5 Allgemeine Hinweise zur Programmierung

In diesem Kapitel finden Sie Tips, Tricks und weitere hilfreiche Informationen, die Ihnen die Arbeit bei der Programmierung etwas erleichtern. Machen Sie sich vor der Erstellung eines Makros zuerst mit den nachfolgend beschriebenen Redewendungen vertraut, damit Sie die Erklärungen und Hinweise in den restlichen Kapiteln dieses Buches, insbesondere in Kapitel »6 Makro-Programmierbefehle« auch verstehen. Die in dem kleinen Beispiel von Kapitel »3 Arbeiten mit Makros« verwendeten Makrobefehle gehören zu der Kategorie »produktspezifische Befehle«. Mit diesen Befehlen alleine können Sie aber noch keine leistungsfähigen Makros erstellen. Erst in Verbindung mit den Makro-Programmierbefehlen und den WordPerfect-Systemvariablen können Sie dieses Ziel erreichen.

5.1 Produktspezifische Befehle

Ein produktspezifischer Befehl ist ein Befehl, der während der Makro-Aufzeichnung über die Tastatur aufgrund eines ausgewählten Menüpunkts in das Makro eingefügt wird. Er steht symbolisch für einen ausgewählten Menüpunkt. Wählen Sie z. B. [*Format, Seite, Zentrieren*], wird der produktspezifische Befehl

CenterCurrentPage()

eingefügt. Die beiden Klammern am Ende des Befehls sind aus syntaktischen Gründen erforderlich. Für die Programmierung stehen Ihnen derzeit mehr als 1.800 produktspezifische Befehle zur Verfügung, d. h. fast jeder Menüpunkt der WordPerfect-Menüs entspricht einem produktspezifischen Befehl. Die Namen dieser Befehle wurden weitestgehend den Namen der zugehörigen Menüpunkten angepaßt.

Ein produktspezifischer Befehl kann ein einfacher Befehl sein wie z. B. *CenterCurrentPage()*, oder ein Befehl, der für die Ausführung weitere Informationen benötigt. Diese weiteren Informationen werden in den Feldern *Parameter* und *Auflistung* gespeichert.

Beispiel:

Hard Return	()
Type	(Text:"Das ist ein Test")
Display	(State:On!)

Wie Sie diese Befehle erstellen oder einfügen, entnehmen Sie bitte dem Kapitel »4 Makrobefehlsmanager«. Nach dem Aufrufen des Makrobefehlsmanagers und der Auswahl von [*WordPerfect-DE*] erhalten Sie eine alphabetische Auflistung der produktspezifischen Befehle. Die mit Fragezeichen gekennzeichneten Befehle am Anfang der Liste sind WordPerfect-Systemvariable, die ebenfalls in alphabetischer Reihenfolge aufgelistet werden (weitere Informationen hierzu finden Sie in Abschnitt »5.3 Variable«). Befehle, die Sie nicht sofort finden können, sind wahrscheinlich unter einem anderen Namen gespeichert. Möchten Sie z. B. zurück zum Dokumentanfang, ist dieser Befehl nicht unter *DokumentAnfang* zu finden, sondern unter *PosDocTop*. Achtung: Verschiedene Befehle sind nicht in der richtigen Reihenfolge einsortiert (z. B. bei Umlauten). Blättern Sie darum auch einige Anweisungen vor oder zurück.

Die Eingabe der produktspezifischen Befehle kann in Groß-, in Kleinbuchstaben oder aus einer Kombination von beiden erfolgen. Die Schreibweise hat keinen Einfluß auf die Makro-Ausführung.

Eine produktspezifische Befehlsfolge besteht aus dem eigentlichen Befehl, aus einem oder mehreren Parametern und evtl. aus Ausdrücken:

AbbreviationCreate (Template:CurrentDoc!)

Befehl

Ein Befehl ist die Anweisung zur Ausführung einer Funktion, in diesem Fall *AbbreviationCreate*. Ihm kann eine Produktkennung vorausgehen (z. B. *A.AbbreviationCreate*), die durch einen Punkt von dem Befehl getrennt werden muß. Diese Produktkennung können Sie weglassen, wenn Sie am Beginn des Makros den Befehl *Application*, der die Produktkennung beinhaltet, verwenden. Leerstellen innerhalb eines Befehls (z. B. *Abbreviation Create*) führen zu einem Fehler beim Kompilieren und sind darum nicht erlaubt.

Parameter

Ein Befehl kann einen oder mehrere Parameter enthalten, wobei für jeden Parameter ein eigener Name reserviert ist. Viele Befehle brauchen keine Parameter, sie enden in der Regel mit **()**, wie z. B. *HardReturn()*. Parameter sind eine oder mehrere Informationen, die den Befehl vervollständigen, damit er ordnungsgemäß ausgeführt werden kann. Sie entsprechen in den meisten Fällen einer Option eines Untermenüs. Bei *AbbreviationCreate* könnte man z. B. *Template* auswählen. Bei anderen Parametern müssen Sie vielleicht manuell weitere Angaben ergänzen (z. B. einen Dateinamen).

Alle Parameter eines Befehls werden durch Klammern eingeschlossen. Jeder Parametername endet mit einem Doppelpunkt. Sind mehrere Parameter möglich, werden diese jeweils durch ein Semikolon getrennt. Parameternamen (z. B. *Template*:) sind Lesbarkeitswörter und können auch weggelassen werden:

AbbreviationCreate(CurrentDoc!)

Ausdrücke

Ausdrücke sind Informationen aus der Liste [Auflistungen], die den zugehörigen Parameter vervollständigen. Jeder Ausdruck muß mit einem Semikolon enden (ist nur ein Parameter vorhanden, entfällt das Semikolon). Ausdrücke werden durch einen Doppelpunkt vom zugehörigen

Parametername getrennt. Z. B. muß bei der Auswahl von *Template* angegeben werden, welche Vorlage angesprochen werden soll, z. B. *CurrentDoc!*. Danach müßte die Befehlszeile wie folgt aussehen:

AbbreviationCreate(Template:CurrentDoc!)

Produktspezifische Befehle können/müssen in Verbindung mit einem oder mehreren Parametern verwendet werden, die teilweise vorhanden sein müssen bzw. die wahlfrei sind. Wahlfreie Parameter müssen, sofern sie nicht benötigt werden, bei verschiedenen Befehlen durch ein Semikolon ersetzt werden. Wird ein Semikolon vergessen, erfolgt bei der Kompilierung eine Fehlermeldung.

Verschiedene produktspezifische Befehle können während der Makro-Aufzeichnung nicht eingefügt werden, da sie weder über ein WordPerfect-Menü noch über die Tastatur eingegeben werden können. Zum Einfügen solcher Befehle können Sie entweder den Makrobefehlsmanager verwenden, oder Sie können solche Befehle über das Tastaturlayout einer bestimmten Tastenkombination zuordnen. Zum Einfügen des Befehls brauchen Sie dann bei der Makro-Aufzeichnung nur noch diese Tastenkombination zu drücken, der der Befehl zugeordnet wurde (z. B. [Strg-Alt-P]).

5.2 Makro-Programmierbefehle

Erst in Verbindung mit den Makro-Programmierbefehlen können Sie leistungsfähige Makros bzw. Programme erstellen. Diese Befehle sind namentlich und in der Auswirkung den vergleichbaren Befehlen höherer Programmiersprachen (C, Pascal, Cobol, Basic, usw.) ähnlich. Durch diese Befehle können flexible Makros erstellt werden, die aufgrund von Bedingungen Entscheidungen treffen, Unterroutinen oder andere Makros aufrufen, Verzweigungen vornehmen, Rechenoperationen durchführen, Schleifen abarbeiten usw.

Auch Makro-Programmierbefehle werden in Verbindung mit einem oder mehreren Parametern verwendet, die teilweise vorhanden sein müssen bzw. die wahlfrei sind. Wahlfreie Parameter müssen, sofern sie nicht benötigt werden, bei verschiedenen Befehlen durch ein Semikolon ersetzt werden. Wird ein Semikolon vergessen, erfolgt bei der Kompilierung eine Fehlermeldung. Die Eingabe der Makro-Programmierbefehle kann in Groß-, in Kleinbuchstaben oder aus einer Kombination von beiden bestehen. Die Schreibweise hat keinen Einfluß auf die Makro-Ausführung. Werden weitere Parameter zur Ausführung benötigt, kann es sich um folgende Arten von Parametern bzw. Datenkategorien handeln:

Numerischer Ausdruck

Ein numerischer Ausdruck kann eine Variable mit numerischem Inhalt (siehe Abschnitt »5.3 Variable«) oder eine numerische Konstante (siehe Abschnitt »5.4 Konstanten«) sein.

Beispiel:

120	Ganzzahl
2,54	Dezimalzahl
(RF7;(5*RF2)/4)	Rechenformel, (RF7 und RF2 sind Variable mit numerischem Inhalt.)

Zeichenausdruck

Ein Zeichenausdruck kann eine Variable mit alphanumerischem Inhalt (siehe Abschnitt »5.3 Variable«) oder eine alphanumerische Konstante (siehe Abschnitt »5.4 Konstanten«) sein.

Beispiel: »Karl Meier« Konstante

Text1+« Verlag« Variable und Konstante werden zu einem Ausdruck zusammengefaßt. Enthält *Text1* den Inhalt **Markt & Technik**, ergibt sich hieraus der Ausdruck **Markt &Technik Verlag**.

Maßeinheiten

Nur Ziffern, eine Maßeinheit und ein Dezimalkomma bzw. ein Dezimalpunkt sind erlaubt. Die Variable wird als numerisches Datenfeld angesehen. Als Maßeinheiten (= Abkürzungszeichen) sind erlaubt:

c	=	Zentimeter
i	=	Inches
p	=	Punkte (72 Punkte pro Inch)
w	=	WordPerfect Einheiten (1200 Einheiten pro Inch)
"	=	Inches

Beispiel:

1,5c	1,5 Zentimeter
2i	2 Inches

In Verbindung mit den Maßeinheiten können Sie auch mathematische Funktionen ausführen (siehe Abschnitt »5.10 Mathematische Funktionen«). Bei der Verarbeitung zweier unterschiedlicher Maßeinheiten in Verbindung mit mathematischen Operationen enthält das Ergebnis immer die Maßeinheit des linken Operanden.

Wenn über den Befehl *DefaultUnits* eine Maßeinheit vorgegeben wurde, kann das Abkürzungszeichen bei den nachfolgenden Maßeinheiten weggelassen werden. Wurde weder bei der Eingabe ein Abkürzungszeichen verwendet, noch unter *DefaultUnits* eine Maßeinheit eingegeben, werden automatisch *WordPerfect Einheiten* (1/1200 Inches) angenommen.

Verwechseln Sie bei mathematischen Funktionen bitte nicht numerische Ausdrücke mit Maßeinheiten, sonst erhalten Sie falsche Ergebnisse!

Bedingungs-Ausdruck

Oftmals sind in Makros Daten zu vergleichen oder Bedingungen zu prüfen. Je nach Ergebnis muß das Makro unterschiedlich weiterarbeiten. Bei der Prüfung einer Bedingung gibt es immer nur zwei Möglichkeiten: Entweder die Bedingung trifft zu (= True) oder sie ist falsch (= False).

Werden Variable und/oder Konstanten miteinander verglichen, müssen beide denselben Datentyp aufweisen.

Beispiel:
```
If (Firma="WordPerfect")
If (Summe>1000)
If (Summe=Zwischensumme-100)
```

Definieren Sie je Zeile immer nur einen Befehl. Das Makro bleibt dadurch übersichtlich, neue Befehle können einfacher eingefügt werden. Eine genaue Beschreibung der Makro-Programmierbefehle mit vielen Beispielen finden Sie in Kapitel »6 Makro-Programmierbefehle«.

5.3 Variable

Eine Variable ist ein Speicherfeld, das Daten aufnehmen, speichern und abgeben kann. Auch die Durchführung von Berechnungen ist möglich. Die Daten können über verschiedene Befehle einem Feld (= Variable) zugeordnet bzw. aus einem Feld abgerufen werden. Innerhalb eines Makros oder eines Mischvorgangs können die Variablen-Inhalte (wenn erforderlich) beliebig oft gewechselt werden. Wird einer Variablen ein neuer Wert zugewiesen, wird der bisherige Inhalt dadurch überschrieben und ist unwiederbringlich verloren. Variable können Sie mit den Schubladen eines Schranks vergleichen, die alle namentlich gekennzeichnet sind. In jede einzelne Schublade können Sie etwas hineinlegen (= Daten speichern) und auch wieder herausnehmen, wenn Ihnen der Schubladenname bekannt ist. Damit keine Verwechslungen vorkommen, muß jeder Name eindeutig sein, d. h. er darf innerhalb eines Makros für ein und denselben Zweck nur einmal definiert, aber mehrmals angesprochen werden. Es wird keine Unterscheidung zwischen Groß- und Kleinbuchstaben getroffen. Die Variablennamen *Eingabe*, *eingabe* oder *EiNgAbE* sind darum identisch. Ausnahmen werden bei dem jeweiligen Befehl separat erklärt.

Der Variablename darf nicht in Hochkommata eingeschlossen werden, weil er sonst als alphanumerische Konstante erkannt wird.

WordPerfect unterscheidet zwischen Systemvariablen und benutzerdefinierten Variablen:

Systemvariable
: Systemvariable sind von WordPerfect benutzte Speicherfelder, die Sie nur lesen können, d. h. Sie können diesen Feldern keine Daten zuordnen. Die Namen sind fest vorgegeben und können von Ihnen nicht verändert werden. Die Inhalte dieser Felder beziehen sich immer auf den aktuellen WordPerfect-Status zum Zeitpunkt der Abfrage. Welche Bedeutung sie haben, können Sie der Auflistung im Makrobefehlsmanager entnehmen. Sie zählen zu den produktspezifischen Befehlen und enthalten vor dem Variablennamen ein Fragezeichen. Wählen Sie im Makrobefehlsmanager [*Befehlsart, WordPerfect-DE*].

Benutzerdefinierte Variable
: Alle Variablen, die Sie während der Makro-Erstellung selbst definieren, sind sog. benutzerdefinierte Variable. Jeder Variablename muß eindeutig sein. Der Inhalt dieser Variablen wird teilweise von Ihnen selbst bestimmt (z. B. durch eine Zuordnung über den Befehl *Assign*), teilweise

wird er durch bestimmte Makro-Programmierbefehle während der Makro-Ausführung zugeordnet (z. B. Ergebnis einer Berechnung oder durch eine Konvertierung über *NumStr*, *StrNum* usw.).

Die Art der benutzerdefinierten Variablen richtet sich nach den zu speichernden Daten. Zwei Kategorien von Variablen werden unterschieden:

Text (= String) — Hierbei handelt es sich um beliebige Zeichen. Buchstaben, Ziffern und Sonderzeichen (z. B. aus den WordPerfect-Zeichensätzen) sind erlaubt. Die Variable wird als alphanumerisches Datenfeld angesehen.

Zahlen — In dieser Gruppe sind nur Ziffern *evtl.* in Verbindung mit dem Dezimalkomma bzw. Dezimalpunkt und mathematischen Vorzeichen erlaubt. Die Variable wird als numerisches Datenfeld angesehen.

Im folgenden Beispiel wird über den Befehl *Assign* der Variablen *Name* der Inhalt der Variablen *Eingabe* zugeordnet. Zwei verschiedene Möglichkeiten stehen zur Verfügung:

```
Assign      (Name;Eingabe)    oder
Name:=      Eingabe           oder
Name=       Eingabe
```

Die Daten in der Variablen *Eingabe* wurden an einer anderen Stelle des Makros, z. B. über einen *GetString*-Befehl über die Tastatur eingegeben. Einer Variablen können Sie auch einen festen Text (= Konstante) zuordnen:

```
Assign      (Name;"Karl Meier")    oder
Name:=      "Karl Meier"           oder
Name=       "Karl Meier"
```

Beachten Sie hierzu bitte den Abschnitt »5.4 Konstanten«.

