

4 Referenzteil

4.1 Anweisungen zur Daten - Definition

DIM	Feld definieren
LOCAL	Lokale Variable definieren
STATIC	statische Variable definieren
RECTYPE	Recordtyp definieren
RECVAR	Record-Element definieren
ENDREC	Recordtyp-Definition beenden
RECORD	Record definieren

4.2 Anweisungen zur Programmablaufsteuerung

IF	If-Abfrage
FOR	For-Schleife
WHILE	While-Schleife
REPEAT	Repeat-Schleife
BREAK	Schleife abbrechen
STOP	Programm beenden
GOTO	Label anspringen
SUBROUTINE	Unterprogramm definieren
DECLARE	Unterprogramm deklarieren
RETURN	Unterprogramm verlassen
GOSUB	Unterprogramm aufrufen

4.3 Befehle zur Datenein- und -ausgabe

OPEN	Datei öffnen
CLOSE	Datei schließen
INPUT	Daten lesen
LINE INPUT "	
PRINT	Daten ausgeben (Bildschirm, Datei)
LPRINT	Daten ausgeben (Drucker)
WRITE	Daten ausgeben
GET	Random-Datei lesen
PUT	Random-Datei schreiben
CHDIR	Verzeichnis wechseln
GETDIR	Verzeichnis feststellen
ERASE	Datei löschen
RENAME	Datei umbenennen
INKEY\$	Tastatur prüfen
MOUSE	Maus-Parameter lesen

4.4 Mathematische Funktionen

SIN	Sinus
COS	Cosinus
TAN	Tangens
ATN	Arcustangens
EXP	Exponentialfunktion
LOG	natürlicher Logarithmus
SQR	quadratische Wurzel

4.5 String-Statements und -Funktionen

SET\$	Zeichen setzen
DELETE\$	Zeichen löschen
INSERT\$	Zeichen einfügen
INSTR%	Teilstring suchen
LEFT\$	linken Teilstring bilden
RIGHT\$	rechten Teilstring bilden
MID\$	Teilstring bilden
TRIM\$	nachfolgende Leerstellen abschneiden
LTRIM\$	führende Leerstellen abschneiden
LCASE\$	Lowercase bilden
UCASE\$	Uppercase bilden
LEN%	Stringlänge feststellen
ASC%	ASCII-Wert bilden
CHR\$	Zeichen aus ASCII-Wert bilden
STR\$	String aus Zahl bilden
VAL	Real aus String bilden
VAL%	Integer aus String bilden
DOSTOWIN\$	Zeichenwandlung IBM nach ANSI
WINTODOS\$	Zeichenwandlung ANSI nach IBM

4.6 Grafik

CLS	Bildschirm löschen
COLOUR	Farben einstellen
CURSOR	Cursor-Darstellung
LOCATE	Cursor setzen in Zeichen-Einheiten
STYLE	Linien- und Schraffur-Stil
MAPMODE	Metrik setzen
MOVETO	Cursor setzen in Pixel-Einheiten
LINETO	Linie zeichnen
ARC	Bogen/Kreis
PIE	Kuchen
RECT	Rechteck
BITMAP	Bitmap zeichnen

FONT Zeichenfont einstellen

4.7 Verschiedenes

REM Kommentar

' Kommentar

ERROR% Runtime-Error feststellen

EXEC externes Programm starten

INTERRUPT Interrupt generieren

TIMEDATE\$ Zeit und Datum feststellen

TRACE Programm austesten

4.8 Windows

FILES Dateien zur Auswahl anbieten

DIALOG Dialogbox

DLGITEM Dialogitem lesen/setzen

FORM Datenerfassung für Record

MENU Menu erzeugen

MENUITEM Menuitem manipulieren

MESSAGEBOX Meldung erzeugen

SHOW Fenster-Darstellung ändern

SIZE Fenstergröße ändern

CLIP Clipboard Lesen/Schreiben

Anhang A Runtime-Errors

Erläuterungen:

Angaben in rechteckigen Klammern [] sind optional.

--> bedeutet Bildschirmausgabe durch das Programm

<-- bedeutet Eingabe durch den Benutzer

Mit Benutzer ist innerhalb des Referenzteils derjenige gemeint, welcher die mit WinBasic erstellten Programme benutzt, er ist prinzipiell zu unterscheiden von dem Programmierer, welcher diese Programme mit WinBasic erstellt.

4.1 Anweisungen zur Daten - Definition

Alle in diesem Kapitel genannten Programmanweisungen (Statements) sind nicht ausführbar und dienen der Definition von Daten und Datentypen.

DIM

Art: Statement

Form:

Dim Name(Länge[,Länge..) [,Name(..)..]

Name:Feldname

Länge: Integer-Konstante

Beschreibung:

'DIM' legt die Größe von Feldern fest. <Name> ist ein gültiger Variablenname, <Länge> die Länge des Felds. Bei nachfolgender Verwendung des Feldes kann auf die einzelnen Elemente des Feldes durch Angabe eines Index zwischen 1 und <Länge> zugegriffen werden.

Mehrdimensionale Felder sind möglich. Der Typ des Feldes wird bestimmt durch den Typ von <Name>. Zugelassen als Grundtyp sind alle einfachen Datentypen. Innerhalb einer 'DIM'-Anweisung können mehrere Felder definiert werden. Die einzelnen Definitionen werden durch Komma getrennt.

Beispiel:

```
Dim Int%(10), Str$(20)  
Str$(20) = "Zuweisung an Element 20"
```

LOCAL

Art: Statement

Form:

Local Name [,Name...]

Name: Variablenname

Beschreibung:

Die Variable <Name> wird als lokale Variable innerhalb einer Subroutine deklariert. Die Variable ist dann außerhalb der Subroutine nicht bekannt. Im Gegensatz dazu sind Variablen die im Verlauf einer Subroutine ohne vorhergehende 'LOCAL'-Deklaration eingeführt werden 'Global', d.h. auf diese Variable können auch das Hauptprogramm und andere Subroutines zugreifen. Als lokal deklarierte Variablen koexistieren mit globalen Variablen gleichen Namens, die im Hauptprogramm oder in anderen Subroutines erzeugt werden.

'LOCAL'-Anweisungen müssen am Anfang der Subroutine stehen. Sie dürfen nicht die Namen der Formalparameter haben. Mehrere Variablen können durch Kommas getrennt deklariert werden

'LOCAL'-Variablen sind flüchtig, d. h. zwischen zwei Aufrufen der Subroutine bleibt ihr Wert nicht erhalten. Beim Eintreten in eine Subroutine ist ihr Wert unbestimmt.

Beispiel:

```
Subroutine Sub1(x%)  
    Local r, a1%  
    Local s$  
    ...  
Endsub
```

Siehe auch:

STATIC

STATIC

Art: Statement

Form:

Static Name [,Name...]

Name: Variablenname

Beschreibung:

'STATIC' deklariert wie 'LOCAL' lokale Variablen. Im Gegensatz zu 'LOCAL'-Variablen sind 'STATIC'-Variablen nichtflüchtig, d.h. ihr Wert bleibt zwischen zwei Subroutine-Aufrufen erhalten. Bei Programmstart ist ihr Wert 0 bei numerischen Variablen bzw. "" bei Stringvariablen, genau wie bei globalen Variablen

Beispiel:

```
Static a22%, w#
```

Siehe auch:

LOCAL

RECTYPE

Art: Statement

Form:

Rectype Name

Name: Variablen-Name (Real-Konvention)

Beschreibung:

'RECTYPE' definiert einen zusammengesetzten Datentyp unter dem Typnamen <Name>. Record-Definitionen durch den Befehl 'RECORD' beziehen sich auf den hier definierten Namen <Name>. Die Zusammensetzung des Records aus einfachen Datentypen wird durch nachfolgende 'RECVAR'-Anweisungen definiert. Die Rectype-Definition wird durch 'ENDREC' abgeschlossen.

<Name> ist ein Variablen-Name nach der Namens-Konvention für Real-Zahlen, d. h. ohne Erweiterungszeichen %, &, #, \$.

Beispiel:

```
RECTYPE Artikeltype  
  RECVAR....  
ENDREC
```

Siehe auch:

RECVAR, ENDREC, RECORD

RECVAR

Art: Statement

Form:

Recvar Name Länge[:Nkomma] [,Name..]

Name: Variablenname (s. u.)

Länge: Integer-Konstante

Nkomma: Integer-Konstante

Beschreibung:

'RECVAR'-Anweisungen legen die Zusammensetzung eines Record-Typs aus einfachen Datentypen fest. Diese bilden die Elemente des Records bzw des Record-Typs. Die einzelnen Elemente eines Recordtyps können einzeln zeilenweise oder mehrere pro Zeile durch Kommas getrennt definiert werden. Den Abschluß der Definitionssequenz bildet eine 'ENDREC'-Anweisung.

<Name> ist ein Variablenname des Typs Real, Integer oder String. Long und Double sind nicht erlaubt.

Reals und Integer werden innerhalb von Records in ASCII-Darstellung erzeugt. Die Länge der Zahlen ist variabel, deshalb muß sie bei der Definition mit <Länge> festgelegt werden. Bei Reals muß zusätzlich durch einen Doppelpunkt getrennt die Anzahl der Nachkommastellen mit <Nkomma> festgelegt werden. Die Zahl der Vorkommastellen ist dann <Länge>-<Nkomma>-1.

Auch Strings haben innerhalb von Records eine feste Länge, deshalb muß auch hier mit <Länge> eine maximale Stringlänge festgelegt werden. Wird einem Recordelement vom Typ String ein gewöhnlicher String zugewiesen wird dieser auf die festgelegte Länge gebracht, entweder durch Abschneiden überschüssiger Zeichen oder durch Auffüllen mit Leerzeichen. Beim Zuweisen von String-Recordelementen an gewöhnliche Strings ist die Existenz von nachfolgenden Blanks zu beachten, diese können gegebenenfalls mit 'TRIM\$' entfernt werden.

Beispiel

```
Rectype Atype
...Recvar a% 6
...Recvar summe 8:2
...Recvar str$ 12
Endrec
Record Atype Arec, Arec1
```

```
Arec.str$ = "abc"
a$ = Arec.str$
Print "a$=";a$;"!"
-->a$=abc      !
                (9 Leerzeichen in a$)
```

Siehe auch:

RECORD

ENDREC

Art: Statement

Form:

Endrec

Beschreibung:

Die Sequenz der 'RECVAR'-Anweisungen und damit die Definition eines Record-Typs durch 'RECTYPE' wird beendet.

Siehe auch:

RECTYPE, RECVAR

RECORD

Art: Statement

Form:

Record Rtype Name [,Name...]

Rtype: Name eines Recordtyps

Name: Name des definierten Records

Beschreibung:

Durch 'RECORD' wird ein Record mit Namen <Name> definiert der die Struktur des Record-Typs <Rtype> hat. Auf die Elemente des Records <Name> wird zugegriffen durch Angabe von 'Name.Element', wobei Element der Name eines Record-Elements laut 'RECVAR'-Anweisungen bei der Definition des Record-Typs <Rtype> ist.

<Name> ist ein Variablen-Name nach der Namens-Konvention für Fließkommazahlen, d. h. ohne Erweiterungszeichen %, &, #, \$.

Mit einer 'RECORD' Anweisung können mehrere Records definiert werden, ihre Namen werden durch Komma getrennt.

Beispiel:

siehe RECTYPE.

4.2 Anweisungen zur Programmablaufsteuerung

IF

Art: Statement

Form:

1. If Ausdruck Then Statement
2. If Ausdruck Then
 ...
 [Elseif Ausdruck Then
 ...]
 [Else
 ...]
 Endif
 Ausdruck: logischer Ausdruck
 Statement: einfache Programmanweisung
 eine oder mehrere Anweisungen
 ...

