Function Prototype Name** *(Parameter;Parameter;...)*

Um Makros ausführen zu können, müssen diese unter einem eindeutigen Namen gespeichert und kompiliert werden. Beim Kompilieren werden nur formale, aber keine logischen Fehler erkannt. So kann es durchaus vorkommen, daß ein mit Run oder Use aufgerufenes Makro fehlerfrei kompiliert wurde, die Ausführung jedoch einen logischen Fehler erzeugt. Um diese Run-Time-Fehler auszuschließen, kann über Function Prototype die Syntax einer Function-Routine überprüft werden. Verwenden Sie diesen Befehl in dem betreffenden Hauptmakro, um bereits zur Compile-Zeit Fehler zu ermitteln.

Parameter

Name Name der zu testenden Function-Routine, der mit einem Buchstaben beginnen muß.

Parameter Verwenden Sie hier dieselben Parameter, die auch bei der zu testenden Function-Routine verwendet werden. Mehrere Parameter sind jeweils durch ein Semikolon zu trennen. Wird ein Parameter mit einem Ampersand verwendet (z. B. &Feld1) muß auch die zugehörige Variable im Function-Befehl ein Ampersand enthalten.

Beispiel

```
Application          (A1;"WordPerfect";Default;"DE")
Function             Prototype Aufruf (Feld3)
Feld1=               "Karl"
Feld2=               Aufruf(Feld1)
MessageBox           (Mvar;"Function Prototype testen";Feld2)

Function             Aufruf(Feld3)
Feld3=               Feld3 + " Meier"
Return               (Feld3)
EndFunc

Call                 (Berechnen)
```

Beim Aufruf der Function-Routine wird der Inhalt von Feld1 (= Karl) übergeben und in Feld3 gespeichert. Innerhalb der Routine wird in Feld3 die Konstante » Meier« ergänzt, so daß danach in Feld3 der Inhalt »Karl Meier« vorhanden ist. Return gibt den generierten Inhalt von Feld3 an den aufrufenden Befehl (= Feld2) zurück. Nach der Ausführung von Return und EndProc enthält Feld2 darum den in der Function-Routine generierten Inhalt »Karl Meier«. Würde in diesem Fall Return weggelassen werden oder keinen Wert zurückgeben, würde in Feld2 0 gespeichert. Feld1 bleibt unverändert. Nach der Ausführung von EndProc wird der dem aufrufenden Befehl folgende Befehle (= MessageBox) ausgeführt und anschließend der Call-Befehl. Die Function-Routine wird übersprungen und nicht nochmals ausgeführt.

Zwischen Function und dem nachfolgenden Labelnamen darf zum Einrücken nicht F7 verwendet werden, sonst werden Compile-Fehler angezeigt, obwohl die Syntax stimmt. Verwenden Sie zum Einrücken ⇥ oder Leerstellen.

Weitere Hinweise siehe: Function, Procedure, Use

6.58 GetNumber

Befehlsform: **GetNumber***(Variable;Meldung;Boxentitel)*

Über diesen Befehl können Sie die Eingabe eines numerischen Wertes anfordern. Nach der Ausführung dieses Befehls erscheint ein Dialogfeld. Der Titel entspricht der Variablen Boxentitel. Unter Meldung können Sie einen Bedienungshinweis eingeben, der bei der Anzeige der Box oberhalb des Eingabefeldes erscheint. Sie können nur numerische Werte eingeben (ggf. in Verbindung mit Dezimalkomma bzw. Dezimalpunkt und mit mathematischen Vorzeichen + oder–), andernfalls erfolgt beim Drücken von ↵ oder beim Klicken auf [OK] eine Fehlermeldung. Wiederholen Sie dann die Eingabe, und geben Sie einen numerischen Wert ein. Wenn Sie nach der Anzeige eines Fehlerhinweises auf [Abbrechen] klicken, reagiert das Makro entsprechend und akzeptiert diese Wahl als Abbruch. Haben Sie zuvor OnCancel verwendet, verzweigt das Makro zu dem angegebenen Label und arbeitet dort weiter. Die Verwendung der mathematischen Vorzeichen + und – ist erlaubt. Sie müssen unmittelbar vor der Zahl angegeben werden, z. B. –10,50. Der eingegebene Wert wird in Variable gespeichert. Trennen Sie die Ganzzahl und den Dezimalbruch durch einen Dezimalpunkt bzw. ein Dezimalkomma.

Parameter

Variable Beliebiger Variablenname zur Speicherung der eingegebenen Zahl.

Meldung Beliebiger Text, z. B. Bedienungshinweis zur Eingabe einer Zahl.

Boxentitel Beliebiger Text, der in der Titelleiste des Dialogfelds erscheint.

Beispiel: *GetNumber*(Plz;"Bitte Postleitzahl eingeben.";"Adresse erfassen")
In diesem Beispiel wird eine fünfstellige Postleitzahl eingegeben.

Weitere Hinweise siehe: GetString, NumStr, StrNum

6.59 GetString

Befehlsform: **GetString***(Variable;Meldung;Boxentitel;Länge)*

Über diesen Befehl können Sie die Eingabe eines beliebigen Textes anfordern. Nach der Ausführung dieses Befehls erscheint ein Dialogfeld. Der Boxentitel entspricht der Variablen Boxentitel. Unter Meldung können Sie einen Bedienungshinweis eingeben, der bei der Anzeige der Box oberhalb des Eingabefeldes erscheint. Sie können einen beliebigen Text eingeben. Wenn Sie die Variable Länge verwenden, können Sie die Eingabe auf eine bestimmte Anzahl von Zeichen begrenzen. Die Eingabe über die Tastatur wird blockiert, wenn die maximale Anzahl der einzugebenden Zeichen erreicht wurde. Sofern Sie Länge nicht verwenden, können Sie maximal 64 KByte Text eingeben, ansonsten darf der unter Länge angegebene Wert nicht überschritten werden. Der eingegebene Text wird in Variable gespeichert.

Parameter

Variable Beliebiger Variablenname.

Meldung Beliebiger Text, z. B. Bedienungshinweis zur Eingabe eines Textes.

Boxentitel Beliebiger Text, der in der Titelleiste des angezeigten Dialogfeldes erscheint.

Länge Bei diesem Parameter können Sie einen numerischen Wert oder eine Variable mit numerischem Inhalt angeben, die die Eingabe einer bestimmten Textlänge erlaubt und auch kontrolliert. Haben Sie z. B. 12 eingegeben, wird der Eingabetext auf zwölf Zeichen begrenzt. Ein dreizehntes Zeichen kann nicht mehr eingegeben werden. Lassen Sie diesen Parameter weg, wenn Sie die Texteingabe nicht begrenzen möchten.

Beispiel: *GetString*(Nachname;"Bitte Nachname eingeben.";"Adresse erfassen";20)

Möchten Sie in dem Dialogfeld einen Wert in Hochkommata anzeigen, müssen Sie diesen Text durch doppelte Hochkommata einschließen:

GetString(Eingabe;"Bitte Text eingeben:" ;"Befehl ""CharLen" testen.")

Nach der Ausführung dieses Befehls erscheint in der Titelleiste des Dialogfelds:

Befehl »CharLen« testen.

Weitere Hinweise siehe: GetString, NumStr, StrNum

6.60 GetUnits

Befehlsform: **GetUnits**(*Variable;Meldung;Boxentitel*)

Über diesen Befehl können Sie die Eingabe von Maßeinheiten anfordern. Nach der Ausführung dieses Befehls erscheint ein Dialogfeld. Der Boxentitel entspricht der Variablen *Boxentitel*. Unter *Meldung* können Sie einen Bedienungshinweis eingeben, der bei der Anzeige des Dialogfeldes oberhalb des Eingabefeldes angezeigt wird. Den Maßeinheiten können Sie Abkürzungen wie c (= Zentimeter) oder i (= Inches) anhängen. Geben Sie keine Abkürzung der Maßeinheit ein, wird automatisch die standardmäßig vorgegebene Maßeinheit bzw. die unter *DefaultUnits* vorgegebene, verwendet. Die eingegebenen Daten werden formal überprüft. Wird ein Fehler erkannt, erfolgt eine Fehlermeldung. Geben Sie die Maßeinheiten in diesem Fall erneut in der richtigen Form ein. Die Eingabe wird in *Variable* gespeichert.

Parameter

Variable	Beliebiger Variablenname zur Speicherung des eingegebenen Wertes.
Meldung	Beliebiger Text, z. B. Bedienungshinweis zur Eingabe einer Maßeinheit.
Boxentitel	Beliebiger Text, der in der Titelleiste des angezeigten Dialogfeldes erscheint.

Beispiel:

```
GetUnits (LRand;"Bitte linken Rand eingeben«;"Randeinstellung«)
RandLinks(LRand)
```

Der im Dialogfeld eingegebene Wert (z. B. 1,5) wird in der Variablen *LRand* gespeichert. Mit dem nachfolgenden Befehl wird der neue linke Rand im Dokument auf Cursorposition gesetzt.

Weitere Hinweise siehe: DefaultUnits, GetString, GetNumber

6.61 Global

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

In einem Makro sind drei Variablen-Tabellen möglich, die lokale, globale und Perist-Variable speichern. Variable, die in Verbindung mit *Global* definiert wurden, stehen makroübergreifend zur Verfügung. Hierdurch können Variable von einem Makro an über *Chain* oder *Run* aufgerufene Makros übergeben werden. Im aufgerufenen Makro muß der Befehl *VarErrChk* verwendet werden, weil beim Kompilieren dieses Makros die Variablen noch nicht zur Verfügung stehen (es sei denn, das aufrufende Makro wurde zuvor bereits ausgeführt). Beachten Sie bitte auch die Befehle *Persist* und *PersistAll*, da über diese beiden Befehle auch Variable für Mischvorgänge übernommen werden können.

Parameter

Variable Beliebige Variablenamen, die in einem aufgerufenen Makro benötigt werden.

Beispiel

```
Application          (A1; "WordPerfect"; Default; "DE")
Global               (Feld1;Feld2;Feld3)
Feld1=               "Hier ist Feld 1  "
Feld2=               "Hier ist Feld 2  "
Feld3=               "Hier ist Feld 3  "
Run                  ("Global1.wcm")
Quit

Global1.WCM

Application          (A1; "WordPerfect"; Default; "DE")
VarErrChk            (Off!)
Prompt               ("Befehl ""GLOBAL"" testen";Feld1+Feld2+Feld3;;;)
Wait                 (30)
EndPrompt
Return
```

In dem ersten Makro werden die Variablen *Feld1*, *Feld2* und *Feld3* als globale Variable definiert. Dadurch können sie in dem aufgerufenen Makro GLOBAL1.WCM weiterverarbeitet werden. In dem aufgerufenen Makro ist der Befehl *VarErrChk* erforderlich, da beim Kompilieren dieses Makros die im Prompt-Befehl angesprochenen Variablen noch nicht zur Verfügung stehen (ansonsten erfolgt eine Fehlermeldung). Im zweiten Makro werden die Daten der aus dem ersten Makro übergebenen Variablen über *Prompt* angezeigt.

*Weitere Hinweise siehe:*Chain, Discard, IfExists, Local, Persist, PersistAll, Run, VarErrChk

6.62 Go

Befehlsform: **Go**(*Label*)

Verzweigung zu einer Adreßmarke (Label) innerhalb des aktuellen Makros. Die Verzweigung kann z. B. direkt oder in Verbindung mit einer *In..*-Struktur aufgrund einer Bedingung erfolgen. Die sequentielle Abarbeitung einer Befehlsfolge wird durch *Go* unterbrochen und bei dem angegebenen Label fortgeführt. Jeder Labelname kann mit einem @ enden. Wurden Makros der Versionen WPWin 5.1/5.2 konvertiert, wird dieses Zeichen nicht entfernt.

Der unter *Go* angegebene Label muß in dem aktuellen Makro definiert sein. Nach Ausführung des *Go*-Befehls verzweigt das Makro zu dem angegebenen Label. Die daran unmittelbar anschließende Befehlsfolge wird solange ausgeführt, bis wieder ein *Go*-Befehl auftritt oder das Makro beendet wird. Einem *Go*-Befehl dürfen keine weiteren Befehle folgen, da diese nicht ausgeführt werden (siehe Beispiel 2). Der einzige gültige Befehl, der dann noch folgen

darf, muß immer ein *Label*-Befehl sein, es sei denn, der *Go*-Befehl ist der letzte Befehl des Makros oder er befindet sich innerhalb einer *If-*, *For-*, *ForEach* oder *Repeat*-Struktur. Beenden Sie einen Befehlsblock, zu dem Sie mit *Go* verzweigt sind, nie mit *Return*, da hierdurch das Makro beendet wird (es sei denn, dies wäre beabsichtigt), um zu einem aufrufenden Makro zurückzukehren.

In Verbindung mit *Go*-Befehlen kann ein sog. Spaghetti-Programm entstehen, wenn Sie zuviele *Go*-Befehle verwenden. Damit ist gemeint, daß innerhalb des Makros zuviel gesprungen wird, mal nach vorne, mal nach hinten, wodurch das Makro unübersichtlich wird. Versuchen Sie besonders in Verbindung mit Schleifenverarbeitung die Befehle *For*, *ForEach*, *ForNext*, *Repeat* oder *While* zu verwenden. Beachten Sie hierzu bitte auch die Regeln der strukturierten Programmierung.

In einer Mischdatei kann der *Go*-Befehle auch dazu verwendet werden, einen bestimmten Teil der Mischdatei zu überspringen.

Parameter

Label Jeder Labelname muß mit einem Buchstaben beginnen und kann mit einem @ enden. Die maximale Länge beträgt 30 Zeichen. Der Name darf nicht in Hochkommata eingeschlossen sein. Der Labelname, der hier angegeben wird, muß in einem *Label*-Befehl innerhalb desselben Makros definiert sein. Ist dies nicht der Fall, erfolgt beim Kompilieren eine Fehlermeldung. Beachten Sie bitte auch die Hinweise bei dem Befehl *Label*.

Beispiel

```
1.      Label               (Anfang)   //Label Anfang.
        Call                (Drucken)  //Aufruf der Unterroutine Drucken.
        Prompt              ("Hinweis";"Text wurde gedruckt";;;)
        Go                  (Laden)    //Verzweigung zu dem Label Laden.

        Label               (Drucken)  //Beginn der Unterroutine Drucken.
        DruckenSeite        ()
        Return                         //Ende der Unterroutine Drucken.

        Label               (Laden)    //Beginn des Labels Laden.
        DateiLadenDlg       ()         //Laden einer Datei.
        Go                  (Anfang)   //Verzweigung zu dem Label Anfang.
```

In diesem Beispiel werden die Befehle *Go* und *Call* verwendet. Der Unterschied besteht darin, daß bei einem *Call* eine Unterroutine innerhalb desselben Makros aufgerufen wird, nach deren Abarbeitung mit dem Befehl weitergearbeitet wird, der dem *Call*-Befehl folgt (hier: *Prompt*). Bei einem *Go* wird zu dem angegebenen Label verzweigt, d. h. die Unterroutine *(Drucken)* wird nicht ausgeführt. Nach Beendigung der Unterroutine *(Laden)* wird wieder an den Anfang des Makros verzweigt.

```
2.      If                  (Zähler=5) //Beginn der IF-Struktur, Prüfung von Werten.
          Go                (Suchen)   //Verzweigen nach Suchen.
          Call              (Drucken)  //wird nicht ausgeführt.
```

```
  Call              (Ersetzen) //wird nicht ausgeführt.
Else
  Call              (Sichern)  //Aufruf von "Sichern",wenn Variable nicht = 5.
EndIf                          //Ende der IF-Struktur.
```

Die beiden Befehle *Call*(Drucken) und *Call*(Ersetzen) kommen nie zur Ausführung, da aufgrund des vorausgehenden *Go*-Befehls das Makro zu dem Label *(Suchen)* verzweigt.

Nachfolgend zwei Beispiele, wie Sie nicht programmieren sollten:

3.
```
Label               (Anfang)
Call                (Drucken)
Go                  (Sichern)
Go                  (Anfang)   //Befehl wird nie ausgeführt.

Label               (Drucken)
:::::
Return

Label               (Sichern)
::::::
Return                         //Makro wird hier beendet.
```

Durch den letzten *Return*-Befehl erfolgt kein Rücksprung zu dem ersten *Go*, um z. B. durch den zweiten *Go* (was übrigens natürlich nicht möglich ist) wieder an den Schleifenanfang zu dem Label *(Anfang)* zu verzweigen. Das Makro wird mit diesem *Return* beendet, auch dann, wenn hinterher noch Befehle folgen.

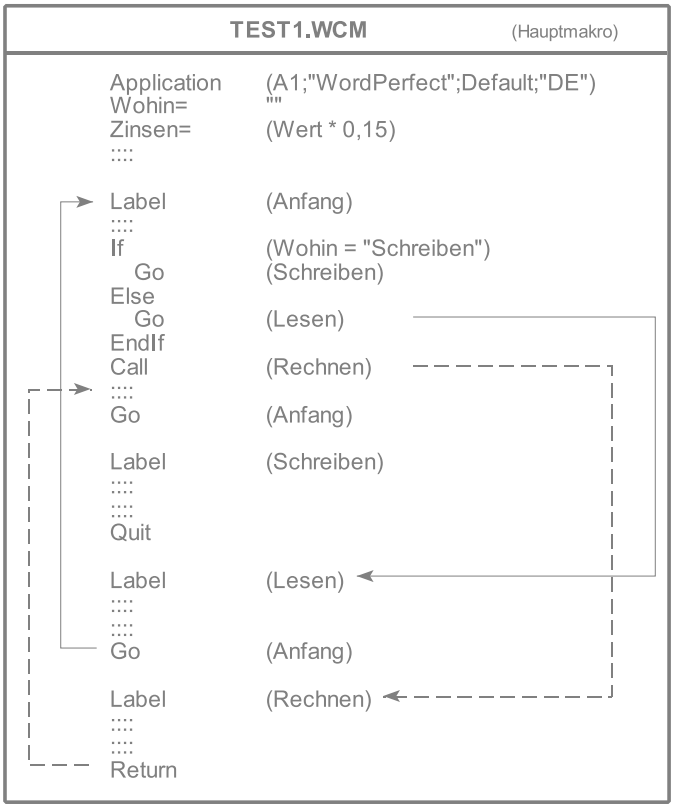
4.
```
Label               (Anfang)
Call                (Drucken)
Call                (Sichern)
Go                  (Anfang)

Label               (Drucken)
:::::
Go                  (Sichern)
Return                         //Dieser Befehl wird nie ausgeführt.

Label               (Sichern)
::::::
Return
```

Hier wird die Unterroutine (*Drucken*) aufgerufen und von dort nach (*Sichern*) verzweigt. Der hinter dem *Go* definierte *Return* wird nie ausgeführt. In der Unterroutine (*Sichern*) erfolgt durch *Return* kein Rücksprung zum aufrufenden *Call*.

Dies ist keine saubere Programmierung und führt darum in der Regel zu Fehlern!

Weitere Hinweise siehe: If, Label, On...

Verzweigungen mit GO



Regeln:

GO Label muß im selben Makro vorhanden sein.

Jedem GO-Befehl muß ein**neuer** Label folgen (Ausnahme siehe IF-Befehl).

Jeder angesprungene Label muß enden mit:

Go: Sprung zum nächsten Label.

Return: Makro wird beendet (oder Rücksprung zu einem evtl. aufrufenden Makro).

Quit: Makro wird beendet.

6.63 GoOnLine

Befehlsform: **GoOnline**(*Service;Ziel*)

Aufbau einer Verbindung zu einem Online-Service.

Parameter

Service	Anbieter eines Online-Services.	
	Internet!	Verbindung ins Internet aufbauen (Standardvorgabe).
	CompuServe!	Verbindung zu CompuServe aufbauen
Other!	Verbindung zu einem anderen Service aufbauen.	
Ziel	Ziel angeben, zu dem die Verbindung aufgenommen werden soll.	

Beispiel

```
GoOnLine  (Internet;"http//home.netscaspe.com")
GoOnLine  (Internet;"http//home.netscaspe.com")
```

Starten des Internet-Browsers auf der NetScape-Home-Page.

6.64 If

Befehlsform: **If**(*Test*)

Vergleich zweier oder mehrerer Bedingungen/Werte. In Abhängigkeit des Ergebnisses (»wahr« oder »nicht wahr«) können Sie einen oder mehrere Befehle ausführen, Unterroutinen aufrufen oder Verzweigungen vornehmen. Werden die Bedingungen erfüllt, ist das Ergebnis »wahr«. In diesem Fall werden die unmittelbar auf *If* folgenden Befehle bis zum nächsten *Else* oder, wenn kein *Else* vorhanden ist, bis zum nächsten zugehörigen *EndIf* ausgeführt. Bei geschachtelten *If*-Strukturen erfolgt die Ausführung bis zum Beginn des nächsten *If*-Befehls innerhalb derselben Struktur bzw. beim letzten *If* bis zum ersten *Else*. Eine Bedingung gilt als nicht erfüllt, wenn das Ergebnis einer Prüfung »nicht wahr« ist. In diesem Fall werden die dem *Else* folgenden Befehle bis zum nächsten zugehörigen *EndIf* ausgeführt. Ist kein *Else* vorhanden, werden die zwischen *If* und *EndIf* definierten Befehle nicht ausgeführt. Das Makro arbeitet in diesem Fall hinter dem *EndIf* weiter. Ausnahmen bestehen bei geschachtelten *If*-Bedingungen.

Parameter

Test

Unter *Test* ist im allgemeinen ein Vergleich in Verbindung mit einer Variablen, einer numerischen Konstante (= fixer Wert, z. B. 1, 1000, 2000,50, 7000) oder einer alphanumerische Konstante (= ein oder mehrere Zeichen beliebiger Art wie Ziffern, Buchstaben oder Sonderzeichen) zu verstehen. Alphanumerische Werte müssen durch Hochkommata eingeschlossen sein. Jede *If*-Struktur muß mit einem *EndIf* enden. Bei geschachtelten *If*-Strukturen müssen so viele *EndIf* angegeben werden, wie *If* definiert wurden.

Beispiel

```
1       If                  (ZählerEin=ZählerAus)
          Go                (Abrechnung)
        EndIf
```

Das Makro verzweigt nach (*Abrechnung*), wenn die angegebenen Variablen denselben Inhalt haben. Ist das nicht der Fall, wird der *Go*-Befehl nicht ausgeführt.

```
2       If                  (ZählerEin=ZählerAus)
          Go                (Zwischensumme)
        Else
          Call              (Endesumme)
        EndIf
```

Das Makro verzweigt nach (*Zwischensumme*), wenn die angegebenen Variablen denselben Inhalt haben, andernfalls (*Else*) wird die Unterroutine (*Endesumme*) aufgerufen.

```
3       If                  (ZählerEin=ZählerAus)
        Else
          Call              (Endesumme)
        EndIf
```

Die Unterroutine (*Endesumme*) wird aufgerufen, wenn die Bedingung nicht zutrifft. Trifft die Bedingung zu, wird hinter *EndIf* weitergearbeitet.

```
4       If                  ((ZählerEin=1) or (Kennung="n"))
          Go                (Zwischensumme)
        EndIf
```

Hier werden zwei Bedingungen verknüpft. Achten Sie auf die doppelten Klammern.

Das Makro verzweigt nach (Zwischensumme), wenn Variable *ZählerEin* den Wert 1 enthält oder wenn die Variable *Kennung* den Kleinbuchstaben n enthält. D. h. eine der beiden Bedingungen muß zutreffen. Der *Go*-Befehl wird auch dann ausgeführt, wenn beide Bedingungen zutreffen.

```
5       If                  ((ZählerEin=1) and (Kennung="n"))
          Go                (Zwischensumme)
        EndIf
```

oder

```
        If                  (ZählerEin=1 and Kennung="n«)
```

Hier werden zwei Bedingungen verknüpft. Das Makro verzweigt nach (Zwischensumme), wenn Variable *ZählerEin* den Wert 1 enthält und wenn die Variable *Kennung* den Kleinbuchstaben n enthält. D. h. beide Bedingungen müssen zutreffen. Die nachfolgende Abfrage (= geschachtelte *If*-Bedingung) erzielt dieselbe Wirkung:

```
If                  (ZählerEin=1)
  If                (Kennung="n")
  Go                (Zwischensumme).
  EndIf
EndIf
```

6

```
If                  (Eingabe="z")Assign(Kennz;"Neuzugang")Else
   If               (Eingabe="a")Assign(Kennz;"Löschung")Else
      If            (Eingabe="v")Assign(Kennz;"Änderung")
                    Else
                    Prompt("Fehler";"Falsches Änderungsmerkmal";1;;)Pause
                    EndPrompt
                    Go(Anfang)
      EndIf
   EndIf
EndIf
```

Hierbei handelt es sich um eine geschachtelte *If*-Struktur. Achten Sie auf die Anzahl von *Else* und *EndIf*. Trifft eine der Bedingungen zu, wird der jeweils zugehörige *Assign*-Befehl ausgeführt. Trifft keine der drei Bedingungen zu, wird über *Prompt* ein Fehlerhinweis am Bildschirm angezeigt.

7

```
Antwort=            True

If                  (Antwort)
   Call             (Bearbeiten)
EndIf
```

Über die Bedingungs-Variablen *False* und *True* kann einer Variablen ein Zustand zugeordnet werden, der in Verbindung mit *If* geprüft werden kann. Die beiden Begriffe dürfen nicht in Hochkommata angegeben werden!

In diesem Fall trifft die Bedingung zu, wenn *Antwort True* enthält. Enthält *Antwort False*, trifft die Bedingung nicht zu.

```
If                  (Not Antwort)
   Call             (Bearbeiten)
EndIf
```

Hier wurde die Abfrage umgedreht. Die Bedingung trifft dann zu, wenn *Antwort False* enthält. Enthält *Antwort True*, trifft die Bedingung zu.

8

```
If                  (?TabelleInTabelle = 0)
  InTabelle=        False
Else
  InTabelle=        True
EndIf
```

```
        ::::
        If                  (InTabelle) :::
        If                  (Not InTabelle) :::
```

Mit dem ersten *If*-Befehl wird geprüft, ob sich der Cursor in einer Tabelle befindet (0 = Nein, 1 = Ja). Je nach Position wird die Variable *InTabelle* (Name ist frei wählbar) auf *False* (= Nein) oder *True* (= Ja) gesetzt. Mit den beiden letzten Abfragen kann im weiteren Makroverlauf geprüft werden, wo sich der Cursor befand (siehe auch Punkt 7).

```
9       If                  (?DokLeer)
        Call                (FehlerLeeresDokument)
        EndIf
```

Systemvariable (hier: *?DokLeer*) können geprüft werden, um im Makro bestimmte Funktionen zu steuern. In diesem Beispiel wird geprüft, ob sich der Cursor in einem leeren Dokumentfenster befindet. Wenn ja, wird eine Unterroutine aufgerufen. Auch hier kann wie unter Punkt 7 die Abfrage in Verbindung mit *Not* umgekehrt werden:

```
        If                  (Not ?DokLeer)

10      If                  (Monate[7] = "Juli")
        If                  (Umsatz[Werk;Abteilung;Gruppe] = GeplanterUmsatz+10000)

        If                  (Umsatz94[Werk;Abteilung;Gruppe] >

Umsatz93[Werk;Abteilung;Gruppe])

In diesen Beispielen werden jeweils die Felder von Arrays überprüft. Achten Sie darauf, daß
die Indizes in eckigen Klammern angegeben werden.
```

Weitere Hinweise siehe: EndIf, Else

6.65 IfPlatform

Befehlsform: **IfPlatform**(*PlattFormKennung;PlattFormKennung...*)

Über diesen Befehl haben Sie die Möglichkeit Makros zu schreiben, die in unterschiedlichen Systemen lauffähig sind (die Namen der Systeme sind fest vorgegeben, siehe unten). Bestimmte Makros brauchen dadurch nicht in jedem System separat erstellt zu werden, was zwangsläufig zu Redundanzen führt, bei Änderungen müßten mehrere Makros bearbeitet, getestet und dokumentiert werden. Mit *IfPlatForm* wird das System/die Systeme festgelegt, in welchem die Befehle, die zwischen *IfPlatForm* und *EndIfPlatform* definiert sind, ausgeführt werden sollen. In allen anderen Systemen, die nicht angegeben wurden, wird die Befehlsfolge nicht ausgeführt. Befehle werden auch dann nicht ausgeführt, wenn sie von dem betreffenden System nicht interpretiert werden können.