Wenn die zuzuordnenden Daten nicht nur aus Ziffern (einschl. Vorzeichen, Dezimalkomma bzw. -punkt oder Maßeinheiten) bestehen, müssen Sie diese durch Hochkommata einschließen, ansonsten werden die Daten als Variable-Name interpretiert, was zu Fehlern bei der Kompilierung führt.

Müssen Variable gesondert definiert und für den Makrostart mit bestimmten Daten vorbesetzt (initialisiert) werden, sollten Sie die Definition der Variablen ggf. am Beginn des Makros in alphabetischer Reihenfolge vornehmen, damit das Makro übersichtlich bleibt und leicht bearbeitet werden kann.

In einer Variablen kann zu einer Zeit immer nur ein Wert gespeichert werden. Wird der Variablen ein neuer Wert zugewiesen, geht der bisherige Wert dieser Variablen unwiederbringlich verloren.

Es können in einem Makro nur die Variablen ordnungsgemäß weiterverarbeitet werden, denen vor der Ausführung eines Befehls, der Bezug auf eine Variable nimmt, bereits ein Wert zugewiesen wurde. Wird eine Variable angesprochen, die zuvor nicht definiert wurde, erfolgt beim Kompilieren des Makros eine Fehlermeldung, sofern nicht mit *VarErrChk* gearbeitet

wurde. Eine Wertzuweisung erfolgt in der Regel durch Befehle wie z. B. *Assign*, *Integer*, *GetString* usw. Nach dem Makroende werden in der Regel automatisch alle Variablen gelöscht, sofern sie nicht in Verbindung mit *Global*, *Persist* oder *PersistAll* definiert wurden. Variable werden wie folgt definiert:

Name=	("Karl Meier")	= Zuweisung: **Karl Meier**
Ergebnis=	**Integer**(12345,67)	= Zuweisung: **12345**
Ergebnis=	**Fraction**(12345,67)	= Zuweisung: **67**
GetNumber	(DMBetrag;"Bitte Kaufpreis eingeben";"Verkauf")	= Der eingebene Verkaufspreis wird der Variablen *DMBetrag* zugewiesen. Die eingebenen Daten müssen numerisch sein.

Nach der Ausführung solcher oder ähnlicher Befehle, wird eine Variable erstellt (sofern sie nicht mit *Assign* definiert wurde) und mit einem Wert versehen. Ab diesem Zeitpunkt kann sie in dem aktuellen Makro bis zum Beenden dieses Makros verwendet werden bzw. bis sie gelöscht wird.

Variable können nur in dem Makro angesprochen werden, in dem sie definiert wurden. Makros, die mit CHAIN oder RUN aufgerufen werden, können vielleicht nicht auf Variable der aufrufenden Makros zurückgreifen, d. h. eine Datenübergabe von einem Makro in ein anderes ist nicht möglich (Ausnahme: siehe unten). Grund: Jedes Makro muß, bevor es aufgerufen werden kann, kompiliert werden. Bei diesem Kompilieren wird geprüft, ob eine Variable innerhalb des aktuellen Makros definiert wurde (siehe oben). Ist dies nicht der Fall, erfolgt eine Fehlermeldung, und das Makro wird nicht in einen lauffähigen Zustand gebracht. Dies wiederum ist aber nötig, damit es über CHAIN oder RUN aufgerufen werden kann.

Ausnahme: Makros, die von anderen Makros aufgerufen werden, übernehmen oft Daten aus dem aufrufenden Makro. Die hierfür erforderlichen Variablen dürfen in dem aufgerufenen Makro nicht definiert werden, da Sie in dem aufrufenden Makro bereits als globale oder als Persist-Variable definiert werden müssen. Beim Kompilieren eines aufgerufenen Makros können dadurch beim Kompilieren Fehler angezeigt werden, weil diese Variablen nicht definiert sind. Verwenden Sie in diesem Fall *VarErrChk(Off!)*, um die Überprüfung auszuschalten.

Variable können an jeder Stelle des aktuellen Makros aufgerufen und ggf. inhaltlich beliebig oft geändert werden, sofern sie vorher definiert wurden (siehe oben). Hierbei wird die Datenkategorie (Text/Zahlen/Maßeinheiten, siehe oben) berücksichtigt, denn bestimmte Befehle können nur bestimmte Datenkategorien verarbeiten. Haben Sie beispielsweise einer Variablen einen Text zugeordnet, können Sie diese Variable in keinem Befehl verwenden, der in dieser Variablen numerische Daten erwartet wie z. B. *Integer*, *NumStr*, *Fraction* usw.

Mit den folgenden Befehlen können Sie z. B. Variablen Werte zuweisen:

Assign	ForEach	NumStr	SubChar
CharLen	Fraction	StrLen	SubStr

CharPos	Integer	Strnum
For	Menu	StrPos

5.4 Konstanten

Konstanten sind fixe Werte, die einer Variablen zugeordnet werden oder die bei Bedingungsabfragen zur Entscheidungsfindung verwendet werden können. Es gibt numerische (= numerische Ausdrücke) und alphanumerische (= Zeichen-Ausdrücke) Konstanten. Letztere werden auch als String bezeichnet und sind immer durch Hochkommata eingeschlossen. Alphanumerische Konstanten können jedes beliebige Zeichen enthalten, also Ziffern, Buchstaben und Sonderzeichen. Numerische Konstanten dürfen nur Ziffern, den Dezimalpunkt bzw. das Dezimalkomma und sonstige mathematische Vorzeichen enthalten. Sie dürfen nicht durch Hochkommata eingeschlossen werden. Konstanten können während einer Makroausführung oder eines Mischvorgangs nicht verändert werden. Sind trotzdem Änderungen erforderlich, müssen Sie das Makro zum Ändern in ein Dokumentfenster laden.

Assign	(Name;"Karl Meier")	oder
Name:=	"Karl Meier"	oder
Name=	"Karl Meier"	

In diesem Beispiel enthält die Variable immer die zugeordnete Zeichenfolge, sofern Sie nicht durch einen anderen Befehl zu einem späteren Zeitpunkt überschrieben wird. In Hochkommata eingeschlossene Zeichen werden als alphanumerische Strings bezeichnet. Nach der Ausführung des Befehls enthält die Variable *Name* den Inhalt *Karl Meier*.

Zinsen= 12345,67

Der Variablen *Zinsen* wird ein numerischer Wert zugeordnet. Mit numerischen Werten können Sie innerhalb der Makros mit verschiedenen Befehlen Rechenoperationen durchführen. Numerische Werte dürfen nur Ziffern, ein mathematisches Vorzeichen (+ /-) sowie das Dezimalkomma bzw. den Dezimalpunkt enthalten. Sie dürfen nicht durch Hochkommata eingeschlossen werden. Nach der Ausführung des Befehls enthält die Variable *Zinsen* den numerischen Inhalt ***12345,67***.

Zinsen= »12345,67«

In diesem Beispiel wird der Variablen *Zinsen* ein String zugeordnet. Mit dieser Variablen sind keine Rechenoperationen möglich, da mit Strings nicht gerechnet werden kann. Nach der Ausführung des Befehls enthält die Variable *Zinsen* den alphanumerischen Inhalt »***12345,67***«.

Text= »Beschreibung des »StrNum«-Befehls.«

Enthalten Konstanten Hochkommata, müssen diese jeweils doppelt angegeben werden, ansonsten erhalten Sie beim Kompilieren einen Feh-

lerhinweis. In diesem Beispiel wird über *Prompt* folgender Text angezeigt:

Beschreibung des »StrNum«-Befehls.

Wenn die zuzuordnenden Daten nicht nur aus Ziffern bestehen, müssen Sie diese durch Hochkommata einschließen, ansonsten wird der Begriff als Variable-Name interpretiert, was zu Fehlern bei der Kompilierung führt.

Müssen Variable gesondert definiert und mit bestimmten Daten vorbesetzt (initialisiert) werden, sollten Sie die Definition der Variablen am Beginn des Makros in alphabetischer Reihenfolge vornehmen, damit das Makro übersichtlich bleibt und leicht bearbeitet werden kann. Das gilt insbesondere für das Ansprechen von Dateinamen. Geben Sie darum ein und denselben Dateinamen nicht mehrmals in einem Makro an unterschiedlichen Stellen ein, sondern weisen Sie diesen am Makroanfang einer Variablen zu. Das hat den Vorteil, daß beim Ändern des Dateinamens nur die Variable geändert werden muß und nicht mehrere Definitionen innerhalb des Makros. Besonders bei umfangreichen Makros sollte diese Regel eingehalten werden.

Beispiel:

```
Laufwerk=       "C:"
Pfad=           "\WPWIN7\ADRESSEN\"
Dateiname=      "FIRMEN.DAT"
```

Achten Sie auf die korrekte Verwendung des Backslash (\).

Werden Laufwerk, Pfad und Dateiname im Makro mehrmals benötigt, sollten Sie immer als Variable und niemals als Konstante verwendet werden. Änderungen sind dann nur an einer Stelle erforderlich, nämlich bei der Definition der Variablen, und nicht an mehreren Stellen innerhalb des Makros (Fehlerquelle!). Die Variablen könnten z. B. zum Öffnen einer Datei wie folgt verwendet werden:

FileOpen(Laufwerk+Pfad+Dateiname**)**
entspricht
C:\WPWIN7\ADRESSEN\FIRMEN.DAT

5.5 Entscheidungen

Durch die Verwendung von Entscheidungs-Befehlen, die in erster Linie bei der Makro-Steuerung in Verbindung mit den Befehlen *Case*, *Case Call* und *CaseOf* verwendet werden, wird der Makroablauf beeinflußt. Würden keine Bedingungen (siehe unten) oder Entscheidungen zur Makrosteuerung verwendet, würde das Makro immer dieselben Befehle ausführen, beginnend am Makroanfang, bis zum Makroende oder bis zu einem der Befehle *Quit* oder *Return*. In Abhängigkeit des Ergebnisses einer Bedingung (»wahr« oder »nicht wahr«) können Sie bestimmte Aktionen ausführen, d. h. das Makro führt je nach Entscheidung jeweils andere Arbeiten aus. Trifft eine Entscheidung zu (d. h. die Bedingung ist »wahr«), wird die dafür vorgesehene Anweisung ausgeführt.

Eine Entscheidung gilt als nicht erfüllt, wenn das Ergebnis einer Prüfung »nicht wahr« ist. In diesem Fall wird die Anweisung ausgeführt, die als *Default Label* definiert wurde.

Weitere Informationen hierzu finden Sie in Kapitel »6 Makro-Programmierbefehle« bei den Befehlen *Case* und *Case Call* und in Abschnitt »5.10 Mathematische Funktionen«.

5.6 Bedingungen

Durch die Verwendung von Bedingungs-Befehlen, die in erster Linie bei der Makro-Steuerung in Verbindung mit Befehlen wie z. B. *For*, *If*, *Repeat* und *While* verwendet werden, wird der Makroablauf entscheidend beeinflußt. Würden keine Bedingungen oder Entscheidungen (siehe oben) zur Makrosteuerung verwendet, müßte das Makro immer dieselben Befehle ausführen, beginnend am Makroanfang, bis zum Makroende oder bis zu einem der Befehle *Quit* oder *Return*. In Abhängigkeit des Ergebnisses einer Bedingung (»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 z. B. die unmittelbar auf *If* folgenden Befehle bis zum nächsten *Else* oder, wenn kein *Else* vorhanden ist, bis zum *EndIf* ausgeführt, bei den anderen genannten Befehlen wird aufgrund einer Bedingung die Schleife beendet oder wiederholt.

Eine Bedingung gilt als nicht erfüllt, wenn das Ergebnis einer Prüfung »nicht wahr« ist. In diesem Fall werden z. B. bei einer *If*-Bedingung die dem Else folgenden Befehle bis zum 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 zugehörigen *EndIf* weiter. Ausnahmen bestehen bei geschachtelten *If*-Bedingungen. Bei *For*, *ForNext*, *Repeat* oder *While* wird die zugehörige Schleife verlassen.

Bei der Mehrzahl dieser Befehle werden bei den Bedingungen meistens zwei oder mehr Variablen und/oder Konstanten miteinander verglichen, um eine Entscheidung zu treffen. Bei diesen Vergleichen können immer nur Variable derselben Art verwendet werden. Es ist nicht erlaubt, eine numerische Variable mit einer String-Variablen zu vergleichen oder einen String mit einer numerischen Konstanten. Bei der Makroausführung würde in diesem Fall ein Fehler auftreten. Vor einem Vergleich muß darum sichergestellt sein, daß die zu vergleichenden Operanden denselben Datentyp aufweisen.

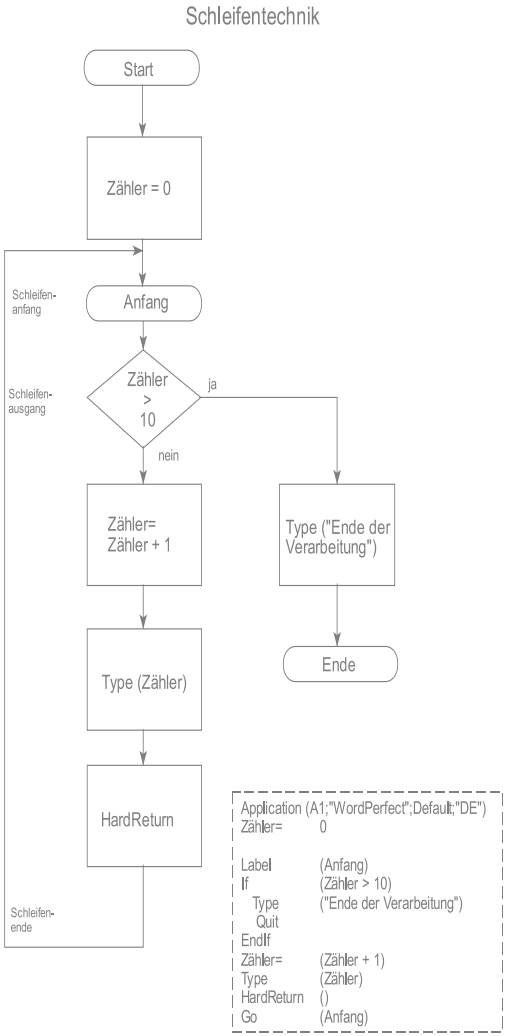
Weitere Informationen hierzu finden Sie in Kapitel »6 Makro-Programmierbefehle« bei den Befehlen *For*, *ForNext*, *If*, *Repeat* und *While*.

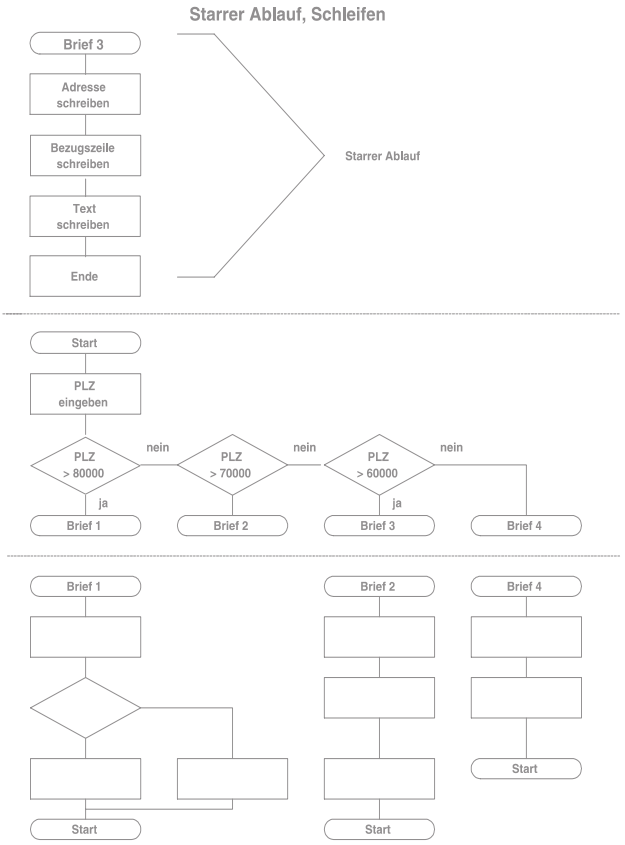
5.7 Schleifen

Eine Schleife bewirkt den ein- oder mehrmaligen Durchlauf ein und derselben Befehlsfolge in Verbindung mit Variablen, die in der Regel bei jedem Durchlauf teilweise andere Daten enthalten können. Innerhalb eines Makros können beliebig viele Schleifen auftreten, wobei auch eine oder mehrere Schleifen geschachtelt sein können. Dies bedeutet: Innerhalb einer Schleife ist eine weitere Schleife enthalten, die wiederum eine Schleife enthalten kann usw. Eine Schleife beginnt immer mit einem Label (siehe auch unten) und endet entweder mit einem *Return*- oder mit einem *Go*-Befehl.