Beschreibung:

Die 'IF'-Anweisung ist in zwei Formen möglich. In der ersten Form besteht die ganze Anweisung aus einer Zeile. Falls der logische Ausdruck <Ausdruck> WAHR ist wird die Programm-Anweisung <Statement> ausgeführt, bei FALSCH wird keine Anweisung ausgeführt. <Statement> ist eine einfache Anweisung, d.h. es dürfen nicht mehrere Anweisungen durch ':' getrennt nach 'THEN' folgen.

In der zweiten Form steht hinter 'THEN' keine Anweisung. Es folgen eine oder mehrere Zeilen mit Programmanweisungen, bis die 'IF'-Anweisung mit 'ENDIF' abgeschlossen wird.

Wahlweise können innerhalb der 'IF'...'ENDIF'-Sequenz eine oder mehrere 'ELSEIF' und eine 'ELSE'-Anweisung stehen. Nach einer 'ELSE'-Anweisung können keine 'ELSEIF'-Anweisungen mehr folgen. Die Anweisungen zwischen 'ELSE' und 'ENDIF' werden ausgeführt wenn sowohl <Ausdruck> aus der 'IF'-Anweisung als auch <Ausdruck> aus allen vorhergehenden 'ELSEIF'-Anweisungen FALSCH sind
Anweisungen die auf eine 'ELSEIF'-Anweisung folgen werden ausgeführt, wenn <Ausdruck> der 'ELSEIF'-Anweisung WAHR ist und <Ausdruck> der 'IF'-Anweisung und <Ausdruck> aller eventuell vorhergehenden 'ELSEIF'-Anweisungen FALSCH sind. Weiter folgende 'ELSEIF'-Anweisungen (wenn vorhanden) werden dann nicht mehr geprüft.

Beispiel

```
Input "a%=",1,a%
If a% = 1 Then
    Gosub Sub1()
Elseif a% = 2 Then
    Gosub Sub2()
Elseif a% = 3 Then
    Gosub Sub3()
Else
    Print "Fehlanzeige"
Endif
```

FOR ... ENDFOR

Art: Statement

Form:

1. For Zähl = Anf To Ende [Step Schritt]

...
Endfor

2. For Zähl = Anf Downto Ende [Step Schritt]

...
Endfor

Zähl: Integer, Real, Long, Double - Variable

Anf, Ende, Schritt:

Integer, Real, Long, Double

Beschreibung:

Die FOR-Anweisung stellt eine Programm-Schleife dar. Die Programmanweisungen zwischen 'FOR' und 'ENDFOR' werden unter Kontrolle der Variablen <Zähl> wiederholt durchlaufen. <Zähl> wird zunächst auf den Wert <Anf> gesetzt. Bei jedem Durchlauf wird ihr Wert um <Schritt> erhöht. Ist der Anweisungsteil 'STEP Schritt' nicht vorhanden, wird 1 als Schrittweite angenommen. Wenn <Zähl> = <Ende> ist, wird die Schleife nach dem Durchlauf bei 'ENDFOR' verlassen.

In der 1. Form wird hochgezählt, d.h. <Schritt> wird zu <Anf> addiert. In der 2. Form wird heruntergezählt, <Schritt> wird von <Zähl> subtrahiert.

<Anf>, <Ende>, <Schritt> können Konstanten oder Variablen sein. Sie sollten alle vom gleichen Typ sein, um Wandlungsoperationen zwischen den einzelnen Typen zu vermeiden.

Als <Zähl> und damit entsprechend letztem Abschnitt auch <Anf>, <Ende>, <Schritt> sollten nur Integer- oder Long-Variablen eingesetzt werden, da Real- und Double-Variablen nur eine beschränkte Genauigkeit haben. Damit ist der Vergleich ob <Zähl> = <Ende> ist im allgemeinen nicht korrekt durchzuführen! Die Schleife wird eventuell zu spät beendet.

Da der Zahlenbereich von Integers im allgemeinen ausreichend ist, werden sie bevorzugt als <Zähl> in 'FOR'-Schleifen eingesetzt.

'FOR'-Schleifen können vorzeitig durch die 'BREAK'-Anweisung verlassen

werden. Wird eine 'BREAK'-Anweisung beim Durchlaufen einer 'FOR'-Schleife erkannt, wird sofort zur nächsten Anweisung nach 'ENDFOR' verzweigt.

Beispiel

```
For i% = 1 To 20
  s% = s% + i%
  If s% > 100 Then Break
    'Vorzeitiger Abbruch
  Print "i%=";i%*10
Endfor
```

Siehe auch:

BREAK

WHILE ... WEND

Art: Statement

Form:

```
While Ausdruck
...
Wend
```

Ausdruck: Logischer Ausdruck

Beschreibung:

'WHILE' leitet eine Programm-Schleife ein, die solange durchlaufen wird wie der logische Ausdruck <Ausdruck> WAHR ist. Ist <Ausdruck> bei Eintritt in die Schleife FALSCH, wird diese nicht durchlaufen, sondern direkt hinter das Ende der Programmschleife verzweigt. Das Ende der Programmschleife wird durch 'WEND' markiert.

Unabhängig vom Ergebnis von <Ausdruck> kann die Schleife mit 'BREAK' verlassen werden. Wird eine 'BREAK'-Anweisung beim Durchlaufen einer 'WHILE'-Schleife erkannt, wird sofort zur nächsten Anweisung nach 'WEND' verzweigt.

Beispiel

```
i% = 0
While i% <= 20
  s% = s% + i%
  If s% > 100 Then Break
    'Vorzeitiger Abbruch
  Print "i%=";i%*10
Wend
```

Siehe auch:

BREAK

REPEAT ... UNTIL

Art: Statement

Form:

Repeat

...

Until Ausdruck

Ausdruck: Logischer Ausdruck

Beschreibung:

'REPEAT' leitet eine Programmschleife ein, die durch 'UNTIL' abgeschlossen wird. Ist <Ausdruck> bei Überprüfung durch 'UNTIL' FALSCH wird die Schleife wiederholt, ist sie WAHR wird die Schleife verlassen und zur nächsten Anweisung fortgeschritten.

Im Gegensatz zur 'WHILE'-Schleife wird die 'REPEAT'-Schleife immer mindestens einmal durchlaufen, da über die Wiederholung der Schleife erst am Schleifenende durch 'UNTIL' entschieden wird.

Wird eine 'BREAK'-Anweisung beim Durchlaufen einer 'REPEAT'-Schleife erkannt, wird sofort zur nächsten Anweisung nach 'UNTIL' verzweigt.

Beispiel:

```
i% = 0
Repeat
  s% = s% + i%
  If s% > 100 Then Break
    'Vorzeitiger Abbruch
  Print "i%=";i%*10
Until i% > 20
```

Siehe auch:

BREAK

BREAK

Art: Statement

Form:
Break

Beschreibung:

Durch 'BREAK' wird die momentan abgearbeitete Schleife bedingungslos verlassen. Die Programmausführung wird mit der nächsten Anweisung nach dem Schleifenende fortgesetzt.

Ausserhalb von Schleifen ist die Verwendung von 'BREAK' nicht zugelassen.

Siehe auch:

FOR, WHILE, REPEAT

STOP

Art: Statement

Form:

Stop [Prog\$]

Prog\$:String

Beschreibung:

Mit 'STOP' beendet ein Programm den Lauf des Programms <Prog\$> oder, wenn <Prog\$> fehlt, den eigenen Lauf.

GOTO

Art: Statement

Form:

Goto Label

Beschreibung:

'GOTO' bewirkt eine unbedingte Verzweigung des Programmablaufs zu der Anweisung, welche auf die Sprungmarke <Label> folgt. <Label> ist ein beliebiger alphanumerischer Name mit einem angehängtem ':'. Auf der gleichen Zeile darf keine Programmanweisung stehen.

Beispiel:

```
    If a% = 0 Then Goto Ende
Ende:
    Stop
```

SUBROUTINE

Art: Statement

Form:

```
Subroutine Name([(VAR]Par[,...])]
```

```
...
```

```
Endsub
```

Name: Variablenname

VAR: reserviertes Wort

Par: Variablenname

Beschreibung:

'SUBROUTINE' leitet ein Unterprogramm ein. Alle nachfolgenden Programmanweisungen sind Bestandteil des Unterprogramms, bis dieses durch die Anweisung 'ENDSUB' abgeschlossen wird.

<Name> ist der Name des Unterprogramms, auf den sich Aufrufe des Unterprogramms beziehen. Ein Unterprogramm kann sowohl als Prozedur durch die Anweisung 'GOSUB' als auch als Funktion in einem arithmetischen Ausdruck durch Nennung von <Name> aufgerufen werden. Dem Unterprogramm-Namen <Name> muß ein Paar runder Klammern folgen, auch wenn durch 'SUBROUTINE' ein Unterprogramm ohne Parameter definiert wird.

Der Name des Unterprogramms legt auch seinen Datentyp fest, analog zu den Namenskonventionen für den Datentyp von Variablen. Dies ist wichtig für Unterprogramme, die als Funktion in arithmetischen Ausdrücken aufgerufen werden sollen und mit 'RETURN' einen Wert zurückgeben.

Zwischen den runden Klammern werden die Formal-Parameter <Par> des Unterprogramms definiert. Dieses sind beliebige Variablen (einfache Datentypen, Felder, Records) die als Platzhalter für die Aktual-Parameter stehen, die beim Unterprogrammaufruf übergeben werden.

Beim Aufruf eines Unterprogramms müssen Formal- und Aktual-Parameter nach Zahl und Art übereinstimmen. Bei Feldern muss in Klammern die Anzahl der Dimension des Feldes angedeutet werden. Soll zum Beispiel ein zweidimensionales Feld beim Unterprogrammaufruf übergeben werden, muß der Formalparameter z. B. 'Feld(,)' heißen.

Bei Records muß das reservierte Word 'RECORD', dann der Record-Type, dann der Formalparametername angegeben werden (siehe Beispiel).

Hat das Unterprogramm als Formalparameter einen einfachen Datentyp, kann dieser beim Aufruf auch durch ein Feld-Element eines Feldes vom gleichen Typ als Aktualparameter abgedeckt werden.

Formal-Parameter werden vom Übersetzer wie lokale Variablen behandelt (siehe 'LOCAL') und können wie diese innerhalb des Unterprogramms verarbeitet werden. Einfache Datentypen werden 'by value' übergeben, Felder und Records 'by reference'. 'by value' bedeutet, daß bei Unterprogrammaufruf eine lokale Kopie des übergebenen Aktualparameters erzeugt und der Wert hierin abgelegt wird. Bei Änderung der Variablen wird die Kopie verändert, das Original im rufenden Programm bleibt erhalten. 'by reference' bedeutet, daß die Adresse des Aktualparameter übergebenen wird und Änderungen das Original ändern.

Felder und Records lassen sich wegen ihrer Größe nur 'by reference' übergeben. Bei einfachen Datentypen läßt sich eine Übergabe 'by reference' anstelle der standardmäßigen Übergabe 'by value' durch Angabe des reservierten Wortes 'VAR' vor dem Namen des Formal-Parameter by der 'SUBROUTINE'-Deklaration erzwingen. Diesen Formal-Parametern dürfen dann beim Unterprogramm-Aufruf keine Konstanten übergeben werden!

Ein Unterprogramm muß definiert sein, bevor sein Name im Programmtext als Aufruf erscheint. Unterprogramme können von anderen Unterprogrammen gerufen werden.

