OWBasic 5.03
Befehlsreferenz

?	Ausgabe von Werten
ABS	Absoluter Betrag
ABS#	Absoluter Betrag
ACOS#	Umkehrung der Winkelfunktionen
ACOSH#	Inverse Hyperbelfunktionen
AND	Arithmetische Operatoren
APOTIME%	Zeit bis zum automatischen Abschalten
APPEND	Verwendung von mehr als einem MEMO
ARRAY	Feld-Vereinbarungen
ARRAYPARA	Felder als Prozedur-Parameter
ARRAYSIZE	Feldgröße ermitteln
ASC	Code eines Zeichens
ASIN#	Umkehrung der Winkelfunktionen
ASINH#	Inverse Hyperbelfunktionen
ASK	Frage an den Nutzer
ASK!	Frage an den Nutzer
ATAN#	Umkehrung der Winkelfunktionen
ATAN2	Arcus Tangens von 2 Variablen
ATAN2#	Arcus Tangens von 2 Variablen
ATANH#	Inverse Hyperbelfunktionen
BATTSTAT	Batteriezustand
BATTSTAT#	Batteriezustand
BIT	Wert eines Bit
BITMAP	Bitmaps in Integer-Felder speichern
BITS	Bits und Bytes
BOX	Box zeichnen
BREAK	Schleifensteuerung
BUZZER	Signalton
BYTE	Wert eines Bytes
BYTES	Zerlegen eines Integerwertes in Bytes
CALCINPUT#	Anzeige eines On-Screen-Taschenrechners
CARTX#	Umrechnung in kartesische Koordinaten
CARTY#	Umrechnung in kartesische Koordinaten
CASE	Mehrfache Alternativen
CEIL	Umwandlung vom Typ Float nach Integer
CEIL#	Umwandlung vom Typ Float nach Integer
CHARACTER	Zeichen eines Strings
CHAR	Zeichen zu einem Code
CHR$	Zeichen zu einem Code
CIRCLE	Kreis zeichnen
CLEAR	Löschen eines Feldes
CLEARF	Löschen eines Feldes

CLEARS	Löschen eines Feldes
CLIPBOARD	System-Clipboard
CLS	Löschen des Bildschirms
COMPERROR	Compiler-Fehler
CONST	Konstanten und initialisierte Felder
CONTINUE	Schleifensteuerung
CONTROL	Steueranweisungen
CONVERSION	Datentyp-Konvertierungen
COS	Winkelfunktionen
COS#	Winkelfunktionen
COSH#	Hyperbelfunktionen
COUNTC	Zählung von Zeichen
DATATYPE	Datentypen
DEC	Inkrementieren/Dekrementieren
DECF	Inkrementieren/Dekrementieren
DEFAULT	Definition eines Default-Datentyps
DEG#	Umrechnung von Winkeln in Grad
DIM	Feld-Vereinbarungen
DIRECTIVE	Direktiven
DO	Programm-Schleife
DRAWICON	Zeichnen eines Icons
DRAWSTRING	Text zeichnen
DRAWTEXT	Text zeichnen
DUP$	Zeichen vervielfachen
ELLIPSE	Ellipse zeichnen
ELSE	Bedingte Ausführung von Anweisungen
END	Ende der Programm-Abarbeitung
ENDCASE	Mehrfache Alternativen
ENDP	Definition von Anwenderprozeduren
ENUM	Deklaration von Aufzählungstypen
ERROR	Einen Fehler auslösen
ERROREXAMPLE	Beispiel zur Auwertung von Laufzeitfehlern
ERRORHANDLER	Fehlerbehandlung durch das Programm selbst
ERRORHANDLING	Fehler-Behandlung
ERROROFF	Fehlerbehandlung ausschalten
ERROROON	Fehlerbehandlung ausschalten
EXAMPLES	Beispiel-BASIC-Programme
EXEC	Start eines anderen BASIC-Programmes
EXITHANDLER	Abfangen des Programmabbruchs
EXP#	Exponentialfunktionen
EXP10#	Exponentialfunktionen
EXP2#	Exponentialfunktionen
EXPRESSION	Ausdrücke und Zuweisungen
FASTTIMER	Schneller Kurzzeit-Timer
FBBBYTE	Lesen und Schreiben im Dateipuffer (binär)
FBBFLOAT#	Lesen und Schreiben im Dateipuffer (binär)
FBBHASH	Errechnung des Hashwertes des Dateipuffers (binär)
FBBINIT	Lesen und Schreiben im Dateipuffer (binär)
FBBINT	Lesen und Schreiben im Dateipuffer (binär)
FBBPUTBYTE	Lesen und Schreiben im Dateipuffer (binär)