Der Eintritt in eine Schleife kann auch durch die Befehle *Call*, *Case*, *Case Call* oder *Go* eingeleitet werden. Das Verlassen der Schleife muß durch Zutreffen einer Bedingung erfolgen, um keine Endlosschleife (= Loop) zu erzeugen. Sollte das dennoch einmal geschehen sein, können Sie das Makro durch Drücken von (Esc) jederzeit abbrechen. Welche Art der Schleifenverarbeitung Sie benutzen möchten, hängt von den Arbeiten ab, die das Makro ausführen muß.

Zusätzlich zu den zuvor beschriebenen Befehlen leiten auch die Befehle *For*, *ForEach*, *ForNext*, *Repeat* oder *While* eine Schleife ein. Das Schleifenende wird hier durch die Befehle *EndFor*, *Until* und *EndWhile* gekennzeichnet. Das Schleifenende wird auch hier durch eine Bedingung herbeigeführt. Die genannten Befehle können Sie auch schachteln und dabei auch untereinander kombinieren.





Einfache IF-Schleife:

```
Assign        (Zähler;0)
Label         (Anfang)                                    //Schleifenanfang
If            (Zähler=3)
Quit                                                      //Schleifenausgang
EndIf
Assign        (Zähler;Zähler+1)
Drucken:=     NumStr(Zähler)
Type          ("Das ist Schleifendurchlauf Nr.:")
Type          (Drucken)
HardReturn    ()
Go (Anfang)                                               //Schleifenende
```

Dieses Makro druckt die Zeilen:

```
Das ist Schleifendurchlauf Nr.: 1
Das ist Schleifendurchlauf Nr.: 2
Das ist Schleifendurchlauf Nr.: 3
```

Schleifen wiederholen also immer wiederkehrende Befehle so oft, bis eine bestimmte Bedingung zutrifft, in diesem Fall, bis die Variable *Zähler* den Wert 3 enthält. Das Makro wird dann beendet. Schleifen in Verbindung mit *Go* sind die für Anfänger am einfachsten zu handhabende Schleifen. Werden zu viele *Go* verwendet, wird das Makro unübersichtlich, da hierdurch ein sog. »Spaghetti-Code« erzeugt wird, der bei späteren Modifizierungen nur schwer zu »entwirren« ist. Versuchen Sie darum nach Möglichkeit andere, leistungsstärkere Befehle zu verwenden (*Call*, *Case Call*, *For*, *ForEach*, *ForNext*, *Repeat* oder *While*). Durch strukturiertes Programmieren können Sie die meisten *Go*-Befehle vermeiden.

Der *If*-Befehl kontrolliert in dem gezeigten Beispiel die Anzahl der Schleifendurchgänge und beendet die Schleife, wenn eine bestimmte Bedingung zutrifft, in diesem Fall: Wenn die Variable *Zähler* den Wert *3* enthält. Wird dieser Befehl vergessen, läuft das Makro endlos! Die o. g. Zeilen werden dann so oft gedruckt, bis Sie das Makro manuell abbrechen. Ohne Schleifenverarbeitung müßten Sie folgendes programmieren, um dasselbe Ergebnis zu erhalten:

```
Assign        (Zähler;0)
Assign        (Zähler;Zähler+1)
Type          ("Das ist Schleifendurchlauf Nr.:")
Type          (Zähler)
HardReturn    ()
Assign        (Zähler;Zähler+1)
Type          ("Das ist Schleifendurchlauf Nr.:")
Type          (Zähler)
HardReturn    ()
Assign        (Zähler;Zähler+1)
Type          ("Das ist Schleifendurchlauf Nr.:")
Type          (Zähler)
HardReturn    ()
```

Stellen Sie sich den Aufwand vor, Sie müßten nicht nur 3, sondern 500 Zeilen drucken!

Geschachtelte Schleife:

```
::::
While              (Anfang<Ende+1)              //Beginn äußere Schleife
   While           (Zähler<11)                  //Beginn innere Schleife
      Ergebnis=    (Zähler*Anfang)
      Type         (Zähler)
      Type         (" * ")
      Type         (Anfang)
      Type         (" = ")
      Type         (Ergebnis)
      Zähler=      (Zähler+1)
      HardReturn   ()
   EndWhile                                     //Ende innere Schleife
   HardReturn      ()
   Anfang=         (Anfang+1)
   Zähler=         1
EndWhile                                        //Ende äußere Schleife
HardReturn         ()
Type               ("Ende der Berechnung")
::::
```

Diese Befehle sind einem Makro entnommen, mit dem ein beliebiges Einmaleins errechnet werden kann (detaillierte Beschreibung siehe Kapitel »9 Fallstudien«). Die beiden *While*-Befehle bilden hierbei eine äußere und eine innere Schleife:

Äußere Schleife: Die erste *While*-Schleife steuert die Anzahl der zu druckenden Kolonnen, 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.

Die äußere Schleife übernimmt hierbei die Hauptsteuerung. Die innere Schleife wird so lange ausgeführt, bis die Bedingung der äußeren Schleife zutrifft. Ist das der Fall, wird hinter dem letzten *EndWhile*-Befehl weitergearbeitet. Weitere Informationen hierzu finden Sie in Kapitel »6 Makro-Programmierbefehle« bei den Befehlen *Case*, *Case Call*, *For*, *ForEach*, *ForNext*, *If*, *Repeat* und *While*.

5.8 Unterroutinen

Unterroutinen sind ein oder mehrere zusammengefaßte Makro-Befehle oder Strukturen, die durch die Befehle *Go*, *Call*, *Case* oder *Case Call* zur Ausführung gebracht werden. Die Anzahl der Unterroutinen in einem Makro ist nicht begrenzt. Sie sparen hierdurch das mehrmalige Definieren eines oder mehrerer identischer Befehle bzw. Befehlsfolgen an unterschiedlichen Stellen eines Makros. Eine Unterroutine können Sie von jeder beliebigen Stelle eines Makros aus aufrufen bzw. dorthin verzweigen. Unterroutinen können entweder verwendet werden um Makros übersichtlicher zu gestalten (strukturieren des Makros) oder um Befehlsfolgen, die in einem Makro mehrmals vorkommen, nur einmal zu definieren. Das hat auch den Vorteil, daß Änderungen oder Ergänzungen nur einmal in der betreffenden Unterroutine durchgeführt werden müssen, wodurch zusätzliche Fehlerquellen minimiert werden. Je nach Art des Aufrufs müssen die Unterroutinen wieder mit *Go* oder mit *Return* beendet werden, um Fehler bei der Ausführung auszuschließen. Danach erfolgt entweder eine Verzweigung zu einem weiteren Label oder die Rückkehr zum aufrufenden Befehl:

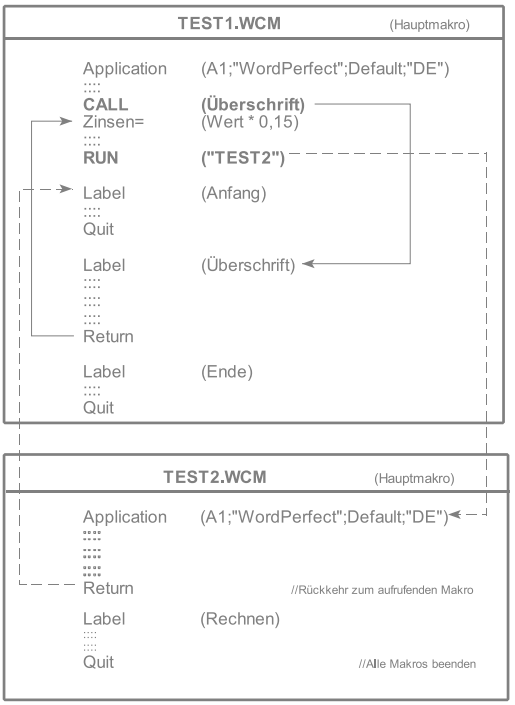
Verzweigung Der betreffende Label wird z. B. mit *Case*, *Go* oder *On...* zur Ausführung gebracht. Dem letzten Befehl der Unterroutine muß wiederum ein *Go*-Befehl folgen, es sei denn, das Makro wird innerhalb oder am Ende der Routine beendet (mit *Quit* oder *Return*) bzw. die Routine wird vorzeitig verlassen. Hierdurch wird gewährleistet, daß nur ganz bestimmte Befehle, nämlich die zwischen dem *Label*- und dem letzten *Go*-Befehl, ausgeführt werden, sofern nicht durch gesetzte Bedingungen vom Ablauf abgewichen wird.

Aufruf Bei Verwendung der Befehle *Call*, *Case Call* oder *On..Call* wird die aufgerufene Unterroutine ausgeführt. Eine Unterroutine beginnt immer mit einem Label und muß mit einem Return enden. Zur Ausführung kommen die Befehle, die zwischen *Label* und dem zugehörigen *Return* defi-

niert wurden, es sei denn, die Unterroutine wird mit *Quit* oder *Return* aufgrund einer zutreffenden Bedingung vorzeitig beendet. Der Aufruf einer Unterroutine wird so interpretiert, als wären die Befehle der Unterroutine an der aufrufenden Stelle definiert. Nach der Ausführung der Befehle erfolgt ein Rücksprung. Es werden die dem aufrufenden Befehl folgenden Befehle ausgeführt.

Jede Unterroutine muß durch einen eindeutigen Namen mit dem Befehl *Label* gekennzeichnet sein. Sind doppelte Namen vorhanden oder wurden Namen vergessen, erfolgt bei der Kompilierung eine Fehlermeldung. Sie können nur Unterroutinen des aktuellen Makros aufrufen, also keine Unterroutinen in Makros, die mit *Chain* oder *Run* aufgerufen werden. Demzufolge können Sie in unterschiedlichen Makros gleiche Label-Namen verwenden. Beachten Sie bitte auch die Befehle *Include* und Use.

Unterprogrammtechnik



Regeln:

Call ——→	Label muß im selben Makro vorhanden sein.
Run – – –→	Ruft ein eigenständiges Makro auf.
Return	Kehrt zum aufrufenden Befehl / Makro zurück.
Quit	Beendet **komplette** Makroverarbeitung.

Beispiel:

```
Label               (Anfang)                        //Label Anfang.
::::::
Call                (Drucken)                       //Aufruf der Unterroutine Drucken.
Prompt              (" ";"Text wurde gedruckt";;;)  //Meldung anzeigen.
Wait                (20)                            //Meldung für 20/10 Sekunden anzeigen.
Go                  (Öffnen)                        //Verzweigung zu dem Label Öffnen.
::::::
Label               (Drucken)                       //Beginn der Unterroutine Drucken.
PrintPage                                           //Aktuelle Seite Drucken.
CloseNoSave         ::::                            //Bildschirm löschen ohne zu speichern.
Return                                              //Ende der Unterroutine Drucken.
::::::

Label               (Öffnen)                        //Beginn des Labels  Öffnen.
FileOpen            ::::                            //Öffnen einer Datei.
Go                  (Anfang)                        //Verzweigung zu dem Label Anfang.

Label               (Speichern)                     //Beginn des Labels Speichern.
::::::
```

Das Beispiel zeigt Ausschnitte eines größeren Makros, die u. a. eine Seite drucken bzw. eine Datei öffnen (über manuelle Eingabe des Dateinamens). Die gezeigten Befehle werden wie folgt ausgeführt:

- Der *Call*-Befehl ruft die Unterroutine *Drucken* auf.
  1. Drucken der aktuellen Seite.
  2. Löschen des Bildschirmes ohne die Datei zu speichern.
- Durch *Return* erfolgt der Rücksprung zum aufrufenden Befehl.
- Danach wird der unmittelbar folgende Befehl (hier: der *Prompt*-Befehl) ausgeführt.
- Die Meldung »Text wurde gedruckt« wird für 20/10 Sekunden am Bildschirm angezeigt.
- Das Programm verzweigt zum Label *Öffnen*.
- Die manuelle Eingabe eines Dateinamens wird gefordert.
- Danach erfolgt die Verzweigung zu dem Label *Anfang*.

Weitere Informationen hierzu finden Sie in Kapitel »6 Makro-Programmierbefehle« bei den Befehlen *Call*, *Case*, *Case Call*, *Go*, *Label* und *Return*.

5.9 Dialogfelder

Eine Vielzahl von WordPerfect-Optionen werden über Dialogfelder ausgewählt bzw. Daten müssen manuell in Dialogfelder eingegeben werden. Bei der Makro-Aufzeichnung können Sie nun entscheiden, ob Sie die Dateneingabe oder die Optionsauswahl während der Makro-Aufzeichnung oder erst bei der Makro-Ausführung vornehmen wollen.

Dialogfelder während der Makro-*Aufzeichnung* bearbeiten

Bei der Aufzeichnung eines Makros können Sie wie üblich auch Dialogfelder bearbeiten. Nach der Anzeige eines Dialogfeldes besteht die Möglichkeit, in Textfeldern Daten einzugeben, Kontroll- oder Optionsfelder zu aktivieren bzw. zu deaktivieren usw. Nach dem Verlassen eines Dialogfeldes werden die vorgenommenen Änderungen (und nur diese!) in dem Makro gespeichert. Felder, die nicht geändert wurden, werden auch nicht in dem Makro gespeichert.

Bei der späteren Ausführung dieses Makros werden in dem gewählten Dialogfeld immer die während der Makro-Aufzeichnung vorgenommenen Änderungen in das betreffende Dialogfeld übernommen. Das Dialogfeld wird hierbei nicht angezeigt. Zusätzliche Änderungen können aber erst dann vorgenommen werden, wenn das Makro beendet ist. Aber dann kann es bereits zu spät sein. Wenn Sie z. B. [*Format, Ränder*] wählen, und für linken/rechten Rand die Werte 2c bzw. 2,5c eingeben, werden diese Werte auch in das Makro fest übernommen:

MarginLeft (2,0003c)

MarginRight(2,4998c)

Dialogfelder während der Makro-*Ausführung* bearbeiten

Dialogfelder, die erst während der Makro-Ausführung bearbeitet werden können (Eingabe aktueller Informationen), dürfen Sie nicht, wie zuvor beschrieben, während der Makro-Aufzeichnung bearbeiten. Bei der späteren Makro-Ausführung muß darum das betreffende Dialogfeld angezeigt werden, damit der Benutzer seine Daten eingeben kann. Das Makro wird in diesem Fall nach der Anzeige des Dialogfeldes unterbrochen, damit Sie Daten eingeben, verändern oder Options-/Kontrollfelder aktivieren oder deaktivieren können. Nach dem Klikken auf [*OK*] oder auf [*Abbrechen*] wird das Makro fortgesetzt. Um das Dialogfeld während der Makro-Ausführung anzuzeigen, müssen Sie folgende Schritte durchführen:

1. Rufen Sie bei der Makro-Aufzeichnung zu gegebener Zeit das Dialogfeld auf, das während der späteren Makro-Ausführung modifiziert werden soll. Wenn Sie z. B. [*Format, Ränder*] wählen, wird folgendes Dialogfeld angezeigt (Abbildung 5.1). :

2. Aktivieren Sie das Kontrollfeld im rechten Teil der Titelleiste. Nach dem Klicken mit der Maus erscheint in diesem Feld ein diagonales Kreuz. Änderungen, die jetzt Sie jetzt in dem Dialogfeld noch vornehmen, werden nicht in dem Makro gespeichert.