Beispiel:

```
Dim areal(10,10)
```

```
Subroutine Sub1%(a$, a(,), b%, Record Rtype R)
```

```
Local i%, d%
```

```
d% = 0
```

```
For i% = 1 To b%
```

```
    d% = d% + a(i%,i%)
```

```
Endfor
```

```
Print a$
```

```
Return d%
```

```
Endsub
```

```
Gosub Sub1%(meldung$(23), areal, 5, R1)
```

```
Dgnl% = Sub1%("Funktion", areal, 3, R1)
```

Dieses Beispiel zeigt ein Unterprogramm von Typ Integer. Es zeigt sowohl Eigenschaft einer Funktion, indem es einen Wert zurückgibt, der in arithmetischen Ausdrücken verarbeitet wird, als auch Eigenschaften einer Prozedur, indem es

eine Meldung ausgibt, ohne das das rufende Programm dieses unmittelbar gewahr wird.

Siehe auch:

GOSUB, RETURN

DECLARE

Art: Statement

Form:

```
Declare Name[[[VAR]Par[,...]]]
```

Name: Variablenname

VAR: reserviertes Wort

Par: Variablenname

Beschreibung:

In WinBasic müssen Unterprogramme definiert sein, bevor sie aufgerufen werden koennen. Mit DECLARE kann man ein Unterprogramm deklarieren, d.h. Name und Formalparameter erklären, ohne daß eine vollständige Definition an Ort und Stelle zu geben ist. Die Definition durch SUBROUTINE kann dann später oder in einem anderen Modul erfolgen.

Die Syntax von DECLARE ist identisch mit der von SUBROUTINE.

Unterprogrammname und Parameternamen müssen mit der in der zugehörigen SUBROUTINE-Anweisung genau übereinstimmen! Abweichungen werden von WinBasic nicht geprüft und führen zu Programmfehlern.

Beispiel:

```
Declare Info(s$)
...
...
Gosub Info("Text")
...
...
Subroutine Info(s$)
  REM Hier Anweisungen der Subroutine
Endsub
```

Siehe auch:

SUBROUTINE

RETURN

Art: Statement

Form:

Return Wert

Wert: Variable, Konstante

Beschreibung:

'RETURN' beendet ein Unterprogramm und gibt dem aufrufenden Programmteil den Wert <Wert> zurück. 'RETURN' kann am Ende eines Unterprogramms stehen, aber auch an beliebiger Stelle für einen vorzeitigen Abbruch. Ein Unterprogramm braucht keine 'RETURN'-Anweisung zu enthalten, es wird dann an der 'ENDSUB'-Anweisung beendet.

Der Datentyp von <Wert> muß mit dem Datentyp des Unterprogramms übereinstimmen.

Außerhalb von Unterprogrammen ist 'RETURN' nicht zugelassen.

Beispiel:

```
Subroutine Sub2%()  
  ...  
  If ... Then Return 0.0  
  ...  
  Return a  
Endsub
```

Siehe auch:

SUBROUTINE

GOSUB

Art: Statement

Form:

Gosub Name[[Par][,...]]

Beschreibung:

'GOSUB' ruft das Unterprogramm <Name> auf. Die Programmabarbeitung verzweigt zum ersten ausführbaren Befehl des Unterprogramms. Nach Beenden des Unterprogramms durch 'ENDSUB' oder 'RETURN' wird die Programmausführung mit der nächsten auf 'GOSUB' folgenden Programmanweisung fortgeführt.

Hat das gerufene Unterprogramm <Name> keine Parameter, so kann an den Namen ein Paar leerer runder Klammern angehängt werden, diese können entfallen. Hat das Unterprogramm Parameter, dann werden die als Aktualparameter zu übergebenden Variablen zwischen runden Klammern durch Komma getrennt aufgezählt. Ihre Anzahl und die Datentypen müssen mit der Anzahl und den Datentypen der Formalparameter des Unterprogramms übereinstimmen.

Beispiel:

siehe 'SUBROUTINE'

Siehe auch:

SUBROUTINE, RETURN

4.3 Befehle zur Dateibearbeitung

inklusive Dateneingabe über Tastatur und Datenausgabe auf Bildschirm und Drucker.

OPEN

Art: Statement

Form:

Open IO, Kennung, Name\$[, Rec]

IO:	String
Kennung:	s. u.
Name\$:	String
Rec:	Record-Type

Beschreibung:

Die Datei <Name\$> wird zur Bearbeitung geöffnet. <Name\$> kann ein vollqualifizierter Dateiname sein mit Angabe von Laufwerk und Verzeichnis. <IO> ist ein String aus einem einzelnen Zeichen, das die gewünschte Dateibearbeitung festlegt:

- "I" INPUT Datei wird gelesen
- "O" OUTPUT Datei wird geschrieben
- "U" UPDATE Datei wird gelesen und geschrieben
- "A" APPEND Datei wird geschrieben, neue Sätze
werden am Dateiende angehängt.
- "R" RANDOM Random-Datei

Kennung ist ein '#' gefolgt von einer Ganzzahl zwischen 1 und 250. Nachfolgende Lese- oder Schreibzugriffe erfolgen nicht über den Dateinamen <Name\$>, sondern über diese Kennung. Solange die Datei geöffnet ist darf mit der gleichen Kennung keine weitere Datei geöffnet werden.

Für INPUT geöffnete Dateien werden mit den Befehlen 'INPUT' und 'LINE INPUT' gelesen, für OUTPUT oder APPEND geöffnete Dateien werden mit 'PRINT' oder 'WRITE' geschrieben. Die Anzahl der mit einem Zugriff gelesenen bzw. geschriebenen Zeichen ist beliebig und kann schwanken. Für UPDATE geöffnete Dateien können gelesen und geschrieben werden. Die Lese- und Schreib-Aktionen findet immer an der Stelle der Datei statt, die bei der vorhergehenden Aktion erreicht wurde.

Wird das Ende einer für INPUT oder UPDATE geöffneten Datei erreicht, werden keine Zeichen gelesen und ein Runtime-Error generiert, der mit der Funktion 'ERROR%' festgestellt werden kann.

Eine für INPUT oder UPDATE zu öffnende Datei muß vorhanden sein, ansonsten wird ein Runtime-Error generiert. Eine für OUTPUT zu öffnende Datei wird neu kreiert, eine eventuell vorhandene wird dabei inhaltlich gelöscht. Eine für APPEND geöffnete Datei wird kreiert, wenn sie noch nicht existiert.

Die Angabe von <Rec> erfolgt nur für RANDOM-Dateien. Aus dem Record-Type übernimmt der Übersetzer die Satzlänge. RANDOM-Dateien können nach dem Öffnen sowohl gelesen ('GET') als auch geschrieben ('PUT') werden, jedoch nur mit Sätzen fester Länge.

Runtime-Errors: 1, 4

Beispiel:

```
Open "I", #1, "c:\Mverz\Fakt.txt"
Rectype Arttype
...
Endrec
Open "R", #9, "artikel.dat", Arttype
```

Siehe auch:

CLOSE, INPUT, LINE INPUT, PRINT, WRITE, GET, PUT

CLOSE

Art: Statement

Form:

Close Kennung

Kennung: siehe 'OPEN'

Beschreibung:

Die mit 'OPEN' unter <Kennung> geöffnete Datei wird geschlossen. Es können keine weiteren Lese- oder Schreib-Operationen stattfinden. <Kennung> darf wieder zur Öffnung anderer Dateien verwendet werden.

Beispiel:

Close #3

Siehe auch:

OPEN

INPUT

Art: Statement

Form:

1. Input Prompt\$, len%, Var
2. Input Kennung, Var [,Var...]

Prompt\$: String
Var: einfache Variable jeden Typs
Kennung: siehe 'OPEN'

Beschreibung:

'INPUT' liest Daten über eine Dialogbox (1. Form) oder aus Dateien ein (2. Form).

In der 1. Form ist Input eine Alternative zu einer Dialogbox, deren Programmierung einen höheren Aufwand erfordert. In dem Erfassungsdialog wird <Prompt\$> als Eingabeaufforderung ausgegeben, anschließend folgt ein Editierfeld, in dem der laufende Wert der Variablen dargestellt wird und durch den Programm-Benutzer geändert werden kann. <len%> ist die Länge des Editierfelds.

In der 2. Form können mit einer 'INPUT'-Anweisung mehrere Variablen gelesen werden, ihre Namen werden durch Kommas getrennt.

Es werden Daten im ASCII-Format erwartet. Als Trennzeichen zwischen den Werten werden ',' und 'CR' (Zeilenende) erkannt.

Man kann die Werte auch durch ein Paar Anführungszeichen "..." einrahmen. Bei String-Daten können die Strings dann auch Kommas enthalten, die ansonsten als Trennzeichen behandelt würden.

Runtime-Errors: 2

Beispiel:

```
Input "Bitte Breite eingeben:",6,breite%
-->Bitte Breite eingeben:
<--1
Print a%
1
```

Input #1, a%, b%, c%

Siehe auch:

OPEN, LINE INPUT

LINE INPUT

Art: Statement

Form:

Line Input Kennung, Var [,Var...]

Kennung: siehe 'OPEN'

Var: Variable

Beschreibung:

'LINE INPUT' arbeitet wie 'INPUT' in der 2. Form, mit dem Unterschied, das als Trennzeichen nur 'CR' (Zeilenende) gewertet wird.

Runtime-Errors: 2

Siehe auch:

OPEN, INPUT

PRINT

Art: Statement

Form:

1. Print Data [,Data...]
2. Print Kennung, Data [,Data...]

Data: Variable, Konstante

Kennung: siehe OPEN.

Beschreibung:

'PRINT' gibt Daten auf den Bildschirm (1. Form) oder in eine Datei aus (2. Form). Die Daten werden in ASCII-Form ausgegeben. Werden die einzelnen Ausgabedaten <Data> durch Kommas getrennt, werden die Daten auf Spalten positioniert mit einer Spaltenbreite von 12 Zeichen. Werden statt Kommas Semikolons gesetzt, werden die Daten ohne Zwischenraum hintereinandergesetzt.

Folgt hinter dem letzten <Data> ein Komma oder Semikolon, wird beim nächsten 'PRINT' an der aktuellen Stelle weiter ausgegeben, ansonsten wird ein 'CR' (Zeilenende) erzeugt. Der nächste 'PRINT' beginnt dann auf einer neuen Zeile. Die Ausgabedaten <Data> können durch die Klausel 'USING' gefolgt von einer Ausgabemaske vor der Ausgabe aufbereitet werden. Die Ausgabemaske enthält u. a. die Sonderzeichen '#' und '.'. '.' markiert den Dezimalpunkt, '#' ist ein Platzhalter für beliebige Zeichen aus dem auszugebenden <Data> und wird durch diese überschrieben. Alle anderen Zeichen der Ausgabemaske werden unverändert ausgegeben. Die Ausgabemaske gilt solange für jedes <Data> bis eine neue definiert wird oder mit 'USING ""' die Ausgabemaske intern gelöscht wird. Sind die ersten beiden Zeichen der Ausgabemaske zwei Kommas, so wird das Komma statt des Punkts als Dezimalzeichen gewertet; die beiden führenden Kommas werden ansonsten ignoriert.

Runtime-Errors: 2

Beispiel:

```
Print "123";"456","789"
-->123456    789
A = 1000.0
Print A,Using"##,###.##";A,Using",,##.###,##";A
-->1000.000000 1,000.00  1.000,00
```

Siehe auch:

OPEN, WRITE, LPRINT

LPRINT

Art: Statement

Form:

Lprint Data [,Data...]

Data: Variable, Konstante

Beschreibung:

'LPRINT' arbeitet wie 'PRINT' in der 1. Form, gibt jedoch die Daten nicht auf dem Bildschirm sondern auf dem Drucker aus. Die auszugebenden Daten werden zunächst im Hintergrund gesammelt und erst auf das Zeichen 'FORMFEED' hin wird eine Seite ausgegeben. Die Ausgabe von 'FORMFEED' (ASC-Zeichen Nr. 12) erreicht man durch LPrint Char\$(12).

