StarBASIC

Datentypen Fehlercodes Tastaturbelegung Klassendefinitionen Dialog-Funktionsreferenz Dialoge - Einführung Makrobefehle

StarWriter Hilfe

Gesamtverzeichnis aller Funktionen und Befehle

M

MID$() MKDIR

N

NAME

O

OCT$() ON ERROR GOTO ON n GOSUB ON n GOTO ON TIMER(n) GOSUB OPEN OPTION BASE

P

PRINT PRINT #

R

RANDOMIZE REDIM REPLY() RESET RESTORE RESUME RETURN RIGHT$() RMDIR RND() RTRIM

S

SHARED SIN() SPACE$() SPC() SQR() STATIC STOP SUB SYSTEM

T

TAB() THIS TIME$() TIMER TIMER() TYPE

U

UBOUND() UCASE$() UNLOCK USING

V

VAL() VARPTR()

W

WHILE...WEND WRITE WRITE #

Datentypen

StarBASIC kennt die folgenden Datentypen:

INTEGER

Typ	Ganze Zahlen
Wertebereich	-32768 bis 32768
Speicherbedarf	2 Bytes

LONG

Typ	Ganze Zahlen
Wertebereich	-2.147.483.648 bis 2.147.483.647
Speicherbedarf	4 Bytes

SINGLE

Typ	Gleitpunktzahlen
Wertebereich	-3,402823E+38 bis -3,402823E-38 und +3,402823E-38 bis +3,402823E+38
Speicherbedarf	4 Bytes

DOUBLE

Typ	Gleitpunktzahl
Wertebereich	-1.797693134862315E+308 bis -1.797693134862315E-308 und +1.797693134862315E-308 bis +1.797693134862315E+308
Speicherbedarf	8 Bytes

STRING

Typ	Zeichenkette variabler Länge
Länge	0 bis 32767 Zeichen
Speicherbedarf	8 Bytes plus dem String

STRING*n

Typ	Zeichenkette fester Länge
Länge	die angegebene Länge (mit Leerstellen gefüllt)
Speicherbedarf	die angegebene Länge

BASICSTRING

Typ	Interne Repräsentation Wird von Entwicklern von Libraries genutzt.

POINTER

Typ	Zeiger (wird als LONG-Zahl behandelt)
Speicherbedarf	4 Bytes

ANY

Dies ist kein Datentyp, sondern eine Art der Parametrisierung eines Unterprogramms. `ANY` sagt aus, daß bei dem entsprechenden Parameter keine Typprüfung stattfindet.

INTEGER

Eine `INTEGER`-Variable kann ganzzahlige Werte zwischen -32768 und 32767 aufnehmen. Sie belegt nur 2 Bytes im Speicher und ist daher die ökonomischste Form der Ablage von Zahlen.

LONG

eine `LONG`-Variable kann ganzzahlige Werte zwischen -2.147.483.648 und 2.147.483.647 aufnehmen. Sie belegt 4 Bytes im Speicher. Dies ist die größte ganzzahlige Variable.

SINGLE

Eine `SINGLE`-Variable nimmt Gleitpunktwerte zwischen-3,402823E+38 und -3,402823E+38 auf. Die Genauigkeit ist 6 Nachkommastellen. Sie belegt 4 Bytes im Speicher.

DOUBLE

Der Datentyp `DOUBLE` nimmt Werte zwischen -1.797693134862315E+308 und 1.797693134862315E+308 auf. Die Genauigkeit beträgt 15 Stellen. Er belegt 8 Bytes im Speicher.

STRING

Der Datentyp `STRING` nimmt beliebige Textstring bis zu einer Länge von 32767 Zeichen auf. Die Speicherbelegung errechnet sich aus der Variablen, die 8 Bytes belegt, sowie dem für den String selbst benötigten Speicher.

STRING*n

Der Datentyp `STRING*n` (wobei n die Länge des Strings angibt) nimmt Textstrings mit einer festen Länge auf. Ist der zugweiesene String zu kurz, wird er mit Leerstellen aufgefüllt. Der String belegt exakt die angegebene Anzahl von Bytes im Speicher.

BASICSTRING

Der Datentyp `BASICSTRING` ist die interne Repräsentation des Datentyps `STRING`. Er wird bei externen Unterprogrammen angegeben, die den vollen Zugriff auf die interne Struktur einer `STRING`-Variable benötigen und ist daher nur für den Ersteller von externen Libraries von Interesse.

POINTER

Eine Variable des Datentyps `POINTER` enthält die Adresse einer durch `TYPE` oder `CLASS` definierten Variable, die mit `DIM` oder `ALLOCATE` angelegt wurde. Intern wird ein Pointer als `LONG`-Zahl dargestellt, mit der auch gerechnet werden kann.

ANY

Dies ist kein Datentyp, sondern eine Art der Parametrisierung eines Unterprogramms. `ANY` sagt aus, daß bei dem entsprechenden Parameter keine Typprüfung stattfindet.

ABS()

Ermitteln des Absolutwerts eines Ausdrucks.

Verwendung

```
N = ABS (Ausdruck)
```

Parameter

`Ausdruck`	Der Ausdruck, dessen Absolutwert ermittelt werden soll.

Returnwert

Der positive Wert des Ausdrucks.

ALLOCATE

Anfordern von dynamischen Speicher für Datenstrukturen.

Verwendung

```
    ALLOCATE Pointer-Variable [, Pointer-Variable, ...]
```

Beschreibung

Für die angegebene Pointer-Variable wird ein Speicherbereich angefordert, der groß genug ist, um die Struktur aufzunehmen, auf die der Pointer zeigt. Dieser Bereich wird mit Null initialisiert und dessen Adresse wird in der Pointer-Variablen abgelegt. Anschließend kann die Struktur über den Pointer adressiert werden.

Anmerkung

Ein mit `ALLOCATE` angeforderter Speicherbereich kann mit ERASE wieder freigegeben werden.

Beispiel

```
' Beispielsprogramm einer verketteten Liste
' Die Programmdatei wird eingelesen und umgekehrt wieder ausgegeben

TYPE LISTE
     P     AS POINTER TO LISTE
     ZEILE AS STRING*80
     ZNR   AS INTEGER
END TYPE

DIM KOPF AS LISTE
DIM P AS POINTER TO LISTE

' Einlesen der Liste
NR = 0
OPEN "I",#1,"LISTE.BAS"
WHILE NOT EOF (1)
    ALLOCATE P
    LINE INPUT #1, P.ZEILE
    NR = NR + 1
    P.ZNR = NR
    P.P = KOPF.P
    KOPF.P = P
WEND

' Ausgabe der Liste mit Loeschen der Elemente
WHILE KOPF.P <> 0
    P = KOPF.P
    PRINT USING "###: &" P.ZNR, P.ZEILE
    KOPF.P = P.P
    ERASE P
WEND
END
```

AS

Beschreibung der Variablenart.

Verwendung

Die AS-Klausel wird als Teil der Anweisungen DIM, COMMON, SHARED, STATIC, SUB oder FUNCTION verwendet. Sie beschreibt den Datentyp der deklarierten Variable.

Parameter

Datentypname	Der Name des Datentyps. Neben den durch die TYPE-Anweisung definierten Datentypen für Strukturen oder der CLASS-Anweisung für Klassen kann dies noch eines der folgenden Worte sein: INTEGER - kurze Integerzahl LONG - lange Integerzahl SINGLE - kurze Gleitpunktzahl DOUBLE - lange Gleitpunktzahl STRING - String variabler Länge STRING*n - String fester Länge (n) Bei durch TYPE oder CLASS definierten Datentypen kann vor dem Typ auch eine der Klauseln POINTER TO oder PTR TO angegeben werden. Dies bewirkt, daß nicht eine Struktur, sondern nur ein Zeiger auf die Struktur definiert wird.

Beispiel

```
DIM A AS INTEGER

TYPE Struktur
      A AS INTEGER
      B AS LONG
      C AS DOUBLE
END TYPE

DIM INSTANZ AS Struktur
```

ASC()

Ermitteln des ASCII-Codes eines Stringzeichens.

Verwendung

```
N = ASC (Stringvariable)
```

Parameter

`Stringvariable`	Das erste Zeichen des in der Variable gespeicherten Strings wird als ASCII-Code zurückgeliefert.

Returnwert

Der ASCII-Code des ersten Zeichens des Strings.

Anmerkungen

Die Funktion CHR$() ist das Gegenstück zu dieser Funktion.

Beispiel

```
N = ASC ("Hallo")
PRINT N                         ' ergibt die Zahl 72
```

ATN()

Berechnen des Arcus Tangens.

Verwendung

```
N = ATN (Ausdruck)
```

Parameter

`Ausdruck`	Der numerische Ausdruck, dessen Arcus Tangens ermittelt werden soll.

Returnwert

Der Arcus Tangens des Ausdrucks.

Anmerkungen

Der Wertebereich des Parameters darf sich zwischen -Pi/2 und Pi/2 bewegen. Der Wert wird immer als lange Gleitpunktzahl berechnet, wenn der Parameter ebenfalls eine lange Gleitpunktzahl ist, sonst ist das Ergebnis einfach genau.

Querverweise

SIN(), COS(), TAN()

BEEP

Ausgabe eines Warntons.

Verwendung

```
BEEP
```

Beschreibung

Es wird ein Warnton auf dem Lautsprecher ausgegeben.

CALL

Aufruf einer Prozedur.

Verwendung

```
CALL Prozedurname [(Parameter)]
Prozedurname [Parameter]
```

Beschreibung

Die CALL-Anweisung wird für den Aufruf einer Prozedur verwendet. Optional kann das Wort CALL auch fortgelassen werden in diesem Fall dürfen die Parameter nicht geklammert werden.

Parameter

Prozedurname	Name der aufzurufenden Prozedur.

Parameter	Die Liste der Parameter. Die Parameter werden *by reference* übergeben, d.h. die formalen Parameter werden beim Aufruf des Unterprogramms durch die aktuellen Parameter ersetzt. Wird der Inhalt eines formalen Parameters geändert, wird auch der Inhalt des aktuellen Parameters geändert. Array-Variable werden durch ein nachgestelltes Klammerpaar () gekennzeichnet. Beispiel:
	
	`DIM A% (100)`
	`CALL PROZEDUR (A%())`
	
	Ein Parameter kann auch *by value* übergeben werden, wenn er geklammert wird. Dies veranlaßt den Interpreter, den Parameter als arithmetischen Ausdruck anzusehen, dessen Ergebnis (in diesem Fall nur der Parameter) in einer temporären Variablen abgelegt wird. Beispiel:
	
	`TEXT$ = "Hallo"`
	`CALL PROZEDUR ((TEXT$))`

Anmerkungen

Eine Prozedur kann auch ohne die Verwendung der CALL-Anweisung aufgerufen werden. In diesem Fall müssen die Klammern um die Parameterliste fortgelassen werden. Beispiel:

```
DECLARE PROZEDUR (A%, B%)

CALL PROZEDUR (4, 5)              ' Verwendung von CALL
PROZEDUR 4, 5                     ' Direkter Aufruf
```

Querverweise

SUB, FUNCTION, DECLARE

CDBL()

Umwandeln eines Wertes in eine lange Gleitpunktzahl.

Verwendung

```
N# = CDBL (Ausdruck)
```

Parameter

`Ausdruck`	Der Ausdruck wird ausgerechnet und in eine lange Gleitpunktzahl umgerechnet.

Returnwert

Das Ergebnis des Ausdrucks als lange Gleitpunktzahl.

Anmerkungen

Diese Funktion kann zur expliziten Typkonversion von Variablen und Teilausdrücken verwendet werden.

Querverweise

`CINT()`, `CLNG()`, `CSNG()`

CHDIR

Wechsel des aktuellen Verzeichnisses.

Verwendung

```
CHDIR Verzeichnis
```

Beschreibung

Das aktuelle Verzeichnis wird gewechselt.

Parameter

`Verzeichnis`	String, der den Namen des neuen aktuellen Verzeichnisses angibt. Der Name muß den Regeln des verwendeten Dateisystems entsprechen.

Anmerkungen

Diese Anweisung wird direkt auf das entsprechende Betriebssystemkommando abgebildet. Die dort geltenden Restriktionen gelten auch für diese Anweisung. So wird unter MS-DOS beispielsweise nur das aktuelle Verzeichnis, jedoch nicht das aktuelle Laufwerk gewechselt.

Querverweise

MKDIR, RMDIR

CHR$()

Umwandeln eines ASCII-Codes in einen String.

Verwendung

```
S$ = CHR$ (n)
```

Parameter

`n`	Der ASCII-Code des zu erzeugenden Zeichens.

Returnwert

Ein String mit einer Länge von einem Zeichen, das das dem ASCII-Code entsprechende Zeichen darstellt.

Anmerkungen

Die Funktion ASC() ist das Gegenstück zu dieser Funktion.

CINT()

Umwandeln eines Wertes in eine kurze Integerzahl.

Verwendung

```
N% = CINT (Ausdruck)
```

Parameter

`Ausdruck`	Der Ausdruck wird ausgerechnet und in eine kurze Integerzahl umgerechnet.

Returnwert

Das Ergebnis des Ausdrucks als kurze Integerzahl.

Anmerkungen

Diese Funktion kann zur expliziten Typkonversion von Variablen und Teilausdrücken verwendet werden.

Querverweise

`CLNG()`, `CSNG()`, `CDBL()`

CLASS

Definition einer Klasse

Verwendung

```
CLASS Klassenname [PRIVATE|PROTECTED|PUBLIC Basisklassenname]
      [PRIVATE|PROTECTED|PUBLIC] Variable
      [PRIVATE|PROTECTED|PUBLIC] [VIRTUAL] Unterprogramm
END CLASS

Variable:
      Variable AS [POINTER|PTR TO] Datentyp

Unterprogramm:
      FUNCTION|SUB Name [CDECL] [LIB "Name"] ([Parameter]) [AS Typ]

Parameter:
      [OPTIONAL] [BYVAL] Parametername [Datentyp | AS Typ]
```

Beschreibung

Diese Anweisung deklariert einen neuen, zusammengesetzten Datentyp mit eingebauter Intelligenz. Die einzelnen Variablen und Methoden der Klasse werden innerhalb der `CLASS...END CLASS`-Anweisung deklariert. Es können weder Arrays noch Strings variabler Länge deklariert werden die Verwendung von Strukturen und Klassen innerhalb der Klasse ist jedoch möglich. Die Deklaration von Variablen dieses Datentyps erfolgt mit Hilfe der DIM-Anweisung. Der Zugriff auf einzelne Elemente einer solchen Variablen erfolgt, indem dem Variablennamen erste ein Punkt und dann der Name des Elements nachgestellt wird.

Wird dem Datentyp die Klausel POINTER TO bzw. PTR TO vorangestellt, was nur bei durch mit TYPE deklarierten Datentypen möglich ist, wird keine Struktur bzw. Klasse, sondern nur ein Zeiger darauf eingetragen.

Die Definition der Methoden ist analog der DECLARE-Anweisung mit der Ausnahme, daß die Sprachangabe `PASCAL` nicht gestattet ist.

Parameter

`PRIVATE`	Das entsprechende Element ist nur für Methoden der eigenen Klasse zugänglich. Es ist gegen Zugriffe aus dem Hauptprogramm heraus oder aus Methoden von abgeleiteten Klassen gesperrt. Dies ist die Voreinstellung für alle Elemente einer Klasse. Wird `PRIVATE` hinter dem Klassennamen angegeben, werden alle `PUBLIC`-Elemente der Basisklasse automatisch für diese Klasse als `PRIVATE` deklariert. Wird die Angabe ohne eine Deklaration als einzelnes Wort einer Zeile gemacht, gilt sie für alle nachfolgenden Deklarationen.
`PROTECTED`	Das entsprechende Element ist nur für Methoden der eigenen sowie von abgeleiteten Klassen zugänglich. Es ist gegen Zugriffe aus dem Hauptprogramm heraus gesperrt. Wird `PROTECTED` hinter dem Klassennamen angegeben, werden alle `PUBLIC`-Elemente der Basisklasse automatisch für diese Klasse als `PROTECTED` deklariert. Wird die Angabe ohne eine Deklaration als einzelnes Wort einer Zeile gemacht, gilt sie für alle nachfolgenden Deklarationen.

PUBLIC	Das entsprechende Element ist frei zugänglich und entspricht somit einem in einer TYPE-Anweisung definierten Element. Wird PRIVATE hinter dem Klassennamen angegeben, verbleiben alle PUBLIC-Elemente der Basisklasse PUBLIC. Wird die Angabe ohne eine Deklaration als einzelnes Wort einer Zeile gemacht, gilt sie für alle nachfolgenden Deklarationen.
VIRTUAL	Die Methode wird ausdrücklich als virtuell deklariert. Normalerweise werden alle Methoden automatisch als virtuell deklariert. Dieses Verhalten kann jedoch per Kommandozeilen-Option oder mit Hilfe des Metakommandos $NONVIRTUAL unterbunden werden, so daß virtuelle Methoden explizit als solche deklariert werden müssen. Konstruktoren und Destruktoren sind jedoch immer virtuell.
FUNCTION | SUB	Ein Unterprogramm wird entweder als Funktion oder als Unterprogramm deklariert.
Programmname[Datentyp]	Dies ist der Name des Unterprogramms. Optional kann bei Funktionen der Typ der Funktion als Suffixzeichen (s.o.) folgen. Werden sowohl ein Suffixzeichen als auch die AS-Klausel verwendet, müssen beide Datentypen übereinstimmen. Bei Methodendefinitionen setzt sich der Name aus dem Klassennamen, gefolgt von einem Punkt, sowie dem Namen der Methode zusammen.
CDECL | PASCAL	Angabe der Aufrufart. Erfolgt keine Angabe, wird angenommen, daß es sich um ein Unterprogramm nach Pascal-Konventionen handelt, wenn in der Parameterliste eines der Worte BYVAL oder BASICSTRING verwendet wird, wenn es sich um eine Funktion des Typs BASICSTRING handelt oder wenn die LIB-Angabe verwendet wird.
LIB "Libraryname"	Optional kann die Angabe der Library erfolgen, in der die Routine zu suchen ist. Unter Windows kann dies der Name einer DLL sein. In diesem Fall wird versucht, die Routine innerhalb der angegebenen DLL zu finden.
LINK nnnn	Bei Angabe dieser Klausel wird eine dynamische Verbindung zu einer anderen Applikation aufgebaut. Der angegebene Library-Name bezeichnet dabei den Namen der Applikation.
(Parameter)	Die Liste der Parameter ist optional. Wird die Routine ohne Parameter aufgerufen, muß ein leeres Klammerpaar () angegeben werden.
AS Typ	Der Typ einer Funktion kann auch durch eine AS-Klausel angegeben werden. Wird die AS-Klausel verwendet, muß der Datentyp-Suffix, falls angegeben, genau dem angegebenen Typ entsprechen.

Die Definition eines Parameters ist wie folgt:

OPTIONAL	Dieses Wort deklariert alle folgenden Parameter als optional. Werden optionale Parameter nicht angegeben, werden diese beim Funktionsaufruf mit 0 bzw. dem NULL-Pointer initialisiert.
BYVAL	Wird das Wort BYVAL verwendet, wird der Parameter *by value* übergeben. Es wird also kein Pointer auf den Wert, sondern der Wert selbst übergeben.
Parametername	Dies ist der Name des Parameters. Er ist frei vergebbar.

`Datentyp`	Der Datentyp eines Parameters kann wie in der DIM-Anweisung entweder durch den Datentyp-Suffix oder durch die AS-Klausel definiert werden.
`AS Typ`	Wird kein Datentyp-Suffix verwendet, kann der Datentyp mit der AS-Klausel definiert werden. Der Datentypname ist einer der Namen INTEGER, LONG, SINGLE, DOUBLE, STRING, BASICSTRING oder ANY oder auch der Name einer mit der TYPE-Anweisung definierten Datenstruktur. Der Datentyp ANY sagt aus, daß keine Typüberprüfung stattfindet. Es ist die Aufgabe des Unterprogramms, herauszufinden, um welchen Datentyp es sich handelt.

Der Datentyp ist einer der folgenden Suffixe:

%	INTEGER
&	LONG
!	SINGLE
#	DOUBLE
$	STRING

Der Datentypname ist einer der Namen INTEGER, LONG, SINGLE, DOUBLE, STRING oder BASICSTRING. Namen von mit TYPE definierten Datenstrukturen können nur als Parameter, jedoch nicht als Returnwert verwendet werden.

Querverweise

TYPE, DECLARE

CLNG()

Umwandeln eines Wertes in eine lange Integerzahl.

Verwendung

```
N& = CLNG (Ausdruck)
```

Parameter

`Ausdruck`	Der Ausdruck wird ausgerechnet und in eine lange Integerzahl umgerechnet.

Returnwert

Das Ergebnis des Ausdrucks als lange Integerzahl.

Anmerkungen

Diese Funktion kann zur expliziten Typkonversion von Variablen und Teilausdrücken verwendet werden.

Querverweise

`CDBL()`, `CINT()`, `CSNG()`

CLOSE

Schließen einer oder mehrerer Dateien.

Verwendung

```
CLOSE [Kanalnummer, ...]
```

Beschreibung

Diese Anweisung schließt alle über die angegebenen Kanalnummern angesprochenen Dateien. Werden keine Kanalnummern angegeben, werden alle offenen Dateien geschlossen.

Parameter

`Kanalnummer`	Die Kanalnummer der anzusprechenden Datei. Dies ist ein Integer-Ausdruck. Vor dem Ausdruck kann optional auch ein Doppelkreuz (#) stehen, um den Ausdruck als Kanalnummer zu kennzeichnen.

Beispiel

Schließen der Kanalnummern 1 bis 5:

```
FOR I% = 1 TO 5
    CLOSE #I%
NEXT
```

Querverweise

OPEN

COMMON

Deklaration von Variablen, auf die von mehreren Modulen aus zugegriffen werden kann.

Verwendung

```
COMMON [SHARED] [/Blockname/] Variable [AS Typ] [, ...]
```

Beschreibung

Die angegebenen Variablen werden als global für andere Module zugänglich deklariert. Optional kann der Block auch mit einem Namen versehen werden. Ein COMMON-Block kann auch von Modulen angesprochen werden, die sich in untergeordneten Libraries befinden. Arrays müssen nach der COMMON-Anweisung mit der DIM-Anweisung deklariert werden.

Ein COMMON-Block kann durch mehrere COMMON-Anweisungen beschrieben werden.

Parameter

SHARED	Dieses optionale Wort kennzeichnet die deklarierten Variablen als für Unterprogramme zugänglich.
Variable	Name der zu deklarierenden Variablen. Die Typkennung kann entfallen, wenn der Typ entweder durch eine DEFxxx-Anweisung oder durch die AS-Klausel vorgegeben ist. Arrays werden durch zwei Klammern "()" kenntlich gemacht. Optional kann zwischen den Klammern die Anzahl der Dimensionen angegeben werden. Dadurch kann bereits während der Compilierung des Programms eine Überprüfung der Anzahl der Dimensionen eines Arrays erfolgen.
AS Typ	Deklaration des Variablentyps. Neben den durch die TYPE-Anweisung oder CLASS-Anweisung definierten Datentypen für Strukturen kann dies noch eines der folgenden Worte sein: INTEGER - kurze Integerzahl LONG - lange Integerzahl SINGLE - kurze Gleitpunktzahl DOUBLE - lange Gleitpunktzahl STRING - String variabler Länge STRING*n - String fester Länge (n) Bei durch TYPE oder CLASS definierten Datentypen kann vor dem Typ auch eine der Klauseln POINTER TO oder PTR TO angegeben werden. Dies bewirkt, daß nicht eine Struktur, sondern nur ein Zeiger auf die Struktur definiert wird.

Anmerkungen

Die in einer COMMON-Zone vergebenen Variablennamen dürfen sich nicht mit anderen Variablennamen im Programm überschneiden.

Arrays in einer COMMON-Zone können von mehreren Modulen aus dimensioniert werden. Ist das Array bereits dimensioniert, erfolgt beim Programmablauf nur eine Überprüfung auf Übereinstimmung der Dimensionierung. Das Array bleibt dabei unverändert.

Die Variablen innerhalb eines COMMON-Blocks werden allein durch ihre Position und nicht durch ihren Namen adressiert. Daher ist es sehr wichtig, daß die COMMON-Anweisung in allen Modulen, die den Block verwenden, identisch formuliert ist. Ist dies nicht der Fall, führt dies zu sehr schwer lokalisierbaren Fehlern. Beispiele:

Programm 1:

```
COMMON /TEST/ A, B, C
A = 1 : B = 2 : C = 3 PRINT A, B, C
```

Programmausgabe:

```
    1    2    3
```

Programm 2:

```
COMMON /TEST/ A, C, B PRINT A, B, C
```

Programmausgabe:

```
    1    3    2
```

CONST

Deklaration von Konstanten.

Verwendung

```
CONST Name = Ausdruck [, Name = Ausdruck, ...]
```

Beschreibung

Diese Anweisung weist einem Zahlenwert einen symbolischen Namen zu. Im Gegensatz zu Variablen hängt der Typ der Konstanten, falls er nicht explizit per Typkennung angegeben wird, von der Art des Ausdrucks ab. Es ist also nicht möglich, zwei Konstanten unterschiedlichen Typs, aber mit dem gleichen Namen zu definieren, wie es bei Variablen üblich ist. Wird eine Typkennung angegeben, kann sie daher später fortgelassen werden. Die DEFxxx-Anweisungen beeinflussen den Typ der Konstanten nicht.

Eine Konstante kann nur am Anfang des Haupt- bzw. Unterprogramms deklariert werden. Im Hauptprogramm definierte Konstanten sind für Haupt- und Unterprogramme global, während in einem Unterprogramm definierte Konstante lokal sind.

Der Name einer Konstanten kann nicht mehr für Variable oder Unterprogramme verwendet werden.

Parameter

`Name`	Der Name der Konstanten.
`Ausdruck`	Der Ausdruck, dessen Wert dem Namen zugewiesen werden soll. Der Ausdruck darf keine Variable enthalten, da er in einen konstanten Wert umgerechnet werden muß.

Beispiel

```
CONST FALSE = 0, TRUE = NOT FALSE, MAXIMUM% = 1000
```

CONSTRUCTOR

Automatische Initialisierung einer Instanz einer Klasse.

Verwendung

```
CLASS Name
      ....
      SUB CONSTRUCTOR ([Parameter])
      ....
END CLASS
```

Beschreibung

Der reservierte Unterprogrammname `CONSTRUCTOR` wird zur Definition eines Unterprogramms verwendet, das automatisch bei der Einrichtung einer Variable der Klasse aufgerufen wird. Es hat zur Aufgabe, die Instanz gezielt zu initialisieren. Weiter müssen, falls die Basisklasse oder in der Klasse selbst definierte Klassen einen Konstruktor mit Parametern besitzen, diese Konstruktoren aufgerufen werden.

Der Konstruktor der Basisklasse muß dann explizit aufgerufen werden, wenn er mit Parametern versorgt werden muß. In diesem Fall ist der Aufruf in der Form

```
CALL BASE.CONSTRUCTOR (Parameter)
```

als erste Anweisung innerhalb des Konstruktors anzugeben. Hat der Konstruktor der Basisklasse keine Parameter, muß er nicht explizit angegeben werden. Die Konstruktoren von Elementen der Klasse müssen ebenfalls explizit in der Form

```
CALL Elementname.CONSTRUCTOR (Parameter)
```

angegeben werden, wenn die Konstruktoren Parameter besitzen oder eine bestimmte Reihenfolge der Aufrufe gewünscht wird. Auch hier braucht ein Konstruktor nicht angegeben werden, wenn er keine Parameter besitzt. Alle Konstruktor-Aufrufe müssen vor dem eigentlichen Code des Konstruktors angegeben werden.

Beispiel

```
CLASS Basis
PUBLIC SUB CONSTRUCTOR (I%)
END CLASS

CLASS Eingebettet
PUBLIC SUB CONSTRUCTOR (A$)
END CLASS

CLASS Abgeleitet
      Instanz AS Eingebettet
PUBLIC SUB CONSTRUCTOR ()
END CLASS

SUB Abgeleitet.CONSTRUCTOR()
      CALL BASE.CONSTRUCTOR (55)
      CALL Instanz.CONSTRUCTOR ("Text")
END SUB
```

Querverweise

DESTRUCTOR, CLASS

COS()

Berechnen des Cosinus.

Verwendung

```
N = COS (Ausdruck)
```

Parameter

`Ausdruck`	Der numerische Ausdruck, dessen Cosinus ermittelt werden soll.

Returnwert

Der Cosinus des Ausdrucks.

Anmerkungen

Der Wert wird immer als lange Gleitpunktzahl berechnet, wenn der Parameter ebenfalls eine lange Gleitpunktzahl ist; sonst ist das Ergebnis einfach genau.

Querverweise

ATN(), SIN(), TAN()

CSNG()

Umwandeln eines Wertes in eine kurze Gleitpunktzahl.

Verwendung

```
N! = CSNG (Ausdruck)
```

Parameter

`Ausdruck`	Der Ausdruck wird ausgerechnet und in eine kurze Gleitpunktzahl umgerechnet.

Returnwert

Das Ergebnis des Ausdrucks als kurze Gleitpunktzahl.

Anmerkungen

Diese Funktion kann zur expliziten Typkonversion von Variablen und Teilausdrücken verwendet werden.

Querverweise

`CDBL()`, `CINT()`, `CLNG()`

CURRENCY$()

Abfrage des aktuellen Währungssymbols.

Verwendung

Diese Funktion liefert einen String zurück, der das Währungssymbol des aktuell definierten Landes enthält. Sie kann zum Beispiel in PRINT-Anweisungen eingesetzt werden.

Returnwert

Das Währungssymbol als String.

Beispiel

```
A = 123.45
PRINT "**######.## &" A, CURRENCY$
```

ergibt:

```
****123,45 DM
```

DATA

Definition von Datenzeilen.

Verwendung

```
DATA Konstante [, Konstante, ...]
```

Beschreibung

Diese Anweisung definiert eine Liste von Konstanten, die mit der READ-Anweisung eingelesen werden können. Diese Daten werden auf Modulebene abgelegt, wobei alle `DATA`-Zeilen zu einem Datenblock zusammengefaßt werden.

Parameter

`Konstante`	Die Liste der Konstanten, die abgelegt werden soll. Die einzelnen Elemente der Liste werden voneinander durch Kommata getrennt. Strings müssen nur in Anführungszeichen eingeschlossen werden, wenn sie Leerzeichen, Kommata oder Semikolons enthalten.

Anmerkungen

Da `DATA`-Zeilen Code erzeugen, sollten sie am Programmanfang bzw. am Programmende zusammengefaßt werden. `DATA`-Zeilen in einer Schleife können diese durchaus verlangsamen!

Querverweise

READ, RESTORE

DATE$()

Ermitteln des Tagesdatums.

Verwendung

```
S$ = DATE$
```

Returnwerte

Das Tagesdatum in der Form "MM-TT-JJJJ", wobei "MM" den Monat im Bereich zwischen 1 bis 12, "TT" den Tag im Bereich zwischen 1 bis 31 und "JJJJ" das Jahr im Bereich zwischen 1980 und 2099 bezeichnet.

Anmerkungen

Mit der Funktion TIME$() kann die Uhrzeit ermittelt werden.

Das Datum kann nicht geändert werden.

Querverweise

TIME$

DECLARE

Prototyp-Deklaration eines Unterprogramms.

Verwendung

Syntax für StarBASIC-Unterprogramme:

```
DECLARE FUNCTION|SUB Name ([Parameter]) [AS Datentypname]
```

Syntax für externe Unterprogramme:

```
DECLARE FUNCTION|SUB Name [CDECL|PASCAL] [LIB "Name"] [LINK nnnn]
([Parameter]) [AS Datentypname]
```

Beschreibung

Mit dieser Anweisung wird ein Unterprogramm vor der eigentlichen Codierung deklariert. Durch diese Deklaration werden bei jeder Verwendung des Unterprogrammnamens Anzahl und Typ der Parameter überprüft. Sämtliche in einem Modul verwendeten Unterprogramme müssen vor der Verwendung per `DECLARE`-Anweisung deklariert werden.

Parameter

`FUNCTION | SUB`	Ein Unterprogramm wird entweder als Funktion oder als Unterprogramm deklariert.
`Programmname[Datentyp]`	Dies ist der Name des Unterprogramms. Optional kann bei Funktionen der Typ der Funktion als Suffixzeichen (s.o.) folgen. Werden sowohl ein Suffixzeichen als auch die AS-Klausel verwendet, müssen beide Datentypen übereinstimmen. Bei Methodendefinitionen setzt sich der Name aus dem Klassennamen, gefolgt von einem Punkt, sowie dem Namen der Methode zusammen.
`CDECL | PASCAL`	Angabe der Aufrufart. Erfolgt keine Angabe, wird angenommen, daß es sich um ein Unterprogramm nach Pascal-Konventionen handelt, wenn in der Parameterliste eines der Worte `BYVAL` oder `BASICSTRING` verwendet wird, wenn es sich um eine Funktion des Typs `BASICSTRING` handelt oder wenn die LIB-Angabe verwendet wird.
`LIB "Libraryname"`	Optional kann die Angabe der Library erfolgen, in der die Routine zu suchen ist. Unter Windows kann dies der Name einer DLL sein. In diesem Fall wird versucht, die Routine innerhalb der angegebenen DLL zu finden.
`LINK nnnn`	Bei Angabe dieser Klausel wird eine dynamische Verbindung zu einer anderen Applikation aufgebaut. Der angegebene Library-Name bezeichnet dabei den Namen der Applikation.
`(Parameter)`	Die Liste der Parameter ist optional. Wird die Routine ohne Parameter aufgerufen, muß ein leeres Klammerpaar `()` angegeben werden.
`AS Datentypname`	Der Typ einer Funktion kann auch durch eine AS-Klausel angegeben werden. Wird die AS-Klausel verwendet, muß der Datentyp-Suffix, falls angegeben, genau dem angegebenen Typ entsprechen.

Die Definition eines Parameters ist wie folgt:

```
[OPTIONAL] [BYVAL] Parametername [Datentyp | AS Datentypname]
```

`OPTIONAL`	Dieses Wort deklariert alle folgenden Parameter als optional. Werden optionale Parameter nicht angegeben, werden diese beim Funktionsaufruf mit 0 bzw. dem NULL-Pointer initialisiert.
`BYVAL`	Wird das Wort `BYVAL` verwendet, wird der Parameter *by value* übergeben. Es wird also kein Pointer auf den Wert, sondern der Wert selbst übergeben.
`Parametername`	Dies ist der Name des Parameters. Er ist frei vergebbar.
`Datentyp`	Der Datentyp eines Parameters kann wie in der DIM-Anweisung entweder durch den Datentyp-Suffix oder durch die AS-Klausel definiert werden.
`AS Datentypname`	Wird kein Datentyp-Suffix verwendet, kann der Datentyp mit der AS-Klausel definiert werden. Der Datentypname ist einer der Namen INTEGER, LONG, SINGLE, DOUBLE, STRING, BASICSTRING oder ANY oder auch der Name einer mit der TYPE-Anweisung definierten Datenstruktur. Der Datentyp ANY sagt aus, daß keine Typüberprüfung stattfindet. Es ist die Aufgabe des Unterprogramms, herauszufinden, um welchen Datentyp es sich handelt.

Der Datentyp ist einer der folgenden Suffixe:

%	INTEGER
&	LONG
!	SINGLE
#	DOUBLE
$	STRING

Der Datentypname ist einer der Namen INTEGER, LONG, SINGLE, DOUBLE, STRING oder BASICSTRING. Namen von mit TYPE definierten Datenstrukturen können nur als Parameter, jedoch nicht als Returnwert verwendet werden.

Beispiele

```
DECLARE FUNCTION XXX (A%, B AS INTEGER)
```

Arrays werden dadurch kenntlich gemacht, daß dem Namen der Variablen ein Klammernpaar () nachgestellt wird:

```
DECLARE FUNCTION XXX (A() AS INTEGER)
```

Optional kann auch die Anzahl der Dimension angegeben werden:

```
DIM TEXT$ (10, 100)
DECLARE SUB PROZEDUR (A(2) AS STRING)
```

DEF FN

Definition einer DEF FN-Funktion.

Verwendung

```
DEF FNname [(Parameter)] = arithmetischer Ausdruck

DEF FNname [(Parameter)] [STATIC | LOCAL]
    ....
    FNname = arithmetischer Ausdruck
    ....
END DEF
```

Beschreibung

Die Definition einer DEF FN-Funktion wird mit dem Anweisungspaar DEF FN / END DEF durchgeführt. Nach der DEF FN-Anweisung folgt der Rumpf der Funktion. Dieser Rumpf sollte eine Wertzuweisung auf den Funktionsnamen enthalten. Der dort zugewiesene Wert wird als Funktionsergebnis zurückgeliefert. Wird keine Wertzuweisung durchgeführt, ist der Funktionswert 0 bzw. ein Leerstring bei String-Funktionen. Der Funktionsname muß immer mit den Buchstaben "FN" beginnen.

Im Allgemeinen wird der Rumpf einer Funktion sequentiell abgearbeitet. Soll eine Funktion vorzeitig verlassen werden, kann die EXIT DEF-Anweisung verwendet werden.

Die einzeilige Version der Anweisung ist historisch bedingt. Sie ist der mehrzeiligen Version ebenbürtig, kann jedoch nur eine Anweisung enthalten.

Parameter

Name	Funktionsname. Der Name folgt den Regeln für Variable, d.h. er kann von einem der Typkennungsbuchstaben %, !, &, # oder $ gefolgt werden, um den Typ des Funktionsergebnisses kenntlich zu machen. Globale Funktionen sollten immer mit einem Typkennungsbuchstaben versehen werden, damit der Typ des Funktionsergebnisses immer korrekt ist.

(Parameter)	Die optionale Liste der formalen Parameter. Der Typ der formalen Parameter kann entweder über die Typkennung oder über die AS-Anweisung deklariert werden:
	
	`FUNCTION XXX (A%, B AS INTEGER)`
	
	Arrays werden dadurch kenntlich gemacht, daß dem Namen der Variablen ein Klammernpaar () nachgestellt wird:
	
	`FUNCTION XXX (A() AS INTEGER)`

STATIC	Der optionale Zusatz STATIC deklariert sämtliche lokalen, innerhalb des Unterprogramms deklarierten Variablen als statisch.
LOCAL	Der optionale Zusatz LOCAL deklariert die Funktion als lokal. Eine lokale Funktion ist für andere Module unsichtbar.

Anmerkungen

Die DEF FN-Anweisung ist ein historisches Relikt. Daher gelten einige Besonderheiten:

1. Die Parameter werden *by value* übergeben. Die formalen Parameter gelten als lokale Variable, die vor dem Funktionsaufruf mit den Werten der aktuellen Parameter besetzt werden. Eine Änderung des Inhalts eines formalen Parameters hat keine Auswirkungen auf den Inhalt des aktuellen Parameters.

2. Im Gegensatz zu anderen BASIC-Dialekten verhält sich eine `DEF FN`-Funktion wie eine normale Funktion. Sie ist rekursiv sein; sie kann lokale Variable enthalten etc.

DEFINT, DEFLNG, DEFSNG, DEFDBL, DEFSTR

Deklaration von impliziten Variablentypen.

Verwendung

```
DEFxxx Buchstabe | Buchstabe-Buchstabe [, ...]
```

Beschreibung

Mit dieser Anweisung wird eine implizite Bindung von Variablennamen an einen Datentyp vorgenommen. Fällt der Anfangsbuchstabe des Variablennamens in einen der angegebenen Buchstabenbereiche, erhält die Variable automatisch den damit verbundenen Datentyp, ohne daß eine Typkennung angefügt werden muß. Wird jedoch eine Typkennung angegeben, hat diese Vorrang vor dem Anfangsbuchstaben. Überschneiden sich die Buchstabenbereiche von mehreren `DEFxxx`-Anweisungen, gilt die zuletzt angegebene Anweisung.

Parameter

`Buchstabe`	Ein Buchstabe bzw. ein Buchstabenbereich.

Anmerkungen

Mehrere Variablen können den gleichen Namen, jedoch unterschiedliche Typen haben. So kennzeichnen die Variablen `I%, I&, I!, I#` und `I$` unterschiedliche Variable. Diese Regel gilt auch für Arrays daher sind die Variablen `A$ (5)` und `A$` auch zwei verschiedene Variable.

Der Typ von Elementen von Strukturen wird von diesen Anweisungen nicht beeinflußt.

Beispiel

```
DEFINT A-E,I-N
DEFSNG X

ITALIEN = 5                  ' kurze Integerzahl
X3 = 3                       ' kurze Gleitpunktzahl
I# = 68                      ' lange Gleitpunktzahl!
```

DESTRUCTOR

Automatische Initialisierung einer Instanz einer Klasse.

Verwendung

```
CLASS Name
      ....
      SUB DESTRUCTOR ()
      ....
END CLASS
```

Beschreibung

Der reservierte Unterprogrammname `DESTRUCTOR` wird zur Definition eines Unterprogramms verwendet, das automatisch bei der Löschung einer Variable der Klasse aufgerufen wird. Es hat zur Aufgabe, die Instanz gezielt zu deinitialisieren. Ein Destruktor kann keine Parameter haben.

Im Gegensatz zu Konstruktoren werden die Destruktoren von eingebetteten Instanzen sowie der Destruktor der Basisklasse automatisch aufgerufen. Nach dem Code des Destruktors werden die Destruktoren von eingebetteten Instanzen in der umgekehrten Reihenfolge ihrer Deklaration aufgerufen. Anschließend wird der Destruktor der Basisklasse aufgerufen.

Im Allgemeinen werden Destruktoren in der umgekehrten Reihenfolge von Konstruktoren aufgerufen. Dies gilt insbesondere bei Arrays. Destruktoren von globalen Variablen werden in der umgekehrten Reihenfolge ihrer Deklaration im Quellprogramm aufgerufen. Werden die Variablen nicht in dieser Reihenfolge eingerichtet, werden die Destruktoren demnach nicht in der umgekehrten Reihenfolge wie die Konstruktoren aufgerufen. Das folgende Beispiel zeigt dies auf:

```
CLASS Klasse
      BEZ AS STRING*4
PUBLIC
      SUB CONSTRUCTOR (A$)
      SUB DESTRUCTOR()
END CLASS

SUB Klasse.CONSTRUCTOR (A$)
      BEZ = A$
      PRINT "Konstruktor " BEZ
END SUB

SUB Klasse.DESTRUCTOR ()
      PRINT "Destruktor " BEZ
END SUB

10    GOTO 30
20    DIM A AS Klasse ("A")
      GOTO 999
30    DIM B AS Klasse ("B")
      GOTO 20
999   END
```

ergibt:

```
Konstruktor B
Konstruktor A
```

```
Destruktor B
Destruktor A
```

Querverweise

CONSTRUCTOR, CLASS, THIS

DIM

Deklaration von Variablen.

Verwendung

```
DIM [SHARED] Variable [AS Typ] [, Variable [AS Typ], ...]
```

Beschreibung

Mit Hilfe dieser Anweisung können skalare Variable, Arrays und Strukturen deklariert werden. Die Deklaration von skalaren Variablen ist nicht unbedingt notwendig, da sie auch implizit erfolgen kann das gleiche gilt für eindimensionale Arrays mit bis zu 10 Elementen. Größere Arrays und Strukturen müssen hingegen mit dieser Anweisung deklariert werden. Soll eine im Hauptprogramm deklarierte Variable auch für Unterprogramme zugänglich sein, muß sie ebenfalls mit dieser Anweisung deklariert werden. Die `DIM`-Anweisung muß am Anfang eines Programms bzw. Unterprogramms stehen.