3. Klicken Sie zum Verlassen des Dialogfelds auf *[OK]*. Wurde das genannte Feld markiert, wird in diesem Fall nur der Befehl

   FormatMarginsDlg()

   in das Makro aufgenommen. Es erfolgt keine Datenübernahme! Diese Beschreibung gilt sinngemäß für alle verfügbaren Dialogfelder. Die Befehle zum Anzeigen von Dialogfeldern sind produktspezifische Befehle, die Sie bei Bedarf auch über den Makrobefehlsmanager einfügen können. Sie sind an der Endung *...Dlg()* zu erkennen.

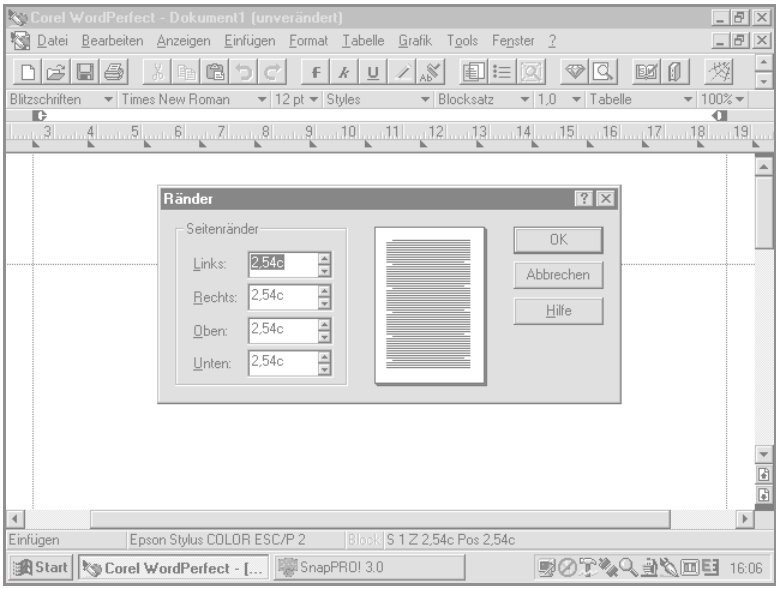


Bild 5.1: Format, Ränder

Wenn Sie zum Verlassen eines Dialogfeldes auf *[Abbrechen]* klicken, werden keine Informationen in das Makro übernommen.

4. Bei der Makro-Ausführung wird beim Antreffen eines *...Dlg()*-Befehls das Makro für die Dateneingabe unterbrochen.

Dialogfelder, die über andere Dialogfelder aufgerufen werden, können auf diese Art weder aufgezeichnet noch ausgeführt werden. Über die Makro-Programmierbefehle *Dialog...* oder den Dialogeditor können Sie eigene Dialogfelder erstellen und im Makro verwenden.

5.10 Mathematische Funktionen

Bei den Makro-Programmierbefehlen können Sie in Verbindung mit numerischen Ausdrükken (siehe Abschnitt »5.2 Makro-Programmierbefehle«) mathematische Funktionen verwenden, um z. B. Bedingungen zu prüfen und je nach Ergebnis in dem Makro entsprechend zu reagieren. In einem numerischen Ausdruck sind auch mehrere Funktionen möglich. Die Verwendung von Klammern ist erlaubt.

5.10.1 Mathematische Operatoren

Folgende Funktionen können Sie verwenden (*Z1* und *Z2* stehen für Zahl1 und Zahl2):

Wert	Operation

-Z1 Bewirkt die Umkehrung des Wertes einer Variablen.

Beispiel:

```
Z1=   111
Z1=   Z1
```

Wenn Variable *Z1* den Wert **111** enthält, wird nach Ausführung der Operation in *Z1* der Wert **-111** gespeichert.

Z1+Z2 Addiert *Z1* und *Z2*.

Beispiel:

```
Z1=   111
Z2=   50
Z1=   Z1+Z2
```

Nach der Ausführung der Operation wird das Ergebnis der Addition in *Z1* gespeichert. Ergebnis: **161**.

Hinweis: Mit dem Pluszeichen **+** können Sie auch zwei oder mehrere alphanumerische Variablen oder Konstanten miteinander verbinden:

Beispiel: **Text1=**"Mein Name ist "+Vorname+" "+ Nachname

Enthält die Variable *Vorname* = *Hans* und die Variable *Nachname* = *Meier* wird in *Text1* "Mein Name ist Hans Meier" gespeichert.

Z1-Z2 Subtrahiert *Z2* von *Z1*.

Beispiel:

```
Z1=   111
Z2=   50
Z1=   Z1-Z2
```

Nach der Ausführung der Operation wird das Ergebnis der Subtraktion in *Z1* gespeichert. Ergebnis: **61**.

Z1*Z2 Multiplikation von *Z1* mit *Z2*.

Beispiel:

```
Z1=   111
Z2=   10
Z1=   Z1*Z2
```

Nach der Ausführung der Operation wird das Ergebnis der Multiplikation in *Z1* gespeichert. Ergebnis: **1110**.

Z1/Z2 Division von *Z1* durch *Z2*.

Beispiel:

```
Z1=   111
Z2=   10
Z1=   Z1/Z2
```

Nach der Ausführung der Operation wird das Ergebnis der Division (Ganzzahl und Rest) in *Z1* gespeichert. Ergebnis: **11,1**.

Z1%Z2 Ermittlung des Restes der Division von *Z1* durch *Z2*.

Beispiel:

Z1=	111
Z2=	10
Z1=	Z1 % Z2

Nach der Ausführung der Operation wird der Rest der Division in *Z1* gespeichert. Ergebnis: 1.

Z1 DIV Z2 Ermittlung der Ganzzahl der Division von *Z1* durch *Z2* (Modulo).

Beispiel:

Z1=	111
Z2=	10
Z1=	Z1 DIV Z2

Nach der Ausführung der Operation wird die Ganzzahl der Division in *Z1* gespeichert. Ergebnis: **11**.

Z1 MOD Z2 Ermittlung des Restes der Division von *Z1* durch *Z2* (Modulo).

Beispiel:

Z1=	111
Z2=	10
Z1=	Z1 MOD Z2

Nach der Ausführung der Operation wird der Rest der Division in *Z1* gespeichert. Ergebnis: 1.

5.10.2 Vergleichs-Operatoren

Folgende Funktionen können Sie beim Vergleichen verwenden (*A1* und *A2* stehen für Ausdruck1 und Ausdruck2):

Wert	Operation
NOT A	Kehrt die Bedeutung des Bedingungs-Ausdrucks um. Aus *True* wird *False* bzw. aus *False* wird *True*.
A AND A2	Logische *UND*-Verknüpfung. Das Ergebnis ist wahr, wenn *A* und *A2* identisch sind.
A OR A2	Logische *ODER*-Verknüpfung. Das Ergebnis ist wahr, wenn *A* oder *A2* bzw. beide Bedingungen erfüllt sind.
A XOR A2	Exklusives *Oder*. Das Ergebnis ist wahr, wenn genau eine der beiden Bedingungen erfüllt ist.
A1 = A2	Die Bedingung trifft zu, wenn *A1* und *A2* gleich sind.
A1 <> A2	Die Bedingung trifft zu, wenn *A1* nicht gleich *A2* ist.
A1 > A2	Die Bedingung trifft zu, wenn *A1* größer als *A2* ist.
A1 < A2	Die Bedingung trifft zu, wenn *A1* kleiner als *A2* ist.

Wert	Operation
A1 <= A2	Die Bedingung trifft zu, wenn *A1* kleiner oder gleich *A2* ist.
A1 >= A2	Die Bedingung trifft zu, wenn *A1* größer oder gleich *A2* ist.

5.10.3 Rangfolge der Operatoren

Ausdrücke können mehrere Operatoren enthalten. Bei der Auswertung dieser Ausdrücke muß eine bestimmte Reihenfolge zur Ermittlung korrekter Ergebnisse eingehalten werden. Diese Reihenfolge berücksichtigt im wesentlichen die arithmetischen Regeln. Bei der Auswertung von Ausdrükken wird folgende Rangfolge der Operatoren eingehalten:

1. (), – (unäres Minus), + (unäres Plus), ~ (bitweises Nicht), NOT (Logisches Nicht)
2. * (Multiplikation), / (Division), % (MOD), DIV (Ganzzahl)
3. + (Addition), – (Subtraktion)
4. << / >> (Operandenfolge um eine Stelle nach links / rechts verschieben)
5. < (kleiner), <= (kleiner oder gleich), > (größer), >= (größer oder gleich), <> oder != (nicht gleich), = (gleich)
6. & (bitweises Und), | (bitweises Oder), ^ (bitweises XOR)
7. AND (logisches Und), XOR (logisches XOR)
8. OR (logisches Oder)
9. := oder = einer Variablen einen Wert zuweisen.

Die Rangfolge kann durch Setzen von Klammern geändert werden. Die zuerst auszuwertenden Elemente eines Ausdrucks müssen Sie darum in Klammern setzen. Diese Elemente werden dann vor den Elementen außerhalb der Klammer ausgewertet. Wurden mehrere Klammern geschachtelt, wird die innerste Klammer zuerst ausgewertet.

Beispiel:

```
8 * 54        = 40
8 * 5 + 4     = 44
8 * (5 + 4)   = 72
```

Bei Operatoren mit gleicher Rangfolge erfolgt die Abarbeitung von links nach rechts.

5.11 Reservierte Wörter

Hierbei handelt es sich um Wörter, die die Funktion eines Befehls genau festlegen (z. B. *Prompt* = Anzeigen einer Meldung). Das Falschschreiben oder Ersetzen eines dieser Wörter durch ein anderes Wort ist nicht erlaubt und führt immer zu Fehlern bei der Kompilierung. Diese Wörter haben eine bestimmte syntaktische Bedeutung und müssen daher genauso geschrieben und sinngemäß verwendet werden, wie es in Kapitel »6 Makro-Programmierbefehle« vorgegeben ist. Diese Wörter dürfen Sie auch nicht als Variable- oder Labelnamen verwenden. Darunter fallen:

- Alle produktspezifischen Befehle.
- Alle Programmier-Befehle.
- Alle WordPerfect-Systemvariable.
- Alle mathematischen und Vergleichs-Operatoren.

Bei den Befehlen sind Parameternamen und Ausdrücke eingeschlossen.

Kontrollieren Sie darum bei unklaren Fehlermeldungen, die im ersten Moment keinen Fehler erkennen lassen, ob nicht ein solches Wort falsch verwendet oder falsch geschrieben wurde.

Zusätzlich zu den reservierten Wörtern sind auch sog. Lesbarkeitswörter vorhanden. Sie werden meistens in Verbindung mit den produktspezifischen Befehlen verwendet und dienen lediglich der besseren Lesbarkeit eines Befehls. Beim Weglassen solcher Wörter läuft das Makro trotzdem fehlerfrei. Sofern sie verwendet werden, sind sie an richtiger Stelle und in der richtigen Schreibweise zu benutzen, ansonsten erfolgt beim Kompilieren eine Fehlermeldung.

Beispiel:
```
Type    (Text:"Das ist ein Test")
Display (State:On!)
```

Text: und **State:** sind in diesen Beispielen Lesbarkeitswörter, die auch weggelassen werden können:

Beispiel:
```
Type    ("Das ist ein Test")
Display (On!)
```

5.12 Makro-Design

Wer bereits Programmiererfahrung gesammelt hat, hat sich bestimmt schon eigene Regeln für die Programmierung auferlegt. Die folgenden Hinweise gelten darum in erster Linie für solche Anwender, die noch keine oder nur wenig Programmiererfahrung besitzen. Um Makros übersichtlich und transparent zu halten, kann man sich bei der Vergabe von Makronamen einer bestimmten Systematik bedienen. Wenn Sie mehrere unterschiedliche Projekte betreuen, sollten Sie die ersten drei Stellen des Makronamens mit einer Abkürzung dieses Projekts belegen, die übrig bleibenden Stellen (max. 5) können Sie mit einer weiteren »sprechenden« Abkürzung belegen. Dieser Abkürzung sollte man problemlos entnehmen können, welche Arbeiten dieses Makro ausführt, z. B.:

FINDRUCK.WCM Projekt »**Fin**anzen«. Drucken von Daten.

FINADRES.WCM Projekt »**Fin**anzen«. Adressenaufkleber drucken.

EINDRUCK.WCM Projekt »**Ein**kauf«. Drucken von Daten.

EINADRES.WCM Projekt »**Ein**kauf«. Adressenaufkleber drucken.

Aufgrund dieser vereinbarten Namenskonventionen ist sofort erkennbar, welchem Projekt die Makros zugeordnet sind, und was sie bewirken. Erlegen Sie sich bitte bestimmte Regeln für die Makro-Definition und die Befehlsanordnung auf, bzw. verwenden Sie in allen Makros denselben Formalismus. Das hat den Vorteil, daß Sie oder ein Kollege/eine Kollegin sich in

den Makros sofort zurechtfinden. Dies trifft insbesondere auf die Vergabe von Label-Namen zu. Bei größeren Makros sollten Sie z. B. diese mit einer fortlaufenden Nummer versehen, damit Sie nicht unnötige Zeit mit dem Suchen der Befehlsroutinen vergeuden müssen.

```
Label      (H01Anfang)          //Label "H01Anfang"
If         (Text="OK")
Call       (U100Drucken)        //Aufruf der Unterroutine
                                "U100Drucken"
Else
Go         (H02Speichern)       //Verzweigung zu "H02Speichern"
:::::
Go         (H03Laden)           //Verzweigung zu "H03Laden"

Label      (H02Speichern)       //Beginn des Labels
                                "H02Speichern"
:::::
Go         (H01Anfang)          //Verzweigung zu "H01Anfang"

Label      (H03Laden)           //Beginn des Labels "H03Laden"
:::::
Go         (H01Anfang)          //Verzweigung zu "H01Anfang"

Label      (U100Drucken)        //Beginn der Unterroutine
                                "U100Drucken"
:::::
Return                          //Ende der Unterroutine
                                "U100Drucken"
```

Verzweigungen innerhalb der Unterroutinen können Sie dann z. B. mit U105.., U110.., U115.. usw. (in diesem Fall bei Call-Aufrufen) benennen.

Beachten Sie bitte, daß Label immer mit einem Buchstaben beginnen müssen! Sonderzeichen und Leerstellen sind nicht erlaubt.

Vergeben Sie innerhalb der oben vorgeschlagenen Nummernkreise die Nummern der Label-Namen immer im Abstand von 5 oder 10, damit Sie beim späteren Einfügen von neuen Labels bei der Nummernvergabe noch »Luft« haben und keine Umnumerierungen vornehmen müssen. Zusätzlich könnten Sie auch für Hauptroutinen vor der Nummer des Label-Namens ein »H...« (= Hauptroutine) vergeben. Anstelle der Punkte werden fortlaufende Nummern eingefügt. Bei der Verwendung von Unterroutinen, könnte man »U...« (= Unterroutine) verwenden. Auch hier werden anstelle der Punkte fortlaufende Nummern vergeben.