Während der Ausgabe erscheint eine Messagebox, die das Abbrechen des Druckvorgangs erlaubt. Die Ausgabe kann sehr lange dauern, insbesondere bei Arbeiten mit Grafik oder Fonts auf Druckern hoher Auflösung. Zum eigentlichen Druck erscheint der Windows-Spooler (falls Windows mit Spooling installiert wurde) als Symbol und druckt im Hintergrund aus).

WRITE

Art: Statement

Form:

Write Kennung, Data [,Data...]

Beschreibung:

Wie 'PRINT' in der zweiten Form gibt 'WRITE' Daten in Dateien aus. Kommas zwischen den einzelnen <Data> bewirken jedoch keine Spalteneinteilung, sondern eine Ausgabe von Kommas in die Datei. Beim Wiedereinlesen der Datei mit Input können diese dann als Trennzeichen interpretiert werden.

Semikolons werden wie Kommas behandelt. Steht am Zeilenende kein Komma (Semikolon), wird eine neue Ausgabezeile erzeugt (CR), sonst ein Komma ausgegeben und bei der nächsten Ausgabe-Anweisung an dieser Stelle fortgesetzt.

Strings werden mit einem Paar Anführungszeichen "\"" ausgegeben, um eventuell im String enthaltene Kommas beim Wiedereinlesen nicht als Trennzeichen wirken zu lassen (siehe 'INPUT').

Runtime-Errors: 2

Beispiel:

```
A = 100.0 : A$ = "ABC,DEF"  
Write #2,A,A$,12  
-->100.0,"ABC,DEF",12    (in Datei)
```

Siehe auch:

INPUT, PRINT

GET

Art: Statement

Form:

Get Kennung, Satznr%, Satz

Kennung: siehe 'INPUT'

Satznr%: Integer

Satz: Record

Beschreibung:

'GET' liest Daten aus einer Random-Datei. Die Datei muß mit 'OPEN' als Random-Datei geöffnet werden. Random-Dateien bestehen aus Sätzen fester Länge. Die Sätze werden als Ganzes gelesen und geschrieben. Sätze werden durch eine Satznummer identifiziert. Der mit 'GET' aus der Datei gelesene Satz mit Satznummer <Satznr%> wird im Record <Satz> abgelegt.

Runtime-Errors: 2, 3

Beispiel:

Get #3,101,Asatz

Siehe auch:

PUT, OPEN

PUT

Art: Statement

Form:

Put Kennung, Satznr%, Satz

Kennung: siehe 'INPUT'

Satznr%: Integer

Satz: Record

Beschreibung:

'PUT' schreibt Sätze fester Länge in eine Random-Datei. Die Datei muß als Random-Datei geöffnet werden (siehe OPEN). Die Daten aus dem Record <Satz> werden unter der Satznummer <Satznr%> gespeichert.

Runtime-Errors: 2, 3

Beispiel:

Put #2,999,Asatz

Siehe auch:

INPUT, GET

CHDIR

Art: Statement

Form:

Chdir Name\$

Name\$: String

Beschreibung:

Das aktuelle Verzeichnis wird auf <Name\$> gesetzt.

Beispiel:

Chdir "\windows\system"

Siehe auch:

GETDIR

GETDIR

Art: Statement

Form:

Getdir Name\$

Name\$: String-Variable

Beschreibung:

Das aktuelle Verzeichnis wird ermittelt und in <Name\$> abgelegt.

Beispiel:

```
GetDir Dir$  
Print Dir$  
-->"c:\windows\system"
```

Siehe auch:

CHDIR

ERASE

Art: Statement

Form:

Erase Name\$

Name\$: String

Beschreibung:

'ERASE' löscht von der Platte die Datei <Name\$>. <Name\$> kann ein vollqualifizierter Pfadname mit Angabe von Laufwerk und Verzeichnis sein.

Beispiel:

Erase "c:\mverz\abc.txt"

RENAME

Art: Statement

Form:

Rename Alt\$, Neu\$

Alt\$: String

Neu\$: String

'RENAME' benennt eine Datei um. <Alt\$> ist der bestehende Name einer Datei. Dieser Name kann Laufwerk und Verzeichnis enthalten. <Neu\$> ist der neue Dateiname, er darf weder Laufwerk noch Verzeichnis enthalten, da diese Angaben von <Alt\$> übernommen werden.

Beispiel:

Rename "c:\mverz\abc", "123"

INKEY\$

Art: Funktion

Form:

a\$ = Inkey\$()

Beschreibung:

'INKEY\$' stellt fest, ob seit der letzten 'INKEY\$'-Anweisung ein Zeichen über Tastatur eingegeben wurde. Wenn ja, wird dieses Zeichen in <a\$> zurückgegeben, wenn nein wird eine leere Zeichenkette "" zurückgegeben. 'INKEY\$' wartet also nicht auf eine Eingabe wie 'INPUT', sondern kehrt sofort zurück.

a\$ enthält nur druckbare Zeichen oder die Funktionstasten F1 bis F9 bzw. SHIFT-F1 bis SHIFT-F9, andere Tastendrücke werden nicht übermittelt. Die einfachen Funktionstasten werden durch die Codes 187 - 195 dargestellt, die Shift-Funktionstasten durch die Codes 212 - 220.

Beispiel:

```
a$ = Inkey$()  
If Asc%(a$) = 195 Then Break    'F9 = Ende
```

MOUSE

Art: Statement

Form

Mouse Knopf%, x%, y%

Knopf%, x%, y%: Integer-Variablen

Beschreibung:

Die Anweisung 'MOUSE' sollte nur nach dem Windows-Label `_MOUSE` folgen, das beim Drücken eines Maus-Knopfes durch den Benutzer angesprungen wird. Die Parameter `<Knopf%>`, `<x%>` und `<y%>` geben dann an, welcher Knopf gedrückt wurde, und wie die aktuellen Maus-koordinaten sind. (Diese dürfen nicht mit den aktuellen Koordinaten für Text- und Grafik-Ausgabe verwechselt werden). `<Knopf%>` enthält folgende Werte:

- 1 linke Maustaste gedrückt
- 2 rechte Maustaste gedrückt
- 3 mittlere Maustaste gedrückt

Beispiel:

`_MOUSE`

`Mouse mousebutton%, xmouse%, ymouse%`

`If mousebutton% = 1 Then MoveTo xmouse%, ymouse%`

Die aktuelle Koordinaten für Text- und Grafik-Ausgaben wird auf den vom Programm Benutzer ausgewählten Punkt gelegt.

4.4 Mathematische Funktionen

Alle mathematische Funktionen nehmen eine numerische Konstante oder Variable (vorzugsweise vom Datentyp Real) als Argument und liefern ein Funktionsergebnis vom Datentyp Real. Wird ein anderer Datentyp als Real als Argument gegeben, wird intern eine Typwandlung durchgeführt unter Verbrauch von Speicherplatz und Rechenzeit durchgeführt.

Einige Funktionen verlangen den Wert des Arguments in Radian. Es gilt:

π Radian = 360 Grad

SIN

Art: Funktion

Form:

$\text{Sin}(y)$

y: numerische Variable

Beschreibung:

'SIN' ermittelt den Sinus-Wert des Funktionsarguments <y>. Der Wert von <y> muß in Radians gegeben sein.

COS

Art: Funktion

Form:

$\text{Cos}(y)$

y: numerische Wert

Beschreibung:

'COS' ermittelt den Cosinus-Wert des Funktionsarguments $\langle y \rangle$. Der Wert von $\langle y \rangle$ muß in Radians gegeben sein.

TAN

Art: Funktion

Form:

$\text{Tan}(y)$

y: numerischer Wert

Beschreibung:

'TAN' ermittelt den Tangens-Wert des Funktionsarguments $\langle y \rangle$. Der Wert von $\langle y \rangle$ muß in Radians gegeben sein.

ATN

Art: Funktion

Form:

$\text{Atn}(y)$

y: numerischer Wert

Beschreibung:

'ATN' ermittelt den Arcustangens-Wert des Funktionsarguments $\langle y \rangle$. Das Ergebnis liegt im Bereich $-\pi/2$ bis $+\pi/2$.

EXP

Art: Funktion

Form:

$\text{Exp}(y)$

y: numerischer Wert

Beschreibung:

'EXP' ermittelt den Exponential-Wert des Funktionsarguments <y>.

LOG

Art: Funktion

Form:

$\text{Log}(y)$

y: numerischer Wert

Beschreibung:

'LOG' ermittelt den natürlichen Logarithmus des Funktionsarguments <y>. Der Wert von <y> muß größer als Null sein.

SQR

Art: Funktion

Form:

$\text{Sqr}(y)$

y: numerischer Wert

Beschreibung:

'SQR' ermittelt die quadratische Wurzel des Funktionsarguments <y>. Der Wert von <y> darf nicht negativ sein.

4.5 String-Statements und -Funktionen

SET\$

Art: Statement

Form:

Set\$ String\$, Index%, Teilstring\$

String\$:	Stringvariable
Index%:	Integer
Teilstring\$:	String

Beschreibung:

'SET\$' überschreibt die Zeichen der Stringvariablen <String\$> mit Zeichen aus dem String <Teilstring\$>. <Index%> ist die Stelle innerhalb von <String\$>, wo das erste Zeichen von <Teilstring\$> plziert wird. Das Überschreiben vergrößert die ursprüngliche Länge von <String\$>, wenn die Länge von <Teilstring\$> das erfordert.

Beispiel:

```
A$ = "ABcd" : Teil$ = "xx"  
Set$ A$,2,Teil$  
Print A$  
-->Axxd
```

Siehe auch:

INSERT\$

DELETE\$

Art: Statement

Form:

Delete\$ String\$, Index%, Anzahl%

String\$: Stringvariable

Index%: Integer

Anzahl%: Integer

Beschreibung:

'DELETE\$' löscht in der Stringvariablen <String\$> ab der Stelle <Index%> <Anzahl%> Zeichen. <String\$> wird entsprechend kürzer.

Beispiel:

```
A$ = "ABcd"  
Delete A$,2,2  
Print A$  
-->Ad
```

INSERT\$

Art: Statement

Form:

Insert\$ String\$, Index%, Teilstring\$

String\$: Stringvariable

Index%: Integer

Teilstring\$: String

Beschreibung:

'INSERT\$' fügt <Teilstring\$> an der Stelle <Index%> in <String\$> ein.

Beispiel:

A\$ = "Ad" : Teil\$ = "Bc"

Insert\$ A\$,2,Teil\$

Print A\$

-->ABcd

Siehe auch:

SET\$

INSTR%

Art: Funktion

Form:

Instr%(String\$, Teilstring\$)

String\$: String

Teilstring\$: String

Beschreibung:

'INSTR%' prüft ob <Teilstring\$> in <String\$> enthalten ist. Wenn ja, gibt 'INSTR%' die Stelle innerhalb von <String\$> zurück, an der <Teilstring\$> beginnt. Andernfalls wird 0 zurückgegeben.

Beispiel:

```
A$ = "ABcd"  
index% = Instr% (A$, "Bc")  
Print index%  
-->2
```

LEFT\$

Art: Funktion

Form:

Left\$(String\$, Anzahl%)

String\$: String
Anzahl%: Integer

Beschreibung:

'LEFT\$' bildet aus dem 'linken' Teil von <String\$> einen Teilstring der Länge <Anzahl%>, d.h. die ersten <Anzahl%> Zeichen bilden den zurückgegebenen neuen String. <String\$> bleibt unverändert.

Beispiel:

```
A$ = Left$("ABcd",2)
Print A$
-->AB
```

Siehe auch:

RIGHT\$, MID\$

RIGHT\$

Art: Funktion

Form:

Right\$(String\$, Anzahl%)

String\$: String
Anzahl%: Integer

Beschreibung:

'RIGHT\$' bildet aus dem 'rechten' Teil von <String\$> einen Teilstring der Länge <Anzahl%>, d.h. die letzten <Anzahl%> Zeichen bilden den zurückgegebenen neuen String. <String\$> bleibt unverändert.