FBBPUTFLOAT	Lesen und Schreiben im Dateipuffer (binär)
FBBPUTINT	Lesen und Schreiben im Dateipuffer (binär)
FBBPUTSTRING	Lesen und Schreiben im Dateipuffer (binär)
FBBSEEK	Lesen und Schreiben im Dateipuffer (binär)
FBBSIZE	Aktuelle Größe des binären Dateipuffers
FBBSTRING$	Lesen und Schreiben im Dateipuffer (binär)
FBDATE	Lesen und Schreiben von Datum und Zeit im Dateipuffer
FBHASH	Errechnung des Hashwertes des Dateipuffers (Text)
FBINIT	Schreiben und Lesen im Dateipuffer (Text)
FBPUTDATE	Lesen und Schreiben von Datum und Zeit im Dateipuffer
FBPUTTIME	Lesen und Schreiben von Datum und Zeit im Dateipuffer
FBSTORE	Schreiben und Lesen im Dateipuffer (Text)
FBSTRING	Schreiben und Lesen im Dateipuffer (Text)
FBSTRINGEXAMPLE	Beispiel für das Lesen von Zeichenketten aus dem Dateipuffer
FBTIME	Lesen und Schreiben von Datum und Zeit im Dateipuffer
FFUNC	Prozeduren und Funktionen zur Arbeit mit reellen Werten
FILE	Zugriff auf Dateien mittels Dateizeiger
FILEBUFFER	Interner Datei-Puffer
FILEEXAMPLE	Beispiel für Zugriff auf Dateien mit Dateipointern
FILEFIND	Suchen eines Datensatzes nach dem Namen
FILELABEL$	Titel einer Datei
FILENEXT	Den nächsten Datensatz finden
FILEPREV	Den vorherigen Datensatz finden
FILEREAD	Lesen eines Datensatzes
FILEREMOVE	Datensatz löschen
FILEWRITE	Schreiben eines Datensatzes
FILEXCHG	Verschieben eines Datensatzes
FINDSHARED	Suche nach Datensätzen des shared File
FLASH	Zugriff auf Dateien im FlashRAM
FLASHREMAKE	Flash-Speicherverwaltung
FLASHSIZE	Gesamtgröße des Flash-Speichers und freier Speicher
FLOAT	Umwandlung in eine Gleitkommazahl
FLOAT#	Umwandlung in eine Gleitkommazahl
FLOOR	Umwandlung vom Typ Float nach Integer
FLOOR#	Umwandlung vom Typ Float nach Integer
FONT	Zeichensätze für Textdarstellung
FOR	Programm-Schleife
FORM	Formulare
FORMAT	Quelltext-Gestaltung
FORMBUTTON	Button in Formularen
FORMCHECKBOX	CheckBoxen in Formularen
FORMDISPLAY	Darstellung eines Formulars
FORMENTRY	Definieren eines Steuerelements eines Formulars
FORMEXAMPLE	Beispiel-Formular
FORMINIT	Initialisieren eines Formulars
FORMRADIOBUTTON	RadioButton in Formularen
FORMRESETSTATUS	Status des Formularelements ändern
FORMSELECTOR	Selektoren für Elemente von Formularen
FORMSETSTATUS	Change Status of Form Element
FORMTOUCH	Formular-Eingaben
FPOINTER	Funktionszeiger
FPTR	Funktionszeiger