Parameter

`SHARED` — Dieses optionale Wort kennzeichnet die deklarierten Variablen als für Unterprogramme zugänglich.

`Variable` — Name der zu deklarierenden Variablen. Die Typkennung kann entfallen, wenn der Typ entweder durch eine DEFxxx-Anweisung oder durch die AS-Klausel vorgegeben ist. Arrays erwarten hinter dem Namen noch die Beschreibung der Dimensionen (s.u.).

`AS Typ` — Deklaration des Variablentyps. Neben den durch die TYPE- oder CLASS-Anweisung definierten Datentypen für Strukturen kann dies noch eines der folgenden Worte sein:

INTEGER	- kurze Integerzahl
LONG	- lange Integerzahl
SINGLE	- kurze Gleitpunktzahl
DOUBLE	- lange Gleitpunktzahl
STRING	- String variabler Länge
STRING*n	- String fester Länge (n)

Bei durch TYPE oder CLASS definierten Datentypen kann vor dem Typ auch eine der Klauseln POINTER TO oder PTR TO angegeben werden. Dies bewirkt, daß nicht eine Struktur, sondern nur ein Zeiger auf die Struktur definiert wird.

Anmerkungen

Soll ein Array deklariert werden, müssen Anzahl und Größe der Dimensionen wie folgt angegeben werden:

```
Name ( Anzahl | Untergrenze TO Obergrenze [, ...] )
```

Jede Dimension kann entweder durch einen Ausdruck oder durch einen Bereich beschrieben werden. Wird nur eine Zahl angegeben, erstreckt sich der Bereich von der durch die OPTION BASE-Anweisung eingestellten Untergrenze (0 oder 1, die Vorgabe ist 0) bis zur angegebenen Obergrenze. Beispiele:

```
OPTION BASE 1 : DIM A (5, 10) ' 5x10 Elemente
```

```
OPTION BASE 0 : DIM A (5, 10) ' 6x11 Elemente
DIM B (-3 TO 3)               ' Bereichsangabe
DIM C (5, A TO B, 3)          ' Mischangaben sind möglich
```

Querverweise

OPTION BASE, REDIM, ERASE

DO...LOOP

Führt einen Anweisungsblock so lange aus, bis eine Bedingung FALSCH ist.

Verwendung

Möglichkeit 1:

```
DO
    ....
    ....
LOOP
```

Möglichkeit 2:

```
DO
    ....
    ....
LOOP [WHILE | UNTIL] Ausdruck
```

Möglichkeit 3:

```
DO [WHILE | UNTIL] Ausdruck
    ....
    ....
LOOP
```

Beschreibung

Die erste Schreibweise definiert eine Endlosschleife. Die Anweisungen werden endlos wiederholt. Die anderen beiden Schreibweisen definieren eine Überprüfung nach (Schreibweise 2) oder vor (Schreibweise 3) jedem Schleifendurchlauf, wobei der nächste Durchlauf entweder erfolgt, wenn der Ausdruck einen Wert gleich Null (`WHILE`) oder ungleich Null (`UNTIL`) hat.

Die Schleife kann jederzeit mit der EXIT DO-Anweisung verlassen werden.

Die vereinfachten Darstellungen wären:

Schreibweise 1:

```
Schleife:
    ....
    ....
    GOTO Schleife
```

Schreibweise 2 mit `WHILE`:

```
Schleife:
    ....
    ....
    IF Ausdruck <> 0 THEN GOTO Schleife
```

Schreibweise 2 mit `UNTIL`:

```
Schleife:
    ....
    ....
    IF Ausdruck = 0 THEN GOTO Schleife
```

Schreibweise 3 mit `WHILE`:

```
Schleife:
    IF Ausdruck = 0 THEN GOTO Weiter
    ....
    ....
    GOTO Schleife
Weiter:
    ....
```

Schreibweise 3 mit `UNTIL`:

```
Schleife:
    IF Ausdruck <> 0 THEN GOTO Weiter
    ....
    ....
    GOTO Schleife
Weiter:
    ....
```

Parameter

`Ausdruck`	Der Ausdruck, der zwecks Abbruch der Schleife überprüft wird. Ist er entweder Null (bei `WHILE`) oder nicht Null (bei `UNTIL`), wird die Schleife verlassen und das Programm an der der `LOOP`- Anweisung folgenden Anweisung fortgesetzt.

Beispiel

Die folgenden beiden Beispiele entfernen führende Leerstellen aus einem String. Das erste Beispiel arbeitet mit einer primitiven Endlosschleife, während das zweite Beispiel eine Abbruchbedingung formuliert.

Beispiel 1:

```
A$ = "   Hallo"

DO
    IF LEFT$ (A$, 1) = " " THEN EXIT DO
    A$ = MID$ (A$, 2)
LOOP
```

Beispiel 2:

```
A$ = "   Hallo"

DO UNTIL LEFT$ (A$, 1) = "  "
    A$ = MID$ (A$, 1)
LOOP
```

END

Abschluß eines Blocks oder eines Programms.

Verwendung

```
END [DEF | FUNCTION | IF | SUB | TYPE]
```

Beschreibung

Die `END`-Anweisung schließt einen Block oder ein Programm ab.

Parameter

`END`	Programmende. Die Verwendung ohne Parameter beendet das laufende Programm.
`END` DEF	Ende einer DEF FN-Funktion.
`END` FUNCTION	Ende einer Funktion.
`END` IF	Ende eines `IF/THEN/ELSE`-Blocks.
`END` SUB	Ende einer Prozedur.
`END` TYPE	Ende einer Struktur-Deklaration.

Querverweise

SUB, FUNCTION, DEF FN, TYPE, IF...THEN...ELSE

ENDPRINT

Abschluß eines Druckvorgangs.

Verwendung

```
ENDPRINT
```

Beschreibung

Diese Anweisung veranlaßt bei Betriebssystemen mit Spooling das System dazu, einen Druckauftrag abzuschließen und die Datei dem Spooler zuzuführen.

EOF()

Abfrage, ob das Dateiende erreicht ist.

Verwendung

```
N = EOF (Kanalnummer)
```

Beschreibung

Diese Funktion überprüft, ob bei der der Kanalnummer zugeordneten Datei das Dateiende erreicht ist.

Parameter

`Kanalnummer`	Die Kanalnummer der anzusprechenden Datei. Das sonst optionale Doppelkreuz darf hier nicht angegeben werden.

Returnwerte

Die Funktion liefert TRUE zurück, wenn das Dateiende erreicht ist, sonst FALSE.

Beispiel

Das folgende Programm gibt eine Datei auf dem Bildschirm aus:

```
INPUT "Dateiname:", NAME$
OPEN "R", 1, NAME$
WHILE NOT EOF (1)
    LINE INPUT #1, ZEILE$
    PRINT ZEILE$
WEND
END
```

ERASE

Löschen von Arrays und Pointers.

Verwendung

```
ERASE Array|Pointer [, Array|Pointer, ...]
```

Beschreibung

Die angegebenen Arrays bzw. Pointers werden gelöscht und der ihnen zugewiesene Speicherplatz wird freigegeben.

Die Pointer-Variable zeigt auf einen mit ALLOCATE angeforderten Speicherbereich. Dieser wird wieder freigegeben. Nach erfolgreicher Freigabe wird der Pointer mit Null initialisiert. Ist der Pointer ungültig, weil er nicht auf einen mit ALLOCATE angeforderten Speicherbereich zeigt, wird ein Laufzeitfehler 101 (`Ungültiger Pointer`) erzeugt.

Anmerkungen

Nach dem Ende des Programmlaufs werden alle durch ALLOCATE angeforderten Speicherbereiche ebenso wie alle Arrays wieder freigegeben.

Parameter

`Array`	Namen der zu löschenden Arrays.
`Pointer`	Namen der zu löschenden Pointers.

Querverweise

DIM, REDIM, COMMON

ERL()

Ermitteln der Zeile, in der ein Laufzeitfehler auftrat.

Verwendung

```
N = ERL
```

Returnwerte

Da die Zeilennummern im Quelltext nicht in das Programm übernommen werden, wird immer der Wert 0 zurückgeliefert.

Anmerkungen

Die Funktion ERR() liefert den Fehlercode eines Laufzeitfehlers zurück.

Querverweise

ERROR, ERR(), ON ERROR GOTO, RESUME

ERR()

Ermitteln des zuletzt aufgetretenen Laufzeitfehlers.

Verwendung

```
N = ERR
```

Returnwerte

Der Code des zuletzt aufgetretenen Laufzeitfehlers.

Anmerkungen

Diese Funktion wird meist in einer Fehlerbehandlungsroutine eingesetzt, die durch die ON ERROR-Anweisung aktiviert wurde.

Die Funktion ERL() liefert normalerweise die Nummer der Zeile zurück, in der der Fehler auftrat. Sie ist als Leerfunktion mit dem Returnwert 0 implementiert.

Querverweise

ERROR, ERL(), ON ERROR GOTO, RESUME

ERROR

Gezieltes Erzeugen eines Laufzeitfehlers.

Verwendung

```
ERROR Ausdruck
```

Beschreibung

Mit Hilfe dieser Anweisung wird gezielt ein Laufzeitfehler erzeugt. Unmittelbar nach Abarbeitung des `ERROR`-Anweisung wird die Fehlerbehandlung aktiviert. Wurde keine ON ERROR-Anweisung durchlaufen, wird der Fehlercode der Applikation mitgeteilt, die darauf individuell verschieden reagieren kann. Ist jedoch eine ON ERROR-Anweisung aktiv, wird zu dem in der ON ERROR-Anweisung angegebenen Label verzweigt.

Parameter

`Ausdruck`	Ein arithmetischer Ausdruck, dessen Integer-Wert als Fehlercode verwendet wird. Es können beliebige Fehlercodes erzeugt werden.

Beispiel

In folgendem Beispiel würde der Text "Feierabend!" ausgegeben und das Programm beendet werden:

```
        ON ERROR GOTO Fehler
        ....
        ERROR 999
        ....

    Fehler:
        IF ERR = 999 THEN PRINT "Feierabend!"
        END
```

Querverweise

ERR(), ERL(), ON ERROR GOTO, RESUME

EXIT

Vorzeitiges Verlassen eines Blocks oder eines Unterprogramms.

Verwendung

```
EXIT [DEF | DO | FOR | FUNCTION | SUB]
```

Beschreibung

Die EXIT-Anweisung bricht die Abarbeitung eines Blocks oder eines Unterprogramms ab.

Parameter

EXIT DEF	Verlassen einer DEF FN-Funktion. Das Programm wird hinter dem Aufruf der Funktion fortgesetzt. Der arithmetische Ausdruck, dessen Teil der Funktionsaufruf war, wird zu Ende berechnet.
EXIT DO	Verlassen eines DO/LOOP-Blocks. Das Programm wird hinter der LOOP-Anweisung fortgesetzt.
EXIT FOR	Verlassen einer FOR/NEXT-Schleife. Das Programm wird hinter der NEXT-Anweisung fortgesetzt.
EXIT FUNCTION	Verlassen einer Funktion. Das Programm wird hinter dem Aufruf der Funktion fortgesetzt. Der arithmetische Ausdruck, dessen Teil der Funktionsaufruf war, wird zu Ende berechnet.
EXIT SUB	Verlassen einer Prozedur. Das Programm wird hinter dem Prozeduraufruf fortgesetzt.

Querverweise

FOR...NEXT, DO...LOOP, SUB, FUNCTION, DEF FN

EXP()

Berechnen der *e*-Funktion.

Verwendung

```
N = EXP (Ausdruck)
```

Parameter

`Ausdruck`	Der numerische Ausdruck, dessen *e*-Funktion ermittelt werden soll.

Returnwert

Die *e*-Funktion des Ausdrucks.

Anmerkungen

Ist der Parameter größer als 88.02969, wird der Laufzeitfehler 6 (`Arithmetischer Überlauf`) erzeugt. Der Wert wird immer als lange Gleitpunktzahl berechnet, wenn der Parameter ebenfalls eine lange Gleitpunktzahl ist sonst ist das Ergebnis einfach genau.

Querverweise

`LOG()`

FIX()

Ermitteln des ganzzahligen Anteils einer Zahl.

Verwendung

```
N = FIX (Ausdruck)
```

Parameter

`Ausdruck`	Der numerische Ausdruck, der ausgerechnet und dessen ganzzahliger Anteil ermittelt werden soll.

Returnwerte

Der ganzzahlige Anteil des Ausdrucks.

Anmerkungen

Der Unterschied zur Funktion INT() liegt bei den negativen Zahlen. Diese Funktion liefert bei einer negativen Zahl die erste Ganzzahl größer als den Parameter zurück, während INT() die erste Ganzzahl kleiner als den Parameter zurückliefert.

Beispiel

```
X = -12.34
Y = INT (X)          ' -13
Z = FIX (X)          ' -12
```

Querverweise

INT()

FOR...NEXT

Aufbau einer Programmschleife mit Schleifenvariable.

Verwendung

```
FOR Variable = Startausdruck TO Endausdruck [STEP Inkrement]
....
NEXT [Variable [, Variable, ...]]
```

Beschreibung

Die angegebene Variable wird mit dem Wert des Startausdrucks besetzt. Anschließend werden die Anweisungen zwischen der FOR-Anweisung und der NEXT-Anweisungen durchlaufen. Das Inkrement wird auf die Variable aufaddiert. Ist der neue Wert der Variablen kleiner als der Endausdruck, werden die Anweisungen zwischen FOR und NEXT erneut durchlaufen. Das Inkrement wird erneut addiert. Das Ganze wird so oft wiederholt, bis die Variable einen den Wert des Endausdrucks erreicht oder überschritten hat.

Die FOR-Schleife kann wie folgt nachgebildet werden:

```
    VARIABLE = STARTAUSDRUCK
Schleife:
    ....
    VARIABLE = VARIABLE + INKREMENT
    IF VARIABLE < ENDAUSDRUCK THEN GOTO Start
```

Parameter FOR

Variable	Schleifenvariable. Die Variable muß eine numerische Variable sein.
Startausdruck	Ein arithmetischer Ausdruck, der beliebig formuliert sein kann. Dieser Ausdruck wird der Variablen vor Beginn der Schleife zugewiesen.
Endausdruck	Ein arithmetischer Ausdruck, mit dem der Wert der Variablen verglichen wird. Ist der Wert der Variablen kleiner (bei negativem Inkrement: größer) als der Endausdruck, wird die Schleife erneut durchlaufen.
Inkrement	Das Inkrement wird nach jedem Durchlauf der Schleife auf die Schleifenvariable aufaddiert. Ist kein Inkrement angegeben, ist das Inkrement 1.

Parameter NEXT

Variable	Die Schleifenvariable kann bei der NEXT-Anweisung mit angegeben werden. Bei geschachtelten Schleifen können alle Schleifenvariablen bei einer NEXT-Anweisung angegeben werden (siehe Beispiel).

Anmerkungen

Die FOR-Anweisung kann mehrfach geschachtelt werden. Für jede FOR-Anweisung ist eine entsprechende NEXT-Anweisung anzugeben, die das Ende der jeweiligen FOR-Schleife markiert. Die Variablen der NEXT-Anweisungen müssen in der umgekehrten Reihenfolge wie die FOR-Anweisungen angegeben werden.

Eine FOR-Schleife kann vorzeitig mit der EXIT FOR-Anweisung verlassen werden.

Beispiele

Eine dreifach geschachtelte FOR-Anweisung:

```
N = 0

FOR I = 1 TO 10
    FOR J = 1 TO 10
        FOR K = 1 TO 10
            N = N + 1
        NEXT
    NEXT
NEXT

PRINT "Anzahl Durchläufe: " N
```

Die innere Schleife würde 10*10*10 = 1000mal durchlaufen werden. Für die NEXT-Anweisung gibt es noch zwei mögliche Schreibweisen:

```
N = 0

FOR I = 1 TO 10
    FOR J = 1 TO 10
        FOR K = 1 TO 10
            N = N + 1
        NEXT K
    NEXT J
NEXT I

PRINT "Anzahl Durchläufe: " N
```

oder auch:

```
N = 0

FOR I = 1 TO 10
    FOR J = 1 TO 10
        FOR K = 1 TO 10
            N = N + 1
NEXT K, J, I

PRINT "Anzahl Durchläufe: " N
```

Querverweise

EXIT FOR

FREEFILE()

Ermitteln der nächsten freien Kanalnummer.

Verwendung

```
N = FREEFILE
```

Beschreibung

Die nächste freie Kanalnummer wird zurückgeliefert.

Anmerkungen

Diese Funktion kann innerhalb von Unterprogrammen verwendet werden, um jederzeit eine eindeutige Kanalnummer zur Verfügung zu haben.

FUNCTION

Definition einer Funktion.

Verwendung

```
FUNCTION Name [(Parameter)] [STATIC | LOCAL]
    ....
    Name = arithmetischer Ausdruck
    ....
END FUNCTION
```

Beschreibung

Die Definition einer Funktion wird mit dem Anweisungspaar `FUNCTION / END FUNCTION` durchgeführt. Nach der `FUNCTION`-Anweisung folgt der Rumpf der Funktion. Dieser Rumpf sollte eine Wertzuweisung auf den Funktionsnamen Dateien. Der dort zugewiesene Wert wird als Funktionsergebnis zurückgeliefert. Wird keine Wertzuweisung durchgeführt, ist der Funktionswert 0 bzw. ein Leerstring bei String-Funktionen.

Im Allgemeinen wird der Rumpf einer Funktion sequentiell abgearbeitet. Soll eine Funktion vorzeitig verlassen werden, kann die EXIT FUNCTION-Anweisung verwendet werden.

Parameter

`Name` — Funktionsname. Der Name folgt den Regeln für Variable, d.h. er kann von einem der Typkennungsbuchstaben %, !, &, # oder $ gefolgt werden, um den Typ des Funktionsergebnisses kenntlich zu machen. Globale Funktionen sollten immer mit einem Typkennungsbuchstaben versehen werden, damit der Typ des Funktionsergebnisses immer korrekt ist. Bei Methodendefinitionen setzt sich der Name aus dem Klassennamen, gefolgt von einem Punkt, sowie dem Namen der Methode zusammen.

`(Parameter)` — Die optionale Liste der formalen Parameter. Der Typ der formalen Parameter kann entweder über die Typkennung oder über die AS-Anweisung deklariert werden:

```
FUNCTION XXX (A%, B AS INTEGER)
```

Arrays werden dadurch kenntlich gemacht, daß dem Namen der Variablen ein Klammernpaar () nachgestellt wird:

```
FUNCTION XXX (A() AS INTEGER)
```

`STATIC` — Der optionale Zusatz `STATIC` bedeutet, daß alle lokale Variable beim Programmstart angelegt werden und ihren Wert zwischen mehreren Aufrufen des Unterprogramms beibehatlten.

`LOCAL` — Der optionale Zusatz `LOCAL` deklariert die Funktion als lokal. Eine lokale Funktion ist für andere Module unsichtbar.

Anmerkungen

Funktionen sind rekursiv, d.h. eine Funktion kann sich selbst entweder direkt oder indirekt über andere Funktionen selbst aufrufen.

Die Parameter werden *by reference* übergeben, d.h. die formalen Parameter werden beim Aufruf des Unterprogramms durch die aktuellen Parameter ersetzt. Wird der Inhalt eines formalen Parameters geändert, wird auch der Inhalt des aktuellen Parameters geändert.

Ein Parameter kann auch *by value* übergeben werden, wenn er geklammert wird. Dies veranlaßt den Interpreter, den Parameter als arithmetischen Ausdruck anzusehen, dessen Ergebnis (in diesem Fall nur der Parameter) in einer temporären Variablen abgelegt wird. Beispiel:

```
TEXT$ = "Hallo"
I% = 5 + FUNKTION ((TEXT$))
```

GOSUB

Aufruf eines einfachen Unterprogramms.

Verwendung

```
    GOSUB Label
    ....

Label:
    ....
    RETURN [Label2]
```

Beschreibung

Die Programmausführung verzweigt zum angegebenen Label, wobei die aktuelle Position im Programm zwischengespeichert wird. Mit der RETURN-Anweisung kann wieder zu der der GOSUB- Anweisung folgenden Anweisung zurückgesprungen werden.

Parameter

`Label`	Ziel-Label, wohin verzweigt werden soll. Ein Label ist entweder ein Textstring oder eine Integerzahl. Wird ein Textstring als Label verwendet, muß er von einem Doppelpunkt gefolgt werden.
`Label2`	Die RETURN-Anweisung kann optional mit einem Label versehen werden. Es wird dann statt zu der der GOSUB-Anweisung folgenden Anweisung zu diesem Label verzweigt.

Anmerkungen

Die GOSUB-Anweisung wird aus historischen Gründen unterstützt. Sie hat jedoch einige Nachteile:

- Es können keine Parameter übergeben werden.
- Es können keine lokalen Variablen verwendet werden. Alle Variablen des Haupt- bzw. Unterprogramms, in dem sich die GOSUB-Anweisung befindet, können unkontrolliert verändert werden.

Es wird empfohlen, statt dessen die SUB-Anweisung für die Codierung von Prozeduren zu verwenden.

Die Anweisung ON TIMER (n) GOSUB verzweigt zum angegebenen Unterprogramm, wenn `n` Sekunden abgelaufen sind. Dies ist jedoch nur möglich, wenn bei Ablauf des Timers das Modul aktiv ist, in dem sich das Unterprogramm befindet. Ist dies nicht der Fall, wird mit der Verzweigung so lange gewartet, bis das entsprechende Modul wieder aktiv ist. Dieses Verhalten folgt aus der Tatsache, daß Unterprogramme per GOSUB nur im eigenen Modul angesprungen werden können.

Querverweise

ON...GOSUB, RETURN

GOTO

Ansprung eines Labels.

Verwendung

```
GOTO Label
```

Beschreibung

Das Programm springt den angegebenen Label an.

Parameter

`Label`	Ein Label ist eine Markierung, die eine bestimmte Stelle im Programm kennzeichnet. Dieses Label kann entweder aus einer Zahl oder aus einem Symbol bestehen, das von einem Doppelpunkt gefolgt wird.

Beispiel

Das klassische Ratespiel (rate meine Zahl):

```
    DEFINT A-Z
    RANDOMIZE
Start:
    N = INT (100 * RND + 1)
Raten:
    INPUT "Rate mal: ", I
    IF I = N THEN GOTO Ende
    IF I < N THEN PRINT "Zu klein!" : GOTO Raten
    IF I > N THEN PRINT "Zu Groß!"  : GOTO Raten
Ende:
    PRINT "Geraten!"
    GOTO Start
```

HEX$()

Umwandeln einer Zahl in einen Hex-String.

Verwendung

```
S$ = HEX$ (Ausdruck)
```

Parameter

`Ausdruck`	Der numerische Ausdruck, der umgerechnet und in einen Hex-String umgewandelt werden soll.

Returnwerte

Der gerundete ganzzahlige Anteil des Parameters wird in die Hexadezimale Schreibweise umgewandelt und als String zurückgeliefert. Der String enthält weder ein führendes "&H" noch ein eventuelles nachlaufendes "&".

Anmerkungen

Die Funktion OCT$() wandelt eine Zahl in einen Oktalstring um.

Querverweise

`OCT$()`

IF...THEN...ELSE

Bedingte Verzweigung.

Verwendung

Schreibweise 1:

```
IF Ausdruck THEN Anweisungen [ELSE Anweisungen]
```

Schreibweise 2:

```
IF Ausdruck THEN
    ....
    ....
[ELSEIF Ausdruck THEN
    ....
    ....]
[ELSE
    ....
    ....]
END IF
```

Beschreibung

Der Ausdruck wird ausgewertet. Hat er einen Wert ungleich Null, werden die Anweisungen ausgeführt, die sich hinter dem Wort `THEN` befinden. Hat er hingegen den Wert Null, werden, falls vorhanden, die Anweisungen hinter dem Wort **`ELSE`** ausgeführt.

Das Wort `ELSEIF` beginnt eine bedingte `ELSE`-Anweisung. Dieser Teil ist genau wie der `ELSE`- Teil der Anweisung optional.

Je nach der Komplexität der Anweisungen kann die einzeilige oder die mehrzeilige Form gewählt werden.

Parameter

`Ausdruck`	Der arithmetische Ausdruck wird ausgerechnet. Hat der einen Wert ungleich Null, wird der `THEN`-Teil der Anweisung ausgeführt ist der Wert Null, wird der `ELSEIF`- bzw. `ELSE`-Teil ausgeführt.

Anmerkungen

Der einzeilige Teil kann in jedem Zweig mehrere Anweisungen enthalten, wenn als Trenner zwischen den Anweisungen der Doppelpunkt verwendet wird:

```
IF A = 0 THEN B = 1 : PRINT "JA!" : ELSE B = 0 : PRINT "NEIN!"
```

Aus historischen Gründen kann auch nach dem Wort `THEN` direkt eine Zeilennummer (kein Zeilenlabel!) angegeben werden. Dies führt zum Ansprung der Zeilennummer, wenn die Bedingung wahr ist:

```
IF A = 0 THEN 9000
```

Beispiel

Die folgende Funktion bestimmt die Anzahl der Ziffern, die eine Integerzahl enthält:

```
FUNCTION Digits (N%)
IF N% < 10 THEN
    Digits = 1
ELSEIF N% < 100 THEN
    Digits = 2
ELSEIF N% < 1000 THEN
    Digits = 3
ELSEIF N% < 10000 THEN
    Digits = 4
ELSE
    Digits = 5
END IF
END FUNCTION

I = 1234
PRINT I " hat " Digits (I) " Ziffern"
```

INPUT

Anfordern einer Benutzereingabe.

Verwendung

```
INPUT ["Aufforderung"][ | ,] Variable [, Variable, ...]
```

Beschreibung

Diese Anweisung fordert eine Benutzereingabe an. Es wird ein Dialogfenster geöffnet. Dieses Dialogfenster nimmt eventuelle bislang noch nicht ausgegebene Texte sowie die Eingabeanforderung auf. In einem Editfeld kann eine Eingabe gemacht werden.

Wird nur eine Variable ohne Eingabeaufforderung angegeben, wird kein Editfeld erzeugt vielmehr wird die numerische Entsprechung des aktivierten Buttons (siehe Seite) in der Parameter-Variablen abgelegt. Wird eine Variable mit Eingabeaufforderung angegeben, werden ein Edit-Feld sowie ein OK-Button erzeugt. In diesem Fall wird kein Button-Status zurückgeliefert. Werden zwei oder mehr Variablen angegeben, nimmt die letzte Parameter-Variable den Wert des aktivierten Buttons auf. Es wird ein Editfeld erzeugt, in das der Anwender seine Eingaben macht. Diese Eingaben werden in der üblichen Weise mit den Parameter-Variablen verbunden. Fehlerhafte Eingaben führen zu einem Warntext und der Wiederholung der Eingabe.

Parameter

,	Eventuelle Kommata und/oder Semikolons werden aus Kompatibilitätsgründen ignoriert.
`Aufforderung`	Die Eingabeaufforderung. Diese wird, falls vorhanden, in das Fenster ausgegeben. Es kann sich dabei nur um eine Stringkonstante handeln, nicht um eine Variable oder einen Ausdruck.
`Variable`	Variablenliste zur Aufnahme der Benutzereingabe sowie des Status des aktivierten Buttons.

Beispiel

Eine INPUT-Anweisung mit 2 Zeilen und einem Prompt:

```
print using "~&" "Persönliche Daten"
print "Es sollen Ihre persönliche Daten erfaßt werden."
print "Dafür müssen einige Angaben gemacht werden."
input "Bitte geben Sie Vor- und Nachnamen ein:", VN$, NN$
```

Fehler! Datei kann nicht geöffnet werden!

Querverweise

INPUT #, LINE INPUT, LINE INPUT #

INPUT

Einlesen von ASCII-Daten aus einer Datei.

Verwendung

```
INPUT # Kanalnummer, Variable [, Variable, ...]
```

Beschreibung

Eine ASCII-Datei wird sequentiell gelesen. Die darin enthaltenen Daten werden in die angegebenen Variablen übertragen.

Die einzelnen Daten-Elemente können in der Datei voneinander durch Leerstellen, Tab-Zeichen, Kommata oder auch durch Zeilenvorschübe getrennt sein. Wird das Dateiende erreicht, ehe alle Variablen belegt sind, werden die noch nicht belegten Variablen nicht verändert.

Parameter

`Kanalnummer`	Die Kanalnummer der anzusprechenden Datei. Dies ist ein Integer-Ausdruck. Vor dem Ausdruck muß ein Doppelkreuz (#) stehen, um den Ausdruck als Kanalnummer zu kennzeichnen.
`Variable`	Die Liste der Variablen, die die Daten der Datei aufnehmen sollen.

Anmerkungen

Wird ein String auf eine numerische Variable zugewiesen, führt dies zu einem Laufzeitfehler 13 (`Ungültiger Datentyp`).

Eine Datei kann zeilenweise mit der LINE INPUT-Anweisung eingelesen werden.

Bei dieser Anweisung ist das Doppelkreuz vorgeschrieben, um sie nicht mit der INPUT-Anweisung für Tastatureingaben zu verwechseln.

QueDateien

INPUT, LINE INPUT, LINE INPUT #

INSTR()

Ermitteln der Position eines Teilstrings in einem String.

Verwendung

```
N = INSTR ([Anfang, ] String, Suchstring)
```

Parameter

`Anfang`	Optional kann die Startposition der Suche angegeben werden. Fehlt dieser Parameter wird beim Start des Strings (Zeichen Nr. 1) begonnen.
`String`	Der String, der durchsucht werden soll.
`Suchstring`	Der String, der gesucht werden soll.

Returnwerte

Wurde der Teilstring gefunden, wird die Position des Strings, beginnend mit der Zahl 1, zurückgeliefert. Ist der gesuchte Teilstring nicht im String vorhanden oder liegt die Startposition hinter dem Stringende, wird der Wert 0 zurückgeliefert.

INT()

Ermitteln des ganzzahligen Anteils einer Zahl.

Verwendung

```
N = INT (Ausdruck)
```

Parameter

`Ausdruck`	Der numerische Ausdruck, der ausgerechnet und dessen ganzzahliger Anteil ermittelt werden soll.

Returnwerte

Der ganzzahlige Anteil des Ausdrucks.

Anmerkungen

Der Unterschied zur Funktion FIX() liegt bei den negativen Zahlen. Diese Funktion liefert bei einer negativen Zahl die erste Ganzzahl kleiner als den Parameter zurück, während FIX() die erste Ganzzahl größer als den Parameter zurückliefert.

Beispiel

```
X = -12.34
Y = INT (X)          ' -13
Z = FIX (X)          ' -12
```

Querverweise

FIX()

KILL

Löschen einer Datei.

Verwendung

```
KILL Dateiname
```

Beschreibung

Die angegebene Datei wird gelöscht.

Parameter

`Dateiname`	String, der den Namen der zu löschenden Datei angibt. Der Name muß den Konventionen des verwendeten Dateisystems entsprechen.

Anmerkungen

Die Datei kann nur gelöscht werden, wenn sie weder von der eigenen noch von anderen Applikationen geöffnet wird und wenn sie nicht schreibgeschützt ist.

LBOUND()

Ermitteln der unteren Indexgrenze einer Array-Dimension.

Verwendung

```
N = LBOUND (Arrayname [, Dimension])
```

Beschreibung

Der niedrigste zulässige Wert für die angegebene Array-Dimension wird zurückgeliefert.

Parameter

`Arrayname`	Der Name des Arrays.
`Dimension`	Optional kann eine Dimension angegeben werden. Wird keine Angabe gemacht, wird der niedrigste zulässige Wert für die erste Dimension zurückgeliefert. Der Parameter darf Werte zwischen 1 und der für das Array definierten Anzahl Dimensionen annehmen.

Anmerkungen

Die Funktion UBOUND() liefert den höchsten zulässigen Wert für eine Array-Dimension zurück.

Beispiel

```
DIM TEST (5, 10, 15)

FOR I% = 1 TO 3
    PRINT "TEST Dimension " I "=" LBOUND (TEST, I)
NEXT
```

Querverweise

UBOUND()

LCASE$()

Konvertierung eines Strings in Kleinbuchstaben.

Verwendung

```
S$ = LCASE$ (Stringausdruck)
```

Parameter

`Stringausdruck`	Der Stringausdruck wird berechnet und der daraus resultierende String in Kleinbuchstaben umgewandelt.

Returnwerte

Der in Kleinbuchstaben konvertierte String.

Anmerkungen

Die Funktion UCASE$() konvertiert einen String in Großbuchstaben.

Querverweise

UCASE$()

LEFT$()

Erzeugen eines linken Teilstrings.

Verwendung

```
S$ = LEFT$ (Stringausdruck, n)
```

Parameter

`Stringausdruck`	Der Stringausdruck wird berechnet und als Ausgangsbasis genutzt.
`n`	Die Länge des linken Teilstrings. Ist als Länge der Wert 0 angegeben, wird ein Leerstring erzeugt.

Returnwerte

Der linke Teilstring.

Querverweise

MID$(), RIGHT$()

LEN()

Ermitteln der Länge eines Strings.

Verwendung

```
N = LEN (Stringausdruck) oder
N = LEN (Variable)
```

Parameter

`Stringausdruck`	Der angegebene Stringausdruck wird berechnet und seine Länge zurückgeliefert.
`Variable`	Wird eine Variable angegeben, wird der für diese Variable notwendige Speicherbereich zurückgeliefert. Dies ist vor allem in Zusammenhang mit Pointer-Arithmetik interessant.

Returnwerte

Die Länge des Strings bzw. die Größe der Variablen.

LET

Zuweisung eines Ausdrucks auf eine Variable.

Verwendung

```
[LET] Variable = Ausdruck
```

Beschreibung

Vor einer Wertzuweisung auf eine Variable kann optional das Wort `LET` stehen. Ist dies der Fall, wird das Wort ignoriert.

LINE INPUT

Einlesen einer Zeile vom Bildschirm.

Verwendung

```
LINE INPUT ["Aufforderung"] Stringvariable
```

Beschreibung

Es wird eine Zeile vom Benutzer angefordert. Der Inhalt dieser Zeile wird in die angegebene Stringvariable übertragen.

Parameter

`Aufforderung`	Die Eingabeaufforderung. Diese wird, falls vorhanden, in das Fenster ausgegeben. Es kann sich dabei nur um eine Stringkonstante handeln, nicht um eine Variable oder einen Ausdruck.
`Stringvariable`	Die Stringvariable, die die Zeile aufnehmen soll.

Querverweise

INPUT, INPUT #, LINE INPUT #

LINE INPUT

Zeilenweises Einlesen einer ASCII-Datei.

Verwendung

```
LINE INPUT #Kanalnummer, Stringvariable
```

Beschreibung

Die über die Kanalnummer angesprochene Datei wird zeilenweise in eine Stringvariable eingelesen. Ist das Dateiende erreicht, wird der Inhalt der Variablen nicht verändert.

Parameter

`Kanalnummer`	Die Kanalnummer der anzusprechenden Datei. Dies ist ein Integer-Ausdruck. Vor dem Ausdruck kann optional auch ein Doppelkreuz (#) stehen, um den Ausdruck als Kanalnummer zu kennzeichnen.
`Stringvariable`	Die Stringvariable, die die Zeile aufnehmen soll.

Beispiel

Das folgende Programm gibt eine Datei auf dem Bildschirm aus:

```
INPUT "Dateiname:", NAME$
OPEN "R", 1, NAME$
WHILE NOT EOF (1)
    LINE INPUT #1, ZEILE$
    PRINT ZEILE$
WEND
END
```

Querverweise

INPUT, INPUT #, LINE INPUT

LOCK

Sperren einer Datei gegenüber anderen Applikationen.

Verwendung

```
LOCK Kanalnummer
```

Beschreibung

Die Datei wird gegen Zugriffe von anderen Applikationen aus gesperrt. Solange die Sperre aktiv ist, kann keine andere Applikation auf die Datei zugreifen. Wird dieser Zugriff versucht, bleibt die andere Applikation so lange blockiert, bis die Datei wieder freigegeben wird.

Parameter

`Kanalnummer`	Die Kanalnummer der anzusprechenden Datei. Dies ist ein Integer-Ausdruck. Vor dem Ausdruck muß ein Doppelkreuz (#) stehen, um den Ausdruck als Kanalnummer zu kennzeichnen.

Anmerkungen

Diese Funktion ist abhängig vom Betriebssystem. Unter DOS wird beispielsweise nach einer gewissen Zeit mit einem Fehlercode abgebrochen.

Querverweise

`UNLOCK`

LOF()

Ermitteln der Größe einer Datei.

Verwendung

```
N = LOF (Kanalnummer)
```

Beschreibung

Diese Funktion liefert die Größe einer Datei in Bytes zurück.

Parameter

`Kanalnummer`	Die Kanalnummer der anzusprechenden Datei. Das sonst optionale Doppelkreuz darf hier nicht angegeben werden.

Returnwerte

Die Größe der Datei in Bytes.

Querverweise

EOF()

LOG()

Berechnen des Logarithmus Naturalis auf der Basis von *e*.

Verwendung

```
N = LOG (Ausdruck)
```

Parameter

`Ausdruck`	Der numerische Ausdruck, dessen Logarithmus ermittelt werden soll.

Returnwert

Der Logarithmus Naturalis des Ausdrucks.

Anmerkungen

Der Wert wird immer als lange Gleitpunktzahl berechnet, wenn der Parameter ebenfalls eine lange Gleitpunktzahl ist sonst ist das Ergebnis einfach genau.

Beispiel

Der Zehnerlogarithmus kann mit folgender Funktion berechnet werden:

```
FUNCTION LOG10 (N#)
    LOG10 = LOG (N) / LOG (10#)
END FUNCTION
```

Querverweise

EXP()

LPRINT

Ausgabe von Daten auf dem Drucker.

Verwendung

```
LPRINT
LPRINT [USING Format] Ausdruck [[|,] Ausdruck ...] [|,]
```

Beschreibung

Die angegebenen Ausdrücke werden auf dem Drucker ausgegeben. Wird mit der USING-Klausel ein Formatstring angegeben, werden die Daten vorher formatiert.

Wird die Anweisung ohne Parameter verwendet, erfolgt ein Zeilenvorschub.

Parameter

`Format`	Der Formatstring wird bei der USING-Klausel mit angegeben er enthält Formatierungsanweisungen. Die zulässigen Formatzeichen sind auf Seite beschrieben.
`Ausdruck`	Die Liste der auszugebenden Ausdrücke. Jeder Ausdruck wird ausgerechnet und nach eventueller vorheriger Formatierung ausgegeben. Soll ein String ausgegeben werden, wenn der Formatstring einen numerischen Wert beschreibt, führt dies zu einem Laufzeitfehler 13 (`Ungültiger Datentyp`).
`,`	Wird ein Komma als Trenner oder hinter dem letzten Ausdruck verwendet, werden die Daten in Zonen von etwa 14 Leerstellen Breite plaziert.
`;`	Wird ein Semikolon als Trenner oder hinter dem letzten Ausdruck verwendet, werden die Daten unmittelbar hintereinander plaziert. Das gleiche gilt für Leerstellen zwischen den Ausdrücken.

Anmerkungen

Wird die Liste der Ausdrücke weder durch ein Komma noch durch ein Semikolon abgeschlossen, erfolgt ein Zeilenvorschub.

Fehlt die USING-Klausel, werden Zahlen mit einer nachfolgenden Leerstelle ausgegeben. Ist die Zahl positiv, wird eine führende Leerstelle ausgegeben. Bei negativen Zahlen wird stattdessen ein Minuszeichen ausgegeben.

Die Funktionen SPC() und TAB() können zur Formatierung herangezogen werden.

Querverweise

PRINT, SPC(), TAB(), WRITE #

LTRIM$()

Entfernen von führenden Leerstellen im String.

Verwendung

```
S$ = LTRIM$ (Stringausdruck)
```

Parameter

`Stringausdruck`	Der Stringausdruck wird berechnet und als Ausgangsbasis genutzt.

Returnwerte

Der berechnete String ohne führende Leerstellen.

Anmerkungen

Die Funktion RTRIM$() entfernt nachlaufende Leerstellen.

Querverweise

`RTRIM$()`

MID$()

Erzeugen oder Ersetzen eines Teilstrings.

Verwendung

```
S$ = MID$ (Stringausdruck, Anfang [, Länge])
```

Parameter

`Stringausdruck`	Der Stringausdruck wird berechnet und als Quellstring genutzt.
`Anfang`	Die Position innerhalb des Strings, wo die Erzeugung bzw. die Ersetzung beginnen soll.
`Länge`	Die Länge des zu erzeugenden Teilstrings.

Returnwerte

Der erzeugte Teilstring.

Querverweise

`LEFT$()`, `RIGHT$()`

MKDIR

Einrichten eines Unterverzeichnisses.

Verwendung

```
MKDIR Verzeichnis
```

Beschreibung

Das angegebene Unterverzeichnis wird neu eingerichtet.

Parameter

`Verzeichnis`	String, der den Namen des neuen Verzeichnisses angibt. Der Name muß den Regeln des verwendeten Dateisystems entsprechen.

Anmerkungen

Diese Anweisung wird direkt auf das entsprechende Betriebssystemkommando abgebildet. Die dort geltenden Restriktionen gelten auch für diese Anweisung.

Querverweise

CHDIR, RMDIR

NAME

Umbenennen einer Datei.

Verwendung

NAME AlterName AS NeuerName

Beschreibung

Die angegebene Datei erhält den neuen Namen.

Parameter

`AlterName`	Der Name der umzubenennenden Datei.
`NeuerName`	Der neue Dateiname.

Anmerkungen

Die Anweisung wird unmittelbar auf das darunterliegende Dateisystem abgebildet. Die dort gültigen Bedingungen für das Umbenennen von Dateien müssen daher beachtet werden. So müssen die Dateinamen den gültigen Konventionen entsprechen.

Als Seiteneffekt kann eine Datei unter MS-DOS von einem Verzeichnis in ein anderes Verzeichnis verlegt werden.

OCT$()

Umwandeln einer Zahl in einen Oktalstring.

Verwendung

```
S$ = OCT$ (Ausdruck)
```

Parameter

`Ausdruck`	Der numerische Ausdruck, der umgerechnet und in einen Oktalstring umgewandelt werden soll.

Returnwerte

Der gerundete ganzzahlige Anteil des Parameters wird in die Oktale Schreibweise umgewandelt und als String zurückgeliefert. Der String enthält weder ein führendes "&O" noch ein eventuelles nachlaufendes "&".

Anmerkungen

Die Funktion HEX$() wandelt eine Zahl in einen Hex-String um.

Querverweise

HEX$()

ON ERROR GOTO

Definition einer Fehlerbehandlung.

Verwendung

```
ON ERROR GOTO [Label | 0]
```

Beschreibung

Fehler, die während des Programmablaufs auftreten, können durch die `ON ERROR GOTO`-Anweisung abgefangen werden. Durch die Angabe der `ON ERROR`-Anweisung wird im Fehlerfall auf das angegebene Label im Programm verzweigt. Die dort definierte Routine kann den Fehlercode abfragen und gezielt auf den Fehler reagieren. Sie hat anschließend die Möglichkeit, mit der RESUME- Anweisung den Programmlauf an der Stelle fortzuführen, wo der Fehler auftrat.