Beispiel:

```
Print Right$("ABcd",2)
-->cd
```

Siehe auch:

LEFT\$, MID\$

MID\$

Art: Funktion

Form:

Mid\$(String\$, Index%, Anzahl%)

String\$:	String
Index%:	Integer
Anzahl%:	Integer

Beschreibung:

'MID\$' bildet aus <String\$> einen Teilstring der Länge <Anzahl%> beginnend ab Zeichen <Index%>.

Beispiel:

```
A$ = "ABcd"  
A$ = Mid$(A$,2,2)  
Print A$  
-->Bc
```

Siehe auch:

LEFT\$, RIGHT\$

TRIM\$

Art: Funktion

Form:

Trim\$(String\$)

String\$: String

Beschreibung:

'TRIM\$' schneidet von <String\$> eventuell vorhandene nachfolgende Leerstellen ab.

Beispiel:

```
Rectype Rtype
  Recvar A$ 12
Endrec
Record Rtype R
R.a$ = "123"
B$ = R.a$+"!"
Print B$
-->123      !
B$ = Trim$(R.a$)+"!"
Print B$
-->123!
```

Siehe auch:

LTRIM\$

LTRIM\$

Art: Funktion

Form:

Ltrim\$(String\$)

String\$: String

Beschreibung:

'LTRIM\$' schneidet von <String\$> eventuell vorhandene führende Leerstellen ab.

Beispiel:

```
B$ = " 123"  
Print B$  
--> 123  
B$ = Trim$(B$)  
Print B$  
-->123
```

Siehe auch:

TRIM\$

LCASE\$

Art: Funktion

Form:

Lcase\$(String\$)

String\$: String

Beschreibung:

'LCASE\$' wandelt alle Großbuchstaben in <String\$> in Kleinbuchstaben im zurückgegebenen String. <String\$> bleibt unverändert.

Beispiel:

```
B$ = "ABcd"  
Print B$  
-->ABcd  
B$ = Lcase(B$)  
Print B$  
-->abcd
```

Siehe auch:

UCASE\$

UCASE\$

Art: Funktion

Form:

Ucase\$(String\$)

String\$: String

Beschreibung:

'UCASE\$' wandelt alle Kleinbuchstaben in <String\$> in Großbuchstaben im zurückgegebenen String. <String\$> bleibt unverändert.

Beispiel:

```
Print Ucase$("ABcd")  
-->ABCD
```

Siehe auch:

LCASE\$

LEN%

Art: Funktion

Form:

Len%(String\$)

String\$: String

Beschreibung:

'LEN%' ermittelt die Länge von <String\$>, d.h. die Anzahl der enthaltenen Zeichen.

Beispiel:

```
Print Len%("ABcd")  
-->4
```

ASC%

Art: Funktion

Form:

Asc%(String\$)

String\$: String

Beschreibung:

'ASC%' liefert den ASCII-Zahlenwert des ersten Zeichens von <String\$>.

Beispiel:

```
Print Asc%("ABcd")  
-->65
```

Siehe auch:

CHR\$

CHR\$

Art: Funktion

Form:

Chr\$(n%)

n%: Integer

Beschreibung:

'CHR\$' wandelt den Wert von <n%> in ein Zeichen laut ACSII-Tabelle.
Umkehrfunktion zu 'ASC%'

Beispiel:

Print Chr\$(65)
-->A

Siehe auch:

ASC%

STR\$

Art: Funktion

Form:

Str\$(n%)

n%: numerische Konstante / Variable

Beschreibung:

'STR\$' macht aus der Zahl <n%> eine Zeichenkette.

Beispiel:

```
A$ = "***"+Str$(100.0)+"KG"
```

```
Print A$
```

```
-->***100.000000KG
```

Siehe auch:

VAL%, VAL

VAL

Art: Funktion

Form:

Val(String\$)

String\$: String

Beschreibung:

'VAL' wandelt <String\$> in eine Realzahl. Zusammen mit 'VAL%' Umkehrfunktion zu 'STR\$'.

Beispiel:

```
R = 100.0 + VAL("10.0")  
Print R  
-->110.000000
```

Siehe auch:

VAL%, STR\$

VAL%

Art: Funktion

Form:

Val%(String\$)

String\$: String

Beschreibung:

'VAL%' wandelt <String\$> in eine Integerzahl. Zusammen mit 'VAL'
Umkehrfunktion zu 'STR\$'.

Beispiel:

```
N% = 100 + VAL("10")  
Print N%  
-->110
```

Siehe auch:

VAL, STR\$

DOSTOWIN\$

Art: Funktion

Form:

DosToWin\$(String\$)

String\$: String

Beschreibung:

MS-Windows verwendet für Textzeichen den ANSI-Zeichensatz, MS-DOS dagegen den 'IBM Extended Character Set'. Diese Zeichensätze sind bei vielen Zeichen identisch z. B. bei 'A'-'Z', 'a'-'z', '0'-'9', aber schon bei Umlauten treten Differenzen auf. DOSTOWIN\$ wandelt einen String, der eine Zeichenkette nach dem MS-DOS-Zeichensatz enthält (IBM-Set), in einen String um, der den Windows-Konventionen genügt (ANSI-Set). Das kann sehr nützlich sein, wenn man z. B. eine Textdatei verarbeiten will, die unter MS-DOS erstellt wurde.

Beispiel:

```
Open "I", #1, "Text.dat"
While error%() <> 3
  Line Input #1, a$
  a$ = DosToWin$(a$)
  Print a$
Wend
Close #1
```

Siehe auch:

WINTODOS\$

WINTODOS\$

Art: Funktion

Form:

WinToDos\$(String\$)

String\$: String

Beschreibung:

WINTODOS ist die Umkehrfunktion zu DOSTOWIN. Sie wandelt eine Zeichenkette nach Windows-Konditionen (ANSI) in eine Zeichenkette nach DOS-Konventionen (IBM).

Siehe auch:

DOSTOWIN

4.6 Bildschirm-Steuerung

CLS

Art: Statement

Form:

Cls [n]

n: Integer

Beschreibung:

'CLS' löscht den Bildschirm. Optional kann eine Hintergrundfarbe mit <n> gesetzt werden. Mögliche Hintergrundfarben sind:

- | | |
|---|------------|
| 0 | Weiß |
| 1 | Hellgrau |
| 2 | Mittelgrau |
| 3 | Dunkelgrau |
| 4 | Schwarz |

Beispiel:

Cls 4 ' Schwarzer Hintergrund

COLOUR

Art: Statement

Form:

Colour v%[,h%]

v%: Integer

h%: Integer

Beschreibung:

'COLOUR' stellt die Farben für Bildschirm-Ausgaben mit 'PRINT' und für die grafischen Befehle (LINETO, ARC...) ein. <v%> ist die Vordergrundfarbe, <h%> die Hintergrundfarbe, ist letztere nicht angegeben, bleibt die Hintergrundfarbe unverändert. Die möglichen Farben sind:

0	Schwarz
1	Blau
2	Grün
3	Hellblau
4	Rot
5	Violett
6	Gelb/Braun
7	Weiß

Bei grafischen Ausgaben (LINETO, ARC...) wird die eingestellte Hintergrundfarbe zum Füllen von Zwischenräumen (z. B. bei punktierten Linien oder bei Schraffuren) benutzt. Bei Textausgabe wird der Text mit der Hintergrundfarbe unterlegt.

Wird die Hintergrundfarbe <h%> negativ eingegeben, so werden Texte nicht mit dieser Hintergrundfarbe unterlegt, die Wirkung bei grafischen Befehlen bleibt erhalten.

Standardmäßig ist die Vordergrundfarbe 'Schwarz' und die Hintergrundfarbe 'Weiß'.

Beispiel:

Colour 7,1 ' Weiß auf Blau

CURSOR

Art: Statement

Form:

Cursor n%
n%: Integer

Beschreibung:

Die Form des Cursors wird durch <n%> festgelegt. Folgende Werte sind möglich:

- 0 Pfeil (Standard)
- 1 Strich
- 2 Uhrglas
- 3 Kreuz

LOCATE

Art: Statement

Form:

```
Locate x%, y%  
x%: Integer  
y%: Integer
```

Beschreibung:

'LOCATE' setzt die aktuellen Koordinaten des Bildschirms auf Spalte <x%> und Zeile <y%>. Dabei werden Höhe und Breite des aktuell eingestellten Zeichenfonts berücksichtigt. Nachfolgende Textausgaben werden an dieser Stelle begonnen.

'LOCATE' setzt auch die Startpunkte der grafischen Funktionen (LINETO, ARC...). 'LOCATE' und 'MOVETO'/'LINETO' setzen beide die aktuellen Koordinatenwerte, erstere Funktion in Zeicheneinheiten, letztere in Pixelwerten.

Beispiel:

```
Locate 20, 15
```

Siehe auch:

MOVETO

STYLE

Art: Statement

Form:

Style linienart%, füllmuster%, stiftdicke%

linienart%: Integer

füllmuster%: Integer

stiftdicke%: Integer

Beschreibung:

'STYLE' legt Parameter für nachfolgende Grafik-Ausgaben fest. Text-Ausgaben werden nicht beeinflußt.

<linienart%> legt die Linienart für Linien, Bögen und die Umrandung von Kuchen und Rechtecken fest. Folgende Werte sind möglich:

- | | |
|---|--------------------|
| 0 | durchgezogen |
| 1 | gestrichelt |
| 2 | punktiert |
| 3 | strichpunktiert |
| 4 | strich-punkt-punkt |

Die Linien werden in der aktuellen Vordergrundfarbe gezeichnet, bei den Linienarten 1 bis 4 werden die Zwischenräume mit der Hintergrundfarbe gefüllt.
<füllmuster%> legt eine Schraffur fest, mit der das Innere von Kuchen und Rechtecken gefüllt wird:

- | | | |
|---|-------------------------|------------|
| 0 | nicht schraffiert | |
| 1 | horizontale Linien | ----- |
| 2 | vertikale Linien | |
| 3 | vorwärtsdiagonal | //////// |
| 4 | rückwärtsdiagonal | \\\\\\\\\\ |
| 5 | Kreuzschraffur | +++++++ |
| 6 | Kreuzschraffur diagonal | xxxxxxx |

Die Schraffurlinien werden mit der aktuellen Vordergrundfarbe gezeichnet, die Zwischenräume mit der Hintergrundfarbe.

<stiftdicke%> legt die Linienstärke als Mehrfaches der Standardbreite fest.

Die folgenden grafischen Befehle können ein *Geräte-Kennzeichen* enthalten, welches das Ausgabegerät kennzeichnet. Folgende Kennzeichen sind möglich:

- 1 Ausgabe auf Drucker (Printer)
- 2 Ausgabe auf Bildschirm (Screen)
- 3 Ausgabe auf Bildschirm und Drucker

Fehlt das Geräte-Kennzeichen, wird auf den Bildschirm ausgegeben.

Die tatsächliche Druckausgabe auf dem Drucker erfolgt nach Ausgabe eines 'FORMFEED'-Zeichens (siehe 'LPRINT').

MAPMODE

Art: Statement

Form:

Mapmode mode% [,G%]

mode%: Integer

G%: Integer

Beschreibung:

MAPMODE legt die Bedeutung der Punkt-Koordinaten für die Ausgabe-geräte Bildschirm und Drucker fest, dabei bedeutet <mode%>:

- 1 TEXT-Modus
- 2 METRIK-Modus

Im TEXT-Modus entspricht einer Einheit bei Koordinatenwerten ein Pixel auf dem Bildschirm und dem Drucker, im METRIK-Modus ist eine Koordinateneinheit 1/10 mm.