Parameter

PlattFormKennung Angabe des Systems, in dem die zwischen *IfPlatForm* und *EndIfPlatForm* definierten Befehle ausgeführt werden sollen. Es erfolgt keine Unterscheidung zwischen Groß- und Kleinbuchstaben. Folgende Parameter stehen zur Verfügung:

```
DG!       DOS!      MAC!      NEXT!     NONE!     OS2!
UNIX!     VAX!      WIN!      WIN32!    WIN95!    WINNT!
```

Beispiel

```
Application          (A1;"WordPerfect";Default;"DE")
:::
IfPlatform           ("DOS")    Läuft nur unter DOS.
  Nest               ("C:\WP61\MAKROS\RECHNUNG.WPM")
  ::::
EndIfPlatForm

IfPlatform           ("WIN")    Läuft nur unter Windows.
  RUN                ("C:\WPWIN7\MAKROS\RECHNUNG.WCM")
EndIfPlatForm
```

In diesem Beispiel wird unter DOS über den Befehl *Nest* und in Windows über den Befehl *Run* jeweils ein Makro RECHNUNG aufgerufen. Beachten Sie bitte, daß die Makronamen der aufgerufenen Makros unterschiedliche Dateinamen-Erweiterungen haben. Die Makros laden zur Erstellung einer Rechnung dieselbe Formulardatei (WP-DOS-6.0- und WP-WIN6.0/6.0a/6.1-Dateien sind kompatibel).

Weitere Hinweise siehe: EndIfPlatform

6.66 Include

Befehlsform: **Include**(*Makrodatei*)

Über diese Funktion können Makrodateien mit ausführbaren Befehlen, *Function*- und/oder *Procedure*-Routinen des unter ***Makrodatei*** angegebenen Makros aufgerufen werden. Da der Befehl nicht ausgeführt wird, sondern nur eine Verbindung zu dem angegebenen Makro herstellt, kann er in dem aufrufenden Makro an beliebiger Stelle definiert sein, wegen der Übersichtlichkeit aber am besten am Anfang eines Makros. Die Befehle der angegebenen Makrodatei werden automatisch kompiliert, wenn das Makro, das den *Include*-Befehle enthält, kompiliert wird. Die Befehle der angegebenen Makrodatei werden ausgeführt, bevor die Makroausführung in dem aufrufenden Makro fortgesetzt wird.

Parameter

Makrodatei Vollständiger Name der Makrodatei in Form einer Konstanten (Variable sind nicht erlaubt).

Beispiel

```
Aufrufendes Makro
Application          (A1;"WordPerfect";Default;"DE")
Include              ("C:\WPWIN7\MAKROS\INCLUDE.WCM")
Feld1=               "Karl"
Feld2=               Aufruf(Feld1)
MessageBox           (Mvar;"Function Prototype testen";Feld2)
```

Über INCLUDE aufgerufene Makrodatei (»C:\WPWIN7\MAKROS\INCLUDE.WCM«)

```
Application          (A1;"WordPerfect";Default;"DE")
Function             Aufruf(Feld3)
Feld3=               Feld3 + " Meier"
Return               (Feld3)
EndFunc
```

Die Beschreibung dieses Beispiels finden Sie unter Abschnitt »6.57 Function Prototype«. Der Unterschied besteht darin, daß bei *Include* die Function-Routine ausgegliedert und als selbständige Makrodatei behandelt wird.

Zwischen *Function* und dem nachfolgenden Labelnamen darf zum Einrücken nicht F7 verwendet werden, sonst werden Compile-Fehler angezeigt, obwohl die Syntax stimmt. Verwenden Sie zum Einrücken ⇥ oder Leerstellen.

Weitere Hinweise siehe: Chain, Function, Procedure, Use

6.67 Indirect

Befehlsform: *Variable1*=**Indirect**(*Variable2*)

Über diesen Befehl können Sie Variablen- und Labelnamen aus einer Kombination von Variableninhalten, Zeichen und/oder Ziffern generieren, um die Makroverarbeitung noch flexibler zu gestalten. Der Befehl kann überall dort benutzt werden, wo Variable und Label verwendet werden. Die derzeit aktuelle Variable (= *Variable2*) muß vor der Verwendung von *Indirect* definiert und mit einem Wert versehen worden sein. Die neu generierte Variable wird in *Variable1* gespeichert.

Parameter

Variable1 Variable zur Aufnahme des unter *Variable2* generierten Wertes. Dies kann der Name einer Variablen oder eines Labels sein.

Variable2 Variable, Zeichen oder Ziffern, die in Verbindung mit + zu einem Variablen- oder Labelnamen zusammengesetzt werden können. Labelnamen können hier nur generiert werden, um sie z. B. in einem *Call*-Befehl zu verwenden, nicht aber um die Definition eines Labels vorzunehmen

Beispiel

```
1.      Feld1=              "Hund"
        Feld2=              "Katze"
        Feld3=              "Maus"
        :::::
        Feld10=             "Elefant"
```

Um diese zehn Variablen zu schreiben müßte normalerweise zehnmal *Type* verwendet werden. In Verbindung mit *ForNext* genügt ein Schreibbefehl:

```
        ForNext             (Feldnummer;1;10)
          Type              ("Tier"+Feldnummer+"=  "+Indirect("Feld"+Feldnummer))
          HardReturn        ()
        EndFor
```

Aufgrund des *Type*-Befehls wird eine Textkonstante erzeugt (Tier?=) und die zu schreibenden Variablennamen generiert (Feld+Feldnummer). *Feld* entspricht hierbei dem Teil des Variablennamens, der immer identisch ist, *Feldnummer* (wird durch *ForNext* hochgezählt) enthält der Reihe nach die Zahlen 1 – 10. Der *Type*-Befehl generiert darum folgende Zeilen:

```
        Tier1=              Hund
        Tier2=              Katze
        Tier3=              Maus
        :::::
        Tier10=             Elefant
2.      ForEach             (Aufruf;{"Lesen";"Prüfen";"Schreiben"})
          Call              (Indirect(Aufruf))
        EndFor
```

In diesem Beispiel werden der Reihe nach die unter *ForEach* angegebenen Unterroutinen über *Call* aufgerufen. Die Labelnamen der Unterroutinen müssen vorhanden sein.

```
3.      ForNext             (Nummer;1;5)
         Call               (Indirect("Upro"+Nummer))
        EndFor
```

Aufgrund von *ForNext* werden in Verbindung mit *Call* die Labelnamen *Upro1*, *Upro2*, *Upro3*, *Upro4* und *Upro5* generiert und der Reihe nach aufgerufen. Die Labelnamen der Unterroutinen müssen vorhanden sein.

```
4.      Switch              (Exists(Indirect("Feld"+Nummer)))
```

Überprüfen, ob die Variable *Feld?* vorhanden ist. Anstelle des Fragezeichens wird der Wert von *Nummer* ergänzt, z. B. *Feld5*.

```
5.        ForNext             (Zähler;1;5)
            Assign            (Indirect("Feld"+Zähler);"abc")
            Type              (Indirect("Feld"+Zähler))
            HardReturn        ()
          EndFor
```

Definition von fünf Variablen mit den Namen *Feld1 – Feld5*. Jeder Variablen wird der Inhalt *abc* zugeordnet. Zur Kontrolle werden die Variableninhalt gedruckt.

Weitere Hinweise siehe: Assign, Label

6.68 Integer

Befehlsform: *Variable*=**Integer**(*NumAusdruck*)
Integer(*Variable;NumAusdruck*) (WPWin 5.2)

Die Ganzzahl von *NumAusdruck* (z. B. ein ermitteltes Rechenergebnis) wird einer Variablen zugeordnet. Wurde z. B. bei einer Rechenoperation das Ergebnis 2,50 ermittelt, wird der Wert vor dem Komma (hier: 2) in der angegebenen Variablen gespeichert, der Wert hinter dem Komma wird nicht berücksichtigt. Beachten Sie bitte auch den Befehl *Fraction*, mit dem Sie den Wert hinter dem Komma ermitteln können. Die Variable wird auf Null gesetzt, wenn keine Ganzzahl ermittelt wurde. Enthält *NumAusdruck* einen negativen Wert, wird die Ganzzahl ggf. gerundet: Aus -123,45 wird -124. Beachten Sie bitte auch Kapitel »5.17 Änderungen gegenüber WPWin 5.1/5.2«.

Parameter

Variable Beliebiger Variablenname. Enthält nach der Ausführung des Befehls die Ganzzahl.

NumAusdruck Numerischer Wert oder Ausdruck, aus dem die Ganzzahl zu ermitteln ist.

Beispiel

Achtung: Die Befehle *Fraction* und *Integer* in dem folgenden Beispiel dienen nur zu Demonstrationszwecken. Die Mehrwertsteuer läßt sich natürlich mit dem folgenden Befehl auch einfacher berechnen:

SteuerDMPF= Betrag*0.15 oder SteuerDMPF= Betrag*=,15

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

Label               (Anfang)
GetNumber           (Betrag;"Betrag eingeben.";"MwSt-Berechnung")
SteuerDM=           Integer(Betrag*0.15)
```

```
SteuerPF=           Fraction(Betrag*0.14)
SteuerDMPF=         (SteuerDM+SteuerPF)
Prompt              ("Ermittelte Mehrwertsteuer:";SteuerDMPF;;;)
Pause
EndPrompt
Go                  (Anfang)
```

Die Beschreibung zu diesem Beispiel finden Sie unter dem Befehl *Fraction*. Dieses Beispiel finden Sie auf der CD-ROM unter dem Dateinamen FRACTION.WCM. Beachten Sie bitte auch die mathematischen Operatoren in Kapitel »5.10 Mathematische Funktionen«.

Weitere Hinweise siehe: Fraction

6.69 Kommentar

Befehlsform: //Kommentar**[FNZ]**

Der Text zwischen // und dem nächsten festen HardReturn [FNZ] wird als Kommentar interpretiert. Kommentare können mehrere Zeilen lang sein. Nutzen Sie [Einrücken] oder [⇥], um erforderliche Einrückungen vorzunehmen. Die Formatierung des Makros wird dadurch übersichtlicher. Befehle, die innerhalb des zuvor beschriebenen Bereichs definiert wurden, gelten ebenfalls als Kommentar, d. h. sie werden nicht ausgeführt. Nutzen Sie Kommentare zur

- Dokumentation.
- Beschreibung komplizierter Abläufe. Dies ist besonders dann wichtig, wenn das Makro zu einem späteren Zeitpunkt durch eine andere Person als durch den Ersteller geändert werden muß. Die Einarbeitung in die Logik großer Makros wird dadurch wesentlich erleichtert.
- vorübergehenden »Ausblendung« von Makrobefehlen oder Funktionen, um z. B. beim Testen eines Makros bestimmte Makroteile vom Test auszuschließen (schnellerer Testdurchlauf). Sie brauchen dann zum Testen die vorübergehend nicht benötigten Befehle nicht zu löschen und anschließend wieder neu zu erfassen.

Geben Sie die Kommentare am besten rechts der Befehle ein, die einer Erläuterung bedürfen. Bei einwandfrei verständlichen Befehlen können Sie Kommentare auch weglassen.

```
Application (A1;"WordPerfect";Default;"DE")
GetNumber(Betrag;"Betrag eingeben";"MWSt-Berechnung")
SteuerDM=Integer(Betrag*0.15)
//Mehrwertsteuer (Ganzzahl) errechnen).
SteuerPF=Fraction(Betrag*0.15)
//Mehrwertsteuer (Dezimalbruch) errechnen.
Assign(SteuerDMPF;SteuerDM+SteuerPF)
//Ganzzahl und Dezimalbruch addieren.
SteuerDMPF=NumStr(SteuerDMPF;2)
```

```
//Numerischen Wert in String konvertieren.
Prompt("Ermittelte Mehrwertsteuer:";SteuerDMPF;;;)
//Anzeige des Ergebnisses in einer Box.
Pause
//Anzeigen lassen, bis auf [OK] geklickt wird.
Endprompt
//Box wieder ausblenden.
```

Unübersichtliche Darstellung

```
Application (A1;"WordPerfect";Default;"DE")
 //*********************************************************************
 //*            Makro zur Berechnung der Mehrwertsteuer                *
 //*********************************************************************
 GetNumber(Betrag;"Betrag eingeben";"MWSt-Berechnung")//Betrag eingeben.
SteuerDM=Integer(Betrag*0.15)//Mehrwertsteuer (Ganzzahl) errechnen).

SteuerPF=Fraction(Betrag*0.15)//Mehrwertsteuer (Dezimalbruch) errechnen.
Assign (SteuerDMPF;SteuerDM+SteuerPF)//Ganzzahl und Dezimalbruch addieren.
SteuerDMPF=NumStr(SteuerDMPF;2)//Numerischen Wert in String
                               //konvertieren.
                               //Anzeige des Ergebnisses in einer Box.
 Prompt("Ermittelte Mehrwertsteuer:";SteuerDMPF;;;)
 Pause                         //Anzeigen, bis auf [OK] geklickt wird.
Endprompt                      //Box wieder ausblenden.
```

Übersichtliche Darstellung

6.70 Label

Befehlsform: **Label**(*Name*) oder **Label**(*Name*@)

Jeder Label stellt den Beginn einer Befehlsfolge dar, die durch Befehle wie *Call*, *Case*, *Case Call* aufgerufen (Unterroutine) bzw. zu der über *Go* verzweigt wird. Auch über die Befehle ***On...*** wird zu einem Label verzweigt. Ein Label entspricht einer »Hausnummer« (= Programmadresse) innerhalb des aktuellen Makros, d. h. ein Teil des Makros wird mit dieser »Hausnummer« gekennzeichnet. Durch einen *Go*-Befehl kann das Makro darum direkt zu dieser Adresse verzweigen, ohne die dazwischen liegenden Befehle auszuführen. Der Aufruf bzw. die Verzweigung ist von jeder beliebigen Stelle des aktuellen Makros aus möglich, d. h. es kann auch vom Ende des Makros an den Anfang gesprungen werden. Der Parameter *Name* kann mit einem @ enden (siehe unten). Wurden Makros der Versionen WPWin 5.1/5.2 konvertiert, wird dieses Zeichen nicht entfernt.

Die Länge des Namens darf 30 Zeichen nicht übersteigen. Verwenden Sie keine zu langen Namen, denn diese Namen müssen Sie ggf. mehrmals verwenden (wenn Routinen öfters aufgerufen werden), wobei das Eintippen dann mühselig sein kann (Fehlerquelle!).

Verwenden Sie mehrere Labels innerhalb eines Makros, müssen diese unterschied- liche Namen aufweisen. Kein Name darf doppelt vergeben werden, ansonsten erfolgt bei der Kompilierung eine Fehlermeldung. Sie können nur Labels des aktuellen Makros aufrufen, also keine Labels in Makros, die mit *Run* oder *Chain* aufgerufen wurden. In unterschiedlichen Makros können Sie gleiche Labelnamen verwenden, wobei die Anzahl der Label-Befehle nicht begrenzt ist.

Der unter *Label* angegebene Name muß von mindestens einer Stelle im Makro »angesprungen« werden. Ist dies nicht der Fall, erfolgt beim Kompilieren eine Fehlermeldung. Überprüfen Sie in diesem Fall den Labelnamen, vielleicht haben Sie sich bei der Eingabe nur vertippt. Prüfen Sie die Makrostellen, an denen dieser Label aufgerufen werden soll, und vergleichen Sie die Labelnamen. WordPerfect macht keine Unterscheidung zwischen Groß- und Kleinbuchstaben.

Parameter

Name

Die Labelnamen sind frei wählbar. Sie müssen mit einem Buchstaben beginnen und können mit einem @ enden. Beachten Sie bitte die zuvor beschriebenen Einschränkungen. Verwenden Sie nur Namen, die auf den Inhalt der nachfolgenden Befehle schließen lassen. Wird mit diesen Befehlen z. B. eine Zinsberechnung durchgeführt, könnte man den Label z. B. (*Zinsen*) nennen.

Bei größeren Makros sollten Sie Labelnamen in Verbindung mit einer fortlaufenden Nummer verwenden, damit Sie nicht unnötige Zeit mit dem Suchen der Befehlsroutinen vergeuden müssen. Diese Routinen sind dann in aufsteigender Reihenfolge nach diesen Nummern zu ordnen.

Beispiel

```
Label(A001Laden)
:::
Return

Label(A010Speichern)
:::
Return

Label(A020Berechnen)
:::
Return

Label(A030Drucken)
:::
Return
```

Nehmen Sie die Numerierung in Zehnersprüngen vor, damit Sie nachträglich vergessene Routinen in der Nummernfolge einfügen können. Weitere Informationen entnehmen Sie bitte dem Kapitel »5.12 Makro-Design«.

Beispiel

Folgende Label stellen gültige Label-Namen dar:

```
Label(Anfang)
Label(A01OSpeichern)
Label(DateiSpeichern)
Label(DateiÖffnen@)            (@ = Zwingend erforderlich in WPWin 5.2)
```

Folgende Label stellen ungültige Label-Namen dar:

```
Label(A01O  Speichern)         Leerstelle nicht erlaubt
Label(100_Lesen)               Label beginnt mit einer Ziffer
Label(A100-Lesen)              Minuszeichen/Bindestrich nicht erlaubt
Label(A100_Lesen)              Unterstreichungsstrich nicht erlaubt
```

Weitere Hinweise siehe: Go, Call, Case Call, ON...

6.71 Local

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

In einem Makro sind drei Variablen-Tabellen möglich, die lokale, globale und Per-sist-Variable verwalten können. Variable, die in Verbindung mit *Local* definiert wurden, können nur innerhalb des aktuellen Makros angesprochen werden. In Makros, die mit *Chain* oder *Run* aufgerufen wurden, stehen dies e Variablen nicht zur Verfügung. Nach Beendigung des aktuellen Makros stehen die hier über *Locaμμμl* definierten Variable nicht mehr zur Verfügung.

Parameter

Variable Beliebige Variable, die als lokale Variable definiert werden sollen. Wird das Makro beendet, in dem die Variablen definiert wurde, werden die Variablen gelöscht.

Weitere Hinweise siehe: Discard, Global, Persist, PersistAll

6.72 MacroArgs

Befehlsform: *Wert*=**MacroArgs[ ]**

Hier handelt es sich um eine Array-Variable, die Parameter-Werte enthält , die durch *Chain*, *Nest* oder *Run* an das aufgerufene Makro übergeben wurden.

Beispiel

Aufrufendes Makro MACARGS1:

```
Application          (A1;"WordPerfect";Default;"DE")
Run                  ("C:\wpwin7\makros\macargs2";{"A";"B";"C";"Hugo"})
Quit
```

Aufgerufenes Makro MACARGS2:

```
Application          (A1;"WordPerfect";Default;"DE")
Anzahl=              Dimensions(MacroArgs[];0)
If                   (Anzahl != 0)
   ForNext           (Wert;1;Anzahl;1)
   MessageBox        (x;"Element-Werte: ";"MacroArgs[" + Wert + "] = " + MacroArgs[Wert])
   EndFor
Else
   MessageBox(x;"Fehler";"Es wurden keine Werte übergeben")
EndIf
```

In der FOR-Schleife werden über MessageBox der Reihe nach die übergebenen Werte »A«, »B«, »C« und »Hugo« angezeigt. Die beiden Makros finden Sie auf der CD-ROM unter den Dateinamen MACARGS1.WCM und MACARGS2.WCM.

6.73 MacroDialogResult

Befehlsform: *Variable*=**MacroDialogResult**

In Dialogfeldern können über den Befehl *DialogAddPushButton* benutzerspezifische Tasten definiert werden (gilt auch für die Optionsfelder des Dialogeditors). In Abhängigkeit der gedrückten Taste muß danach in dem Makro weitergearbeitet werden. Welche der definierten Tasten nun gedrückt wurde, wird in der Variablen *MacroDialogResult* gespeichert (hierbei handelt es sich um eine Variable, nicht um einen Befehl). Um später kontrollieren zu können, welche der Taste gedrückt wurde, muß unmittelbar hinter jedem ***DialogAddPushButton-Befehl*** der Inhalt von *MacroDialogResult* einer anderen Variablen zugeordnet werden (= Sicherstellen der gedrückten Taste). Standardmäßig wird 1 der OK- und 2 der Abbrechen-Taste zugeordnet. Der Inhalt von *MacroDialogResult* wird nach der Ausführung von *DialogDestroy* bzw. *DialogDismiss* gelöscht und kann darum nicht mehr überprüft werden, es sei denn, der Inhalt von *MacroDialogResult* wurde vorher in eine andere Variable »gerettet«, die dann in den nachfolgenden Befehlen angesprochen werden kann. Anstelle des deutschen Ausdrucks können Sie auch den englischen verwenden: *MacroDialogResult*.

Beispiel

```
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; 0; "Taste &A")
```

```
TasteA=              MacroDialogResult
DialogAddPushButton  (100; 0; 50; 40; 50; 15; 1; "Taste &B")
TasteB=              MacroDialogResult
DialogDisplay        (100; TasteB)
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)
```

In diesem Beispiel wird ein Dialogfeld mit den Tasten *TasteA*, *TasteB*, *OK* und *Abbrechen* angezeigt. Welche der Tasten gedrückt wurde, wird in der Variablen *MacroDialogResult* gespeichert. Der Inhalt der Variablen muß unmittelbar hinter der betreffenden Tastendefinition in einer weiteren Variablen gesichert werden. Nach dem Klicken auf einer Taste wird in Verbindung mit dem *Switch*-Befehl angezeigt, welche Taste gedrückt wurde. Der Befehl *DialogDestroy* darf erst dann ausgeführt werden, wenn *MacroDialogResult* nicht mehr benötigt wird, es sei denn, der Inhalt von *MacroDialogResult* wurde vorher in eine andere Variable »gerettet«. In dem folgenden Beispiel wäre dann unter *Switch* der Name dieser Variablen anzugeben:

```
DialogDisplay        (100; TasteB)
Auswahl=             MacroDialogResult
DialogDestroy        (100)
Switch               (Auswahl)
  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
```

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

Weitere Hinweise siehe: DialogAddPushButton

6.74 MacroInfo

Befehlsform: *Variable1*=**MacroInfo**(*Info;CallLevel;{<Wert>Variable2}*)

Über diesen Befehl erhalten Sie Informationen über das System, den Makrostatus, Informationen über einen Label oder über Routinen innerhalb eines Makros.

Beispiel

Die Makroausführung ist abhängig von aufgerufenen Labels und Routinen, nach den jeweils zugeordneten Daten und der Strukturtiefe. Wird in einer Hauptroutine eines Makros Label »A« aufgerufen und von hier aus die Procedure »B«, die wiederum die Funktion »C« aufruft, ist folgendes zutreffend:

Die Call-Tiefe ist 2 (die Call-Tiefe ist 0, wenn das Makro startet), und die Daten für A, B oder C werden durch das Makrosystem gespeichert.

Der/die aktuelle Label/Routine ist »C«.

Wenn Daten für »C« erforderlich sind, muß CallLevel als »Current!« festgelegt werden.

Wenn Daten für »B« erforderlich sind, muß CallLevel als »Previous!« festgelegt werden, weil Routine »B« Routine »C« aufgerufen hat.

Wenn Daten für »A« erforderlich sind, muß CallLevel als »Previous! + 1!« festgelegt werden.

Immer wenn Daten für eine Routine vor Previous! benötigt werden, definieren Sie CallLevel als »Previous! +1«, »Previous! + 2« usw.

Sie können auch einen numerischen Wert für den CallLevel-Parameter definieren, durch den auf einfache Art und Weise Daten von allen aktiven Labels und Routinen in einer FOR-Schleife ermittelt werden. Definieren Sie z. B. Current! als 0, Previous! als 1, den Label oder die Routine vor Previous! als 3 usw.

Starten Sie die Schleife mit 0, und verwenden Sie MacroInfo(CallDepth!), um die Schleife in Verbindung mit Variable1 in der CallDepth! zu beenden.

Parameter

Info	System-Informationen:		
		PlatFormName!	String
		PlatFormVersion!	Numerisch
		PlatFormVersionString!	String
		PerfectFitVersion!	Numerisch
		PerfectFitVersionString!	String
		PerfectScriptVersion!	Numerisch
		PerfectScriptVersionString!	String
		PerfectScriptLanguageCode!	String
		PerfectScriptLanguageName!	String
	Makro-Informationen:		
		DialogsExist!	Ausdruck
		BreakpointsExist!	Ausdruck

	ErrorNumber!	Numerisch
	MacroDialogResult!	Beliebig
	MacroArgs!	Array
	CallDepth!	Numerisch (0=Makrostart)
Label/Routinen-Inform.		
	InRoutine!	Ausdruck
	InCallback!	Ausdruck
	InAssert!	Ausdruck
	FileName!	String
	LineNumber!	Numerisch (Makro-Zeilennummer)
	LabelOrRoutineName!	Label
	ChainFileName!	String
	ChainFileArgs!	Array
	DefaultUnits!	Numerisch
	PersistAllOn!	Ausdruck
	VarrErrChkOn!	Ausdruck
	CancelOn!	Ausdruck
	CancelSetup!	Ausdruck
	CancelCall!	Ausdruck
	CancelHandler!	Label
	ErrorOn!	Ausdruck
	ErrorSetup!	Ausdruck
	ErrorCall!	Ausdruck
	ErrorHandler!	Label
	NotFoundOn!	Ausdruck
	NotFoundSetup!	Ausdruck
	NotFoundCall!	Ausdruck
	NotFoundHandler!	Label
	UserDefinedConditionOn!	Ausdruck
	UserDefinedConditionSetup!	Ausdruck
	UserDefinedConditionCall!	Ausdruck
	UserDefinedConditionHandler!	Label

CallLevel Wird bei verschiedenen Info-Typen zum Definieren eines Labels oder einer Routine zur Datenübergabe verwendet. Standardwert: Current!.

Current!
Previous!

Wert Wird bei verschiedenen Info-Typen verwendet, um zusätzliche Angaben zu definieren.

6.75 MakroStatusPrompt(nur bis Version 6.1)

Befehlsform: **MakroStatusPrompt**(*Status;Meldung*)

Dieser Befehl zeigt eine Meldung in der Statuszeile so lange an, bis sie durch eine andere Meldung überschrieben (z. B. erneuter *MakroStatusPrompt*) bzw. ausgeblendet wird. Die Meldung wird nur dann angezeigt, wenn die Statuszeile eingeblendet ist. Stellen Sie ggf. im Makro sicher, daß dies der Fall ist. Verwenden Sie hierzu den produktspezifischen Befehl *StatusBarShow(On!)*.

Parameter

Status Anzeigen (*On!*) oder Ausblenden (*Off!*) der Meldung.

Meldung Beliebiger Text (Konstante oder Variable).

Beispiel

```
MakroStatusPrompt    (On!;"Berechnung wird gerade durchgeführt, bitte warten.")