Parameter

`Label`	Label, zu dem das Programm verzweigen soll.
`0`	Wird die Zahl 0 angegeben, wird die programmspezifische Fehlerbehandlung wieder deaktiviert. Wird die Anweisung innerhalb einer Fehlerbehandlung ausgeführt, erfolgt sofort die übliche Fehlermeldung.

Beispiel

Das folgende Beispiel bricht den Programmlauf bei einem Speichermangel ab. Alle anderen Fehler werden von der Applikation bearbeitet. Durch die ERROR-Anweisung wird unmittelbar die Fehlerbehandlung der Applikation aktiviert, da es sich um einen Fehler handelt, der innerhalb der Fehlerbehandlung aktiviert wurde.

```
        ON ERROR GOTO Fehler
        ....

    Fehler:
        IF ERR = 7 THEN PRINT "Speichermangel!" : END
        ERROR ERR
```

Anmerkungen

Die Anweisung kann in einem Programm beliebig oft wiederholt werden.

Querverweise

ERR(), ERL(), ERROR, RESUME

ON n GOSUB

Aufruf eines Unterprogrammteils auf Grund des Wertes eines Ausdrucks.

Verwendung

```
ON Ausdruck GOSUB Label [, Label, ...]
```

Beschreibung

Der Ausdruck wird ausgerechnet und in eine Integerzahl konvertiert. Ist der Wert des Ausdrucks Eins, wird anschließend der erste Label der Liste angesprungen. Ist der Wert Zwei, wird der zweite Label angesprungen etc. Dabei wird die Rückkehr-Adresse intern gespeichert, so daß eine Rückkehr mit Hilfe der RETURN-Anweisung möglich ist. Ist der Ausdruck kleiner als Eins oder größer als die Anzahl der angegebenen Labels, wird nicht verzweigt. Ist der Ausdruck negativ oder größer als 255, wird ein Laufzeitfehler 5 (`Fehler bei Funktionsaufruf`) erzeugt.

Parameter

`Ausdruck`	Der arithmetische Ausdruck, dessen Wert zur Verzweigung herangezogen wird.
`Label`	Die Liste der Labels, zu denen verzweigt wird. Ein Label ist eine Markierung, die eine bestimmte Stelle im Programm kennzeichnet. Dieses Label kann entweder aus einer Zahl oder aus einem Symbol bestehen, das von einem Doppelpunkt gefolgt wird.

Anmerkungen

Die Liste der Labels kann bis zu 255 Einträge umfassen.

Beispiel

Ein einfaches Menüsystem:

```
Start:
    PRINT "1. Abrechnen"
    PRINT "2. Datensicherung"
    PRINT "3. Programmende"
    INPUT "Ihre Wahl :", I%
    IF I% >= 1 AND I% <= 3 THEN
        ON I% GOSUB Abrechnen, Sichern, Ende
    ELSE
        PRINT "Falsche Eingabe!"
    END IF
    GOTO Start
Abrechnen:
    ....
    RETURN
Sichern:
    ....
    RETURN
Ende:
    END
```

Querverweise

ON n GOTO

ON n GOTO

Verzweigung auf Grund des Wertes eines Ausdrucks.

Verwendung

```
ON Ausdruck GOTO Label [, Label, ...]
```

Beschreibung

Der Ausdruck wird ausgerechnet und in eine Integerzahl konvertiert. Ist der Wert des Ausdrucks Eins, wird anschließend der erste Label der Liste angesprungen. Ist der Wert Zwei, wird der zweite Label angesprungen etc. Ist der Ausdruck kleiner als Eins oder größer als die Anzahl der angegebenen Labels, wird nicht verzweigt. Ist der Ausdruck negativ oder größer als 255, wird ein Laufzeitfehler 5 (`Fehler bei Funktionsaufruf`) erzeugt.

Parameter

`Ausdruck`	Der arithmetische Ausdruck, dessen Wert zur Verzweigung herangezogen wird.
`Label`	Die Liste der Labels, zu denen verzweigt wird. Ein Label ist eine Markierung, die eine bestimmte Stelle im Programm kennzeichnet. Dieses Label kann entweder aus einer Zahl oder aus einem Symbol bestehen, das von einem Doppelpunkt gefolgt wird.

Anmerkungen

Die Liste der Labels kann bis zu 255 Einträge umfassen.

Beispiel

Ein einfaches Menüsystem:

```
Start:
    PRINT "1. Abrechnen"
    PRINT "2. Datensicherung"
    PRINT "3. Programmende"
    INPUT "Ihre Wahl :", I%
    IF I% >= 1 AND I% <= 3 THEN
        ON I% GOTO Abrechnen, Sichern, Ende
    ELSE
        PRINT "Falsche Eingabe!"
    END IF
    GOTO Start
Abrechnen:
    ....
    GOTO Start
Sichern:
    ....
    GOTO Start
Ende:
    END
```

Querverweise

ON...GOSUB

ON TIMER(n) GOSUB

Ansprung eines Unterprogrammteils aufgrund des Ablaufs eines Zeitgebers.

Verwendung

```
ON TIMER(n) GOSUB Label
```

Beschreibung

Wenn diese Anweisung ausgeführt wird, wird ein Zeitgeber gestartet. Ist die angegebene Zeit abgelaufen, wird zum angegebenen Label verzweigt. Beim Ansprung wird die aktuelle Position innerhalb des Programms zwischengespeichert, so daß aus der Bearbeitungsroutine mit Hilfe der RETURN- Anweisung jederzeit an die Stelle zurückgekehrt werden kann, an der das Programm unterbrochen wurde.

Parameter

`n`	Die Zeit in Sekunden, nach der das Programmteil angesprungen werden soll. Dieser Wert darf sich zwischen 1 und 86400 bewegen, was einem Zeitraum zwischen einer Sekunde und 24 Stunden entspricht.
`Label`	Ein Label ist eine Markierung, die eine bestimmte Stelle im Programm kennzeichnet. Dieses Label kann entweder aus einer Zahl oder aus einem Symbol bestehen, das von einem Doppelpunkt gefolgt wird.

Anmerkungen

Mit Hilfe der TIMER-Anweisung kann der Zeitgeber gestartet und angehalten werden.

Das Unterprogramm kann nur ausgeführt werden, wenn das Modul, in dem sich das Unterprogramm befindet, aktiv ist. Ist dies nicht der Fall, wird mit der Ausführung gewartet, bis das Modul wieder aktiv ist.

Beispiel

Das folgende Programm gibt nach einer Stunde einen Text aus:

```
        TIMER ON
        ON TIMER (3600) GOSUB Wecken
        WHILE 1    : REM Totschleife
        WEND
        END

    Wecken:
        PRINT "Die Zeit ist abgelaufen!"
        RETURN
```

Querverweise

TIMER

OPEN

Öffnen einer ASCII-Datei.

Verwendung

```
OPEN Zugriffsart, #Kanalnummer, Dateiname [, Pufferlänge]
```

Beschreibung

Die `OPEN`-Anweisung öffnet eine Datei und verknüpft die Datei mit einer Kanalnummer. Die Datei kann entweder zum Lesen, zum Schreiben oder für beide Zugriffsarten geöffnet werden.

Parameter

`Zugriffsart`	Ein Stringausdruck, dessen erster Buchstabe einer der folgenden sein muß: O Schreibzugriff I Lesezugriff R Schreib/Lesezugriff A Schreibzugriff mit Anfügen Die Zugriffsart "A" gestattet das Anfügen von Daten an das Ende einer Datei.
`Kanalnummer`	Die Kanalnummer der anzusprechenden Datei. Dies ist ein Integer-Ausdruck. Vor dem Ausdruck kann optional auch ein Doppelkreuz (#) stehen, um den Ausdruck als Kanalnummer zu kennzeichnen.
`Dateiname`	Der Name der Datei.
`Pufferlänge`	Diese Angabe kann aus Kompatibilitätsgründen gemacht werden. Sie wird ignoriert.

Beispiel

Das folgende Beispiel fügt einen Satz an eine Datei LOG an:

```
SET LOG$ = "Formatieren durchgeführt."

OPEN "A", #1, "LOG"
PRINT #1, LOG$
```

CLOSE Dateien

Querverweise

`CLOSE`

OPTION BASE

Einstellen der Index-Untergrenze.

Verwendung

```
OPTION BASE 0 | 1
```

Beschreibung

Die Index-Untergrenze für Array-Dimensionen ohne explizite Bereichsangabe kann mit dieser Anweisung entweder auf den Wert 0 (Voreinstellung) oder den Wert 1 eingestellt werden.

Parameter

`0 | 1`	Die Untergrenze ist entweder 0 oder 1.

Anmerkungen

Greifen mehrere Programme auf ein in einem COMMON-Bereich deklariertes Array zu, muß sichergestellt sein, daß alle Programme die gleiche Index-Untergrenze haben, da das Array sonst von den Programmen unterschiedlich adressiert wird. Daher wird die Verwendung dieser Anweisung nicht empfohlen. Die AS-Klause in der DIM-Anweisung läßt einen breiteren Spielraum bezüglich der Indexgrenzen zu.

Querverweise

DIM, REDIM, COMMON

PRINT

Ausgabe von Daten auf dem Bildschirm.

Verwendung

```
PRINT
PRINT [USING Format] Ausdruck [[|,] Ausdruck ...] [|,]
```

Beschreibung

Die angegebenen Ausdrücke werden auf dem Bildschirm ausgegeben. Wird mit der USING-Klausel ein Formatstring angegeben, werden die Daten vorher formatiert.

Bei fensterorientierten Benutzeroberflächen erfolgen die Ausgaben in ein Dialogfenster.

Wird die Anweisung ohne Parameter verwendet, erfolgt ein Zeilenvorschub.

Parameter

`Format`	Der Formatstring wird bei der USING-Klausel mit angegeben; er enthält Formatierungsanweisungen. Die zulässigen Formatzeichen sind auf Seite beschrieben.
`Ausdruck`	Die Liste der auszugebenden Ausdrücke. Jeder Ausdruck wird ausgerechnet und nach eventueller vorheriger Formatierung ausgegeben. Soll ein String ausgegeben werden, wenn der Formatstring einen numerischen Wert beschreibt, führt dies zu einem Laufzeitfehler 13 (`Ungültiger Datentyp`).
,	Wird ein Komma als Trenner oder hinter dem letzten Ausdruck verwendet, werden die Daten in Zonen von etwa 14 Leerstellen Breite plaziert.
;	Wird ein Semikolon als Trenner oder hinter dem letzten Ausdruck verwendet, werden die Daten unmittelbar hintereinander plaziert. Das gleiche gilt für Leerstellen zwischen den Ausdrücken.

Anmerkungen

Wird die Liste der Ausdrücke weder durch ein Komma noch durch ein Semikolon abgeschlossen, erfolgt ein Zeilenvorschub.

Fehlt die USING-Klausel, werden Zahlen mit einer nachfolgenden Leerstelle ausgegeben. Ist die Zahl positiv, wird eine führende Leerstelle ausgegeben. Bei negativen Zahlen wird stattdessen ein Minuszeichen ausgegeben.

Querverweise

`LPRINT`, `SPC()`, `TAB()`, `WRITE`

PRINT

Ausgabe von Daten in eine ASCII-Datei.

Verwendung

```
PRINT #Kanal
PRINT #Kanal, [USING Format] Ausdruck [[|,] Ausdruck...] [|,]
```

Beschreibung

Die angegebenen Ausdrücke werden in die durch die Kanalnummer bezeichnete Datei ausgegeben. Wird mit der USING-Klausel ein Formatstring angegeben, werden die Daten vorher formatiert. Wird die Anweisung nur mit Kanalnummer ohne Parameter verwendet, erfolgt ein Zeilenvorschub.

Parameter

`Kanal`	Die Kanalnummer der anzusprechenden Datei. Dies ist ein Integer-Ausdruck. Vor dem Ausdruck muß ein Doppelkreuz (#) stehen, um den Ausdruck als Kanalnummer zu kennzeichnen.
`Format`	Der Formatstring wird bei der USING-Klausel mit angegeben er enthält Formatierungsanweisungen. Die zulässigen Formatzeichen sind auf Seite beschrieben.
`Ausdruck`	Die Liste der auszugebenden Ausdrücke. Jeder Ausdruck wird ausgerechnet und nach eventueller vorheriger Formatierung ausgegeben. Soll ein String ausgegeben werden, wenn der Formatstring einen numerischen Wert beschreibt, führt dies zu einem Laufzeitfehler 13 (`Ungültiger Datentyp`).
,	Wird ein Komma als Trenner oder hinter dem letzten Ausdruck verwendet, werden die Daten in Zonen von etwa 14 Leerstellen Breite plaziert.
;	Wird ein Semikolon als Trenner oder hinter dem letzten Ausdruck verwendet, werden die Daten unmitDateienntereinander plaziert. Das gleiche gilt für Leerstellen zwischen den Ausdrücken.

Anmerkungen

Wird die Liste der Ausdrücke weder durch ein Komma noch durch ein Semikolon abgeschlossen, erfolgt ein Zeilenvorschub.

Fehlt die USING-Klausel, werden Zahlen mit einer nachfolgenden Leerstelle ausgegeben. Ist die Zahl positiv, wird eine führende Leerstelle ausgegeben. Bei negativen Zahlen wird stattdessen ein Minuszeichen ausgegeben.

Querverweise

`PRINT`, `SPC()`, `TAB()`, `WRITE #`

RANDOMIZE

Initialisieren des Zufallszahlen-Generators.

Verwendung

```
RANDOMIZE [Ausdruck]
```

Beschreibung

Der Zufallszahlen-Generator wird mit dem angegebenen Ausdruck initialisiert. Fehlt der Ausdruck, wird ein zufälliger Wert zur Initialisierung herangezogen.

Parameter

`Ausdruck`	Initialisierungswert des Generators. Er darf sich zwischen den Werten -32767 und 32767 bewegen.

Anmerkungen

Die Kette der Zufallszahlen kann mit der Funktion RND() abgerufen werden.

Querverweise

`RND()`

READ

Einlesen von Datenzeilen.

Verwendung

```
READ Variable [, Variable ...]
```

Beschreibung

Diese Anweisung liest die mit der `READ`-Anweisung definierten Konstanten ein die übergebene Liste der Variablen ein. Der Einlesevorgang beginnt bei der ersten Datenzeile des Programms und setzt sich über alle Datenzeilen fort. Wird die `READ`-Anweisung aufgerufen, nachdem alle Datenzeilen eingelesen worden sind, wird ein Laufzeitfehler 4 (`READ ohne DATA`) erzeugt. Ebenso wird ein Laufzeitfehler 13 (`Ungültiger Datentyp`) erzeugt, wenn versucht wird, einen Stringwert in eine numerische Variable einzulesen.

Parameter

`Variable`	Variablenliste, die die Konstanten aufnehmen soll.

Querverweise

`DATA`, `RESTORE`

REDIM

Ändern der Größe eines Arrays.

Verwendung

```
REDIM [SHARED] Array [AS Typ] [, Array [AS Typ], ...]
```

Beschreibung

Diese Anweisung ändert die Größe der angegebenen Arrays. Die Arrays werden neu eingerichtet und mit dem Wert 0 bzw. dem Leerstring initialisiert.

Parameter

`SHARED`	Dieses optionale Wort kennzeichnet die angegebenen Arrays als für Unterprogramme zugänglich. Es kann nur im Hauptprogramm verwendet werden.
`Array`	Name des Arrays. Die Typkennung kann entfallen, wenn der Typ entweder durch eine DEFxxx-Anweisung oder durch die AS-Klausel vorgegeben ist. Hinter dem Namen folgt noch die Beschreibung der Dimensionen (s.u.).
`AS Typ`	Deklaration des Variablentyps. Neben den durch die TYPE- oder CLASS-Anweisung definierten Datentypen für Strukturen kann dies noch eines der folgenden Worte sein: INTEGER - kurze Integerzahl LONG - lange Integerzahl SINGLE - kurze Gleitpunktzahl DOUBLE - lange Gleitpunktzahl STRING - String variabler Länge STRING*n - String fester Länge (n)

Bei durch TYPE oder CLASS definierten Datentypen kann vor dem Typ auch eine der Klauseln POINTER TO oder PTR TO angegeben werden. Dies bewirkt, daß nicht eine Struktur, sondern nur ein Zeiger auf die Struktur definiert wird.

Anmerkungen

Die Dimensionen des Arrays werden wie folgt angegeben:

```
Name ( Anzahl | Untergrenze TO Obergrenze [, ...] )
```

Jede Dimension kann entweder durch eine Zahl oder durch einen Bereich beschrieben werden. Wird nur eine Zahl angegeben, erstreckt sich der Bereich von der durch die OPTION BASE-Anweisung eingestellten Untergrenze (0 oder 1, die Vorgabe ist 0) über die angegebene Anzahl der Elemente. Beispiele:

```
DIM A (5, 10)               ' 5 mal 10 Elemente
DIM B (-3 TO 3)             ' Bereichsangabe
DIM C (5, 1 TO 10, 3)       ' Mischangaben sind möglich
```

Arrays in COMMON-Blöcken können nicht in der Größe verändert werden.

Querverweise

DIM, ERASE, OPTION BASE

REPLY()

Abschließen einer Bildschirmausgabe und Ausgabe einer Message-Box.

Verwendung

```
Antwort = REPLY
```

Beschreibung

Diese Funktion gibt alle bis dahin erfolgten Druckausgaben in einer Message Box aus und liefert den Wert des aktivierten Buttons als Ergebnis zurück. Sämtliche auf Seite definierten Dialogformate werden ausgewertet.

Returnwerte

Der Return ist einer der folgenden:

-1	OK	(entspricht TRUE)
0	Abbruch	(entspricht FALSE)
1	Ja	
2	Nein	
3	Wiederholen	

Anmerkungen

Diese Funktion ist nur in Implementationen in grafischen Oberflächen definiert. Unter MS-DOS beispielsweise ist diese Funktion undefiniert.

Beispiel

```
Datei$ = "TEST.DAT"
print using "~?4&" "Datei sichern"
print "Soll die Datei " Datei$ " gesichert werden?"
```

RESET

Schließen aller Dateien.

Verwendung

```
RESET
```

Beschreibung

Alle offenen Dateien werden geschlossen.

Anmerkungen

Diese Anweisung ist historisch bedingt und ein Synonym für die CLOSE-Anweisung ohne Parameter.

RESTORE

Zurücksetzen des Lesezeigers für die READ-Anweisung.

Verwendung

```
RESTORE [Label]
```

Beschreibung

Der interne Lesezeiger, der auf das nächste Datenelement zeigt, das mit der READ-Anweisung eingelesen werden soll, wird entweder auf den Anfang oder auf die angegebene Zeile zurückgesetzt. Die nächste **READ**-Anweisung liest die Daten dann von der angegebenen Position an. Ist der Label nicht unmittelbar vor einer DATA-Zeile, wird aDateienquentiell nächste DATA-Zeile hinter dem Label positioniert.

Parameter

`Label`	Ein Label ist eine Markierung, die eine bestimmte Stelle im Programm kennzeichnet. Dieses Label kann entweder aus einer Zahl oder aus einem Symbol bestehen, das von einem Doppelpunkt gefolgt wird.

Querverweise

DATA, READ

RESUME

Fortführung des Programms nach einem Fehler.

Verwendung

```
RESUME [0]
RESUME NEXT
RESUME Label
```

Beschreibung

Wird nach dem Auftreten eines Fehlers die Fehlerbehandlungsroutine angesprungen, befindet sich das Programm in einem besonderen Fehlerbehandlungs-Status. Dieser Status führt vor allem dazu, daß bei einem erneuten Auftreten eines Fehlers die Fehlerbehandlungsroutine nicht mehr erneut angesprungen wird, sondern vielmehr die Applikation benachrichtigt wird. Dieser Fehlerbehandlungs- Status wird durch die `RESUME`-Anweisung wieder zurückgenommen.

Parameter

`0`	Der optionale Parameter 0 führt dazu, daß das Programm an der Stelle fortgesetzt wird, die zum Fehler geführt hat.
`NEXT`	Das Programm wird bei der Anweisung fortgesetzt, die der Anweisung folgt, die den Fehler verursacht hat.
`Label`	Das Programm wird bei dem angegebenen Label fortgesetzt.

Beispiel

Das folgende Beispiel durchläuft eine Schleife von 3 bis -3. Sie wird beim Erreichen des Werts -1 durch die Fehlerbehandlung abgebrochen, da die Funktion SQR() keine negativen Parameter akzeptiert:

```
        ON ERROR GOTO Fehler              ' Aktivieren

        FOR I = 3 TO -3 STEP -1           ' Beginn der Schleife
        PRINT SQR (I)                     ' Ausdruck von _I
        NEXT                              ' Ende der Schleife
    Ende:
        END                               ' Programmende

    Fehler:
        IF ERR = 5 THEN                   ' Falscher Parameter?
            PRINT "Negativ!"              ' Dann ausgeben
            RESUME Ende                   ' und raus aus Schleife
        END IF
        ERROR ERR                         ' sonst Applikation rufen
```

Querverweise

ERR(), ERL(), ERROR, ON ERROR GOTO

RETURN

Rückkehr aus einem mit der GOSUB-Anweisung aufgerufenen Unterprogramms.

Verwendung

```
RETURN [Label]
```

Beschreibung

Es wird zu der Position im Programm zurückverzweigt, in der die letzte GOSUB-Anweisung abgearbeitet wurde.

Parameter

`Label`	Ziel-Label, wohin verzweigt werden soll. Ein Label ist entweder ein Textstring oder eine Integerzahl. Wird ein Textstring als Label verwendet, muß er von einem Doppelpunkt gefolgt werden. Ist ein Label angegeben, wird die zwischengespeicherte Position ignoriert.

Anmerkungen

Die Anweisungs-Kombination GOSUB/RETURN wird aus historischen Gründen unterstützt. Sie hat jedoch einige Nachteile:

- Es können keine Parameter übergeben werden.
- Es können keine lokalen Variablen verwendet werden. Alle Variablen des Haupt- bzw. Unterpgrogramms, in dem sich die GOSUB-Anweisung befindet, können unkontrolliert verändert werden.

Es wird empfohlen, statt dessen die SUB-Anweisung für die Codierung von Prozeduren zu verwenden.

Querverweise

GOSUB, ON n GOSUB

RIGHT$()

Erzeugen eines rechten Teilstrings.

Verwendung

```
S$ = RIGHT$ (Stringausdruck, n)
```

Parameter

`Stringausdruck`	Der Stringausdruck wird berechnet und als Ausgangsbasis genutzt.
`n`	Die Länge des rechten Teilstrings. Ist als Länge der Wert 0 angegeben, wird ein Leerstring erzeugt.

Returnwerte

Der rechte Teilstring.

Querverweise

MID$(), LEFT$()

RMDIR

Löschen eines Unterverzeichnisses.

Verwendung

```
RMDIR Verzeichnis
```

Beschreibung

Das angegebene Unterverzeichnis wird gelöscht.

Parameter

`Verzeichnis`	String, der den Namen des des zu löschenden Verzeichnisses angibt. Der Name muß den Regeln des verwendeten Dateisystems entsprechen.

Anmerkungen

Diese Anweisung wird direkt auf das entsprechende Betriebssystemkommando abgebildet. Die dort geltenden Restriktionen gelten auch für diese Anweisung. So muß normalerweise das zu löschende Unterverzeichnis leer sein, d.h. es darf weder Dateien noch weitere Unterverzeichnisse enthalten.

Querverweise

CHDIR, MKDIR

RND()

Abrufen eDateienllszahl.

Verwendung

```
N = RND [(n)]
```

Parameter

`(n)`	Ist der Parameter angegeben, wird immer die zuletzt erzeugte Zufallszahl zurückgeliefert.

Returnwerte

Die nächste Zufallszahl in der Reihe. Der Wert liegt zwischen 0 und 1.

Anmerkungen

Die Reihe der Zufallszahlen kann mit der RANDOMIZE-Anweisung beliebig beeinflußt werden.

Beispiel

Um Integer-Zufallszahlen in einem bestimmten Bereich zu erhalten, kann folgende Funktion verwendet werden:

```
FUNCTION INTRND (LOWER%, UPPER%)
    INTRND = INT ((UPPER - LOWER + 1) * RND + LOWER)
END FUNCTION
```

Querverweise

RANDOMIZE

RTRIM$()

Entfernen von nachlaufenden Leerstellen im String.

Verwendung

```
S$ = RTRIM$ (Stringausdruck)
```

Parameter

`Stringausdruck`	Der Stringausdruck wird berechnet und als Ausgangsbasis genutzt.

Returnwerte

Der berechnete String ohne nachlaufende Leerstellen.

Anmerkungen

Die Funktion LTRIM$() entfernt führende Leerstellen.

Querverweise

LTRIM$()

SELECT CASE

Mehrfache Verzweigung auf Grund eines Ausdrucks.

Verwendung

```
SELECT CASE Ausdruck
CASE Vergleiche
    ....
    ....
[CASE Vergleiche
    ....
    ....]
[CASE ELSE
    ....
    ....]
END CASE
```

Vergleiche sind:

```
Ausdruck [, Ausdruck, ...]
Ausdruck TO Ausdruck
IS Vergleichsoperator Ausdruck
```

Beschreibung

Der angegebene Ausdruck wird mit den Vergleichs-Ausdrucken verglichen. Findet sich dabei eine wahre Bedingung, wird in den Anweisungs-Block verzweigt, der dem Vergleichs-Ausdruck folgt. Trifft kein Vergleich zu, wird zu dem Anweisungsblock verzweigt, der den Worten `CASE ELSE` folgt. Ist kein derartiger Block definiert, wird DAS Programm hinter der `CASE`-Anweisung fortgesetzt.

Parameter

`Ausdruck`	Der Ausdruck, der zum Vergleich herangezogen soll.
`Vergleiche`	Ein Ausdruck oder mehrere Ausdrücke der folgenden Form:

`A1, A2, A2`	einer der Ausdrücke A1, A2, A3.
`A1 TO A2`	im Bereich A1 bis A2.
`IS Operator A1`	Direkter Vergleich mit A1.

Beispiel

Das folgende Programm zeigt die Möglichkeit, verschiedene Bedingungen zu formulieren:

```
INPUT "Gib eine Zahl zwischen 1 und 100: ", I%
SELECT CASE I%
    CASE IS < 10, 77, 80 TO 89
        PRINT "I ist kleiner als 10, 77 oder zwischen 80 und 89"
    CASE 11,13,15,17,19
        PRINT "I ist zwischen 10 und 20 und gerade"
    CASE IS > 90
        PRINT "I ist größer als 90"
    CASE ELSE
        PRINT "I ist irgend etwas!"
```

```
END CASE
```

SGN()

Ermitteln des Vorzeichens eines Ausdrucks.

Verwendung

```
N = SGN (Ausdruck)
```

Parameter

`Ausdruck`	Der numerische Ausdruck, dessen Vorzeichen nach Berechnung ermittelt werden soll.

Returnwerte

`1`	Ausdruck > 0
`0`	Ausdruck = 0
`-1`	Ausdruck < 0

SHARED

Deklaration einer globalen Variablen innerhalb eines Unterprogramms.

Verwendung

```
SHARED Variable [AS Typ] [, Variable [AS Typ], ...]
```

Beschreibung

Diese Anweisung kann in einem Unterprogramm angegeben werden. Sie deklariert die angegebenen, bereits im Hauptprogramm deklarierten Variablen als innerhalb des Unterprogramms global.

Parameter

`Variable` — Name der zu deklarierenden Variable. Die Typkennung kann entfallen, wenn der Typ entweder durch eine DEFxxx-Anweisung oder durch die AS-Klausel vorgegeben ist. Arrays erwarten hinter dem Namen noch die Beschreibung der Dimensionen (s.u.).

`AS Typ` — Deklaration des Variablentyps. Neben den durch die TYPE- oder CLASS-Anweisung definierten Datentypen für Strukturen kann dies noch eines der folgenden Worte sein:

INTEGER	- kurze Integerzahl
LONG	- lange Integerzahl
SINGLE	- kurze Gleitpunktzahl
DOUBLE	- lange Gleitpunktzahl
STRING	- String variabler Länge
STRING*n	- String fester Länge (n)

Bei durch TYPE oder CLASS definierten Datentypen kann vor dem Typ auch eine der Klauseln POINTER TO oder PTR TO angegeben werden. Dies bewirkt, daß nicht eine Struktur, sondern nur ein Zeiger auf die Struktur definiert wird.

Anmerkungen

Ist die Variable mit der DIM...AS-Anweisung deklariert worden, muß die gleiche Deklaration in der `SHARED`-Anweisung erfolgen.

Die `SHARED`-Anweisung kann nur am Anfang eines Unterprogramms verwendet werden.

SIN()

Berechnen des Sinus.

Verwendung

```
N = SIN (Ausdruck)
```

Parameter

`Ausdruck`	Der numerische Ausdruck, dessen Sinus ermittelt werden soll.

Returnwert

Der Sinus des Ausdrucks.

Anmerkungen

Der Wert wird immer als lange Gleitpunktzahl berechnet, wenn der Parameter ebenfalls eine lange Gleitpunktzahl ist sonst ist das Ergebnis einfach genau.

Querverweise

ATN(), COS(), TAN()

SPACE$()

Erzeugen eines Strings aus Leerstellen.

Verwendung

```
S$ = SPACE$ (n)
```

Parameter

`n`	Anzahl der Leerstellen, die der String enthalten soll. Es kann ein String zwischen 0 und 32767 Zeichen Länge erzeugt werden.

Returnwert

Ein String, bestehend aus `n` Leerstellen.

Anmerkungen

Die Funktion STRING$() erzeugt Strings aus einem frei wählbaren Zeichen.

Querverweise

`STRING$()`

SPC()

Ausgabe von Leerstellen.

Verwendung

```
SPC (Anzahl)
```

Beschreibung

Die angegebene Anzahl von Leerstellen wird ausgegeben. Die Funktion darf nur als Parameter einer PRINT-Anweisung verwendet werden.

Parameter

`Anzahl`	Die Anzahl auszugebender Leerstellen.

Querverweise

TAB()

SQR()

Berechnen der Quadratwurzel eines Wertes.

Verwendung

```
N = SQR (Ausdruck)
```

Parameter

`Ausdruck`	Der numerische Ausdruck, dessen Wurzel ermittelt werden soll.

Returnwert

Die Quadratwurzel des Ausdrucks.

Anmerkungen

Der Wert wird immer als lange Gleitpunktzahl berechnet, wenn der Parameter ebenfalls eine lange Gleitpunktzahl ist sonst ist das Ergebnis einfach genau.

STATIC

Deklaration von lokalen statischen Variablen.

Verwendung

```
STATIC Variable [AS Typ] [, Variable [AS Typ], ...]
```

Beschreibung

Diese Anweisung kann in einem Unterprogramm angegeben werden. Sie deklariert die angegebenen Variablen als statisch und lokal. Die Variablen können nur innerhalb des Unterprogramms angesprochen werden im Gegensatz zu normalen lokalen Variablen behalten statische Variable jedoch ihren Wert bei, so daß er bei einem nachfolgenden Aufruf des Unterprogramms wieder zur Verfügung steht.

Parameter

`Variable` — Name der zu deklarierenden Variable. Die Typkennung kann entfallen, wenn der Typ entweder durch eine DEFxxx-Anweisung oder durch die AS-Klausel vorgegeben ist. Arrays erwarten hinter dem Namen noch die Beschreibung der Dimensionen (s.u.).

`AS Typ` — Deklaration des Variablentyps. Neben den durch die TYPE- oder CLASS-Anweisung definierten Datentypen für Strukturen kann dies noch eines der folgenden Worte sein:

INTEGER	- kurze Integerzahl
LONG	- lange Integerzahl
SINGLE	- kurze Gleitpunktzahl
DOUBLE	- lange Gleitpunktzahl
STRING	- String variabler Länge
STRING*n	- String fester Länge (n)

Bei durch TYPE oder CLASS definierten Datentypen kann vor dem Typ auch eine der Klauseln POINTER TO oder PTR TO angegeben werden. Dies bewirkt, daß nicht eine Struktur, sondern nur ein Zeiger auf die Struktur definiert wird.

Anmerkungen

Die `STATIC`-Anweisung kann nur am Anfang eines Unterprogramms verwendet werden.

STOP

Beenden des Programmlaufs.

Verwendung

```
STOP
```

Beschreibung

Der Programmlauf wird beendet. Alle noch offenen Dateien werden geschlossen.

Anmerkungen

Diese Anweisung ist ein Synonym für die Anweisungen END oder SYSTEM.

STR$()

Umwandeln einer Zahl in einen String.

Verwendung

```
S$ = STR$ (Ausdruck, [Maske])
```

Parameter

`Ausdruck`	Der numerische Ausdruck wird berechnet und in einen String umgewandelt.
`Maske`	Optional kann für die Umwandlung eine USING-Maske angegeben werden, wie sie bei den Anweisungen PRINT USING und PRINT # USING verwendet wird. Wenn die Zahl nicht in das `USING`-Feld paßt, wird im Gegensatz zur PRINT USING-Anweisung kein führendes Prozentzeichen ausgegeben.

Returnwerte

Der Wert des Ausdrucks in Form eines Strings.

Anmerkungen

Die Funktion VAL() ist das Gegenstück zu dieser Funktion. Sie konvertiert einen String in eine Zahl.

Bei der Konversion einer einfachgenauen Gleitpunktzahl können Rundungsfehler auftreten.

Beispiel

```
A# = 12345.678
S$ = STR$ (A#)                 ' S$ = "12345.678"
S$ = STR$ (A#, "####,.##")     ' S$ = "12.345,68"
```

Querverweise

VAL(), USING

STRING$()

Erzeugen eines Strings aus gleichen Zeichen.

Verwendung

```
S$ = STRING$ (Länge, Stringausdruck)
```

Parameter

`Länge`	Anzahl der Zeichen, die der String enthalten soll. Es kann ein String zwischen 0 und 32767 Zeichen Länge erzeugt werden.
`Stringausdruck`	Der String wird aus dem ersten Buchstaben des übergebenen Stringausdrucks erzeugt.

Returnwert

Ein String, bestehend aus Länge des angegebenen Zeichens.

Anmerkungen

Die Funktion SPACE$() erzeugt einen String aus Leerzeichen.

Querverweise

`SPACE$()`

SUB

Definition einer Prozedur.

Verwendung

```
SUB Name [(Parameter)] [STATIC | LOCAL]
    ....
END SUB
```

Beschreibung

Die Definition einer Prozedur wird mit dem Anweisungspaar `SUB/END SUB` durchgeführt. Nach der `SUB`-Anweisung folgt der Rumpf der Prozedur.

Im Allgemeinen wird der Rumpf einer Prozedur sequentiell abgearbeitet. Soll eine Prozedur vorzeitig verlassen werden, kann die EXIT SUB-Anweisung verwendet werden.

Parameter

`Name`	Name der Prozedur. Bei Methodendefinitionen setzt sich der Name aus dem Klassennamen, gefolgt von einem Punkt, sowie dem Namen der Methode zusammen.

`(Parameter)`	Die optionale Liste der formalen Parameter. Der Typ der formalen Parameter kann entweder über die Typkennung oder über die `AS`-Anweisung deklariert werden:
	`SUB PROZEDUR (I%, TEXT AS STRING*40)`
	Arrays werden dadurch kenntlich gemacht, daß dem Namen der Variablen ein Klammernpaar () nachgestellt wird:
	`SUB PROZEDUR (I%, A() AS STRING)`

`STATIC`	Der optionale Zusatz `STATIC` deklariert sämtliche Variable innerhalb der Prozedur als statisch. Sie werden zu Beginn der Laufzeit angelegt und behalten ihren Wert während des Programmlaufs bei.
`LOCAL`	Der optionale Zusatz `LOCAL` deklariert die Funktion als lokal. Eine lokale Funktion ist für andere Module unsichtbar.

Anmerkungen

Prozeduren sind rekursiv, d.h. eine Prozedur kann sich selbst entweder direkt oder indirekt über andere Prozeduren selbst aufrufen. Die Parameter werden *by reference* übergeben, d.h. die formalen Parameter werden beim Aufruf des Unterprogramms durch die aktuellen Parameter ersetzt. Wird der Inhalt eines formalen Parameters geändert, wird auch der Inhalt des aktuellen Parameters geändert.

Ein Parameter kann auch *by value* übergeben werden, wenn er geklammert wird. Dies veranlaßt den Interpreter, den Parameter als arithmetischen Ausdruck anzusehen, dessen Ergebnis (in diesem Fall nur der Parameter) in einer temporären Variablen abgelegt wird. Beispiel:

```
TEXT$ = "Hallo"
CALL PROZEDUR ((TEXT$))
```

Querverweise

DEF FN, FUNCTION, END SUB, EXIT SUB

SYSTEM

Beenden des Programmlaufs.

Verwendung

```
SYSTEM
```

Beschreibung

Der Programmlauf wird beendet. Alle noch offenen Dateien werden geschlossen.

Anmerkungen

Diese Anweisung ist ein Synonym für die Anweisungen END oder STOP.

TAB()

Vorrücken der Druckposition auf eine bestimmte Spalte.

Verwendung

```
TAB (Spalte)
```

Beschreibung

Die TAB-Funktion bewirkt, daß die nächste Ausgabe in der angegebenen Spalte erfolgt. Ist die angegebene Spalte bereits erreicht oder überschritten, erfolgt keine Bewegung. Die Ausgabespalte zählt ab 1.

Die Funktion kann nur als Parameter in einer PRINT-Anweisung verwendet werden.

Parameter

Spalte	Spalte, zu der vorgerückt werden soll. Die Spalte wird durch Ausgabe von Leerstellen erreicht. Dies bedeutet, daß eine tabellarische Ausgabe von Daten bei Verwendung von Proportionalschrift nur sehr eingeschränkt möglich ist.

Beispiel

Es sei eine Liste von Namen und Werten gegeben, die tabellarisch ausgegeben werden sollen:

```
DIM NAME$ (5), WERTE (5)
...
...
FOR I = 1 TO 5
    PRINT NAME$ (I), TAB (30), WERTE (I)
NEXT
I = REPLY : REM Auslösung der Ausgabe
```

Querverweise

SPC()

TAN()

Berechnung des Tangens.

Verwendung

```
N = TAN (Ausdruck)
```

Parameter

`Ausdruck`	Der numerische Ausdruck, dessen Tangens ermittelt werden soll.

Returnwert

Der Tangens des Ausdrucks.

Anmerkungen

Der Wertebereich des Parameters darf sich zwischen -Pi/2 und Pi/2 bewegen. Der Wert wird immer als lange Gleitpunktzahl berechnet, wenn der Parameter ebenfalls eine lange Gleitpunktzahl ist sonst ist das Ergebnis einfach genau.

Querverweise

SIN(), COS(), ATN()

THIS

Bezugnahme auf die eigene Instanz in einer Methode.

Beschreibung

Normalerweise werden Elemente einer Klasse innerhalb einer Methode direkt, d.h. ohne einen Strukturnamen angesprochen. Soll jedoch die gesamte Instanz referenziert werden, kann auf diese Instanz mit der Variable `THIS` Bezug genommen werden.

Beispiel

Dieses Beispiel ist etwas herbeigeholt, da die Routine `Drucken()` genausogut als Methode der Klasse hätte definiert werden können.

```
CLASS Beispiel
      A AS STRING*10
      B AS STRING*20
      PUBLIC SUB Ausgabe()
END CLASS

SUB Drucken (X AS Beispiel)
      PRINT X.A
      PRINT X.B
END SUB

SUB Beispiel.Ausgabe()
      CALL Drucken (THIS)
END SUB
```

Querverweise

CLASS, CONSTRUCTOR, DESTRUCTOR

TIME$()

Ermitteln der Uhrzeit.

Verwendung

```
S$ = TIME$
```

Returnwerte

Die Uhrzeit in der Form "HH:MM:SS", wobei "HH" die Stunden im Bereich 0 bis 23, "MM" die Minuten im Bereich 0 bis 59 und "SS" die Sekunden im Bereich 0 bis 59 darstellen.

Anmerkungen

Mit der Funktion DATE$() kann das Tagesdatum ermittelt werden. Die Uhrzeit ist nicht änderbar.

Querverweise

DATE$()

TIMER

Steuerung des Zeitgebers.

Verwendung

```
TIMER [ON | OFF | STOP]
```

Beschreibung

Wird durch die Anweisung ON TIMER (n) GOSUB ein Zeitgeber aktiviert, kann er durch diese Anweisung gesteuert werden.

Parameter

`ON`	Der Zeitgeber wird aktiviert. Ist eine voreingestellte Zeit bereits abgelaufen, wird sofort zu dem in der ON TIMER (n) GOSUB- Anweisung angegebenen Label verzweigt.
`OFF`	Der Zeitgeber wird deaktiviert. Läuft die durch die `ON`-Anweisung eingestellte Zeit ab, wird die Tatsache jedoch zwischengespeichert, so daß die Anweisung `TIMER ON` sofort zum Ansprung der durch die `ON`-Anweisung definierte Routine führt.
`STOP`	Der Zeitgeber wird deaktiviert. Läuft die durch die `ON`-Anweisung eingestellte Zeit ab, wird der Zeitgeber ignoriert.

Querverweise

`ON TIMER (n) GOSUB`, `TIMER()`

TIMER()

Rückgabe der abgelaufenen Tageszeit in Sekunden.

Verwendung

```
A = TIMER
```

Beschreibung

Die seit Mitternacht abgelaufene Zeit in Sekunden wird zurückgeliefert.

Anmerkungen

Die Anweisung TIMER schaltet einen Zeitgeber ein oder aus.

Beispiel

```
PRINT "Sekunden seit Mitternacht:" TIMER
```

Querverweise

TIMER, ON TIMER (N) GOSUB

TYPE

Deklaration von Strukturen.

Verwendung

```
TYPE Datentypname
    Variable AS [POINTER|PTR TO] Datentyp
    ....
END TYPE
```

Beschreibung

Diese Anweisung deklariert einen neuen, zusammengesetzten Datentyp. Die einzelnen Variablen der Struktur werden innerhalb der `TYPE...END TYPE`-Anweisung deklariert. Es können weder Arrays noch Strings variabler Länge deklariert werden die Verwendung von Strukturen innerhalb von Strukturen ist jedoch möglich. Die Deklaration von Variablen dieses Datentyps erfolgt mit Hilfe der DIM-Anweisung. Der Zugriff auf einzelne Elemente einer solchen Variablen erfolgt, indem dem Variablennamen erste ein Punkt und dann der Name des Elements nachgestellt wird.

Wird dem Datentyp die Klausel POINTER TO bzw. PTR TO vorangestellt, was nur bei durch mit `TYPE` oder CLASS deklarierten Datentypen möglich ist, wird keine Struktur, sondern nur ein Zeiger darauf eingetragen.