Im Metrik-Modus können auf verschiedenen Druckern relativ genau gleichgroße grafische Darstellungen erreicht werden, mit einer Auflösung von 1/10 mm.

<G%>: siehe Erklärung vor 'MAPMODE'.

Beispiel:

Mapmode 2, 3 'METRIK (1/10mm) auf Bildschirm und
'Drucker

MOVETO

Art: Statement

Form:

MoveTo x%, y% [,G%]

x%: Integer

y%: Integer

G%: Integer

Beschreibung:

'MOVETO' setzt die aktuellen Koordinaten des Ausgabegeräts auf den Wert <x%>, <y%>.

'MOVETO' setzt auch den Startpunkt für nachfolgende Textausgaben mit 'PRINT'. 'LOCATE' und 'MOVETO'/'LINETO' setzen beide die aktuellen Koordinatenwerte, erstere Funktion in Zeicheneinheiten, letztere in Koordinatenwerten.

<G%>: siehe Erklärung vor 'MAPMODE'.

Beispiel: siehe LINETO

Siehe auch:

LOCATE, LINETO

LINETO

Art: Statement

Form:

LineTo x%, y% [,G%]

x%: Integer

y%: Integer

G%: Integer

Beschreibung:

'LINETO' zieht eine Linie von den aktuellen Koordinaten (s. 'MOVETO') zu den Koordinaten <x%>, <y%>. Diese werden dann zu den aktuellen Koordinaten wie bei 'MOVETO'. Z. B. Polygonzüge brauchen daher nur einmal mit 'MOVETO' auf den Startpunkt gebracht werden, alle Eckpunkte und der Endpunkt werden dann durch einfache 'LINETO'-Befehle gezeichnet.

Die Linien werden mit der aktuell eingestellten Linienart und -stärke (s. STYLE) sowie mit der aktuell eingestellten Farbe (s. COLOUR) gezeichnet. Bei punktierten/strichlierten Linien werden die Zwischenräume mit der eingestellten Hintergrundfarbe gezeichnet.

<G%>: siehe Erklärung vor 'MAPMODE'.

Beispiel:

MoveTo 50, 50

LineTo 100, 50

LineTo 50, 100

LineTo 50, 50

--> Ein Dreieck wird gezeichnet.

Siehe auch:

COLOUR, STYLE

ARC

Art: Statement

Form:

Arc r%, anfang%, ende% [,G%]

r%	Integer
anfang%:	Integer
ende%:	Integer
G%	Integer

Beschreibung:

'ARC' zeichnet einen Bogen oder Kreis. <r%> ist der Radius des Bogens, <anfang%> und <ende%> sind die Anfangs- und Endwinkel in (Alt-)Grad. Der Bogen wird gegen den Uhrzeigersinn gezeichnet. Der Winkel 0 liegt in Richtung der nach unten gerichteten y-Achse.

Der Bogenmittelpunkt liegt auf den aktuellen Koordinaten (siehe 'MOVETO').

Sind Anfangs- und Endwinkel gleich, wird ein Kreis gezeichnet.

Der Bogen wird mit der aktuell eingestellten Farbe bzw. Linienart gezeichnet (siehe 'LINETO').

<G%>: siehe Erklärung vor 'MAPMODE'.

Beispiel:

```
MoveTo 300, 300, 2
Arc 100, 90, 270, 2
-->Ein nach unten offener Halbkreis wird auf den
Drucker ausgegeben.
```

Siehe auch:

COLOUR, STYLE

PIE

Art: Statement

Form:

Pie r%, anfang%, ende% [,G%]

r%	Integer
anfang%:	Integer
ende%:	Integer
G%:	Integer

Beschreibung:

'PIE' zeichnet einen Kuchen. Ein Kuchen besteht aus einem Bogen, dessen Endpunkte durch Linien mit dem Bogen-Mittelpunkt verbunden sind, das Innere kann mit Farbe/Schraffur ausgefüllt sein. <r%> ist der Radius des Bogens, <anfang%> und <ende%> sind die Anfangs- und Endwinkel in (Alt-)Grad. Der Bogen wird gegen den Uhrzeigersinn gezeichnet. Der Winkel 0 liegt in Richtung der nach unten gerichteten y-Achse.

Sind Anfangs- und Endwinkel gleich, wird ein ausgefüllter Kreis gezeichnet.

Der Kuchenmittelpunkt liegt auf den aktuellen Koordinaten (siehe 'MOVETO').

Der Bogen und die Linien werden mit der aktuell eingestellten Farbe bzw. Linienart gezeichnet. Das Innere des Kuchens wird mit der aktuell eingestellten Schraffur gefüllt (siehe STYLE).

<G%>: siehe Erklärung vor 'MOVETO'.

Siehe auch:

COLOUR, STYLE

RECT

Art: Statement

Form:

Rect breite%, höhe% [,G%]

breite%: Integer

höhe%: Integer

G%: Integer

Beschreibung:

'RECT' zeichnet ein ausgefülltes Rechteck. <breite%> ist die Ausdehnung in x-Richtung, <höhe%> die Ausdehnung in y-Richtung.

Die linke obere Ecke des Rechtecks liegt auf den aktuellen Koordinaten.

Die Randlinien werden mit der aktuell eingestellten Farbe bzw. Linienart gezeichnet. Das Innere des Rechtecks wird mit der aktuell eingestellten Schraffur gefüllt (siehe STYLE).

<G%>: siehe Erklärung vor 'MAPMODE'.

Siehe auch:

COLOUR, STYLE

BITMAP

Art: Statement

Form:

Bitmap file\$, [G%]

file\$: String

Beschreibung

'Bitmaps' sind pixelorientierte Zeichnungen wie sie z. B. vom Windows-Programm 'Paint.exe' erzeugt werden. Durch die Anweisung BITMAP wird die in der Datei <file\$> gespeicherte Bitmap auf Bildschirm oder Drucker ausgegeben. Die linke obere Ecke der Bitmap liegt dabei auf den aktuellen Koordinaten.

<G%>: siehe Erklärung vor 'MAPMODE'.

Bitmaps können aus Zeichnungen von 'Paint.exe' mithilfe des Programms 'Bitmap.exe' erzeugt werden, siehe dazu die Erläuterung in der Einführung.

FONT

Art: Statement

Form:

Font höhe%, breite%, gewicht%, stil%, familie%

höhe%: Integer

breite%: Integer

gewicht%: Integer

stil%: Integer

familie%: Integer

Beschreibung:

MS-Windows erlaubt das Ausgeben von Zeichenketten auf Bildschirm und Drucker in verschiedenen Schriftarten oder Fonts. 'FONT' stellt eine Schriftart für nachfolgende Ausgaben mit 'PRINT' ein. Mit <familie%> wird die Font-Familie, also das Erscheinungsbild der Zeichen eingestellt:

- 1 Times Roman
- 2 Helvetica
- 3 Swiss
- 4 Script (Schreibschrift)
- 5 Roman
- 6 System (Standard)
- 7 Courier (Schreibmaschine)

Mit <höhe%> und <breite%> wird die Größe in Punkten (Begriff aus der Setztechnik, 1 Punkt gleicht ungefähr 1/72 Zoll) festgelegt. Verschiedene Font-Familien sind in MS-Windows in abgestuften Größen vorhanden, andere wiederum sind stufenlos skalierbar. Information über implementierte Fonts und verfügbare Größen für den Bildschirm und Ihren Drucker gibt Ihnen das mitgelieferte Programm 'fonts.exe'. Hinter den Schriftproben werden Höhe und Breite des angezeigten Fonts mit ^ bzw > angezeigt.

Skalierbare Fonts werden dort nur in einer Größe (30 Punkte?) dargestellt, mit Ausnahme des Systemfonts, der nicht skalierbar ist aber nur in einer Größe verfügbar ist. Skalierbare Fonts können in jeder Höhe und Breite generiert werden.

Auch bei nicht skalierbaren Fonts kann man Zwischengrößen erhalten. MS-Windows synthetisiert sie aus den vorhandenen. Die exakte Einhaltung der vorgegebenen Werte für Höhe und Breite ist jedoch nicht gewährleistet und die

Qualität der Zeichen i. a. schlechter.

<gewicht%> legt fest, wie fett die Zeichen sind. Z. Zt. sind zwei Werte verfügbar:

400 Normal
700 Fett

Alle anderen Werte werden auf diese reduziert.

Mit <stil%> kann man noch besondere Darstellungen verlangen:

0 Normal
1 Kursiv
2 Unterstrichen
3 Kursiv und unterstrichen

Beispiel

Font 18, 12, 700, 0, 1
Print "Hello Windows"
Es wird der Font Times Roman fett in der Höhe 18, Breite 12
eingestellt.

4.7 Verschiedenes

REM

Art: Statement

Form:

```
Rem Str  
Str:      Kommentar
```

Beschreibung:

'REM' leitet eine Kommentarzeile ein. <Str> ist ein Kommentar, der vom Übersetzer nicht ausgewertet wird. 'REM' ist eine nichtausführbare Anweisung.

Beispiel:

```
Rem Dies ist ein Kommentar
```

Siehe auch:

,

'

Art: Statement

Form:

'Str

Str: Kommentar

Beschreibung:

' leitet einen Kommentar ein. <Str> ist ein Kommentar, der vom Übersetzer nicht ausgewertet wird. ' ist eine nichtausführbare Anweisung. ' kann auch auf eine andere Anweisung in der Zeile folgen.

Beispiel:

x% = 0 'Dies ist ein Kommentar

Siehe auch:

REM

ERROR%()

Art: Funktion

Form:

Error%()

Beschreibung:

'ERROR%()' liefert den zuletzt entstandenen Runtime-Error zurück.

Beispiel:

```
If Error%() <> 0 Then  
    Print "Laufzeit-Fehler"  
Endif
```

EXEC

Art: Statement

Form:

Exec Programm\$, Com\$, Show%

Programm\$: String

Com\$: String

Show%: Integer

Beschreibung:

Mit 'EXEC' können während der Abarbeitung eines Programms weitere Programme geladen und gestartet werden.

<Programm\$> ist der Name des zu startenden Programms, <Com\$> eine Kommandozeile, die dem Programm beim Start übergeben wird und <Show%> legt fest, in welcher Art das Hauptfenster des Programms beim Start dargestellt wird (Symbol, überlappendes Fenster...). Die möglichen Werte für <Show%> sind dieselben wie bei der Anweisung 'SHOW' (siehe dort).

<Programm\$> wird nur gestartet, wenn es nicht bereits läuft.

Das erste Zeichen von <Com\$> muß ein Leerzeichen sein!

Das startende Programm läuft weiter und kann parallel zum gestarteten Programm arbeiten.

Beispiel:

Exec "abc.exe", " abc.dat", 1

Siehe auch

SHOW

TIMEDATE\$

Art: Funktion

Form:

Timedate\$(n%)

n%: Integer

Beschreibung:

'TIMEDATE\$' ermittelt Datum bzw. Uhrzeit. <n%> ist ein Parameter, der das gewünschte Ergebnis selektiert:

n%=1	Zeit
n%=2	Datum amerikanische Form
n%=3	Datum deutsche Form

Beispiel:

```
Print Timedate$(1)
-->17:24:12
Print Timedate$(2)
-->10/28/89
Print Timedate$(3)
-->28.10.89
```

TRACE

Art: Statement

Form:

1. Trace on
2. Trace off
3. Trace var[, var...]

var: Variable

Beschreibung:

'TRACE ON' versetzt das Programm in einen Einzelschritt-Modus. Es wird jeweils eine Zeile des Programmtexts ausgeführt (nicht eine Anweisung, da eine Zeile mehrere Anweisungen beinhalten kann!). Dann wird das Fenster des Editors aufgeblendet und die nächste auszuführende Zeile invers dargestellt. Durch eine Messagebox kann der Benutzer die nächste Zeile ausführen lassen oder den Einzelschritt-Modus abbrechen (Knopf "Abbrechen").