FRACT#	Gebrochener Anteil einer Gleitkommazahl
FUNC	Definition einer Anwender-Funktion
FUNCTION	Prozeduren und Funktionen
GETFP	Dateizeiger zu einer Dateinummer ermitteln
GETICON	Bildschirminhalt als Icon ablegen
GETSYSVAR	OWBasic-Optionen abfragen/verändern
GLOBAL	Globale Variablen und Konstanten
GOSUB	Unterprogramm-Aufruf
GOTO	Sprung zu einer Marke
GRAD#	Umrechnung von Winkeln in Neugrad
GRAPHIC	Grafikprozeduren
GRAPHIO	Ein- und Ausgaben auf dem Grafik-Bildschirm
ICON	Bitmaps in Integer-Felder speichern
ICONHASH	Berechnen eines Hashwertes eines Icons
ICONPIX	Wert des Pixel im Icon
IF	Bedingte Ausführung von Anweisungen
IFUNC	Prozeduren und Funktionen zur Arbeit mit Integerwerten
IGRAPH	Interne Grafik darstellen
IN	Port-Eingabe
INC	Inkrementieren/Dekrementieren
INCF	Inkrementieren/Dekrementieren
INCLUDE	Verwendung von mehr als einem MEMO
INFLOCATE	Abfrage der Kursorposition
INPUT	Eingabe von Werten
INPUTF	Bearbeiten eines Variablenwertes
INPUTI	Bearbeiten eines Variablenwertes
INPUTS	Bearbeiten eines Variablenwertes
INPUTTEXT	Bearbeiten eines Variablenwertes
INPUTRC	Eingabe von Werten
INRANGE!	Test, ob ein Wert in einem vorgegebenen Bereich liegt
INSIDE!	Test, ob Koordinatenwerte in einem Fenster liegen
INSIDEG!	Test von Koordinatenwerten
INT	Umwandlung in eine Integer-Zahl
INT%	Umwandlung in eine Integer-Zahl
INTRO	Einführung
INTSIZE	Größe eines Integers in Byte
JUMP	Sprung über Prozeduren hinweg
KBDRAW	Tastatur darstellen
KBWAIT!	Auf Tasteneingabe warten
LABEL	Marken
LEFT$	Linker Teilstring
LEN	Stringlänge
LEVERPUSH	Status des Action-Reglers
LIGHT	Hintergrund-Beleuchtung schalten und abfragen
LIGHT!	Hintergrund-Beleuchtung schalten und abfragen
LINE	Linie zeichnen
LN#	Logarithmen
LOADFILE	Datei laden

LOADMEMO	Memo lesen
LOADMEMOFP	Memo lesen
LOADMEMONAME	Lade ein Memo nach seinem Namen
LOADQM	Einlesen eines Quickmemos
LOADQMFP	Einlesen eines Quickmemos
LOADQMHASH	Lesen eines Quickmemos selektiert über eine Hashwert
LOADSHARED	Zugriff auf die gemeinsame Datei
LOADSHAREDFLOAT	Speichern/Laden eines Gleitkomma-Arrays im Shared File
LOADSHAREDINT	Speichern/Laden eines Integer-Arrays im Shared File
LOCAL	Variablendeklaration
LOCATE	Setzen des Kursors
LOG#	Logarithmen
LOG2#	Logarithmen
LOOP	Programm-Schleife
LOWCASE$	Zeichenkette in Klein-/Großbuchstaben wandeln
MAX	Minimum und Maximum
MAX#	Minimum und Maximum
MESSAGEBOX	Darstellung einer Meldung
MID$	Teilstring
MIN	Minimum und Maximum
MIN#	Minimum und Maximum
MODE	Zugriff auf andere Anwendungen
MODEFIND	Findet ein Addin nach dem Namen
MODEICON	Menü-Icon eines Modus laden
MODEJUMP	Modus starten
MODELISTICON	List-Icon eines Modus laden
MODENAME	Name eines Modus bestimmen
MODEVER$	Version eines Modus bestimmen
MODES	Modi für Dateiarbeit
MSGBOX	Darstellung einer Message-Box auf dem Bildschirm
MULDIV	Kombinierte Multiplikation und Division
NAMEDFILE	Ermitteln von Modus und Submodus von Addin-Dateien
NEW25	Neu in Version 2.5
NEW30	Neu in Version 3.0
NEW32	Neu in Version 3.2
NEW40	Neu in Version 4.0
NEW42	Neu in Version 4.2
NEW44	Neu in Version 4.4
NEW50	Neu in Version 5.0
NEWTOUCH	Test auf Berührung
NEWTOUCH!	Test auf Berührung
NEXT	Programm-Schleife
NOT!	Logische Negation
NRT#	Wurzel
OF	Mehrfache Alternativen
OLDER25	Kleine Probleme mit älteren Programmen
OPERATOR	Arithmetische Operatoren
OPTIONS	Menüpunkt Options