MakroStatusPrompt    (Off!)
```

Weitere Hinweise siehe: Tastatur

6.76 Menu

Befehlsform: **Menu**(*Auswahl;Typ;HorizPosition;VertikPosition;{Option;..Option}*)

Zur Steuerung von Makros können Sie eigene Menüs erstellen, die Ihnen bei der Auswahl bestimmter Verarbeitungsschritte behilflich sind. Nach der Ausführung dieses Befehls wird ein Menü mit den Menüpunkten eingeblendet, die Sie unter *Option* spezifiziert haben. Die angezeigten Menüpunkte können Sie mit der Maus oder mit den zugehörigen Ziffern/Buchstaben auswählen, d. h. jedem Menüpunkt ist je nach Definition (Parameter *Typ*) eine Zahl oder ein Buchstabe zugeordnet (siehe unten). Diese werden von WordPerfect automatisch in aufsteigender Reihenfolge (bei Ziffern bei 1, bei Buchstaben bei A beginnend) den Menüpunkten zugeordnet. Sie werden jeweils vor dem zugehörigen Menüpunkt angezeigt. Nach der Anzeige des Menüs wird die getroffene Auswahl in dem Feld *Auswahl* gespeichert und kann danach im Makro entsprechend abgefragt werden. Wurde das Menü versehentlich angezeigt, können Sie es durch Drücken von Alt wieder ohne Folgen ausblenden.

Achtung! Bei der Auswahl eines Menüpunkts wird dem Feld *Auswahl* immer ein numerischer Wert zugeordnet, auch dann, wenn Sie unter *Typ* den Parameter *Letter* gewählt haben. D. h. bei der Auswahl von A wird 1, bei der Auswahl von H wird 8 zugeordnet usw.

Parameter

Auswahl Speicherung der getroffenen Menüauswahl. Hier werden in jedem Fall Ziffern gespeichert, auch dann, wenn Menükennbuchstaben angezeigt werden! Haben Sie als *Typ Letter* gewählt, werden den Buchstaben intern Ziffern zugeordnet. Für A steht 1, für B steht 2, für H steht 8 usw.

Typ Jedem Menüpunkt wird je nach Auswahl eine Menükennziffer oder ein Menükennbuchstabe zugeordnet. Die Zuordnung trifft WordPerfect automatisch, bei 1 beginnend (beachten Sie bitte auch den Parameter *Option*). Möchten Sie Ziffern verwenden, müssen Sie hier ***Digit*** eingeben, wenn Sie Buchstaben verwenden wollen, wählen Sie ***Letter***. Die Texte *Digit* und *Letter* dürfen nicht durch Hochkommata eingeschlossen werden.

HorizPosition Horizontale Position, an der das Menü am Bildschirm erscheinen soll. Wird keine bestimmte Position gewünscht, können Sie den Parameter weglassen, indem Sie ein Semikolon setzen. In diesem Fall erscheint das Menü in der Mitte des Anwendungsfensters. Die Angabe erfolgt in Pixel, gemessen vom linken Rand des Anwendungsfensters aus. Beachten Sie bitte hierbei die unterschiedlichen Pixelgrößen, die von der Bildschirmauflösung abhängig sind.

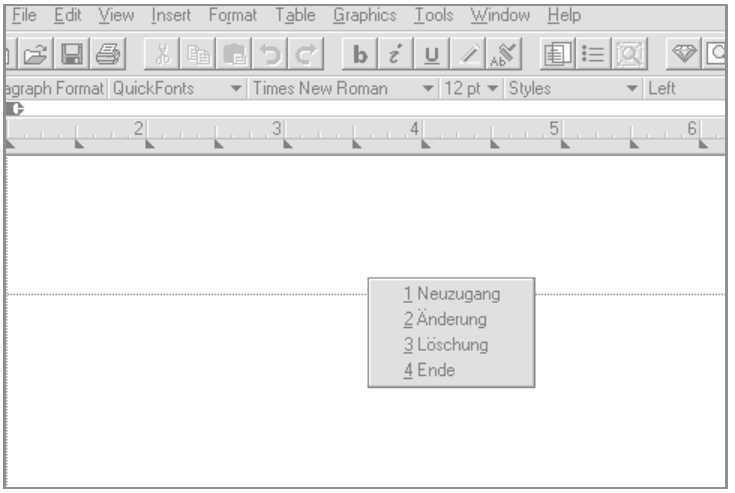
VertikPosition Vertikale Position, an der das Menü am Bildschirm erscheinen soll. Wird keine bestimmte Position gewünscht, können Sie den Parameter weglassen, indem Sie ein Semikolon setzen. In diesem Fall erscheint das Menü in der Mitte des Anwendungsfensters. Die Angabe erfolgt in Pixel, gemessen vom oberen Rand des Anwendungsfensters aus. Beachten Sie bitte hierbei die unterschiedlichen Pixelgrößen, die von der Bildschirmauflösung abhängig sind.

Option Hier müssen Sie die anzuzeigenden Menüpunkte in der gewünschten Reihenfolge angeben. Das können Konstanten sein oder Variablennamen. Die Anzahl der Menüpunkte ist je nach *Typ* begrenzt. Bei *Digit* sind maximal 9 Optionen erlaubt, bei *Letter* maximal 26.

Beispiel

```
Application                    (A1;"WordPerfect";Default;"DE")
Assign                         (Menüpunkt4;"Überleitung")
Assign                         (WelchesMenü;1)
Menu
(Auswahl;Letter;;;{"Neuzugang";"Änderung";"Löschung";Überleitung})
Case Call  (Auswahl;{                                  //Achtung: Geschweifte Klammer.
                               1;Neu;
                               2;Ändern;
                               3;Löschen;
                               4;Menüpunkt4};          //Achtung: Geschweifte Klammer.
                               Neu)
Label                          (Neu)
Run                            ("C:\WPWIN7\MAKROS\NEU.WCM")
Quit
```

```
Label                          (Ändern)
Run                            ("C:\WPWIN7\MAKROS\AENDERN.WCM")
Quit

Label                          (Löschen)
Run                            ("C:\WPWIN7\MAKROS\LOESCHEN.WCM")
Quit
Label                          (Menüpunkt4)
If                             (WelchesMenü=1)
   Run                         ("C:\WPWIN7\MAKROS\UEBERLEI.WCM")
Else
   Run                         ("C:\WPWIN7\MAKROS\UMBUCH.WCM")
Quit
```



Bei der Ausführung dieses Makros wird das abgebildete Menü eingeblendet. Über *Case Call* wird die Menüauswahl überprüft und aufgrund der Auswahl jeweils ein Makro aufgerufen. Beachten Sie bitte, daß hier bei *Case Call* Ziffern geprüft werden, obwohl unter *Typ Letter* gewählt wurde (siehe oben)! Die ersten drei Menüpunkte wurden bei dem Befehl *Menu* als Konstanten definiert, der vierte Menüpunkt als Variable.

Bei dem vierten Menüpunkt haben Sie somit die Möglichkeit, je nach Bedarf unterschiedliche Menüpunkte anzuzeigen. Der Text für diesen Menüpunkt kann an einer beliebigen Stelle des Makros in die Variable *Menüpunkt4* übertragen werden. In der Unterroutine (*Überleiten*) muß dann noch geprüft werden, welcher Menüpunkt gesetzt wurde, um das entsprechende Makro aufzurufen. Wenn Sie komfortablere Menüs erstellen wollen, können Sie diese über die Befehle Dialog... oder den Dialogeditor erstellen. Dieses Beispiel finden Sie auf der CD-ROM unter dem Dateinamen MENU.WCM.

Weitere Hinweise siehe: Dialog-Befehle

6.77 Menulist

Befehlsform: **MenuList**(*Variable;{Menüpunkt,Menüpunkt;...};Titel;HorPos;VerPos*)

MenuList ist ein Makrobefehl von WP-DOS-6.0 und entspricht dem WP-Windows-Befehl *Menu*. Er wurde in den Befehlsumfang von WordPerfect für Windows aufgenommen, um mit der DOS-Version kompatibel zu sein. Entnehmen Sie darum bitte die allgemeine Beschreibung dem Befehl *Menu*. Mit Alt oder Esc, wird ein Menü ohne Auswahl wieder auszublenden.

Parameter

Variable Nach der Anzeige des Menüs wird in dieser Variablen die getroffene Auswahl gespeichert. Das kann bei der Verwendung von Menükennziffern eine Ziffer von 1–9 sein oder bei Menükennbuchstaben von 1–26. Achten Sie darauf, daß auch bei der Auswahl von Buchstaben nur Ziffern zurückgegeben werden (A = 1, E = 5 usw.). Wurde (Esc) oder (Alt) gedrückt, wird 0 zurückgegeben.

Menüpunkt Angabe der Menüpunkte in der anzuzeigenden Reihenfolge. Bis zu neun Menüpunkte werden beim Anzeigen durch Ziffern gekennzeichnet, mehr als neun Menüpunkte durch Buchstaben.

Titel Titel der Menüliste.

HorPos Horizontale Position (siehe unter *Menu*).

VerPos Vertikale Position (siehe unter *Menu*).

Beispiel: Siehe sinngemäß unter *Menu*.

Weitere Hinweise siehe: Menu

6.78 Messagebox

Befehlsform: *Variable1*=**MessageBox**(*Variable2;Titel;Nachricht;Style;ParameterDaten*)

Der unter *Nachricht* definierte oder generierte Text wird in einem Dialogfeld mit der unter *Titel* eingegebenen Überschrift angezeigt. Das Dialogfeld wird in der Bildschirmmitte eingeblendet. Bei Bedarf können Sie jedes Dialogfeld mit einem der zur Verfügung stehenden Icons versehen (siehe unten). Zu einer Zeit kann immer nur eine Meldung am Bildschirm angezeigt werden. Der Befehl kann auch in Verbindung mit *If* verwendet werden.

Parameter

Variable1 Beim Ausblenden der Nachrichtenbox wird in dieser Variablen der Wert der gedrückten Taste gespeichert. Durch Prüfen der Variablen kann im Makro entsprechend reagiert werden. Kann die Nachricht wegen Platzmangels nicht angezeigt werden, enthält diese Variable 0. Folgende Rückmeldungen sind möglich:

AbbortRetryIgnore!	CancelButton!	Error!
IgnoreButton!	YesNo!	OK!
RetryCancel!	YesNoCancel!	OKCancel!

Variable2 Angabe der gedrückten Taste (siehe *Variable1*). Wird dieser Parameter weggelassen, muß dafür eine Semikolon gesetzt werden.

Titel Hinweistext in der Titelzeile der Nachrichtenbox. Wird kein Titel eingegeben, wird *PerfectScript* angezeigt.

Nachricht Der angegebene Text wird in der Nachrichtenbox angezeigt. Je nach Style (siehe unten) können verschiedene Icons links des Textes und verschiedene Befehlstasten in der Mitte unterhalb des Textes angezeigt werden. Mehrere Texte können in Verbindung mit dem Auslassungszeichen ^ und dem Parameter *ParameterDaten* angezeigt werden, wenn der Parameter *HasParameters!* verwendet wurde. Zum Einfügen eines festen Zeilenumbruchs können Sie den Befehl *NTOC*(*0F90Ah*) verwenden (siehe Beispiel).

Style Styles zur Gestaltung und Handhabung der Nachrichtenbox. Mehrere Styles können durch kombiniert werden. Aus den folgenden Gruppen darf jeweils nur ein Style verwendet werden:

Tasten:

AbortRetryIgnore!	Abbrechen, Wiederholen, Ignorieren
OK!	OK (auch wenn keine Taste definiert)
OKCancel!	OK, Abbrechen
RetryCancel!	Wiederholen, Abbrechen
YesNo!	Ja, Nein
YesNoCancel!	Ja, Nein, Abbrechen
Icons:	
IconNone!	Kein Icon.
IconAsterisk!	Kleines i in blauem Kreis.
IconExclamation!	Ausrufezeichen in gelbem Kreis.
IconHand!	Stopzeichen in rotem Kreis.
IconInformation!	Kleines i in blauem Kreis.
IconQuestion!	Fragezeichen in grünem Kreis.
IconStop!	Stopzeichen in rotem Kreis.

Modalität:

ApplicationModal!	Wechsel zu einem anderen Programm. Zuvor muß durch Anklicken einer der Tasten die Nachricht ausgeblendet werden. Wird keine der drei Möglichkeiten gewählt, gilt dieser Parameter als Standardwert.
SystemModal!	Ausblenden der Nachricht bevor das Makro fortgesetzt wird bzw. zu einem anderen Programm gewechselt wird.
TaskModal!	Wechsel zu einem anderen Programm. Zuvor muß durch Anklicken einer der Tasten die Nachricht ausgeblendet werden.

Weitere:

Beep!	Akustisches Signal beim Anzeigen der Nachrichtenbox.
HasParameters!	Die mit einer Nummer versehenen Auslassungszeichen sind durch den zugehörigen Zeichenstring von *ParameterDaten* zu ersetzen (siehe Beispiel).

ParameterDaten Text für den Parameter *Nachricht*. Maximal zehn Parameter sind erlaubt, die jeweils durch ein Semikolon zu trennen sind. Achtung: Die Numerierung beginnt bei 0.

Beispiel

```
1         P0=                 "Ja für weiter oder"
          P1=                 "Nein zum Beenden."
          MessageBox          (Auswahl;"Adressen erfassen"; "Wählen Sie ^0 ^1";
                              YesNo! | IconQuestion! | HasParameters!;{P0;P1})
          If                  (Auswahl = 6)
            Call              (Prüfen)
          Else
            Quit
          EndIf
```

Die Meldung *Wählen Sie Ja für weiter oder Nein zum Beenden.* wird angezeigt. Links der Meldung wird ein Icon mit einem Fragezeichen eingeblendet und darunter jeweils eine Taste mit [Ja] und [Nein]. Vergessen Sie nicht den Parameter *HasParameters!*, sonst wird nur der Text *Wählen Sie ^0 ^1* angezeigt. Der Wert der gedrückten Taste wird in *Auswahl* gespeichert. Über einen *If*-Befehl kann dann die weitere Makroverarbeitung gesteuert werden. Enthält *Auswahl* den Wert 6, wurde [Ja] gedrückt, bei 7 [Nein].

```
2         Vorschub=           Ntoc(0F90Ah)
          Text1=              "Fehler bei der Dateneingabe:"
          Text2=              "Postleitzahl fehlt"
          MessageBox          (Auswahl;"Adressen erfassen";
                              Text1+Vorschub+Vorschub+Text2;IconStop!)
```

Bei dieser Nachricht wird zwischen *Text1* und *Text2* eine Leerzeile eingefügt. Einzelne Leerzeilen können Sie auch wie folgt einfügen:

```
MessageBox  (Auswahl;"Adressen erfassen";Text1+Ntoc(0F90Ah)+Text2;IconStop!)
```

Weitere Hinweise siehe: Prompt

6.79 Net-Befehle

Mit den Net...-Befehlen können Sie innerhalb eines Netzwerks arbeiten.

NetAddConnection

Variable=**NetAddConnection**(RemoteName;LocalName;Provider;Typ;Paßwort;UserName;Optionen)

Verbindung zu einem Netzwerk herstellen.

Parameter

Variable Enthält 0, wenn die Verbindung aufgebaut werden konnte. Bei alle anderen Werten konnte keine Verbindung hergestellt werden.

RemoteName Name eines Netzwerkes (beachten Sie die Namenskonventionen).

LocalName Names eines lokalen Netzwerkes, z. B. F: oder LPT1:. Wird dieser Parameter weggelassen, wird eine Netzwerkverbindung ohne Umleitung eines lokalen Zuordnung aufgebaut.

Provider Definiert einen Netzwerk-Provider, zu dem eine Verbindung aufgebaut werden soll. Wird dieser Parameter weggelassen oder wird "" verwendet, versucht das System den RemoteName zu bestimmen. Verwenden Sie diesen Parameter nur dann, wenn ein Netzwerk-Provider angegeben werden muß. Andernfalls entscheidet Windows, welcher Netzwerk-Provider zugeordnet werden sollte. Wird dieser Parameter angegeben, versucht Windows die Verbindung nur zu diesem Provider herzustellen.

Typ Disk! (= Standard)

Printer!

Paßwort Zugehöriges Paßwort, um die Verbindung aufzubauen. Wird kein Paßwort angegeben, wird das zum User gehörende Paßwort verwendet. Wird der Parameter weggelassen oder wird "" verwendet, wird kein Paßwort benutzt.

UserName Benutzername zum Aufbau der Verbindung. Ist kein Benutzername verfügbar, wird der Default-Username verwendet (wird vom System angeboten).

Optionen	DontAutoRestore!	Kein automatischer Aufbau die bisherige Verbindung (= Standard).
	AutoRestore!	Automatischer Aufbau der bisherigen Verbindung.

NetCancelConnection

Variable=**NetCancelConnection**(ResourceName;Optionen;Force)

Abbruch einer Netzwerkverbindung.

Parameter

Variable	Enthält *True*, wenn die Verbindung beendet werden konnte oder *False*, wenn die Verbindung nicht beendet werden konnte.	
ResourceName	Remote-Netzwerk oder lokales Laufwerk.	
Optionen	DontAutoRestore!	Keine Speicherung der Netzwerkverbindung.
	AutoRestore!	Windows stellt keine erneute Verbindung aufgrund einer nachträglichen versuchten Anmeldung her.
Force	Wird *True* angegeben, wird die Verbindung abgebrochen, gleichgültig, ob Dateien oder Jobs noch geöffnet sind. Wird der Parameter weggelassen oder *False* verwendet, wird der Verbindungsabbruch nicht ausgeführt, wenn noch offene Dateien oder Jobs vorhanden sind.	

NetConnectionDlg

Variable=**NetConnectionDlg**(Type)

Anzeigen eines Dialogfelds zum Aufbau einer Netzwerkverbindung.

Parameter

Variable	Zurückgegebener Statuscode des Dialogfelds.
Type	Disk!

NetDisconnectDlg

Variable=**NetDisconnectDlg**(Type)

Anzeigen eines Dialogfelds zum Beenden einer Netzwerkverbindung.

Parameter

Variable	Zurückgegebener Statuscode des Dialogfelds.
Type	Disk! (Standard)
	Printer!

NetGetConnection

Variable=**NetGetConnection**(LocalName)

Ermittelt den Namen einer Netzwerk-Komponente in Verbindung mit einem lokalen Gerät.

Parameter

Variable	Enthält die ermittelte Netzwerk-Komponente. Das Feld ist leer, wenn ein Fehler auftrat.
LocalName	Name des lokalen Geräts

NetGetUniversalname

Variable=**NetGetUniversalName**(Name)

Ermittelt den UNC-Namen einer lokalen Datei.

Parameter

Variable	UNC-Name des unter Name angegebenen Dateinamens.
Name	Lokaler Name einer Datei.

NetGetUser

Variable=**NetGetUser()**

Ermittelt die Netzwerk-ID des Benutzers.

Parameter

Variable	Enthält die ermittelte Netzwerk-ID eines Benutzers.

6.80 NewDefault

Befehlsform: **NewDefault**(*Produktabkz*)

Angabe einer neuen Produkt-Kennung, die ab diesem Zeitpunkt aktiviert wird. Sind in einem Makro mehrere dieser Befehle vorhanden, hat ein Befehl so lange Gültigkeit, bis ein neuer Befehl *NewDefault* ausgeführt wird, oder bis *EndApp* auftritt. Zu einer Zeit kann immer nur eine Produktabkürzung aktiv sein. Jeder Produktbefehl, der keine Produktkennung aufweist, wird so behandelt, als hätte er die hier angegebene Produktabkürzung. Wird keine Produktabkürzung vorgegeben, erfolgt bei der Makro-Kompilierung eine Fehlermeldung. Eine Produktabkürzung wird üblicherweise am Beginn eines Makros mit dem Befehl *Application* gesetzt. 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

Produktabkz Produkt-Kennung. Die Angabe muß mit einer Produkt-Kennung, die vorher im Makro unter *Application* angegeben wurde, übereinstimmen. Die Angabe darf nicht in Hochkommata eingeschlossen werden.

Beispiel: **NewDefault**(A1)

Weitere Hinweise siehe: Application, EndApp

6.81 Next

Befehlsform: Next

Dieser Befehl wird immer in Verbindung mit den Befehlen *For*, *ForEach*, *ForNext*, *Repeat* oder *While* verwendet. Normalerweise wird erst beim Antreffen der Befehle *EndFor*, *EndWhile* oder *Until* zum Beginn der Schleife verzweigt, um den nächsten Schleifendurchgang zu durchlaufen. Über den Befehl *Next* kann die unmittelbar anschließende Befehlsfolge der aktuellen Schleife bis zum zugehörigen *End...* übersprungen werden, um mit dem nächsten Schleifendurchgang fortzufahren. Die Ausführung von *Next* kann z. B. in Verbindung mit einem *If*-Befehl gesteuert werden (siehe Beispiel).

Beispiel

```
ForNext             (Zähler;1;10;1)
  Betrag=           (Brutto-Abzug)
  If                (Betrag=0)
     Next
  EndIf
  ::::
EndFor
```

In diesem Beispiel werden die Befehle zwischen *EndIf* und dem zugehörigen *EndFor* nicht ausgeführt, wenn *Betrag* den Wert 0 enthält. In diesem Fall wird die Verarbeitung wieder bei *ForNext* fortgesetzt.

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

6.82 NotFound

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

Eine *NotFound*-Bedingung wird gesetzt, wenn bei einem Suchvorgang das Such- argument im Dokument nicht gefunden wurde. Standardmäßig wird in diesem Fall das Makro beendet. Um trotzdem mit der Makroverarbeitung fortfahren zu können, können Sie diese Bedingung ausschalten. Über *NotFound* können Sie die Bedingung aktivieren oder deaktivieren. Zusätzlich können Sie den Befehl in Verbindung mit *OnNotFound* verwenden. Wenn in einem Makro eine *OnNotFound*-Bedingung vor einer *NotFound*-Bedingung auftritt, kann das Makro bei dem unter *OnNotFound* angegebenen Label weiterarbeiten.

Parameter

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

Status Wählen Sie hier aus, ob die *NotFound*-Bedingung aktiviert oder deaktiviert werden soll:

On!	Bedingung wird aktiviert, Makro bricht ab.
Off!	Bedingung wird deaktiviert, Makro kann bei Bedarf weiterarbeiten.

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

Weitere Hinweise siehe: Assert, OnNotFound

6.83 NToC

Befehlsform: *Variable*=**NToC**(*Zahl*) oder
Variable=**NToC**(ZeichenSatz;Zeichen)

Die erste Variante konvertiert einen Tastenwertes (numerischer Wert) in das entsprechende Zeichenäquivalent. Beachten Sie bitte auch den Befehl *CToN*. Es wird kein *sinnvolles* Ergebnis angezeigt, wenn der Befehl bei einer Zahl verwendet wird, zu der es kein Zeichenäquivalent gibt.

Die zweite Variante konvertiert das Zeichen einer WP-Zeichensatznummer/Zeichennummer in das zugehörige Zeichen. Wird die Zeichensatznummer weggelassen, wird Zeichensatz 0 (= ASCII) angenommen.

Parameter

Variable Enthält das konvertierte Zeichen.

Zahl Tastenwert, der in ein Zeichen umzurechnen ist. Nach der Ausführung dieses Befehls wird das ermittelte Zeichen auf Cursorposition eingefügt. In Verbindung mit *Assign* kann es auch in einer Variablen gespeichert werden (siehe Beispiel).

ZeichenSatz Zeichensatznummer eines WP-Zeichensatzes.

Zeichen Zeichennummer innerhalb des unter *ZeichenSatz* ausgewählten Zeichensatzes.

Beispiel

```
1         GetString            (Eingabe;"Bitte Tastenwert eingeben.";
                               "Befehl ""NToC""  testen")
2         Rechnen=             Eingabe
3         Ausgabe=             NtoC(Eingabe)
4         Zeichensatznummer=   (Rechnen DIV  256)
5         Zeichennummer=       (Rechnen MOD 256)
6         Meldung=             "Ermitteltes Zeichen = "+Ausgabe+",
                               Zeichensatznummer: "+Zeichensatznummer+",
                               Zeichennummer: "+Zeichennummer
7         Prompt               "Befehl  ""NtoC""  testen";Meldung;;;)
            Wait               (30)
          EndPrompt
```

In dem gezeigten Beispiel soll für eine eingegebene Zahl das zugehörige Zeichen ermittelt werden.

1 Anzeige einer Dialogbox, Übernahme der eingetippten Zahl in die Variable *Eingabe*.

2 Speichern der Eingabe in der Variablen *Rechnen*, da nach der Ausführung von NToC der Inhalt von *Eingabe* verändert ist. Die Variable *Rechnen* wird benötigt, um die Zeichensatz- und die Zeichennummer zu errechnen.

4 Errechnen der Zeichensatznummer.

5 Errechnen der Zeichennummer.

6 Generieren einer Meldung.

7 Anzeigen der Meldung.

Sie finden ein Beispiel für diesen Befehl auf der CD-ROM unter dem Dateinamen NTOC.WCM.

Weitere Hinweise siehe: CtoN

6.84 NumStr

Befehlsform: *StringVariable*=**NumStr**(*NumVariable;NachKomma;VorKomma*)
NumStr(*StringVariable;NachKomma;NumVariable*) *(WPWin 5.2)*

Umwandeln eines numerischen Ausdrucks in einen alphanumerischen Ausdruck. Dieser wird in *StringVariable* gespeichert. Aus 5,50 wird dann der String »5,50«. Obwohl beide Zahlen am Bildschirm identisch aussehen, werden sie von WordPerfect unterschiedlich interpretiert. Mit 5,50 können Sie z. B. Rechenoperationen durchführen, mit dem String »5,50« nicht. Umgekehrt können Sie »5,50« drucken oder über *Prompt* anzeigen, 5,50 müssen Sie für diese Funktionen über *NumStr* erst in einen String konvertieren. Wenn Sie den Parameter *NachKomma* verwenden, können Sie 0–16 Dezimalstellen angeben. Wird dieser Parameter nicht verwendet, werden standardmäßig sechs Dezimalstellen berücksichtigt.

Beispiel

```
NumStr          (Drucken;4;123,4567890)     oder
Drucken         NumStr(123,4567890;4)
Type            (Drucken)
```

Der numerische Wert 123,4567890 wird hier in den String »123,4568« konvertiert (um diese Zahl z. B. zu drucken) und in der Variablen *Drucken* gespeichert. Die letzte der vier Stellen wird hierbei ggf. aufgerundet.

Nehmen Sie keine Konvertierung unter Beibehaltung desselben Variablennamens vor, denn die korrekte Ausführung mathematischer Funktionen ist dann mit dieser Variablen nicht mehr möglich. Ist die Konvertierung dennoch erforderlich, verwenden Sie für die konvertierte Variable einen anderen Namen, z. B. *DruBetrag=NumStr(Betrag;2)*. In diesem Fall können Sie mit der ursprünglichen Variablen *Betrag* weiterhin mathematische Funktionen ausführen, wogegen das mit *DruBetrag* zu Problemen führen kann.

Achtung: Wurde das Makro unter WPWIN 5.x erstellt, muß der Befehl zwingend dem neuen Format angepaßt werden (war unter 6.1 nicht erforderlich). Das Makro ist in Version 7 zwar lauffähig, der Befehl liefert jedoch falsche Ergebnisse.

Parameter

StringVariable Beliebiger Variablenname. In dieser Variablen wird der konvertierte numerische Wert als String gespeichert.

NumVariable Hier können Sie einen numerischen Wert oder eine Variable mit einem numerischen Inhalt angeben. Dieser Wert wird in einen String konvertiert und in *StringVariable* gespeichert.

NachKomma Geben Sie die Anzahl der Dezimalstellen (numerische Ganzzahl) an, die bei der Konvertierung berücksichtigt werden sollen. Ist der angegebene Wert größer als die Anzahl der Stellen hinter dem Komma, wird automatisch aufgerundet. Wenn Sie hier keine Angaben machen, wer-

den standardmäßig sechs Dezimalstellen berücksichtigt. Die Verwendung ist wahlfrei. Wird jedoch *VorKomma* verwendet, muß als Ersatz ein Semikolon angegeben werden.

VorKomma Numerischer Ausdruck, der die Anzahl der Stellen vor dem Komma angibt. Die Verwendung ist wahlfrei.

Weitere Hinweise siehe: StrNum

6.85 OnCancel

Befehlsform: *RetLabel*=**OnCancel**(*Label*) oder **Oncancel**(*Label*)

Dieser Befehl dient der Regelung von Ausnahmesituationen. Hier wird festgelegt, was zu tun ist, wenn ein Makro abgebrochen wird oder von einer Unterroutine bzw. von einem mit *Run* aufgerufenen Makro der Befehl *Assert*(*CancelCondition!*) oder *Return*(CancelCondition!) benutzt wurde. Das Makro verzweigt in diesem Fall zu dem angegebenen Label. Der Befehl muß im Makro so plaziert werden, daß er bereits vor einer evtl. auftretenden Abbruchbedingung ausgeführt wurde. Tritt eine Abbruchbedingung auf bevor *OnCancel* oder *OnCancel Call* ausgeführt wurde, wird das Makro beendet. Sind in einem Makro mehrere *OnCancel*-Befehle definiert, tritt der *OnCancel*-Befehl in Kraft, der als letzter ausgeführt wurde.

Während der Makroausführung wird immer der jeweils zuletzt ausgeführte *OnCancel*-Befehl berücksichtigt. Stellen Sie darum sicher, daß beim Einfügen neuer Bedingungen auch evtl. zusätzliche *OnCancel*-Befehle eingefügt werden.

Wird *OnCancel* oder *OnCancel Call* nicht verwendet, wird durch Drücken von [*Abbrechen*] das Makro beendet. WordPerfect zeigt in diesem Fall noch einen Hinweis an, der Sie auf den vom Benutzer erzeugten Abbruch aufmerksam macht. Möchten Sie für bestimmte Bereiche eines Makros die Abbruchbedingung ausschalten, verwenden Sie ab der gewünschten Stelle den Befehl *Cancel*(*Off!*). Mit *Cancel*(*On!*) können Sie die Abbruchbedingung jederzeit wieder einschalten.

Parameter

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

Label Geben Sie hier einen Labelnamen an, zu dem das Makro verzweigen soll, wenn eine Abbruchbedingung auftritt oder gesetzt wurde. Die Verarbeitung wird bei diesem Label fortgesetzt. Hier müssen Sie nun entscheiden, was in diesem Fall getan werden muß. Sie können das Makro beenden oder an einer anderen Stelle innerhalb des Makros mit der Verarbeitung fortfahren (z. B. unter anderen Bedingungen).

Beispiel

```
Application          (A1; "WordPerfect"; Default; "DE")
Zähler=              0
OnCancel             (Ende)

Label                (Anfang)
Zähler=              Zähler+1
Anzeigen=            NumStr(Zähler)
Prompt               ("Drücken Sie Esc oder klicken Sie auf Abbrechen";Anzeigen;;;)
Go                   (Anfang)

Label                (Ende)
Prompt               ("Abbruch";"Sie haben das Makro abgebrochen";;;)
Wait                 (30)
EndPrompt
Quit
```

Dieses Makro addiert eine 1 zu dem Inhalt der Variablen *Zähler* und zeigt das Ergebnis über *Prompt* am Bildschirm an. Die Schleife wird so lange wiederholt, bis Esc gedrückt wurde. Nach dem Drücken von Esc tritt die Bedingung OnCancel 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.