Beispiel

Deklaration eines Personen-Stammsatzes. Das Geburtsdatum sowie die Adresse sind selbst Strukturen. Anschließend wird eine Variable `Leute()` deklariert und mit Daten versehen.

```
TYPE Datum                        ' Datum:
     Tag   AS INTEGER
     Monat AS INTEGER
     Jahr  AS INTEGER
END  TYPE

TYPE Adresse                      ' Wohnort:
     Strasse AS STRING*40
     PLZ     AS INTEGER
     Ort     AS STRING*20
END  TYPE

TYPE Stammsatz                    ' Stammsatz:
     Nachname AS STRING*20
     Vorname  AS STRING*20
     Geboren  AS Datum
     Wohnhaft AS Adresse
END  TYPE

DIM Meyer AS Stammsatz, Leute (5) AS Stammsatz
Meyer.Nachname         = "Meyer"
Meyer.Vorname          = "Herbert"
Meyer.Geboren.Tag      = 25
Meyer.Geboren.Monat    = 10
Meyer.Geboren.Jahr     = 1953
Meyer.Wohnhaft.Strasse = Feldweg 53"
Meyer.Wohnhaft.PLZ     = 2120
```

```
Meyer.Wohnhaft.Ort      = "Lüneburg"
```

UBOUND()

Ermitteln der oberen Indexgrenze einer Array-Dimension.

Verwendung

```
N = UBOUND (Arrayname [, Dimension])
```

Beschreibung

Der höchste zulässige Wert für die angegebene Array-Dimension wird zurückgeliefert.

Parameter

`Arrayname`	Der Name des Arrays.
`Dimension`	Optional kann eine Dimension angegeben werden. Wird keine Angabe gemacht, wird der höchste zulässige Wert für die erste Dimension zurückgeliefert. Der Parameter darf Werte zwischen 1 und der für das Array definierten Anzahl Dimensionen annehmen.

Anmerkungen

Die Funktion LBOUND() liefert den niedrigsten zulässigen Wert für eine Array-Dimension zurück.

Beispiel

```
DIM TEST (5, 10, 15)

FOR I% = 1 TO 3
    PRINT "TEST Dimension " I "=" UBOUND (TEST, I)
NEXT
```

Querverweise

LBOUND()

UCASE$()

Konvertierung eines Strings in Großbuchstaben.

Verwendung

```
S$ = UCASE$ (Stringausdruck)
```

Parameter

`Stringausdruck`	Der Stringausdruck wird berechnet und der daraus resultierende String in Großbuchstaben umgewandelt.

Returnwerte

Der in Großbuchstaben konvertierte String.

Anmerkungen

Die Funktion LCASE$() konvertiert einen String in Kleinbuchstaben.

Querverweise

LCASE$()

UNLOCK

Freigabe einer zuvor mit LOCK gesperrten Datei.

Verwendung

```
UNLOCK Kanalnummer
```

Beschreibung

Eine Datei kann gegen Zugriffe von anderen Applikationen aus mit Hilfe der LOCK-Anweisung gesperrt werden. Solange die Sperre aktiv ist, kann keine andere Applikation auf die Datei zugreifen. Wird dieser Zugriff versucht, bleibt die andere Applikation so lange blockiert, bis die Datei wieder mit dieser Anweisung freigegeben wird.

Parameter

`Kanalnummer`	Die Kanalnummer der anzusprechenden Datei. Dies ist ein Integer-Ausdruck. Vor dem Ausdruck muß ein Doppelkreuz (#) stehen, um den Ausdruck als Kanalnummer zu kennzeichnen.

Anmerkungen

Diese Funktion ist abhängig vom Betriebssystem. Unter DOS wird beispielsweise nach einer gewissen Zeit mit einem Fehlercode abgebrochen.

Querverweise

LOCK

USING

Formatieren von Druckausgaben.

Verwendung

```
PRINT USING "Formatmaske"; Daten, ...
```

Beschreibung

Die PRINT-Anweisung kann um eine Formatspezifikation erweitert werden, mit der Zahlen vor der Ausgabe formatiert werden können. Die erweiterte `PRINT USING`-Anweisung hat als ersten Parameter einen Formatstring. Dies ist ein String, der das numerische Format der auszugebenden Zahlen bzw. das Format eines Strings beschreibt. Ein Beispiel:

```
I = 123.456
PRINT USING "####.##", I
```

hätte die Ausgabe der Variablen `I` als "123.46" (gerundet) zur Folge.

Der Formatstring wird sequentiell abgearbeitet. Ist er zu Ende, ehe alle angegebenen Variablen ausgegeben worden sind, beginnt die Abarbeitung des Strings von vorne. Somit ist es beispielsweise unnötig, eine Formatangabe für mehrere Zahlen einer Zeile zu machen, wenn alle Zahlen im selben Format ausgegeben werden sollen. Beispiel:

```
I = 1 : J = 3 : K = 4
PRINT USING "#.##   " I, J, K
```

Der Ausdruck hätte das Format

```
1.00   3.00   4.00
```

Andere Zeichen als die in den nachfolgend angegebenen Formatzeichen werden direkt ausgegeben.

Stringformatierung

Für die Formatierung von Strings stehen folgende Möglichkeiten zur Verfügung:

**\ **

Ausgabe des Strings in einer bestimmten Länge. Die Feldlänge entspricht der Anzahl der Leerzeichen zwischen den Schrägstrichen plus die beiden Schrägstriche selbst. Die Angabe von nur zwei Schrägstrichen entspricht also einer Feldlänge von 2, die Schrägstriche plus einer Leerstelle einer Feldlänge von 3 usw. Ist der String länger als die Feldlänge wird er abgeschnitten ist er kürzer, wird rechts mit Leerstellen aufgefüllt. Beispiel:

```
a$ = "Hallo, Leute"
PRINT USING "\   \" A$
```

ergibt die Ausgabe `Hallo`

!

Nur das erste Zeichen des Strings wird ausgegeben.

&

Der String wird unformatiert ausgegeben.

Numerische Formatierung

Für numerische Werte stehen folgende Zeichen zur Verfügung. es ist dabei zu beachten, daß jedes Formatzeichen zur Berechnung der Feldweite mit herangezogen wird so vergrößert sich die Feldweite beispielsweise bei der Angabe des Formatzeichens "**" um zwei Stellen.

Darstellung einer Ziffer. Ist die Zahl der Ziffern kleiner als die Anzahl der Doppelkreuze, wird die Zahl im Feld rechtsbündig mit führenden Leerstellen ausgegeben. Ist die Anzahl der Ziffern größer, wird die Zahl gerundet. Wird die Zahl in exponentieller Notation ausgegeben, wird immer nur eine Ziffer vor dem Dezimalpunkt ausgegeben.

. Der Punkt gibt die Lage des Dezimalpunktes an. Er wird abhängig von der Landessprache entweder als Punkt oder als Komma dargestellt.

, Das Komma darf vor dem Punkt als Formatzeichen gesetzt werden. Es geht in die Feldweitenberechnung mit ein. Es zeigt an, daß 1000er-Punkte mit ausgegeben werden sollen. Weiter werden auch die nationalen Zeichen für Dezimalpunkt und Komma genommen, so daß der Dezimalpunkt z.B. in Deutschland ein Komma ist. Beispiel:

```
A = 12345.67
PRINT USING "#######,.## DM" A
```

ergibt die Ausgabe `12.345,67 DM`

, Das Komma wird abhängig von der Landessprache entweder als Komma oder als Punkt dargestellt.

+ Darf nur als erstes Formatzeichen verwendet werden. Gibt das Vorzeichen der Zahl unmittelbar vor der Zahl aus. Beispiel:

```
A = 12345.67
PRINT USING "+#####.##" A
```

ergibt die Ausgabe `+12345,67`

- Darf nur als letztes Formatzeichen verwendet werden. Gibt, falls die Zahl negativ ist, ein Minuszeichen hinter der Zahl aus, sonst eine Leerstelle. Beispiel:

```
A = -12345.67
PRINT USING "#####.##-" A
```

ergibt die Ausgabe `12345,67-`

****** Führende Leerstellen werden als Sternchen dargestellt. Beispiel:

```
A = 12345.67
PRINT USING "*#######.## DM" A
```

ergibt die Ausgabe `***12345,67 DM`

$$ Vor der Zahl erscheint ein Dollarzeichen. Beispiel:

```
A = 12345.67
PRINT USING "$#######.## DM" A
```

ergibt die Ausgabe `$12345,67 DM`

****$** Führende Leerstellen werden als Sternchen dargestellt. Unmittelbar vor der Zahl wird ein Dollarzeichen ausgegeben. Beispiel:

```
A = 12345.67
PRINT USING "**$#######.## DM" A
```

ergibt die Ausgabe `****$12345,67 DM`

^^^^ Angabe der Position des Exponenten. Für dreistellige Exponenten können auch fünf Zeichen angegeben werden. Beispiel:

```
A = 12345.67
PRINT USING "#.###^^^^^" A
```

ergibt die Ausgabe `1,235E+004`

_ Der Unterstrich führt dazu, daß das folgende Zeichen nicht als Formatzeichen interpretiert, sondern direkt ausgegeben wird. Ein Unterstrich ist durch zwei Unterstriche anzugeben.

% Ist die Zahl im Feld nicht darstellbar, wird die Zahl ohne Rücksicht auf die Feldgrenzen mit einem vorangehenden Prozentzeichen ausgegeben.

Dialogformate

Fensterorientierte Dialoge können durch die Angabe von speziellen Formatzeichen optisch beeinflußt werden. Ist das erste Zeichen eines Formatstrings eine Tilde (~), wird die auszugebende Zeile als Titelzeile des Fensters übernommen. Unmittelbar nach der Tilde dürfen noch folgende Formatzeichen erscheinen:

?	Ausgabe eines Frage-Icons.
W	Ausgabe eines Warn-Icons.
E	Ausgabe eines Fehler-Icons.

Die Button-Kombinationen des Fensters können wie folgt beeinflußt werden:

1	OK
2	OK, Abbruch
3	Wiederholen, Abbruch
4	Ja, Nein
5	Ja, Nein, Abbruch

Buttons

Die mit der obigen Formatanweisung definierten Buttons werden nur in Verbindung mit der Funktion REPLY() ausgegeben. Diese Funktion zeigt den bislang aufgelaufenen Text in Form einer Message Box an und liefert den Wert des aktivierten Buttons als Funktionsergebnis zurück. Die INPUT-Anweisung zeigt den Text zusammen mit einem Edit-Feld und den Buttons OK und Abbruch an. Dort wird das

Buttonformat nicht ausgewertet; der Abbruch-Button bewirkt das Ende des Programmlaufs.

Die Werte der Buttons werden wie folgt zurückgemeldet:

-1	OK	(entspricht TRUE)
0	Abbruch	(entspricht FALSE)
1	Ja	
2	Nein	
3	Wiederholen	

Beispiele für Dialoge

Beispiel 1: ein einfacher Dialog erscheint wie folgt:

```
Datei$ = "TEST.DAT"
print using "~?4&" "Datei sichern"
print "Soll die Datei " Datei$ " gesichert werden?"
```

Beispiel 2: Eine INPUT-Anweisung mit 2 Zeilen und einem Prompt:

```
print using "~&" "Persönliche Daten"
print "Es sollen Ihre persönliche Daten erfaßt werden."
print "Dafür müssen einige Angaben gemacht werden."
input "Bitte geben Sie Vor- und Nachnamen ein:", VN$, NN$
```

Beispiel 3: Die Ausgabe eines Textes, der länger ist als 2 Zeilen. Dabei wird auch ein Info-Icon mit ausgegeben:

```
print using "~W&" "Testausgabe 10 Zeilen mit Info-Icon"
for i% = 1 to 10
    print "Dies ist Zeile" i%
next
line input "Geben Sie irgend etwas zur Bestätigung ein:" l$
```

VAL()

Konversion eines Strings in eine Zahl.

Verwendung

```
N = VAL (Stringausdruck)
```

Parameter

`Stringausdruck`	Der angegebene Stringausdruck wird berechnet. Anschließend wird versucht, die im String enthaltenen Zeichen als Zahl zu interpretieren, wobei führende Leerstellen ignoriert werden.

Returnwert

Die konvertierte Zahl. Konnte keine Zahl erkannt werden, ist der Wert 0.

Anmerkungen

Die Funktion STR$() konvertiert eine Zahl in einen String.

VARPTR()

Ermitteln der Adresse einer Variablen.

Verwendung

```
N = VARPTR (Variable)
```

Beschreibung

Die Adresse einer Variablen wird ermittelt und zurückgeliefert. Diese Adresse bleibt während der Laufzeit des Programms konstant eine Ausnahme bilden Strings, deren Adresse sich verschieben kann. Von Interesse ist diese Funktion in Zusammenhang mit Pointern, um die Adresse einer Struktur zu erhalten.

Parameter

`Variable`	Eine beliebige Variable.

Returnwert

Die Adresse der Variablen. Es wird immer die gesamte 32-Bit-Adresse zurückgeliefert.

WHILE...WEND

Ausführen eines Anweisungsblock, bis eine Bedingung FALSCH ist.

Verwendung

```
WHILE Ausdruck
    ....
    ....
WEND
```

Beschreibung

Der arithmetische Ausdruck wird berechnet. Hat er einen Wert ungleich Null, werden die Anweisungen bis zur WEND-Anweisung durchlaufen. Dann wird der Wert des Ausdrucks erneut berechnet. Dieser Vorgang wird so lange wiederholt, bis der Wert des Ausdrucks Null ergibt. In diesem Fall wird das Programm bei der nächsten Anweisung hinter der WEND-Anweisung fortgesetzt.

Die Vereinfachung der WHILE-Anweisung wäre:

```
Schleife:
    IF Ausdruck = 0 THEN GOTO Weiter
        ....
        ....
    GOTO Schleife
Weiter:
    ....
```

Parameter

Ausdruck	Der arithmetische Ausdruck, dessen Wert überprüft wird.

Anmerkungen

Mehrere WHILE-Anweisungen können ineinander verschachtelt werden. In diesem Fall beendet jede WEND-Anweisung die entsprechende WHILE-Anweisung.

Die Anweisung DO...LOOP ist wesentlich mächtiger als die WHILE...WEND-Anweisung und daher vorzuziehen.

Beispiel

Das folgende Beispiel entfernt führende Leerstellen in einem String:

```
A$ = "   Hallo"

WHILE LEFT$ (A$, 1) = " "
    A$ = MID$ (A$, 2)
WEND
```

WRITE

Ausgabe von Daten auf dem Bildschirm.

Verwendung

```
WRITE
WRITE Ausdruck [, Ausdruck ...]
```

Beschreibung

Die angegebenen Ausdrücke werden auf dem Bildschirm ausgegeben.

Wird die Anweisung ohne Parameter verwendet, erfolgt ein Zeilenvorschub.

Parameter

`Ausdruck`	Die Liste der auszugebenden Ausdrücke. Jeder Ausdruck wird ausgerechnet und ausgegeben. Zwischen die einzelnen Ausdrücke werden Kommata gesetzt. Strings werden in Anführungszeichen eingeschlossen.

Anmerkungen

Im Gegensatz zur PRINT-Anweisung erfolgt die Ausgabe so, daß die Daten mit der INPUT-Anweisung wieder eingelesen werden können.

Beispiel

```
NAME$ = "Meyer"
I     = 10
J     = 20.5
WRITE I, J, NAME$
PRINT I, J, NAME$
INPUT ANTWORT
```

Querverweise

PRINT

WRITE

Ausgabe von Daten in eine ASCII-Datei.

Verwendung

```
WRITE #Kanal
WRITE #Kanal, Ausdruck [, Ausdruck ...]
```

Beschreibung

Die angegebenen Ausdrücke werden in die durch die Kanalnummer bezeichnete Datei ausgegeben.

Wird die Anweisung nur mit Kanalnummer ohne Parameter verwendet, erfolgt ein Zeilenvorschub.

Parameter

`Kanal`	Die Kanalnummer der anzusprechenden Datei. Dies ist ein Integer-Ausdruck. Vor dem Ausdruck muß ein Doppelkreuz (#) stehen, um den Ausdruck als Kanalnummer zu kennzeichnen.
`Ausdruck`	Die Liste der auszugebenden Ausdrücke. Jeder Ausdruck wird ausgerechnet und ausgegeben. Zwischen die einzelnen Ausdrücke werden Kommata gesetzt. Strings werden in Anführungszeichen eingeschlossen.

Anmerkungen

Im Gegensatz zur PRINT #-Anweisung erfolgt die Ausgabe so, daß die Daten mit der INPUT #-Anweisung wieder eingelesen werden können.

Querverweise

PRINT #

Klassendefinitionen

Bitmap CheckBox ComboBox Control Dialog DialogBase EditField FixedText
GroupBox Icon ListBox PopupMenu PushButton RadioButton ScrollBar

CLASS DialogBase

```
CLASS DialogBase
   Internal AS STRING*4
PROTECTED
   SUB Get CDECL()
   SUB Set CDECL()
PUBLIC
   X0    AS INTEGER
   Y0    AS INTEGER
   NX    AS INTEGER
   NY    AS INTEGER
   SUB      SetText CDECL (T$)
   FUNCTION GetText CDECL () AS STRING
END CLASS
```

Beschreibung

Die Basisklasse des Dialogsystems stellt die Funktionalität zur Verfügung, die von allen Dialogelementen benötigt wird. Von dieser Klasse werden nie direkt Instanzen erzeugt.

Konstante

FALSE = 0	Die Konstante FALSE wird bei logischen Zustandsänderungen verwendet.
TRUE = NOT FALSE	Die Konstante TRUE (-1) wird bei logischen Zustandsänderungen verwendet.
IsAuto = -32767	Diese Konstante veranlaßt die automatische Berechnung von Lage und Größe des Elements.

Variable

Internal AS STRING*4	Ein Pointer auf die interne C++-Repräsentation der Klasse.
X0, Y0 AS INTEGER	Die Start-Koordinaten des Elements. Diese Werte werden relativ zum Parent gewertet. Wird einer der Werte mit der Konstante *IsAuto* besetzt, wird das Element in der entsprechenden Richtung relativ zum Parent zentriert.
NX, NY AS INTEGER	Die Größe des Elements. Beim Dialog selbst ist dies die Größe des Arbeitsbereichs ohne Rand und Titel. Wird einer der Werte mit der Konstante *IsAuto* besetzt, erfolgt die Berechnung der Elementgröße abhängig vom Typ des Elements.

Methoden

SUB GET()	Diese interne Methode wird von StarBASIC vor jedem Zugriff auf eine Variable aufgerufen. Dadurch stehen in den Variablen immer die aktuellen Werte des Elements zur Verfügung. Diese Methode ist intern auf alle Klassen korrekt abgebildet, so daß die abgeleiteten Klassen keine eigenen GET()-Methoden haben müssen.
SUB SET()	Diese interne Methode wird von StarBASIC nach jeder Zuweisung

	eines Wertes auf eine Variable aufgerufen. Dadurch wird jede Änderung sofort auf das Element abgebildet. Diese Methode ist intern auf alle Klassen korrekt abgebildet, so daß die abgeleiteten Klassen keine eigenen GET()-Methoden haben müssen.
FUNCTION GETTEXT() AS STRING	Es wird der Text ausgelesen, mit dem das Element belegt ist. Die Art des Textes ist dabei vom Typ des Elements abhängig.
SUB SETTEXT (TEXT$)	Der Text des Elements wird neu definiert. Wo und wie der Text erscheint, hängt vom Typ des Elements ab.

CLASS Control

```
CLASS Control PUBLIC DialogBase
PUBLIC
   Value   AS INTEGER
   Visible AS INTEGER
   Enabled AS INTEGER
END CLASS
```

Beschreibung

Dies ist die Basisklasse für alle Dialogelemente, die Events erzeugen. Die einzige Klasse, die nicht von dieser Klasse abgeleitet ist, ist die Klasse *Dialog*.

Konstante

HasBorder = 1	Dieses Flag-Bit kann für alle Elemente verwendet werden. Es sorgt für eine Umrandung des Elements.

Variable

Value AS INTEGER	Der Wert des Elements hängt stark vom Typ des Elements ab. Er spiegelt den Status des Elements wieder.
Visible AS INTEGER	Wird dieses Feld mit dem Wert 0 (der Konstanten FALSE) belegt, wird das Element unsichtbar. Ein anderer Wert als 0 (beispielsweise die Konstante TRUE) macht das Element sichtbar. Das Element wird mit dem Wert TRUE initialisiert.
Enabled AS INTEGER	Wird dieses Feld mit dem Wert 0 (der Konstanten FALSE) belegt, kann das Element nicht mehr angewählt werden. Ein anderer Wert als 0 (beispielsweise die Konstante TRUE) macht das Element wider anwählbar. Das Element wird mit dem Wert TRUE initialisiert.

CLASS Dialog

```
CLASS Dialog PUBLIC DialogBase
END CLASS
```

Beschreibung

Diese Klasse instanziiert den StarBASIC-Dialog. Sie hat keine eigene Funktionalität. Die initialisierende Routine ist DIALOG.

Sonderverhalten Variable

X0, Y0 AS INTEGER	Die Start-Koordinaten des Dialogs. Diese Werte werden relativ zum Parent gewertet. Wird einer der Werte mit der Konstante *IsAuto* besetzt, wird der Dialog in der entsprechenden Richtung relativ zum Parent zentriert. Kann kein Parent ermittelt werden, wird der Dialog in der Bildschirmmitte eingerichtet.
NX, NY AS INTEGER	Die Größe des Arbeitsbereichs des Dialogs. Wird einer der Werte mit der Konstante *IsAuto* besetzt, erhält der Dialog eine Standardgröße von 100x100 Einheiten.

Sonderverhalten Methoden

FUNCTION GETTEXT() AS STRING	Es wird die Überschrift des Dialogs zurückgeliefert.
SUB SETTEXT (TEXT$)	Die Überschrift des Dialogs wird neu definiert.

Beispiel

Das folgende Beispiel zeigt eine kurze Meldung mit einem Icon und einem OK-Button an.

```
' $INCLUDE: 'DIALOG.INC'

DIM Dlg AS Dialog
DIM Ok AS PushButton
DIM Info AS Icon
DIM Text AS FixedText

DIALOG         Dlg,      20,  20,140, 60, "Meldung"
  FIXEDTEXT    TEXT,     40,  10, 90, 16, "", IsCenter
  ICON         Info,     10,  15, InfoIcon
  PUSHBUTTON   Ok,       50,  35, 40, 14, "OK", IsDefault

  A$ = "Dies ist ein kleiner Text, der ausgegeben wird."
  TEXT.SetText (A$)
  ON CLICK (Ok) GOSUB Done
  STARTDIALOG
  END

Done:
  ENDDIALOG
  RETURN
```

CLASS Bitmap

```
CLASS Bitmap PUBLIC Control
PUBLIC
   SUB Load CDECL (FILENAME$)
END CLASS
```

Beschreibung

Diese Klasse instanziiert eine Bitmap. Die initialisierende Routine ist BITMAP. Die Bitmap wird aus einer Datei geladen.

Methoden

SUB LOAD (FILENAME$)	Mit dieser Methode kann in das bestehende Element eine neue Bitmap geladen werden. Dies ermöglicht Animationseffekte etc.

Sonderverhalten Variable

NX, NY AS INTEGER	Die Größe der Bitmap. Diese beiden Variable werden mit der in der Datei angegebenen Größe vorbesetzt. Die Änderung dieser Variablen dehnt die Bitmap entsprechend der neuen Größe.
Value AS INTEGER	Diese Variable hat immer den Wert 0. Sie wird nicht interpretiert.

Sonderverhalten Methoden

FUNCTION GETTEXT() AS STRING	Ein leerer Text wird zurückgeliefert.
SUB SETTEXT (TEXT$)	Diese Methode hat keine Wirkung.

Events

ON CLICK (Variable) GOSUB	Wird das Dialogelement angeklickt, wird zum angegebenen GOSUB-Label verzweigt.

Beispiel

Siehe Beispiel bei der Klasse *Icon*.

CLASS CheckBox

```
CLASS CheckBox PUBLIC Control
END CLASS
```

Beschreibung

Diese Klasse instanziiert eine CheckBox. Sie hat keine eigene Funktionalität. Die initialisierende Routine ist CHECKBOX.

Konstante

IsTriState = 8	Dieses Flag-Bit definiert die Checkbox als Tristate-Box.
TriState = 1	Der dritte mögliche Returnwert neben TRUE und FALSE.

Sonderverhalten Variable

NX, NY AS INTEGER	Wird eine der beiden Variablen mit dem Wert *IsAuto* belegt, wird der vorgegebene Text zur Größenberechnung hinzugezogen. Das Element wird so berechnet, daß um den Text ein Rand von 2 Einheiten verbleibt.
Value AS INTEGER	Hier wird der aktuelle Zustand der Checkbox abgelegt (TRUE oder FALSE). Bei Tristate-Boxen kann auch der Wert *TriState* (1) erscheinen. Das Ändern des Wertes veranlaßt eine Statusänderung des Elements.

Sonderverhalten Methoden

FUNCTION GETTEXT() AS STRING	Der Text, der neben der Checkbox erscheint, wird zurückgeliefert.
SUB SETTEXT (TEXT$)	Der Text, der neben der Checkbox erscheint, wird neu definiert.

Events

ON ENTER (Variable) GOSUB	Was das Dialogelement inaktiv und wird es angeklickt oder für Tastatureingaben aktiviert, wird auf den angegebenen GOSUB-Label verzweigt.
ON LEAVE (Variable) GOSUB	War das Dialogelement aktiv und es wird ein anderes Element aktiviert, wird auf den angegebenen GOSUB-Label verzweigt.
ON CLICK (Variable) GOSUB	Wird das Dialogelement angeklickt oder es wird die Leertaste gedrückt, wenn das Element aktiv ist, wird zum angegebenen GOSUB-Label verzweigt.

Beispiel

Das folgende Beispielsprogramm simuliert das Verhalten von Radiobuttons in einem Dialog.

```
REM $INCLUDE: 'DIALOG.INC'
DIM Dlg AS Dialog
DIM Ok AS PushButton, Cancel AS PushButton
```

```
DIM Check1 AS CheckBox, Check2 AS CheckBox, Check3 AS CheckBox

DIALOG           Dlg, IsAuto,IsAuto,165, 63, "Test Check-Boxes"
  CHECKBOX       Check1,    9,    6, 95, 14, "Auswahl 1"
  CHECKBOX       Check2,    9,   20, 95, 14, "Auswahl 2"
  CHECKBOX       Check3,    9,   36, 95, 14, "Auswahl 3"
  PUSHBUTTON     Ok,      115,    6, 45, 14, "OK", IsDefault
  PUSHBUTTON     Cancel,  115,   26, 45, 14, "Abbrechen", IsCancel

ON CLICK (Ok)     GOSUB Ok
ON CLICK (Cancel) GOSUB Cancel
ON CLICK (Check1) GOSUB Auswahl1
ON CLICK (Check2) GOSUB Auswahl2
ON CLICK (Check3) GOSUB Auswahl3

Res$ = Check1.GetText() : Check1.Value = TRUE
STARTDIALOG
PRINT Res$ : END

Auswahl1:
   IF Check1.Value THEN
      Check2.Value = FALSE
      Check3.Value = FALSE
      Res$ = Check1.GetText()
   ELSE
      Res$ = "Kein Ergebnis"
   END IF
   RETURN
Auswahl2:
   IF Check2.Value THEN
      Check1.Value = FALSE
      Check3.Value = FALSE
      Res$ = Check2.GetText()
   ELSE
      Res$ = "Kein Ergebnis"
   END IF
   RETURN
Auswahl3:
   IF Check3.Value THEN
      Check1.Value = FALSE
      Check2.Value = FALSE
      Res$ = Check3.GetText()
   ELSE
      Res$ = "Kein Ergebnis"
   END IF
   RETURN
Ok:
   ENDDIALOG : RETURN
Cancel:
   Res$ = "Abbruch"
   ENDDIALOG : RETURN
```

CLASS EditField

```
CLASS EditField PUBLIC Control
END CLASS
```

Beschreibung

Diese Klasse instanziiert ein ein- oder mehrzeiliges Eingabefeld. Sie hat keine eigene Funktionalität. Die initialisierende Routine ist EDITFIELD.

Konstante

IsLeft = 2	Der Text wird linksbündig ausgegeben.
IsCenter = 4	Der Text wird zentriert ausgegeben.
IsRight = 8	Der Text wird rechtsbündig ausgegeben.
IsMultiLine = 16	Das Eingabefeld ist mehrzeilig. Der Zeilentrenner ist das CRLF.
HasVScroll = 32	Das Eingabefeld erhält einen vertikalen Scrollbar (nur mehrzeilige Felder).
HasHScroll = 64	Das Eingabefeld erhält einen horizontalen Scrollbar (nur mehrzeilige Felder).

Sonderverhalten Variable

NX, NY AS INTEGER	Wird eine der beiden Variablen mit dem Wert *IsAuto* belegt, wird der vorgegebene Text zur Größenberechnung hinzugezogen. Das Element wird so berechnet, daß um den Text ein Rand von 2 Einheiten verbleibt.
Value AS INTEGER	Diese Variable ist ungleich 0 (TRUE), wenn Änderungen am Feld vorgenommen worden sind. Wird auf diese Variable ein Wert zugewiesen, wird er in das Editfeld übernommen, so daß die Anzeige der Änderungen auch zurückgenommern werden kann.

Sonderverhalten Methoden

FUNCTION GETTEXT() AS STRING	Der Text wird zurückgeliefert. Er kann bei mehrzeiligen Feldern auch CRLF-Sequenzen als Zeilentrenner enthalten.
SUB SETTEXT (TEXT$)	Der Text wird neu definiert.

Events

ON ENTER (Variable) GOSUB	Was das Dialogelement inaktiv und wird es angeklickt oder für Tastatureingaben aktiviert, wird auf den angegebenen GOSUB-Label verzweigt.
ON LEAVE (Variable) GOSUB	War das Dialogelement aktiv und es wird ein anderes Element aktiviert, wird auf den angegebenen GOSUB-Label verzweigt.

ON CHANGE (Variable) GOSUB

Nach jeder Änderung des Inhalts des Eingabefelds wird auf den angegebenen GOSUB-Label verzweigt.

ON KEY (Variable) GOSUB Bei einer Tastatureingabe wird zum angegebenen GOSUB-Label verzweigt. Das *Value*-Feld enthält den Code der Taste. Die Taste kann ignoriert werden, indem das Feld *Value* auf 0 gesetzt wird.

Beispiel

```
REM $INCLUDE: 'DIALOG.INC'

DIM Dlg AS Dialog
DIM Ok AS PushButton, Cancel AS PushButton
DIM Test AS EditField

DIALOG           Dlg,  IsAuto,IsAuto,165, 63, "Test Edit-Feld"
  EDITFIELD      Test,      9,     6, 95, 45, "Überschreibe mich!",
IsMultiLine+HasBorder+HasVScroll+HasHScroll
  PUSHBUTTON     Ok,      115,     6, 45, 14, "OK", IsDefault
  PUSHBUTTON     Cancel,  115,    26, 45, 14, "Abbrechen", IsCancel

ON CLICK (Ok)     GOSUB Ok
ON CLICK (Cancel) GOSUB Cancel

Res$ = "Kein Ergebnis"
STARTDIALOG
PRINT Res$
END

Ok:
   Res$ = Test.GetText()
   ENDDIALOG
   RETURN

Cancel:
   Res$ = "Abbruch"
   ENDDIALOG
   RETURN
```

CLASS FixedText

```
CLASS FixedText PUBLIC Control
END CLASS
```

Beschreibung

Diese Klasse instanziiert ein Textelement. Sie hat keine eigene Funktionalität. Die initialisierende Routine ist FIXEDTEXT.

Konstante

IsLeft = 2	Der Text wird linksbündig ausgegeben.
IsCenter = 4	Der Text wird zentriert ausgegeben.
IsRight = 8	Der Text wird rechtsbündig ausgegeben.

Sonderverhalten Variable

NX, NY AS INTEGER	Wird eine der beiden Variablen mit dem Wert *IsAuto* belegt, wird der vorgegebene Text zur Größenberechnung hinzugezogen. Das Element wird so berechnet, daß um den Text ein Rand von 2 Einheiten verbleibt.
Value AS INTEGER	Diese Variable hat immer den Wert 0. Sie wird nicht interpretiert.

Sonderverhalten Methoden

FUNCTION GETTEXT() AS STRING	Der Text wird zurückgeliefert.
SUB SETTEXT (TEXT$)	Der Text wird neu definiert.

Events

ON CLICK (Variable) GOSUB	Wird das Dialogelement angeklickt, wird zum angegebenen GOSUB-Label verzweigt.

Beispiel

Siehe Beispiel bei der Klasse *Icon*.

CLASS GroupBox

```
CLASS GroupBox PUBLIC Control
END CLASS
```

Beschreibung

Diese Klasse instanziiert ein Gruppenfeld. Sie hat keine eigene Funktionalität. Die initialisierende Routine ist GROUPBOX. Das Ende einer Gruppe wird mit der ENDGROUP-Anweisung angezeigt. Ein Gruppenfeld definiert eine Gruppe von Elementen, die oft voneinander abhängig sind.

Sonderverhalten Variable

NX, NY AS INTEGER	Wird einer der Variablen mit der Konstante *IsDefault* belegt, wird der Wert 50 angenommen.
Value AS INTEGER	Diese Variable hat immer den Wert 0. Sie wird nicht interpretiert.

Sonderverhalten Methoden

FUNCTION GETTEXT() AS STRING	Der Text wird zurückgeliefert.
SUB SETTEXT (TEXT$)	Der Text wird neu definiert.

Beispiel

Das folgende Beispiel zeigt die Verwendung eines Gruppenfeldes in Zusammenhang mit Radiobuttons.

```
REM $INCLUDE: 'DIALOG.INC'

DIM Dlg AS Dialog
DIM Group AS GroupBox
DIM Ok AS PushButton, Cancel AS PushButton
DIM Check1 AS RadioButton, Check2 AS RadioButton, Check3 AS RadioButton

DIALOG           Dlg, IsAuto,IsAuto,165, 63, "Test Check-Boxes"
  GROUPBOX       Group,     5,    5,100, 55, "Gruppe"
    RADIOBUTTON  Check1,    9,   15, 95, 14, "Auswahl 1"
    RADIOBUTTON  Check2,    9,   28, 95, 14, "Auswahl 2"
    RADIOBUTTON  Check3,    9,   41, 95, 14, "Auswahl 3"
  ENDGROUP
  PUSHBUTTON     Ok,      115,    6, 45, 14, "OK", IsDefault
  PUSHBUTTON     Cancel,  115,   26, 45, 14, "Abbrechen", IsCancel

ON CLICK (Ok)     GOSUB Ok
ON CLICK (Cancel) GOSUB Cancel
ON CLICK (Check1) GOSUB Set1
ON CLICK (Check2) GOSUB Set2
ON CLICK (Check3) GOSUB Set3

Res$ = Check1.GetText()
Check1.Value = TRUE
STARTDIALOG
PRINT Res$
```

```
END

Set1:
   Res$ = Check1.GetText() : RETURN
Set2:
   Res$ = Check2.GetText() : RETURN
Set3:
   Res$ = Check3.GetText() : RETURN
Ok:
   ENDDIALOG : RETURN
Cancel:
   Res$ = "Abbruch"
   ENDDIALOG : RETURN
```

CLASS Icon

```
CLASS Icon PUBLIC Control
PUBLIC
   SUB Load CDECL (FILENAME$)
END CLASS
```

Beschreibung

Diese Klasse instanziiert ein Icon. Die initialisierende Routine ist ICON. Das Icon wird aus einer Datei geladen.

Konstante

CONST DefaultIcon = "#0001"	Diese Konstante lädt das system-interne Default-Icon.
CONST InfoIcon = "#0002"	Diese Konstante lädt das system-interne Informations-Icon.
CONST WarnIcon = "#0003"	Diese Konstante lädt das system-interne Icon für Warnungen.
CONST ErrorIcon = "#0004"	Diese Konstante lädt das system-interne Icon für Fehlermeldungen.
CONST QueryIcon = "#0005"	Diese Konstante lädt das system-interne Icon für Anfragen.

Methoden

SUB LOAD (FILENAME$)	Mit dieser Methode kann in das bestehende Element ein neues Icon geladen werden. Dies ermöglicht Animationseffekte etc.

Sonderverhalten Variable

NX, NY AS INTEGER	Die Größe des Icons. Diese beiden Variable werden mit der in der Datei angegebenen Größe vorbesetzt. Eine Änderung dieser Variablen ist nicht möglich, da die Größe eines Icons festliegt.
Value AS INTEGER	Diese Variable hat immer den Wert 0. Sie wird nicht interpretiert.

Sonderverhalten Methoden

FUNCTION GETTEXT() AS STRING	Ein leerer Text wird zurückgeliefert.
SUB SETTEXT (TEXT$)	Diese Methode hat keine Wirkung.

Events

ON CLICK (Variable) GOSUB	Wird das Dialogelement angeklickt, wird zum angegebenen GOSUB-Label verzweigt.

Beispiel

Das folgende Programm zeigt die Verwendung von Icons, statischen Texten und Bitmaps. Die Datei "BLAETTER.BMP" enthält eine Bitmap, die Datei "FILES03A.ICO" ein Icon, das einen geschlossenen Karteischrank zeigt, während die Datei "FILES03B.ICO" ein Icon mit einem geöffnenten Karteischrank enthält.

Datei BLAETTER.BMP:

Dateien FILES03A.ICO und FILES03B.ICO:

```
REM $INCLUDE: 'DIALOG.INC'

CLASS MyIcon PUBLIC Icon
PUBLIC
   Raus AS INTEGER
END CLASS

DIM Dlg AS Dialog
DIM Ok AS PushButton
DIM Text AS FixedText, Meldung AS FixedText
DIM anIcon (5) AS MyIcon
DIM Blaetter AS Bitmap

DIALOG  Dlg, IsAuto,IsAuto,165,165, "Test statische Controls"
  FIXEDTEXT Text, 9,10, 95, 10, "Klick mich!", HasBorder+IsCenter
  ICON           anIcon(1), 6,   30, InfoIcon
  ICON           anIcon(2),26,   30, QueryIcon
  ICON           anIcon(3),46,   30, WarnIcon
  ICON           anIcon(4),66,   30, ErrorIcon
  ICON           anIcon(5),86,   30, "files03a.ico"
  FIXEDTEXT      Meldung, 115,   30, 45, 20, "", HasBorder+IsCenter
  BITMAP         Blaetter, 20,   60, "blaetter.bmp", HasBorder
  PUSHBUTTON     Ok,      115,    6, 45, 14, "OK", IsDefault

' Die Bitmap zentrieren:

Blaetter.X0 = (Dlg.NX - Blaetter.NX) / 2

' Die Meldung ausgeben:

GOSUB Clear

ON CLICK (Ok)         GOSUB Ok
ON CLICK (Text)       GOSUB Text
ON CLICK (Meldung)    GOSUB Clear
ON CLICK (anIcon (1)) GOSUB Icon1
ON CLICK (anIcon (2)) GOSUB Icon2
ON CLICK (anIcon (3)) GOSUB Icon3
ON CLICK (anIcon (4)) GOSUB Icon4
ON CLICK (anIcon (5)) GOSUB Icon5
ON CLICK (Blaetter)   GOSUB Blatt

STARTDIALOG
END

' Event-Handler:
```

```
Text:
   Meldung.SetText ("Text geklickt!") : RETURN
Clear:
   IF anIcon(5).Raus THEN A$ = "offen" : else A$ = "geschlossen"
   Meldung.SetText ("Schublade " + A$) : RETURN
Icon1:
   Meldung.SetText ("Info-Icon geklickt!") : RETURN
Icon2:
   Meldung.SetText ("Frage-Icon geklickt!") : RETURN
Icon3:
   Meldung.SetText ("Warn-Icon geklickt!") : RETURN
Icon4:
   Meldung.SetText ("Fehler-Icon geklickt!") : RETURN
Icon5:
   IF anIcon(5).Raus THEN
      anIcon(5).Load "files03a.ico"
      anIcon(5).Raus = FALSE
      Meldung.SetText ("Schublade geschlossen")
   ELSE
      anIcon(5).Load "files03b.ico"
      anIcon(5).Raus = TRUE
      Meldung.SetText ("Schublade offen")
   END IF
   RETURN
Blatt:
   Meldung.SetText ("Bitmap geklickt!") : RETURN
Ok:
   ENDDIALOG
   RETURN
```

CLASS ListBox

```
CLASS ListBox PUBLIC Control
PUBLIC
   Count AS INTEGER
   SUB      Add CDECL (T$, OPTIONAL BYVAL I%)
   SUB      Remove CDECL (OPTIONAL BYVAL I%)
   FUNCTION GetIndex CDECL (T$) AS INTEGER
   FUNCTION GetItem  CDECL (OPTIONAL BYVAL I%) AS STRING
END CLASS
```

Beschreibung

Diese Klasse instanziiert eine Listbox. Die initialisierende Routine ist LISTBOX.

Konstante

IsSorted = 2	Dieses Flag-Bit definiert eine Listbox, deren Einträge alphabetisch sortiert sind.
IsDropDown = 4	Dieses Flag-Bit definiert eine Listbox, die einzeilig erscheint und die bei Bedarf aufgeklappt werden kann.

Variable

Count AS INTEGER	Die Anzahl Einträge in der Listbox.

Methoden

SUB Add (T$, I%)	Der angegebene Text wird in die Listbox eingetragen. Der zweite Parameter ist optional und bezeichnet die Position (ab 1), an der der Eintrag stehen soll. Wird er nicht angegeben, wird der Eintrag an das Ende angefügt. Der zweite Parameter ist ohne Wirkung, wenn die Listbox als sortierte Listbox deklariert wurde.
SUB Remove (I%)	Der angegebene Eintrag (ab 1) wird entfernt. Wird kein Wert angegeben, wird der aktuelle Eintrag enmtfernt.
FUNCTION GetIndex (T$) AS INTEGER	Die Position des angegebenen Eintrags wird zurückgeliefert (ab 1).
FUNCTION GetItem (I%) AS STRING	Der Text, der unter der angegebenen Position (ab 1) in der Listbox eingetragen ist, wird zurückgeliefert. Wird kein Wert angegeben, wird der aktuelle Eintrag zurückgeliefert.

Sonderverhalten Variable

NX, NY AS INTEGER	Wird einer der Variablen mit der Konstante *IsDefault* belegt, wird der Wert 50 angenommen.
Value AS INTEGER	Dieses Feld enthält die Position des aktuellen Eintrags (ab 1). Ist kein Eintrag aktuell angewählt, wird der Wert 0 zurückgeliefert.

Sonderverhalten Methoden

FUNCTION GETTEXT() AS STRING
: Der aktuelle Eintrag wird zurückgeliefert.

SUB SETTEXT (TEXT$)
: Ist der übergebene Text in der Listbox vorhanden, wird der Eintrag zum aktuellen Eintrag.

Events

ON ENTER (Variable) GOSUB
: Was das Dialogelement inaktiv und wird es angeklickt oder für Tastatureingaben aktiviert, wird auf den angegebenen GOSUB-Label verzweigt.

ON LEAVE (Variable) GOSUB
: War das Dialogelement aktiv und es wird ein anderes Element aktiviert, wird auf den angegebenen GOSUB-Label verzweigt.

ON CHANGE (Variable) GOSUB
: Ändert sich die Auswahl der Listbox, wird zum angegebenen GOSUB-Label verzweigt. Das Feld *Value* enthält den Index des ausgewählten Eintrags ab 1.