OR	Arithmetische Operatoren
OUT	Port-Ausgabe
OVERLAY	Prozeduren als Overlay übersetzen
OWBVERSION	Information über das System
PARAMETER$	Aufrufparameter des aktuellen Programms
PARAMETERLIST	Parameterliste von Anwender-Prozeduren und -Funktionen
PEEK	Lesen von Bytes aus dem Hauptspeicher
PIXEL	Pixel lesen
PLAY	Spielen einer Melodie
POINTER	Zeigeroperationen
POINTEREXAMPLE	Beispiel zur Verwendung von Pointern
POKE	Schreiben von Bytes in den Hauptspeicher
POLPHI#	Umrechnung in Polarkoordinaten
POLR#	Umrechnung in Polarkoordinaten
POLYGON	Darstellung eines Polygons
POS	Suche nach Teilstring
POW#	Potenz
POWEROFF	Ausschalten des Systems
POWI#	Potenz mit ganzzahligem Exponenten
PRINT	Ausgabe von Werten
PROC	Definition von Anwenderprozeduren
PROCDESC	Format der Prozedur- und Funktions-Beschreibungen
PROCEDURE	Prozeduren und Funktionen
PROGRAMMING	Programmierung in OWBasic
PROGRAMMMENU	Definieren eines Programm-Menüs
PSET	Pixel setzen
PVMODEL	Information über das System
QUICKMEMO	Lesen und Schreiben von Quickmemos
RAD#	Umrechnung von Winkeln in Bogenmaß
RANDOMIZE	Neustart des Zufallszahlen-Generators
RCMODE	Fernbedienungsmodus
REMOVESHARED	Löschen eines Datensatzes der gemeinsamen Datei
RESIZEICON	Größe eines Icons ändern
RESTART	Neustart von OWBasic
RETURN	Ende der Subroutine
RIGHT$	Rechter Teilstring
RND	Zufallszahl
RND#	Zufallszahl
ROUND	Umwandlung vom Typ Float nach Integer
ROUND#	Umwandlung vom Typ Float nach Integer
RUNNING	OWBasic-Programme auf Pocketviewer starten
RUNTIMEERROR	Laufzeit-Fehler
SAVEFILE	Datei schreiben
SAVEMEMO	Schreiben eines MEMO
SAVEMEMOFP	Schreiben eines MEMO
SAVEQM	Speichern eines Quickmemos
SAVEQMFP	Speichern eines Quickmemos
SAVESHARED	Zugriff auf die gemeinsame Datei