```
Application          (A1; "WordPerfect"; Default; "DE")
FileNew              ()
Zähler=              0
RetCancel=           Cancel(Off!)
RetLabel=            OnCancel(Abbruch)
Type                 ("Return-Label = " + RetLabel)
HardReturn           ()
Display              (On!)

Label                (Anfang)
Zähler=              (Zähler+1)
If                   (Zähler=10)
   Assert            (CancelCondition!)
EndIf
Type                 (Zähler)
HardReturn           ()
Go                   (Anfang)

Label                (Abbruch)
Type                 ("Das Makro wurde beim 10. Durchlauf abgebrochen.")
::::::
RetLabel=            OnCancel(Weiter)
If                   RetLabel=("Abbruch")
   ::::::
Else
   ::::::
EndIf
RetCancel=           Cancel(Off!)
```

```
HardReturn          ()
Type                ("Return-Label des zuletzt ausgeführten ONCANCEL = " + RetLabel)
HardReturn          ()
Type                ("Letzter Cancel-Status = " + RetCancel)

Label               (Weiter)
:::::
```

In diesem Beispiel wird unter *OnCancel* der Label *Abbruch* definiert, zu dem verzweigt werden soll, wenn eine *Cancel*-Bedingung auftritt. Beim ersten *OnCancel* ist die Variable *RetLabel* leer, weil zuvor kein *OnCancel* ausgeführt wurde. Nach dem Eintreten der Cancel-Bedingung (innerhalb der If-Schleife) verzweigt das Makro zu dem Label *Abbruch*. Für die weitere Programmverarbeitung wird hier eine neue *OnCancel*-Bedingung gesetzt, wobei die Variable *RetLabel* jetzt den Labelnamen der zuvor ausgeführten *OnCancel*-Bedingung enthält (= Abbruch).

Diese Beispiele finden Sie auf der CD-ROM unter den Dateinamen CANCEL.WCM und ONCANCEL.WCM. Beachten Sie bitte auch die Beispielmakros RETURN.WCM und RETURN1.WCM.

Weitere Hinweise siehe: Assert(CancelCondition!), Cancel(On!/Off!), Label, OnCancel Call, Return.

6.86 OnCancel Call

Befehlsform: *RetLabel*=**OnCancel Call**(*Label*) oder **OnCancel Call**(*Label*)

Dieser Befehl entspricht weitgehend dem Befehl *OnCancel*. Der einzige Unterschied: Bei *OnCancel Call* wird eine Unterroutine aufgerufen, die durch *Return* beendet werden muß. Nach der Ausführung dieser Unterroutine wird im Makro an der Stelle weitergearbeitet, an der der Abbruch vorgenommen wurde. Wird *OnCancel* oder *OnCancel Call* nicht verwendet, wird durch Drücken von [*Abbrechen*] oder Esc das Makro beendet.

Parameter

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

Label Geben Sie hier den Namen einer Unterroutine an, die aufgerufen werden soll, wenn eine Abbruchbedingung auftritt oder gesetzt wurde. Die Verarbeitung wird bei diesem Label fortgesetzt. Die Unterroutine muß mit einem *Return* enden. Danach wird das Makro an der Stelle fortgesetzt, an der es unterbrochen wurde. Beachten Sie bitte auch die Hinweise bei dem Befehl *Label*.

Beispiel

```
Application          (A1; "WordPerfect"; Default; "DE")
Zähler=              0

Label                (Anfang)
OnCancel Call        (Abbruch)
Zähler=              Zähler+1
Anzeigen=            NumStr(Zähler)
Prompt               ("Drücken Sie Esc oder klicken Sie auf Abbrechen";Anzeigen)
Go                   (Anfang)

Label                (Abbruch)
OnCancel             (Ende)
Prompt               ("Was jetzt?";"Soll das Makro weiterarbeiten?")
Wait                 (30)
EndPrompt
Return

Label                (Ende)
Prompt               ("Abbruch";"Sie haben das Makro abgebrochen.")
Quit
```

Dieses Makro addiert eine 1 zu dem Inhalt der Variablen *Zähler* und zeigt das Ergebnis über *Prompt* am Bildschirm an. Die Schleife wird so lange wiederholt, bis Esc gedrückt oder auf [*Abbrechen*] geklickt wurde. Nach dem Drücken von Esc tritt die Bedingung *OnCancel Call* in Kraft, das Makro ruft die Unterroutine (*Abbruch*) auf. Hier wird eine Meldung am Bildschirm angezeigt, die auf den Abbruch hinweist.

Nach dem Klicken auf [*OK*] setzt das Makro seine Arbeit an der unterbrochenen Stelle fort. Wenn Sie auf [*Abbrechen*] klicken, wird das Makro beendet.

Beachten Sie bitte, das in der Unterroutine (*Abbruch*) zusätzlich der Befehl *OnCancel* verwendet wurde. Wird dieser Befehl hier nicht ausgeführt, ist der Befehl *OnCancel Call*, der weiter oben angegeben wurde, immer noch aktiv. Das Makro würde dann in eine Endlosschleife laufen.

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

Dieses Beispiel finden Sie auf der CD-ROM unter dem Dateinamen ONCANCAL.WCM. Beachten Sie bitte auch die Beispielmakros RETURN.WCM und RETURN1.WCM.

Weitere Hinweise siehe: AssertCancel, CancelOn, CancelOff, OnCancel Call Return.

6.87 OnCondition / OnCondition Call

Befehlsform: *Variable*=**OnCondition**(*Bedingung;Label*)
Variable=**OnCondition Call**(*Bedingung;Label*)

Dieser Befehl dient der Regelung von Ausnahmesituationen. Hier wird festgelegt was zu tun ist, wenn eine *Cancel-*, *Error-*, *NotFound* oder benutzerdefinierte Bedingung auftritt. *OnCondition* verzweigt in diesem Fall zu dem angegebenen Label bzw. ruft die angegebene Routine auf. Ein Makro mit mehr als einer *OnCondition* bezieht sich immer auf den zuletzt ausgeführten *OnCondition*-Befehl, wenn eine *Cancel-*, *Error*, *NotFound-* oder benutzerdefinierte Bedingung auftritt. Diese Bedingungen beenden normalerweise ein Makro, wenn sie nicht durch *OnCondition* abgefangen werden.

Während der Makroausführung wird immer der jeweils zuletzt ausgeführte *OnCondition*-Befehl berücksichtigt. Stellen Sie darum sicher, daß beim Einfügen neuer Bedingungen auch zusätzliche *OnCondition*-Befehle eingefügt werden.

Eine benutzerdefinierte Bedingungsnummer muß größer oder gleich 100 sein. Sie wird definiert durch »UserDefinedCondition + Numerischer Wert«, z. B. UserDefinedCondition! + 1 entspricht 101. Verwenden Sie *Assert*, um die Definition vorzunehmen.

Bei der Verwendung von *OnCondition Call* wird die als Label angegebene Routine aufgerufen und ausgeführt. Danach wird die Makroausführung mit dem Befehl fortgesetzt, der *OnCondition Call* folgt.

Parameter

Variable Labelname des zuletzt ausgeführten *OnCondition*-Befehls. Wurde kein Label angegeben, wird "" zurückgegeben.

Bedingung Verknüpft eine Bedingung mit einem Label:

CancelCondition!

ErrorCondition!

NotFoundCondition!

ExitCondition!

UserDefinedCondition!

Label Geben Sie hier einen Labelnamen an, zu dem das Makro verzweigen soll, wenn eine der genannten Bedingungen auftritt. Die Verarbeitung wird bei diesem Label fortgesetzt. Hier müssen Sie nun entscheiden, was in diesem Fall getan werden muß. Sie können das Makro beenden oder an einer anderen Stelle innerhalb des Makros mit der Verarbeitung fortfahren (z. B. unter anderen Bedingungen).

Weitere Hinweise siehe: Assert, Condition, OnCondition Call.

6.88 OnDDEAdvice Call

Befehlsform: Ergebnis=**OnDDEAdvice Call**(*PgmKennz;Übergabe;Label*)

Dieser Befehl stellt eine Verknüpfung mit einem zuvor zugeordneten Programm her. Hier kann geprüft werden, ob eine über *DDERequest* erhaltene Variable verändert wurde. Wenn ja, wird die unter *Label* angegebene Unterroutine ausgeführt.

Parameter

Ergebnis Durch diese Variable wird festgestellt, ob über *OnDDEAdvice Call* eine erfolgreiche Verbindung hergestellt wurde. Das ist dann der Fall, wenn diese Variable 0 enthält.

PgmKennz Angabe der unter *DDEInitiate* und *DDERequest* festgelegten Verknüpfungs-Variablen.

Übergabe WordPerfect-Variable oder String, die/der den in dem verknüpften Programm verwendeten Variablennamen enthält. Da jedes Programm seine eigenen Variablen benutzt, schlagen Sie bitte in der zugehörigen Dokumentation des betreffenden Programms nach. Dieser Name muß mit der unter *DDERequest* angegebenen Variablen *Übergabe* übereinstimmen.

Label Name der auszuführenden Unterroutine, wenn in Verbindung mit *Ergebnis* erkannt wurde, daß eine Veränderung stattgefunden hat.

Beispiel: Ergebnis=**OnDDEAdvice**(PgmKenn;"R1C1";Fehler)

Weitere Hinweise siehe: DDEInitiate, DDEPoke, DDERequest, DDETerminate

6.89 OnError

Befehlsform: *RetLabel*=**OnError**(*Label*) oder **OnError**(*Label*)

Dieser Befehl dient der Regelung von Ausnahmesituationen. Der auf *OnError* folgende Label verweist auf eine Befehlsfolge, die regelt was zu tun ist, wenn ein Fehler bei der Makro-Ausführung, bei WordPerfect oder beim Rücksprung von DOS oder Windows auftrat. Gleiches gilt, wenn ein *Assert*(*ErrorCondition*!) oder ein *Return* (ErrorCondition!) von einer Unterroutine oder von einem mit *Run* aufgerufenen Makro gemeldet wird.

Das Makro *verzweigt* beim Auftreten eines Fehlers zu dem angegebenen Label. Der Befehl muß im Makro so plaziert und ausgeführt werden, daß er bereits vor einer evtl. auftretenden Fehlerbedingung ausgeführt wurde. Tritt ein Fehler auf bevor *OnError* oder *OnError Call* ausgeführt wurde, wird das Makro beendet. Sind in einem Makro mehrere *OnError*-Befehle definiert, tritt der Befehl in Kraft, der als letzter ausgeführt wurde. Stellen Sie darum sicher,

daß beim Einfügen neuer Bedingungen auch zusätzliche *OnError*-Befehle eingefügt werden. Möchten Sie für bestimmte Bereiche eines Makros die Fehlerbedingung ausschalten, verwenden Sie ab der gewünschten Stelle den Befehl *Error*(*Off!*). Mit *Error*(*On!*) können Sie die Fehlerbedingung jederzeit wieder einschalten.

Parameter

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

Label Geben Sie hier einen Labelnamen an, zu dem das Makro verzweigen soll, wenn eine Errorbedingung auftritt oder gesetzt wurde. Die Verarbeitung wird bei diesem Label fortgesetzt. Hier müssen Sie nun entscheiden, was in diesem Fall getan werden muß. Sie können das Makro beenden oder an einer anderen Stelle innerhalb des Makros mit der Verarbeitung fortfahren (z. B. unter anderen Bedingungen).

Beispiel: Siehe unter *Error*(..) und *Return*. Beachten Sie bitte auch das Beispiel unter *OnCancel*, und wenden Sie es sinngemäß an.

Weitere Hinweise siehe: Assert(ErrorCondition!), Error(On!/Off!), OnError Call, Return

6.90 OnError Call

Befehlsform: *RetLabel*=**OnError Call**(*Label*) oder **OnError Call**(*Label*)

Dieser Befehl entspricht weitgehend dem Befehl *OnError*. Er dient der Regelung von Ausnahmesituationen. Der auf *OnError Call* folgende Label verweist auf eine Unterroutine, die aufgerufen wird, wenn ein Fehler bei der Makro-Ausführung, bei WordPerfect oder beim Rücksprung von DOS oder Windows auftrat. Die Unterroutine muß mit *Return* enden. Die Verarbeitung wird danach an der Stelle innerhalb des Makros fortgeführt, an der der Fehler auftrat. Gleiches gilt, wenn ein *Return*(ErrorCondition!) von einer Unterroutine oder von einem mit *Run* aufgerufenen Makro gemeldet wird.

Parameter

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

Label Geben Sie hier den Namen einer Unterroutine an, die aufgerufen werden soll, wenn eine Fehlerbedingung auftritt oder gesetzt wurde. Die Verarbeitung wird bei diesem Label fortgesetzt. Die Unterroutine muß mit einem *Return* enden. Danach wird das Makro an der Stelle fortgesetzt, an der der Fehler auftrat.

Beispiel: Siehe unter *Error* und *Return*. Beachten Sie bitte auch das Beispiel unter *OnCancel*, und wenden Sie es sinngemäß an.

Weitere Hinweise siehe: Assert(ErrorCondition!), Error(On!/Off!), OnError, Return

6.91 OnExit

Befehlsform: *RetLabel*=**OnExit**(*Label*)

Mit OnExit wird ein Label oder eine Routine festgelegt, zu der verzweigt bzw. die ausgeführt werden soll, wenn eine Ende-Bedingung auftritt. Der Label/dieRoutine werden ausgeführt, wenn folgendes eintrifft:

Quit Makroende (logisches Ende).

Return Rücksprung von einem Hauptmakro (das zuerst gestartete Makro).

Makroende Alle Befehle wurden ausgeführt (Makro ohne Quit-Befehl = physisches Ende).

ExitCondition! In einer Routine wurde z. B. über Assert die ExitCondition!-Bedingung gesetzt.

Parameter

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

Label Label, zu dem verzweigt werden soll, bzw. Routine, die auszuführen ist.

Weitere Hinweise siehe: Assert(ExitCondition!), Return

6.92 OnNotFound

Befehlsform: *RetLabel*=**OnNotFound**(*Label*) oder **OnNotFound**(*Label*)

Dieser Befehl dient der Regelung von Ausnahmesituationen. Hier wird festgelegt, was zu tun ist, wenn bei einem Suchvorgang »Nicht gefunden« auftritt oder wenn *Assert*(*NotFoundCondition!*) bzw. Return(NotFoundCondition!) ausgeführt wurden. Das Makro verzweigt in diesen Fällen zu dem angegebenen Label. Der Befehl muß im Makro so plaziert werden, daß er bereits vor einer auftretenden »Nicht gefunden«-Bedingung ausgeführt wurde. Tritt diese Bedingung auf bevor *OnNotFound* ausgeführt wurde, wird das Makro beendet. Sind in einem Makro mehrere *OnNotFound*-Befehle definiert, tritt der Befehl in Kraft, der als letzter ausgeführt wurde.

Während der Makroausführung wird immer der jeweils zuletzt ausgeführte *OnNotFound*-Befehl berücksichtigt. Stellen Sie darum sicher, daß beim Einfügen neuer Bedingungen auch zusätzliche *OnNotFound*-Befehle eingefügt werden (wenn erforderlich).

Parameter

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

Label Geben Sie hier einen Labelnamen an, zu dem das Makro verzweigen soll, wenn eine Nicht-Gefunden-Bedingung auftritt oder gesetzt wurde. Die Verarbeitung wird bei diesem Label fortgesetzt. Hier müssen Sie nun entscheiden, was in diesem Fall getan werden muß. Sie können das Makro beenden oder an einer anderen Stelle innerhalb des Makros mit der Verarbeitung fortfahren (z. B. unter anderen Bedingungen).

Beispiel

```
Label                (Fußnote)
Call                 (Suchtext)
For                  (Zähler;1;Zähler<9999;Zähler+1)
  OnNotFound         (Ende)
  Notennummer=       NumStr(Zähler)
  FootnoteCreate     (FootnoteNewNumber:Notennummer)
  OnNotFound Call    (NichtGefNoten)
  SearchString       (Suchtext)

  SuchenVorwärts     (SuchModus:Erweitert!)
  ::::::
EndFor
Go                   (Anfang)

Label                (NichtGefNoten)
```

```
Prompt              ("Suchen in Fuß-/Endnoten";"Begriff wurde in dieser
                    Fuß-/Endnote nicht gefunden, bitte [Makro,Pause]
                    wählen zum Weitersuchen.";;;)
Wait                (30)
EndPrompt
Return

Label               (Ende)
Quit
```

In diesem Beispiel (Ausschnitt aus einem Makro) werden alle Fußnoten eines Dokuments nach einem zuvor eingegebenen Text durchsucht. In dieser *For*-Schleife sind zwei Befehle *OnNotFound*.. vorhanden:

OnNotFound Dieser Befehl beendet die *For*-Schleife, wenn keine weiteren Fußnoten mehr vorhanden sind.

OnNotFound Call Dieser Befehle wird ausgeführt, wenn in der aktuellen Fußnote der eingegebene Suchtext nicht vorhanden ist. Es wird eine Unterroutine zur Anzeige einer Fehlermeldung aufgerufen und anschließend die *For*-Schleife fortgesetzt.

Beachten Sie bitte auch die Beispiele unter *Return* bzw. unter *OnCancel*, und wenden Sie sie sinngemäß an.

Weitere Hinweise siehe: Assert(NotFoundCondition!), OnNotFound Call, Return.

6.93 OnNotFound Call

Befehlsform: *RetLabel*=**OnNotFoundCall**(*Label*) oder **OnNotFound Call**(*Label*)

Dieser Befehl entspricht weitgehend dem Befehl *OnNotFound*. Der einzige Unterschied: Bei *OnNotFound Call* wird eine Unterroutine aufgerufen, die durch *Return* beendet werden muß. Nach der Ausführung dieser Unterroutine wird die Verarbeitung an der Stelle fortgesetzt, an der die Bedingung »Nicht gefunden« erzeugt wurde. Das Makro wird beendet, wenn beim Auftreten von »Nicht gefunden« kein *OnNotFound Call*- oder *OnNotFound*-Befehl definiert wurde.

Parameter

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

Label Geben Sie hier den Namen einer Unterroutine an, die aufgerufen werden soll, wenn eine Nicht-Gefunden-Bedingung auftritt oder gesetzt

wurde. Die Verarbeitung wird bei diesem Label fortgesetzt. Die Unterroutine muß mit einem *Return* enden. Danach wird das Makro an der Stelle fortgesetzt, an der der Fehler auftrat.

Beispiel: Siehe unter *OnNotFound* bzw. unter OnCancel«, und wenden Sie sie sinngemäß an.

Weitere Hinweise siehe: Assert(NotFoundCondition!), Label, OnNotFound

6.94 Pause

Befehlsform: Pause

Dieser Befehl bewirkt eine vorübergehende Unterbrechung der Makro-Ausführung, um z. B. in WordPerfect außerhalb der Makro-Kontrolle zu arbeiten. Der Aufruf muß bei der Makroerstellung (Aufzeichnung der Tastaturanschläge) über den Menüpunkt [*Tools, Makro, Pause*] erfolgen. Das Makro wird so lange unterbrochen, bis die Pause über die Auswahl desselben Menüpunkts wieder beendet wird.

Möchten Sie die Pause durch Drücken einer bestimmten Taste beenden, müssen Sie mit dem produktspezifischen Befehl *PauseTaste* arbeiten (siehe Kapitel »8 Ausgewählte produktspezifische Befehle«). Wenn Sie ein neues Makro im Dokumentfenster erstellen, können Sie diesen Befehl manuell eingeben oder über den Makrobefehlsmanager auswählen. Er kann nicht während der Makroaufzeichnung eingefügt werden.

Durch Anklicken von [*Tools, Makro*] können Sie kontrollieren, ob sich das Makro im Pausestatus befindet. Wenn dies der Fall ist, befindet sich vor dem Menüpunkt *Pause* ein Häkchen. Zusätzlich können Sie Pausen auch über die folgenden produktspezifischen Befehle steuern, die allerdings nicht während der Makroaufzeichnung zur Verfügung stehen (müssen nachträglich ins Makro eingefügt werden):

PauseDefinieren Angabe eines Makrobefehls, bei dessen Ausführung die Pause beendet wird.

PauseBefehl Angabe eines Makrobefehls, bei dessen Ausführung eine Pause beginnt.

Weitere Hinweise siehe: Kapitel »8.7 PauseCommand«, »8.8 PauseSet« und »8.9 KauseKey«

6.95 Persist/PersistAll

Befehlsform: **Persist**(*Variable1;Variable2...*)
PersistAll

In einem Makro sind drei Variablen-Tabellen möglich, die lokale, globale und Persist-Variable verwalten können. Variable, die in Verbindung mit *Persist/PersistAll* definiert wurden, ste-

hen makroübergreifend zur Verfügung. Hierdurch können Variable von einem Makro an über *Chain* oder *Run* aufgerufene Makros übergeben werden. In dem aufgerufenen Makro muß der Befehl *VarErrChk* verwendet werden, weil beim Kompilieren des aufgerufenen Makros die Variablen noch nicht zur Verfügung stehen (es sei denn, das aufrufende Makro wurde zuvor bereits ausgeführt). Über diese beiden Befehle können, im Gegensatz zu dem Befehl *Global*, auch Variable für Mischvorgänge übergeben werden. Über *Persist* können nur die jeweils genannten Variablen festgelegt werden, wogegen durch *PersistAll* alle Variablen des aktuellen Makros in die Persist-Tabelle übernommen werden.

Parameter

Variable Beliebige Variable, die an andere Makros oder an Mischvorgänge übergeben werden sollen.

Beispiel

```
Application          (A1; "WordPerfect"; Default; "DE")
VarErrChk            (Off!)
OnCancel             (Abbruch)
```

PersistAll

```
Label                (Anfang)
GetString            (Text;"Bitte beliebigen Text eingeben:";"Befehl  ""PERSIST""  testen")
Run                  ("PERSTEXT.WCM")
Prompt               ("Befehl  ""PERSIST""  testen";Drucken;;;)
  Wait               (50)
EndPrompt
Go                   (Anfang)

Label                (Abbruch)
Quit
```

PERSTEXT.WCM

```
Application          (A1; "WordPerfect"; Default; "DE")
//                   Über die Variable TEXT wird ein beliebiger Text übergeben.
VarErrChk            (Off!)
PersistAll
Drucken=             ("Folgender Text wurde eingegeben: " + Text)
//                   Der zusammengesetzte Text wird im aufrufenden Makro angezeigt.
Return
```

In dem ersten Makro werden über *PersistAll* alle Variablen als Persist-Variable definiert. Dadurch können Sie in dem aufgerufenen Makro PERSTEXT.WCM weiterverarbeitet werden. In dem aufgerufen Makro ist der Befehl *VarErrChk* erforderlich, da beim Kompilieren dieses Makros die im Prompt-Befehl angesprochenen Variablen noch nicht zur Verfügung stehen (ansonsten erfolgt eine Fehlermeldung). Wird von einem Makro ein Mischvorgang gestartet, können die Variablen auch dort verwendet werden.

Weitere Hinweise siehe: Assign, Discard, Global, Local, Persistall

6.96 Procedure

Befehlsform: **Procedure** *Name* (*Parameter;Parameter;...*)

Über diesen Befehl wird der Beginn einer Befehlsfolge (Unterroutine) definiert, die bei Bedarf aufgerufen werden kann. Der Aufruf ist von jeder beliebigen Stelle des aktuellen Makros aus möglich. In Verbindung mit *Use* können auch *Procedure*-Routinen anderer Makros aufgerufen werden. Der Unterschied zu anderen aufrufenden Befehlen besteht darin, daß entweder eine oder mehrere Variable an die Unterroutine weitergegeben werden können.

Die Makro-Ausführung wird an der aufrufenden Stelle unterbrochen, um die unter *Name* angegebene Unterroutine auszuführen. Die Unterroutine muß mit *EndFunc* oder mit *Return* enden. Danach wird mit dem Befehl fortgefahren, der dem aufrufenden Befehl folgt. *Procedure*-Routinen werden im Gegensatz zu *Label*-Routinen nur dann ausgeführt, wenn sie aufgerufen wurden (wird eine solche Routine z. B. direkt am Makroanfang definiert, wird sie trotzdem erst dann ausgeführt, wenn sie aufgerufen wird). Um den Wert der übergebenen Variablen zu ändern, können Sie der betreffenden Variablen des aufrufenden Befehls und der zugehörigen Variablen in dem *Procedure*-Befehl ein **&** voranstellen. In diesem Fall wird die Adresse (Speicheradresse) der Variablen übergeben und nicht deren Inhalt.

Parameter

Name Der Name ist frei wählbar, er muß jedoch mit einem Buchstaben beginnen. Verwenden Sie aussagefähige Namen, die auf die Funktion der Routine Bezug nehmen. Wird in der Routine eine Zinsberechnung durchgeführt, könnte man die Routine z. B. *Zinsen* nennen. Verwenden Sie keine zu langen Bezeichnungen (max. 30 Zeichen), denn diese Namen müssen Sie ggf. mehrmals verwenden (wenn eine Routine oft aufgerufen wird), wobei das Eintippen dann mühselig sein kann (Fehlerquelle!). Enthält ein Makro mehrere *Procedure*-Routinen, müssen diese unterschiedliche Namen aufweisen. Kein Name darf doppelt vergeben werden, ansonsten erfolgt bei der Kompilierung eine Fehlermeldung.

Parameter Dieser Variablen wird von dem aufrufenden Befehl ein Wert übergeben, mit dem in der Procedure-Routine weitergearbeitet werden kann. Mehrere Variable sind jeweils durch ein Semikolon zu trennen. Wird der Variablen ein & vorangestellt, muß auch der zugehörigen Variablen unter *Procedure* ein & vorangestellt werden.

Beispiel

```
1.      Feld1=          10
        Aufruf          (Feld1)                 //(aufrufender Befehl)
        Type            (Feld1)
        Type            (Feld2)
        Procedure       Aufruf(Feld2)
        Feld2=          Feld2 + 20
        EndProc
```

Beim Aufruf der *Procedure*-Routine wird der Inhalt von *Feld1* (= 10) übergeben und in *Feld2* gespeichert. Innerhalb der Routine wird zu *Feld2* der Wert 20 addiert, so daß danach in *Feld2* der Wert 30 vorhanden ist. Nach der Ausführung von *EndProc* enthält *Feld1* weiterhin den Wert 10.