Beispiel

Das folgende Programm zeigt die Verwendung einer Listbox.

```
REM $INCLUDE: 'DIALOG.INC'

DIM Dlg AS Dialog
DIM Ok AS PushButton, Cancel AS PushButton
DIM Test AS ListBox

DIALOG           Dlg,  IsAuto,IsAuto,165, 63, "Test Listbox"
  LISTBOX        Test,       9,     6, 95, 50, IsSorted+HasBorder
  PUSHBUTTON     Ok,       115,     6, 45, 14, "OK", IsDefault
  PUSHBUTTON     Cancel,   115,    26, 45, 14, "Abbrechen", IsCancel

ON CLICK (Ok)     GOSUB Ok
ON CLICK (Cancel) GOSUB Cancel

Test.Add ("Apfel")
Test.Add ("Banane")
Test.Add ("Zitrone")
Test.Add ("Birne")
Test.Add ("Pflaume")
Test.Add ("Kirsche")
Test.SetText ("Kirsche")
Res$ = "Kein Ergebnis"
N = 1
STARTDIALOG
PRINT Res$
END
```

```
Ok:
   Res$ = Test.GetText()
   ENDDIALOG : RETURN

Cancel:
   Res$ = "Abbruch"
   ENDDIALOG : RETURN
```

CLASS ComboBox

```
CLASS ComboBox PUBLIC ListBox
END CLASS
```

Beschreibung

Diese Klasse instanziiert eine Combobox. Sie hat keine eigene Funktionalität. Die initialisierende Routine ist COMBOBOX. Da die Klasse von der Klasse *ListBox* abgeleitet ist, übernimmt sie deren Methoden und Konstante.

Sonderverhalten Variable

NX, NY AS INTEGER	Wird einer der Variablen mit der Konstante *IsDefault* belegt, wird der Wert 50 angenommen.
Value AS INTEGER	Dieses Feld enthält die Position des aktuellen Eintrags (ab 1). Ist kein Eintrag aktuell angewählt, wird der Wert 0 zurückgeliefert.

Sonderverhalten Methoden

FUNCTION GETTEXT() AS STRING
: Der Inhalt des Eingabefeldes wird zurückgeliefert.

SUB SETTEXT (TEXT$)
: Der Text wird in das Eingabefeld übernommen.

Events

ON ENTER (Variable) GOSUB
: Was das Dialogelement inaktiv und wird es angeklickt oder für Tastatureingaben aktiviert, wird auf den angegebenen GOSUB-Label verzweigt.

ON LEAVE (Variable) GOSUB
: War das Dialogelement aktiv und es wird ein anderes Element aktiviert, wird auf den angegebenen GOSUB-Label verzweigt.

ON CHANGE (Variable) GOSUB
: Ändert sich die Auswahl der Combobox, wird zum angegebenen GOSUB-Label verzweigt. Das Feld *Value* enthält den Index des ausgewählten Eintrags ab 1.

ON KEY (Variable) GOSUB
: Bei einer Tastatureingabe wird zum angegebenen GOSUB-Label verzweigt. Das Value-Feld enthält den Code der Taste. Die Taste kann ignoriert werden, indem das Feld Value auf 0 gesetzt wird.

Beispiel

Das Beispiel für die Listbox kann auch für die Combobox verwendet werden.

CLASS PopupMenu

```
CLASS PopupMenu PUBLIC Control
PUBLIC
   SUB Check  CDECL (BYVAL ID%, BYVAL MODE%)
   SUB Enable CDECL (BYVAL ID%, BYVAL MODE%)
   FUNCTION IsChecked CDECL (BYVAL ID%) AS INTEGER
   FUNCTION IsEnabled CDECL (BYVAL ID%) AS INTEGER
END CLASS
```

Beschreibung

Diese Klasse instanziiert ein Menü. Die initialisierende Routine ist POPUPMENU. Ein Menü wird durch eine Reihe der Anweisungen MENUITEM, BITMAPITEM, SUBMENU und SEPARATOR aufgebaut. Die Definition des Menüs endet mit einer ENDMENU-Anweisung.

Lage, Größe und Text der Klasse beschreiben nicht das Menü selbst, sondern vielmehr einen speziellen Button, der, wenn er aktiviert wird, das Menü anzeigt.

Konstante

HasCheck = 64	Dieses Flag-Bit wird bei den Anweisungen MENUITEM und BITMAPITEM verwendet. Es trägt ein Häkchen neben dem Menüeintrag ein.
IsDisabled = 128	Dieses Flag-Bit wird bei den Anweisungen MENUITEM und BITMAPITEM verwendet. Der so markierte Menüeintrag wird grau dargestellt und kann nicht angewählt werden.

Methoden

SUB Check (ID%, MODE%)	Der angegebene Menüeintrag wird entweder mit einem Häkchen versehen (MODE% ist ungleich 0) oder ein eventuell bestehendes Häkchen wird entfernt. Diese Methode entspricht dem Flag-Bit *HasCheck*, wird aber üblicherweise im CLICK-Handler gerufen.
SUB Enable (ID%, MODE%)	Der angegebene Menüeintrag wird entweder verfügbar gemacht (MODE% ungleich 0) oder nicht verfügbar gemacht. Diese Methode entspricht dem Flag-Bit *IsDisabled*, wird aber üblicherweise im CLICK-Handler gerufen.
FUNCTION IsChecked (ID%) AS INTEGER	Der Status des angegebenen Eintrags wird zurückgeliefert. Der Returnwert ist TRUE, wenn der Eintrag mit einem Häkchen markiert ist, sonst FALSE.
FUNCTION IsEnabled (ID%) AS INTEGER	Der Status des angegebenen Eintrags wird zurückgeliefert. Der Returnwert ist TRUE, wenn der Eintrag anwählbar ist, sonst FALSE.

Sonderverhalten Variable

X0, Y0 AS INTEGER	Diese beiden Variablen beschreiben die Lage des Buttons, der das Menü auslöst.

NX, NY AS INTEGER — Diese beiden Variablen beschreiben die Größe des Buttons, der das Menü auslöst. Wird eine der beiden Variablen mit dem Wert *IsAuto* belegt, wird der vorgegebene Text zur Größenberechnung hinzugezogen. Das Element wird so berechnet, daß um den Text ein Rand von 2 Einheiten verbleibt.

NX, NY AS INTEGER — Wird einer der Variablen mit der Konstante *IsDefault* belegt, wird der Wert 50 angenommen

Value AS INTEGER — Dieses Feld enthält die ID-Nummer des aktuell ausgewählten Menüeintrags. Eine Änderung der Variable hat keinen Effekt.

Sonderverhalten Methoden

FUNCTION GETTEXT() AS STRING — Der Text des Buttons, der das Menü auslöst, wird zurückgeliefert.

SUB SETTEXT (TEXT$) — Der Text des Buttons, der das Menü auslöst, wird neu definiert.

Events

ON ENTER (Variable) GOSUB — Was das Dialogelement inaktiv und wird es angeklickt oder für Tastatureingaben aktiviert, wird auf den angegebenen GOSUB-Label verzweigt.

ON LEAVE (Variable) GOSUB — War das Dialogelement aktiv und es wird ein anderes Element aktiviert, wird auf den angegebenen GOSUB-Label verzweigt.

ON CLICK (Variable) GOSUB — Wird das Dialogelement angeklickt oder es wird die Leertaste gedrückt, wenn das Element aktiv ist, wird zum angegebenen GOSUB-Label verzweigt. Dort kann das Menü initialisiert werden, ehe es angezeigt wird.

ON CHANGE (Variable) GOSUB — Wird ein Menüpunkt angewählt, wird das Feld *Value* mit dem ID des Elements besetzt und es wird zum angegebenen GOSUB-Label verzweigt.

Beispiel

Das folgende Beispiel zeigt den Aufbau eines mehrstufigen Menüs.

```
REM $INCLUDE: 'DIALOG.INC'

OPTION BASE 1

DIM Dlg AS Dialog
DIM Text AS FixedText
DIM Menu AS PopupMenu
DIM Ok AS PushButton
DIM Check% (4)

DIALOG           Dlg,  IsAuto,IsAuto,165, 63, "Test Menues"
```

```
  FIXEDTEXT Text, 70, 34, 40, 12, "Nichts", IsCenter+HasBorder
  POPUPMENU      Menu,     20,    30, 40, 20, "Menue 1"
    MENUITEM     "Menue-Item ~1", 1001
    MENUITEM     "Menue-Item ~2", 1002, IsDisabled
    MENUITEM     "Menue-Item ~3", 1003
    SEPARATOR
    BITMAPITEM   "blaetter.bmp",  1004
    SEPARATOR
    SUBMENU      "Popup-Menue ~A", 1010
      MENUITEM   "Popup-Submenue ~1", 1011
      MENUITEM   "Popup-Submenue ~2", 1012
      MENUITEM   "Popup-Submenue ~3", 1013
      MENUITEM   "Popup-Submenue ~4", 1014
    ENDMENU
    SEPARATOR
    MENUITEM     "Menue-Item 5", 1005
  ENDMENU
  PUSHBUTTON     Ok, 115, 6, 45, 14, "OK", IsDefault

ON CLICK  (Ok)   GOSUB Ok
ON CLICK  (Menu) GOSUB Prepare
ON CHANGE (Menu) GOSUB Changed

Res$ = "Kein Ergebnis"
STARTDIALOG
PRINT Res$
END

Ok:
   Res$ = "Menu ID " + Text.GetText()
   ENDDIALOG
   RETURN
' Vorbereiten des Menues
Prepare:
   FOR I% = 1 TO 4
      Menu.Check (I% + 1010, Check% (I%))
   NEXT
   RETURN
' Reaktion auf Auswahl
Changed:
   N = Menu.Value
   IF N > 1010 THEN Check% (N - 1010) = NOT Menu.IsChecked (N)
   Text.SetText (STR$ (N))
   RETURN
```

CLASS PushButton

```
CLASS PushButton PUBLIC Control
END CLASS
```

Beschreibung

Diese Klasse instanziiert einen einfachen Button. Sie hat keine eigene Funktionalität. Die initialisierende Routine ist PUSHBUTTON.

Konstante

IsDefault = 2	Der Button wird mit der ENTER-Taste automatisch aktiviert.
IsCancel = 4	Der Button wird aktiviert, wenn die ESC-Taste gedrückt wird oder der Dialog ohne Betätigung eines Elements (per Fensterelement) geschlossen wurde.

Sonderverhalten Variable

NX, NY AS INTEGER	Wird eine der beiden Variablen mit dem Wert *IsAuto* belegt, wird der vorgegebene Text zur Größenberechnung hinzugezogen. Das Element wird so berechnet, daß um den Text ein Rand von 2 Einheiten verbleibt.
Value AS INTEGER	Diese Variable hat immer den Wert 0. Sie wird nicht interpretiert.

Sonderverhalten Methoden

FUNCTION GETTEXT() AS STRING
: Der Text, der im Button erscheint, wird zurückgeliefert.

SUB SETTEXT (TEXT$)
: Der Text, der im Button erscheint, wird neu definiert.

Events

ON ENTER (Variable) GOSUB
: Was das Dialogelement inaktiv und wird es angeklickt oder für Tastatureingaben aktiviert, wird auf den angegebenen GOSUB-Label verzweigt.

ON LEAVE (Variable) GOSUB
: War das Dialogelement aktiv und es wird ein anderes Element aktiviert, wird auf den angegebenen GOSUB-Label verzweigt.

ON CLICK (Variable) GOSUB
: Wird das Dialogelement angeklickt oder es wird die Leertaste gedrückt, wenn das Element aktiv ist, wird zum angegebenen GOSUB-Label verzweigt.

Beispiel

Siehe Beispiel bei der Klasse *Dialog*.

CLASS RadioButton

```
CLASS RadioButton PUBLIC Control
END CLASS
```

Beschreibung

Diese Klasse instanziiert einen Radio-Button. Sie hat keine eigene Funktionalität. Die initialisierende Routine ist RADIOBUTTON. Ein Radiobutton ist eine Checkbox, die alle anderen Radiobuttons innerhalb einer Gruppe automatisch deaktiviert, wenn sie aktiviert wird. Dadurch ist immer nur ein Radiobutton in einer Gruppe aktiv.

Sonderverhalten Variable

NX, NY AS INTEGER	Wird eine der beiden Variablen mit dem Wert *IsAuto* belegt, wird der vorgegebene Text zur Größenberechnung hinzugezogen. Das Element wird so berechnet, daß um den Text ein Rand von 2 Einheiten verbleibt.
Value AS INTEGER	Hier wird der aktuelle Zustand des Radiobuttons abgelegt (TRUE oder FALSE). Das Ändern des Wertes veranlaßt eine Statusänderung des Elements.

Sonderverhalten Methoden

FUNCTION GETTEXT() AS STRING	Der Text, der neben dem Radiobutton erscheint, wird zurückgeliefert.
SUB SETTEXT (TEXT$)	Der Text, der neben dem Radiobutton erscheint, wird neu definiert.

Events

ON ENTER (Variable) GOSUB	Was das Dialogelement inaktiv und wird es angeklickt oder für Tastatureingaben aktiviert, wird auf den angegebenen GOSUB-Label verzweigt.
ON LEAVE (Variable) GOSUB	War das Dialogelement aktiv und es wird ein anderes Element aktiviert, wird auf den angegebenen GOSUB-Label verzweigt.
ON CLICK (Variable) GOSUB	Wird das Dialogelement angeklickt oder es wird die Leertaste gedrückt, wenn das Element aktiv ist, wird zum angegebenen GOSUB-Label verzweigt.

Beispiel

Siehe Beispiel bei der Klasse *GroupBox*.

CLASS ScrollBar

```
CLASS ScrollBar PUBLIC Control
PUBLIC
   Minimum  AS INTEGER
   Maximum  AS INTEGER
   SmallInc AS INTEGER
   LargeInc AS INTEGER
END CLASS
```

Beschreibung

Diese Klasse instanziiert einen Scrollbar. Die initialisierende Routine ist SCROLLBAR.

Konstante

IsHorizontal = 0	Dises Flag-Bit definiert einen horizontalen Scrollbar. Es kann auch fortgelassen werden.
IsVertical = 2	Dieses Flag-Bit definiert einen vertikalen Scrollbar.
IsSynchronous = 4	Dieses Flag-Bit veranlaßt den Scrollbar, bei Statusänderungen unverzüglich den CHANGED-Handler zu aktivieren. Dadurch kann kontinuierlich reagiert werden, wenn der Scrollbar verändert wird. Fehlt dieses Bit, wird der CHANGED-Handler erst nach dem Ende der Aktivierung gerufen.

Variable

Minimum AS INTEGER	Der Minimalwert des Scrollbars. Er wird mit 0 vorbesetzt.
Maximum AS INTEGER	Der Maximalwert des Scrollbars. Er wird mit 100 vorbesetzt.
SmallInc AS INTEGER	Der Wert, um den der aktuelle Wert verändert wird, wenn "zeilenweise" erhöht oder erniedrigt wird. Er wird mit 1 vorbesetzt.
LargeInc AS INTEGER	Der Wert, um den der aktuelle Wert verändert wird, wenn "seitenweise" erhöht oder erniedrigt wird.

Sonderverhalten Variable

NX, NY AS INTEGER	Wird eine der beiden Variablen mit dem Wert *IsAuto* belegt, wird der Wert 15 eingesetzt.
Value AS INTEGER	Diese Variable enthält die aktuelle Position des Scrollbars. Ein Änderung dieses Wertes bewirkt eine Änderung der Position des Scrollbars.

Sonderverhalten Methoden

FUNCTION GETTEXT() AS STRING	Ein leerer Text wird zurückgeliefert.
SUB SETTEXT (TEXT$)	Diese Methode hat keine Wirkung.

Events

ON ENTER (Variable) GOSUB

Was das Dialogelement inaktiv und wird es angeklickt oder für Tastatureingaben aktiviert, wird auf den angegebenen GOSUB-Label verzweigt.

ON LEAVE (Variable) GOSUB

War das Dialogelement aktiv und es wird ein anderes Element aktiviert, wird auf den angegebenen GOSUB-Label verzweigt.

ON CHANGE (Variable) GOSUB

Ändert sich die Position des Scrollbars, wird zum angegebenen GOSUB-Label verzweigt. Das Feld *Value* enthält die aktuelle Position.

Beispiel

Das folgende Beispiel zeigt, wie die Werte von Scrollbars abgefragt werden können.

```
REM $INCLUDE: 'DIALOG.INC'

DIM Dlg AS Dialog
DIM Ok AS PushButton
DIM HMeldung AS FixedText, VMeldung AS FixedText
DIM HScroll AS ScrollBar, VScroll AS ScrollBar

DIALOG           Dlg,  IsAuto,IsAuto,165, 63, "Test Scrollbars"
  SCROLLBAR      HScroll,  35,    40, 95, 10, IsSynchronous
  FIXEDTEXT      HMeldung,135,    40, 20, 10, "", HasBorder
  SCROLLBAR      VScroll,   5,     5, 10, 50, IsVertical
  FIXEDTEXT      VMeldung, 20,     5, 20, 10, "", HasBorder
  PUSHBUTTON     Ok,  115, 6, 45, 14, "OK", IsDefault+IsCancel

ON CLICK  (Ok)      GOSUB Ok
ON CHANGE (HScroll) GOSUB Hchange
ON CHANGE (VScroll) GOSUB Vchange

HScroll.Minimum  = 1
HScroll.Maximum  = 1000
HScroll.Value    = 500
HScroll.LargeInc = 50

' Texte initialisieren
GOSUB Hchange : GOSUB Vchange
STARTDIALOG
END

Hchange:
   HMeldung.SetText (STR$ (HScroll.Value)) : RETURN

Vchange:
   VMeldung.SetText (STR$ (VScroll.Value)) : RETURN

Ok:
   ENDDIALOG
```

RETURN

Fehlercodes

2 Syntaxfehler
Dieser Fehler tritt auf, wenn die READ-Anweisung ungültige Daten zuweisen soll oder ein Array mehrfach dimensioniert wird.

3 RETURN ohne GOSUB
Eine RETURN-Anweisung wurde ausgeführt, ohne daß zuvor eine GOSUB-Anweisung ausgeführt wurde.

4 READ ohne DATA
Eine READ-Anweisung versuchte, Daten aus einer DATA-Anweisung zu lesen, obwohl bereits alle DATA-Anweisungen abgearbeitet worden sind.

5 Fehler bei Funktionsaufruf
Dieser Fehler wird von allen internen Unterprogrammen zurückgeliefert. Meist ist ein Parameter der Funktion unzulässig. Der Fehler kann z.B. auch auftreten, wenn ein String zu lang wird.

6 Arithmetischer Überlauf
Es trat ein Überlauf bzw. Unterlauf ein.

8 Array bereits dimensioniert
Ein Array darf nur einmal mit einer DIM-Anweisung dimensioniert werden. Soll es erneut dimensioniert werden, ist die REDIM-Anweisung zu verwenden.

9 Ungültiger Index
Der angegebene Index liegt außerhalb des zulässigen Wertebereichs.

10 Array nicht dimensioniert
Es wurde auf ein Array zugegriffen, ohne daß es zuvor mit der DIM-Anweisung dimensioniert wurde.

11 Division durch 0
Eine Division durch 0 trat auf. Dieser Fehler kann auch während der Compilation auftreten.

13 Ungültiger Datentyp
Der Datentyp eines Ausdrucks kann nicht in der angegebenen Variablen abgelegt werden. Dies ist vor allem bei Strings und numerischen Variablen und Ausdrücken der Fall.

20 RESUME ohne Fehler
Die RESUME-Anweisung wurde außerhalb einer Fehlerbehandlungsroutine verwendet.

52 Ungültige Kanalnummer
Die verwendete Kanalnummer bei einer Schreib/Lese-Operation ist nicht mit einer OPEN-Anweisung mit einer Datei verbunden worden.

53 Datei XXXXX konnte nicht geöffnet werden
Die angesprochene Datei bzw. das angesprochene Modul wurde nicht gefunden. Dieser Fehler kann auch während der Compilation auftreten, wenn eine $INCLUDE-Datei nicht gefunden wurde.

54 Ungültiger Dateizugriff
Die Zugriffsart auf eine Datei ist bei der OPEN-Anweisung ungültig. Es sind nur die Buchstaben I, O, R und A zugelassen.

55 Datei bereits offen
Die angesprochene Datei ist bereits offen, oder es wurde versucht, eine geöffnete Datei zu löschen.

58 Dateiname bereits vergeben
Dieser Fehler tritt beim Umbenennen einer Datei auf, wenn es eine Datei dieses Namens bereits gibt.

61 Platte voll
Es ist für die gerade erstellte Datei kein Platz mehr auf der Platte.

62 Dateiende erreicht
Es wurde ein Lesezugriff hinter dem Ende einer Datei versucht.

67 Zu viele offene Dateien
Es können nicht mehr als 15 Dateien gleichzeitig geöffnet sein.

70 Zugriff verweigert
Es wurde versucht, auf eine schreibgeschützte Datei zu schreiben oder auf eine gesperrte Datei zuzugreifen.

73 Nicht implementiert
Die betroffene Anweisung ist Bestandteil eines anderen BASIC-Dialekts, der von StarBASIC nicht unterstützt wird.

74 RENAME auf zwei Platten
Es wurde versucht, eine Datei per Umbenennung auf eine andere Platte zu übertragen. Eine Übertragung per Umbenennung kann nur zwischen zwei Verzeichnissen derselben Platte erfolgen.

75 Dateizugriffsfehler
Ein allgemeiner E/A-Fehler trat beim Zugriff auf eine Datei auf.

76 Pfad nicht gefunden
Der angegebene Pfad konnte nicht gefunden werden.

100 Kein Zeitgeber verfügbar
Alle im System definierten Zeitgeber sind bereits belegt.

101 Ungültiger Pointer
Der Pointer war nicht initialisiert oder (bei DELETE) er zeigte nicht auf einen zugewiesenen Speicherbereich.

102 Keine Verbindung
Ein dynamischer Link ist nicht zustande gekommen.

103 Falsche Verbindung
Der dynamische Link ist fehlerhaft.

104 Ungültige Daten
Der Empfänger verweigert die Annahme der Daten.

105 Kein Dialog aktiv
Die STARTDIALOG- oder die ENDDIALOG-Anweisung wurde ausgeführt, ohne daß ein Dialog definiert war.

106 Dialog bereits aktiv
Verschachtelte DIALOG-Anweisungen sind unzulässig. EIn Dialog muß erst gestartet weden, ehe ein neuer Dialog definiert werden kann.

107 Dialogelement nicht aktiv
Es wurde versucht, auf ein uninitialisiertes Dialogelement zuzugreifen.

108 ON-Bedingung für Dialogelement nicht unterstützt
Es wurde versucht, einen Event-handler zu registrieren, der vom angesprochenen Element nicht unterstützt wird.

109 Keine Elementgruppe aktiv
Eine ENDGROUP-Anweisung schließt eine vorangegangene GROUPBOX-Anweisung ab.

110 Elementgruppe bereits aktiv
Verschachtelte GROUPBOX-Anweisungen sind unzulässig.

111 Kein aktives Menü
Eine ENDMENU-Enawieung schließt eine vorangegangene POPUPMENU- oder SUBMENU-Anweisung ab.

112 Menü bereits aktiv
Verschachtelte POPUPMENU-Anweisungen sind unzulässig.

900 Modul XXXXX nicht in Library
Es wurde versucht, ein Modul mit Externreferenzen oder COMMON-Zonen zu aktivieren, das sich nicht in einer Library befindet.

901 Unterprogramm XXXXX nicht gefunden
Eine benötigte externe Funktion konnte nicht gefunden werden.

902 Library XXXXX nicht gefunden
Eine explizit mit der LIB-Klausel der DECLARE-Anweisung angeforderte Library konnte nicht gefunden werden.

903 Überlauf der Tokentabelle
Dieser Fehler tritt dann auf, wenn während eines Programmlaufs mehr als 4.194.304 Ladevorgänge eines Moduls in eine Library stattfanden. Anschließend können keine Module mehr in Libraries geladen werden.

904 Binärdatei ist zu alt
Die Datei wurde mit einer veralteten Version des Compilers erzeugt. Bitte neu compilieren.

905 COMMON-Zone XXX bereits anders deklariert
Die angegebene COMMON-Zone stimmt im Format nicht mit einer bereits vorhandenen COMMON-Zone gleichen Namens überein.

906 Fehler in Binärdatei
In der Binärdatei sind fehlerhafte Symbol-Informationen abgespeichert.

990 Stack-Überlauf
Dieser Fehler tritt meist bei rekursiven Unterprogrammen auf, wenn die Verschachtelung zu tief ist.

991 Zu wenig Hauptspeicher
Normalerweise sollte dieser Fehler nie auftreten. Wenn Hauptspeicher zu knapp wird, kann die

Applikation bereits im Vorwege darauf reagieren.

992 Interner Fehler
Ein interner Fehler trat in der Virtuellen Maschine auf (z.B. ein undefinierter Befehlscode).

993 Undefiniertes Objekt
Es wurde eine Methode für ein nicht (mehr) existentes Objekt gerufen.

1001 Name erwartet
Hier wurde ein Name, z.B. von einer Variablen oder einer Funktion, erwartet.

1002 Fehler im Klammerung
Es fehlen schließende Klammern in einem Ausdruck.

1003 Dimensionsangabe erwartet
Es fehlt die Dimensionierung eines Arrays.

1004 'XXXXX' erwartet
Das Symbol "XXXXX" wurde an dieser Stelle erwartet. Dabei kann es sich auch um ein Zeichen wie das Sternchen oder das Gleichheitszeichen handeln.

1005 Variable 'XXXXX' anders deklariert
Die Variable wurde z.B. als COMMON mit der AS-Anweisung deklariert, obwohl sie ohne AS-Anweisung deklariert wurde. Ein Vermischen von einer Deklaration über den Datentyp und der Deklaration mit der AS-Klausel ist nicht möglich.

1006 Zu viele Dimensionen
Die maximale Anzahl von Array-Dimensionen ist 60.

1007 Zu viele Parameter
Die maximale Anzahl von Unterprogramm-Parametern ist 60.

1008 Zu viele Labels
Es können nur maximal 256 Labels bei einer ON...GOSUB/ON...GOTO-Anweisung angegeben werden.

1009 "FN" ist für DEF FN-Funktionen reserviert
Ein Unterprogrammname darf nicht mit den Buchstaben "FN" beginnen, da diese beiden Buchstaben für DEF FN-Funktionen reserviert sind.

1010 Funktionsname muß mit 'FN' beginnen
Der Name einer Funktion,die mit der DEF-Anweisung deklariert wird, muß mit den Buchstaben 'FN' beginnen.

1011 XXXXX: String fester Länge unzulässig
Strings fester Länge können nicht als formale Parameter deklariert werden. Der Parameter XXXXX ist davon betroffen.

1012 Keine Strings variabler Länge oder Arrays
Strukturen dürfen weder Strings variabler Länge noch Arrays enthalten.

1013 Ungültiger COMMON-Blockname
Der angegebene COMMON-Blockname ist nicht verwendbar (weil er z.B. ein reserviertes Wort darstellt).

1014 XXXXX nicht am Programmanfang
Die Anweisung XXXXX muß vor allen anderen Anweisungen, die Code erzeugen, stehen.

1015 Fehler in Blockschachtelung (XXXXX)
Eine der Anweisungen NEXT, WEND, END IF, END SELECT oder LOOP wurde ohne das entsprechende Gegenstück FOR, WHILE, IF, SELECT CASE oder DO vorgefunden.

1016 DECLARE erforderlich: XXXXX
Das Unterprogramm XXXXX wurde aufgerufen, ohne daß es vorher deklariert wurde.

1017 Unterprogramm XXX anders deklariert
Zwischen der Deklaration und der Definition eines Unterprogramms treten Unterschiede auf. Dieser Fehler tritt auch auf, wenn eine SUB als FUNCTION und umgekehrt verwendet wurde.

1018 Interner Fehler (XXXXX)
Dieser Compilerfehler kann als Folgefehler auftreten, wenn eine Programmschleife nicht ordentlich abgeschlossen wurde. Der Text "XXXXX" ist in diesem Falle "BACKCHAIN". Weiter kann dieser Fehler auftreten, wenn eine interne Hilfsfunktion nicht korrekt aufgerufen wurde. In diesem Fall wird der Name der Hilfsfunktion angegeben.

1019 Label XXXXX bereits definiert
Das Label XXXXX tritt mehrfach in einem Programm auf.

1020 Strukturelement XXXXX unbekannt
In der angesprochenen Datenstruktur ist ein Element namens XXXXX unbekannt.

1021 ELSE/ELSEIF ohne IF
Eine der beiden Anweisungen ELSE oder ELSIF wurde vorgefunden, ohne daß vorher eine IF-Anweisung erkannt wurde.

1022 Numerische Konstante erwartet
An dieser Stelle darf nur ein konstanter Zahlenwert stehen (z.B: als Inkrementwert einer FOR-Schleife).

1023 Variable erwartet
An dieser Stelle wird eine einfache Variable erwartet (z.B. als FOR-Variable oder in einer READ-Anweisung).

1024 Ungültige EXIT-Anweisung
Die verwendete EXIT-Anweisung ist an dieser Stelle ungültig. Dies ist der Fall, wenn eine EXIT-Anweisung außerhalb des betreffenden Blocks gefunden wurde.

1025 Fehler in Parameterliste (XXXXX)
Die Liste der formalen Parameter einer Funktion ist fehlerhaft, wenn z.B. ein Parametername mehrfach verwendet wurde. XXXXX bezeichnet den Namen der Funktion.

1026 Unterprogramm XXXXX bereits definiert
Das Unterprogramm XXXXX ist bereits in diesem Modul definiert worden.

1027 Unterprogramm XXXXX ist als C oder PASCAL deklariert
Es wurde versucht, ein Unterprogramm zu schreiben, das den selben Namen trägt wie ein Unterprogramm, das bereits als CDECL oder PASCAL deklariert wurde.

1028 Fehler in Parameterliste (ON xxxx GOSUB)
Ein explizit deklariertes ON...GOSUB-Event ist mit falschen Parametern versorgt worden.

1029 Anweisung XXXXX in Unterprogramm unzulässig
Die Anweisung XXXXX kann nicht innerhalb eines Unterprogramms verwendet werden (z.B. DECLARE).

1030 Anweisung XXXXX in Hauptprogramm unzulässig
Die Anweisung XXXXX darf nicht außerhalb eines Unterprogramms verwendet werden (z.b. STATIC).

1031 Nur AS in TYPE zulässig
In einem TYPE-Block dürfen nur die Anweisungen REM und AS verwendet werden.

1032 Unzulässiges Zeichen in Zahl
Eine Zahl enthält ein unzulässiges Zeichen, wie z.B. die Zahl &H123G&.

1033 Integerzahl erwartet
An dieser Stelle wird eine Zahl zwischen 1 und 32767 erwartet.

1034 Ungültiges Symbol: 'XXXXX'
Der Quelltext enthält ein ungültiges Symbol, wie z.B. ein Controlzeichen.

1035 Ungültiger Konstanten-Ausdruck
Der Ausdruck kann nicht zu einer Konstanten umgerechnet werden.

1036 Label XXXXX nicht definiert
Eine GOTO- oder GOSUB-Anweisung enthält ein Label, das nicht im Modul definiert ist.

1038 Label erwartet
Das Label fehlt oder ist ungültig bei einer GOTO- oder GOSUB-Anweisung.

1039 Ungültiger Indexbereich
Der angegebene Indexbereich bei einer DIM-Anweisung ist ungültig (z.B. weil der untere Index größer ist als der obere Index).

1040 String erwartet
Als Parameter darf nur ein String angegeben werden.

1041 Parameter XXXXX anders deklariert
Die Deklaration des Parameters XXXXX weicht von einer vorangehenden Deklaration ab.

1043 Struktur XXXXX ist > 64 KBytes
Eine Struktur darf bis zu 64 KBytes groß sein.

1044 Strukturname unzulässig
Die Verwendung einer Struktur ist an dieser Stelle unzulässig (z.B. als Teil eines arithmetischen Ausdrucks).

1045 Falsche Anzahl Dimensionen
Die Anzahl der Dimensionen bei einem Array-Element entspricht nicht der Dimensionierung des Arrays.

1046 Unzulässiger Operator: 'XXXXX'
Die Verwendung des angegebenen Operators ist hier nicht zugelassen.

1047 Option falsch oder unbekannt: 'XXXXX'

Der Compiler erkannte die Option XXXXX nicht.

1048 Keine Struktur
An dieser Stelle wird die Angabe einer Struktur-Variablen erwartet.

1049 Symbol XXXXX bereits definiert
Der angegebene Name ist bereits anderweitig verwendet worden.

1050 Zu viele Unterprogramme
Ein Modul kann nur maximal 63 Unterprogramme enthalten.

1051 Nur Pointer auf TYPE-Variable zugelassen
Ein Pointer kann nur auf eine Variable zeigen, deren Datentyp mit einer TYPE-Anweisung definiert wurde.

1052 Programm wird > 64 K
Das Programm zusammen mit den statischen Datenbereichen ist mehr als 64 KBytes groß. Eventuell ist ein String fester Länge zu lang.

1053 Zu viele COMMON-Zonen
Es können nur bis zu 22 COMMON-Zonen pro Modul und bis zu 64 COMMON-Zonen pro Library angelegt werden.

1054 Keine Quelldatei angegeben
Der Compiler wurde aufgerufen, ohne daß die Angabe einer Quelldatei erfolgte.

1055 Mehr als eine Quelldatei angegeben
Es ist nicht möglich, mehr als eine Quelldatei auf einmal zu compilieren.

1060 Variable XXX ist PRIVATE
Die Variable XXX ist als PRIVATE deklariert. Auf diese Variable können nur Methoden der eigenen Klasse zugreifen.

1061 Variable XXX ist PROTECTED
Die Variable XXX ist als PROTECTED deklariert. Auf diese Variable können nur Methoden der eigenen Klasse sowie von abgeleiteten Klassen zugreifen.

1062 Basisklasse XXX unbekannt
Die angegebene Basisklasse ist unbekannt.

1063 Parameter bei Destruktor nicht gestattet
Ein Destruktor hat grundsätzlich keine Parameter.

1064 XXX ist keine Klasse
An dieser Stelle wurde die Angabe einer Klasse erwartet.

1065 Pointer in COMMON-Zone unzulässig: XXX
COMMON-Zonen dürfen keine Pointers enthalten.

1066 Destruktor in COMMON-Zone unzulässig: XXX
Klassen mit Destruktoren können nicht in COMMON-Zonen abgelegt werden.

1067 Methode XXX kann nur mit CALL aktiviert werden
Der Aufruf einer als SUB deklarierten Methode muß mit dem CALL-Befehl erfolgen.

1068 Klasse XXX ist keine abgeleitete Klasse
Beim ALLOCATE-Befehl können nur von der Klasse des Pointers abgeleitete Klassen angegeben werden.

1070 Direkter Aufruf eines Konstruktors unzulässig
Der Konstruktor einer Klasse kann nicht direkt aufgerufen werden.

1071 Direkter Aufruf eines Destruktors unzulässig
Der Destruktor einer Klasse kann nicht direkt aufgerufen werden.

1072 Konstruktor der Basisklasse nicht aufgerufen
Da die Basisklasse einen Konstruktor hat, muß dieser explizit aufgerufen werden.

1073 Konstruktor der Variablen XXX nicht aufgerufen
Da die Variable XXX der Klasse selbst eine Klasse mit einem Konstruktor ist, muß dieser explizit aufgerufen werden.

2000 Zeile zu lang
Die maximale Zeilenlänge beträgt 256 Zeichen.

2001 Name zu lang
Ein Name darf bis zu 40 Zeichen lang sein.

2002 Undefinierte Variable: XXXXX
Die Variable XXXXX wurde intern definiert.

2003 Konstante XXXXX neu definiert
Die Konstante XXXXX war bereits definiert. Die Definition ist durch die neue Definition ersetzt worden.

Tastaturbelegungen

Editierfunktionen

Entf	Text löschen
Shift-Entf	Text in Clipboard verschieben
Shift-Einfg	Text aus Clipboard einfügen
Ctrl-Einfg	Text in Clipboard kopieren

Programmstart

F5	Programm starten
Shift-F5	Programm compilieren
Alt-F5	Programm anhalten
F7	Vorheriger Programmfehler
F8	Nächster Programmfehler

Verschiedenes

F2	Datei speichern
F4	Suchen/ersetzen wiederholen

Funktions-Referenz Dialoge

Dieser Abschnitt beschreibt die StarBASIC-Funktionen, die für die Arbeit mit Dialogen zur Verfügung stehen. Die einzelnen Beschreibungen sind alphabetisch sortiert.

Alphabetische Liste der Funktionen

`BITMAP Variable, x0, y0, Nummer [, Flags]`
Definition einer Bitmap.

`BITMAPITEM "Dateiname", Id [, flags]`
Definition eines Menüeintrags, bestehend aus einer Bitmap.

`CHECKBOX Variable, x0, y0, nx, ny [, Text [, Flags]]`
Definition einer Checkbox.

`COMBOBOX Variable, x0, y0, nx, ny [, Flags]`
Definition einer Combo-Box.

`DIALOG Variable, x0, y0, nx, ny [, Überschrift]`
Beginn der Deklaration eines Dialogs.

`EDITFIELD Variable, x0, y0, nx, ny [, Text [, Flags]]`
Definition eines Eingabefeldes.

`ENDDIALOG`
Beenden eines Dialogs.

`ENDGROUP`
Beenden einer Elementgruppe.

`ENDMENU`
Beenden der Definition einer Menüs oder Untermenüs.

`FIXEDTEXT Variable, x0, y0, nx, ny [, Text [, Flags]]`
Definition eines Text-Elements.

`GROUPBOX Variable, x0, y0, nx, ny [, Text]`
Beginn der einer Gruppen-Definition.

`ICON Variable, x0, y0, Nummer [, Flags]`
Definition eines Icons.

`LISTBOX Variable, x0, y0, nx, ny [, Flags]`
Definition einer List-Box.

`MENUITEM text, id [, flags]`
Definition eines Menü-Eintrags.

`POPUPMENU Variable, x0, y0, nx, ny, text`
Beginn der Definition eines Menüs.

`RADIOBUTTON Variable, x0, y0, nx, ny [, Text]`

Definition eines Radio-Buttons.

`STARTDIALOG`
Aktivierung eines Dialogs.

`SEPARATOR`
Definition einer Trennlinie in einem Menü.

`SUBMENU text, id`
Beginn der Definition eines Untermenüs.

Querverweis: Liste der Funktionen nach Funktionsgruppen

Liste der Funktionen nach Funktionsgruppen

Dialogaufbau

DIALOG Variable, x0, y0, nx, ny [, Überschrift]
Beginn der Deklaration eines Dialogs.

STARTDIALOG
Aktivierung eines Dialogs.

ENDDIALOG
Beenden eines Dialogs.

Dialogelemente

BITMAP Variable, x0, y0, Nummer [, Flags]
Definition einer Bitmap.

CHECKBOX Variable, x0, y0, nx, ny [, Text [, Flags]]
Definition einer Checkbox.

COMBOBOX Variable, x0, y0, nx, ny [, Flags]
Definition einer Combo-Box.

EDITFIELD Variable, x0, y0, nx, ny [, Text [, Flags]]
Definition eines Eingabefeldes.

FIXEDTEXT Variable, x0, y0, nx, ny [, Text [, Flags]]
Definition eines Text-Elements.

ICON Variable, x0, y0, Nummer [, Flags]
Definition eines Icons.

LISTBOX Variable, x0, y0, nx, ny [, Flags]
Definition einer List-Box.

RADIOBUTTON Variable, x0, y0, nx, ny [, Text]
Definition eines Radio-Buttons.

Element-Gruppen

GROUPBOX Variable, x0, y0, nx, ny [, Text]
Beginn der einer Gruppen-Definition.

ENDGROUP
Beenden einer Elementgruppe.

Menüs

POPUPMENU Variable, x0, y0, nx, ny, text
Beginn der Definition eines Menüs.

SUBMENU text, id
: Beginn der Definition eines Untermenüs.

`BITMAPITEM "Dateiname", Id [, flags]`
: Definition eines Menüeintrags, bestehend aus einer Bitmap.

`MENUITEM text, id [, flags]`
: Definition eines Menü-Eintrags.

`SEPARATOR`
: Definition einer Trennlinie in einem Menü.

`ENDMENU`
: Beenden der Definition einer Menüs oder Untermenüs.

BITMAP

Deklaration einer Bitmap.

Verwendung

```
BITMAP Variable, x0, y0, Dateiname [, Flags]
```

Beschreibung

Im aktuellen Dialog wird eine Bitmap deklariert. Die Bitmap wird im Größenverhältnis 1:1 aus der angegebenen Datei geladen. Die Größe kann nachträglich verändert werden.

Parameter

Variable	Bitmap-Variable. Diese muß vom Type *Bitmap* sein. Lage und Größe werden mit den tatsächlichen Werten für Lage und Größe initialisiert. Die Variable *Value* ist 0.
x0, y0	Die linke obere Ecke des Dialogelements relativ zur linken oberen Ecke der Dialogfläche. Wird für einen der Werte der Wert *IsAuto* angegeben, wird das Element gegenüber der Dialogfläche zentriert.
Dateiname	Der Name der Datei, die die Bitmap enthält.
flags	Flag-Bits:
HasBorder	Die Bitmap wird umrandet.

Events

ON CLICK (Variable) GOSUB	Wird das Dialogelement angeklickt, wird zum angegebenen GOSUB-Label verzweigt.

Anmerkungen

Die Methode *GetText()* liefert einen Leerstring zurück; die Methode *SetText()* wird ignoriert.

BITMAPITEM

Deklaration eines Menüeintrags, der aus einer Bitmap besteht.

Verwendung

```
POPUPMENU ....
  ....
  BITMAPITEM dateiname, id [, flags]
  ....
ENDMENU
```

Beschreibung

Im aktuellen Menü wird ein Menüeintrag deklariert. Dieser Menüeintrag besteht aus einer Bitmap, die aus einer Datei geladen wird.

Parameter

dateiname	Der Name der Datei, die die Bitmap enthält.
id	Das ID des Eintrags. Der Eintrag wird über diesen Wert identifiziert.
flags	Optionale Flag-Bits:
IsChecked	Der Eintrag wird mit einem Häkchen versehen.
IsDisabled	Der Eintrag ist disabled. Er wird grau dargestellt und kann nicht angewählt werden.

Events

Alle Menü-Events werden über die Menü-Variable des Menüs abgewickelt.

Anmerkungen

Für textuelle Menüeinträge ist die MENUITEM-Anweisung zu verwenden. Trennlinien werden mit der SEPARATOR-Anweisung erzeugt.

Querverweise

POPUPMENU, ENDMENU, SUBMENU, MENUITEM, SEPARATOR

CHECKBOX

Deklaration einer Checkbox.

Verwendung

```
CHECKBOX Variable, x0, y0, nx, ny [, Text [, Flags]]
```

Beschreibung

Im aktuellen Dialog wird eine Checkbox deklariert.

Parameter

Variable Checkbox-Variable. Diese muß vom Type *Checkbox* sein. Lage und Größe werden mit den tatsächlichen Werten für Lage und Größe initialisiert. Die Variable *Value* ist TRUE, wenn die Checkbox aktiv ist, sonst FALSE.

x0, y0 Die linke obere Ecke des Dialogelements relativ zur linken oberen Ecke der Dialogfläche. Wird für einen der Werte der Wert *IsAuto* angegeben, wird das Element gegenüber der Dialogfläche zentriert.

nx, ny Die Größe des Dialogelements. Hat *nx* den Wert *IsAuto*, wird das Element so breit, daß der Text voll lesbar ist. Hat *ny* den Wert *IsAuto*, wird das Element so hoch, daß eine Textzeile darin Platz hat.

text Der Text, mit dem das Dialogelement beschriftet wird. Dieser Parameter ist optional. Er muß angegeben werden, wenn Flag-Bits angegeben werden.

flags Flag-Bits:

IsTriState 3-State-Button. Neben TRUE und FALSE kann das *Value*-Feld auch den Wert *TriState* (1) enthalten.