SAVESHAREDFLOAT	Speichern/Laden eines Gleitkomma-Arrays im Shared File
SAVESHAREDINT	Speichern/Laden eines Integer-Arrays im Shared File
SCROLL	Verschieben eines Bildschirmausschnittes
SERIAL	Kommunikation über die serielle Schnittstelle
SETAPOTIME	Zeit bis zum automatischen Abschalten
SETBIT	Setzen eines Bit
SETBYTE	Setzen eines Bytes
SETFFORMAT	Format für Gleitkommazahlen festlegen
SETF	Setzen des Bereiches eines Arrays
SETI	Setzen des Bereiches eines Arrays
SETJUMP%	Sprung über Prozeduren hinweg
SETB	Setzen des Bereiches eines Arrays
SETC	Setzen des Bereiches eines Arrays
SETS	Setzen des Bereiches eines Arrays
SETICONPIX	Setzen eines Pixels im Icon
SETSYSVAR	OWBasic-Optionen abfragen/verändern
SETTIME	Setzen der Systemzeit
SFUNC	Prozeduren und Funktionen zur Arbeit mit Strings
SGN	Vorzeichen eines Integerwertes
SHAREDFILE	Shared File
SHL	Bitverschiebung nach links
SHL%	Bitverschiebung nach links
SHOW	Darstellung aktualisieren
SHR	Bitverschiebung nach rechts
SHR%	Bitverschiebung nach rechts
SIN	Winkelfunktionen
SIN#	Winkelfunktionen
SINH#	Hyperbelfunktionen
SOUND	Tonerzeugung
SPLIT	Zerlegung eines String mit Trennzeichen
SPLITN	Abspalten eine Anzahl von Zeichen von einem String
SQR	Quadratwurzel
SQR#	Quadratwurzel
SQR%	Quadratwurzel
SQUARE	Das Quadrat einer Zahl
SQUARE%	Das Quadrat einer Zahl
SRLCLOSE	Schließen der seriellen Schnittstelle
SRLOPEN	Serielle Schnittstelle öffnen
SRLRBUF	Status des Empfangspuffers der seriellen Schnittstelle
SRLRBYTE	Empfang eines Zeichens von der seriellen Schnittstelle
SRLSBYTE	
SRLSTAT	Status der seriellen Schnittstelle
SRLTBUF	Status des Sendepuffers der seriellen Schnittstelle
STARTTIMER	Timer
STEP	Programm-Schleife
STOPTIMER	Timer
STRING	Umwandlung in einen String
STRING$	Umwandlung in einen String
STRINGCONSTANTS	Zeichenketten-Konstanten
STRINGHEIGHT	Texthöhe für Darstellung ermitteln
STRINGSIZE	Textlänge für Darstellung ermitteln
SYNC	Mit einer internen Uhr synchronisieren

SYS	Aufruf spezieller System-Funktionen
SYSDLG	Aufruf eines System-Dialogs
SYSLANG	Systemsprache
SYSTEM	Systemnahe Prozeduren
TAN#	Winkelfunktionen
TANH#	Hyperbelfunktionen
TCHINSIDE!	Test, ob Touch-Koordinaten in einem Fenster liegen
TCHSET	Setzen des Touchscreen-Status
TEXT	Standard-Text-Ein- und Ausgabe
TEXTBOX	Textbox
THEN	Bedingte Ausführung von Anweisungen
TIME	Zeit und Datum
TIMER	Timer
TIMER#	Timer
TO	Programm-Schleife
TOUCH	Status des Touchscreen
TOUCH!	Status des Touchscreen
TOUCHED	Test auf Berührung einer definierten Fläche
TOUCHED!	Test auf Berührung einer definierten Fläche
TOUCHSCREEN	Touch-Screen-Abfrage
TRIGMODE	Winkelfunktions-Argumente
TRIM$	Zeichenkette ohne führende und nachfolgende Leerzeichen
UNTIL	Programm-Schleife
UPCASE$	Zeichenkette in Klein-/Großbuchstaben wandeln
USERFUNC	Definition einer Anwender-Funktion
USERPROC	Definition von Anwenderprozeduren
VAR	Parameterliste von Anwender-Prozeduren und -Funktionen
VARIABLES	Datentypen und Variablen
VERSION	Version setzen
WAIT	Pause in der Programm-Abarbeitung
WAITTOUCH	Warten auf eine Berührung des Touchscreen
WEND	Programm-Schleife
WHILE	Programm-Schleife
XCHGS	Austauschen zweier Strings
XOR	Arithmetische Operatoren
ZEICHENSATZ	Zeichensätze für Textdarstellung

ABS

Absoluter Betrag
ABS%(i%)
Der absolute Betrag von i.

ABS# - Absoluter Betrag

ABS#

Absoluter Betrag
ABS#(f#)
Der absolute Betrag von f.

APOTIME%

Zeit bis zum automatischen Abschalten
APOTIME%()
SETAPOTIME time%
Die Funktion APOTIME%() fragt die Zeit bis zum automatischen Abschalten des PV in 500ms-Einheiten ab. Mit der Prozedur SETAPOTIME kann diese verändert werden.
Wenn versucht wird, die Zeit auf einen illegalen Wert zu setzen, werden automatisch 6 Minuten eingestellt.