```
2.      Feld1=          10
        Aufruf          (&Feld1)                    //(aufrufender Befehl)
        Type            (Feld1)
        Type            (Feld2)
        Procedure       Aufruf(&Feld2)
        Feld2=          Feld2 + 20
        EndProc
```

Beim Aufruf der *Procedure*-Routine wird der Inhalt von *Feld1* (= 10) übergeben und in *Feld2* gespeichert. Innerhalb der Routine wird zu *Feld2* der Wert 20 addiert, so daß danach in *Feld2* der Wert 30 vorhanden ist. Nach der Ausführung von *EndProc* enthält *Feld1* den in der *Procedure*-Routine ermittelten Wert, der über *Feld2* zurückgegeben wird.

Weitere Hinweise siehe: Call, Function, Label, Return, Use

6.97 Procedure Prototype

Befehlsform: **Procedure Prototype** *Name* (*Parameter*;*Parameter*;...)

Um Makros ausführen zu können, müssen diese unter einem eindeutigen Namen gespeichert und kompiliert werden. Beim Kompilieren werden nur formale aber keine logischen Fehler erkannt. So kann es durchaus vorkommen, daß ein mit *Run* oder *Use* aufgerufenes Makro fehlerfrei kompiliert wurde, die Ausführung jedoch einen logischen Fehler erzeugt. Um diese Run-Time-Fehler frühzeitig zu entdecken bzw. diese nach Möglichkeit auszuschließen, kann über *Procedure Prototype* die Syntax einer *Procedure*-Routine überprüft werden. Verwenden Sie diesen Befehl in dem betreffenden Hauptmakro, um bereits zur Compile-Zeit Fehler zu ermitteln.

Parameter

Name Name der zu testenden *Procedure*-Routine, der mit einem Buchstaben beginnen muß.

Parameter Verwenden Sie hier dieselben Parameter, die auch bei der zu testenden *Procedure*-Routine verwendet werden. Mehrere Parameter sind jeweils durch ein Semikolon zu trennen. Wird ein Parameter mit einem Ampersand verwendet (z. B. &Feld1) muß auch die zugehörige Variable im *Function*-Befehl ein Ampersand enthalten.

Beispiel

```
Application     (A1;"WordPerfect";Default;"DE")
Procedure       Prototype Prüfen(Eingabe)
Repeat
```

```
GetString           (Laufwerk;Length=1;"Bitte Laufwerk eingeben";"Procedure Prototype
                     testen")
Call                Prüfen(Laufwerk)
Until               ((Laufwerk = "c") or (Laufwerk = "d"))
: : : :

Procedure           Prüfen(Eingabe)
If                  ((ToLower(Eingabe) = "c") or (ToLower(Eingabe) = "d"))
                    MessageBox (MVar;"Hinweis"; "Laufwerk OK")
     Else
                    MessageBox (MVar;"Hinweis"; "Falsches Laufwerk")
EndIf
EndProc
```

In diesem Beispiel wird über *GetString* die Eingabe eines Laufwerkbuchstabens verlangt. Nur die Buchstaben **c** und **d** sind zugelassen. Das eingegebene Zeichen wird der Procedure *Prüfen* übergeben. Diese Procedure kann auch in einem anderen Makro vorhanden sein. Die Eingabe wird so lange wiederholt, bis ein richtiges Laufwerk eingegeben wurde.

Zwischen *Procedure* und dem nachfolgenden Labelnamen darf zum Einrücken nicht F7 verwendet werden, sonst werden Compile-Fehler angezeigt, obwohl die Syntax stimmt. Verwenden Sie zum Einrücken ⇆ oder Leerstellen.

Weitere Hinweise siehe: Function, Procedure, Use

6.98 Prompt

Befehlsform: **Prompt**(*Boxentitel;Meldung;IconNr;HorizPosition;VertikPosi- tion*)

Der unter *Meldung* definierte Text (Konstante oder Inhalt einer Variablen) wird in einem Dialogfeld mit der unter *Boxentitel* eingegebenen Überschrift angezeigt. Das Dialogfeld erscheint nur ganz kurz am Bildschirm und wird sofort wieder ausgeblendet. Verwenden Sie unmittelbar hinter dem Prompt-Befehl die Befehle *Pause* oder *Wait*, damit die Meldung am Bildschirm stehen bleibt. Die Anzeige müssen Sie durch Klicken auf [*OK*] oder [*Abbrechen*] beenden. Standardmäßig wird das Dialogfeld in der Bildschirmmitte eingeblendet. Wenn Sie eine andere Position wünschen, können Sie diese in den vorgesehenen Variablen eingeben. Bei Bedarf können Sie jedes Dialogfeld mit einem der vier zur Verfügung stehenden Icons versehen (siehe unten). Zu einer Zeit kann immer nur eine Meldung am Bildschirm angezeigt werden. Wird trotzdem ein weiterer Prompt-Befehl ausgeführt, wird dadurch die vorherige Meldung überschrieben.

Parameter

Boxentitel Text, der in der Titelleiste des Prompt-Dialogfeldes angezeigt werden soll. Der Text kann aus einer Konstanten bestehen oder in einer Variablen enthalten sein.

Meldung Ein String, der in dem Dialogfeld z. B. als Bedienungshinweis oder als Ergebnis einer Verarbeitung erscheinen soll. Hier können Sie eine Textkonstante eingeben. Diese muß in Hochkommata eingeschlossen sein. Sind innerhalb der Textkonstanten Hochkommata erforderlich, müssen Sie diese jeweils doppelt angeben (Beispiel siehe unten). Wird keine Meldung angegeben, wird der unter *Boxentitel* definierte Text übernommen. Unter *Boxentitel* wird dafür *PerfectScript* angezeigt.

Sie können auch den Inhalt einer Variablen anzeigen, wobei dieser Inhalt möglicherweise zuvor mit *Assign* aus mehreren Einzelinformationen zusammengesetzt worden sein kann. Variable mit numerischem Inhalt können ohne vorherige Konvertierung (über *NumStr*) angezeigt werden.

IconNr Anzeigen eines Icons im linken Teil der Box. Zum Kombinieren mehrerer Icons verwenden Sie | zwischen den Parametern (nicht alle Parameter können kombiniert werden).

NoIcon!	Kein Icon anzeigen.
StopSign!	Stop-Zeichen.
QuestionMark!	Fragezeichen.
ExclamationPoint!	Ausrufezeichen.
InformationIcon!	I-Zeichen.
Pause!	Anzeigen, bis eine Taste gedrückt wird.
Beep!	Akustisches Signal.
NoButtons!	Prompt wird ohne OK- und ohne Abbrechen-Taste angezeigt. Verwenden Sie EndPrompt zum Ausblenden der Meldung.

HorizPosition Geben Sie bei diesem Parameter die horizontale Position ein, an der das Dialogfeld erscheinen soll. Die Eingabe ist in Pixel (Bildschirmpunkten) vorzunehmen, gerechnet ab dem linken Rand des Anwendungsfensters. Die Angabe ist zusätzlich von der Bildschirmauflösung abhängig.

VertikPosition Geben Sie bei diesem Parameter die vertikale Position ein, an der das Dialogfeld erscheinen soll. Die Eingabe ist in Pixel (Bildschirmpunkten) vorzunehmen, gerechnet ab dem oberen Rand des Anwendungsfensters. Diese Angabe ist zusätzlich von der Bildschirmauflösung abhängig.

Wird auf die beiden letzten Parameter verzichtet, wird das Dialogfeld in der Fenstermitte plaziert.

Wichtig: Für jeden zwischendrin weggelassenen Parameter müssen Sie ein Semikolon setzen, sofern anschließend weitere Parameter folgen.

Wenn Sie [*Abbrechen*] wählen, verhält sich das Makro wie bei einem Abbruch. Wurde zuvor *OnCancel* oder *OnCancel Call* ausgeführt, wird bei dem angegebenen Label weitergearbeitet. Wurde einer der beiden Befehle nicht ausgeführt, wird das Makro beendet.

Beispiel

```
1       Prompt          ("Achtung!";"Wollen Sie diese Datei speichern?";2;;)

                        Beide Meldungen werden als Textkonstanten eingegeben.
                        Zusätzlich wird über den Parameter QuestionMark! ein Icon
                        eingeblendet.

2       Meldung1=       ("Achtung!")
        Meldung2=       ("Wollen Sie diese Datei speichern?")
        Prompt          (Meldung1;Meldung2;QuestionMark!)

                        Beide Meldungen werden jeweils einer Variablen entnommen.
                        Zusätzlich wird über den Parameter QuestionMark! ein Icon
                        eingeblendet.

3       Prompt          (Meldung1;Meldung2;QuestionMark!;100;200)

                        Beide Meldungen werden jeweils einer Variablen entnommen. Das
                        Dialogfeld wird 100 Punkte vom linken Rand und 200 Punkte vom
                        oberen Rand des aktiven Anwendungsfensters plaziert.

4       Prompt          ("T e s t";"Befehl  ""STRLEN""  testen";;;)

                        Bei der Verwendung von Hochkommata innerhalb einer
                        Textkonstanten müssen Sie diese jeweils doppelt angeben.

5       Prompt          "Zinsen berechnen";NumStr(Ergebnis);;;)

                        Der Inhalt der numerischen Variablen Ergebnis wird vor dem
                        Anzeigen in einen String konvertiert.
```

Weitere Hinweise siehe: EndPrompt, Pause, Wait

6.99 Quit

Befehlsform: Quit

Dient zum Beenden eines Makros. Auch verschachtelte oder verkettete Makros werden beendet. Wird der Befehl *Quit* z. B. innerhalb eines mit *Run* aufgerufenen Makros ausgeführt, wird das aufgerufene sowie das aufrufende Makro beendet. Auch Makros, die mit *Chain* noch auszuführen wären, werden nicht gestartet. Den Befehl können Sie z. B. auch innerhalb von *If*-Strukturen verwenden, um bei bestimmten Bedingungen das Makro zu beenden. Verwenden Sie *Quit* niemals zum Beenden von Unterroutinen, die mit *Call* oder *Case Call* aufgerufen werden (es sei denn, es ist gewollt), sonst erfolgt kein Rücksprung zum aufrufenden Befehl. Auch hier wird das Makro beendet. Die Position dieses Befehls innerhalb eines Makros ist entscheidend für den weiteren Makroablauf. Beachten Sie hierzu bitte die Beispiele.

Beispiel

```
1       Zähler=             0
        Label               (Anfang)
        IF                  (Zähler>10)
           Quit
        Else
           Zähler=          (Zähler+1)
           Zinsen=          (Kapital*0,14)
           KapitalNeu=      (Kapital+Zinsen)
        EndIf
        Go                  (Anfang)
```

Befehle, die hinter *Quit* definiert wurden, werden nach der Ausführung von *Quit* nicht mehr berücksichtigt. In diesem Fall wird die Schleife beendet, wenn der Zähler größer als zehn ist.

```
2       Zähler=             0
        Label               (Anfang)
        Prompt              ::::::
        Quit
        Zähler=             (Zähler+1)
        Zinsen=             (Kapital*0,14)
        KapitalNeu=         (Kapital+Zinsen)
        Go                  (Anfang)
```

Die Befehle hinter *Quit* werden in diesem Fall nicht mehr ausgeführt, da *Quit* an der falschen Stelle verwendet wird.

```
3       Zähler=             0
        Repeat
           Call             (Rechnen)
        Until               (Zähler>10)
        Quit

        Label               (Rechnen)
        Zähler=             (Zähler+1)
        Zinsen=             (Kapital*0,14)
        KapitalNeu=         (Betrag+Zinsen)
        Return
```

Obwohl hier weitere Befehle hinter *Quit* folgen (= Unterroutine), werden diese so oft aufgerufen, bis die *Repeat*-Schleife abgearbeitet wurde. Wäre hier kein *Quit* vorhanden, würde nach dem Beenden der Schleife das Makro automatisch bei dem Label (Rechnen) weiterarbeiten, d. h. alle nachfolgenden Befehle werden nochmals durchlaufen, was zu Fehlern führen würde.

Weitere Hinweise siehe: Return

6.100 Region-Befehle

Dialogfelder müssen vor der Ausführung eines Makros entweder über *DialogAdd...*-Befehle oder über den Makro-Dialogeditor erstellt werden. Die dabei festgelegten Elemente wie z. B. Tasten, Kontrollfelder, Optionsfelder, Einträge von Listenfeldern usw. sind in dem Dialogfeld fixiert und können normalerweise während des Makroablaufs nicht verändert werden. Mit den *Region*-Befehl können Sie jetzt auch während des Makroablaufs Elemente je nach Bedarf dynamisch ändern, so daß das Dialogfeld selbst und weitere Elemente ein anderes Layout bzw. andere Feldinhalte bekommen. Was wie gehandhabt wird, hängt von der Logik des Makros ab.

Bei der Beschreibung dieser Befehle werden Parameter, die bei allen Befehlen identisch sind, hier anschließend beschrieben.

NamedRgn Eingabe von *DialogId* und *ContrlId* zur Kennzeichnung des Dialogfelds und des betreffenden Dialogfeld-Eintrags. Die beiden Angaben müssen in Hochkommata eingeschlossen und durch einen Punkt getrennt werden. Wird keine *ControlID* angegeben, wird teilweise ein anderes Ergebnis geliefert. Beachten Sie bitte die Beschreibung bei den betreffenden Befehlen. Beachten Sie bzgl. von *DialogId* und *ControlId* die Hinweise in Kapitel »6.37 Dialog«. Wenn Sie Dialogfelder über den Dialogeditor erstellt haben, beachten Sie bitte die Erklärungen zu *Bereich* in den Kapiteln »11.1.1 Dialogfeld-Eigenschaften« und »11.6 Controls (Elemente)«.

RegionAddListItem

Befehlsform: **RegionAddListItem**(*NamedRgn*;*Eintrag*)

Hinzufügen eines neuen Eintrags zu einem Listenfeld, einem Kombinationsfeld oder einer Pop-Up-Schaltfläche.

Eintrag Neuer Listeneintrag, der einer Liste hinzuzufügen ist.

RegionEnableWindow

Befehlsform: **RegionEnableWindow**(*NamedRgn*;*Status*)

Die Bedienung mit der Maus und der Tastatur kann für eine bestimmte ControlID (Element) in Abhängigkeit der Variablen ***Status*** zugelassen oder gesperrt werden.

NamedRgn Wird keine ControlID angegeben, wirkt sich die Funktion auf das gesamte Dialogfeld aus.

Status Maus- und Tastatureingaben sperren oder zulassen:

0 Disable	Eingaben gesperrt.
1 Enable!	Eingabe erlaubt.

RegionGetCheck

Befehlsform: *MakroVariable*=**RegionGetCheck**(*NamedRgn*)

Der Befehl überprüft, ob ein Kontrollfeld aktiviert oder deaktiviert ist.

MakroVariable Nach der Ausführung des Befehls enthält diese Variable den aktuellen Status des unter ***NamedRgn*** angegebenen Kontrollfelds.

Up!
Down!
Checked!
Unchecked!
Indeterminate!

RegionGetHandle

Befehlsform: *Variable*=**RegionGetHandle**(*NamedRgn*)

Liefert die ausgeführte Windows-Funktion (= numerischer Wert) des genannten Bereichs.

Variable Enthält die ausgeführte Windows-Funktion des angegebenen Bereichs. Verwenden Sie OnError zum Abfangen von Fehlern, wenn *NamedRgn* nicht vorhanden ist.

RegionGetListCount

Befehlsform: *Variable*=**RegionGetListCount**(*NamedRgn*)

Ermitteln der Anzahl von Einträgen in einem Listen- oder Kombinations-Dialogfeld.

Variable Enthält die Anzahl der Einträge des angegebenen Bereichs.

RegionGetModified

Befehlsform: *Variable*=**RegionGetModified**(*NamedRgn*)

Prüft, ob der Inhalt eines Textfeldes modifiziert wurde (z. B. bei DialogAddCounter, DialogAddDate, DialogAddEditBox, DialogAddFilenameBox). Siehe auch *Region- SetModified*.

Variable Enthält *True*, eine Modifizierung fand statt und *False*, wenn nicht.

RegionGetSelectedText

Befehlsform: MakroVariable=;**RegionGetSelectedText**(*NamedRgn*)

Ausgewählter Text aus dem unter ***NamedRgn*** angegebenen Element (z. B. Listenfeld, Kombinationsfeld, Zahlenfeld usw.) wird in ***MakroVariable*** übernommen.

MakroVariable Nach der Ausführung des Befehls enthält diese Variable den ausgewählten Text.

RegionGetWindowText

Befehlsform: MakroVariable=**RegionGetWindowText**(*NamedRgn*)

Ermitteln des Textes der Titelzeile oder des angegebenen Bereichs. Wird kein Bereich angegeben, wird der Text der Titelzeile übertragen.

MakroVariable Nach der Ausführung des Befehls enthält diese Variable den ausgewählten Text.

RegionIsVisible

Befehlsform: *Variable*=**RegionIsVisible**(*NamedRgn*)

Legt den Anzeigestatus eines Bereichs oder eines Fensters fest.

Variable Es wird ein Wert ungleich Null zurückgegeben, wenn das Fenster sichtbar ist oder 0, wenn es nicht sichtbar ist. Das Windows-Flag WS_VISIBLE legt den Wert der Variablen fest. Weitere Informationen entnehmen Sie bitte dem Windows-Programmierhandbuch.

RegionMoveWindow

Befehlsform: **RegionMoveWindow**(*NamedRgn*;*Links*;*Oben*;*Breite*;*Höhe*)

Verschieben oder Ändern der Größe eines Dialogfelds oder eines einzelnen Elements. Wird unter *NamedRgn* keine *ControlID* angegeben, bezieht sich das Verschieben oder Ändern der Größe auf das gesamte Dialogfeld. Die Angaben erfolgen in Dialogfeldeinheiten

Links Anzahl der Dialogfeldeinheiten vom linken Bildschirmrand bis zum linken Rand des Dialogfelds oder vom linken Rand des Dialogfelds zum linken Rand des unter *ControlID* genannten Elements.

Oben Anzahl der Dialogfeldeinheiten vom oberen Bildschirmrand bis zum oberen Rand des Dialogfelds oder vom oberen Rand des Dialogfelds bis zum oberen Rand der unter *ControlID* genannten Elements.

Breite Breite des Dialogfelds oder des unter *ControlID* genannten Elements.

Höhe Höhe des Dialogfelds oder des unter *ControlID* genannten Elements.

RegionRemoveListItem

Befehlsform: **RegionRemoveListItem**(*NamedRgn*;*Eintrag*)

Löschen eines Eintrags eines Listen-/Kombinationsfelds oder einer Pop-Up-Schaltfläche.

Eintrag Listeneintrag, der gelöscht werden soll.

RegionResetList

Befehlsform: **RegionResetList**(*NamedRgn*)

Löschen einer kompletten Liste eines Listenfelds, eines Kombinationsfelds oder einer Pop-Up-Schaltfläche.

NamedRgn Angabe der DialogID und der ControlID, die die zu löschende Liste enthält.

RegionSelectListItem

Befehlsform: **RegionSelectListItem**(*NamedRgn;Eintrag*)

Auswählen eines Listeneintrags eines Listen-/Kombinationsfelds oder einer Pop-Up-Schaltfläche.

Eintrag Listeneintrag, der ausgewählt werden soll.

RegionSetBitmapFilename

Befehlsform: **RegionSetBitmapFilename**(*NamedRgn;Dateiname*)

Lädt eine neue Bitmap-Datei in ein Bitmap-Element.

Dateiname Voller Pfadname der zu ladenden Bitmap-Datei.

Ein Beispiel finden Sie unter DialogAddBitmap.

RegionSetCheck

Befehlsform: **RegionSetCheck**(*NamedRgn;Status*)

Der Befehl aktiviert oder deaktiviert das unter ***NamedRgn*** angegebene Kontrollfeld.

Status Aktivieren oder Deaktivieren des unter ***NamedRgn*** angegebenen Kontrollfelds.

Check!	Kontrollfeld wird aktiviert.
UnCheck!	Kontrollfeld wird deaktiviert
Gray!	Feld wird grau hinterlegt.

RegionSetFocus

Befehlsform: **RegionSetFocus**(*NamedRgn*)

Das unter *NamedRgn* angegebene Element wird aktiviert (ist eingabebereit).

RegionSetModified

Befehlsform: **RegionSetModified**(*NamedRgn;Status*)

Setzt den Modifizierungsstatus eines Editier-Elements (DialogAddCounter, DialogAddDate, DialogAddEditBox, DialogAddFilenameBox). Beachten Sie bitte auch RegionGetModified.

NamedRgn	Dialog ID und Control ID oder Dialog ID und Name des Bereichs. IDs müssen in Hochkommata angegeben und durch einen Punkt getrennt werden.	
Status	Modified!	(Änderung durchgeführt).
	NotModified!	(Keine Änderung durchgeführt).

RegionSetProgressPercent

Befehlsform: **RegionSetProgressPercent**(*NamedRgn;Prozent*)

Setzen eines Prozentwertes für ein DialogAddProgress-Element.

Prozent Anzuzeigender Prozentwert.

Ein Beispiel finden Sie unter DialogAddProgress.

RegionSetWindowText

Befehlsform: **RegionSetWindowText(***NamedRgn;Eintrag***)**

Ersetzen des Textes der Titelzeile oder der angegebenen ControlID. Wird unter *NamedRgn* keine ControlID angegeben, wird der Text der Titelzeile ersetzt.

Eintrag Angabe des Ersatztextes.

RegionShowWindow

Befehlsform: **RegionShowWindow**(*NamedRgn;Status*)

Anzeigen oder Ausblenden eines Dialogfelds oder eines Elements. Wird unter *NamedRgn* keine ControlID angegeben, wird das gesamte Dialogfeld angezeigt oder ausgeblendet.

Status	Dialogfeld bzw. Element anzeigen oder ausblenden.	
	Hide!	Dialogfeld bzw. Element ausblenden.
	Show!	Dialogfeld bzw. Element anzeigen.
	Weiterhin sind folgende Angaben möglich:	
	ShowMinimized!	ShowMaximized! Maximize!

ShowNoActivate!	Normal!	ShowNormal!
Minimize!	ShowMinNoActive!	ShowNA!
ShowRestore!	ShowDefault!	

6.101 Registry-Befehle

Über die Registry-Befehle können Windows-Registrations-Datenbanken (*.REG) bearbeitet werden. Die Bearbeitung sollten Sie nur dann vornehmen, wenn Sie sich mit der Handhabung dieser Datenbanken bestens auskennen, ansonsten können schwerwiegende Fehler auftreten, die z. B. das Ausführen einer Anwendung nicht mehr ermöglichen, so daß eine Neuinstallation dieser Anwendung erforderlich ist. Schlagen Sie bitte in jedem Fall in dem Windows-Programmier-Handbuch nach.

Da diese Befehle von den wenigsten Lesern benötigt werden, wird wegen begrenzten Platzes in diesem Buch auf eine detaillierte Beschreibung zugunsten wichtigerer Befehle verzichtet. Schlagen Sie wegen Einzelheiten bitte in der Makro-Online-Hilfe nach, wenn Sie diese Befehle trotzdem benötigen.

RegistryCloseKey	Schließen eines geöffneten Schlüssels.
RegistryCreateKey	Erstellen oder Öffnen eines Schlüssels.
RegistryDeleteKey	Löschen eines Schlüssels.
RegistryEnumKey	Spezifizieren eines Schlüssels.
RegistryEnumValue	Spezifizieren eines Wertes.
RegistryOpenKey	Öffnen eines Schlüssels.
RegistryQueryKeyCount	Ermittelt den Zähler eines untergeordneten Schlüssels.
RegistryQueryLastError	Ermittelt den Fehler eines untergeordneten Schlüssels.
RegistryQueryValue	Ermitteln eines Wertes.
RegistryQueryValueCount	Ermittelt den Zähler von Werten eines Schlüssels.
RegistrySetValue	Speichern eines Wertes.

6.102 Repeat

Befehlsform: Repeat

Dieser Befehl vereinfacht die Schleifenverarbeitung, indem er die nachfolgenden Befehle bis zum nächsten zugehörigen *Until*-Befehl so oft wiederholt, bis die unter *Until* angegebene Bedingung erfüllt ist, sofern sie nicht vorher durch andere Bedingungen innerhalb der Scheife

(z. B. If, Break) beendet wird Danach arbeitet das Makro mit dem Befehl weiter, der unmittelbar hinter *Until* folgt. Die Schachtelung mehrerer *Repeat*-Schleifen ist erlaubt, wobei jede *Repeat*-Schleife jeweils mit *Until* enden muß. Auch *For-*, *ForNext-* und *While*-Schleifen können innerhalb von *Repeat* geschachtelt werden. Die Bedingung zum Beenden der Schleife ist hier am Schleifenende definiert, wogegen sie bei *For*, *ForNext* oder *While* am Schleifenanfang steht. Sofern Sie mit Variablen arbeiten, achten Sie bitte darauf, daß diese vor der Ausführung des Befehls gültige Werte enthalten. Geben Sie diese entweder über die Tastatur ein, oder ordnen Sie diese vor der Ausführung von *Repeat* mit dem Befehl *Assign* zu.

Achtung! Falsche Variableninhalte können zu unvorhersehbaren Ergebnissen führen (z. B. Endlosschleife).

Jede *Repeat*-Struktur muß zwingend mit einem *Until*-Befehl enden. Zwischen *Repeat* und *Until* können Sie fast alle Makrobefehle verwenden.

Beispiel 1

Aufgrund der folgenden *Repeat*-Anweisung wird das kleine Einmaleins von 5 errechnet und das Ergebnis wie folgt ausgedruckt:

```
1       *       5       =       5
2       *       5       =       10
3       *       5       =       15
:
10      *       5       =       50
```