'TRACE OFF' beendet den Einzelschritt-Modus.

Die dritte Variante gibt eine Reihe von Variablen (max. sechs) in einem eigenen Fenster auf dem Bildschirm zur Inspektion aus. Dabei muß der Einzelschritt-Modus nicht eingeschaltet sein.

Alle TRACE-Befehle werden nur dann wirksam, wenn das Programm mit der Option 'Debug' übersetzt und gestartet wurde (siehe Abschnitt 2.4).

Beispiel

Trace k%, a\$(k%)

der Index k% und das entsprechende Element des Feldes a\$ werden in einem separaten Fenster angezeigt.

4.8 Windows

FILES

Art: Statement

Form:

Files Dir\$, Match\$, File\$

Dir\$: String

Match\$: String

File\$: String

Beschreibung:

'FILES' zeigt eine Dateiauswahlbox auf dem Bildschirm. Die anfangs angebotenen Dateien werden durch ein Verzeichnis in <Dir\$> und ein Suchmuster in <Match\$> definiert.

Nach Beenden des Dialogs wird der ausgewählte Dateiname in <File\$> zurückgegeben. Wurde der Dialog abgebrochen ist dies ein Leerstring "".

Beispiel:

Files "\basic", "*.bas", file\$

listet alle Dateien *.bas in Verzeichnis /basic zur Auswahl auf.

DIALOG

Art: Statement

Form:

1. Dialog x%, y%, xd%, yd%, IDitem%, art%, text\$
2. Dialog
3. Dialog @

x%, y%	Integer
xd%, yd%	Integer
IDitem%	Integer
art%	Integer
text\$:	String

Beschreibung:

'DIALOG' erzeugt Dialogboxen. In der 1. Form werden die einzelnen Items (Elemente) einer Dialogbox definiert. In dieser Form wird 'DIALOG' für jedes Item der Dialogbox einmal aufgerufen, bis alle Items definiert sind. Unmittelbar anschließend wird durch die 2. Form die Dialogbox gestartet. Die 3. Form beendet den Dialog und entfernt die Dialogbox.

Alle Anweisungen 'DIALOG' zu einer bestimmten Dialogbox müssen in einer einzigen SUBROUTINE stehen, welche den Dialog steuert. Am Beginn der SUBROUTINE stehen die Definitions-Anweisungen (1. Form), gefolgt von der Start-Anweisung (2. Form). Auf diese Startanweisung folgen ein oder mehrere Windows-Sprungmarken bis zur 'ENDSUB'-Anweisung, welche die SUBROUTINE beendet. Die Windows-Sprungmarken werden von MS-Windows angesprungen, wenn die Items der Dialogbox mit einer entsprechenden Identifikations-Nummer <id%> vom Programm-Benutzer angewählt werden. (siehe Beispiel unten und im Demo-Programm 'Dialog.bas'). Diese Subroutine, die alle Anweisungen für eine bestimmte Dialogbox enthält, wird im folgenden als Dialogbox-Subroutine bezeichnet.

Im 1. Aufruf der 1. Form werden Daten für den äußeren Rahmen der Dialogbox übergeben. <x%> und <y%> sind die Koordinaten der linken oberen Ecke der Dialogbox bezogen auf die linke obere Ecke des Hauptfensters, <dx%> und <dy%> ihre Ausdehnung in x- und y-Richtung. <IDitem%> ist ohne Bedeutung und sollte '0' sein. Als <art%> wird der Wert '0' übergeben (siehe auch Aufstellung unten). Der String <text\$> wird als Titel der Dialogbox übernommen.

Die weiteren Aufrufe der 1. Form definieren die einzelnen Items der Dialogbox. <x%> und <y%> sind die Koordinaten der linken oberen Ecke des Items bezogen

auf die linke obere Ecke der Dialogbox, <dx%> und <dy%> die Ausdehnung in x- und y-Richtung. <IDitem%> ist eine eindeutige Kennzahl, sie wird bei Aktivierung ('Drücken') des Knopfes durch den Programm-Benutzer als 'Message' zur Identifizierung übernommen.

<IDitem%> muß zwischen 3 und 99 liegen! Bei Drücken der Enter-/ Return-Taste wird von Windows eine 1 als Message generiert, bei Drücken von ESC eine 2. Kennzahlen höher als 99 sind anderweitig belegt.

<x%>, <y%>, <dx%>, <dy%> sind keine Pixelwerte, sondern an die Größe des System-Zeichensatzes gekoppelt! In x-Richtung ist die Einheit 1/4 Zeichenbreite, in y-Richtung 1/8 Zeichenhöhe. (gilt auch für den Rahmen)

<art%> legt die Funktion des Items fest:

- | | |
|----|-------------------------------------------|
| 0 | 1. Aufruf, Dialograhmen generieren |
| 1 | CHECKBOX: Kreuzkästchen |
| 5 | LTEXT: linksbündiger konstanter Text |
| 10 | GROUPBOX: Box um eine Gruppe Leuchtknöpfe |
| 11 | PUSHBUTTON: Druckknopf |
| 12 | RADIOBUTTON: Leuchtknopf |
| 13 | DEFBUTTON: Standard-Druckknopf |
| 18 | LEDIT: linksbündiges Editierfeld |

LTEXT und GROUPBOX können nicht vom Programm-Benutzer angewählt werden und daher auch keine Messages erzeugen. Bei der Definition solcher Items kann <IDitem%> einfach '-1' gesetzt werden.

CHECKBOX erzeugt ein kleines Quadrat, das mit einem diagonalen Kreuz (Checkmarke) markiert werden kann und mit der Beschriftung Text versehen ist. Das Quadrat ist immer gleich groß, dx und dy beeinflussen den verfügbaren Platz für Text. Bei Anwahl dieses Knopfes durch den Programm-Benutzer empfängt die Dialogbox-Subroutine die Message IDitem.

LTEXT erzeugt die konstante linksbündige Zeichenkette Text. Dieses Item kann durch den Benutzer nicht angewählt werden und sendet daher keine Message.

GROUPBOX wird im allgemeinen dazu verwendet, eine Gruppe von Leuchtknöpfen gegen die Umgebung abzugrenzen. Text wird als Titel der Box verwendet. Dieses Item kann durch den Benutzer nicht angewählt werden und sendet daher keine Message.

PUSHBUTTON erzeugt einen Druckknopf mit der Beschriftung Text. Bei Anwahl dieses Knopfes durch den Programm-Benutzer empfängt die Funktion Dialogbox-Subroutine die Message IDitem.

RADIOBUTTON erzeugt einen Leuchtknopf. Dieser besteht aus einem Kreis, der bei Anwahl durch einen dicken Punkt ausgefüllt wird und damit 'leuchtet'.

Leuchtknöpfe treten in Gruppen auf, nur einer der Leuchtknöpfe leuchtet, die anderen sind ausgeschaltet. Leuchtknöpfe dienen im allgemeinen zur Auswahl einer Alternative. Damit eine Gruppe von Leuchtknöpfen durch die Anweisung DLGITEM ge-steuert werden kann, müssen die IDitems einer Gruppe Leuchtknöpfe lückenlos aufeinanderfolgen. Text dient zur Beschriftung des Leuchtknopfes. Bei Anwahl eines Leuchtknopfes durch den Programm-Benutzer empfängt die Funktion Dialogbox-Subroutine die Message IDitem.

DEFBUTTON unterscheidet sich dadurch von PUSHBUTTON, daß der erzeugte Knopf als 'Standard-Antwort' angesehen wird. Der Knopf hat eine stärkere Umrandung und löst bei Anwahl durch den Benutzer die Message IDitem aus. Die Dialogbox-Subroutine sollte so programmiert werden, daß bei Drücken der Enter/Return-Taste - was die Message '1' auslöst - dieselbe Reaktion erfolgt wie beim direkten Anwählen.

LEDIT erzeugt ein linksbündiges Editierfeld. Text wird dem Programm-Benutzer als anfänglicher Inhalt des Editierfeldes angeboten. Wenn das Editierfeld den 'Input-Focus' hat, kann der Programm-Benutzer den Text über die Tastatur bearbeiten. Den Input-Focus erhält das Feld entweder durch Anwahl des Feldes durch den Benutzer, durch die Anweisung DLGITEM (siehe dort) oder automatisch beim Start des Dialogs, wenn es das erste Editierfeld des Dialogs ist.

Solange die Dialogbox aktiv ist, ist das Hauptprogramm deaktiviert, es kann aber mit anderen Programmen gearbeitet werden. Ein Beenden eines Dialogs ist nur aus der Dialogbox-Subroutine her möglich.

Beispiel:

Programm 'Dialog.demo' auf der Diskette

Siehe auch:

DLGITEM

DLGITEM

Art: Statement

Form:

1. Dlgitem id%, text\$
2. Dlgitem id%, id1%, id2%, modus%, text\$

id%	Integer
id1%, id2%	Integer
modus%	Integer
text\$:	Stringvariable

Beschreibung:

In der 1. Form kann mit 'DLGITEM' der aktuelle Inhalt eines Editierfeldes einer Dialogbox gelesen werden. <id%> ist die Identifizierung eines Editierfeldes, wie sie beim Aufbau der Dialogbox mit DIALOG vergeben wurde, und <text\$> eine Stringvariable, die den Inhalt aufnimmt.

In der 2. Form wird der Inhalt eines Dialog-Element bzw. -Items geändert werden. Drei Fälle sind zu unterscheiden:

- <id%> ist die Identifizierung eines Editierfelds. Dann wird <text\$> zum Inhalt des Editierfelds. <id1%>, <id2%> müssen null sein. Ist <modus%> = 2, so erhält dieses Editierfeld den Input-Focus, d. h. Tastatur-Eingaben gehen in dieses Feld, ist <modus%> = 0, bleibt der Input-Focus, wo er ist.

- <id%> ist die Identifizierung eines Leuchtknopfes. <text\$> wird zur Beschriftung des Leuchtknopfes. Der Leuchtknopf <id%> wird beleuchtet, alle anderen Leuchtknöpfe mit Kennungen zwischen <id1%> und <id2%> werden abgeschaltet. Ist <modus%> = 2, so erhält Leuchtknopf <id%> den Input-Focus, d. h. Tastatur-Eingaben mit Pfeilen oder TAB werden nun von diesem Dialog-Item aus befolgt.

- <id%> ist die Identifizierung einer Checkbox. <id1%> und <id2%> müssen null sein, <text\$> wird zur Beschriftung der Checkbox. Wenn <modus%> = 4 ist, wird in der Box eine Checkmarke gesetzt, sonst wird sie zurückgenommen. Ist <modus%> = 2, so erhält Checkbox <id%> den Input-Focus, d. h. Tastatur-Eingaben mit Pfeilen oder TAB werden nun von diesem Dialog-Item aus befolgt. Ist <modus%> = 6 werden die Aktionen für 2 und 4 ausgeführt.

Für alle Fälle gilt: ist <text\$> 'chr\$(1)' bleibt der aktuell gezeigte Text unverändert.

Beispiel:

Dlgitem 6, a\$ 'Inhalt von Editierfeld 6 wird in
'a\$ gelesen
Dlgitem 6, 0, 0, 2, "abc"
'Inhalt von Editierfeld 6 wird auf
"abc" gesetzt, Input-Focus auf
'dieses Feld.
Dlgitem 13, 10, 15, 4, "Text"
'Leuchtknopf 13 aus der Gruppe
'10-15 wird beleuchtet.
Dlgitem 19 0, 0, 4, chr\$(1)
'Checkbox 19 erhält Checkmarke,
'Text bleibt unverändert

Siehe auch:

DIALOG

FORM

Art: Statement

Form:

Form Satz, Prompt\$, Antwort%[, x%, y%]

Satz: Record

Format\$ String

Antwort% Integer

x%, y%: Integers

Beschreibung:

'FORM' erfasst Daten für die Elemente des Records <Satz>. Der String <Prompt\$> enthält die konstanten Teile der Erfassungsmaske (durch Semikolons getrennt), nämlich eine Überschrift und die einzelnen Eingabe-Aufforderungen für die Felder.