Gliedern Sie Makros in einen Hauptteil (= Steuerteil) und in mehrere Unterroutinen.

```
Label      (H01OAnfang)          Hauptroutine
Call       (U10OEingabe)         (Makro-Steuerung)
Call       (U11ODrucken)
Call       (U12OBerechnen)
Call       (U13OAdressen)
Call       (U14OSpalten)
Go         (H02OÖffnen)

Label      (H02OÖffnen)          Ab hier Unterroutinen
:::::
Go         (H01OAnfang)

Label      U11ODrucken)
:::::
Return

Label      (U12OBerechnen)
:::::
Return

Label      (U13OAdressen)
MergeFileAssociate
:::::
```

In dem gezeigten Beispiel erfolgt die Makrosteuerung über die ersten fünf Call-Befehle. Das Makro wird hierdurch logisch und optisch untergliedert. Die Folge davon ist eine klare Strukturierung und ein übersichtliches Layout, wodurch das Makro wartungsfreundlich wird. Die Suche nach Fehlern kann dadurch eingegrenzt werden. Die Labelnamen der Unterroutinen sollten so gewählt werden, daß sie Aufschluß darüber geben, was in den Unterroutinen verarbeitet wird.

Versuchen Sie, Unterroutinen nach Möglichkeit mit *Call* oder *Case Call* aufzurufen, um die Verwendung von **Go** einzuschränken! Hierdurch werden Ihre Makros übersichtlicher. Wenn Sie nur mit *Go* arbeiten, führen Sie eine sog. »Spaghetti-Programmierung« durch, die das Makro durch ständiges Hin- und Herspringen unübersichtlich macht. Eine solche Programmierung ist nicht wartungsfreundlich. Beachten Sie hierzu bitte auch die Beschreibung der Befehle *Go*, *Case* und *Case Call*. Nutzen Sie je nach Anforderung die Befehle *For*, *ForNext*, *ForEach*, *Repeat* oder *While*, um die Verwendung von *Go* zumindest einzuschränken. Beachten Sie hierzu bitte die Beispiele bei der Beschreibung der einzelnen Befehle.

Über die Befehle *Function* und *Procedure* können ebenfalls Unterroutinen definiert werden. Auch diese sollten Sie in das angegebene Muster einordnen. Haupt- und Unterroutinen (siehe oben) müssen mit eindeutigen Namen definiert werden. Innerhalb der Routinen genügt es meistens, die erforderlichen Label in Zehnersprüngen zu numerieren, um bei späteren Einfügungen in der Numerierung noch Platz zur Verfügung zu haben:

```
Label      (U110Adressen)                          //Drucken von Adressenaufklebern.
Label      (U110010)
Menu       ....
Case Call  (Auswahl:{1;U110020;2;U110030;3;U110040}U110010)

Label      (U110020)                               //Drucken einbahniger Aufkleber.
::::::
Go         (U110Ende)

Label      (U110030)                               //Drucken zweibahniger Aufkleber.
::::::
Go         (U110Ende)

Label      (U110040)                               //Drucken dreibahniger Aufkleber.
::::::
Go         (U110Ende)

Label      (U110Ende)
Return
```

In dieser Unterroutine wird gefragt, ob ein-, zwei- oder dreibahnige Adressenaufkleber gedruckt werden sollen. Je nach Auswahl verzweigt das Makro durch den *Case*-Befehl zu dem betreffenden Label innerhalb der aufgerufenen Unterroutine. Wurden die dort definierten Befehle abgearbeitet, wird die Unterroutine über einen einheitlichen »Ausgang« (hier: U110Ende) wieder verlassen.

Trennen Sie die einzelnen Unterroutinen durch Leerzeilen und/oder durch Kommentare. Ich empfehle Ihnen, Leerzeilen in Verbindung mit Kommentaren einzufügen. Bei späteren Änderungen eines Makros erkennen Sie dann sofort, was in einer bestimmten Unterroutine abgearbeitet wird. Je gewissenhafter Sie die Dokumentation vornehmen, um so leichter finden Sie sich oder jemand anderes zu einem späteren Zeitpunkt in großen Makros wieder zurecht.

Eine weitere Möglichkeit, ein Makro übersichtlich zu gestalten, sind Einrückungen. Verwenden Sie diese insbesondere bei dem *If*-Befehl. Das folgende Beispiel zeigt Ihnen zwei Varianten, wobei Sie auf Anhieb die Unübersichtlichkeit des ersten Beispiels erkennen:

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

Bild 5.2: Unübersichtliches Layout

```
If          (Eingabe="z«)
            Assign    (Kennz;"Neuzugang«)                  Else
   If       (Eingabe="a«)
            Assign    (Kennz;"Löschung«)                   Else
      If    (Eingabe="v«)
            Assign    (Kennz;"Änderung«)
         Else
            Prompt    (»F e h l e r »;"Falsche Kennung.«;;;)
            Pause
            EndPrompt
            Go        (Anfang)
      EndIf
   EndIf
EndIf
```

Bild 5.3: Übersichtliches Layout

Die hier vorgenommenen Einrückungen (durch Drücken von ⇥) müssen nicht unbedingt vorgenommen werden. Das Makro läuft ohne diese Einrückungen genauso. Sie dienen lediglich der besseren Übersicht. Hier ist z. B. sofort zu erkennen, wo eine *If*-Bedingung beginnt bzw. endet. Gleiches gilt für die Anweisung *EndIf*. Definieren Sie nach Möglichkeit je Zeile nur einen Befehl. Auch dadurch wird Ihr Makro übersichtlicher und leichter zu lesen. Außerdem ist das Einfügen weiterer Befehle einfacher. Verwenden Sie vielleicht eine kleinere Schriftart, damit längere Befehle in eine Zeile passen, oder wählen Sie als Papiergröße A4-Quer. Bei größeren Makros sollten Sie mit Kopf- oder Fußtexten arbeiten, in denen Sie den Makronamen, das aktuelle Tagesdatum und eine Seitennumerierung einbinden können. Der Ausdruck eines größeren Makros (z. B. zu Dokumentationszwecken, zur Fehlersuche oder zum Bearbeiten des Makros) wirkt dann übersichtlicher.

5.13 Testen von Makros

Der Makro-Test ist die Arbeit, die nach der Erstellung des Makros und der Beseitigung möglicher Eingabefehler (Formalfehler) als nächstes durchzuführen ist. Unter einem Test versteht man die logische Überprüfung eines Makros, bei der untersucht wird, ob das Makro das tut, was es tun soll, und ob es auch nicht tut, was es nicht tun soll.

Achtung: Testen Sie ein Makro niemals in Verbindung mit Originaldateien. Erstellen Sie sich kleine übersichtliche Testdateien oder eine Kopie der Originaldatei, mit deren Hilfe Sie Ihre Tests durchführen können. Sie vermeiden hierdurch Datenverluste, wenn ein neu erstelltes oder geändertes Makro noch fehlerhaft arbeitet.

Erstellen Sie darum zum Testen Ihrer Makros kleine Testdateien, die den Makroanforderungen in allen Einzelheiten entsprechen. Nach Möglichkeit sollte die Testdatei so erstellt sein, daß mit deren Daten alle Routinen und Bedingungen des Makros mindestens einmal durchlaufen werden.

Dies könnte beispielsweise dadurch erreicht werden, daß am Ausgang aller Bedingungen Durchlaufzähler definiert werden (z. B. Zählern, nur für den Test), auf den immer dann eine 1 addiert wird, wenn der Makrozweig durchlaufen wird. Keiner dieser Durchlaufzähler darf nach Beendigung des Makros den Wert Null enthalten. Je größer ein Makro ist, und je mehr Bedingungen enthalten sind, um so umfangreicher und schwieriger ist der Test. Es sei schon an dieser Stelle gesagt, daß nicht jedes größere Makro so getestet werden kann, daß seine Fehlerfreiheit garantiert wird (gilt übrigens auch für alle anderen Programmiersprachen).

Stellen Sie sich beispielsweise eine *If*-Bedingung vor. Bei einer Abfrage gibt es immer nur zwei Möglichkeiten, entweder trifft die Bedingung zu, oder sie trifft nicht zu. Hierfür wären also zwei Testfälle zu erstellen, einen für den *JA*-Zweig und einen für den *NEIN*-Zweig. Bei der Verknüpfung mehrerer *If*-Bedingungen wird die Sache schon komplizierter, denn für jede weitere Bedingung erhöht sich die Anzahl der Möglichkeiten und somit auch zwangsläufig die der Testfälle. Aus diesem Grunde sollte man Makros modular aufbauen (*Run*- oder *Chain*-Befehl), wobei dann jedes Makro (wenn logisch möglich) für sich alleine getestet werden kann. Hierdurch bleibt außerdem das Makro übersichtlich.

Die Erfahrung hat gezeigt, daß der Programmierer nicht seine eigenen Makros testen sollte. Vielmehr sollte in einem Programmiererteam jeweils der eine Programmierer die Makros eines anderen Kollegen testen. Zum Erstellen von Testfällen sollten auch EDV-erfahrene Mitarbeiter von Fachabteilungen herangezogen werden. Testet der Programmierer nämlich seine eigenen Makros selbst, erstellt er in der Regel solche Testfälle, die den Makrodurchlauf problemlos bewältigen. Je weniger Fehler er beim Testen entdeckt, um so besser ist sein Makro erstellt (meint er!). Aber gerade diese Aussage ist falsch, denn ein Testfall ist erst dann ein erfolgreicher Testfall, wenn durch ihn ein Fehler entdeckt wird, was durch den Test ja bezweckt wird. Das Ergebnis eines jeden Testfalls muß daher gründlich überprüft werden. Lieber einen oder zwei Testfälle mehr erstellen, als beim produktiven Einsatz des Makros einen »Absturz« zu riskieren, weil irgend ein Makrozweig im Test nicht berücksichtigt wurde. Es ist daher nur von Vorteil für den Programmierer, wenn er die Fehler in seinem eigenen Programm durch umfangreiche Tests entweder selbst oder in Verbindung mit Kollegen ermittelt.

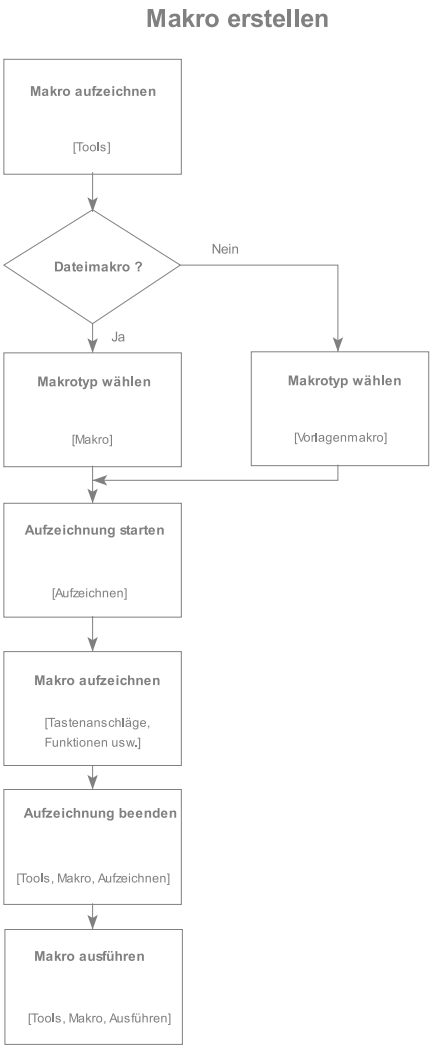
Beim Erstellen von Testfällen sollte also auf die Vollständigkeit (so weit möglich) geachtet werden. Wichtig ist auch die Beschreibung der erwarteten Werte oder der Ergebnisse. Die einmal erstellten Testfälle und die erwarteten Ergebnisse dürfen nach dem Abschluß eines Tests nicht in den Papierkorb wandern oder gelöscht werden. Vielmehr sollte man sie sorgfältig zusammen mit der Makrodokumentation für den Fall aufbewahren, daß sie zu einem späteren Zeitpunkt, sollte das Makro einmal geändert werden, wieder zur Verfügung stehen. Denn auch das systematische Erstellen von Testfällen nimmt eine Menge Arbeitszeit in Anspruch und darf nicht vernachlässigt werden!

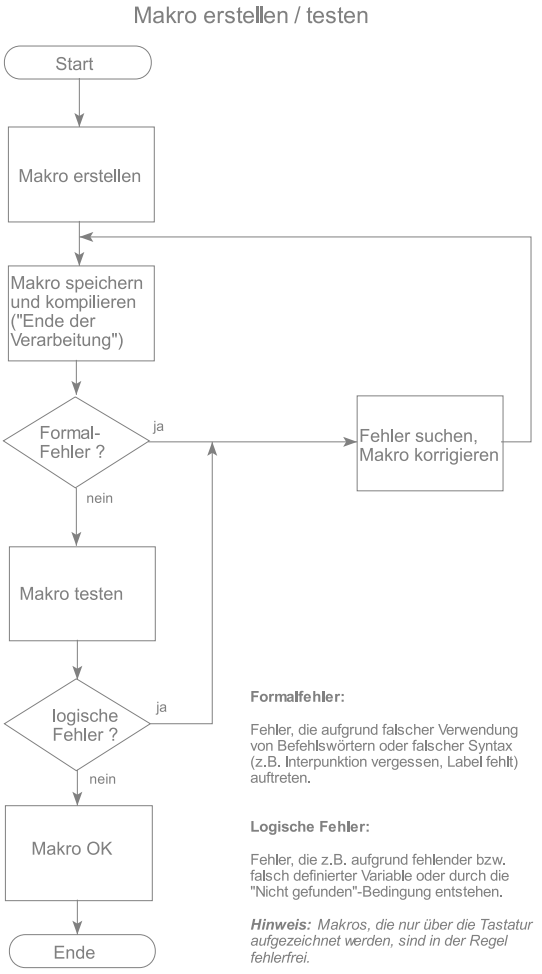
Nutzen Sie nach Möglichkeit beim Testen die extra hierfür vorhandenen Makrobefehle wie z. B. *Display*, *Prompt*, *MessageBox*, *Speed*, *Wait* usw. Größere Makros können Sie schneller testen, wenn Sie bereits getestete Makrozweige, die momentan nicht benötigt werden, entweder auf Kommentar setzen (nur wenn dadurch keine Kompilierfehler entstehen!) oder diese vielleicht mit *Go*-Befehlen überspringen. Kompilierfehler entstehen z.B. dann, wenn Sie einen Label-Befehl auf Kommentar setzen, zu dem von einer anderen Stelle im Makro verzweigt wird.

Dieses Überspringen ist nur dann sinnvoll, wenn die Befehle in diesen Routinen keine Auswirkung auf den restlichen Makroablauf haben. Werden in den auf Kommentar gesetzten

Routinen Variable neu definiert, Werte ermittelt oder Entscheidungen getroffen, die sich auf die Ausführung des Makros auswirken, dürfen Sie diese Routinen beim Testen nicht ausklammern!

Auch fehlerfrei kompilierte Makros können bei der Ausführung Fehler erzeugen, wenn z. B. eine Variable angesprochen wurde, die Sie nicht definiert haben (siehe oben) oder die zur Datenübergabe in Verbindung mit den Befehlen *Global*, *Persist* oder *PersistAll* in einem anderen Makro definiert wurden.





5.14 Fehlersuche

Treten bereits bei der Makro-Kompilierung Fehler auf, wird der fehlerhafte Befehl angezeigt und ein Pfeil zeigt auf die vermeintlich falsche Angabe. Die Anzeige ist manchmal etwas irreführend, weil auch auf einen Fehler hingewiesen wird, der nicht existiert. In den meisten Fällen ist das ein Folgefehler, der aufgrund einer falschen Angabe im vorausgegangenen oder im folgenden Befehl auftritt (z. B. Klammern falsch gesetzt, Semikolon vergessen, Parameter vertauscht usw.). Zur besseren Orientierung wird die Zeilennummer angegeben, bei der der Fehler aufgetreten ist.