Events

ON ENTER (Variable) GOSUB Was das Dialogelement inaktiv und wird es angeklickt oder für Tastatureingaben aktiviert, wird auf den angegebenen GOSUB-Label verzweigt.

ON LEAVE (Variable) GOSUB War das Dialogelement aktiv und es wird ein anderes Element aktiviert, wird auf den angegebenen GOSUB-Label verzweigt.

ON CLICK (Variable) GOSUB Wird das Dialogelement angeklickt oder es wird die Leertaste gedrückt, wenn das Element aktiv ist, wird zum angegebenen GOSUB-Label verzweigt.

COMBOBOX

Deklaration einer Combobox.

Verwendung

```
COMBOBOX Variable, x0, y0, nx, ny [, Flags]
```

Beschreibung

Im aktuellen Dialog wird eine Combobox deklariert.

Parameter

Variable Combobox-Variable. Diese muß vom Type *ComboBox* sein. Lage und Größe werden mit den tatsächlichen Werten für Lage und Größe initialisiert. Die Variable *Value* enthält den Index des aktuellen Eintrags ab 1.

x0, y0 Die linke obere Ecke des Dialogelements relativ zur linken oberen Ecke der Dialogfläche. Wird für einen der Werte der Wert *IsAuto* angegeben, wird das Element gegenüber der Dialogfläche zentriert.

nx, ny Die Größe des Dialogelements. Haben *nx* oder *ny* den Wert *IsAuto*, wird jeweils der Wert 50 angenommen.

flags Flag-Bits:

HasBorder Die Combobox wird umrandet.
IsDropDown Die Combobox wird einzeilig dargestellt und kann bei Bedarf aufgeklappt werden.
IsSorted Die Einträge der Combobox werden alphabetisch sortiert.

Events

ON ENTER (Variable) GOSUB Was das Dialogelement inaktiv und wird es angeklickt oder für Tastatureingaben aktiviert, wird auf den angegebenen GOSUB-Label verzweigt.

ON LEAVE (Variable) GOSUB War das Dialogelement aktiv und es wird ein anderes Element aktiviert, wird auf den angegebenen GOSUB-Label verzweigt.

ON CHANGE (Variable) GOSUB Ändert sich die Auswahl der Combobox, wird zum angegebenen GOSUB-Label verzweigt. Das Feld *Value* enthält den Index des ausgewählten Eintrags ab 1.

ON KEY (Variable) GOSUB Bei einer Tastatureingabe wird zum angegebenen GOSUB-Label verzweigt. Das Value-Feld enthält den Code der Taste. Die Taste kann ignoriert werden, indem das Feld Value auf 0 gesetzt wird.

DIALOG

Beginn des Aufbaus eines Dialogs.

Verwendung

```
DIALOG Variable, x0, y0, nx, ny [, Überschrift]
```

Beschreibung

Die DIALOG-Anweisung eröffnet den Aufbau eines Dialogs. Das Dialogfenster wird in der angegebenen Lage und Größe aufgebaut, wobei sich die Größenangabe auf den Ausgabebereich des Dialogfensters bezieht. Alle nach dieser Anweisung folgenden Anweisungen für Dialogelemente erscheinen in diesem Dialog. Der Dialog wird mit der STARTDIALOG-Anweisung aktiviert und mit der STOPDIALOG-Anweisung beendet.

Eine erneute DIALOG-Anweisung eröffnet einen neuen Dialog. Dieser bleibt so lange aktiv, bis er durch eine STOPDIALOG-Anweisung beendet worden ist. Es können somit mehrere Dialog ineinander verschachtelt werden.

Parameter

Variable	Die Dialog-Variable. Diese muß vom Typ *Dialog* sein. Der Inhalt der Variablen wird mit der tatsächlichen Position und Größe des Dialogs initialisiert.
x0, y0	Die linke obere Ecke des Dialogs. Wird für einen der Werte der Wert *IsAuto* angegeben, wird der Dialog gegenüber dem Applikations-Hauptfenster zentriert.
nx, ny	Die Größe des Ausgabebereichs des Dialogfensters. Haben *nx* oder *ny* den Wert *IsAuto*, wird jeweils der Wert 100 angenommen.
text	Die Überschrift des Dialogs. Dieser Parameter ist optional. Er muß angegeben werden, wenn Flag-Bits angegeben werden.

EDITFIELD

Deklaration eines Eingabefeldes.

Verwendung

```
EDITFIELD Variable, x0, y0, nx, ny [, Text [, Flags]]
```

Beschreibung

Im aktuellen Dialog wird ein Eingabefeld deklariert.

Parameter

Variable Edit-Variable. Diese muß vom Type *EditField* sein. Lage und Größe werden mit den tatsächlichen Werten für Lage und Größe initialisiert. Die Variable *Value* ist TRUE, wenn die Checkbox aktiv ist, sonst FALSE.

x0, y0 Die linke obere Ecke des Dialogelements relativ zur linken oberen Ecke der Dialogfläche. Wird für einen der Werte der Wert *IsAuto* angegeben, wird das Element gegenüber der Dialogfläche zentriert.

nx, ny Die Größe des Dialogelements. Hat *nx* den Wert *IsAuto*, wird das Element so breit, daß der Text voll lesbar ist. Hat *ny* den Wert *IsAuto*, wird das Element so hoch, daß eine Textzeile darin Platz hat.

text Der Text, mit dem das Dialogelement vorbesetzt wird. Dieser Parameter ist optional. Er muß angegeben werden, wenn Flag-Bits angegeben werden.

flags Flag-Bits:

HasBorder Das Eingabefeld wird umrandet.
IsMultiLine Mehrzeilige Eingaben sind möglich.
HasVScroll Mit vertikalem Scrollbar.
HasHScroll Mit horizontalem Scrollbar.

Events

ON ENTER (Variable) GOSUB Was das Dialogelement inaktiv und wird es angeklickt oder für Tastatureingaben aktiviert, wird auf den angegebenen GOSUB-Label verzweigt.

ON LEAVE (Variable) GOSUB War das Dialogelement aktiv und es wird ein anderes Element aktiviert, wird auf den angegebenen GOSUB-Label verzweigt.

ON CHANGE (Variable) GOSUB Nach jeder Änderung des Inhalts des Eingabefelds wird auf den angegebenen GOSUB-Label verzweigt.

ON KEY (Variable) GOSUB Bei einer Tastatureingabe wird zum angegebenen GOSUB-

Label verzweigt. Das *Value*-Feld enthält den Code der Taste. Die Taste kann ignoriert werden, indem das Feld *Value* auf 0 gesetzt wird.

ENDDIALOG

Beenden eines Dialogs.

Verwendung

```
ENDDIALOG
```

Beschreibung

Der aktuelle Dialog wird beendet. Alle internen Daten werden gelöscht, so daß auf die Elemente des Dialogs nicht mher zugegriffen werden kann.

ENDGROUP

Beenden einer Gruppen-Deklaration.

Verwendung

```
GROUPBOX ....
   ....
ENDGROUP
```

Beschreibung

Die Definition der aktuellen Gruppe wird abgeschlossen. Eine Gruppe beginnt mit einer GROUPBOX-Anweisung. Alle Steuerelemente, die nach einer GROUPBOX-Anweisung und vorder zugehörigen ENDGROUP-Anweisung deklariert werden, werden Teil dieser Gruppe.

ENDMENU

Beenden der Deklaration eines Menüs oder Untermenüs.

Verwendung

```
POPUPMENU ....
  ....
  SUBMENU ....
    ....
  ENDMENU
  ....
ENDMENU
```

Beschreibung

Die ENDMENU-Anweisung beendet die Deklaration des aktuellen Menüs oder Untermenüs. Für jede POPUPMENU- oder SUBMENU-Anweisung mollte eine ENDMENU-Anweisung verwendet werden. Wird der Dialog durch die STARTDIALOG-Anweisung aktiviert, werden alle offenen Menüdeklarationen beendet.

Querverweise

POPUPMENU, ENDMENU, SUBMENU, MENUITEM, SEPARATOR

FIXEDTEXT

Deklaration eines Textfeldes.

Verwendung

```
FIXEDTEXT Variable, x0, y0, nx, ny [, Text [, Flags]]
```

Beschreibung

Im aktuellen Dialog wird ein Textfeld deklariert.

Parameter

Variable	Textfeld-Variable. Diese muß vom Type *FixedText* sein. Lage und Größe werden mit den tatsächlichen Werten für Lage und Größe initialisiert. Die Variable *Value* ist 0.
x0, y0	Die linke obere Ecke des Dialogelements relativ zur linken oberen Ecke der Dialogfläche. Wird für einen der Werte der Wert *IsAuto* angegeben, wird das Element gegenüber der Dialogfläche zentriert.
nx, ny	Die Größe des Dialogelements. Hat *nx* den Wert *IsAuto*, wird das Element so breit, daß der Text voll lesbar ist. Hat *ny* den Wert *IsAuto*, wird das Element so hoch, daß eine Textzeile darin Platz hat.
text	Der Text, mit dem das Dialogelement beschriftet wird. Dieser Parameter ist optional. Er muß angegeben werden, wenn Flag-Bits angegeben werden.
flags	Flag-Bits:
IsLeft	Der Text wird linksbündig ausgegeben (Voreinstellung).
IsCenter	Der Text wird zentriert ausgegeben.
IsRight	Der Text wird rechtsbündig ausgegeben.
HasBorder	Das Textfeld wird umrandet.

Events

ON CLICK (Variable) GOSUB	Wird das Dialogelement angeklickt, wird zum angegebenen GOSUB-Label verzweigt.

GROUPBOX

Deklaration eines Gruppenfeldes.

Verwendung

```
GROUPBOX Variable, x0, y0, nx, ny [, Text]
...
ENDGROUP
```

Beschreibung

Es wird ein Gruppenfeld deklariert. Alle nachfoglenden Steuerelemente werden als Teil dieses Gruppenfeldes angesehen. Die Anweisung ENDGROUP beendet das Gruppenfeld.

Parameter

Variable	Gruppenfeld-Variable. Diese muß vom Type *GroupBox* sein. Lage und Größe werden mit den tatsächlichen Werten für Lage und Größe initialisiert. Die Variable *Value* ist 0.
x0, y0	Die linke obere Ecke des Dialogelements relativ zur linken oberen Ecke der Dialogfläche. Wird für einen der Werte der Wert *IsAuto* angegeben, wird das Element gegenüber der Dialogfläche zentriert.
nx, ny	Die Größe des Dialogelements. Haben *nx* oder *ny* den Wert *IsAuto*, wird wird jeweils der Wert 50 angenommen.
text	Der Text, mit dem das Dialogelement beschriftet wird. Dieser Parameter ist optional.

Events

Ein Gruppenfeld kann keine eigenen Events verarbeiten.

Anmerkungen

Die Deklaration einer Gruppe muß erst mit der ENDGROUP-Anweisung abgeschlossen sein, ehe eine neue Gruppe deklariert werden kann.

ICON

Deklaration eines Icons.

Verwendung

```
ICON Variable, x0, y0, Dateiname [, Flags]
```

Beschreibung

Im aktuellen Dialog wird ein Icon deklariert. Eine Größenangabe ist nicht möglich, da die Größe eines Icons vom System vorgegeben ist.

Parameter

Variable	Icon-Variable. Diese muß vom Type *Icon* sein. Lage und Größe werden mit den tatsächlichen Werten für Lage und Größe initialisiert. Die Variable *Value* ist 0.
x0, y0	Die linke obere Ecke des Dialogelements relativ zur linken oberen Ecke der Dialogfläche. Wird für einen der Werte der Wert *IsAuto* angegeben, wird das Element gegenüber der Dialogfläche zentriert.
Dateiname	Der Name der Datei, die das Icon enthält. Für Standard-Icons sind spezielle Strings vordefiniert.
flags	Flag-Bits:
HasBorder	Das Icon wird umrandet.

Standard-Icons

DefaultIcon	Icon, das vom System für Applikationen verwendet wird, die kein eigenes Icon haben.
InfoIcon	Informations-Icon (normalerweise ein Ausrufezeichen).
QueryIcon	Frage-Icon (normalerweise ein Fragezeichen).
WarnIcon	Icon, das für Warnungen verwendet wird.
ErrorIcon	Icon, das für Fehlermeldungen verwendet wird.

Events

ON CLICK (Variable) GOSUB	Wird das Dialogelement angeklickt, wird zum angegebenen GOSUB-Label verzweigt.

Anmerkungen

Die Methode *GetText()* liefert einen Leerstring zurück; die Methode *SetText()* wird ignoriert.

LISTBOX

Deklaration einer Listbox.

Verwendung

```
LISTBOX Variable, x0, y0, nx, ny [, Flags]
```

Beschreibung

Im aktuellen Dialog wird eine Listbox deklariert.

Parameter

Variable	Listbox-Variable. Diese muß vom Type *ListBox* sein. Lage und Größe werden mit den tatsächlichen Werten für Lage und Größe initialisiert. Die Variable *Value* enthält den Index des aktuellen Eintrags ab 1.
x0, y0	Die linke obere Ecke des Dialogelements relativ zur linken oberen Ecke der Dialogfläche. Wird für einen der Werte der Wert *IsAuto* angegeben, wird das Element gegenüber der Dialogfläche zentriert.
nx, ny	Die Größe des Dialogelements. Haben *nx* oder *ny* den Wert *IsAuto*, wird der Wert 50 angenommen.
flags	Flag-Bits:
HasBorder	Die Listbox wird umrandet.
IsDropDown	Die Listbox wird einzeilig dargestellt und kann bei Bedarf aufgeklappt werden.
IsSorted	Die Einträge der Listbox werden alphabetisch sortiert.

Events

ON ENTER (Variable) GOSUB	Was das Dialogelement inaktiv und wird es angeklickt oder für Tastatureingaben aktiviert, wird auf den angegebenen GOSUB-Label verzweigt.
ON LEAVE (Variable) GOSUB	War das Dialogelement aktiv und es wird ein anderes Element aktiviert, wird auf den angegebenen GOSUB-Label verzweigt.
ON CHANGE (Variable) GOSUB	Ändert sich die Auswahl der Listbox, wird zum angegebenen GOSUB-Label verzweigt. Das Feld *Value* enthält den Index des ausgewählten Eintrags ab 1.

Anmerkungen

Die Methoden *GetText()* und *SetText()* lesen den Text des aktuellen Eintrags aus bzw. ändern den aktuellen Eintrag in den übergebenen Text. Dabei wird der in *Value* gespeicherte Wert als aktueller Eintrag verwendet.

MENUITEM

Deklaration eines Menüeintrags.

Verwendung

```
POPUPMENU ....
  ....
  MENUITEM text, id [, flags]
  ....
ENDMENU
```

Beschreibung

Im aktuellen Menü wird ein Menüeintrag deklariert.

Parameter

text	Der Text, mit dem der Eintrag im aktuellen Menü erscheint.
id	Das ID des Eintrags. Der Eintrag wird über diesen Wert identifiziert.
flags	Optionale Flag-Bits:
IsChecked	Der Eintrag wird mit einem Häkchen versehen.
IsDisabled	Der Eintrag ist disabled. Er wird grau dargestellt und kann nicht angewählt werden.

Events

Alle Menü-Events werden über die Menü-Variable des Menüs abgewickelt.

Anmerkungen

Ein Menüeintrag kann auch aus einer Bitmap bestehen. Dazu ist die BITMAPITEM-Anweisung zu verwenden. Trennlinien werden mit der SEPARATOR-Anweisung erzeugt.

Querverweise

POPUPMENU, ENDMENU, SUBMENU, MENUITEM, SEPARATOR

POPUPMENU

Deklaration eines Menüs.

Verwendung

```
POPUPMENU Variable, x0, y0, nx, ny, text
  ....
ENDMENU
```

Beschreibung

Im aktuellen Dialog wird ein Menü deklariert. Das Menü wird nicht in einer Menüleiste, sonder als spezieller Button eingerichtet. Wird der Button angeklickt, erscheint das definierte Menü. Die Parameter der POPUPMENU-Anweisung definieren das Aussehen dieses Buttons. Die Deklaration des Menüs wird mit einer ENDMENU-Anweisung abgeschlossen. Wird der Dialog drch die STARTDIALOG-Anweisung aktiviert, wird das Menü automatisch abgeschlossen. Es können beliebig viele Menüs definiert werden.

Parameter

Variable Menü-Variable. Diese muß vom Type *PopupMenu* sein. Lage und Größe werden mit den tatsächlichen Werten für Lage und Größe initialisiert. Die Variable *Value* enthält die ID-Nummer des aktivierten Items, wenn der CHANGE-Handler gerufen wird.

x0, y0 Die linke obere Ecke des Dialogelements relativ zur linken oberen Ecke der Dialogfläche. Wird für einen der Werte der Wert *IsAuto* angegeben, wird das Element gegenüber der Dialogfläche zentriert.

nx, ny Die Größe des Dialogelements. Hat *nx* den Wert *IsAuto*, wird das Element so breit, daß der Text voll lesbar ist. Hat *ny* den Wert *IsAuto*, wird das Element so hoch, daß eine Textzeile darin Platz hat.

text Der Text, mit dem das Dialogelement beschriftet wird. Dieser Parameter ist optional. Er muß angegeben werden, wenn Flag-Bits angegeben werden.

Events

ON ENTER (Variable) GOSUB Was das Dialogelement inaktiv und wird es angeklickt oder für Tastatureingaben aktiviert, wird auf den angegebenen GOSUB-Label verzweigt.

ON LEAVE (Variable) GOSUB War das Dialogelement aktiv und es wird ein anderes Element aktiviert, wird auf den angegebenen GOSUB-Label verzweigt.

ON CLICK (Variable) GOSUB Wird das Dialogelement angeklickt oder es wird die Leertaste gedrückt, wenn das Element aktiv ist, wird zum angegebenen GOSUB-Label verzweigt. Dort kann das Menü initialisiert werden, ehe es angezeigt wird.

ON CHANGE (Variable) GOSUB Wird ein Menüpunkt angewählt, wird das Feld

Value mit dem ID des Elements besetzt und es wird zum angegebenen GOSUB-Label verzweigt.

PUSHBUTTON

Deklaration eines Pushbuttons (Schaltfläche).

Verwendung

```
PUSHBUTTON Variable, x0, y0, nx, ny [, Text [, Flags]]
```

Beschreibung

Im aktuellen Dialog wird ein Pushbutton deklariert.

Parameter

Variable Button-Variable. Diese muß vom Type *PushButton* sein. Lage und Größe werden mit den tatsächlichen Werten für Lage und Größe initialisiert. Die Variable *Value* ist TRUE, wenn der Pushbutton aktiv ist, sonst FALSE.

x0, y0 Die linke obere Ecke des Dialogelements relativ zur linken oberen Ecke der Dialogfläche. Wird für einen der Werte der Wert *IsAuto* angegeben, wird das Element gegenüber der Dialogfläche zentriert.

nx, ny Die Größe des Dialogelements. Hat *nx* den Wert *IsAuto*, wird das Element so breit, daß der Text voll lesbar ist. Hat *ny* den Wert *IsAuto*, wird das Element so hoch, daß eine Textzeile darin Platz hat.

text Der Text, mit dem das Dialogelement beschriftet wird. Dieser Parameter ist optional. Er muß angegeben werden, wenn Flag-Bits angegeben werden.

flags Flag-Bits:

IsDefault Der Pushbutton wird automatisch aktiviert, wenn die ENTER-Taste gedrückt wird.

IsCancel Der Pushbutton wird automatisch aktiviert, wenn das Dialogfenster geschlossen wird oder die ESC-Taste gedrückt wird.

flags Flag-Bits:

HasBorder Das Eingabefeld wird umrandet.

Events

ON ENTER (Variable) GOSUB Was das Dialogelement inaktiv und wird es angeklickt oder für Tastatureingaben aktiviert, wird auf den angegebenen GOSUB-Label verzweigt.

ON LEAVE (Variable) GOSUB War das Dialogelement aktiv und es wird ein anderes Element aktiviert, wird auf den angegebenen GOSUB-Label verzweigt.

ON CLICK (Variable) GOSUB Wird das Dialogelement angeklickt oder es wird die Leertaste gedrückt, wenn das Element aktiv ist, wird zum angegebenen GOSUB-Label verzweigt.

RADIOBUTTON

Deklaration eines Radiobuttons.

Verwendung

```
RADIOBUTTON Variable, x0, y0, nx, ny [, Text]
```

Beschreibung

Im aktuellen Dialog wird ein Radiobutton deklariert. Ein derartiger Button arbeitet mit anderen Radiobuttons in einer Gruppe derart zusammen, daß, wenn ein Radiobutton aktiviert wird, alle anderen Radiobuttons der Gruppe automatisch deaktiviert werden.

Parameter

Variable Button-Variable. Diese muß vom Type *RadioButton* sein. Lage und Größe werden mit den tatsächlichen Werten für Lage und Größe initialisiert. Die Variable *Value* ist TRUE, wenn der Radiobutton aktiv ist, sonst FALSE.

x0, y0 Die linke obere Ecke des Dialogelements relativ zur linken oberen Ecke der Dialogfläche. Wird für einen der Werte der Wert *IsAuto* angegeben, wird das Element gegenüber der Dialogfläche zentriert.

nx, ny Die Größe des Dialogelements. Hat *nx* den Wert *IsAuto*, wird das Element so breit, daß der Text voll lesbar ist. Hat *ny* den Wert *IsAuto*, wird das Element so hoch, daß eine Textzeile darin Platz hat.

text Der Text, mit dem das Dialogelement beschriftet wird. Dieser Parameter ist optional. Er muß angegeben werden, wenn Flag-Bits angegeben werden.

flags Flag-Bits:

HasBorder Das Eingabefeld wird umrandet.

Events

ON ENTER (Variable) GOSUB Was das Dialogelement inaktiv und wird es angeklickt oder für Tastatureingaben aktiviert, wird auf den angegebenen GOSUB-Label verzweigt.

ON LEAVE (Variable) GOSUB War das Dialogelement aktiv und es wird ein anderes Element aktiviert, wird auf den angegebenen GOSUB-Label verzweigt.

ON CLICK (Variable) GOSUB Wird das Dialogelement angeklickt oder es wird die Leertaste gedrückt, wenn das Element aktiv ist, wird zum angegebenen GOSUB-Label verzweigt.

SCROLLBAR

Deklaration eines Scrollbars.

Verwendung

```
SCROLLBAR Variable, x0, y0, nx, ny [, Flags]
```

Beschreibung

Im aktuellen Dialog wird ein Scrollbar deklariert.

Parameter

Variable	Scrollbar-Variable. Diese muß vom Type *ScrollBar* sein. Lage und Größe werden mit den tatsächlichen Werten für Lage und Größe initialisiert. Die Variable *Value* enthält die aktuelle Position des Scrollbars.
x0, y0	Die linke obere Ecke des Dialogelements relativ zur linken oberen Ecke der Dialogfläche. Wird für einen der Werte der Wert *IsAuto* angegeben, wird das Element gegenüber der Dialogfläche zentriert.
nx, ny	Die Größe des Dialogelements. Hat *ny* den Wert *IsAuto*, wird jeweils der Wert 15 angenommen.
flags	Flag-Bits:
HasBorder	Die Listbox wird umrandet.
IsDropDown	Die Listbox wird einzeilig dargestellt und kann bei Bedarf aufgeklappt werden.
IsSorted	Die Einträge der Listbox werden alphabetisch sortiert.

Events

ON ENTER (Variable) GOSUB	Was das Dialogelement inaktiv und wird es angeklickt oder für Tastatureingaben aktiviert, wird auf den angegebenen GOSUB-Label verzweigt.
ON LEAVE (Variable) GOSUB	War das Dialogelement aktiv und es wird ein anderes Element aktiviert, wird auf den angegebenen GOSUB-Label verzweigt.
ON CHANGE (Variable) GOSUB	Ändert sich die Position des Scrollbars, wird zum angegebenen GOSUB-Label verzweigt. Das Feld *Value* enthält die aktuelle Position des Scrollbars.

Anmerkungen

Die Methode *GetText()* liefert einen Leerstring zurück; die Methode *SetText()* wird ignoriert.

SEPARATOR

Erzeugen einer Trennlinie in einem Menü.

Verwendung

```
POPUPMENU ....
  ....
  SEPARATOR
  ....
ENDMENU
```

Beschreibung

Diese Anweisung erzeugt eine horizontale Trennlinie im ektuellen Menü.

Querverweise

POPUPMENU, ENDMENU, SUBMENU, MENUITEM, SEPARATOR

STARTDIALOG

Aktivieren eines Dialogs.

Verwendung

```
STARTDIALOG
```

Beschreibung

Der aktuelle Dialog wird aktiviert. Sämtliche offenen Gruppen und Menüs werden vorher geschlossen. Die vorher definierten Event-Handler werden aktiviert.

Der Dialog wird durch die ENDDIALOG-Anweisung beendet. Diese wird in einem Event-handler ausgeführt. Anschließend wird der Programmlauf hiner der STARTDIALOG-Anweisung fortgesetzt.

SUBMENU

Deklaration eines Untermenüs.

Verwendung

```
POPUPMENU ....
  ....
  SUBMENU text, id
    ....
  ENDMENU
  ....
ENDMENU
```

Beschreibung

Im aktuellen Menü wird ein Submenü deklariert. Dieses Submenü kann Menüpunkte, weitere Submenüs und Separatoren enthalten. Die Deklaration eines Submenüs wird mit einer ENDMENU-Anweisung abgeschlossen.

Parameter

text	Der Text, mit dem das Submenü im aktuellen Menü erscheint.
id	Das ID des Submenüs. Auch, wenn Submenüs niemals eigene Events erzeugen, muß es ein ID erhalten. Über dieses ID kann es disabled werden. Das Markieren eines Submenüs ist nicht möglich.

Querverweise

POPUPMENU, ENDMENU, SUBMENU, MENUITEM, SEPARATOR

StarBASIC-Dialoge - Einführung

Benötigte Dateien

StarBASIC stellt die Möglichkeit zur Verfügung, vollständige Dialoge zu programmieren. Für diese Programmierung müssen folgende Dateien vorhanden sein:

DIALOG.INC - eine Include-Datei mit den StarBASIC-Definitionen.
SBDIALOG.DLL - der benötigte Runtime-Code.

Der Runtime-Code liegt unter Windows (und später OS/2) als DLL vor, auf anderen Systemen kann die Datei anders heißen oder auch gänzlich fehlen.

Aufbau der Datei DIALOG.INC

Die Datei DIALOG.INC enthält alle Definitionen, die zur Programmierung von Dialgoen notwendig sind. Jedes Dialogelement liegt in Form einer Klassendefinition vor, wobei alle Elemente inner halb eines Klassenbaums definiert sind. Ein Dialogelement enthält Variable (wie Lage und Größe) wosie einige Methoden, mit denen zusätzliche Eigenschaften definiert oder abgefragt werden können. Auch der Dialog selbst ist eine derartige Variablen-Definition.

Neben den Klassendefinitionen sind die Dialoganweisungen sowie die Event-Handler definiert.

Detailierte Informationen finden Sie unter Dialogaufbau, Events, Event Registrierung, Dialogelemente, Statische Elemente, Buttons, Listboxen, Eingabefelder, Scrollbars, Gruppenfelder, Menüs, Anweisungen, Einschränkungen

Aufbau eines Dialogs

Vor dem Aufbau eines Dialogs muß für jedes im Dialog enthaltene Element eine StarBASIC-Variable mit Hilfe der DIM-Anweisung definiert werden. Bei Bedarf können eigene Variable von den vorhandenen Klassen abgeleitet werden.

Der Dialog selbst wird mit einer Reihe von Anweisungen aufgebaut. Diese Anweisungen haben den gleichen Namen wie die dazugehörigen Dialogelemente und erwarten ein Dialogelement als ersten Parameter. Die meisten dieser Anweisungen haben zusätzliche Parameter, die Lage, Größe und Text des Elements festlegen. Weiter können elementspezifisch Flag-Bits angegeben werden, die Aussehen und Verhalten des Elements steuern. Die Deklaration des Dialogs beginnt immer mit der DIALOG-Anweisung, die Lage und Größe des Dialogfensters festlegt. Anschließend folgen die einzelnen Dialogelemente. Jede Anweisung initialisiert das dazugehörige Element mit seinen Parametern.

Als Beispiel sei hier die Deklaration eines Buttons mit der Auschrift "OK" angeführt:

```
REM $INCLUDE: 'DIALOG.INC'

DIM MeinDialog AS DIALOG
DIM OKbutton AS PushButton

DIALOG MeinDialog, ......
  PUSHBUTTON OKbutton, 10, 10, 50, 15, "OK", IsDefault
  ....
```

Die Variable *OKbutton* wird dirch die PUSHBUTTON-Anweisung wie folgt initialisiert:

```
OKbutton.Internal = interne Adresse
OKbutton.X0 = 10
OKbutton.Y0 = 10
OKbutton.NX = 50
OKbutton.NY = 15
OKbutton.SetText ("OK")
```

Auf die Variable *Internal* kann von StarBASIC aus nicht zugegriffen werden. Sie enthält die Adresse des internen StarView-Objekts, das durch die Anweisung erzeugt wurde. Ein initialisiertes Element wird intern daran erkannt, ob diese Variable besetzt ist. Intern wird auch das Flag-Bit *IsDefault* ausgewertet. Durch dieses Bit wird ein Default-Pushbutton erzeugt, der aktiviert wird, wenn die ENTER-Taste gedrückt wird.

Die Felder für Lage und Größe können mit de*r* Konstanten *IsAuto* besetzt werden. Dadurch werden Lage und/oder Größe automatisch berechnet. Im Allgemeinen wird das Element zentriert und so groß gemacht, daß der Text vollständig sichtbar ist. Eine genaue Beschreibung der Wirkung von *IsAuto* findet sich in der Funktions-Referenz.

Nach der Deklaration und Initialisierung der Elemente können die Elemente noch durch Zuweisungen oder Methodenaufrufe verändert werden.

Vor der Aktivierung des Dialogs müssen noch die gewünschten Event-Handler registriert werden. Diese Event-Handler verarbeiten die Rückmeldungen des aktiven Dialogs.

Der Dialog selbst wird mit der STARTDIALOG-Anweisung aktiviert und zur Azeige gebracht. Diese Anweisung unterbricht den StarBASIC-Programmlauf so lange, bis in einem Event-Handler die ENDDIALOG-Anweisung aufgerufen wurde. Anschließend wird der StarBASIC-Programmlauf hinter der STARTDIALOG-Anweisung fortgesetzt.

Alle Maßeinheiten des Dialogs sind auf die Fontgröße bezogen, so daß der Dialog immer gleich aussieht, wenn ein anderer Systemfont verwendet wird. Als Font wird der durch die Applikation gesetzte Applikations-Font verwendet.

Events

Wenn ein Dialog aktiviert wird, wird der Programmlauf von StarBASIC so lange angehalten, wie der Dialog existiert. Alle Änderungen innerhalb des Dialogs lösen Events aus. Diese Events können in StarBASIC mit der ON...GOSUB-Anweisung registriert werden. Typische Events sind beispielsweise

- der Wechsel des Input Focus innerhalb des Dialogs,
- die Eingabe eines Textes in einem Editfeld oder
- das Anklicken eines Buttons.

Für jedes Dialogelement kann eine ganz bestimmte Anzahl von Events definiert werden. Die Art dieser Events hängt vom Typ des Elements ab. Eine Bitmap oder ein Text ist normalerweise statisch; er reagiert jedoch auf das Anklicken durch die Maus, daher kann auch ein Click-Event registriert werden. Ein Button kann auch mit der Tab-Taste angesprungen werden; dafür können zusätzlich Events für den Erhalt und den Verlust des Focus registriert werden.

Registrierung

Die Registrierung eines Events erfolgt mit der Anweisung

```
ON Eventname (Element-Variable) GOSUB Label
```

Diese Anweisung muß zwischen der Initialisierung des Elements und der Aktivierung des Dialogs erfolgen. Beispiel:

```
REM $INCLUDE 'DIALOG.INC'

DIM Dlg AS DIALOG, OK AS Button

DIALOG Dlg IsAuto, IsAuto, 50, 50, "Test"
  PUSHBUTTON OK, 10, 15, 30, 15, "OK"

ON CLICK (OK) GOSUB OKhandler

STARTDIALOG
END

OKhandler:
  ENDDIALOG : RETURN
```

Der Eventhandler *OKhandler* wird nach der Initialisierung der Variable *OK* durch die *PUSHBUTTON*-Anweisung, jedoch vor der *STARTDIALOG*-Anweisung registriert, wobei explizit das *CLICK*-Event registriert wird.

Aktivierung

Ein Event wird durch eine Aktion aktiviert, die mit dem betroffenen Element durchgeführt wird. Im obigen Beispiel wird der Handler aktiviert, wenn der OK-Button angeklickt wird. Durch diese Aktivierung wird das Unterprogramm *OKhandler* angesprungen. Dort kann (fast) jede beliebige Aktion durchgeführt werden. Wird die *RETURN*-Anweisung ausgeführt, wird der StarBASIC-Programmlauf beendet und der Dialog wird fortgesetzt.

Da der Dialog erst durch eine *ENDDIALOG*-Anweisung beendet wird, muß ein Event-Handler diese Anweisung enthalten, da der Dialog sonst immer aktiv bleibt. Der normale StarBASIC-Programmlauf setzt erst wieder ein, wenn der Dialog beendet ist. Das Programm wird dann hinter der *STARTDIALOG*-

Anwesigung fortgesetzt.

Innerhalb eines Event-Handler kann auf jedes Dialogelement zugegriffen werden, um Werte abzufragen oder zu ändern. Jede Änderung eines Wertes wird unmittelbar im Dialog angezeigt.

Event-Arten

Für Dialogelemente stehen folgende Events zur Verfügung:

`CLICK`

Ein CLICK-Event wird bei Buttons und bei statischen Elements wie Icons, Bitmaps oder Texten ausgelöst, wenn das Element angeklickt wird. Bei Buttons kann das Element auch mit Hilfe der Cursor-Steuertasten angewählt und typischerweise mit der Leertaste "angeklickt" werden.

`ENTER`

Das ENTER-Event wird jedesmal dann ausgelöst, wenn das Element den Focus erhält. Dieses Event kann für alle Elemente außer den statischen Elementsn Bitmap, Icon und FixedText registriert werden.

`LEAVE`

Das LEAVE-Event wird jedesmal dann ausgelöst, wenn das Element den Focus verliert. Dieses Event kann für alle Elemente außer den statischen Elementsn Bitmap, Icon und FixedText registriert werden.

`CHANGE`

Das CHANGE-Event wird ausgelöst, wenn sich der Inhalt oder die Selektion eines Elements ändert. Dies gilt für alle Elements außer den verschiedenen Buttons und den statischen Elementen.

`KEY`

Das KEY-Event wird bei Texteingaben ausgelöst, ehe die eingegebene Taste dem Element zugeführt wird. Der Event-Handler hat dabei die Möglichkeit, den Tastencode zu ignorieren. Dieses Event kann bei Editfeldern und Comboboxen registriert werden.

Mögliche Event-Registrierungen

Die folgende Tabelle zeigt im Überblick, welche Events für welche Elemente registriert werden können.

Element	ENTER	LEAVE	CLICK	KEY	CHANGE
Bitmap	---	---	Ja	---	---
Icon	---	---	Ja	---	---
FixedText	---	---	Ja	---	---
PushButton	Ja	Ja	Ja	---	---
CheckBox	Ja	Ja	Ja	---	---
RadioButton	Ja	Ja	Ja	---	---
ListBox	Ja	Ja	---	---	Ja
ComboBox	Ja	Ja	---	Ja	Ja
EditField	Ja	Ja	---	Ja	Ja
PopupMenu	Ja	Ja	Ja	---	Ja
ScrollBar	Ja	Ja	---	---	Ja
GroupBox	---	---	---	---	---

Dialogelemente

Dieser Abschnitt behandelt die möglichen Dialogelemente und die mit ihnen verbundenen StarBASIC-Anweisungen. Die Beschreibung der Elemente selbst ist der Klassenreferenz zu entnehmen; die Funktionsreferenz enthält eine eingehende Beschreibung der StarBASIC-Anweisungen.

Alle Dialogelemente können optional umrandet werden, indem das Flagbit *HasBorder* angegeben wird.

Statische Elemente

Statische Elemente können nicht mit Hilfe der Cursor-Steuertasten angewählt werden; sie können jedoch mit der Maus angeklickt werden.

`Bitmaps`

Variablenklasse: Bitmap
Anweisung: BITMAP

Eine Bitmap wird aus einer Datei geladen und angezeigt. Sie kann durch nachträgliche Änderung ihrer Größe beliebig gedehnt und gestaucht werden.

`Icons`

Variablenklasse: Icon
Anweisung: ICON

Ein Icon wird aus einer Datei geladen und angezeigt. Im Gegensatz zu Bitmaps ist die Größe eines Icons systemabhängig fest vorgegeben. In DIALOG.INC sind einige Konstante definiert, mit denen fest im System eingebaute Icons geladen werden können.

`Texte`

Variablenklasse: FixedText
Anweisung: FIXEDTEXT

Dies ist ein Textfeld, in dem der angegebene Text angezeigt wird. Der Text kann linksbündig (Bit *IsLeft*), rechtsbündig (*IsRight*) oder zentriert (*IsCenter*) ausgegeben werden.

Buttons

`Einfache Buttons`

Variablenklasse: PushButton
Anweisung: PUSHBUTTON

Der normale Button kann durch das Flagbit *IsDefault* als Default-Button deklariert werden, der immer dann aktiviert wird, wenn die ENTER-Taste gedrückt wird. Das Flagbit *IsCancel* zeigt den Button an, der aktiviert wird, wenn die ESC-Taste gedrückt wurde oder der Dialog durch Aktivierung eines Fensterelements (z.b. das Systemmenü bei Windows) geschlossen wurde.

`Checkboxes`

Variablenklasse: CheckBox
Anweisung: CHECKBOX

Die Checkbox hat das Flagbit *IsTristate*. Dieses Bit deklariert die Checkbox als Tri-State-Box, die neben dem Status EIN und AUS auch einen dritten Status (*TriState*) zuläßt. Der aktuelle Status ist im Feld *Value* der Variablen gespeichert.

`Radiobuttons`

Variablenklasse: RadioButton
Anweisung: RADIOBUTTON

Der Radiobutton sorgt innerhalb einer Gruppe dafür, daß immer nur ein einziger Radiobutton aktiv ist. Wird ein Radiobutton aktiviert, werden alle anderen Radiobuttons deaktiviert. Der aktuelle Status ist im Feld *Value* der Variablen gespeichert.

Listboxes

`Einfache Listbox`

Variablenklasse: ListBox
Anweisung: LISTBOX

Die Listbox hat zwei Flagbits. Das Bit *IsSorted* erzeugt eine Listbox, deren Einträge sortiert sind. Das Bit *IsDropDown* erzeugt eine Drop-Down-Listbox. Die Einträge werden über Methoden verwaltet. Das Feld *Value* enthält bei der Listbox den Index des aktuellen Eintrags. Dieser Index läuft ab 1, d.h. der erste Eintrag der Listbox entspricht dem Index 1. Der Wert 0 zeigt an, daß kein Eintrag angewählt ist.

`Combobox`

Variablenklasse: ComboBox
Anweisung: COMBOBOX

Die Combobox enthält zusätzliche zur Listbox ein Eingabefeld, in das Einträge manuell eingetippt werden können. Die Klasse ist von der Klasse *ListBox* abgeleitet, so daß alles, was für die Listbox gilt, auch für die Combobox gilt. Beim KEY-Event enthält das Feld *Value* den Tastencode der Eingabe. Soll der Tastencode ignoriert werden, kann das Feld *Value* im Event-Handler auf 0 gesetzt werden.

Eingabefelder

Variablenklasse: EditField
Anweisung: EDITFIELD

Das Eingabefeld hat eine ganze Reihe von Flagbits. Der Text kann linksbündig (Bit *IsLeft*), rechtsbündig (*IsRight*) oder zentriert (*IsCenter*) ausgegeben werden. Das Bit *IsMultiline* definiert ein mehrzeiliges Eingabefeld. Zusätzlich können noch horizontale (*HasHScroll*) und vertikale (*HasVScroll*) Scrollbars bei mehrzeiligen Editfeldern definiert werden. Das Feld *Value* enthält beim KEY-Event den Tastencode der Eingabe. Soll der Tastencode ignoriert werden, kann das Feld *Value* im Event-Handler auf 0 gesetzt werden.

Scrollbars

Variablenklasse: ScrollBar
Anweisung: SCROLLBAR

Ein Scrollbar hat drei Flagbits. Das Bit *IsHorizontal* ist ein Dummy-Wert, da als Default immer ein horizontaler Scrollbar erzeugt wird. Das Bit *IsVertical* erzeugt einen vertikalen Scrollbar. Das Bit *IsSynchronous* steuert die Aktivierung des Events CHANGE. Normalerweise wird das CHANGE-Event erst aktiviert, wenn die Bedienung des Scrollbars abgeschlossen ist. Ist dieses Bit jedoch gesetzt und wird der Positionsanzeiger des Scrollbars mit der Maus gezogen, wird das CHANGE-Event bei jeder Änderung der Position aufgerufen. Das Feld *Value* enthält die aktielle Position.

Gruppenfelder

Variablenklasse: GroupBox
Anweisungen: GROUPBOX, ENDGROUP

Ein Gruppenfeld wird mit der GROUPBOX-Anweisung deklariert. Alle Elemente, die zu einer Gruppe zusammengefaßt werden, folgen nach dieser Anweisung. Die Gruppe wird mit der ENDGROUP-Anweisung abgeschlossen. Gruppen-Deklarationen können nicht verschachtelt sein.

Menüs

Variablenklasse: PopupMenu
Anweisungen: POPUPMENU, SUBMENU, MENUITEM, BITMAPITEM, SEPARATOR, ENDMENU

Ein Popup-Menü wird als spezieller Button implementiert. Wird dieser Button aktiviert, wird das Menü aufgeklappt. Der CLICK-Handler wird beim Klicken des Buttons aktiviert; hier ist der richtige Ort, um das Menü zu initialisieren. Der CHANGE-Handler wird aufgerufen, wenn ein Menüpunkt ausgewählt wurde. In diesem Fall enthält das Feld *Value* den ID des angewählten Menüpunkts. Die Parameter der POPUPMENU-Anweisung beschreiben Lage, Größe und Text des Menü-Buttons.

Der Aufbau des Menüs selbst geschieht mit einer Reihe von Anweisungen. Ein Menüeintrag wird mit der MENUITEM-Anweisung defineirt. Diesr Eintrag kann auch aus einr Bitmap bestehen; dafür ist die BITMAPITEM-Anweisung zu verwenden. EIn horizontaler Trennstrich wird mit der SEPARATOR-Anweisung erzeugt. Die Deklaration eines Menüs wird mit der ENDMENU-Anweisung abgeschlossen.