```
        FileNew                 ()
1       Multiplikator=          5
2       Zähler=                 1
3       Repeat                                          //Beginn der Schleife.
4          Ergebnis=            Zähler*5
5          Type                 NumStr(Zähler)
           Tab                  ()
           Type                 ("*")
           Tab                  ()
           Type                 NumStr(Multiplikator)
           Tab                  ()
           Type                 ("=")
           Tab                  ()
           Drucken=
           Type                 NumStr(Ergebnis)
6          HardReturn           ()
7          Zähler=              Zähler+1
8       Until                   (Zähler>10)             //Ende der Schleife.
        HardReturn              ()
9       Type                    ("Ende der Berechnung")
```

1 Der Variablen *Multiplikator* wird der Wert 5 zugewiesen. Wenn Sie hier eine andere Zahl angeben, wird das kleine Einmaleins dieser Zahl gedruckt, d. h. wenn Sie diese Variable mit 12 initialisieren, wird das Einmaleins von 12 errechnet.

2 Der Variablen *Zähler* wird der Wert 1 zugewiesen. Diese Variable wird bei der Berechnung als Multiplikant verwendet (1, 2, 3, ...10).

3 Beginn der *Repeat*-Schleife. Die Befehle zwischen *Repeat* und *Until* werden so oft wiederholt, bis die *Until*-Bedingung erfüllt ist.

4 Errechnen des Ergebnisses einer Zeile. Die Variable *Ergebnis* speichert das Ergebnis, *Zähler* dient hier als Multiplikant, die Variable *Multiplikator* als Multiplikator.

5 Drucken einer Zeile, nachdem das Ergebnis ermittelt wurde. Durch die Befehle *Type* wird fixer Text (* bzw. =) und die Variableninhalte auf jeweils einer Tabulatorposition gedruckt. Numerische Werte müssen vor dem Drucken in Strings (alphanumerische Werte) umgesetzt werden. Dies geschieht mit Hilfe des Befehls *NumStr*.

6 Erzeugen eines Zeilenumbruchs, damit die nächste Berechnung am Beginn einer neuen Zeile gedruckt wird.

7 Der Inhalt der Variablen *Zähler* wird um 1 erhöht.

8 Ende der *Repeat*-Schleife. Die Anweisungen zwischen *Repeat* und *Until* werden so oft wiederholt, bis der Inhalt der Variablen *Zähler* einen Wert enthält, der größer ist als 10.

9 Der Text »Ende der Berechnung« wird gedruckt, nachdem die *Repeat*-Schleife beendet wurde.

Wie Sie das Einmaleins einer beliebigen Zahl drucken können, finden Sie bei der Beschreibung des Befehls *While*.

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

Beispiel 2

```
Repeat
   ::::
   Repeat
      ::::
      Repeat
         ::::
      Until          (Anzahl=5)
   Until             ((SeitenNr=10) And (Var5>3))    //Und-Bedingung
Until                ((Var5>10) Or (Var6="Ende"))    //Oder-Bedingung
```

Geschachtelte *Repeat*-Schleifen. Jede Schleife muß jeweils mit einer *Until*-Bedingung enden.

Weitere Hinweise siehe: For, ForNext, Until, While

6.103 Return

Befehlsform:	**Return** **Return**(*Bedingung*) **Return**(*Variable*)

Die Wirkungsweise dieser Befehle hängt davon ab, ob vorher ein *Call*-, ein *Case Call*- oder ein *Run*-Befehl ausgeführt wurde:

Call Durch *Return* wird das Ende einer Unterroutine markiert, die durch *Call* aufgerufen wurde. Das Makro wird hierdurch veranlaßt, zu dem aufrufenden Befehl zurückzukehren. *Return* kann z. B. auch innerhalb der Unterroutine in einer *If*-Struktur ausgeführt werden, um die Unterroutine aufgrund einer zutreffenden Bedingung vorzeitig zu verlassen. Die Verarbeitung wird danach mit dem Befehl fortgesetzt, der dem *Call*-Befehl folgt.

Case Call Siehe sinngemäß unter *Call*.

Run Bei verschachtelten Makros (ein Makro ruft ein weiteres Makro über den Befehl *Run* auf) bewirkt dieser Befehl den Rücksprung zu dem Makro, von dem aus das aktuelle Makro aufgerufen wurde. Werden innerhalbeinesmit *Run* aufgerufenen Makros Unterroutinen mit *Case*-, oder *Case Call*-Befehle aufgerufen, kehrt auch hier das Makro durch *Return* wieder zu dem aufrufenden Befehl zurück. In diesem Fall muß durch eine von Ihnen gesetzte Bedingung zum Makroende oder zu einem *Return*-Befehl verzweigt werden, damit das aktuelle Makro beendet wird und die Rückkehr zum aufrufenden Makro erfolgt.

Wurde vor der Ausführung von *Return* kein *Run*, *Call*, oder *Case Call* ausgeführt, wird das aktuelle Makro beendet.

CancelCondition!

Die Wirkungsweise dieser Option hängt davon ab, ob vorher ein *Call*-, ein *Case Call*- oder ein *Run*-Befehl ausgeführt wurde:

Call Durch *Return*(*CancelCondition!*) wird das Ende einer Unterroutine markiert, die durch *Call* aufgerufen wurde. *Return*(*CancelCondition!*) kann z. B. auch innerhalb der Unterroutine in einer *If*-Struktur ausgeführt werden, um die Unterroutine aufgrund einer zutreffenden Bedingung vorzeitig zu verlassen. Das Makro wird hierdurch veranlaßt, zu dem aufrufenden Befehl zurückzukehren. An dieser Stelle verhält sich das Makro jetzt so, als wäre ein Abbruch erfolgt.

Wurde zuvor

Cancel(Off!) ausgeführt, wird mit dem Befehl fortgefahren, der dem *Call*-Befehl folgt.

Cancel(On!) ausgeführt, wird die Makroverarbeitung bei dem Label fortgeführt, der unter *OnCancel* angegeben wurde.

Case Call Siehe sinngemäß unter *Call*.

Run Bei verschachtelten Makros (ein Makro ruft ein weiteres Makro über den Befehl *Run* auf) bewirkt dieser Befehl den Rücksprung zu dem Makro, von dem aus das aktuelle Makro aufgerufen wurde. An dieser Stelle verhält sich das Makro jetzt so, als wäre ein Abbruch erfolgt (siehe oben). Werden innerhalb eines mit *Run* aufgerufenen Makros Unterroutinen mit *Case-*, oder *Case Call*-Befehle aufgerufen, kehrt auch hier das Makro durch *Return*(CancelCondition!) wieder zu dem aufrufenden Befehl zurück. In diesem Fall muß durch eine von Ihnen gesetzte Bedingung zum Makroende oder zu einem *Return*-Befehl verzweigt werden, damit das aktuelle Makro beendet wird, und die Rückkehr zum aufrufenden Makro erfolgt.

Wurde vor der Ausführung von *Return*(*CancelCondition!*) kein *Run*, *Call*, oder *Case Call* ausgeführt, wird das aktuelle Makro beendet.

ErrorCondition!

Die Wirkungsweise dieser Option hängt davon ab, ob vorher ein *Call-*, ein *Case Call-* oder ein *Run*-Befehl ausgeführt wurde:

Call Durch *Return*(*ErrorCondition!*) wird das Ende einer Unterroutine markiert, die durch *Call* aufgerufen wurde. *Return*(*ErrorCondition!*) kann z. B. auch innerhalb der Unterroutine in einer *If*-Struktur ausgeführt werden, um die Unterroutine aufgrund einer zutreffenden Bedingung vorzeitig zu verlassen. Das Makro wird hierdurch veranlaßt, zu dem aufrufenden Befehl zurückzukehren. An dieser Stelle verhält sich das Makro jetzt so, als wäre ein Fehler aufgetreten.

Wurde zuvor

Error(Off!) ausgeführt, wird mit dem Befehl fortgefahren, der dem *Call*-Befehl folgt.

Error(On!) ausgeführt, wird die Makroverarbeitung bei dem Label fortgeführt, der unter *OnError* angegeben wurde.

Case Call Siehe sinngemäß unter *Call*.

Run Bei verschachtelten Makros (ein Makro ruft ein weiteres Makro über den Befehl *Run* auf) bewirkt dieser Befehl den Rücksprung zu dem Makro, von dem aus das aktuelle Makro aufgerufen wurde. An dieser Stelle verhält sich das Makro jetzt so, als wäre ein Fehler aufgetreten (siehe oben). Werden innerhalb eines mit *Run* aufgerufenen Makros Unterroutinen mit *Case-*, oder *Case Call*-Befehle aufgerufen, kehrt auch hier das Makro durch *Return*(*ErrorCondition!*) wieder zu dem aufrufenden Be-

fehl zurück. In diesem Fall muß durch eine von Ihnen gesetzte Bedingung zum Makroende oder zu einem *Return*-Befehl verzweigt werden, damit das aktuelle Makro beendet wird und die Rückkehr zum aufrufenden Makro erfolgt.

Wurde vor der Ausführung von *Return*(ErrorCondition!) kein *Run*, *Call*, oder *Case Call* ausgeführt, wird das aktuelle Makro beendet.

NotFoundCondition!

Die Wirkungsweise dieser Option hängt davon ab, ob vorher ein *Call*-, ein *Case Call*- oder ein *Run*-Befehl ausgeführt wurde:

Call Durch *Return*(*NotFoundCondition!*) wird das Ende einer Unterroutine markiert, die durch *Call* aufgerufen wurde. *Return*(*NotFoundCondition!*) kann z. B. auch innerhalb der Unterroutine in einer *If*-Struktur ausgeführt werden, um die Unterroutine aufgrund einer zutreffenden Bedingung vorzeitig zu verlassen. Das Makro wird hierdurch veranlaßt, zu dem aufrufenden Befehl zurückzukehren, weil eine Suche nicht erfolgreich abgeschlossen wurde, d. h. ein String, ein Steuerzeichen oder eine Kombination aus beiden wurde nicht gefunden. Wurde zuvor *OnNotFound* ausgeführt, wird die Makroverarbeitung bei dem Label fortgeführt, der unter *OnNotFound* angegeben wurde. Wurde *OnNotFound* nicht verwendet, wird das Makro beendet, wenn eine Suche nicht erfolgreich war.

Case Call Siehe sinngemäß unter *Call*.

Run Bei verschachtelten Makros (ein Makro ruft ein weiteres Makro über den Befehl *Run* auf) bewirkt dieser Befehl den Rücksprung zu dem Makro, von dem aus das aktuelle Makro aufgerufen wurde. Hier gelten dann dieselben Regeln wie zuvor unter *Call* beschrieben.

Wurde vor der Ausführung von *Return*(NotFoundCondition!) kein *OnNotFound* ausgeführt, wird das aktuelle Makro beendet.

Function / Procedure

Rücksprung aus einer *Function*- oder *Procedure*-Routine. Bei Bedarf kann eine Variable an den aufrufenden Befehl zurückgegeben werden (z. B. *Return(Feld1)*). Weitere Informationen hierzu entnehmen Sie bitte der jeweiligen Befehlsbeschreibung.

Beispiel

```
Makro:              RETURN.WCM

Application         (A1;"WordPerfect";Default;"DE")
// Die nachfolgend genannten Bedingungen werden in dem aufgerufenen Makro RETURN1.WCM gesetzt.
OnCancel            (Abbruch)
OnError             (Fehler)
OnNotFound          (NichtGefunden)

Label               (Anfang)
```

```
Run                 ("C:\WPWIN7\MAKROS\RETURN1.WCM")
Go                  (Anfang)

Label               (Abbruch)
Prompt              ("Was wurde gewählt?";"Es wurde ""Abbruch"" gewählt.";;;)
Wait                (30)
EndPrompt
Go                  (Anfang)

Label               (Fehler)
Prompt              ("Was wurde gewählt?";"Es wurde ""Fehler"" gewählt.";;;)
Wait                (30)
EndPrompt
Go                  (Anfang)

Label               (NichtGefunden)
Prompt              ("Was wurde gewählt?";"Es wurde ""Nicht gefunden"" gewählt.";;;)
Wait                (30)
EndPrompt
Go                  (Anfang)
```

```
Makro:              RETURN1.WCM

Application         (A1;"WordPerfect";Default;"DE")
                    // Je nach Dateneingabe wird die betreffende Fehlerbedingung
                    // gesetzt und in das aufrufende Makro (RETURN.WCM)
                    // zurückgesprungen. Dort wird angezeigt, welche Bedingung
                    // gesetzt wurde.

Label               (Anfang)
Getstring           (Eingabe;"Bitte Daten eingeben (a=Abbruch, f=Fehler, n=Nicht
                                                                    gefunden):";
                    "Returnbefehl testen")
If                  (Eingabe = "a")
  Return            (CancelCondition!)Else
  If                (Eingabe = "f")
     Return         (ErrorCondition!)Else
     If             (Eingabe = "n")
        Return      (NotFoundCondition!)
    Else
       Prompt       ("Fehler";"Bitte a, f oder n eingeben.";;;)
       Wait         (30)
       EndPrompt
       Go           (Anfang)
EndIf EndIf EndIf
```

Über diese beiden Makros können Sie die Wirkung von *Return*(...Condition!) testen. Das Makro RETURN.WCM ruft das Makro RETURN1.WCM auf. Das aufgerufene Makro verlangt eine Dateneingabe und setzt eine entsprechende Bedingung. Durch das Ausführen des Befehls *Return*(...Condition!) wird die ausgewählte Bedingung gesetzt und wieder in das aufru-

fende Makro zurückgesprungen. Aufgrund der unter **ON...** vorgegebenen Labelnamen wird je nach Bedingung die zugehörige Unterroutine ausgeführt, die über *Prompt* die ausgewählte Bedingung anzeigt.

Beispiel: siehe sinngemäß unter Case, Case Call, Run.

Weitere Hinweise siehe: Call, Case Call, Function, OnCancel Call, OnError Call, OnNotFound Call, Procedure

6.104 Run / Nest

Befehlsform: **Run**(*Makroname*)
Nest(*Makroname*)

Das unter *Run* angegebene Makro wird unmittelbar ausgeführt. Sie können auch vollqualifizierte Namen verwenden, wie z. B. »C:\WPWIN7\BRIEFE\MAKROS\ BRIEFKOP.WCM«. Der Makroname muß den Windows-95-Konventionen zur Vergabe von Dateinamen entsprechen. Danach übernimmt das aufgerufene Makro die weitere Verarbeitung bzw. die Kontrolle. Für den Rücksprung in das aufrufende Makro muß in dem aufgerufenen Makro der Befehl *Return* ausgeführt werden (beachten Sie bitte auch den folgenden Absatz). Nach Beendigung des aufgerufenen Makros wird in dem aufrufenden Makro mit dem Befehl fortgefahren, der dem *Run*-Befehl folgt.

Nach dem Rücksprung in das aufrufende Makro wird die Kontrolle wieder von diesem übernommen. Wird in dem aufgerufenen Makro ein *Quit* ausgeführt, wird die komplette Makroverarbeitung abgebrochen Es erfolgt kein Rücksprung in das aufrufende Makro.

Der *Nest*-Befehl ist das WP-6.0-DOS-Äquivalent zu dem *Run*-Befehl. Er wurde aus Kompatibilitätsgründen in den Befehlsumfang für WPWin 7 aufgenommen. Die Handhabung und die Wirkungsweise sind identisch.

In dem aufgerufenen Makro können Sie vor dem Rücksprung zum aufrufenden Makro in Verbindung mit Return Bedingungen setzen, die bestimmte Aktionen simulieren, wie z. B.

Return(CancelCondition!) = Abbruchbedingung

Return(ErrorCondition!) = Fehlerbedingung

Return(NotFoundCondition!) = Nicht-Gefunden-Bedingung

um nach dem Rücksprung in das aufrufende Makro in Verbindung mit den Befehlen *OnCancel*, *OnError* und *OnNotFound* entsprechend reagieren zu können (siehe Beispiele unter *Return*).

Nur bereits kompilierte Makros können aufgerufen werden, d. h. das angegebene Makro muß existieren und muß kompiliert sein. Über die Befehle *Global*, *Persist* und *PersistAll* können Daten von einem Makro an ein anderes übergeben werden.

Parameter

Makroname Angabe des aufzurufenden Makros (Makronamens) in Verbindung mit Laufwerksbuchstabe und Ordnername. Vergessen Sie nicht die Dateinamen-Erweiterung. Hier kann ein String oder ein Variablen-Name angegeben werden.

Makros lassen sich über mehrere Ebenen verschachteln. Sie können darum in einem mit *Run* aufgerufenen Makro ein oder mehrere Makros aufrufen, die wiederum weitere Makroaufrufe enthalten (siehe auch unten). Der Rücksprung aus einem aufgerufenen Makro erfolgt immer in das aufrufende Makro.

Beispiel: Ein Hauptmakro ruft ein zweites Makro auf. Das zweite Makro ruft ein drittes Makro auf. Ist das dritte Makro beendet, erfolgt der Rücksprung zu dem zweiten (= aufrufenden) Makro, also nicht in das Hauptmakro! Erst wenn das zweite Makro beendet ist, erfolgt der Rücksprung zum Hauptmakro. Im Prinzip arbeitet dieser Befehl wie ein *Call*-Befehl. Der Unterschied besteht darin, daß beim *Call*-Befehl die auszuführenden Befehle innerhalb des aktuellen Makros definiert sein müssen, während beim *Run*-Befehl ein eigenständiges Makro ausgeführt wird.

Verwenden Sie diesen Befehl dann, wenn ein und dieselbe Befehlsfolge in unterschiedlichen Makros erforderlich ist, also mehrmals vorkommt. Anstelle vieler, gleichartiger Definitionen in unterschiedlichen Makros wird die mehrmals benötigte Makroroutine (= identische Befehlsfolge) einmal erstellt und als eigenes Makro gespeichert.

In dem aufgerufenen Makro können Sie (wenn erforderlich) Bedingungen setzen und aufgrund dieser das Makro vorzeitig in Verbindung mit einer Fehlerbedingung verlassen (= Abbruch):

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

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

Beispiel: *Siehe unter Return*

Weitere Hinweise siehe: Chain, Return..., On...

6.105 SendKeys

Befehlsform: **SendKeys**(*Tastenanschläge;Sprache*)

Über *SendKeys* können Sie Tastaturmakros erstellen, um die angegebenen Tastenanschläge oder die zu drückenden Funktionstasten in das aktive Fenster/die aktive Anwendung zu übergeben bzw. um diese auszuführen. Es werden die unter *Tastenanschläge* angegebenen Tastenanschläge wiederholt und anschließend das Makro mit dem nächsten Befehl fortgesetzt. Die auszuführenden Tastenanschläge werden bei der aktiven Anwendung so lange nicht ausgeführt, bis der nächste Befehl abgearbeitet wurde. Damit können Sie Makros erstellen, die Tastenanschläge in ein Dialogfeld innerhalb von WordPerfect für Windows übernehmen.

Beispiel

```
SendKeys          (»{Alt + D +Ö}«)
```

In diesem Beispiel wird das Dialogfeld [*Datei,Öffnen*] angezeigt.

Zwei verschiedene Sprachen werden unterstützt: Markup-Sprache und Keystring-Sprache (siehe unten).

Parameter

Tastenanschläge	Tastenanschläge, die auszuführen sind.	
Sprache	Geben Sie hier an, welche Sprache verwendet werden soll:	
	Old!	Alte MarkUp-Sprache.
	New!	Neue MarkUp-Sprache.

Zu sendende Tastenanschläge

Die folgenden Meta-Symbole können Sie zum Definieren von Tastenanschlägen verwenden:

Keystring	Hinweis	Keystring	Hinweis
{VKnnn}	nnn = ANSI-Code	{Backspace}	
{Alt}		{Break}	Abbruch
{Ctrl}		{CapsLock}	
{Control}		{Clear}	
{Shift}		{Del}	
{0} – {9}	Zahlen	{Delete}	
{A} – {Z}	Buchstaben	{End}	
{F1} – {F16}	Funktionstaste	{Enter}	
{NumLock}	Num. Zehnerblock	{Esc}	
{NumAdd}		{Escape}	

Keystring	Hinweis	Keystring	Hinweis
{NumSubtract}		{Help}	
{NumMultiply}		{Home}	
{NumDivide}		{Ins}	
{NumDecimal}		{Insert}	
{Num0} – {Num9}		{Minus}	
{Left}		{Pause}	
{Right}		{ScrLock}	
{Up}		{ScrollLock}	
{Dn}		{PrintScrn}	
{Down}		{PrintScreen}	
{PgDn}		{Space}	
{PageDown}		{Tab}	
{PgUp}		{LeftBrace}	{
{PageUp}		{RightBrace}	}
{Bksp}			

Die zuvor genannten Meta-Symbole werden in eine sog. *Keydown message* (WM_KEYDOWN oder WM_SYSKEYDOWN) und eine sog. *Keyup message* (WM_KEYUP oder WM_SYSKEYUP) konvertiert. Es erfolgt keine Unterscheidung zwischen Groß- und Kleinbuchstaben. Leerstellen oder andere Symbole (+ und –) dienen nur zur Trennung. Über + und – können Sie Meta-Symbole kombinieren. Über + wird die links davon definierte Taste so lange gedrückt gehalten, bis die Taste rechts davon ausgeführt wurde ({Shift+F9} bedeutet, daß die Umschalttaste gedrückt gehalten wird, während zusätzlich F9 gedrückt wird. Erst danach wird die Umschalttaste wieder freigegeben. Folgt einem Meta-Symbol kein +, werden die zuvor durch + verbundenen Tasten gedrückt gehalten, bis sie später wieder losgelassen werden ({Control+Shift+} werden solange gedrückt gehalten, bis{Control-Shift-} ausgeführt wird.

Die Tasten + und – können auch gemeinsam verwendet werden: {Shift+Del-Del+Ins} bedeutet z. B. einen markierten Text auszuschneiden (Shift+Del) und auf Cursorposition wieder einzufügen (Del+Ins).

6.106 SizeOf

Befehlsform: *Größe*=**SizeOf**(*Variable*)

Ermittelt die Größe der angegebenen Variablen, um die Daten zu speichern.

Parameter

Größe Ermittelte Größe. Der Wert ist in der Regel um 2 größer als die wirkliche Länge von *Variable.*

Variable Variable, deren Größe ermittelt werden soll. Wird dieser Parameter weggelassen, wird die Größe des PerfectScript-System-Overheads zum Speichern eines Wertes zurückgegeben.

6.107 Speed

Befehlsform: **Speed**(*10tel Sekunden*)

Steuerung der Geschwindigkeit der Makro-Ausführung. Zwischen den einzelnen Tastenanschlägen bzw. zwischen der Anzeige von Menüs/Dialogfeldern und ausgeführten Befehlen wird die Ausführung um die angegebene Zeitspanne verzögert. Hierdurch können Sie z. B. eine Laufschrift erzeugen oder die Makroausführung verzögern, um beim Testen von Makros nach Fehlern zu suchen. Der höchste erlaubte Wert ist 600 (= eine Minute). Verwenden Sie dann zusätzlich den produktspezifischen Befehl *Display(On!)*, damit Sie den Makroablauf am Bildschirm verfolgen können (siehe Kapitel »8 Ausgewählte produktspezifische Befehle«).

Der eingegebene Wert wird in 10tel-Sekunden interpretiert. Je größer der Wert, um so langsamer die Ausführung, je kleiner der Wert, um so schneller die Ausführung. Da die Verwendung dieses Befehls an jeder beliebigen Stelle des Makros möglich ist, können Sie z. B. nur bestimmte Bereiche des Makros eingrenzen. Wenn das Makro wieder mit »normaler« Geschwindigkeit arbeiten soll, geben Sie als Wert 0 ein.

Parameter

10tel Sekunden Angabe der Makro-Ausführungsgeschwindigkeit in 1/10 Sekunden.

Beispiel

```
Speed(20)
```

Weitere Hinweise siehe: Wait, Display

6.108 Step

Befehlsform:	**Step**(On!) **Step**(Off!)

Bei der Fehlersuche kann Ihnen der PerfectScript-Debugger zusätzliche Hilfe leisten. Er erlaubt Ihnen die schrittweise Ausführung eines Makros, wobei auch die aktuell verwendeten Variablen und deren Inhalt angezeigt werden. Die Definition neuer (oder vergessener Variablen) ist möglich. Die Inhalte von Variablen können jederzeit verändert werden. Neu definierte Variable werden nicht automatisch in das Makro übernommen. Sie sind bei Bedarf nachträglich zu ergänzen.

Der PerfectScript-Debugger wird gestartet, indem Sie den Befehl *Step(On!)* in dem betreffenden Makro verwenden. Dieser Befehl kann am Beginn eines Makros eingefügt werden oder an der Stelle, an der Fehler vermutet werden. Das ist insbesondere bei umfangreichen Makros sinnvoll, um nicht jeden einzelnen Befehl bei der Makroausführung zu kontrollieren. Benutzen Sie den PerfectSpript-Debugger wie folgt:

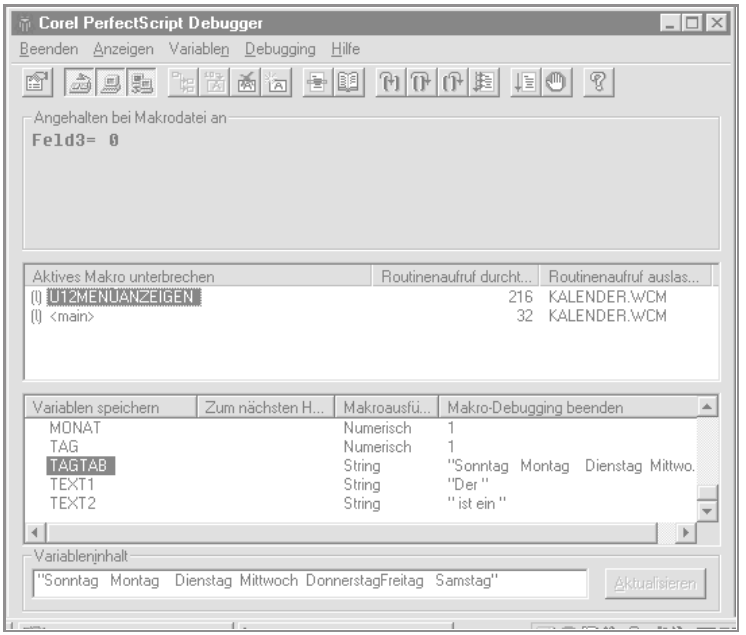
1. Öffnen Sie das zu prüfende Makro.
2. Fügen Sie an der gewünschten Stelle den Befehl Step(On!) ein.
3. Wenn die Kontrolle nur für einen bestimmten Bereich gelten soll, fügen Sie zum Beenden der Kontrolle an der betreffenden Stelle den Befehl Step(Off!) ein.
4. Speichern und kompilieren Sie das Makro.
5. Führen Sie das Makro aus.

Nach dem Starten des Makros und der Ausführung von *Step(On!)* wird der PerfectSpript-Debugger gestartet und das Dialogfeld *PerfectScript Debugger* angezeigt.

Über dieses Dialogfeld kann nun der Makroablauf schrittweise gesteuert werden. Beim jedem Drücken von auf F8 wird der nächste Makrobefehl ausgeführt. In dem Listenfeld [Variablen speichern] werden die bis zu diesem Zeitpunkt verarbeiteten Variablen namentlich und in dem Feld [Inhalt] bei Bedarf inhaltlich angezeigt.

Der Ablauf kann jederzeit abgebrochen (= Makroende) werden.

Weitere Informationen finden Sie in Kapitel »5.14.1 PerfectScript Debugger«.



6.109 StrFill

Befehlsform: *Ergebnis*=**StrFill**(*Zähler;Daten*)

Mit diesem Befehl werden die in *Daten* gespeicherten Daten so oft wiederholt, wie im *Zähler* angegeben wurde.

Parameter

Ergebnis Diese Variable enthält nach der Ausführung des Befehls das ermittelte Ergebnis.

Zähler Anzahl der Wiederholungen.

Daten Daten, die wiederholt werden sollen.

Beispiel

```
Ergebnis=           StrFil(3;"WordPerfect 7")
Type                (Ergebnis)
Als Ergebnis wird »WordPerfect 7WordPerfect 7WordPerfect 7« gedruckt.
```

6.110 StrFraction

Befehlsform: *Ergebnis*=**StrFraction**(*String*)

Dieser Befehl liefert das numerische Ergebnis eines Bruches.

Parameter

Ergebnis Diese Variable enthält nach der Ausführung des Befehls das ermittelte Ergebnis.

String String, der umgewandelt werden soll. Die Daten müssen als »n/n« oder »n n/n« vorhanden sein.

Beispiel