Leider werden im Dokumentfenster keine Zeilennummern angezeigt. Bei kleinen Makros ist es kein Problem die betreffende Zeile zu finden. Bei größeren Makros, die mehrere Seiten lang sind, können dann schon Probleme mit dem Auffinden der fehlerhaften Zeile auftreten. Verwenden Sie in diesem Fall das von WordPerfect mitgelieferte Makro ZEILNUM.WCM, um die fehlerhafte Zeile zu lokalisieren. Beachten Sie bitte auch die Optionen des PerfectScript-Debuggers in Abschnitt »5.14.1 PerfectScript-Debugger«.

Diese Art der Fehlersuche ist nur dann erforderlich, wenn bei der Ausführung des Makros Fehler auftreten. Bei Kompilierfehlern wird der Cursor automatisch in der fehlerhaften Zeile positioniert, sofern die Makroleiste eingeblendet ist.

Beachten Sie hierbei, daß längere Befehle, die durch einen automatischen Zeilenumbruch in der Folgezeile fortgesetzt werden, als eine Zeile zählen. Um das zu vermeiden, sollten Sie mehrzeilige Befehle durch einen festen Zeilenumbruch trennen, sofern das die Syntax erlaubt.

Beispiel:

```
MergeFileAssociate("C:\WPWIN7\DATEN\ADRES-
SEN.FRM";C:\WPWIN7\DATEN\ADRESSEN.DAT")
MergeSelect(All!)
```

Dieser Befehl zählt als eine Zeile, weil er durch einen automatischen Zeilenumbruch getrennt wurde.

```
MergeFileAssociate
("C:\WPWIN7\DATEN\ADRESSEN.FRM";
C:\WPWIN7\DATEN\ADRESSEN.DAT")
MergeSelect(All!)
```

Hier wurde der Befehl in vier Zeilen untergliedert, die sowohl beim Kompilieren als auch beim Numerieren als vier Zeilen interpretiert werden. Außerdem wird die Befehlsfolge übersichtlicher. Diese Zeilen werden jeweils mit einem festen Zeilenumbruch beendet. Achten Sie bitte darauf, daß die Trennung immer bei dem Semikolon erfolgt, sonst erhalten Sie beim Kompilieren einen Fehler. Diese Trennung können Sie nicht bei jedem Befehl vornehmen (z. B. bei der Wertzuweisung zu einer Variablen), sonst erhalten Sie auch hier beim Kompilieren eine Fehlermeldung.

WordPerfect unterscheidet beim Kompilieren u.a. zwischen den beiden folgenden Fehlerarten:

Syntaxfehler Hierbei handelt es sich um schwerwiegende Fehler (Eingabefehler bei Schlüsselwörtern wie z. B. *Brompt* anstelle von *Prompt*), die in jedem Fall geändert werden müssen. Wenn Sie auf [*Kompilierung fortsetzen*] klikken, wird die Kompilierung zwar fortgesetzt und werden weitere Fehlermeldungen angezeigt, das Makro wird aber nach dem Ende der Kompilierung nicht ausgeführt.

Warnung Im Makro wurden Unregelmäßigkeiten festgestellt, wie z. B. die Definition eines Labels, der nicht aufgerufen oder zu dem nicht verzweigt wird. Nach dem Klicken auf [*Kompilierung fortsetzen*] wird die Kompilierung fortgesetzt. Wurden keine Syntaxfehler (siehe oben) angezeigt, wird das Makro trotzdem kompiliert und kann anschließend ausgeführt werden. Sicherheitshalber sollten Sie aber trotzdem die Fehler beseitigen, um ein »sauberes« Makro zu erhalten. Die Kompilierung des Makros wird dann nicht ständig durch Fehlermeldungen unterbrochen.

Ist die Kompilierung fehlerfrei durchgelaufen, heißt das noch lange nicht, daß das Makro auch fehlerfrei arbeitet. Jetzt können bei der Ausführung des Makros immer noch logische Fehler auftreten, die bei der Kompilierung nicht erkennbar sind. Die nachfolgenden Ursachen sind nur einige von vielen, die besonders bei Programmieranfängern häufig auftreten, die sich aber mit zunehmender Erfahrung ausmerzen lassen:

- Variable wurde falsch oder gar nicht initialisiert.
- Variable wurde vergessen zu definieren oder wurde falsch geschrieben.
- Schleifen und Endebedingungen zum Verlassen von Schleifen wurden falsch programmiert.
- *If*-Bedingungen sind fehlerhaft, weil z. B. *EndIf* und *Else* bei geschachtelten *If*-Bedingungen falsch eingesetzt werden.
- *Return*-Befehle wurden vergessen oder falsch gesetzt.

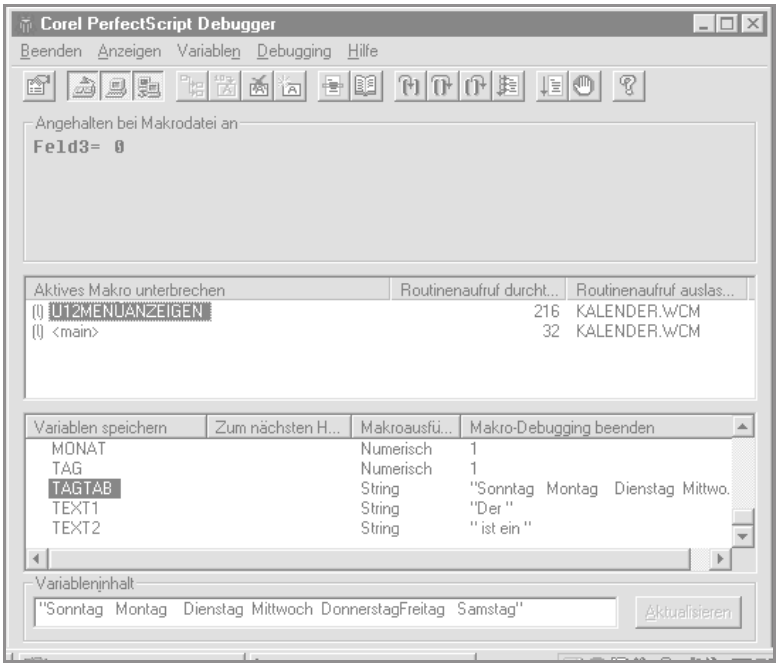
Sollte sich aus diesen Gründen das Makro einmal total »verlaufen« haben, und Sie nicht wissen, wo es abgebrochen wurde bzw. warum eine Endlosschleife entsteht, dann können Sie bei kleineren Makros *Display* oder *MessageBox* verwenden, damit Sie am Bildschirm den Makroablauf verfolgen können. Man kann auch an den »strategisch« wichtigen Punkten über den Befehle *MessageBox* (siehe Kapitel »6.78 MessageBox«) aktuelle Feldinhalte und Nachrichten anzeigen lassen. Zusätzlich können über den Befehl *Step* sog. *Break-Points* gesetzt werden, um an bestimmten Stellen den Makroablauf anzuhalten und bei Bedarf Variableinhalte anzuzeigen oder den Inhalt zu ändern (Makro-Online-Debugging).

5.14.1 PerfectScript-Debugger

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. Bei Bedarf können Sie Breakpoints setzen, um das Makro anzuhalten und um bestimmte Variable zu kontrollieren. Die Definition neuer (oder vergessener Variablen) ist möglich. Die Inhalte von Variablen können jederzeit verändert werden. Neu definierte Variable werden jedoch nicht automatisch in das Makro übernommen. Sie sind bei Bedarf nachträglich zu ergänzen.

Der 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 die Fehler vermutet werden. Das ist insbesondere bei umfangreichen Makros sinnvoll, um nicht jeden einzelnen Befehl bei der Makroausführung zu kontrollieren. Benutzen Sie den 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 des Befehls *Step(On!)* wird der *Corel PerfectScript-Debugger* gestartet und das gleichnamige Dialogfeld angezeigt.

Über dieses Dialogfeld kann nun der Makroablauf schrittweise gesteuert werden. Beim jedem Drücken von (F8) wird der nächste Makrobefehl ausgeführt. In der Spalte [Variablen speichern] werden die bis zu diesem Zeitpunkt verarbeiteten Variablen namentlich und in der Spalte [Makro-Debugging beenden] inhaltlich angezeigt (dieses Feld sollte eigentlich »Inhalt« heißen). Durch Klicken auf den Spaltenköpfen wird die jeweilige Spalte auf- oder absteigend sortiert. Der Ablauf kann jederzeit abgebrochen (= Makroende) werden.

Achtung: In der mir vorliegenden englischen Version sind die Tasten, Spalten und Hilfetexte anders benannt. Auch reagieren manche Tasten der Tastenleiste in der deutschen Beta-Version falsch. Wahrscheinlich handelt es sich hier um Programmfehler, der bei der endgültigen Version wahrscheinlich bereinigt ist.

Das Dialogfeld ist wie folgt aufgebaut (Kurzbeschreibung):

Menüleiste

Menüpunkte und Untermenüs zur Bedienung des Debuggers:

Beenden

Beenden des Debuggers. Gleichzeitig wird die Makroausführung abgebrochen. Es wird ein Hinweis eingeblendet, der den Befehl anzeigt, bei der der Abbruch erfolgte.

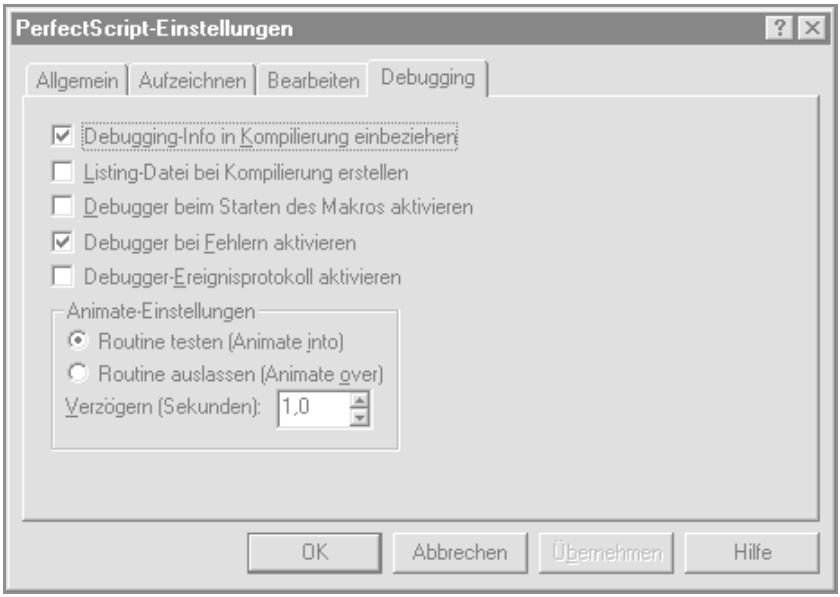
Anzeigen

Anzeigen verschiedener Dialogfeld-Elemente, Kommentare eingeben, Standardeinstellungen festlegen.

Tastenleiste	Ein- oder Ausblenden der Tastenleiste.
Lokal	Ein- oder Ausblenden von Variablen des Typs *Local*.
Global	Ein- oder Ausblenden von Variablen des Typs *Global*.
Persistent	Ein- oder Ausblenden von Variablen des Typs *Persist* bzw. *Persist All*.
Neuanzeige	Neuanzeige des Dialogfeldes.
Ereignisprotokoll	Festhalten von Bemerkungen, zur späteren Auswertung des Makroablaufs. Geben Sie unter [Anmerkung] Ihre Bemerkungen ein, und klicken Sie auf [*Hinzufügen*]. Der Kommentar wird in das Feld [Debugger-Ereignisprotokoll] übernommen. Über [*Löschen*] können alle Kommentare wieder gelöscht werden. Möchten Sie die Kommentare für eine spätere Auswertung speichern, klicken Sie auf [*Speichern*], und geben Sie einen Dateinamen ein. Die Kommentare werden als ASCII-Datei mit der Erweiterung .LOG gespeichert. Über [*Schließen*] können Sie dieses Dialogfeld wieder verlassen. Durch Aktivieren/Deaktivieren von [*Protokoll aktivieren*] können Sie diese Funktion ein- oder ausschalten.
Einstellungen	Standardeinstellungen für den Debugger. Die wichtigsten Einstellungen finden Sie in der Registerkarte *Debugging*: Debugging-Info... Während des Kompilierens werden bestimmte Informationen angezeigt. Nach Beendigung des Kompilierens erscheint eine Statistik. Aktivieren Sie diese Option auch dann, wenn Sie während des Debuggings die jeweilige Makrozeile im Debugger unter [*Angehalten bei Makrodatei an*] sehen möchten. Listing-Datei... Es wird bei der Kompilierung des Makros eine Datei erstellt, die die Makrobefehle und Zeilennummern enthält (hilfreich bei der Fehlersuche). Kann gedruckt werden, auch wenn *Step(On!)* nicht benutzt wird. Ereignisprotokoll... Beim Auftreten von Fehlern wird automatisch der Debugger gestartet.

Variablen

Handhabung von Variablen. Vor der Ausführung eines Menüpunkts ist die jeweilige Variable unter [*Variablen speichern*] zu markieren. Beim Löschen oder Erstellen neuer Variable werden diese Änderungen nicht in das Ursprungsmakro übernommen.



Erweitern	Expandieren bestimmter Variablentypen, wie z. B. mit *Declare* definierte Arrays. Dadurch werden alle Elemente eines definierten Arrays angezeigt. Das Expandieren kann auch durch einen Doppelklick auf die mit einem Pluszeichen gekennzeichnete Variable ausgeführt werden.
Komprimieren	Expandierte Variable wieder komprimieren, die Auflistung wird dadurch übersichtlicher. Das Komprimieren kann auch durch einen Doppelklick auf die mit einem Minuszeichen gekennzeichnete Variable ausgeführt werden.
Aktualisieren	Markierte Variable mit dem unter [Variableninhalt] eingegebenen Wert versehen.
Löschen	Die markierte Variable wird gelöscht. Sie können auch (Entf) drücken. Bedenken Sie bitte, daß das Makro eine Fehlermeldung anzeigt und vielleicht abbricht, wenn die gelöschte Variable zu einem späteren Zeitpunkt wieder benötigt wird.
Neu	Definieren einer neuen Variablen. Sie können auch (Einfg) drücken. Es wird ein Dialogfeld eingeblendet, in dem Sie eine neue Variable definieren können. Geben Sie den Variablennamen ein, wählen Sie den gewünschten Typ aus, und klicken Sie auf [*Erstellen*]. Die neudefinierte Variable wird in die Liste [Variablen speichern] übernommen. Über [Variableninhalt] können Sie danach dieser Variablen einen Wert zuweisen.

Debug

Steuerung des Debuggers.

Weiter — oder F5. Die schrittweise Durchführung des Makros wird beendet und ohne Anzeige des Debugger-Dialogfelds fortgeführt. Dasselbe geschieht, wenn im Makro der Befehl *Step(Off!)* ausgeführt wird

Beenden — oder ALT-F5. Beenden des Debuggers. Gleichzeitig wird die Makroausführung abgebrochen. Es wird ein Hinweis eingeblendet, der die Zeilennummer anzeigt, bei der der Abbruch erfolgte.

Routine testen — oder F8. Das Makro wird schrittweise ausgeführt. Nach jedem Klikken auf dieser Taste wird der nächste Befehl abgearbeitet, wobei die Zeilennummer und der Makroname (wichtig beim Aufruf von selbständigen Makros über den Makrobefehl *Run*) in dem Feld [Angehalten bei Makrodatei an] angezeigt werden.

Routine auslassen — oder F10. Die unter [Routineaufruf und Routineaufruf auslassen] angezeigte Call-Routine wird ausgeführt, aber nicht im Debugging-Mode angezeigt.

Routine verlassen — oder [Umschalten-F7]. Die aktuelle Routine (z. B. For-, While-, Repeat-Schleife usw.), wird zuende geführt, aber nicht im Debugging-Mode angezeigt. Der Debugger wird dabei verlassen. Mit [Alt-Tab] kommen Sie wieder in den Debugger zurück.