Ein Menü kann Untermenüs enthalten, die wiederum Untermenüs enthalten können. Die Schachtelungstiefe ist dabei fast unbegrenzt. Ein Untermenü beginnt mit der SUBMENU-Anweisung und wird ebenfalls mit der ENDMENU-Anweisung beendet.

Jeder Menüeintrag erhält einen numerischen ID, der bei der Auswahl im Feld *Value* der Menü-Variablen zurückgeliefert wird. Einzelne Items können mit Hifle des Flagbits *IsChecked* mit einem Häkchen versehen und mit dem Bit *IsDisabled* grau dargestellt werden. Diese beiden Eigenschaften eines Items können auch mit Methodenaufrufen der Klasse *PopupMenu* jederzeit umdefiniert werden.

Das folgende Beispiel zeigt einen typischen Menüaufbau:

```
DIM MeinMenu AS PopupMenu
DIALOG .....
  POPUPMENU  MeinMenu, 20, 30, 40, 20, "Mein Menü"
    MENUITEM     "Menue-Item ~1", 1001
    MENUITEM     "Menue-Item ~2", 1002, IsDisabled
    MENUITEM     "Menue-Item ~3", 1003
    SEPARATOR
    BITMAPITEM   "filename.bmp",  1004
    SEPARATOR
    SUBMENU      "Popup-Menue ~A", 1010
      MENUITEM   "Popup-Submenue ~1", 1011
      MENUITEM   "Popup-Submenue ~2", 1012
      MENUITEM   "Popup-Submenue ~3", 1013
      MENUITEM   "Popup-Submenue ~4", 1014
    ENDMENU
    SEPARATOR
    MENUITEM     "Menue-Item 5", 1005
  ENDMENU
```

Dialog-Anweisungen

Variablenklasse: Dialog
Anweisungen: DIALOG, STARTDIALOG, ENDDIALOG

Die DIALOG-Anweisung beginnt die Deklaration eines Dialogs.

Nach der Deklaration der Variablen sowie der Registrierung der Event-Handler kann der Dialog mit der STARTDIALOG-Anweisung aktiviert werden. Diese Anweisung bringt den Dialog zur Anzeige. Der StarBASIC-Programmlauf wird erst fortgesetzt, wenn der Dialog beendet ist. Alle noch offenen Gruppen- oder Menüdeklarationen werden vorher abgeschlossen.

Die ENDDIALOG-Anweisung beendet den Dialog wieder. Sie muß innerhalb einer der Event-Handler aufgerufen werden.

Dialoge können ineinander verschachtelt sein. So kann ein Dialog kommplett innerhalb eines StarBASIC-Unterprogramms abgewickelt werden; dieses Unterprogramm kann inerhalb eines Event-Handlers aufgerufen werden.

Einschränkungen

Die aktuelle Implementation der Dialoge hat einige Einschränkungen, die in einer künftigen Version von StarView behoben werden.

1. Das Flagbit *IsCancel* bei PushButtons wird ignoriert.

2. Der Menü-Button wird zur Zeit nur als normaler Button dargestellt.

Übersicht der Makrofunktionen

Gruppe Datei

ChangePrinter(Name$) AS INTEGER

Dem aktuellen Dokument wird ein Drucker zugeordnet. Es wird FALSE zurückgegeben, falls der Drucker nicht gewechselt werden konnte, weil z. B. ein Druckvorgang noch nicht abgeschlossen war.

Bezeichner	Typ	Beschreibung
Name$	*STRING*	*Druckername mit Anschluß*
RETURN	*BOOL*	*TRUE, der Drucker konnte zugeordnet werden* *FALSE, Fehler beim Zuordnen des Druckers*

CloseFile() AS INTEGER

Aktuelles Dokument schließen. Der Returnwert (BOOL) bezeichnet Erfolg oder Mißerfolg.

Bezeichner	Typ	Beschreibung
RETURN	*BOOL*	*TRUE, das Dokument wurde geschlossen* *FALSE, das Dokument kann im Augenblick nicht geschlossen werden*

DocManagerDlg()

Ruft den Dokument-Manager auf.

ExitWriter()

Beendet den StarWriter. Wenn noch bearbeitete Dokumente offen sind, die noch nicht gespeichert wurden, kommt eine Abfrage, ob Sie diese speichern wollen.

ExpandGlossary(OPTIONAL Group$)

Expandiert einen Textbaustein. Optional kann die Gruppe angegeben werden.

MergeFile(OPTIONAL Select$)

Ruft Serienbriefdruck auf.

Bezeichner	Typ	Beschreibung
Select$	*STRING*	*Filterstring für die zu berücksichtigenden Datensätze. Die Bedingungen können durch AND oder OR verknüpft werden.*

Beispiel: "Name='Mei*' AND Vorname='AN*'"
Wird kein String angegeben, werden alle Sätze der Datenbank verwendet

MergeFileDlg()

Ruft den Serienbriefdialog auf

NewFile(OPTIONAL TemplGrp$, TemplName$)

Neue Datei anlegen, optional kann eine Vorlage für die neue Datei angegeben werden.

Bezeichner	Typ	Beschreibung

TemplGrp$	*STRING*	*Name der Dokumentvorlagen-Gruppe, z.B. BEISPIEL*
TemplName$	*STRING*	*Name der Dokumentvorlage, z.B. BRIEF1*

NewFileDefault()

Neue Datei mit Standardeinstellungen wird erzeugt.

OpenFile(OPTIONAL File$, Filter$) AS INTEGER

Eine Datei öffnen. Werden keine Parameter angegeben, wird der Dialog Datei öffnen gestartet.

Bezeichner	Typ	Beschreibung
File$	*STRING*	*Name der Datei*
Filter$	*STRING*	*Filter, mit der die Datei gelesen werden soll*
RETURN	*BOOL*	*TRUE, Datei konnte geöffnet werden* *FALSE, Datei konnte nicht geöffnet werden*

PrintFile(OPTIONAL Start%, End%, Copies%, Collate%, PaperBin%)

Datei drucken, Dialog bei fehlenden Parametern.

Bezeichner	Typ	Beschreibung
Start%	*SHORT*	*von Seite*
End%	*SHORT*	*bis Seite*
Copies%	*SHORT*	*Anzahl der Kopien*
Collate%	*BOOL*	*Sortierung*
PaperBin%	*SHORT*	*Einzugsschacht, Druckerabhängig*

PrintFileDefault()

Druck der aktuellen Datei mit Standardeinstellungen.

PrintFileOptions(PrintGraphic%, OPTIONAL PrintTable%, Pages%,Reverse%, PostIt%, PaperBin%)

Druckoptionen festlegen.

Bezeichner	Typ	Beschreibung
PrintGraphic%	*BOOL*	*Grafiken drucken (BOOL)*
PrintTable%	*BOOL*	*Tabellen drucken (BOOL)*
Pages%	*SHORT*	*Seiten :* *1 = links* *2 = rechts* *3 = alle*
Reverse%	*BOOL*	*umgekehrte Reihenfolge*
PostIt%	*SHORT*	*Notizen* *0 = keine* *1 = nur* *2 = am Dokumentende* *3 = Seitenende*
PaperBin%	*BOOL*	*Papierschacht aus Seitenvorlage*

SaveAll()

Alle geöffneten Dateien speichern.

SaveFile() AS INTEGER

Aktuelles Dokument speichern.

Bezeichner	Typ	Beschreibung
RETURN	*BOOL*	*TRUE, Datei konnte gespeichert werden* *FALSE, Fehler beim Speichern der Datei*

SaveFileAs(OPTIONAL File$, Filter$) AS INTEGER

Aktuelles Dokument unter einem anderen Namen speichern. Wird kein Parameter angegeben, so wird der Dialog Speichern unter hochgefahren.

Bezeichner	Typ	Beschreibung
File$	*STRING*	*Name der Datei*
Filter$	*STRING*	*Filter, mit der die Datei geschrieben werden soll*
RETURN	*BOOL*	*TRUE, das Dokument wurde gespeichert* *FALSE, Fehler beim Speichern des Dokumentes*

SendMail()

Eine elektronische Mail verschicken.

SetDocInfo(OPTIONAL Type%, Value$)

Bei Abwesenheit von Parametern wird der Dialog Dokumentinfo gestartet, ansonsten wird das entsprechende Dokumentinfofeld auf den Wert Value$ gesetzt.

Bezeichner	Typ	Beschreibung
Type%	*SHORT*	*Titel = 0* *Thema = 1* *Schlüsselworte = 2* *Beschreibung = 3* *Info 0 - 3 = 4-7*
Value$	*STRING*	*Die zu setzende Zeichenkette*

Gruppe Bearbeiten

Backspace()

Zeichen links vor dem Cursor löschen.

BufferUpdate (StartStop%)

Ausgabe puffern

Bezeichner	Typ	Beschreibung
StartStop%	*BOOL*	*TRUE = Start der Pufferung* *FALSE = Ende der Pufferung*

Copy()

Den selektierten Bereich in die Zwischenablage kopieren.

Cut()

Den selektierten Bereich ausschneiden und in die Zwischenablage stellen.

Delete()

Zeichen rechts hinter dem Cursor löschen.

DeleteBookmark(Name$)

Textmarke Name$ löschen.

Bezeichner	Typ	Beschreibung
Name$	*STRING*	*Name der zu löschenden Textmarke.*

DelLine()

Eine Zeile löschen.

DelToEndOfLine()

Von der Cursorposition bis zum Ende der Zeile löschen.

DelToEndOfParagr()

Von der Cursorposition bis zum Ende des Absatzes löschen.

DelToEndOfSentence()

Von der Cursorposition bis zum Ende des Satzes löschen.

DelToEndOfWord()

Von der Cursorposition bis zum Wortende löschen.

DelToStartOfLine()

Von der Cursorposition bis zum Zeilenanfang löschen.

DelToStartOfParagr()

Von der Cursorposition bis zum Absatzanfang löschen.

DelToStartOfSentence()

Von der Cursorposition bis zum Anfang des Satzes löschen

DelToStartOfWord()

Von der Cursorposition bis zum Wortanfang löschen

DocStatDlg()

Öffnet den Dialog Dokumentstatistik.

EditIndexEntry()

Indexeintrag bearbeiten

Escape()

Liefert den Tastencode der Taste Escape.

ExecuteMacroField()

Feldbefehl Makro ausführen anstoßen.

FrameContentToBody()

Aus dem Rahmen zum Anker des Rahmens springen.

GetDataBaseFieldValue$(Name$)

Liefert den Wert des Datenbankfeldes Name$ als String. Es wird ein Leerstring zurückgegeben, wenn das Feld nicht vorhanden ist.

Bezeichner	Typ	Beschreibung
Name$	*STRING*	*Name des Datenbankfeldes*

SetupPrinterDlg()

Dialog für die Druckereinrichtung wird aufrufen.

GetSelectedText$()

Liefert den selektierten Text zurück. Ist kein Text selektiert, wird ein Leerstring zurückgegeben.

GetUserFieldValue$(Name$)

Liefert den Wert der Benutzerfeldes Name$ als String.

Bezeichner	Typ	Beschreibung
Name$	*STRING*	*Name des Benutzerfeldes*

GoToPos(OPTIONAL Label$, Select%)

Gehe zu Label$. Label$ ist ein optionales Argument; fehlt es, wird der Dialog Gehe zu gestartet. Eine Zahl als String meint eine Seitenzahl, ein anderer String bezeichnet eine Textmarkenposition.

Bezeichner	Typ	Beschreibung
Label$	*STRING*	*Textmarkenname oder Seitenzahl als String*
Select %	*BOOL*	*TRUE = Inhalt selektieren* *FALSE = nicht selektieren (Standard)*

NewUserField(Name$)

Benutzerfeld anlegen.

Bezeichner	Typ	Beschreibung
Name$	*STRING*	*Name des neuen Benutzerfeldes*

Paste()

Den Inhalt aus der Zwischenablage an aktueller Cursorposition einfügen

PasteSpecialDlg()

Ruft den Dialog Inhalte einfügen... auf.

Redo()

Letzten Befehl wiederholen.

RefreshView()

Bildschirminhalt neu aufbauen.

Repeat()

letzte Aktion wiederholen.

RepeatSearch()

Letzte Suche wiederholen.

ResetAttributs()

Harte Formatierung zurücksetzen.

Replace(Search$, Replace$, OPTIONAL WordOnly%, CaseSensitive%, RegExp%, Backward%, SelectAll%, Region%) AS INTEGER

Ersetzen von Textpassagen. Der Aufruf dieser Funktion ohne Parameter startet den Suchen & Ersetzen Dialog.

Bezeichner	Typ	Beschreibung
Search$	*STRING*	*der Suchbegriff*
Replace$	*STRING*	*der Ersetzungsbegriff*
WordOnly%	*BOOL*	*TRUE = nur ganzes Wort* *FALSE = auch Wortteile (Standard)*
CaseSensitive%	*BOOL*	*TRUE = Groß- / Kleinschreibung beachten* *FALSE = nicht beachten (Standard)*
RegExp%	*BOOL*	*TRUE = Regulärer Ausdruck* *FALSE = kein Regulärer Ausdruck (Standard)*
Backward%	*BOOL*	*TRUE = Suchrichtung rückwärts* *FALSE = Suchrichtung vorwärts (Standard)*
SelectAll%	*BOOL*	*TRUE = alle gefundenen Textstellen selektieren* *FALSE = nicht selektieren (Standard)*
Region%	*BOOL*	*TRUE = nur in Selektionen* *FALSE = Selektionen nicht beachten (Standard)*
RETURN	*INTEGER*	*Anzahl der ersetzen Begriffe*

Search(OPTIONAL Search$, WordOnly%, CaseSensitive%, RegExp%, Backward%, SearchAll%) AS INTEGER

Textsuche. Der Aufruf dieser Funktion ohne Parameter startet den Suchen & Ersetzen Dialog.

Bezeichner	Typ	Beschreibung
Search$	*STRING*	*der Suchbegriff*
WordOnly%	*BOOL*	*TRUE = nur Wort* *FALSE = auch Teilwörter (Standard)*
CaseSensitive%	*BOOL*	*TRUE = Groß-/Kleinschreibung beachten* *FALSE = nicht beachten (Standard)*
RegExp%	*BOOL*	*TRUE = Regulärer Ausdruck* *FALSE = kein regulärer Ausdruck (Standard)*
Backward%	*BOOL*	*TRUE = Suchrichtung rückwärts* *FALSE = Suchrichtung vorwärts (Standard)*
SearchAll%	*BOOL*	*TRUE = alle gefundenen Textstellen selektieren* *FALSE = nicht Selektieren (Standard)*
RETURN	*INTEGER*	*Anzahl der gefundenen Begriffe*

SetDocInfo(OPTIONAL Type%, Value$)

Setzt die Dokument-Info. Bei Abwesenheit von Parametern wird der Dokumentinfodialog gestartet.

Bezeichner	Typ	Beschreibung
Type%	*SHORT*	*Titel = 0* *Thema = 1* *Schlüsselworte = 2* *Beschreibung = 3* *Info 0 - 3 = 4-7*
Value$	*STRING*	*Die zu setzende Zeichenkette*

SetInsMode(OPTIONAL State%)

Einfüge-/Überschreibmodus ein oder aus schalten

Bezeichner	Typ	Beschreibung
State%:	*SHORT*	*SET_OFF (0) = aus*

SET_ON (1) = an
TOGGLE (2) = umschalten

SetUserFieldValue(Name$, Value$)

Setzt das Benutzerfeld Name$ auf den Wert Value$

Bezeichner	Typ	Beschreibung
Name$	*STRING*	*Name des Benutzerfeldes*
Value$	*STRING*	*Der zu setzende Feldinhalt*

Undo()

Letzte Aktion rückgängig machen.

UpdateDocStat()

Dokumentstatistik aktualisieren.

UpdateFields()

Feldinhalte aktualisieren.

Gruppe Cursorbewegung & Selektion

GetCursorXPos() AS LONG

Liefert die X-Koordinate der Cursorposition in 1/10 mm.

Bezeichner	Typ	Beschreibung
	LONG	*X-Koordinate der aktuellen Cursorposition*

GoDown(OPTIONAL Count%, Select%)

Cursor nach unten bewegen.

Bezeichner	Typ	Beschreibung
Count%	*SHORT*	*Anzahl Zeilen nach unten*
Select %	*BOOL*	*TRUE = Inhalt selektieren* *FALSE = nicht selektieren (Standard)*

GoFromIndexMarkToIndex()

Springt von einer Verzeichnismarkierung zum Verzeichnis.

GoLeft(OPTIONAL Count%, Select%)

Nach links springen.

Bezeichner	Typ	Beschreibung
Count%	*SHORT*	*Anzahl der Zeichen, um die nach rechts gewprungen werden soll.*
Select%	*BOOL*	*TRUE = Inhalt selektieren* *FALSE = nicht selektieren (Standard)*

GoPageDown()

Eine Seite nach unten springen.

GoPageUp()

Eine Seite nach oben springen.

GoRight(OPTIONAL Count%, Select%)

Nach rechts springen.

Bezeichner	Typ	Beschreibung
Count%	*SHORT*	*Anzahl der Zeichen, um die nach rechts gesprungen werden soll.*
Select%	*BOOL*	*TRUE = Inhalt selektieren* *FALSE = nicht selektieren (Standard)*

GoToEndOfColumn()

Mit dem Cursor an das Ende der Spalte springen.

GoToEndOfDoc(OPTIONAL Select%)

Mit dem Cursor ans Ende des Dokumentes springen.

Bezeichner	Typ	Beschreibung
Select%	*BOOL*	*TRUE = Inhalt selektieren* *FALSE = (Standard)*

GoToEndOfLine(OPTIONAL Select%)

Mit dem Cursor ans Ende der Zeile springen.

Bezeichner	Typ	Beschreibung
Select%	*BOOL*	*TRUE = Inhalt selektieren* *FALSE = nicht selektieren (Standard)*

GoToEndOfNextColumn()

Mit dem Cursor an das Ende der nächsten Spalte springen.

GoToEndOfNextPage(OPTIONAL Select%)

Mit dem Cursor ans Ende der nächsten Seite springen.

Bezeichner	Typ	Beschreibung
Select%	*BOOL*	*TRUE = Inhalt selektieren* *FALSE = nicht selektieren (Standard)*

GoToEndOfPage(OPTIONAL Select%)

Mit dem Cursor ans Ende der Seite springen.

Bezeichner	Typ	Beschreibung
Select%	*BOOL*	*TRUE = Inhalt selektieren* *FALSE = nicht selektieren (Standard)*

GoToEndOfParagr(OPTIONAL Select%)

Mit dem Cursor zum Ende des Absatzes springen.

Bezeichner	Typ	Beschreibung
Select%	*BOOL*	*TRUE = Inhalt selektieren* *FALSE = nicht selektieren (Standard)*

GoToEndOfPrevColumn()

Mit dem Cursor an das Ende der vorhergehenden Spalte springen.

GoToEndOfPrevPage(OPTIONAL Select%)

Mit dem Cursor ans Ende der vorigen Seite springen.

Bezeichner	Typ	Beschreibung
Select%	*BOOL*	*TRUE = Inhalt selektieren* *FALSE = nicht selektieren (Standard)*

GoToEndOfTable()

Mit dem Cursor an das Ende der Tabelle springen.

GoToFooter()

Mit dem Cursor in die Fußzeile springen.

GoToFootnoteOrAnchor()

Von der Fußnote zu ihrem Anker springen.

GoToFrameOrAnchor()

Mit dem Cursor vom Rahmen in den umgebenden Inhalt springen

GoToHeader()

Mit dem Cursor in die Kopfzeile springen.

GoToNextFootnote()

Mit dem Cursor zur nächsten Fußnote springen.

GoToNextFrame()

Mit dem Cursor in den Inhalt des nächsten Rahmens springen.

GoToNextSentence(OPTIONAL Select%)

Zum nächsten Satz springen.

Bezeichner	Typ	Beschreibung
Select%	*BOOL*	*TRUE = Inhalt selektieren* *FALSE = nicht selektieren (Standard)*

GoToNextTable()

Mit dem Cursor an den Anfang der folgenden Tabelle springen.

GoToNextWord(OPTIONAL Select%)

Mit dem Cursor zum nächsten Wort springen

Bezeichner	Typ	Beschreibung
Select%	*BOOL*	*TRUE = Inhalt selektieren* *FALSE = nicht selektieren (Standard)*

GoToPrevFootnote()

Mit dem Cursor zur vorhergehenden Fußnote springen.

GoToPrevSentence(OPTIONAL Select%)

Zum vorigen Satz springen.

Bezeichner	Typ	Beschreibung
Select%	*BOOL*	*TRUE = Inhalt selektieren* *FALSE = nicht selektieren (Standard)*

GoToPrevTable()

Mit dem Cursor an den Anfang der vorhergehenden Tabelle springen.

GoToPrevWord(OPTIONAL Select%)

Mit dem Cursor zum vorhergehenden Wort springen.

Bezeichner	Typ	Beschreibung
Select%	*BOOL*	*TRUE = Inhalt selektieren* *FALSE = nicht selektieren (Standard)*

GoToStartOfColumn()

Mit dem Cursor an den Anfang der Spalte springen.

GoToStartOfDoc(OPTIONAL Select%)

Mit dem Cursor zum Anfang des Dokumentes springen.

Bezeichner	Typ	Beschreibung
Select%	*BOOL*	*TRUE = Inhalt selektieren* *FALSE = nicht selektieren (Standard)*

GoToStartOfLine(OPTIONAL Select%)

Mit dem Cursor zum Zeilenanfang springen.

Bezeichner Typ	Beschreibung	
Select%	*BOOL*	*TRUE = Inhalt selektieren* *FALSE = nicht selektieren (Standard)*

GoToStartOfNextColumn()

Mit dem Cursor an den Anfang der nächsten Spalte springen.

GoToStartOfNextPage(OPTIONAL Select%)

Zum Anfang der nächsten Seite springen.

Bezeichner	Typ	Beschreibung
Select%	*BOOL*	*TRUE = Inhalt selektieren* *FALSE = nicht selektieren (Standard)*

GoToStartOfPage(OPTIONAL Select%)

Mit dem Cursor zum Anfang der Seite springen.

Bezeichner	Typ	Beschreibung
Select%	*BOOL*	*TRUE = Inhalt selektieren* *FALSE = nicht selektieren (Standard)*

GoToStartOfParagr(OPTIONAL Select%)

Mit dem Cursor zum Anfang des Absatzes springen.

Bezeichner	Typ	Beschreibung
Select%	*BOOL*	*TRUE = Inhalt selektieren* *FALSE = nicht selektieren (Standard)*

GoToStartOfPrevColumn()

Mit dem Cursor an den Anfang der vorhergehenden Spalte springen.

GoToStartOfPrevPage(OPTIONAL Select%)

Mit dem Cursor zum Anfang der vorhergehenden Seite springen

Bezeichner	Typ	Beschreibung
Select%	*BOOL*	*TRUE = Inhalt selektieren* *FALSE = nicht selektieren (Standard)*

GoToStartOfTable()

Mit dem Cursor an den Anfang der Tabelle

GoToTableCell(Cell$, OPTIONAL Select%)

Wenn der Cursor in einer Tabelle steht, gehe zu der Zelle Cell$ (B2), optional mit Selektion.

Bezeichner	Typ	Beschreibung
Cell$	*STRING*	*Zellenadresse als String*
Select%	*BOOL*	*TRUE = Inhalt selektieren* *FALSE = nicht selektieren*

GoUp(OPTIONAL Count%, Select%)

Bewegt den Cursor nach oben.

Bezeichner	Typ	Beschreibung
Count%	*SHORT*	*Anzahl der Zeilen, um die nach oben gesprungen werden soll*
Select%	*BOOL*	*TRUE = Inhalt selektieren* *FALSE = nicht selektieren (Standard)*

IsEndOfDoc() AS INTEGER

Abfrage, ob sich der Cursor am Dokumentende befindet.

Bezeichner	Typ	Beschreibung
RETURN	*BOOL*	*TRUE, der Cursor steht am Ende des Dokumentes und kann nicht weiter nach rechts gesetzt werden* *FALSE, der Cursor steht nicht am Ende des Dokumentes*

Beispiel:

```
function ZeilenZaehlen()
            Zeilen% = 1
            GoToStartOfDoc
            While(IsEndOfDoc=FALSE)
                GoDown
                GoToEndOfLine
                Zeilen% = Zeilen% +1
            Wend
            Print Zeilen%
end function
```

IsEndOfParagr() AS INTEGER

Abfrage, ob sich der Cursor am Absatzende befindet

Bezeichner	Typ	Beschreibung
RETURN	*BOOL*	*TRUE, der Cursor steht am Ende eines Absatzes* *FALSE, der Cursor steht nicht am Ende eines Absatzes*

IsEndOfWord() AS INTEGER

Abfrage, ob sich der Cursor am Wortende befindet

Bezeichner	Typ	Beschreibung
RETURN	*BOOL*	*TRUE, der Cursor steht am Ende eines Wortes* *FALSE, der Cursor steht nicht am Ende eines Wortes*

IsStartOfDoc() AS INTEGER

Abfrage, ob sich der Cursor am Dokumentanfang befindet

Bezeichner	Typ	Beschreibung
RETURN	*BOOL*	*TRUE, der Cursor steht am Anfang des Dokumentes und kann nicht weiter nach links gesetzt werden* *FALSE, der Cursor steht nicht am Anfang des Dokumentes*

Beispiel:

```
function PosTest()
            if IsStartOfDoc then
```

```
            Print("Sie befinden sich am Dokumentanfang")
        else
            if IsEndOfDoc then
              Print("Sie befinden sich am Dokumentende")
            else
              Print("Sie irgendwo im Text")
            end if
        end if
end function
```

IsStartOfParagr() AS INTEGER

Abfrage, ob sich der Cursor am Absatzanfang befindet

Bezeichner	Typ	Beschreibung
RETURN	*BOOL*	*TRUE, der Cursor steht am Anfang eines Absatzes* *FALSE, der Cursor steht nicht am Anfang eines Absatzes*

IsStartOfWord() AS INTEGER

Abfrage, ob sich der Cursor am Wortanfang befindet

Bezeichner	Typ	Beschreibung
RETURN	*BOOL*	*TRUE, der Cursor steht am Anfang eines Wortes* *FALSE, der Cursor steht nicht am Anfang eines Wortes*

SelectAll()

Selektiert den gesamten Fließtext.

SelectTableColumn()

Eine Tabellenspalte selektieren.

SelectTableRow()

Eine Tabellenzeile selektieren.

SelectWord()

Ein einzelnes Wort selektieren.

SetBulletOn()

Aufzählungsliste einschalten.

SetAdditionalMode(OPTIONAL State%)

Umschalten des Mehrfachselektionsmodus.

Bezeichner	Typ	Beschreibung
State%	*SHORT*	*SET_OFF (0) = aus* *SET_ON (1) = an* *TOGGLE (2) = Umschalten (Standard)*

SetExtendedMode(OPTIONAL State%)

Umschalten des Erweiterungsmodus

Bezeichner	Typ	Beschreibung
State%	*SHORT*	*SET_OFF (0) aus* *SET_ON (1) an* *TOGGLE (2) umschalten*

Gruppe Ansicht

ShowBounds()

Textbegrenzung ein- bzw.ausschalten.

ShowFieldName()

Feldbefehlnamen ein- bzw.ausschalten

ShowFields()

Feldbefehle ein- bzw.ausschalten

ShowHorzScrollbar()

Horizontalen Rollbalken ein- bzw.ausschalten

ShowMarks()

Markierungen ein- bzw.ausschalten

ShowMetaChars()

Sonderzeichen ein- bzw.ausschalten

ShowRibbon()

Objektleiste ein- bzw.ausschalten

ShowRuler()

Dokumentleiste ein- bzw.ausschalten

ShowStatusbar()

Statusleiste ein- bzw.ausschalten

ShowTableGrid()

Tabellenumrandung ein- bzw.ausschalten

ShowToolbox()

Funktionsleiste ein- bzw.ausschalten

ShowVertRuler()

Vertikales Lineal ein- bzw.ausschalten

ShowVertScrollbar()

Vertikalen Rollbalken ein- bzw.ausschalten

Zoom(OPTIONAL Type%, Factor%)

Darstellungsmaßstab schalten. Werden keine Parameter angegeben, wird der Maßstabsdialog gestartet.

Bezeichner	Typ	Beschreibung
Type%	*SHORT*	*Art des Maßstabs*
	:	*ein Faktor (0)*
		Seitenbreite (1)
		Seitengröße (2)
		optimale Breite (3)
Factor%	*SHORT*	*der Faktor für die Veränderung des Maßstabs; wird nur berücksichtigt, wenn Factor als Type% angegeben ist; Standard ist 100%*

Gruppe Einfügen

InsertBookmark(OPTIONAL Name$)

Bei Abwesenheit von Parametern wird der Dialog Einfügen Textmarke aufgerufen, ansonsten wird eine Textmarke mit dem Namen Name$ angelegt.

Bezeichner	Typ	Beschreibung
Name$	STRING	Name der Textmarke

InsertBreakDlg()

Der Dialog Einfügen Umbruch wird aufgerufen.

InsertColumnBreak

Spaltenumbruch einfügen.

InsertColumns(OPTIONAL Count%)

Rahmen mit Spalten einfügen. Wird kein Parameter angegeben, wird eine Spalte eingefügt.

Bezeichner	Typ	Beschreibung
Count%	*SHORT*	*Anzahl der Spalten*

InsertField(OPTIONAL Type%, SubType%, Para1$, Para2$, Format%)

Bei Abwesenheit von Parametern wird der Dialog Feld Einfügen aufgerufen, ansonsten wird das angegebene Feld eingefügt; Die Parameter werden in Abhängigkeit vom Feldtyp ausgewertet:

Notiz: Eine Notiz in das Dokument einfügen

Bezeichner	Typ	Beschreibung
Type%	*SHORT*	*TYP_POSTITFLD*
SubType%	*SHORT*	*wird nicht ausgewertet*
Para1$	*STRING*	*Autor*
Para2$	*STRING*	*Wert*
Format%	*SHORT*	*wird nicht ausgewertet*

Datum: Ein Datumsfeld in das Dokument einfügen

Bezeichner	Typ	Beschreibung
Type%	*SHORT*	*TYP_DATEFLD*
SubType%	*SHORT*	*DATE_FIX = 0* *DATE_VAR = 1*
Para1$	*STRING*	*wird nicht ausgewertet*
Para2$	*STRING*	*wird nicht ausgewertet*
Format%	*SHORT*	*DF_SSYS = 0 System kurz* *DF_LSYS =1 System lang* *DF_SHORT =2 DD.MM.JJ* *DF_SCENT = 3 DD.MM.JJJJ* *DF_LMON = 4 DD.MMM JJJJ* *DF_LMONTH = 5 DD.MMMM JJJJ* *DF_LDAYMON = 6 DDD, MMMM JJJJ* *DF_LDAYMONTH = 7 DDDD, MMMM JJJJ*

Uhrzeit: Ein Uhrzeitfeld in das Dokument einfügen

Bezeichner	Typ	Beschreibung
Type%	*SHORT*	*TYP_TIME*
SubType %	*SHORT*	*TIME_FIX = 0* *TIME_VAR = 1*
Para1$	*STRING*	*wird nicht ausgewertet*
Para2$	*STRING*	*wird nicht ausgewertet*
Format%	*SHORT*	*TF_SYSTEM = 0* *TF_SSMM_24 =1 0-24h* *TF_SSMM_12 = 2 0-12h*

Dateiname: Den Dateinamen in das Dokument einfügen

Bezeichner	Typ	Beschreibung
Type%	*SHORT*	*TYP_FILENAME_FLD*
SubType %	*SHORT*	*wird nicht ausgewertet*
Para1$	*STRING*	*wird nicht ausgewertet*
Para2$	*STRING*	*wird nicht ausgewertet*
Format%	*SHORT*	*FF_NAME = 0 Name* *FF_PATHNAME = 1 Pfad/Name* *FF_PATH = 2 Pfad*

Datenbankname: Den logischen Namen der aktiven Datenbank in das Dokument einfügen

Bezeichner	Typ	Beschreibung
Type%	*SHORT*	*TYP_DBNAME_FLD*
SubType%	*SHORT*	*wird nicht ausgewertet*
Para1$	*STRING*	*wird nicht ausgewertet*
Para2$	*STRING*	*wird nicht ausgewertet*
Format%	*SHORT*	*wird nicht ausgewertet*

Kapitel: Die Kapitelüberschrift bzw. -nummer als Referenz einfügen

Bezeichner	Typ	Beschreibung
Type%	*SHORT*	*TYP_CHAPTER_FLD*
SubType%	*SHORT*	*wird nicht ausgewertet*
Para1$	*STRING*	*wird nicht ausgewertet*
Para2$	*STRING*	*wird nicht ausgewertet*
Format%	*SHORT*	*CF_NUMBER = 0 Nummer* *CF_TITLE = 1 Titel* *CF_NUM_TITLE = 2 Nummer + Titel*

Seitennummer: Die Seitennummer in das Dokument einfügen

Bezeichner	Typ	Beschreibung
Type%	*SHORT*	*TYP_PAGENUMBERFLD*
SubType%	*SHORT*	*wird nicht ausgewertet*
Para1$	*STRING*	*wird nicht ausgewertet*
Para2$	*STRING*	*wird nicht ausgewertet*
Format%	*SHORT*	*CHARS_UPPER_LETTER = 0 A B C* *CHARS_LOWER_LETTER = 1 a b c* *ROMAN_UPPER = 2 I II III* *ROMAN_LOWER = 3 i ii iii* *ARABIC = 4 1 2 3* *PAGEDESC = 5 Wie Seitenvorl.*

Dokumentstatistik: Fügt die einzelnen Werte der Dokumentstatistik ein.

Bezeichner	Typ	Beschreibung
Type%	*SHORT*	*TYP_DOCSTATFLD*
SubType%	*SHORT*	*DS_PAGE = 0 Seiten* *DS_PARA = 1 Absätze* *DS_WORD = 2 Wörter* *DS_CHAR = 3 Zeichen* *DS_TBL = 4 Tabellen* *DS_GRF = 5 Grafiken* *DS_OLE = 6 Objekte*
Para1$	*STRING*	*wird nicht ausgewertet*
Para2$	*STRING*	*wird nicht ausgewertet*
Format%	*STRING*	*wird nicht ausgewertet*

Autor: Den Namen bzw. Initialien des Autors in das Dokument einfügen

Bezeichner	Typ	Beschreibung
Type%	*SHORT*	*TYP_AUTHORFLD*
SubType%	*SHORT*	*wird nicht ausgewertet*
Para1$	*STRING*	*wird nicht ausgewertet*
Para2$	*STRING*	*wird nicht ausgewertet*
Format%	*SHORT*	*AF_NAME = 0 Name* *AF_SHORCUT = 1 Initialien*

Versteckter Text: Fügt Versteckten Text in das Dokument ein.

Bezeichner	Typ	Beschreibung
Type	*SHORT*	*TYP_HIDDENTXTFLD*
SubType%	*SHORT*	*wird nicht ausgewertet*
Para1$	*STRING*	*Die Bedingung, ob und wann das Feld Angezeigt werden soll*
Para2$	*STRING*	*Der Inhalt des Versteckten Textes*
Format%	*SHORT*	*wird nicht ausgewertet*

Versteckter Absatz: Einen Absatz nicht mit ausdrucken, wenn die Bedingung TRUE ergibt

Bezeichner	Typ	Beschreibung
Type%	*SHORT*	*TYP_HIDDENTXTFLD*
SubType%	*SHORT*	*wird nicht ausgewertet*
Para1$	*STRING*	*Die Bedingung, wann der Absatz versteckt werden soll.*
Para2$	*STRING*	*wird nicht ausgewertet*
Format%	*STRING*	*wird nicht ausgewertet*

Referenz setzen: Eine Referenz in das Dokument einfügen

Bezeichner	Typ	Beschreibung
Type%	*SHORT*	*TYP_SETREFFLD*
SubType%	*SHORT*	*wird nicht ausgewertet*
Para1$	*STRING*	*Der Name der Referenz*
Para2$	*STRING*	*wird nicht ausgewertet*
Format%	*SHORT*	*wird nicht ausgewertet*

Referenz einfügen: Fügt eine Referenz in das Dokument ein

Bezeichner	Typ	Beschreibung
Type%	*SHORT*	*TYP_GETREFFLD*
SubType%	*SHORT*	*wird nicht ausgewertet*
Para1$	*STRING*	*Der Name der einzufügenden Referenz*

Para2$	*STRING*	*wird nicht ausgewertet*
Format%	*SHORT*	*REF_PAGE = 0 Seite* *REF_CHAPTER = 1 Kapitel* *REF_CONTENT = 2 Text*

DDE-Feld einfügen: Fügt ein Feld mit einer DDE-Verknüpfung ein

Bezeichner	**Typ**	**Beschreibung**
Type%	*SHORT*	*TYP_DDEFLD*
SubType%	*SHORT*	*wird nicht ausgewertet*
Para1$	*STRING*	*Der Name des DDE-Feldes*
Para2$	*STRING*	*Das DDE-Kommando, welches ausgeführt werden soll*
Format%	*SHORT*	*DDEFIELD_COLD = 0 Manuell* *DDEFIELD_HOT = 1 Automatisch*

Macrofeld: Ein Makro in das Dokument einfügen

Bezeichner	**Typ**	**Beschreibung**
Type%	*SHORT*	*TYP_MACROFLD*
SubType%	*SHORT*	*wird nicht ausgewertet*
Para1$	*STRING*	*Der Bereichsname, in dem das Macro gespeichert ist*
Para2$	*STRING*	*Der Name des Makros*
Format%	*SHORT*	*wird nicht ausgewertet*

Dokumentinfo: Fügt Inhalte der Dokumentinfo in das Dokument ein

Bezeichner	**Typ**	**Beschreibung**
Type%	*SHORT*	*TYP_DOCINFOFLD*
SubType%	*SHORT*	*DI_TITEL = 0 Titel* *DI_THEMA = 1 Thema* *DI_KEYS = 2 Schlüsselwörter* *DI_COMMENT = 3 Beschreibung* *DI_INFO1 = 4 Info 1* *DI_INFO2 = 5 Info 2* *DI_INFO3 = 6 Info 3* *DI_INFO4 = 7 Info 4* *DI_CREATE = 8 Erzeugung* *DI_CHANGE = 9 Änderung* *DI_PRINT = 10 Druck*
Para1$	*STRING*	*wird nicht ausgewertet*
Para2$	*STRING*	*wird nicht ausgewertet*
Format%	*STRING*	*RF_AUTHOR = 0 Autor* *RF_TIME = 1 Zeit* *RF_DATE = 2 Datum* *RF_ALL = 3 Alles*

Datenbankfeld: Ein Feld aus der aktiven Datenbank in das Dokument einfügen

Bezeichner	**Typ**	**Beschreibung**
Type%	*SHORT*	*TYP_DBFLD*
SubType%	*SHORT*	*wird nicht ausgewertet*
Para1$	*STRING*	*Der Name des Datenbankfeldes*
Para2$	*STRING*	*wird nicht ausgewertet*
Format%	*SHORT*	*wird nicht ausgewertet*

Benutzerfeld: Fügt ein Benutzerfeld in das Dokument ein.

Bezeichner	Typ	Beschreibung
Type%	*SHORT*	*TYP_USERFLD*
SubType%	*SHORT*	*wird nicht ausgewertet*
Para1$	*STRING*	*Name des Benutzerfeldes*
Para2$	*STRING*	*Wert des Benutzerfeldes*
Format%	*SHORT*	*UF_STANDARD = 0 Standard* *UF_TEXT = 1 Text* *UF_XX = 2 ####* *UF_XX_XX = 3 #.###,##* *UF_XX_XX_CUR = 4 #.###,## DM* *UF_XXP = 5 ####%* *UF_XX_XXP = 6 #.###,##%*

Eingabefeld: Ein Eingabefeld in das Dokument einfügen.

Bezeichner	Typ	Beschreibung
Type%	*SHORT*	*TYP_INPUTFLD*
SubType%	*SHORT*	*VIEW_TXT = 0 Text* *VIEW_USR = 1 Benutzerfeld* *VIEW_VAR = 2 Variable*
Para1$	*STRING*	*Der Inhalt des Eingabefeldes*
Para2$	*STRING*	*Hinweiß für den Eingabedialog setzen*
Format%	*STRING*	*wird nicht ausgewertet*

Variable setzen: Fügt eine Variable in das Dokument ein.

Bezeichner	Typ	Beschreibung
Type%	*SHORT*	*TYP_SETFLD*
SubType %	*SHORT*	*wird nicht ausgewertet*
Para1$	*STRING*	*Der Name der Variable*
Para2$	*STRING*	*Der Wert der Variable*
Format	*SHORT*	*UF_VISIBLE = 0 Sichtbar* *UF_INVISIBLE = 1 Unsichtbar* *UF_XX = 2 ####* *UF_XX_XX = 3 #.###,##* *UF_XX_XX_CUR = 4 #.###,## DM* *UF_XXP = 5 ####%* *UF_XX_XXP = 6 #.###,## %*

Variable anzeigen: Fügt ein Feld zur Anzeige von Variablen ein.

Bezeichner	Typ	Beschreibung
Type%	*SHORT*	*TYP_GETFLD*
SubType%	*SHORT*	*wird nicht ausgewertet*
Para1$	*STRING*	*Der Name der Variable*
Para2$	*STRING*	*wird nicht ausgewertet*
Format%	*SHORT*	*VF_STANDARD = 0 Standard* *VF_TEXT = 1 Text* *VF_XX = 2 ####* *VF_XX_XX = 3 #.###,##* *VF_XX_XX_CUR = 4 #.###,## DM* *VF_XXP = 5 ####%* *VF_XX_XXP = 6 #.###,##%*

Formel einfügen: Eine Formel (Berechnung) in das Dokument einfügen

Bezeichner	Typ	Beschreibung

Type%	*SHORT*	*TYP_FORMELFLD*
SubType%	*SHORT*	*wird nicht ausgewertet*
Para1$	*STRING*	*wird nicht ausgewertet*
Para2$	*STRING*	*Die einzufügende Formel*
Format%	*SHORT*	*VF_STANDARD = 0 Standard* *VF_TEXT = 1 Text* *VF_XX = 2 ####* *VF_XX_XX = 3 #.###,##* *VF_XX_XX_CUR = 4 #.###,## DM* *VF_XXP = 5 ####%* *VF_XX_XXP = 6 #.###,##%*

InsertFile(OPTIONAL File$, Filter$) AS INTEGER

wie OpenFile; die Datei wird im Unterschied zu OpenFile in das aktuelle Dokument eingefügt.

Bezeichner	Typ	Beschreibung
File$	*STRING*	*Name der Datei*
Filter$	*STRING*	*Filter, mit der die Datei gelesen werden soll*
RETURN	*BOOL*	*TRUE, Datei konnte eingefügt werden* *FALSE, Datei konnte nicht eingefügt werden*

InsertFootnote(OPTIONAL Text$)

Einfügen einer Fußnote mit dem Text Text$. Wird kein Text übergeben, wird automatisch der Dialog aufgerufen.

Bezeichner	Typ	Beschreibung
Text$	*STRING*	*Der Text, der in der Fußnote eingefügt werden soll*

InsertFrame(OPTIONAL AnchorType%, Width&, Height&, PosX&, PosY&)

bei Anwesenheit von Parametern wird der Dialog Rahmen aufgerufen, ansonsten der deinierte Rahmen eingefügt.