```
Ergebnis=       StrFraction("1/2")
Type            (Ergebnis)
Ergebnis=       StrFraction("2 1/4")
Type            Ergebnis
```

Als Ergebnis wird »0,5« bzw. »2,25« gedruckt.

6.111 StrInsert

Befehlsform: *Ergebnis*=**StrInsert**(*Ursprung*;*Einfügen*;*StartPos*;*Anzahl*)

Mit diesem Befehl kann ein String in einen bestehenden String eingefügt oder Zeichen eines Strings ersetzt bzw. gelöscht werden.

Parameter

Ergebnis Diese Variable enthält nach der Ausführung des Befehls das ermittelte Ergebnis.

Ursprung Daten, die bearbeitet werden sollen.

Einfügen Daten, die in den String *Ursprung* eingefügt werden sollen. Enthält diese Variable keine Daten, werden die Zeichen in *Ursprung* gelöscht.

StartPos Startposition, ab der die Daten von *Einfügen* in *Ursprung* eingefügt werden sollen. Wird keine Position angegeben, werden die Daten am Ende von *Ursprung* eingefügt. Bei negativen Werten (Werte kleiner als 0) wird die Startposition vom Ende des Strings aus ermittelt.

Anzahl Anzahl der Zeichen, die *Ursprung* durch *Einfügen* ersetzt werden sollen. Wird *Einfügen* nicht angegeben, werden Zeichen in der angegebenen Anzahl aus *Ursprung* gelöscht. Wird *Anzahl* nicht angegeben oder enthält diese den Wert 0, werden keine Zeichen ersetzt. Wird ein Wert kleiner als 0 angegeben, werden alle Zeichen ab der Startposition nach rechts durch *Einfügen* ersetzt bzw. gelöscht.

Beispiel

```
Ergebnis=  StrInsert("WordPerfect";"";5;7)          Ergebnis enthält Word
Ergebnis=  StrInsert("WordPerfect";"xxxx";5)        Ergebnis enthält WordxxxxPerfect
Ergebnis=  StrInsert("WordPerfect";"xxxx";5;4)      Ergebnis enthält Wordxxxxect
Ergebnis=  StrInsert("Wordxxxxect";"Perf";5;4)      Ergebnis enthält WordPerfect
```

6.112 StrIsChar

Befehlsform: *Ergebnis*=**StrIsChar**(*Daten*;*StartPos*;*Optionen*;*Datentyp*)

Ermitteln, ob ein String oder ein Zeichen einem bestimmten Datentyp entspricht.

Parameter

Ergebnis Diese Variable enthält nach der Ausführung des Befehls das ermittelte Ergebnis oder »False«, wenn die Startposition größer ist als die Anzahl der Zeichen des Strings.

Daten Zu überprüfende Daten.

StartPos Startposition, ab der die Überprüfung beginnen soll. Enthält diese Variable 0 oder wird sie weggelassen, werden alle Zeichen in *Daten* überprüft. Ist der Wert von *StartPos* größer als die Anzahl der Zeichen von *Daten*, wird »False« zurückgegeben. Bei Werten die kleiner als 0 sind, beginnt die Überprüfung am Ende von *String* (-1 entspricht dem letzten Zeichen, -2 dem vorletzten usw.).

Optionen Festlegung der Überprüfungsart in Verbindung mit dem angegebenen Datentyp. Wird keine Option angegeben, wird EqualTo! angenommen. Werden weder Optionen noch Datentyp angegeben, wird NotEqualTo! verwendet und WhiteSpace! als Datentyp angenommen.

EqualTo!	Überprüft, ob das betr. Zeichen dem angegebenen Datentyp entspricht.
NotEqualTo!	Überprüft, ob das betr. Zeichen nicht dem angegebenen Datentyp entspricht.

Datentyp Zu überprüfende(r) Datentyp(en). Wird kein Datentyp angegeben, wird WhiteSpace! angenommen. Unter *Datentyp* können benutzerdefinierte Zeichen oder eine Kombination der nachfolgend genannten Datentypen verwendet werden, die durch | zu trennen sind:

Alphabetic!	AlphaNumeric!
Numeric!	Punctuation!
WhiteSpace!	UpperCase!
LowerCase!	

Beispiel

```
Ergebnis=  StrIsChar("BBBBBBBBBB";2;EqualTo!;Alphabetic!)       Ergebnis enthält True
Ergebnis=  StrIsChar("bbbbBBBBBB";5;EqualTo!;Numeric!)          Ergebnis enthält False
Ergebnis=  StrIsChar("BBBBBBBBBB";;EqualTo!;Lowercase)          Ergebnis enthält False
Ergebnis=  StrIsChar("bbbbbbbbbbbbb";;EqualTo!;Lowercase)       Ergebnis enthält True
```

6.113 StrLeft

Befehlsform: *Ergebnis*=**StrLeft**(Daten;*Länge*)

Dieser Befehl ermittelt den linken Teil eines Strings in der angegebenen Länge.

Parameter

Ergebnis Diese Variable enthält nach der Ausführung des Befehls das ermittelte Ergebnis.

Daten Daten, die zu bearbeiten sind.

Länge Anzahl der Zeichen, die aus *String* zu extrahieren sind, gerechnet ab dem Anfang von *Daten*. Ist die angegebene Länge größer als die Länge der Daten, wird der gesamte Inhalt von *Daten* in *Ergebnis* übernommen. Bei einer Länge von 0 bleibt *Ergebnis* leer. Wird keine Länge angegeben, wird der volle Inhalt von *Daten* in *Ergebnis* übernommen.

Beispiel

```
Ergebnis=          StrLeft("WordPerfect";4)
Type               (Ergebnis)
```

Als Ergebnis wird »Word« gedruckt.

6.114 StrLen

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

Die Länge (= Anzahl Zeichen) des in Ausdruck gespeicherten Textes (= String oder Variable) wird ermittelt und in *Variable* gespeichert (siehe auch *CharLen*). Die Verwendung dieses Befehls ist dann sinnvoll, wenn ein eingegebener Text eine bestimmte Länge nicht überschreiten darf, oder wenn die Länge eines Variableninhalts ermittelt werden muß. Der Unterschied zwischen *StrLen* und *CharLen* besteht darin, daß mit *StrLen* die wirkliche Anzahl der Zeichen ermittelt wird (siehe Beispiel). Bei *CharLen* werden auch vorhandene Steuerzeichen mitgezählt. Beachten Sie bitte auch Kapitel »5.17 Änderungen gegenüber WPWin 5.1/5.2«.

Parameter

Variable Beliebiger Variablenname. Speicherung der ermittelten Stringlänge (= Anzahl der Zeichen ohne Steuerzeichen).

Ausdruck String oder Variablenname. Die Anzahl der hier angegebenen Zeichen wird ermittelt und in *Variable* gespeichert. Leerzeichen werden mitgezählt. Variable mit numerischem Inhalt müssen vor der Ausführung dieses Befehls erst mit *NumStr* in einen String konvertiert werden.

Beispiel

```
1         StrLen              (Länge;"Markt & Technik")Länge = 15 Zeichen    oder
          Länge=              StrLen("Markt & Technik")Länge = 15 Zeichen

2         Application         (A1;"WordPerfect";Default;"DE")
          Text1=              "Anzahl in Zeichen:  "
          GetString           (Eingabe;"Bitte Text eingeben:";"Befehl  ""STRLEN""  testen")
          Länge=              StrLen(Eingabe)
          Anzeige=            NumStr(Länge;0)
          Meldung=            (Text1+Anzeige)
          Prompt              ("Befehl  ""STRLEN""  testen";Meldung;;;)
            Wait              (20)
          EndPrompt
```

Durch die Eingabe eines beliebigen Textes können Sie die Anzahl der Zeichen dieses Textes ermitteln. Dieses Beispiel finden Sie auf der CD-ROM unter dem Dateinamen STRLEN.WCM.

Weitere Hinweise siehe: CharLen, StrPos, SubStr

6.115 StrNum

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

Der in *Ausdruck* gespeicherte String wird in einen numerischen Wert konvertiert. Wurde z. B. in *Ausdruck* der Text »12345« gespeichert, wird er in die Zahl 12345 konvertiert. Obwohl beide Werte beim Drucken oder Anzeigen am Bildschirm identisch sind, besteht in der Art der Speicherung ein Unterschied: Nur mit numerischen Werten kann WordPerfect Rechenoperationen durchführen. Vor der Ausführung solcher Operationen ist darum sicherzustellen, daß die verwendeten Variablen auch einen numerischen Inhalt haben. Beachten Sie bitte auch Kapitel »5.17 Änderungen gegenüber WPWin 5.1/5.2«.

Parameter

Variable Beliebiger Variablenname. Speicherung des konvertierten, numerischen Wertes.

Ausdruck String oder Variablenname. Die angegebenen Ziffern werden in einen numerischen Wert konvertiert. Bei der Konvertierung werden Ziffern und das Dezimalkomma berücksichtigt. Buchstaben und alle nachfolgenden Zeichen werden ignoriert.

Beispiel

```
1       Zahl=               StrNum("123,45")      oder
        StrNum              (Zahl;"123,45")

        Nach der Ausführung des Befehls wird in Zahl der Wert 123,45 gespeichert.

2       Zahl=               StrNum("123X34,678")
```

Nach der Ausführung des Befehls wird in *Zahl* der Wert 123 gespeichert, da ab dem Buchstabe X alle nachfolgenden Zeichen ignoriert werden.

Weitere Hinweise siehe: NumStr

6.116 StrPad

Befehlsform: *Ergebnis*=**StrPad**(*Daten*;*Länge*;*Optionen*;*PadString*)

Mit diesem Befehl können Sie einem String eine bestimmte Länge zuweisen.

Parameter

Ergebnis Diese Variable enthält nach der Ausführung des Befehls das ermittelte Ergebnis.

Daten Daten, die zu bearbeiten sind.

Länge Gesamtlänge der Daten. Ist die Länge kleiner oder gleich 0 bzw. kleiner als die Länge von *Daten*, wird der Originalinhalt von *Daten* in *Ergebnis* übernommen.

Optionen Festlegung, wo die Erweiterung stattfinden soll. Standardmäßig wird PadRight! angenommen.

PadRight!	Erweiterung am Ende von Daten.
PadLeft!	Erweiterung am Anfang von Daten.
PadEnds!	Erweiterung am Anfang und am Ende, wobei der Inhalt von Daten zentriert wird.
PadWords!	**Erweiterung zwischen Wörtern. Wörter müssen in PadString** durch Zeichen getrennt werden, andernfalls wird PadEnds! verwendet.

PadString Mehrere Zeichen dieser Variablen erweitern *Daten* auf die angegebene Länge. Jeder beliebige String ist erlaubt. Wird der Parameter weggelassen, werden Leerstellen verwendet. Ist der String leer (""), wird keine Erweiterung durchgeführt.

Beispiel

```
Ergebnis=       StrPad("WordPerfect";20;PadRight!;"abcde")
Ergebnis=       WordPerfectabcdeabcd

Ergebnis=       StrPad("WordPerfect";20;PadLeft!;"12345")
Ergebnis=       123451234WordPerfect

Ergebnis=       StrPad("WP1für2Windows395";25;PadWords!;"12345")
Ergebnis=       WP111für222Windows3339555
```

6.117 StrPos

Befehlsform: *Position*=**StrPos**(*SuchenIn*,*SuchText*)
StrPos(*Position;SuchText;SuchenIn*) (WPWin 5.2)

Mit diesem Befehl können Sie prüfen, ob der Inhalt von *SuchText* in *SuchenIn* enthalten ist. Wenn ja, wird in *Position* die Stelle, an der der gesuchte Ausdruck beginnt, in Form eines numerischen Wertes angegeben. Variable mit numerischem Inhalt müssen vor der Ausführung dieses Befehls mit *NumStr* in einen String konvertiert werden. Möchten Sie z. B. feststellen, ob der Text »Windows« (= *SuchText*) in dem Text »WordPerfect für Windows« (= *SuchenIn*) enthalten ist, und ab welcher Position er beginnt, müßte der Befehl wie folgt lauten:

```
Beginn=StrPos("WordPerfect für Windows";"Windows")Beginn = Stelle 17
```

Der Unterschied zwischen *StrPos* und *CharPos* besteht darin, daß mit *StrPos* die wirkliche Position ermittelt wird, bei *CharPos* werden vorhandene Steuerzeichen mitgezählt.

Achtung: Wurde das Makro unter WPWIN 5.x erstellt, muß der Befehl zwingend dem neuen Format angepaßt werden (war unter 6.1 nicht erforderlich). Das Makro ist in Version 7 zwar lauffähig, der Befehl liefert jedoch falsche Ergebnisse.

Parameter

Position Beliebiger Variablenname. Hier wird die ermittelte Startposition (Position, an der *SuchText* in *SuchenIn* beginnt), gespeichert. Ist *SuchText* nicht in *SuchenIn* enthalten, wird in *Position* der Wert 0 gespeichert. Vor der Weiterverarbeitung sollten Sie darum den Inhalt der Variablen überprüfen.

SuchText String oder Inhalt einer Variablen, für die in *SuchenIn* die Startposition zu ermitteln ist.

SuchenIn String oder Inhalt einer Variablen, in dem/der Inhalt von *SuchText* zu suchen ist.

Beispiel

siehe oben.

Weitere Hinweise siehe: CharPos, StrLen, Substr

6.118 StrReverse

Befehlsform: *Ergebnis*=**StrReverse**(*Daten*)

In *Ergebnis* werden die Zeichen von *Daten* in umgekehrter Reihenfolge gespeichert.

Parameter

Ergebnis Diese Variable enthält nach der Ausführung des Befehls das ermittelte Ergebnis.

Daten Daten, die in umgekehrter Reihenfolge zurückgegeben werden sollen.

Beispiel

```
Ergebnis=       StrReverse("WordPerfect")
Type            (Ergebnis)
```

Als Ergebnis wird »tcefrePdroW« gedruckt.

6.119 StrRight

Befehlsform: *Ergebnis*=**StrRight**(*Daten*;*Länge*)

Dieser Befehl ermittelt den rechten Teil eines Strings in der angegebenen Länge.

Parameter

Ergebnis Diese Variable enthält nach der Ausführung des Befehls das ermittelte Ergebnis.

Daten Daten, die zu bearbeiten sind.

Länge Anzahl der Zeichen, die aus *Daten* zu extrahieren sind, gerechnet ab dem Ende von *Daten*. Ist die angegebene Länge größer als die Länge der Daten, wird der gesamte Inhalt von *Daten* in *Ergebnis* übernommen. Bei einer Länge von 0 bleibt *Ergebnis* leer. Wird keine Länge angegeben, wird der volle Inhalt von *Daten* in *Ergebnis* übernommen.

Beispiel

```
Ergebnis=       StrRight(»WordPerfect«;7)
Type            (Ergebnis)
```

Als Ergebnis wird »Perfect« gedruckt.

6.120 StrScan

Befehlsform: *Ergebnis*=**StrScan**(*Daten*;*StartPos*;*Optionen*;*Datentyp*)

Ermittelt die Position des ersten Auftretens bzw. Nichtauftretens eines Zeichens in Verbindung mit *Datentyp*.

Parameter

Ergebnis Diese Variable enthält nach der Ausführung des Befehls das ermittelte Ergebnis. Wurde keine Übereinstimmung oder das Nichtauftreten eines gesuchten Zeichens festgestellt, wird die Länge von *Daten* + 1 zurückgegeben bzw. 0 wenn die Suche in umgekehrter Reihnfolge erfolgte.

Daten Zu überprüfende Daten.

StartPos Startposition, ab der die Überprüfung beginnen soll. Enthält diese Variable 0 oder wird sie weggelassen, beginnt die Überprüfung bei Zeichen 1. Bei Werten die kleiner als 0 sind, beginnt die Überprüfung am Ende von *Daten* in umgekehrter Reihenfolge.

Optionen Festlegung der Überprüfungsart in Verbindung mit dem angegebenen Datentyp. Wird keine Option angegeben, wird EqualTo! angenommen. Werden weder Optionen noch Datentyp angegeben, wird NotEqualTo! verwendet und WhiteSpace! als Datentyp angenommen.

EqualTo!	Überprüft, bis ein Zeichen des angegebenen Datentyps gefunden wurde.
NotEqualTo!	Überprüft, bis kein Zeichen des angegebenen Datentyps gefunden wurde.

Datentyp Zu überprüfende(r) Datentyp(en). Wird kein Datentyp angegeben, wird WhiteSpace! angenommen. Unter *Datentyp* können benutzerdefinierte Zeichen oder einer der nachfolgend genannten Datentypen verwendet werden:

Alphabetic!	AlphaNumeric!
Numeric!	Punctuation!
WhiteSpace!	UpperCase!
LowerCase!	

Beispiel

```
Ergebnis=StrScan(»WordPerfect 7«; 5;EqualTo!;Numeric!)
Ergebnis=13
```

6.121 StrToChars

Befehlsform: *Ergebnis*=**StrToChars**(*Daten;Optionen;Datentyp*)

Entfernt bestimmte Zeichen in Verbindung mit *Datentyp* aus einer Variablen.

Parameter

Ergebnis Diese Variable enthält nach der Ausführung des Befehls das ermittelte Ergebnis.

Daten Daten, die zu bearbeiten sind.

Optionen Festlegung, welche Zeichen gelöscht werden sollen und welche nicht. Wird der Parameter weggelassen, wird Keep! angenommen. Wird weder *Optionen* noch *Datentyp* verwendet, wird Remove! benutzt und als Datentyp Whitespace! verwendet.

Keep!	Zeichen des angegebenen Datentyps bleiben erhalten, alle anderen werden gelöscht.
Remove!	Zeichen des angegebenen Datentyps werden gelöscht.

Datentyp Datentyp, dessen in *Daten* zugehörige Zeichen zu löschen sind. Wird kein Datentyp angegeben, wird WhiteSpace! angenommen. Unter *Datentyp* können benutzerdefinierte Zeichen oder eine Kombination der nachfolgend genannten Datentypen, die durch | zu trennen sind, verwendet werden:

Alphabetic!	AlphaNumeric!
Numeric!	Punctuation!
WhiteSpace!	UpperCase!
LowerCase!	

Beispiel

```
Ergebnis=          StrToChars("DM 1,50";Remove!;Alphabetic! | Punctuation!)
Type               (Ergebnis)
```

Als Ergebnis wird »150« gedruckt.

6.122 StrTransform

Befehlsform: *Ergebnis*=**StrTransform**(*Daten*;*VonZeichen*;*InZeichen*;*Optionen*)

Umwandeln bestimmter Zeichen eines Strings in andere Zeichen.

Parameter

Ergebnis Diese Variable enthält nach der Ausführung des Befehls das ermittelte Ergebnis.

Daten Daten, die zu bearbeiten sind.

VonZeichen Zeichen in *Daten*, die umzuwandeln sind. Mehrere Zeichen sind erlaubt.

InZeichen Zeichen, die die unter *VonZeichen* gefundenen Zeichen ersetzen sollen. Mehrere Zeichen sind erlaubt.

Optionen Folgende Optionen stehen zur Verfügung:

Characters!	Strings!
All!	FirstOnly!

Beispiel

```
Ergebnis=          StrTransform("WordSort";"or";"xxyy")
Type               (Ergebnis)
```

Bei der Verwendung der Options-Parameter werden folgende Ergebnisse erzielt:

Keine Option:	WxxdSxxt
Characters!	WxxdSxxt
All!	WxxdSxxt
FirstOnly!	WxrdSort
Strings!	WxxyydSxxyyt

6.123 StrTrim

Befehlsform: *Ergebnis*=**StrTrim**(*Daten;Länge;Optionen;TrimZeichen*)

Löschen einer bestimmten Anzahl von Zeichen in Verbindung mit *TrimZeichen*, um einen String zu kürzen.

Parameter

Ergebnis Diese Variable enthält nach der Ausführung des Befehls das ermittelte Ergebnis.

Daten Zu bearbeitende Daten.

Länge Endgültige Länge des zu kürzenden Strings. Ist die angegebene Länge größer als die von *Daten*, wird keine Kürzung vorgenommen und in *Ergebnis* der Inhalt von *Daten* gespeichert. Wird keine Länge angegeben bzw. ist diese kleiner oder gleich 0, wird *Daten* so lange gekürzt, bis alle übereinstimmenden Zeichen an den betreffenden Positionen gelöscht sind. Wenn die Länge von *Daten* nach der Ausführung des Befehls die angegebene Länge überschreitet, wird keine weitere Kürzung vorgenommen.

Optionen Festlegung, wo zu kürzen ist. Zeichen in *Daten*, die den unter *TrimZeichen* angegebenen Zeichen an den festgelegten Positionen entsprechen, werde so lange gelöscht, bis die gewünschte Länge erreicht wurde oder bis keine Zeichen mehr übereinstimmen. Standardmäßig wird TrimRight! angenommen.

TrimRight!	Kürzung am Ende von Daten.
TrimLeft!	Kürzung am Anfang von Daten.
TrimEnds!	Kürzung am Anfang und Ende, der Inhalt von Daten wird zentriert.
TrimWords!	Kürzen mehrerer Zeichen in *Daten*, bis zur gewählten Länge. Wird die gewünschte Länge immer noch überschritten, wird die Kürzung aufgrund von TrimEnds! vorgenommen.

TrimZeichen Zeichen, um die *Daten* gekürzt werden soll. Hier können benutzerdefinierte Zeichen verwendet werden oder einer der nachfolgend genannten Datentypen. Wird dieser Parameter nicht verwendet, wird WhiteSpace! angenommen. Ist der Parameter leer (""), wird keine Kürzung vorgenommen.

Alphabetic!	AlphaNumeric!
Numeric!	Punctuation!
WhiteSpace!	UpperCase!
LowerCase!	

Beispiel

```
Ergebnis=          StrTrim("123456789"; 5;TrimRight!;Numeric)
Ergebnis=          12345

Ergebnis=          StrTrim("WordPerfect"; 3;TrimRight!;Alphabetic!)
Ergebnis=          Wor

Ergebnis=          StrTrim("WordPerfect"; 3;TrimLeft!;Alphabetic!)
Ergebnis=          ect
```

6.124 StrUnit

Befehlsform: *Maßeinheit*=**StrUnit**(*StringWert*)

Der in *StringWert* gespeicherte String wird in eine Maßeinheit konvertiert und in *Maßeinheit* gespeichert. Wurde z. B. in *StringWert* der Text »12345« gespeichert, wird er in die Zahl 12345 konvertiert und die aktuelle Maßeinheit ergänzt. Obwohl beide Werte beim Drucken oder Anzeigen am Bildschirm identisch sind, besteht in der Art der Speicherung ein Unterschied: Nur mit numerischen Werten kann WordPerfect hier z. B. Ränder definieren. Vor der Ausführung solcher Operationen ist darum sicherzustellen, daß die verwendeten Variablen auch einen numerischen Wert enthalten. Beachten Sie bitte auch Kapitel »5.17 Änderungen gegenüber WPWin 5.1/5.2«.

Parameter

Maßeinheit Beliebiger Variablenname. Speicherung des konvertierten, numerischen Wertes.

StringWert String oder Variablenname. Die angegebenen Ziffern werden in einen numerischen Wert konvertiert. Bei der Konvertierung werden Ziffern und das Dezimalkomma berücksichtigt. Buchstaben und alle nachfolgenden Zeichen werden ignoriert.

Beispiel

```
Application        (A1; "WordPerfect"; Default; "DE")
GetString          (LRand;"Bitte Wert einschließlich Maßeinheit eingeben, z. B. 17m
                   (= Millimeter).";"Linke Randeinstellung")
LinkerRand=        StrUnit(LRand)
RandLinks          (RandBreite:LinkerRand)        //Bitte Steuerzeichen kontrollieren.
DruRand=           UnitStr(LinkerRand;Centimeters!)
Prompt             ("Befehl ""STRUNIT"" testen";
                   "Folgende Maßeinheit wurde eingegeben und umgesetzt: "+DruRand;;;)
Wait               (50)
EndPrompt
```

Der über *GetString* eingegebene Wert wird durch *StrUnit* in eine Maßeinheit umgewandelt. Vor der Anzeige durch *Prompt* muß die Maßeinheit über *UnitStr* wieder in einen String umgewandelt werden.

Weitere Hinweise siehe: Assign, NumStr, StrLen, StrNum, StrPos

6.125 SubChar

Befehlsform:	*Extrakt*=**SubChar**(*Ausdruck;StartPos;Länge*)	
	SubChar(*Extrakt;StartPos;Länge;Ausdruck*)	(WPWin 5.2)
	SubChar(*Extrakt;StartPos;Länge;Ausdruck*)	(WPWin 5.2)

Aus dem in *Ausdruck* gespeicherten String können Sie einen Teil extrahieren. Der extrahierte String wird in *Extrakt* gespeichert. Den Beginn des ersten Zeichens ab dem extrahiert werden soll und die Länge des zu extrahierenden Strings müssen Sie in *StartPos* bzw. in *Länge* angeben. Bei diesem Befehl werden auch gespeicherte Steuerzeichen berücksichtigt. Arbeiten Sie mit *SubString*, wenn Sie die wirkliche Anzahl der Zeichen benötigen (also ohne Steuerzeichen). Beachten Sie bitte auch Kapitel »5.17 Änderungen gegenüber WPWin 5.1/5.2«.

Parameter

Extrakt	Beliebiger Variablenname. In dieser Variablen wird der extrahierte String gespeichert.
Ausdruck	String oder Variable, aus dem/der ein Teil extrahiert werden soll. Variable mit numerischem Inhalt müssen vor der Ausführung dieses Befehls mit *NumStr* in einen String konvertiert werden.
StartPos	Startposition, ab der das Extrahieren des in *Ausdruck* gespeicherten Strings beginnen soll. Gezählt wird immer von links nach rechts.
Länge	Länge des Strings, der aus der Variablen *Ausdruck* extrahiert werden soll.

Beispiel

```
Ausschnitt=SubChar("Markt & Technik";6;7)

Aus dem Text Markt & Technik werden ab Position 6 die nächsten 7 Zeichen in der
Variablen Ausschnitt gespeichert, in diesem Fall & Techn.
```

Weitere Hinweise siehe: CharLen, CharPos, StrLen, StrPos

6.126 SubStr

Befehlsform: *Extrakt*=**SubStr**(*Ausdruck;StartPos;Länge*)
SubStr(*Extrakt;StartPos;Länge;Ausdruck*) (WPWin 5.2)

Aus dem in *Ausdruck* gespeicherten String können Sie einen Teil extrahieren. Der extrahierte String wird in *Extrakt* gespeichert. Den Beginn des ersten Zeichens ab dem extrahiert werden soll und die Länge des zu extrahierenden Textes müssen Sie in *StartPos* bzw. in *Länge* angeben.

Achtung: Wurde das Makro unter WPWIN 5.x erstellt, muß der Befehl zwingend dem neuen Format angepaßt werden (war unter 6.1 nicht erforderlich). Das Makro ist in Version 7 zwar lauffähig, der Befehl liefert jedoch falsche Ergebnisse.

Parameter

Extrakt Beliebiger Variablenname. In dieser Variablen wird der extrahierte String gespeichert.

Ausdruck String oder Variable, aus dem/der ein Teil extrahiert werden soll. Variable mit numerischem Inhalt müssen vor der Ausführung dieses Befehls zuerst mit *NumStr* in einen String konvertiert werden.

StartPos Startposition, ab der das Extrahieren des in *Ausdruck* gespeicherten Strings beginnen soll. Gezählt wird immer von links nach rechts.

Länge Länge des Strings, der aus der Variablen *Ausdruck* extrahiert werden soll.

Beispiel

```
Ausschnitt=SubStr("WordPerfect";3;5))
```

Aus dem Text *WordPerfect* werden ab Zeichenposition 3 die nächsten 5 Zeichen in der Variablen *Ausschnitt* gespeichert, in diesem Fall *rdPer*.

In dem folgenden Beispiel wird aus der Variablen *Monate* der Monatsname aufgrund einer eingegebenen Ziffer extrahiert.