Animate — Das Makro wird automatisch schrittweise ausgeführt. Wählen Sie erneut [*Debugging, Animate*], um diese Ausführung zu stoppen.

Haltepunkte — Festlegung von Makrounterbrechungen (Breakpoints). Standardmäßig werden als Breakpoints *MacroStart!* und *MakroEnde!* vorgegeben. Weitere Breakpoints können Sie über [Anweisungsanfang] oder [Zeilennummer] auswählen. Wählen Sie danach unter [Typ] die gewünschte Unterbrechungsart aus. Achtung: Jeder Menüpunkt hat unterschiedliche Optionen, über die Sie festlegen können, was beim Eintreten dieses Haltepunkts geschehen soll. Klicken Sie danach auf [*Hinzufügen*], um den neu ausgewählten Haltepunkt in die Liste zu übernehmen. Über die Kontrollfelder der Spalte [Debugger Termin-Log] wird ein Haltepunkt aktiviert bzw. deaktiviert. Klikken Sie auf [*Entfernen*], um einen markierten Eintrag zu löschen, mit [*Alle löschen*] werden alle Einträge gelöscht. Verwenden Sie [*Aktualisieren*], wenn Sie bei bereits vorhanden Haltepunkten Änderungen vorgenommen haben. Drücken Sie zum Anzeigen von Hilfetexten [*Umschalten-F1*], und klicken Sie danach mit dem als Fragezeichen angezeigten Mauszeiger auf dem gewünschten Feld.

Hilfe

Online-Hilfe zu PerfectScript und dem PerfectScript-Debugger. Sie erhalten auch Hilfe, wenn Sie [*Umschalten-F1*] drücken, und danach auf dem gewünschten Feld bzw. Menüpunkt klikken.

Tastenleiste

In der Tastenleiste sind die wichtigsten Funktionen der in der Menüleiste vorhandenen Menüpunkte verfügbar, um effizienter arbeiten zu können. Die Tastenleiste kann nicht geändert oder ergänzt werden. Plazieren Sie den Mauszeiger auf einer Taste, um so deren Funktion zu ermitteln.

Angehalten bei Makrodatei an

In diesem Feld wird bei der schrittweisen Ausführung des Makros der jeweils nächste Befehl, der durch Drücken von (F8) oder der Auswahl von [*Debugging, Routine testen*] ausgeführt wird, angezeigt. Aktivieren Sie unter [*Anzeigen, Einstellungen, Debugging*] die Option [*Debugging-Info in Kompilierung einbeziehen*], wenn hier die Makrobefehle während des Debuggings angezeigt werden sollen.

Aktives Makro unterbrechen, Routineaufruf durchtesten/ auslassen

Namentliche Anzeige der Routinen, in denen sich das Makro gerade befindet. Zusätzlich wird die Zeilennummer des aktuell abgearbeiteten Befehls und des Makronamens (wichtig beim Aufruf von selbständigen Makros über den Makrobefehl *Run*) angezeigt.

Variablen speichern, zum nächsten Haltepunkt usw.

In diesem Bereich werden nähere Informationen zu den bis zu diesem Zeitpunkt definierten Variablen angezeigt:

Variablen speichern	Namen der Variablen. Die Anzeige erfolgt in der Reihenfolge, wie die Variablen im Makro während der Ausführung definiert werden. Durch Klicken auf den Spaltenkopf kann die Anzeige in auf- oder absteigender Sortierfolge vorgenommen werden. Variable mit einem Pluszeichen können über [*Variablen, Erweitern*] expandiert werden (z. B. Arrays). Danach werden Sie mit einem Minuszeichen versehen. Das Komprimieren erfolgt durch die Auswahl von [*Variablen, Komprimieren*]. Sie können in beiden Fällen auch auf der Variable doppelklicken.
Pool	Anzeige, in welcher Variablentabelle die Variable gespeichert ist: Lokal, Global oder Persist.
Type	Anzeige des Variablentyps: String, numerisch, Nicht definiert, Array usw.
Inhalt	Inhalt der Variablen. Bei der erstmaligen Definition erscheint die Variable mit Ihrem Initialisierungswert. Wird der Variableninhalt während der Makroausführung verändert, wird hier immer der neueste Wert angezeigt.

Variable Contents

Der Inhalt einer Variablen kann jederzeit geändert werden, um dadurch den Makroablauf zu beeinflussen. Markieren Sie unter [Variablen speichern] die zu ändernde Variable, geben Sie unter [Variableinhalt] einen neuen Wert ein, und klicken Sie auf [*Aktualisieren*]. Erst danach wird der neue Wert in die Variable übernommen. Achtung: Das kann beim Makroablauf zu Fehlern führen, wenn diese Variable nur bestimmte Werte enthalten darf!

Numerische Werte	Werden nur Ziffern ohne Leerstellen eingegeben, werden diese als numerische Werte gespeichert. Sollen sie als String gespeichert werden, sind sie in Hochkommata einzugeben.
String	Buchstaben und Sonderzeichen oder Ziffern in Verbindung mit anderen Zeichen werden generell als String definiert. Sie können durch Hochkommata eingeschlossen sein.
Hexadezimalzeichen	Werden nur Buchstaben von A–F (in Groß- oder Kleinbuchstaben) ohne Leerzeichen eingegeben, wird der Variableninhalt generell als 0 (= Null) angezeigt.

Aktualisieren

Die unter [Variablen speichern] markierte Variable wird mit dem unter [Variableninhalt] angegebenen Inhalt versehen.

5.14.2 Tips zur Fehlerlokalisierung

Die nachfolgenden Informationen sollen Sie in erster Linie bei der Fehlersuche unterstützen. Die Auflistung erfolgt nach Stichwörtern in alphabetischer Reihenfolge.

Anzeigen

Sollte in der Statuszeile die Meldung * *Bitte warten* * ohne ersichtlichen Grund angezeigt werden, überprüfen Sie, ob an der betreffenden Makrostelle der Befehl *Display(On!)* definiert wurde.

Ausrufezeichen

Verschiedene Ausdrücke von Parametern müssen jeweils mit einem Ausrufezeichen enden. Beim Vergessen des Ausrufezeichens erfolgt beim Kompilieren des Makros eine Fehlermeldung.

Befehlsform

Überprüfen Sie die Befehlsform (Syntax). Kontrollieren Sie, ob alle angegebenen Parameter, Variablen usw. richtig und in der geforderten Reihenfolge angegeben sind und ob nichts fehlt. Bei manchen Befehlen muß beim Auslassen eines Parameters als Ersatz ein

Semikolon gesetzt werden. Die Befehlswörter dürfen keine Leerzeichen enthalten. Zwischen Befehlen und Parametern sind Leerstellen, Tabstopps, Einrückungen (nicht immer!) und Zeilenschaltungen erlaubt.

Chain

Über *Chain* aufgerufene Makros müssen kompiliert sein. Geben Sie den Dateinamen des Makros einschließlich der Dateinamen-Erweiterung ein. Wird ein und dasselbe Makro in einem Makro mehrmals an unterschiedlichen Stellen aufgerufen, weisen Sie den Makronamen einer Variablen zu, und sprechen Sie unter *Chain* nur den Variablennamen an.

Dateinamen

Wenn Sie in einem Makro ein und denselben Dateinamen mehrmals verwenden, sollten Sie diesen Dateinamen am Beginn des Makros einer Variablen zuordnen. Innerhalb des Makros wird dann anstelle des Dateinamens nur diese Variable angesprochen. Das hat den Vorteil, daß Sie bei der Änderung des Dateinamens diesen nicht mehrmals, sondern nur einmal, nämlich bei der Variablendefinition, ändern müssen.

Doppelpunkt

Parameter-Namen bei produktspezifischen Befehlen müssen mit einem Doppelpunkt enden. Beim Vergessen des Doppelpunktes erfolgt beim Kompilieren des Makros eine Fehlermeldung. Der Doppelpunkt darf nicht mit dem Semikolon verwechselt werden.

Fehler

Makros werden abgebrochen, wenn während der Ausführung nicht korrigierbare Fehler auftreten. Drücken Sie (Esc) (evtl. mehrmals), wenn Sie eine Makroausführung abbrechen wollen. Verwenden Sie die Befehle *OnError*, *OnCancel* oder *OnNotFound*, um einen kontrollierten Abbruch zu erzeugen.

Go

Enden alle Label-Routinen, zu denen mit *Go* verzweigt wurde, wieder mit einem *Go*-Befehl oder ggf. mit *Quit*?

Hochkomma

Textkonstanten müssen Sie in Hochkommata angeben, ansonsten werden sie als Variable interpretiert. Werden innerhalb einer Textkonstanten weitere Hochkommata benötigt, müssen diese jeweils doppelt angegeben werden. Tabstopps, Einrückungen und Zeilenschaltungen sind innerhalb von Konstanten nicht erlaubt.

If

Bei geschachtelten **If**-Befehlen müssen so viele *EndIf* vorhanden sein, wie *If* definiert wurden. Ist dies nicht der Fall, erfolgt beim Kompilieren eine Fehlermeldung.

Klammern

Viele Fehler entstehen dadurch, daß die Klammern ( ) oder { } an den geforderten Stellen vergessen oder verwechselt wurden. Prüfen Sie die Befehlssyntax, und setzen Sie die Klammern entsprechend, oder löschen Sie zuviel angegebene Klammern.

Label

Stellen Sie sicher, daß die aufzurufenden *Label* oder die über *Call* oder *Run* auszuführenden Unterroutinen bzw. Makros auch wirklich vorhanden sind. Bedenken Sie, daß die Schreibweise beim aufrufenden Befehl und bei *Label* oder *Run* identisch sein muß (Groß-/Kleinschreibung wird hier nicht berücksichtigt). Stimmt die Schreibweise nicht überein, wird der Label bzw. das aufzurufende Makro nicht gefunden. Der Name muß durch Klammern eingeschlossen sein.

Beispiel: **Call** (Prüfen) → **Label** (Pruefen)

Die Schreibweise ist hier nicht identisch, es erfolgt beim Kompilieren eine Fehlermeldung. Jeder Label muß mit einem Buchstaben beginnen und kann mit einem @ enden (frühere WP-Versionen). Der Name darf nicht länger als 30 Zeichen sein.

Makro-Verkettung

Wenn Sie mehrere Makros verketten, sollten Sie sicherstellen, daß zu übergebende Variable nur einmal für ein und denselben Zweck vergeben werden. Damit ist gemeint, daß z. B. MAKRO1 eine Variable mit dem Inhalt »100« an MAKRO2 übergibt, um dort eine Berechnung durchzuführen. Wenn Sie jetzt vor der Durchführung der Berechnung in MAKRO2 dieselbe Variable für einen anderen Zweck verwenden (z. B. Speichern eines eingegebenen Textes in Verbindung mit *GetString*), ist das Ergebnis der Berechnung natürlich fehlerhaft.

Maus

Während der Makro-Aufzeichnung im Dokumentfenster können Sie innerhalb von Menüs oder den Dialogfeldern die Maus zur Cursorpositionierung bzw. zum Markieren von Feldern benutzen. Innerhalb des Textfensters kann der Mauszeiger nicht zur Cursorpositionierung verwendet werden. Er wird in diesem Bereich als Kreis mit einer diagonalen Linie angezeigt. Verwenden Sie zur Cursorpositionierung dann die Pfeil- oder die Bildtasten.

NumStr

Numerische Variable, die über *NumStr* in einen String konvertiert und zu einem späteren Zeitpunkt über *StrNum* wieder in einen numerischen Wert konvertiert wurden, können bei mathematischen Funktionen zu falschen Ergebnissen führen, wenn bei *NumStr* zu wenig oder gar keine Kommastellen berücksichtigt wurden.

Run

Die Makroausführung wird abgebrochen, wenn das Makro zu viele *Run*-Befehle enthält, wenn das aufgerufene Makro nicht kompiliert ist oder das Makro nicht gefunden wird (z. B. falscher Pfad/Dateiname). Geben Sie den Dateinamen des Makros einschließlich der Dateinamen-Erweiterung ein, sofern dafür nicht .WCM verwendet wurde. Wird ein und dasselbe Makro in einem Makro mehrmals an unterschiedlichen Stellen aufgerufen, weisen Sie den Makronamen einer Variablen zu, und sprechen Sie unter *Run* nur den Variablennamen an.

Return

Enden alle aufgerufenen Unterroutinen mit *Return*?

Schleifen

Achten Sie bei Schleifenverarbeitung auf eine ordentliche Definition von Schleifenanfang, Schleifenende und Schleifenausgang.

Drücken Sie bei einer vermuteten Endlosschleife (Esc), um das Makro abzubrechen. Sollte dies nicht möglich sein, drücken Sie [Strg-Untbr].

Besonders der Schleifenausgang ist wichtig, sonst läuft das Makro in eine Endlosschleife. Überprüfen Sie dann die gesetzten Bedingungen, insbesondere bei *If*, *For*, *ForEach*, *ForNext*, *Repeat* und *While*.

Bei der Verwendung von Befehlen, die mit einem *End*..-Befehl enden, müssen Sie auf die richtige Plazierung dieses Befehls achten. In keinem Fall dürfen Sie ihn vergessen.

Semikolon

Bei Befehlen mit mehreren Parametern werden die Parameter durch jeweils ein Semikolon getrennt. Beim Vergessen eines Semikolons erfolgt beim Kompilieren des Makros eine Fehlermeldung. Das Semikolon darf nicht mit dem Doppelpunkt verwechselt werden.

Standardvorgaben

In sehr vielen Makros werden auch die WordPerfect-Menüs benutzt, um z. B. bestimmte Steuerzeichen in den Text einzufügen. Bestätigen Sie vorgeschlagene Werte nicht durch Drücken von [↹], um zum nächsten Feld zu springen, da in diesem Fall der aktuelle Wert nicht in dem Makro gespeichert wird.

Beispiel: Sie möchten in Ihrem Text einen Zeilenabstand von zwei Zeilen. Da Sie gerade in diesem Text arbeiten, wird bei der Makrodefinition unter dem Zeilenabstand eine **2** angezeigt. Durch Drücken von [↹] wird keine 2 in das Makro übernommen, sondern lediglich der aktuelle Wert akzeptiert. Wenn bei künftigen Makroausführungen der Standardwert 1 als Zeilenabstand vorgeschlagen wird, wird dieser als Zeilenabstand übernommen. Dies wäre nicht passiert, wenn Sie eine 2 eingegeben hätten, anstatt [↹] zu drücken.

Regel: Die in den Menüs vorgeschlagenen Standardwerte nie durch Drücken von ↵ bestätigen, sondern immer über die Tastatur neu eingeben.

StrNum

Numerische Variable, die über *NumStr* in einen String konvertiert und zu einem späteren Zeitpunkt über *StrNum* wieder in einen numerischen Wert konvertiert wurden, können bei mathematischen Funktionen zu falschen Ergebnissen führen, wenn bei *NumStr* zu wenig oder gar keine Kommastellen berücksichtigt wurden.

Syntaxfehler

Wurden die Befehle bzw. die zu den Befehlen gehörenden Parameter oder Variablen in der richtigen Reihenfolge angegeben und vollständig benutzt? Syntaxfehler werden bereits bei der Makro-Kompilierung angezeigt.

Variable

Wurde die Variable definiert bzw. wurden die Routinen ausgeführt, in denen die Variable erstellt wird (z. B. *If-*, *For-*, *While*-Schleifen usw. oder mit *Call* oder *Case Call* aufgerufene Unterroutinen)?

Wurde auf die betreffende Variable richtig Bezug genommen?

Wurde eine Variable vor ihrer Ausführung mit bestimmten Werten initialisiert?

Wurde die Variable innerhalb eines Befehls oder einer Befehlsfolge an der richtigen Stelle angegeben?

Enthält die Variable einen String, obwohl numerische Daten erwartet werden oder umgekehrt?