Bezeichner	Typ	Beschreibung
AnchorType%	*SHORT*	*FRAMEANCHORPAGE = Seitengebunden* *FRAMEANCHORPARA = Absatz* *FRAMEANCHORCHAR = Zeichen*
Width&	*LONG*	*Angabe der Breite in 1/10mm*
Height&	*LONG*	*Angabe der Höhe in 1/10mm*
PosX&	*LONG*	*Angabe der X-Position der linken oberen Ecke*
PosY&	*LONG*	*Angabe der Y-Position der linken oberen Ecke*

InsertFrameInteractive()

Modus für Rahmen mit der Maus aufziehen einschalten.

InsertGlossary(Group$, Name$)

Autotext mit dem Namen Name$ in der Gruppe Group$ einfügen.

Bezeichner	Typ	Beschreibung
Group$	*STRING*	*Name der Gruppe, in dem der Autotext eingefügt werden soll*

Name$	*STRING*	*Name des Autotextes*

InsertGraphic(OPTIONAL File$, Filter$)

Dialog Grafik einfügen, oder Einfügen der Grafik mit dem Namen File$ und dem Filter Filter$.

Bezeichner	Typ	TypBeschreibung
File$	*STRING*	*Name der Grafikdaten, ggf. mit Pfadangabe*
Filter$	*STRING*	*Name des Grafikfilters*

InsertHardSpace()

Ein hartes Leerzeichen einfügen.

InsertIndex(OPTIONAL CreateFrom%, Title$)

Einfügen eines Stichwortverzeichnisses. Die Form kann durch Addition der Werte kombiniert werden.

Bezeichner	Typ	Beschreibung
CreateForm	*SHORT*	*erzeugen aus:* *1 = Zusammenfassen gleicher Einträge* *2 = Zusammenfassen von Einträgen mit ff* *4 = Groß-/Kleinschreibung beachten* *8 = Schlüssel als eigener Eintrag* *16 = Alphabetisches Trennzeichen*
Title$	*STRING*	*Titel des Verzeichnisses ('Stichwortverzeichnis' ist Standard)*

InsertIndexEntry(OPTIONAL Key$, AltKey$, Entry$)

Einfügen einer Markierung für das Stichwortverzeichnis

Bezeichner	Typ	Beschreibung
Key$	*STRING*	*der Schlüssel des Stichwortes*
AltKey$	*STRING*	*der zweite Schlüssel des Stichwortes*
Entry$	*STRING*	*Text im Stichwortverzeichnis*

InsertLineBreak()

Ein Zeilenumbruch wird einfügt.

InsertObjectDlg()

Der Dialog Objekt einfügen (OLE) wird aufgerufen.

InsertPageBreak(OPTIONAL Style$)

Ein Seitenumbruch wird eingefügt.

Bezeichner	Typ	Beschreibung
Style$	*STRING*	*Die Seitenvorlage für die neue Seite. Standard ist die aktuelle Seitenvorlage.*

InsertParagrBreak()

Ein Absatzumbruch wird eingefügt.

InsertRecord(Select$)

Einfügen eines Datensatzes

Bezeichner	Typ	Beschreibung
Select$	*STRING*	*Filterstring für den einzufügenden Datensatz. Die Bedingungen können durch AND oder OR verknüpft werden. Treffen die Filterbedingungen auf mehrere Datensätze zu, wird der zuerst gefundene verwendet*

Beispiel "Name='Mei*' AND Vorname='AN*' "
Wird kein String angegeben, werden alle Sätze der Datenbank verwendet

InsertRecordDlg()

Der Dialog Datensatz einfügen wird aufgerufen.

InsertSoftHyphen()

Ein weicher Trenner wird eingefügt.

InsertString(A$)

Eine Zeichenkette wird eingefügt

Bezeichner	Typ	Beschreibung
A$	*STRING*	*Der einzufügende Text*

InsertSymbolDlg()

Der Dialog Einfügen Sonderzeichen wird aufgerufen.

InsertTable(OPTIONAL Row%, Col%)

Bei Abwesenheit von Parametern wird der Dialog Tabelle einfügen aufgerufen bzw. die Konvertierung des selektierten Inhalts in eine Tabelle ausgeführt. Andernfalls wird eine Tabelle mit entsprechenden Zeilen / Spalten eingefügt. Spaltenstandard ist 5.

Bezeichner	Typ	Beschreibung
Row%	*SHORT*	*Anzahl der Zeilen*
Col%	*SHORT*	*Anzahl der Spalten*

InsertTOC(OPTIONAL CreateFrom%, MaxLevel%, Titel$)

Einfügen eines Inhaltsverzeichnisses. Die Form kann durch Addition der Werte kombiniert werden.

Bezeichner	Typ	Beschreibung
CreateForm%	*SHORT*	*erzeugen aus:* *1 = Aus Markierungen* *2 = Aus Gliederung*
MaxLevel%	*SHORT*	*Einträge bis zu dieser Ebene werden berücksichtigt (1-5)*
Title$	*STRING*	*Titel des Verzeichnisses ('Inhaltsverzeichnis' ist Standard)*

InsertTOCEntry(Level%, OPTIONAL Entry$)

Einfügen einer Markierung für das Inhaltsverzeichnis.

Bezeichner	Typ	Beschreibung
Level%	*SHORT*	*Ebene der Eintrags (1-5)*
Entry$	*STRING*	*Text im Inhaltsverzeichnis*

InsertUserEntry(IndexName$, OPTIONAL Level%, Entry$)

Einfügen einer Markierung für das Benutzerverzeichnis.

Bezeichner	Typ	Beschreibung
IndexName$	*STRING*	*Name des Benutzerverzeichnisses. Falls das Verzeichnis nicht existiert, wird es angelegt.*
Level%	*SHORT*	*Ebene der Eintrags*
Entry$	*STRING*	*Text im Benutzerverzeichnis*

InsertUserIndex(OPTIONAL CreateFrom%, IndexName$, MaxLevel%, Title$, Template$)

Einfügen eines Benutzerverzeichnisses. Die Form der Erzeugung kann durch Addition der Werte Kombiniert werden

Bezeichner	Typ	Beschreibung
CreateForm%	*SHORT*	*erzeugen aus:* *1 = Markierungen* *4 = Absatzvorlagen* *8 = OLE* *16 = Tabellen* *32 = Grafiken*
IndexName$	*STRING*	*Name des Benutzerverzeichnisses*
MaxLevel%	*SHORT*	*Eintrage bis zu dieser Ebene werden berücksichtigt*
Title$	*STRING*	*Titel des Verzeichnisses ('Benutzerverzeichnis' ist Standard)*
Template$	*STRING*	*Name der Absatzvorlage, deren Inhalt in das Benutzerverzeichnis übernommen werden soll.*

MergeRecord(Select$)

Mischen der Serienbrieffelder.

Bezeichner	Typ	Beschreibung
Select$	*STRING*	*Filterstring für den einzufügenden Datensatz. Die Bedingungen können durch AND oder OR verknüpft werden. Treffen die Filterbedingungen auf mehrere Datensätze zu, wird der zuerst gefundene verwendet*

Beispiel "Name='Mei*' AND Vorname='AN*'"
Wird kein String angegeben, werden alle Sätze der Datenbank verwendet

PostIt()

Einfügen einer Notiz.

SelectDataBase(Name$)

Auswahl einer Datenbank.

Bezeichner	Typ	Beschreibung
Name$	*STRING*	*Der logische Name der Datenbank.*

Gruppe Format

BackgroundDlg()

Der Dialog Format Hintergrund wird aufgerufen.

BorderDlg()

Der Dialog Format Umrandung wird aufgerufen.

FormatTableDlg()

Der Dialog Format Tabelle wird aufgerufen (nur wenn der Cursor in einer Tabelle steht).

FrameDlg()

Startet den Dialog Format Rahmen.

GraphicDlg()

Der Dialog Format Grafik wird aufgerufen.

SetBackground(Style%, OPTIONAL ColorRed&, ColorGreen&, ColorBlue&)

Hintergrund einstellen.

Bezeichner	Typ	Beschreibung
Style%	*SHORT*	*Das Hintergrundmuster; der Zahlenwert entspricht der Position des Musters innerhalb der Muster-Listbox des entsprechenden Dialogs.*
ColorRed&	*LONG*	*Rotanteil der Hintergrundfarbe*
ColorGreen&	*LONG*	*Grünanteil der Hintergrundfarbe*
ColorBlue&	*LONG*	*Blauanteil der Hintergrundfarbe*

SetBorder(LineLocation%, LineType%, OPTIONAL ColorRed&, ColorGreen&, ColorBlue&, Space&)

Attribut Umrandung setzen.

Bezeichner	Typ	Beschreibung
LineLocation%	*SHORT*	*Linienposition:* *1 = Oben* *2 = Unten* *4 = Links* *8 = Rechts*
LineType%	*SHORT*	*Linienart, der Zahlenwert entspricht der Position des Linientypes innerhalb der Linientyp-Listbox des entsprechenden Dialogs.*
ColorRed&	*LONG*	*Rotanteil der Linenfarbe*
ColorGreen&	*LONG*	*Grünanteil der Linenfarbe*
ColorBlue&	*LONG*	*Blauanteil der Linenfarbe*
Space%	*SHORT*	*Abstand der Linie zum Inhalt*

SetShadowStyle(ShadowLocation%, OPTIONAL ColorRed&, ColorGreen&, ColorBlue&, Width%)

Attribut Schatten setzen.

Bezeichner	Typ	Beschreibung
ShadowLocation%	*SHORT*	*Position des Schattens, der Zahlenwert entspricht der Position der Angabe innerhalb der Schattenposition-Listbox des entsprechenden Dialogs.*
ColorRed&	*LONG*	*Rotanteil der Schattenfarbe*
ColorGreen&	*LONG*	*Grünanteil der Schattenfarbe*
ColorBlue&	*LONG*	*Blauanteil der Schattenfarbe*
Width%	*LONG*	*Breite des Schattens*

SetTab (How%, Position&, OPTIONAL Adjust%, DecimalChar$, FillChar$)

Tabulatoren setzen, löschen oder ändern.

Bezeichner	Typ	Beschreibung
How%	*SHORT*	*beschreibt die Aktion, die erfolgen soll* *SET_TAB = Tab setzen oder bei vorhandenem Tab Attribute modifizieren* *CLEAR_TAB = Löschen eines Tabs* *CLEAR_ALLTABS =Alle Tabs löschen*
Position&	*LONG*	*Position des TabStops in 1/10mm (dient bei einer Änderung als Identifizierung)*
Adjust%	*SHORT*	*Ausrichtung des TabStops:* *0 = links* *1 = rechts* *2 = zentriert* *3 = dezimal*
DecimalChar$	*STRING*	*Dezimalzeichen*
FillChar$	*STRING*	*Füllzeichen*

TabDlg()

Der Dialog Format Tabulator wird aufgerufen.

TableSelect()

Tabelle selektieren

ThesaurusDlg()

Der Dialog Thesaurus wird aufgerufen.

UndoToLastVersion()

Die letzte Version des Dokumentes wird wieder hergestellt.

Gruppe Vorlagen

FrameStyleDlg()

Der Dialog Rahmenvorlage wird aufgerufen.

GetCharStyleAt(Idx%) AS STRING

Zeichenvorlage an der Position Idx% erfragen. Die erste hat den Index 0.

Bezeichner	Typ	Beschreibung
Idx%	*SHORT*	*Index der Zeichenvorlage*
RETURN	*STRING*	*Name der Zeichenvorlage*

GetCharStyleCount() AS SHORT

Anzahl der Zeichenvorlagen erfragen

Bezeichner	Typ	Beschreibung
RETURN	*STRING*	*Anzahl der vorhandenen Zeichenvorlagen*

GetPageStyleAt(Idx%) AS STRING

Seitenvorlage an der Position Idx% erfragen. Die erste hat den Index 0.

Bezeichner	Typ	Beschreibung
Idx%	*SHORT*	*Index der Seitenvorlage*
RETURN	*STRING*	*Name der Seitenvorlage*

GetPageStyleCount() AS SHORT

Anzahl der Seitenvorlagen erfragen

Bezeichner	Typ	Beschreibung
RETURN	*STRING*	*Anzahl der vorhandenen Seitenvorlagen*

GetParagrStyleAt(Idx%) AS STRING

Absatzvorlage an der Position Idx% erfragen. Die erste hat den Index 0.

Bezeichner	Typ	Beschreibung
Idx%	*SHORT*	*Index der Absatzvorlage*
RETURN	*STRING*	*Name der Absatzvorlage*

GetParagrStyleCount() AS SHORT

Anzahl der Seitenvorlagen erfragen

GetDocInfo(Type%) AS STRING

Abfragen der aktuellen Werte in der Dokumentinformation.

Bezeichner	Typ	Beschreibung
Type%	*SHORT*	*Titel = 0*

		Thema = 1 *Schlüsselworte = 2* *Beschreibung = 3* *Info 0 - 3 = 4-7* *Datei = 8* *Erzeugungsdatum = 9* *Änderungsdatum = 10* *Druckdatum = 11*
RETURN	*STRING*	*der Inhalt des Feldes als String*

GetSelectedText() AS STRING

selektierten Text erfragen

Bezeichner	**Typ**	**Beschreibung**
RETURN	*STRING*	*der selektierte String*

NewPageStyle(Name$, OPTIONAL FollowStyle$)

Erzeugt eine neue Seitenvorlage mit dem Namen Name$, optional mit der Folgevorlage FollowStyle$.

Bezeichner	**Typ**	**Beschreibung**
Name$	*STRING*	*Name der neuen Seitenvorlage*
FollowStyle$	*STRING*	*Name der Folgevorlage. Wird dieser Wert nicht angegeben, wird der neuen Seitenvorlage keine Folgevorlage zugewiesen.*

NewStyleByExample()

Ruft den Dialog zur Erzeugung einer neue Absatzvorlage aus den Attributen des aktuellen Bereiches auf.

SetCharStyle(OPTIONAL Style$)

Zeichenvorlage anwenden. Wird kein Style$ angegeben, wird der Dialog Zeichenvorlagen aufgerufen

Bezeichner	**Typ**	**Beschreibung**
Style$	*STRING*	*Name der Zeichenvorlage*

SetPageStyle(OPTIONAL Style$)

Seitenvorlage anwenden. Wird kein Style$ angegeben, wird der Dialog Seitenvorlagen aufgerufen

Bezeichner	**Typ**	**Beschreibung**
Style$	*STRING*	*Name der Seitenvorlage*

SetPageStyleHeader(On%, Dist&, Fix&, ShareContent%. OPTIONAL Style$)

Kopfzeile für die Seitenvorlage setzen

Bezeichner	**Typ**	**Beschreibung**
On%	*SHORT*	*Kopfzeile an / aus:* *0 = aus* *1 = an*
Dist&	*LONG*	*Abstand zum Fließtext in 1/10mm*
Fix&	*LONG*	*Feste Kopfzeilenhöhe in 1/10mm; 0 bedeutet variable Höhe*

ShareContent%	*BOOL*	*TRUE = Inhalt rechts/ links gleich (Standard)* *FALSE = Inhalt ungleich*
Style$	*STRING*	*Name der Seitenvorlage; Wird dieser Parameter nicht angegeben, wird die aktuelle Seitenvorlage geändert.*

SetPageStyleFooter(On%, Dist&, Fix&, ShareContent%. OPTIONAL Style$)

Fußzeile für die Seitenvorlage setzen

Bezeichner	Typ	Beschreibung
On%	*BOOL*	*Fußzeile an / aus:* *0 = aus* *1 = an*
Dist&	*LONG*	*Abstand zum Fließtext in 1/10mm*
Fix&	*LONG*	*Feste Fußzeilenhöhe in 1/10mm; 0 bedeutet variable Höhe*
ShareContent%	*BOOL*	*TRUE = Inhalt rechts/ links gleich (Standard)* *FALSE = Inhalt ungleich*
Style$	*STRING*	*Name der Seitenvorlage; Wird dieser Parameter nicht angegeben, wird die aktuelle Seitenvorlage geändert.*

SetPageStyleLeftRightMargin(Left&, Right&, OPTIONAL Style$)

Setzt den linken oder rechten Rand. Optional kann die Angabe einer Seitenvorlage erfolgen, sonst wird aktuelle Seitenvorlage verwendet.

Bezeichner	Typ	Beschreibung
Left&	*LONG*	*Größe des linken Randes in 1/10mm*
Right&	*LONG*	*Größe des rechten Randes in 1/10mm*
Style$	*STRING*	*Name der Seitenvorlage, wird dieser Parameter nicht angegeben, wird die aktuelle Seitenvorlage geändert.*

SetPageStyleNumberFormat(Format%, OPTIONAL Style$)

Setzt das Seitennummernformat für die aktuelle Seite. Optional kann die Angabe einer Seitenvorlage erfolgen.

Bezeichner	Typ	Beschreibung	
Format%	*SHORT*	*CHARS_UPPER_LETTER = 0*	*A B C*
		CHARS_LOWER_LETTER = 1	*a b c*
		ROMAN_UPPER = 2	*I II III*
		ROMAN_LOWER = 3	*i ii iii*
		ARABIC = 4 *1 2 3*	
		PAGEDESC = 5 *Wie Seitenvorl.*	
Style$	*STRING*	*Name der Seitenvorlage, wird dieser Parameter nicht angegeben, wird die aktuelle Seitenvorlage geändert.*	

SetPageStylePage(Page%, OPTIONAL Style$)

Setzt linke, rechte oder alle Seiten in der optional angegebenen Seitenvorlage. Wird keine Seitenvorlage angegeben, wird die aktuelle Seitenvorlage geändert.

Bezeichner	Typ	Beschreibung
Page%	*SHORT*	*Seitenlayout für:* *1 = linke Seiten*

		2 = rechte Seiten *3 = Alle Seiten (Standard)* *7 = Gespiegelt*
Style$	*STRING*	*Name der Seitenvorlage, wird dieser Parameter nicht angegeben, wird die aktuelle Seitenvorlage geändert.*

SetPageStylePaperBin(PaperBin%, OPTIONAL Style$)

Setzt den Druckerschacht für die aktuelle Seite, wenn optional keine Angabe einer Seitenvorlage gesetzt wurde.

Bezeichner	Typ	Beschreibung
PaperBin%	*SHORT*	*Druckerschachtnummer, Druckertreiber abhängig. Der Erste Schacht des Druckertreibers hat die Nummer 0.*
Style$	*STRING*	*Name der Seitenvorlage, wird dieser Parameter nicht angegeben, wird die aktuelle Seitenvorlage geändert.*

SetPageStylePaperSize(Width&, Height&, OPTIONAL Style$)

Setzt die Papiergröße in der optional angegebenen Seitenvorlage. Wird keine Seitenvorlage angegeben, wird die aktuelle verändert

Bezeichner	Typ	Beschreibung
Width&	*LONG*	*Seitenbreite in 1/10mm*
Hight&	*LONG*	*Seitenhöhe in 1/10mm*
Style$	*STRING*	*Name der Seitenvorlage, wird dieser Parameter nicht angegeben, wird die aktuelle Seitenvorlage geändert.*

SetPageStyleTopBottomMargin(Top&, Bottom&, OPTIONAL Style)

Setzt die Ränder oben und unten in der aktuellen Seitenvorlage, wenn optional keine angegeben wurde.

Bezeichner	Typ	Beschreibung
Top&	*LONG*	*Oberer Rand in 1/10mm*
Buttom&	*LONG*	*Unterer Rand in 1/10mm*
Style$	*STRING*	*Name der Seitenvorlage, wird dieser Parameter nicht angegeben, wird die aktuelle Seitenvorlage geändert.*

SetParagrStyle(OPTIONAL Style$)

Absatzvorlage anwenden.

Bezeichner	Typ	Beschreibung
Style$	*STRING*	*Name der anzuwendenen Absatzvorlage;Wird dieser Parameter nicht mit angegeben, dann wird der Dialog Absatzvorlagen aufgerufen.*

UpdateStyleByExample()

Übernimmt die harte Attributierung des Absatzes in die aktuelle Absatzvorlage.

Gruppe Text

GrowFontSize()

vergrößert die Fontgröße um einen Schritt.

SetAbsLineSpace(Type%, Value&)

Zeilenabstand absolut einstellen (metrisch)

Bezeichner	Typ	Beschreibung
Type%	*SHORT*	*LINEMIN = minimaler Abstand* *LINEFIX = Durchschuß*
Value&	*LONG*	*Zeileabstand bzw. Duchschußwert in 1/10mm*

SetBold(OPTIONAL State%)

Schaltet die Fettschrift an- oder ausschalten

Bezeichner	Typ	Beschreibung
State%	*SHORT*	*SET_OFF = aus* *SET_ON = an* *TOGGLE = Umschalten (Standard)*

SetCaseMap(CaseMap%)

Textauszeichnungen wie Kapitälchen, Gemeine etc. setzen.

Bezeichner	Typ	Beschreibung
CaseMap%	*SHORT*	*Die Textauszeichnung; der Zahlenwert entspricht der Position der Auszeichnung innerhalb der Auszeichnungs-Listbox im entsprechenden Dialog.*

SetColor(ColorRed&, ColorGreen&, ColorBlue&)

Zeichenfarbe nach dem Farbmodell RGB einstellen.

Bezeichner	Typ	Beschreibung
ColorRed&	*LONG*	*Rotanteil der Zeichenfarbe*
ColorGreen&	*LONG*	*Grünanteil der Zeichenfarbe*
ColorBlue&	*LONG*	*Blauanteil der Zeichenfarbe*

SetFont(Name$, OPTIONAL Family%, Pitch%, CharSet%)

Neuen Font einschalten.

Bezeichner	Typ	Beschreibung
Name$	*STRING*	*Fontname, z.B. Times New Roman*
Family%	*SHORT*	*Fontfamilie:* *0 = Unbekannt* *1 = Dekorativ* *2 = Modern* *3 = Roman* *4 = Skript*

		5 = Swiss *6 = System*
Pitch%	*SHORT*	*Fontart:* *0 = Unbekannt* *1 = Fixfont* *2 = Proportionalfont*
CharSet%	*SHORT*	*Zeichensatz:* *0 = Unbekannt* *1 = Ansi* *2 = Mac* *3 = PC 437* *4 = PC 850* *5 = PC 860* *6 = PC 861* *7 = PC 863* *8 = PC 865* *9 = System* *10 = Symbol*

SetHyphenZone(On%, CharCount%)

automatische Silbentrennung umschalten

Bezeichner	Typ	Beschreibung
On%	*SHORT*	*aus = 0* *an = 1*
CharCount%	*SHORT*	*Nach anzahl Zeichen*

SetItalic(OPTIONAL State%)

Kursivschrift an- oder ausschalten.

Bezeichner	Typ	Beschreibung
State%	*SHORT*	*SET_OFF = aus* *SET_ON = an* *TOGGLE = Umschalten (Standard)*

SetKeepTogether(On%)

Absatz »nicht trennen« einschalten.

Bezeichner	Typ	Beschreibung
On%	*SHORT*	*aus = 0* *an = 1*

SetKerning(Kerning#)

Stellt das Kerning ein. Angabe in Punkt.

Bezeichner	Typ	Beschreibung
Kerning#	*INTEGER*	*Kerning in Punktgröße, negative Werte erlaubt.*

SetLanguage(Language%)

Sprachattribut setzen.

Bezeichner	Typ	Beschreibung
Language%	*SHORT*	*Sprache:* *0 = System* *1031 = Deutsch* *2055 = Deutsch (CH)* *1033 = Englisch (US)* *1036 = Französich* *1040 = Italienisch* *1034 = Spanisch* *2070 = Portugisisch* *1030 = Dänisch* *2067 = Flämisch* *1053 = Schwedisch* *1037 = Finnisch* *1044 = Norweg. Bokmal* *2068 = Norweg. Nynorsk*

SetLeftRightMargin(LeftMargin&,OPTIONAL RightMargin&, FirstLineIndent&)

linken und rechten Absatzrand in 1/10 mm setzen.

Bezeichner	Typ	Beschreibung
LeftMargin&	*LONG*	*Einzug linker Rand in 1/10mm*
RightMargin&	*LONG*	*Einzug rechter Rand in 1/10mm*
FirstLineIndent&	*LONG*	*Einzug der ersten Zeile links in 1/10mm*

SetOrphan(Rows%)

Schusterjungenregelung setzen.

Bezeichner	Typ	Beschreibung
Rows%	*SHORT*	*Anzahl Zeilen*

SetOutline(OPTIONAL State%)

Outlineschrift an- oder ausschalten.

Bezeichner	Typ	Beschreibung
State%	*SHORT*	*SET_OFF = aus* *SET_ON = an* *TOGGLE = Umschalten (Standard)*

SetPageBreak(Style%)

Seitenumbruch an den Absatz binden.

Bezeichner	Typ	Beschreibung
Style%	*SHORT*	*0 = kein Seitenumbruch* *1 = davor* *2 = dahinter* *3 = davor und dahinter*

SetParagrAdjust(Adjust%)

Absatzausrichtung einstellen.

Bezeichner	Typ	Beschreibung
Adjust%	*SHORT*	*0 = linksbündig* *1 = rechtsbündig* *2 = zentriert* *3 = Blocksatz*

SetPropLineSpace(Type%, OPTIONAL Value%)

Relativen Zeilenabstand einstellen.

Bezeichner	Typ	Beschreibung
Type%	*SHORT*	*LINE1 = einzeilig* *LINE15 = 1.5zeilig* *LINE2 = zweizeilig* *LINEPROP = proprotional*
Value%	*LONG*	*Proportionaler Zeilenabstand in Prozentangabe. Dieser Parameter wird nur bei LINEPROP berücksichtigt. Der Standard-Wert beträgt100%.*

SetShadow(OPTIONAL State%)

Schattenschrift an- oder ausschalten.

Bezeichner	Typ	Beschreibung
State%	*SHORT*	*SET_OFF = aus* *SET_ON = an* *TOGGLE = Umschalten (Standard)*

SetSize(Size%)

Fontgröße setzen.

Bezeichner	Typ	Beschreibung
Size%	*SHORT*	*Fontgröße in Punkt*

SetStrikeOut(OPTIONAL State%)

Durchstreichen an- oder ausschalten.

Bezeichner	Typ	Beschreibung
State	*SHORT*	*SET_OFF = aus* *SET_ON = an* *TOGGLE = Umschalten (Standard)*

SetSubScript (OPTIONAL State%, nEsc%, nSize%)

Tiefstellung an- oder ausschalten.

Bezeichner	Typ	Beschreibung
State%	*SHORT*	*SET_OFF = aus* *SET_ON = an* *TOGGLE = Umschalten (Standard)*

nEsc%	*SHORT*	*Tiefstellung in Prozent*
nSize%	*SHORT*	*Zeichengröße in Prozent*

SetSuperScript (OPTIONAL State%, nEsc%, nSize%)

Hochstellung an- oder ausschalten

Bezeichner	Typ	Beschreibung
State%	*SHORT*	*SET_OFF = aus* *SET_ON = an* *TOGGLE = Umschalten (Standard)*
nEsc%	*SHORT*	*Hochstellung in Prozent*
nSize%	*SHORT*	*Zeichengröße in Prozent*

SetTopBottomMargin(TopMargin&,BottomMargin&)

oberen und unteren Absatzabstand in 1/10mm setzen.

Bezeichner	Typ	Beschreibung
TopMargin&	*LONG*	*Oberer Absatzabstand in 1/10mm*
ButtomMargin&	*LONG*	*Unterer Absatzabstand in 1/10mm*

SetUnderline(OPTIONAL State%)

Unterstreichen an- oder ausschalten

Bezeichner	Typ	Beschreibung
State%	*SHORT*	*SET_OFF = aus* *SET_ON = an* *TOGGLE = Umschalten (Standard)*

SetWidow(Rows%)

Hurenkinderregelung setzen.

Bezeichner	Typ	Beschreibung
Rows%	*SHORT*	*Anzahl Zeilen*

ShrinkFontSize()

Verkleinern der Fontgröße

Gruppe Rahmen

AlignBottom()

Selektierten Rahmen am unteren Rand ausrichten.

AlignHorizontalCenter()

Selektierten Rahmen horizontal zentrieren.

AlignLeft()

Selektierten Rahmen am linken Rand ausrichten.

AlignRight()

Selektierten Rahmen am rechten Rand ausrichten.

AlignTop()

Selektierten Rahmen am oberen Rand ausrichten.

AlignVerticalCenter()

Selektierten Rahmen vertikal zentrieren.

SetFrameColumns(Count%, GutterWidth&, OPTIONAL LineAdj%, LineHeight%, PenStyle%)

Spalten in einem Rahmen setzen.

Bezeichner	Typ	Beschreibung
Count%	*SHORT*	*Anzahl der Spalten*
GutterWidth&	*LONG*	*Abstand zwischen den Spalten in 1/10mm*
LineAdj%	*SHORT*	*Trennlinien Ausrichtung* *0 = Oben* *1 = Zentriert* *2 = Unten*
LineHight%	*SHORT*	*Trennlinienhöhe in Prozentangabe*
PenStyle%	*SHORT*	*Trennlinienart; der Zahlenwert entspricht der Position der Auszeichnung innerhalb der Trennlinen-Listbox im entsprechenden Dialog. Erster Eintrag beginnt bei 0.*

SetFrameColumnWidth(Idx%, Width&)

Unterschiedliche Spaltenbreite für die Spalten eines Rahmens setzen.

Bezeichner	Typ	Beschreibung
Idx%	*SHORT*	*Nummer des Spalte (erste Spalte hat die Nummer 0)*
Width&	*SHORT*	*Breite der Spalte in 1/10mm*

SetFrameDist(Left&, Right&, Top&, Bottom&)

Rahmenabstände setzen.

Bezeichner	Typ	Beschreibung

Left&	*LONG*	*linker Abstand in 1/10mm*
Right&	*LONG*	*rechter Abstand in 1/10mm*
Top&	*LONG*	*oberer Abstand in 1/10mm*
Bottom&	*LONG*	*unterer Abstand in 1/10mm*

SetFrameMacro(MacName$, LibName$)

Einem Rahmen das Makro zuweisen, das bei einer Selektion des Rahmens ausgeführt wird.

Bezeichner	Typ	Beschreibung
MacName$	*STRING*	*Name des Macros*
LibName$	*STRING*	*Bereich, in dem das Macro sich befindet*

SetFrameOptions(FixContent%, Print%)

Rahmenoptionen setzen.

Bezeichner	Typ	Beschreibung
FixContent%	*BOOL*	*TRUE = Inhalt schützen* *FALSE = Inhalt nicht schützen (Standard)*
Print%	*BOOL*	*TRUE =Rahmen drucken (Standard)* *FALSE = Rahmen nicht Drucken*

SetFramePosition(hStyle%, vStyle%, OPTIONAL hPos&, vPos&, Fix%)

Rahmenposition setzen.

Bezeichner	Typ	Beschreibung
hStyle%	*SHORT*	*Art der Positionierung, folgende Werte sind erlaubt:* *0 = X-Koordinate der Position des Rahmens relativ zum Anker* *1 = Am rechten Rand der PrintArea oder der Seite ausrichten* *2 = Zentrieren auf Seite oder PrintArea* *3 = Links ausrichten*
vStyle%	*SHORT*	*Art der Positionierung, folgende Werte sind erlaubt:* *0 = Y-Koordinate der Position des Rahmens relativ zum Anker* *1 = Am oberen Rand der PrintArea oder der Seite ausrichten* *2 = Zentrieren auf Seite oder PrintArea* *3 = Am unteren Rand ausrichten*
hPos&	*LONG*	*Absolute horizontale Position des Rahmens; dieser Wert ist nur von Bedeutung, wenn hStyle% 0 ist.*
vPos&	*LONG*	*Absolute vertikale Position des Rahmens; dieser Wert ist nur von Bedeutung, wenn vStyle% 0 ist.*
Fix%	*BOOL*	*TRUE = feste Position und Größe* *FALSE = automatische Höhe (Standard)*

SetFrameSize(Width&, Height&)

Rahmengröße setzen.

Bezeichner	Typ	Beschreibung
Width&	*LONG*	*Breite des Rahmens in 1/10mm*
Height&	*LONG*	*Höhe des Rahmens in 1/10mm; Die Angabe einer negativen Höhe bedeutet, daß der Minimalwert für einen Rahmen mit automatischer Höhe gesetzt wird.*

SetFrameToTop()

Rahmen in den Vordergrund stellen.

SetFrameToBottom()

Rahmen in den Hintergrund stellen.

SetWrapOff

Rahmenattribut »kein Umlauf« setzen.

SetWrapOn

Rahmenattribut »Umlauf« setzen.

SetWrapThrough

Rahmenattribut »Durchlauf« setzen.

Gruppe Grafik

FlipGraphicHorizontal()

Grafik horizontal spiegeln (nur Bitmap-Grafiken).

FlipGraphicVertical()

Grafik vertikal spiegeln (nur Bitmap-Grafiken)

SetGraphicMirror(Type%)

Grafik spiegeln (nur Bitmap-Grafiken).

Bezeichner	Typ	Beschreibung
Type%	*SHORT*	*Art der Spiegelung:* *0 = nicht spiegeln* *1 = vertikal spiegeln* *2 = horizontal spiegeln* *3 = vertikal und horizontal spiegeln*

Gruppe Tabelle

DeleteTableColumns()

Löschen der selektierten Spalten einer Tabelle.

DeleteTableRows()

Löschen der selektierten Zeilen einer Tabelle.

InsertTableColumns(OPTIONAL Count%, After%)

Einfügen von Tabellenspalten.

Bezeichner	Typ	Beschreibung
Count%	*SHORT*	*Anzahl der einzufügenden Spalten (wird mit der Zahl der selektierten Zellen multipliziert).*
After%	*BOOL*	*Boolscher Wert der beschreibt, ob vor oder hinter dem Cursor eingefügt werden soll.* *TRUE = hinter dem Cursor einfügen (Standard)* *FALSE = vor dem Cursor einfügen*

InsertTableRows(OPTIONAL Count%, After%)

Einfügen von Tabellenzeilen.

Bezeichner	Typ	Beschreibung
Count%	*SHORT*	*Anzahl der einzufügenden Zeilen (wird mit der Zahl der selektierten Zellen multipliziert).*
After%	*BOOL*	*Boolscher Wert der beschreibt, ob vor oder hinter dem Cursor eingefügt werden soll.* *TRUE = hinter dem Cursor einfügen (Standard)* *FALSE = vor dem Cursor einfügen*

MergeTableCells()

Zusammenfassen der selektierten Tabellenzellen.

SetTableAlign(Align%)

Tabellenausrichtung setzen.

Bezeichner	Typ	Beschreibung
Align%	*SHORT*	*Art der Ausrichtung:* *0 = links* *1 = rechts* *2 = zentriert*

SetTableCellsReadOnly()

Zellenschutz Tabelle für die Zellen setzen.

SetTableDistance(Top&, Bottom&)

Abstand der Tabelle nach oben und unten.

Bezeichner	Typ	Beschreibung
Top&	*LONG*	*Abstand der Tabelle nach oben in 1/10mm*
Bottom&	*LONG*	*Abstand der Tabelle nach unten in 1/10mm*

SetTableGrid(LineType%, OPTIONAL ColorRed&, ColorGreen&, ColorBlue&)

Zellenumrandung der Tabelle einstellen.

Bezeichner	Typ	Beschreibung
LineType%	*SHORT*	*Linienart, der Zahlenwert entspricht der Position des Linientypes innerhalb der Linientyp-Listbox des entsprechenden Dialogs.*
ColorRed&	*LONG*	*Rotanteil der Linienfarbe*
ColorGreen&	*LONG*	*Grünanteil der Linienfarbe*
ColorBlue&	*LONG*	*Blauanteil der Linienfarbe*

SetTableRowHeight(OPTIONAL Height&)

Setzen einer festen Zeilenhöhe des selektierten Teils der Tabelle.

Bezeichner	Typ	Beschreibung
Height&	*LONG*	*die zu setzende Höhe in 1/10mm.*

SetTableShadow(ShadowLocation%, OPTIONAL ColorRed%, ColorGreen%, ColorBlue%, Width%)

Tabellenschatten festlegen.

Bezeichner	Typ	Beschreibung
ShadowLocation%	*SHORT*	*Position des Schattens, der Zahlenwert entspricht der Position der Angabe innerhalb der Schattenposition-Listbox des entsprechenden Dialogs.*
ColorRed&	*LONG*	*Rotanteil der Schattenfarbe*
ColorGreen&	*LONG*	*Grünanteil der Schattenfarbe*
ColorBlue&	*LONG*	*Blauanteil der Schattenfarbe*
Width%	*SHORT*	*Breite des Schattens*

SetTableWidth(Width&)

Tabellenbreite einstellen.

Bezeichner	Typ	Beschreibung
Width&	*LONG*	*Breite der Tabellen in 1/10mm* *-1 bedeutet automatische Breite*

SplitTableCell(OPTIONAL eDir%, Count%)

Aufteilen der selektierten Tabellenzellen.

Bezeichner	Typ	Beschreibung
eDir%	*SHORT*	*horizontal oder vertikal aufteilen:* *0 = Spalte*

		1 = Zeile
Count%	*SHORT*	*Anzahl, wie häufig geteilt werden soll*

UnsetTableCellsReadOnly()

Zellenschutz einer Tabelle aufheben

Gruppe Extras

AutoCorrect()

AutoKorrektur wird angestoßen; entspricht der Eingabe eines Leerzeichens bei der Texterfassung.

Calculate()

Selektierten Bereich berechnen. Das Ergebnis wird in die Zwischenablage gestellt.

ConfigColorsDlg()

Der Dialog Farbeinstellung wird aufgerufen.

ConfigKeyboard(OPTIONAL Key$)

Auf Tastaturkonfiguration Key$ umschalten. Wird der Parameter nicht angegeben, wird der Dialog Tastaturbelegung aufgerufen.

Bezeichner	Typ	Beschreibung
Key$	*STRING*	*Name der Tastaturbelegung*

ConfigMenu(OPTIONAL Menu$)

Auswahl der Menükonfiguration Menu$. Wird der Parameter nicht angegeben, wird der Dialog Menükonfiguration aufgerufen

Bezeichner	Typ	Beschreibung
Menu$	*STRING*	*Name der Menükonfiguration*

ConfigToolbox(OPTIONAL Toolbox$, Align%, Buttontype%)

Auswahl der Symbolleistenkonfiguration Toolbox$. Zusätzlich einstellung von Ausrichtung und Art der Darstellung

Bezeichner	Typ	Beschreibung
Toolbox$	*STRING*	*Name der Symbolleistenkonfiguration*
Align%	*SHORT*	*Ausrichtung:* *0 = Links* *1 = Oben* *2 = Rechts* *3 = Unten*
Buttontype%	*SHORT*	*Symbolbuttonart:* *0 = Symbol* *1 = Text* *2 = Symbol + Text*

ConvertTableText(OPTIONAL Delim$)

Konvertierung zwischen Text und Tabelle

Bezeichner	Typ	Beschreibung
Delim$	*STRING*	*Das Trennzeichen als Text*

GlossaryDlg()

Der Dialog Textbaustein einfügen wird aufgerufen.

HyphenateDlg()

Der Dialog Silbentrennung wird aufgerufen.

MacroDlg()

Der Dialog Makro wird aufgerufen.

NewGlossary(Group$, Name$, ShortName$)

Neuen Autotext anlegen.

Bezeichner	Typ	Beschreibung
Group$	*STRING*	*Name des Bereiches*
Name$	*STRING*	*Autotext-Name*
ShortName$	*STRING*	*Kürzel*

NumberingDlg()

Der Dialog Numerierung/Aufzählung wird aufgerufen.

NumberingOutlineDlg()

Der Dialog Kapitelnumerierung wird aufgerufen.

RecordMacroDlg()

Der Dialog Makro aufzeichnen wird aufgerufen.

SetActualGlossary(Name$)

Setzt die aktuelle Autotextgruppe

Bezeichner	Typ	Beschreibung
Name$	*STRING*	*Der Name der Gruppe*

SetEnglishMetric(On%)

Umschalten um zwischen einer metrischen Angabe von Maßeinheiten und der Angabe in Punkt. Dies wird nur bei der Wiedergabe von Makros wirksam, nicht bei deren Aufzeichnung.

Der Defaultwert ist die Verwendung einer metischen Angabe. Ein Makro, daß auf die Verwendung von Punkt als Maßeinheit umschaltet, sollte dies vor dem Beenden des Makros wieder rückgängig machen.

Bezeichner	Typ	Beschreibung
On%	*BOOL*	*TRUE = Einschalten der Verwendung von Point als Maßeinheit* *FALSE = Ausschalten der Verwendung von Point als Maßeinheit*

Sort(OPTIONAL Order%, Direction%, Delimiter$, Col1%, Type1%, Col2%, Type2%, Col3%, Type3%)

Selektierten Bereich sortieren. Werden keine Parameter angegeben, wird aufsteigend alphanumerisch zeilenweise sortiert.

Bezeichner	Typ	Beschreibung
Order%	*SHORT*	*0 = aufsteigend (Standard)* *1 = absteigend*
Direction%	*SHORT*	*Sortierrichtung:* *0 = Spalten* *1 = Zeilen (Standard)*
Delimiter$	*STRING*	*Trenner zwischen Spalten (Standard: Tabulator)*
Col1-3%	*SHORT*	*Spalte / Zeile des ersten, zweiten und dritten Schlüssels (Standard: 1)*
Type1-3%	*SHORT*	*Sortierart:* *0 = alphanumerisch (Standard)* *1 = numerisch*

SortDlg()

Der Dialog Sortieren wird aufgerufen.

SpellCheckerDlg()

Der Dialog Rechtschreibkorektur wird aufgerufen.

Gruppe Numerierung

DecNumLevel()
Aufzählungsliste herunterstufen.

DecNumOutlineLevel()
Punkte einer Aufzählungsliste mit ihren Unterpunkten nach unten verschieben.

GoToNextNumEntry()
Zum nächsten Eintrag der gleichen Ebene.

GoToPrevNumEntry()
Zum vorhergehenden Eintrag der gleichen Ebene.

IncNumLevel()
Aufzählungsliste heraufstufen.

IncNumOutlineLevel()
Punkte einer Aufzählungsliste mit ihren Unterpunkten nach oben verschieben.

MoveContentDown()
Selektierten Inhalt einer Aufzählungsliste nach unten schieben.

MoveContentUp()
Selektierten Inhalt einer Aufzählungsliste nach oben schieben.

MoveNumOutlineContentDown()
Punkte einer Aufzählungsliste mit ihren Unterpunkten herunterstufen.

MoveNumOutlineContentUp()
Punkte einer Aufzählungsliste mit ihren Unterpunkten heraufstufen

SetNumNoNumber()
Absatz ohne Numerierung einfügen

SetNumOff()
Aufzählungsliste ausschalten.

SetNumOn()
Aufzählungsliste einschalten.

Gruppe Fenster

ArrangeAllWin()

Alle Fenster anordnen.

ArrangeCascadeWin()

Fenster überlappend anordnen.

ArrangeHorizontalWin()

Fenster horizontal anordnen.

ArrangeVerticalWin()

Fenster vertikal anordnen.

NewWindow()

Neue Sicht auf das aktive Dokument öffnen.

Gruppe Hilfe

AboutDlg()

Der Dialog Info über StarWriter wird aufgerufen.

ShowHelpIndex()

Der Hilfeindex wird aufgerufen.