```
1       Application         (A1;"WordPerfect";Default;"DE")
2       Monate=             "Januar Februar März April Mai Juni Juli August September
                             OktoberNovemberDezember"
3       Label               (Anfang)
4       GetNumber           (Monatsnummer;"Bitte Monatsnummer eingeben.";" ")
5       If                  ((Monatsnummer<1)or(Monatsnummer>12))
            Prompt          ("Fehler";"Eingegebene Monatsnummer ungültig";;;)
            Wait            (20)
            Go              (Anfang)
        EndIf
6       Monatsnummer=       Monatsnummer-1
7       TabPos=             Monatsnummer*9+1
```

```
8       Monatsname=         SubStr(Monate;TabPos;9)
9       Prompt              ("Monatsname anzeigen";Monatsname;;;)
10      Wait                (20)
        EndPrompt
11      Go                  (Anfang)
```

Leser, die andere Programmiersprachen kennen, werden bemerken, daß es sich bei der Definition der Variablen *Monate* indirekt um eine Tabelle (= Array) handelt. Diese Tabelle hat zwölf Felder (= Monate), die alle dieselbe Länge haben müssen. Die Länge richtet sich hier nach dem längsten Monatsnamen (= September) und beträgt neun Stellen. Beachten Sie zum Definieren von Arrays bitte auch den Befehl *Declare*.

1 Applikationskennung.

2 Festlegung der Monatsnamen. Monatsnamen, die kürzer sind als neun Stellen, müssen am Ende mit Leerstellen aufgefüllt werden. Erst hinter dem letzten Hochkomma dürfen Sie [↵] drücken, ansonsten erhalten Sie bei der Kompilierung eine Fehlermeldung.

3 Zu diesem Label wird immer wieder verzweigt, nachdem ein Monatsname ermittelt und über *Prompt* angezeigt wurde.

4 Anzeigen eines Dialogfeldes und Eingabe einer Zahl. Die eingegebene Zahl wird in der Variablen *Monatsnummer* gespeichert. Über *GetNumber* ist nur die Eingabe eines numerischen Wertes möglich, andernfalls erfolgt eine Fehlermeldung.

5 Die eingegebene Zahl wird überprüft. Es wird eine Fehlermeldung angezeigt, wenn sie kleiner als eins oder größer als zwölf ist.

6 Von der Variablen *Monatsnummer* wird 1 subtrahiert, damit die Berechnung in der folgenden Zeile richtig ausgeführt wird.

7 Ermitteln der Stellenposition, ab der der gesuchte Monat beginnen muß.

Beispiel: Sie haben 5 eingegeben, als Ergebnis muß Mai angezeigt werden. Von diesem Wert wird 1 subtrahiert, so daß der Variableninhalt von Monatsnummer jetzt 4 beträgt. Da jedes Monatsfeld neun Zeichen lang ist, wird dieser Wert mit neun multipliziert. Ergebnis: 36. Dies ist aber erst die letzte Stelle des vierten Monats. Darum wird noch eine 1 addiert. Das Ergebnis (37) wird in der Variablen TabPos gespeichert.

8 Über *SubStr* wird jetzt der Monatsname aus der Variablen *Monate* extrahiert. Die Position, ab der extrahiert werden soll, befindet sich in der Variablen *TabPos*, z. B. 37. Die Länge des zu extrahierenden Strings ist mit 9 definiert. D. h. ab der Stelle 37 wird ein String in der Länge von neun Zeichen extrahiert.

9 Anzeige des ermittelten Monatsnamens.

10 Das Dialogfeld wird für 20/10 Sekunden angezeigt.

11 Das Makro verzweigt zum Anfang. Klicken Sie zum Beenden des Makros auf [*Abbrechen*].

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

Weitere Hinweise siehe: CharLen, CharPos, Declare, StrLen, StrPos

6.127 Switch

Befehlsform: **Switch**(*Test*)

Über diesen Befehl wird in Verbindung mit den unter *Test* angegebenen Variablen oder Konstanten entschieden, welche *CaseOf*-Anweisungen ausgeführt werden sollen. Dieser Befehl tritt immer in Verbindung mit den Befehlen *CaseOf* und *Continue* auf, wobei *Continue* nur bei Bedarf verwendet wird. Die Angabe in *Test* wird mit jedem Parameter der *CaseOf*-Befehle verglichen. Der Vergleich erfolgt in der angegebenen Reihenfolge der *CaseOf*-Befehle. Es sind mehrere *CaseOf*-Befehle erlaubt. Trifft die Angabe bei einem **CaseOf**-Befehl zu, werden die Anweisungen ausgeführt, die dem zutreffenden Befehl folgen. Die *Switch*-Befehlsstruktur 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 noch weitere zutreffen würden (siehe Befehl *Continue*). Über den Befehl *Default* können Sie festlegen, wie weitergearbeitet werden soll, wenn keine der *CaseOf*-Bedingungen zutrifft. *Default* muß immer dem letzten *CaseOf*-Befehl folgen. Jede *Switch*-Struktur muß mit *EndSwitch* enden.

Parameter

Test Variablenname(n) oder Konstante(n). Werden mehrere Konstanten oder Variable angegeben, müssen diese durch ein Semikolon getrennt werden (= oder-Bedingung). Zwischen Groß- und Kleinbuchstaben wird unterschieden.

Beispiel: Siehe unter *CaseOf* und *Continue*.

Weitere Hinweise siehe: CaseOf, Continue, Default, EndSwitch

6.128 Time-Befehle

Mit den nachfolgend beschriebenen Befehlen können Sie Uhrzeitangaben in verschiedenen Variationen dem System entnehmen, in einer Variablen speichern und in dem Makro weiterverwenden. Wird der Parameter *Zeit* weggelassen, wird für das betreffende Feld der jeweilige Wert der aktuellen Systemzeit verwendet.

TimeHour *Variable*=**TimeHour**(*Zeit*)

Ermittelt die Stunde (24-Stundentag).

TimeHundreth *Variable*=**TimeHundreth**(*Zeit*)

Ermittelt die 100stel Sekunden.

TimeMinute *Variable*=**TimeMinute**(*Zeit*)

Ermittelt die Minuten.

TimeSecond *Variable*=**TimeSecond**(*Zeit*)

Ermittelt die Sekunden.

TimeString *Variable*=**TimeString**(*Zeit*;*Format*)

Ermitteln der Uhrzeit als String.

Format — Festlegung des Uhrzeitformats. 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.

Stunden	h	Stunden des 12-Stundentags ohne führende Null bei einstelligen Werten.
	hh	Stunden des 12-Stundentags mit führender Null bei einstelligen Werten.
	H	Stunden des 24-Stundentags ohne führende Null bei einstelligen Werten.
	HH	Stunden des 24-Stundentags mit führender Null bei einstelligen Werten.
Minuten	m	Minuten ohne führende Null bei einstelligen Werten.
	mm	Minuten mit führende Null bei einstelligen Werten.
Sekunden	s	Sekunden ohne führende Null bei einstelligen Werten.
	ss	Sekunden mit führender Null bei einstelligen Werten.
Time-Marker	t	Ein-Zeichen-Kennung für 12-Stundentag »a« oder »p«,
	tt	Zwei-Zeichen-Kennung für 12-Stundentag »am« oder »pm«.

Weitere Hinweise siehe: Date-Befehle

6.129 ToInitialCaps

Befehlsform: *Ergebnis*=**ToInitialCaps**(*Daten*;*Style*)

Der erste Buchstabe in *Daten* wird in einen Großbuchstaben umgewandelt. Sind mehrere Wörter enthalten, wird der erste Buchstabe jedes Wortes in einen Großbuchstaben umgewandelt.

Parameter

Ergebnis	Diese Variable enthält die umgewandelten Daten.	
Daten	Daten, die umgewandelt werden sollen.	
Style	FirstCharOnly!	Nur der erste Buchstabe wird groß geschrieben.
	FirstOfEachWord!	Der erste Buchstabe jedes Wortes wird groß geschrieben (= Standard).

Beispiel

```
Ergebnis=       ToInitialCaps(»wordPerfect 7 für windows 95«)
Type            (Ergebnis)
```

Als Ergebnis wird »WordPerfect 7 Für Windows 95« gedruckt.

6.130 ToLower/ToUpper

Befehlsform: **ToLower**(*Ausdruck*)
ToUpper(*Ausdruck*)

Über diese beiden Befehle erfolgt eine Konvertierung von Groß- in Kleinbuchstaben oder umgekehrt:

ToLower	konvertiert den Inhalt einer Variablen oder eines Strings in Kleinbuchstaben. Zahlen und Sonderzeichen bleiben unberücksichtigt.
ToUpper	konvertiert den Inhalt einer Variablen oder eines Strings in Großbuchstaben. Zahlen und Sonderzeichen bleiben unberücksichtigt.

Diese Befehle können nicht alleine ausgeführt werden, sie treten immer in Verbindung mit Befehlen wie z. B. *Assign*, *If* und *Type* auf. Beachten Sie bitte auch Kapitel »5.17 Änderungen gegenüber WPWin 5.1/5.2«.

Parameter

Ausdruck	Variable oder Konstante, die den zu konvertierenden Text enthält.

Beispiel

```
1       If              (ToLower(Kennzeichen)="v")
        Kennz=          ToLower(Kennzeichen)
        Type            (ToLower("Das ist ein Test")
```

In diesen drei Beispielen werden die Möglichkeiten zur Verwendung von *ToLower* dargestellt. Die *If*-Bedingung trifft zu, wenn *Kennzeichen* ein kleines **v** enthält. Ist ein großes **V** vorhanden, wird es in ein kleines **v** konvertiert. Im zweiten Beispiel wird der Inhalt von *Kennzeichen* in Kleinbuchstaben umgesetzt und in *Kennz* gespeichert. In dem letzten Beispiel wird der angegebene Text in Kleinbuchstaben geschrieben. Dasselbe gilt sinngemäß für die Verwendung von *ToUpper*.

```
2          Application          (A1; "WordPerfect"; Default; "DE")
           Display              (on!)
           FileNew              ()
           Weiter=              "j"
           Anfang=              1

           While                (ToLower(Weiter)="j")
              ForNext           (Zähler;Anfang;5;1)
                 Type           (Zähler)
                 If             (Zähler=5)
                    Getstring   (Weiter;"Soll das Makro fortgesetzt werden (j/n)?";
                                "Befehl ""ToLower"" testen";1)
                    Anfang=     1
                 EndIf
              EndFor
           EndWhile
```

Die *While*-Schleife wird so lange ausgeführt, bis über *GetString* ein anderes Zeichen als ein j oder J eingegeben wird.

Weitere Hinweise siehe: Assign, If, ToUpper

6.131 UnitStr

Befehlsform: *StringWert*=**UnitStr**(*Maßeinheit*)

Der in *Maßeinheit* gespeicherte Wert wird in einen String konvertiert und in *StringWert* gespeichert. Wurde z. B. in *Maßeinheit* der Wert 12,345 gespeichert, wird er in den String »12,345« konvertiert und die aktuelle Maßeinheit ergänzt. Obwohl beide Werte beim Drucken oder Anzeigen am Bildschirm identisch sind, besteht in der Art der Speicherung ein Unterschied: Nur mit numerischen Werten kann WordPerfect hier z. B. Ränder definieren. Vor dem Drukken muß der Wert in einen String umgewandelt werden. Beachten Sie bitte auch Kapitel »5.17 Änderungen gegenüber WPWin 5.1/5.2«.

Parameter

StringWert Beliebiger Variablenname. Speicherung des konvertierten Strings.

Maßeinheit Numerischer Wert oder Variablenname. Die angegebenen Ziffern werden in einen String konvertiert. Bei der Konvertierung werden Ziffern und das Dezimalkomma berücksichtigt.

Beispiel

```
           Application          (A1; "WordPerfect"; Default; "DE")
           GetUnits             (LRand;"Bitte Wert einschließlich Maßeinheit eingeben, z. B.
                                17m (= Millimeter).";"Linke Randeinstellung")
           RandLinks            (RandBreite:LinkerRand)
```

```
        //                  Bitte Steuerzeichen kontrollieren.
        DruRand=            UnitStr(LinkerRand;Centimeters!)
        Prompt              ("Befehl ""STRUNIT"" testen";
                            "Folgende Maßeinheit wurde eingegeben und umgesetzt: "+
                            DruRand;;;)
        Wait                (50)
        EndPrompt
```

Der über *GetUnits* eingegebene Wert muß vor der Anzeige durch *Prompt* über *UnitStr* in einen String umgewandelt werden.

Weitere Hinweise siehe: Assign, NumStr, StrLen, StrNum, StrPos

6.132 Until

Befehlsform: **Until**(*Test*)

Jede *Repeat*-Struktur (Schleifenverarbeitung) muß mit einem *Until*-Befehl enden. Hierbei werden die Befehle zwischen *Repeat* und *Until* so oft wiederholt, bis die unter *Until* angegebene Bedingung erfüllt ist, sofern sie nicht vorher durch andere Bedingungen innerhalb der Scheife (z. B. If, Break) beendet wird. Wenn Sie mit Variablen arbeiten, achten Sie bitte darauf, daß diese vor der Ausführung des Befehls gültige Werte enthalten, sonst kann vielleicht eine Endlosschleife entstehen. Geben Sie diese Werte entweder über die Tastatur ein, oder ordnen Sie diese vor der Ausführung von *Repeat* mit dem Befehl *Assign* zu. Im Gegensatz zu den Befehlen *For*, *ForNext* oder *While*, bei denen die Bedingung am Schleifenanfang steht, ist hier die Bedingung zum Beenden der Schleife am Schleifenende definiert. Die Schachtelung mehrerer *Repeat*-Schleifen ist erlaubt, wobei jede Schleife jeweils mit *Until* enden muß.

Falsche Werte oder falsche Bedingungen können zu unvorhersehbaren Ergebnissen führen (z. B. Endlosschleife).

Parameter

Test Eine oder mehrere Vergleichsoperationen.

Beispiel

```
1       Anzahl=             10
        Repeat
            Type            ("*")
            Anzahl=         (Anzahl-1)
        Until               (Anzahl=0)
```

In diesem Beispiel sollen so viele Sterne gedruckt werden, wie in der Variablen *Anzahl* angegeben wurden. Der numerische (!) Startwert für *Anzahl* wurde am Makroanfang gesetzt. Nach dem Drucken eines Sterns wird von der Variablen *Anzahl* eine 1 subtrahiert. Das wiederholt sich so lange, bis in *Anzahl* der Wert 0 errechnet wurde. In diesem Fall ist die Bedingung erfüllt und die Schleife wird beendet.

```
2       Repeat
           Call             (Drucken)
           Gedruckt=        AktSeite
           ::::
        Until               ((AktSeite < StartPos) or (AktSeite = Gedruckt))
```

Diese Schleife ruft die Unterroutine (Drucken) auf, um die Seiten eines Dokuments vom Dokumentende her zu drucken (Rückwärtsdruck). Die *Until*-Bedingung besteht hier aus zwei Bedingungen, die durch **OR** (= oder) verknüpft sind. Trifft eine dieser beiden Bedingungen zu, wird die Schleife beendet. Achten Sie bitte auf die doppelten Klammern am Anfang und am Ende des Befehls!

```
3       Repeat
           ::::
           Repeat
             ::::
               Repeat
                 ::::
               Until        (Anzahl=5)
           Until            ((SeitenNr = 10) And (Var5 > 3)) //Und-Bedingung
        Until               ((Var5>10) Or (Var6="Ende"))    //Oder-Bedingung
```

Geschachtelte *Repeat*-Schleifen. Jede Schleife muß jeweils mit einer *Until*-Bedingung enden. Ist die dritte Schleife beendet, geht die Steuerung an die zweite Schleife über. Ist die zweite Schleife beendet, geht die Steuerung an die erste Schleife über. Jede Schleife kann mit *Break* oder *Next* vorzeitig beendet werden.

Weitere Hinweise siehe: Break, Next, Repeat

6.133 Use

Befehlsform: **Use**(*Makroname*)

Über diese Funktion können *Function*- und *Procedure*-Routinen des unter *Makroname* angegebenen Makros aufgerufen werden, ohne das betr. Makro komplett auszuführen. Da der Befehl nicht ausgeführt wird, sondern nur eine Verbindung zu dem angegebenen Makro herstellt, kann er in dem aufrufenden Makro an beliebiger Stelle definiert sein, wegen der Übersichtlichkeit aber am besten am Anfang eines Makros. Das angegebene Makro muß kompiliert sein (verwenden Sie in diesem Fall zusätzlich im aufzurufenden Makro *VarErrChk(Off!)*).

Parameter

Makroname Name des Makros, das *Function*- oder *Procedure*-Routinen enthält. Die Kombination von Konstanten und Variablen zum Erzeugen des Makronamens ist nicht erlaubt.

Weitere Hinweise siehe: Chain, Function, Procedure, Run

6.134 Value

Befehlsform: **Variable=Value**(*Funktion*)

Über *Value* können Sie einer Variablen die Kennung eines produktspezifischen Befehls zum Ausführen zuordnen. Dieser Befehl kann auch in Verbindung mit anderen Befehlen wie z. B. *Assign*, *Type*, *If* usw. verwendet werden (beachten Sie bitte auch Kapitel »5.17 Änderungen gegenüber WPWin 5.1/5.2«).

Parameter

Variable Übernahme des unter *Value* angegebenen produktspezifischen Befehls.

Funktion Jeder beliebige produktspezifische Befehl.

Beispiel

```
Befehl=Value(DraftZoom100)

Dieser Befehl ordnet der Variablen Befehl den produktspezifischen Befehl
DraftZoom100 zu.

Type(Value(Indent))

In diesem Beispiel wird Type in Verbindung mit Indent ausgeführt.
```

Weitere Hinweise siehe: PauseCommand, PauseKey, PauseSet

6.135 VarErrChk

Befehlsform: *Variable*=**VarErrChk**(*Status*)

Über diesen Befehl können Sie festlegen, wie bei der Kompilierung nicht definierte Variable behandelt werden sollen. Normalerweise erzeugen nicht definierte Variable beim Kompilieren eine Fehlermeldung. Es kommt nun aber öfters vor, daß in aufgerufenen Makros (z. B. mit *Chain* oder *Run*) mit Variablen aus dem aufrufenden Makro weitergearbeitet werden muß. Bei diesen Variablen handelt es sich in erster Linie um Variable in der Global- und in der Persist-Tabelle. Diese Variablen dürfen in dem aufgerufenen Makro nicht definiert werden, weil sonst die Datenübergabe nicht korrekt funktioniert. Sind in einem Makro zu verarbeitende Variable aber nicht definiert, erfolgt beim Kompilieren eine Fehlermeldung. Um diese Meldung zu unterdrücken und um das Makro zu kompilieren und zu speichern kann mit *VarErrChk* gearbeitet werden.

Parameter

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

Status

On!	Fehlende Variable erzeugen eine Fehlermeldung.
Off!	Fehlende Variable erzeugen keine Fehlermeldung. Verwenden Sie diese Option, wenn ein aufrufendes Makro Daten an ein aufgerufenes übergeben muß. In diesem Fall muß in dem aufgerufenen Makro am Anfang der Befehl *VarErrChk(Off!)* eingefügt werden.

Beispiel

GLOBAL.WCM

```
Application          (A1; "WordPerfect"; Default; "DE")
Global               (Feld1;Feld2)
Feld1=               "Hier ist Feld 1  "
Feld2=               "Hier ist Feld 2  "
Run                  ("GLOBAL1.WCM")
```

GLOBAL1.WCM

```
Application          (A1; "WordPerfect"; Default; "DE")
VarErrChk            (Off!)
Prompt               ("Befehl ""GLOBAL"" testen";Feld1+Feld2;;;)
Wait                 (30)
EndPrompt
```

In dem ersten Makro werden die Variablen *Feld1* und *Feld2* als globale Variable definiert. Dadurch können Sie in dem aufgerufenen Makro GLOBAL1.WCM weiterverarbeitet werden. In dem aufgerufenen Makro ist der Befehl *VarErrChk* erforderlich, da beim Kompilieren dieses Makros die im Prompt-Befehl angesprochenen Variablen noch nicht zur Verfügung stehen (ansonsten erfolgt eine Fehlermeldung).

Weitere Hinweise siehe: Assign

6.136 Wait

Befehlsform: **Wait**(*10tel Sekunden*)

Dieser Befehl bewirkt eine Makro-Unterbrechung für die angegebene Zeit, um z. B. eine bestimmte Meldung für längere Zeit am Bildschirm anzuzeigen, um die Makroausführung zu verzögern oder um beim Testen von Makros nach Fehlern zu suchen. Verwenden Sie beim Testen von Makros zusätzlich die Befehle *Speed* und *Display*, um auch die während des Makroablaufs angezeigten Menüs/Dialogfelder verfolgen zu können. Der eingegebene Wert wird als 10tel-Sekunden interpretiert. Je größer der Wert, um so länger die Unterbrechung, je kleiner der Wert, um so kürzer die Unterbrechung. Die Verwendung dieses Befehls ist an jeder beliebigen Stelle des Makros möglich. Zeichen, die Sie während der Wartezeit eintippen, werden ignoriert.

Parameter

10/tel Sekunden Beliebiger numerischer Wert bis maximal 600 (= 1 Minute). Das Makro wird für die angegebene Zeit unterbrochen.

Beispiel

```
Prompt              ("Fehler";"Eingabedaten falsch.";;;)
Wait                (50)
EndPrompt
```

Das Makro wird für fünf Sekunden unterbrochen, um die unter *Prompt* definierte Meldung am Bildschirm anzuzeigen.

6.137 While

Befehlsform: **While**(*Test*)

Dieser Befehl vereinfacht die Schleifenverarbeitung, indem er die nachfolgenden Befehle bis zum nächsten zugehörigen *EndWhile* so oft wiederholt, wie in der Bedingung *Test* angegeben wurde, sofern sie nicht vorher durch andere Bedingungen innerhalb der Scheife (z. B. If, Break) beendet wird. Die Werte für *Test* können Sie als Konstanten oder als Variable angeben. Auch das Mischen beider Möglichkeiten ist erlaubt. Sofern Sie mit Variablen arbeiten, achten Sie bitte darauf, daß diese vor der Ausführung des Befehls gültige Werte enthalten. Geben Sie diese entweder über Tastatur ein, oder ordnen Sie diese vor der Ausführung von *While* mit dem Befehl *Assign* zu. Auch innerhalb der Schleife können Sie Variable zuordnen, die sich auf die Bedingung auswirken.

Achtung: Falsche Werte oder falsche Bedingungen können zu unvorhersehbaren Ergebnissen führen (z. B. Endlosschleife).

Jede *While*-Struktur muß zwingend mit einem *EndWhile*-Befehl enden. Die dazwischen definierten Befehle werden aufgrund der unter *While* angegebenen Bedingungen wiederholt (= Schleife). Die Ausführung von *EndWhile* bewirkt den Rücksprung zu *While*. Dabei werden die Bedingungen entsprechend geprüft. Dieser Vorgang wird so lange wiederholt, bis die Schleife so oft abgearbeitet wurde, wie in *Test* unter *While* angegeben wurde.

Parameter

Test Eine oder mehrere Vergleichsoperationen.

Beispiel 1

```
1          Zähler=              1
           While                (Zähler<10)
              Zähler=           (Zähler+1)
              Type              ("*")
           EndWhile
```

Beachten Sie bitte auch das Beispiel unter *Until*.

```
2          Application          (A1;"WordPerfect";Default;"DE")
           Label                (Anfang)
           GetNumber            (Anfang;"Bitte Untergrenze eingeben";"1 x 1 errechnen")
           GetNumber            (Ende;"Bitte Obergrenze eingeben";"1 x 1 errechnen")
           If                   (Anfang>Ende)
              Prompt            (" F e h l e r";"Untergrenze ist größer als Obergrenze";1;;)
              Beep
              Wait              (30)
              EndPrompt
              Go                (Anfang)
           EndIf
           Zähler=              1

           While                (Anfang<Ende+1)
              While             (Zähler<11)
                 Ergebnis=      (Zähler*Anfang)
                 Type           NumStr(Zähler)
                 Type           (" * ")
                 Type           NumStr(Anfang)
                 Type           (" = ")
                 Type           NumStr(Ergebnis)
                 Zähler=        (Zähler+1)
                 HardReturn     ()
              EndWhile
              HardReturn        ()
              Anfang=           (Anfang+1)
              Zähler            1
           EndWhile
           HardReturn           ()
           Type                 ("Ende der Berechnung")
```

Mit diesem Makro wird ein beliebiges Einmaleins errechnet. Die beiden *While*-Befehle bilden hierbei eine äußere und eine innere Schleife:

Äußere Schleife: Die erste While-Schleife steuert Anzahl der zu druckenden Kolonnen (z. B. das Einmaleins von fünf bis neun), d. h. sie prüft, ob der eingegebene Bereich zwischen Unter- und Obergrenze einschließlich gedruckt wurde.

Innere Schleife: Diese Schleife druckt jeweils eine Kolonne (z. B. das Einmaleins von fünf). Ist diese Schleife beendet, geht die Kontrolle wieder an die äußere Schleife über.

Eine genaue Beschreibung dieses Beispiels finden Sie in Kapitel »9 Fallstudien«.

Beispiel 2

```
While               (True)
    SearchString    (SuchText)
    ::::
EndWhile
```

Die Schleife wird so lange wiederholt, bis der in *SuchText* angegebene Text nicht mehr vorhanden ist.

Weitere Hinweise siehe: EndWhile

6.138 WordCount

Befehlsform: *Variable*=**WordCount**(*Daten;WortLänge;Separator;Optionen*)

Dieser Befehl ermittelt die Anzahl der Wörter in einem String in Abhängigkeit der gewählten Parameter.

Parameter

Variable Diese Variable enthält nach der Abarbeitung des Befehls das Ergebnis.

Daten Wörter, die gezählt werden sollen.

WortLänge Länge der Wörter, die gezählt werden sollen. Wird der Parameter weggelassen oder enthält er einen Wert kleiner oder gleich 0, werden alle Wörter gezählt.

Separator Ein oder mehrere Zeichen, die zur Worttrennung verwendet werden. Wird der Parameter weggelassen, werden Leerstellen angenommen.

Optionen Festlegung der Art der Zählung. Wird der Parameter weggelassen, bestimmt der Parameter *WortLänge* die Art der Zählung. Standardmäßig wird CountWords! angenommen.

CountWords!	In Abhängigkeit des Parameters WortLänge.
CountShorter!	Zählen von Wörtern, die kürzer sind als in WortLänge angegeben.
CountLonger!	Zählen von Wörtern, die länger sind als in WortLänge angegeben.
ShortestLength!	Ermittelt die Länge des kürzesten Wortes. WortLänge wird ignoriert.
LongestLength!	Ermittelt die Länge des längsten Wortes. Wortlänge wird ignoriert.
AverageLength!	Ermittelt die durchschnittliche Wortlänge. Wortlänge wird ignoriert.

Beispiel

```
Ergebnis=          WordCount("WordPerfect 7 für Windows 95";;;CountWords!)
Ergebnis=          5,0

Ergebnis=          WordCount("WordPerfect 7 für Windows 95";5;;CountShorter!)
Ergebnis=          3,0

Ergebnis=          WordCount("WordPerfect 7 für Windows 95";5;;CountLonger!)
Ergebnis=          2,0

Ergebnis=          WordCount("WordPerfect 7 für Windows 95";5;;ShortestLength!)
Ergebnis=           1,0

Ergebnis=          WordCount("WordPerfect 7 für Windows 95";5;;AverageLength!)
Ergebnis=          4,8
```