Numerische Variable, die über *NumStr* in einen String konvertiert und zu einem späteren Zeitpunkt über *StrNum* wieder in einen numerischen Wert konvertiert wurden, können bei mathematischen Funktionen zu falschen Ergebnissen führen, wenn bei *NumStr* zu wenig oder gar keine Kommastellen berücksichtigt wurden.

VarErrChk

Makros, die von anderen Makros aufgerufen werden, übernehmen oft Daten aus dem aufrufenden Makro. Die hierfür erforderlichen Variablen dürfen in dem aufgerufenen Makro nicht definiert werden, da Sie in dem aufrufenden Makro bereits als globale oder als Persist-Variable definiert werden müssen. Beim Kompilieren eines aufgerufenen Makros können dadurch beim Kompilieren Fehler angezeigt werden, weil diese Variablen nicht definiert sind. Verwenden Sie in diesem Fall *VarErrChk(Off!)*, um die Überprüfung auszuschalten.

Zeilennummer (Anzeige bei der Kompilierung)

Längere Befehle, die durch einen automatischen Zeilenumbruch in der Folgezeile fortgesetzt werden, werden als eine Zeile gezählt. Die Zeilennumerierung stimmt dann ab diesem Befehl nicht mehr mit einer evtl. gewählten automatischen Zeilennumerierung auf dem Ausdruck eines Makros überein! Um das zu vermeiden, sollten Sie mehrzeilige Befehle durch einen festen Zeilenumbruch trennen, sofern das die Syntax erlaubt. Weitere Informationen hierzu entnehmen Sie bitte dem Anfang von Abschnitt »5.14 Fehlersuche«.

Verwenden Sie das Makro ZEILNUM.WCM, um schnell zu der angezeigten Zeilennummer zu gelangen. Nach dem Aufruf des Makros geben Sie die betr. Zeilennummer ein. Der Cursor wird danach in der angegebenen Zeile positioniert.

Auch über den Corel PerfectScript Debugger können Sie die Zeilennummer ermitteln, wenn Sie beim Kompilieren eine Makroliste erstellen lassen (siehe Abschnitt »5.14.1 PerfectScript Debugger«).

5.14.3 WordPerfect-Hilfe

Natürlich können Sie auch über die übliche WordPerfect-Hilfe Informationen zur Makro-Programmierung und den einzelnen Befehlen erhalten. Wählen Sie [*?, Hilfethemen, Inhalt, Makros*]. Zu den meisten Makro-Programmierbefehlen sind auch Beispiele vorhanden, die Sie zum Testen über die Zwischenablage kopieren und dann in ein WordPerfect-Dokumentfenster einfügen können. Auch das Drucken des Hilfetextes ist möglich. Verfahren Sie wie folgt:

- Wählen Sie [*?, Hilfethemen, Inhalt, Makros*].
- Klicken Sie auf [*Makroprogrammierung*].
- Klicken Sie auf [*Lists of Commands*].
- Wählen Sie aus, ob Sie Hilfe zu Systemvariablen, Produkt- oder Programmierbefehlen wünschen.
- Klicken Sie auf den Buchstaben, mit dem der gesuchte Begriff beginnt, z. B. G für *GetString*.
- Klicken Sie in der angezeigten Liste auf *GetString*.

Dem zugehörigen Hilfetext können Sie jetzt die benötigten Informationen entnehmen. Nach dem Klicken auf dem Beispielsymbol unterhalb des Befehls (sofern vorhanden) wird ein Beispielmakro angezeigt. Durch die Auswahl von [*Optionen, Drucken*] wird das Beispiel gedruckt. Möchten Sie das komplette Beispiel als Vorlage für eigene Makros oder zum Testen verwenden, wählen Sie [Optionen, Kopieren]. Wenn Sie nur einen bestimmten Bereich des Makros benötigen, markieren Sie diesen vor dem Kopieren als Block. Öffnen Sie ein leeres Dokumentfenster, und fügen Sie den Inhalt der Zwischenablage ein. Ersetzen Sie in der *Application*-Zeile die beiden Buchstaben *US* durch *DE*. Löschen Sie überflüssige Kommentarzeilen am Anfang und am Ende des Beispiels, die nicht mit // beginnen, um keine Kompilierfehler zu erhalten.

Sie können bei Bedarf auch ein komplettes Handbuch drucken (Systemvariable, Produktbefehle oder Programmierbefehle). Verfahren Sie wie folgt:

- Wählen Sie [*?*, *Hilfethemen*, *Inhalt*, *Makros*, *Makroprogrammierung*, *Lists of Commands*].
- Klicken Sie auf [*Print My Own Manual(xxx)*], wobei Sie anstelle von *xxx* den jeweiligen Befehlstyp wählen.

5.15 Plausibilität

Alle zu verarbeitenden Daten sind bei der Erfassung einer eingehenden Prüfung zu unterziehen, soweit das mit den Makrobefehlen möglich ist. Zum Teil kann diese Prüfung nur formal durchgeführt werden, d. h. man kann zwar überprüfen, ob z. B. in einer Variablen Daten eingegeben wurden. Es hat aber keinen Sinn, diese Daten zu prüfen, wenn es sich beispielsweise um Namen, Straßen-, Ortsbezeichnungen oder ähnliches handelt. Abhängigkeiten dagegen können ohne weiteres geprüft werden. Eine Adresse ist z. B. nur dann vollständig, wenn Name, Postleitzahl, Wohnort und Straße vollständig vorhanden sind. Wurden diese Werte bei der Eingabe jeweils einer Variablen zugeordnet, müssen die Variablen alle einen Wert enthalten. Fehlt eine dieser Angaben, weil z. B. die Variable *Postleitzahl* keine Daten enthält, muß zwangsläufig eine Fehlermeldung erfolgen (Adresse ohne Postleitzahl ist unvollständig).

Unterscheiden Sie hierbei, ob bestimmte Variable als MUSS- oder als KANN-Variable zu behandeln sind. MUSS-Variable sind solche Felder, die immer einen Wert enthalten müssen, KANN-Variable sind Felder, die die Angaben in den MUSS-Variablen vielleicht ergänzen, die aber nicht unbedingt vorhanden sein müssen. Durch Prüfen der einzelnen Variablen kann das Makro feststellen, ob in den MUSS-Variablen Daten vorhanden sind oder nicht.

5.16 Dokumentation

Jedes erstellte Makro müssen Sie dokumentieren, damit Sie bei späteren Änderungen in die Logik des Makros besser einsteigen können. Dies gilt in erster Linie für größere Makros. Dokumentieren Sie insbesondere geschachtelte **If**-Abfragen, sowie geschachtelte *For*-, *ForNext*, *Repeat*- und *While*-Schleifen. Haben Sie das Makro in Haupt- und Unterroutinen gegliedert, sollte vor jeder Routine eine Kurzbeschreibung zu deren Inhalt angegeben werden. Wenn Sie längere Zeit an einem Makro nicht mehr gearbeitet haben und dann Modifikationen vornehmen möchten, haben Sie mit einer guten Dokumentation keine Probleme, das Makro wieder zu verstehen und zu ändern. Bedenken Sie auch, daß Sie auch Makros erstellen, die nach Ihnen von einer anderen Person gepflegt werden müssen (Makro wird in einer Firma erstellt und an Tochterfirmen weitergegeben). Makros, die nicht ordentlich dokumentiert sind, können dann zu Problemfällen werden.

Erstellen Sie eine Dokumentation immer im Hinblick auf spätere Änderungen, d. h. arbeiten Sie gewissenhaft und gründlich. Jeder Programmierer ist sein »eigener Künstler«, der die Möglichkeiten der Programmierung und die Raffinesse verschiedener Befehle voll ausnutzt. Außer den in Abschnitt »5.12 Makro-Design« bereits gegebenen Empfehlungen sollten Sie in eine Dokumentation noch folgende Punkte einfließen lassen:

- Name des Makros
- Erstellt von
- Erstellt am
- Geändert von
- Geändert am
- Was wird mit diesem Makro bearbeitet?
- Welche Dateien werden verwendet?
- Welche Makros ruft es auf?
- Von welchen Makros wird es aufgerufen?
- Welche Prüfung von Eingabedaten wird vorgenommen?

Steht Ihnen genügend Platz zur Verfügung, sollten Sie Befehle, bei denen auf den ersten Blick nicht sofort zu erkennen ist, was sie ausführen, unmittelbar in derselben Zeile dokumentieren, ansonsten etwas eingerückt in der nächsten Zeile. Zu Beginn eines Makros sollten Sie eine kurze Einleitung schreiben, in der Sie erklären, welche Arbeiten dieses Makro durchführt. Werden Unterroutinen verwendet, bietet es sich an, am Beginn einer jeden Routine eine kleine Inhaltsangabe vorzunehmen. Werden Makros aufgerufen, sollten Sie angeben, welche Arbeiten die aufgerufenen Makros durchführen, welche Daten sie übergeben und welche Daten übernommen werden (siehe Beispiel).

```
// Makroname:                MWSTEUER.WCM
// Funktion:                 Ermitteln der Mehrwertsteuer eines eingegebenen Betrags
// Erstellt am:              1. Mai 1995 von:  H. Göckel
// Geändert am:                          von:
// Verwendete Dateien:       Keine
// Wird aufgerufen von:      Kein Aufruf
// Ruft folgende Makros auf: Kein Aufruf
//
// Beschreibung:             Das Makro dient zur Demonstration der Befehle Fraction und
//                           Integer. Über ein Dialog- feld ist der Betrag einzugeben, von
//                           dem die Mehrwertsteuer zu ermitteln ist. Über die Befehle
//                           Integer und Fraction wird die Mehrwertsteuer errechnet.
//                           Danach wird das ermittelte Ergebnis in einen String
//                           konvertiert, damit es über Prompt am Bildschirm angezeigt
//                           werden kann.

Application                  (A1;"WordPerfect";Default;"DE")
GetString                    (Betrag;"Betrag eingeben.";"MwSt-Berechnung")// Meldung
                                                                          // anzeigen
                                                                          // und Betrag
                                                                          // eingeben.
Betrag=                      StrNum(Betrag)       // Betrag von String in Zahl
konvertieren.
SteuerDM=                    Integer(Betrag*0.15) // Volle DM der Mehrwertsteuer ermitteln.
SteuerPf=                    Fraction(Betrag*0.15)// Pfennige der Mehrwertsteuer ermitteln.
SteuerDMPF=                  (SteuerDM+SteuerPF)  // Volle DM und Pfennige addieren.
SteuerDMPF=                  NumStr(SteuerDMPF)   // DM und Pfennige zum Anzeigen in einen
String konvertieren.
Prompt                       ("Ermittelte Mehrwertsteuer:";SteuerDMPF;;;)
                             //Anzeigen der Mehrwertsteuer.
Pause                        // Ergebnis so lange anzeigen, bis auf [OK] geklickt wird.

EndPrompt                    // Meldung ausblenden.

// Die Befehle Fraction und Integer dienen hier nur zu Demonstrationszwecken.
// Die Mehrwertsteuer läßt sich natürlich mit dem folgenden Befehl auch einfacher
// berechnen:  SteuerDMPF:=Betrag*0.15
```

Bild 5.4: Dokumentation eines Makros

5.17 Änderungen gegenüber WPWin 5.1/5.2

Diverse Befehle, die ein Ergebnis zurückliefern (= Return-Value), wurden gegenüber den früheren Versionen geändert. Bisher enthielten diese Befehle eine Variable, in der das Ergebnis gespeichert wurde. Ab dieser Version sind keine separaten Variablen mehr erforderlich. Vielmehr können diese Befehle in Verbindung mit anderen Befehlen direkt verwendet werden. Den nachfolgenden Beispielen können Sie die Änderungen entnehmen, und sinngemäß auf andere Befehle übertragen (beachten Sie bitte die Beschreibungen der Befehle in Kapitel 6).

Bei der Befehlsbeschreibung sind diese Befehle zusätzlich durch den Hinweis »WPWin 5.2« gekennzeichnet, um Leser, die Makros dieser Version umstellen oder bearbeiten müssen, auf die Änderung hinzuweisen.

Über den Befehl *SubStr* können aus einer Variablen (*Ausdruck*) ab einer bestimmten Position (*Startposition*) eine bestimmte Anzahl von Zeichen (*Länge*) extrahiert werden. Das Ergebnis wird in einer weiteren Variablen (*Extrakt*) gespeichert. Der Befehl wurde jetzt wie folgt geändert (bisheriger Befehl funktioniert auch noch):

Bisher: *SubStr*(Extrakt;Startposition;Länge;Ausdruck)

Neu: Extrakt:=*SubStr*(Ausdruck;Startposition;Länge)

Diese neue Variante hat auch den Vorteil, daß sie direkt in Verbindung mit anderen Befehlen zusammenarbeitet, ohne Zwischenschritte in Anspruch zu nehmen:

1. If (SubStr(Plz;1;1) = 6) (neu)

   Hier wird der Befehl direkt in Verbindung mit einer If-Bedingung verwendet, ein vorheriges Extrahieren in einem separaten Befehl ist darum nicht mehr erforderlich.

   SubStr (PlzEx;1;1;Plz) (wird eingespart)

   If (PlzEx = 6)

In der bisherigen Version mußte das Extrahieren separat vorgenommen werden. In dem If-Befehl wurde dann nur die Variable geprüft, die den extrahierten Wert enthielt.

2. Type (SubStr(Plz;1;3)) Schreiben eines Teilstrings.
3. Assign (PlzEx;SubStr(Plz;1;1)) Zuordnung von Daten.

   PlzEx:= SubStr(Plz;1;1) Zuordnung von Daten.

4. Meldung:= ("Kosten: "+NumStr(Ergebnis)) Zuordnung und gleichzeitige Konvertierung eines numerischen Feldes.
5. Prompt ("Kosten: »+NumStr(Ergebnis);

   "Gesamtsumme ermitteln";;;)

   Promptbefehl mit String und Datenfeld, ohne vorherige Konvertierung des numerischen Feldes.
6. If (StrLen(Eingabe) > 5) Prüfen der Länge eines Strings. Diese Änderungen beziehen sich auf die folgenden Befehle:
   CharLen, CharPos, Exists, Fraction, Integer, NumStr, StrLen, StrNum, StrPos, StrUnit, SubChar, SubStr, UnitStr.

Die Befehle NumStr, StrNum und SubStr müssen in älteren Makros, die in WordPerfect 7 übernommen wurden, zwingend geändert werden. Die Makros sind mit diesen Befehlen mit der zuvor beschriebenen Syntax zwar lauffähig, sie liefern aber falsche Ergebnisse.

5.18 Makro-Konventionen

Makros können produktspezifische Befehle, Makro-Programmierbefehle, Systemvariablen und Daten enthalten. Um diese Kategorien besser unterscheiden zu können, benutzt WordPerfect eine bestimmte Schreibweise:

Produktspezifische Befehle werden in Groß-/Kleinbuchstaben eingefügt:

Beispiel: CenterCurrentPage()

Makro-Programmierbefehle werden in Groß-/Kleinbuchstaben eingefügt:

Beispiel: GetString(Betrag;"Betrag eingeben.";"MWSt-Berechnung")

Systemvariablen werden in Groß-/Kleinbuchstaben mit vorangestelltem Fragezeichen eingefügt:

Beispiel: If(?TabelleInTabelle = 0)

Daten werden so eingefügt, wie Sie von Ihnen eingegeben wurden:

Beispiel: GetString(Betrag;"Betrag eingeben.";"MWSt-Berechnung")

In diesem Buch wird bei den Makro-Programmierbefehlen von dieser Regelung abgewichen, da manche Befehle in dieser Schreibweise schlecht zu lesen sind:

DDEEXECUTEEXT	wird als	DDEExecuteExt	dargestellt
DIALOGADDLISTBOX	wird als	DialogAddListBox	dargestellt

Diese langen Befehlswörter lassen sich in Groß-/Kleinbuchstaben besser lesen. Die Schreibweise hat keinerlei Auswirkung auf die Makro-Ausführung.