'FORM' erzeugt eine Dialogbox auf dem Bildschirm. <x%> und <y%> sind die linke obere Ecke der Dialogbox, bei Fehlen werden Standardwerte angenommen.

In <Antwort%> wird beim Beenden des Erfassungsvorgangs durch den Benutzer der zur Beendung des Dialogs angewählte Knopf zurückgegeben:

1 "OK"-Knopf (oder CR-Taste)

2 "ESC"-Knopf (oder ESC-Taste)

Soll ein Element des Records nicht erfasst werden, wird die zugehörige Eingabe-Aufforderung in String <Prompt\$> in ein Paar runder Klammern gesetzt. Das Element wird dann nur angezeigt, kann aber nicht geändert werden.

Beispiel:

```
Rectype Asatz
  Recvar Nr$ 16
  Recvar Bestand% 6
  Recvar Bezeichnung$ 30
Endrec
Record Asatz Artikelsatz

Open "R", #1, "Artikel.dat"
Get #1, 101, Artikelsatz
S$ = "Artikelstamm;(Artikelnr.);
S$ = S$ + "Bestand;;Bezeichnung:"
Form Artikelsatz, S$, Button%
If Button% = 1 Then Put #1, 101, Artikelsatz
Close #1
```

Daten für Artikel 401 können erfasst werden, die Artikelnummer kann jedoch nicht verändert werden. Wird der Dialog mit "OK" beendet, so wird der Satz zurückgespeichert, ansonsten bleibt er unverändert.

MENU

Art: Statement

Form:

Menu Str\$

Str\$: String-Feld

Beschreibung:

Durch eine 'MENU'-Anweisung wird das Menu des Programms erzeugt. Ein Menu besteht aus einer Menuleiste unterhalb des Titel-Balkens, in der mehrere Menupunkte nebeneinander aufgeführt sind. Bei Anwahl dieser Menupunkte können diese entweder direkt eine Message an das Programm absetzen oder ein Abrollmenu erzeugen, d.h. ein Untermenu mit verschiedenen Menupunkten untereinander.

<Str\$> ist ein String-Feld, das mit jedem Element ein Abrollmenu, also ein Untermenu, oder einen direkt wirkenden Menupunkt erzeugt. Für ein Abrollmenu enthält ein Element von <Str\$> mehrere Stichworte (die Menubefehle) durch Leerstellen getrennt. Das erste Stichwort wird als Titel des Abrollmenus auf der Menuleiste dargestellt, die folgenden Stichworte sind die Menupunkte des Abrollmenus. Enthält ein Element von <Str\$> nur ein Stichwort, so wird es als direkt wirkender Menupunkt in der Menuleiste aufgeführt.

Soll ein Stichwort selber Leerstellen enthalten, so stellt man diese als Unterstriche dar, sie werden beim Programmablauf durch Leerstellen ersetzt.

Trennstriche innerhalb eines Abrollmenus erreicht man durch Angabe von drei Unterstrichen '___'. Besteht ein Element von Str\$ aus dem String ">>>" werden die nachfolgenden Menupunkte rechtsbündig ausgerichtet. Das letzte Element des Feldes <Str\$> muß ein Leerstring sein.

Bei Anwahl eines Menupunkts durch den Benutzer zur Programm-laufzeit wird an das WinBasic-Programm eine Message in Form eines numerischen Codes 'm0nn' abgesandt, die durch eine Windows-Sprungmarke '_m0nn' aufgefangen werden kann. Dabei ist:

m	Zählnummer des Abrollmenus in der Menuleiste
(1..x)	
nn	Zählnummer des Menupunkts innerhalb des
	Abrollmenus (0..x)

Beispiel: (siehe auch Demo-Programm 'Message.bas')

```
Dim HMenu$(4)
hMenu$(1) = "Menu1 Item11__F1 Item12"
hMenu$(2) = "Item2"
hMenu$(3) = "Menu3 Item31 Item32__SHIFT-F3 ____ Item33"
hMenu$(4) = "" 'Ende-Kennzeichen
Menu HMenu$
--> Auf der Menu-Leiste erscheint:
Menu1 Item2 Menu3
```

Bei Anwahl von 'Menu1' erscheint das Abrollmenu:

Menu1	(Message)

Item11 F1	1001
Item12	1002

Bei Anwahl von 'Item2' wird die Message '2000' abgesandt.
Bei Anwahl von 'Menu3' erscheint das Abrollmenu:

Menu3	(Message)

Item31	3001
Item32 SHIFT-F3	3002

Item33	3003

MENUITEM

Art: Statement

Form:

Menuitem item%, onoff%, check%

item% Integer

onoff% Integer

check% Integer

Beschreibung:

Mit 'MENUITEM' kann ein Befehl oder Item des Menus verändert werden.
<item%> ist die Kennzahl des Menuitems, wie sie beim Befehl 'MENU' generiert wurde.

Mit <onoff%> kann das Item ein- oder ausgeschaltet werden:

0 Menuitem ausschalten

1 Menuitem einschalten

Beim Generieren des Menus sind alle Menuitems eingeschaltet. Wird ein Menuitem ausgeschaltet mit <onoff%> = 0, wird es grau in das Menu geschrieben und kann nicht durch den Programm-Benutzer angewählt werden. Mit <onoff%> = 1 kann das Menuitem wieder eingeschaltet werden.

Mit <check%> kann ein Menuitem mit einer Checkmark in Form eines Hakens versehen werden:

0 Haken löschen

1 Haken setzen

Beispiel:

Menuitem 1001, 0, 0	' 1. Item in 1. Abrollmenu 'ausschalten
Menuitem 2003, 1, 1	' 3. Item in 2. Abrollmenu 'abhaken'

Siehe auch:

MENU

MESSAGEBOX

Art: Statement

Form:

MessageBox Text\$, Titel\$, Knöpfe%, Antwort%

Text\$: String

Titel\$ String

Knöpfe% Integer

Antwort% Integer-Variable

Beschreibung:

'MESSAGEBOX' bringt eine Messagebox mit einer Meldung auf den Bildschirm und erwartet vom Benutzer eine Reaktion durch Auswahl einer der angezeigten Knöpfe.

<Text\$> ist der Inhalt der Messagebox, <Titel\$> die Überschrift in der Titelzeile. <Knöpfe%> legt Art und Anzahl der angezeigten Knöpfe fest. Folgende Werte sind möglich:

- 0 OK
- 1 OK Abbrechen
- 2 Abbrechen Wiederholen Ignorieren
- 3 Ja Nein Abbrechen
- 4 Ja Nein
- 5 Wiederholen Abbrechen

Standardmäßig ist der erste dargestellte Knopf die Standard-Auswahl, d. h. beim Drücken der Leertaste wird dieser Knopf aktiviert. Andere Knöpfe können zur Standard-Auswahl durch Addieren eines zusätzlichen Wertes:

- 10 zweiter Knopf
- 20 dritter Knopf

In <Antwort%> wird der vom Benutzer angewählte Knopf identifiziert:

- 1 OK
- 2 Abbrechen
- 3 Abbrechen
- 4 Wiederholen
- 5 Ignorieren

6 Ja
7 Nein

Beispiel:

```
file$ = "Test.dat"
```

```
MessageBox "Datei neu, Erzeugen?", file$, 1+10, a%
```

Die MessageBox hat zwei Knöpfe, 'OK' und 'Abbrechen', wovon der zweite die Standard-Auswahl ist.

SHOW

Art: Statement

Form:

Show art% [, Prog\$]

art% Integer
Prog\$ String

Beschreibung:

Mit 'SHOW' wird die Darstellungsweise des Hauptfensters des Programms <Prog\$> auf <art%> eingestellt. Fehlt <Prog\$> so wird das eigene Hauptfenster eingestellt. Als Werte für <art%> sind möglich:

0	**	Fenster verbergen
1	*	Fenster als überlappendes Fenster zeigen
2	*	Fenster als Symbol zeigen.#
3	*	Fenster als Vollbild zeigen
4		Fenster in gegenwärtiger Größe zeigen
5	*	Fenster in gegenwärtiger Größe zeigen
6	**	Fenster minimieren (Symbol)
7		Fenster als Symbol zeigen.

'Symbol' (Icon) bedeutet, das das Fenster als kleines Quadrat auf dem unteren Bildschirmrand erscheint.

'Vollbild' heißt, das das Fenster den gesamten Bildschirm einnimmt, Ein 'überlappendes Fenster' teilt sich den Bildschirm mit anderen Programmen.

Werte, die mit einem '*' bezeichnet sind, bewirken das das betreffende Fenster aktiv wird. '**' bedeutet, das ein anderes Fenster aktiviert wird.

Beispiel:

Show 7 'eigenes Fenster als Symbol
Show 0, "clock.exe" 'Uhr verbergen.

Siehe auch:

EXEC, SIZE

SIZE

Art: Statement

Form:

Size x%, y%, dx%, dy% [, Prog\$]

x%, y% Integer

dx%, dy% Integer

Prog\$ String

Beschreibung:

'SIZE' definiert die Größe des Hauptfensters von Programm <Prog\$> oder des eigenen Hauptfensters, wenn <Prog\$> fehlt.

<x%>, <y%> sind die Koordinaten des linken oberen Eckpunktes, <dx%> und <dy%> die Ausdehnung des Fensters in x- und y-Richtung.

Siehe auch:

EXEC, SHOW

CLIP

Art: Statement

Form:

Clip Get file\$	'Lesen Clipboard
Clip Put file\$	'Schreiben Clipboard
file\$: String\$	

Beschreibung:

CLIP liest oder schreibt das 'Clipboard'. Das Clipboard ist eine Einrichtung in MS-Windows, die temporär Daten aufnimmt. Mit ihm können Daten zwischen verschiedenen Programmen oder auch innerhalb eines Programms ausgetauscht werden. Die meisten Programme unter Windows sehen den Gebrauch des Clipboards unter einer Menu-Funktion Edit/Editieren vor mit Funktionen wie 'Schneiden'/'Kopieren'/'Kleben'.

Die im Clipboard gespeicherten Daten können von verschiedenem Fornat sein.

WinBasic unterstützt die Formate CL_TEXT, CL_SYLK, CL_DIF beim Lesen des Clipboards und schreibt Daten im Format CL_TEXT. Alle drei Formate sind textorientiert. CF_TEXT ist normaler Text, CL_SYLK und CL_DIF sind spezielle Text-Formate, nämlich 'Microsoft Symbolic Link Format' und 'Software Arts Data Interchange Format'. Es handelt sich hierbei um besonders strukturierte Informationen, nähere Spezifikationen entnehmen Sie bitte den Informationen der Hersteller.

Enthält das Clipboard beim Lesen andere als die genannten Formate, wird die Information ignoriert.

Der Datenaustausch mit dem Clipboard erfolgt über Dateien. <file\$> ist der Name einer Datei aus der Informationen für das Clipboard entnommen werden, bzw. in welcher die Daten des Clipboards abgelegt werden.

Beispiel

```
CLIP GET "clip.dat"
'Die Daten im Clipboard werden in die Datei
'clip.dat' gelesen
```

Anhang A

Runtime-Errors

Datei-Meldungen

- 1 Unter dieser Kennung ist bereits eine Datei geöffnet
- 2 Datei nicht geöffnet
- 3 Schreiben oder Lesen hinter Dateiende
- 4 Zu viele Dateien
- 5 Speichermangel