ARRAY

Feld-Vereinbarungen
DIM <bezeichner>[<last>]
DIM <bezeichner>[<last_row>,<last_column>]
Eindimensionale Felder (Array) werden mit der DIM-Anweisung deklariert. Der Index beim Zugriff auf Elemente des Arrays ist immer ein Integer-Wert und wird in eckigen Klammern angegeben.
Die DIM-Anweisung erfordert die Angabe einer Integer-Konstanten last als Feldgröße. Der Wert von last ist der letzte erlaubte Index in dem Feld.
Die Definition zweidimensionaler Felder erfordert die Angabezweier Integer-Konstanten last_row und last_column, die den maximalen Zeilen- und Spalten-Index-Wert angeben. Das Feld enthält dann last_row+1 Zeilen und last_column+1 Spalten.
Einfache Variablen - Variablen, die nicht mit DIM deklariert wurden - sind äquivalent zu Feldern mit der Größe size gleich 0.
Eindimensionale Felder sind äquivalent zu zweidimensionalen Feldern mit einer Spalte (last_column=0).
Zweidimensionale Felder können auch wie eindimensionale Felder mit last=(last_row+1)*(last_column+1)-1 verwendet werden.
Es ist (z.Z.) unmöglich, Feldgrößen zur Laufzeit zu ändern.
Die Größe eines Feldes kann zur Laufzeit mittels der Funktion ARRAYSIZE ermittelt werden.

Beispiel:

```
DIM a#[100] : ! Float-Array mit 101 Elementen
(Index:0..100)
FOR i=0 to 100
```

```
 a#[i]=FLOAT(i)*3.14
NEXT
```

ARRAYPARA

Felder als Prozedur-Parameter

Soll als Parameter einer Anwenderprozedur ein Feld übergeben werden, so funktioniert dies nur als Variablen-Parameter.
Das bedeutet, daß Prozeduren immer mit dem originalen Feld arbeiten und Änderungen sofort wirksam werden.
Im Prozedurkopf müssen Felder als VAR Parameter mit nachfolgendem leeren eckigen Klammernpaar definiert werden.
Beim Aufruf der Prozedur wird nur der Name des Feldes als aktueller Parameter angegeben.

Beispiel:

```
PROC set VAR f[], n
! setzt das Integer-Feld f auf den Wert n
 FOR i=0 TO arraysize(f)
  f[i]=n
 NEXT
ENDP

DIM A[10]: ! Feld anlegen
set a,9:   ! mit dem Wert 9 belegen
```

ARRAYSIZE

Feldgröße ermitteln

ARRAYSIZE%(ARR[][, mode%=0])

Die Funktion gibt die Größe (Index des letzten Feldelements) des eindimensionale Feldes ARR zurück. ARR kann eine Feld beliebigen Typs sein. Über mode kann die abgefragte Dimension des Arrays angegeben werden (bei zweidimensionalen Arrays), wobei mode die folgenden Werte haben kann:

mode	Ergebnis
0	Größe des Arrays
1	Größe der ersten Dimension
2	Größe der zweiten Dimension

Dies ist besonders nützlich in Prozeduren, bei denen die Feldgröße übergebener Parameter unterschiedlich sein kann:

```
proc setarray var a[],val
for i=0 to arraysize(a)
a[i]=val
next
```

ASC

Code eines Zeichens

ASC%(s$)
Ergibt als Integer-Wert den ASCII-Code des ersten Zeichens des String s$.

CHR$ - Zeichen zu einem Code

ASIN#

Umkehrung der Winkelfunktionen
ASIN#(v#), ACOS#(v#), ATAN#(v#)
Arcussinus, Arcuscosinus bzw. Arcustangens des Arguments v. Das Ergebnis ist ein Winkel im aktuellen Modus.

SIN# - Winkelfunktionen
COS# - Winkelfunktionen
TAN# - Winkelfunktionen

ASINH#

Inverse Hyperbelfunktionen
ASINH#(v#)
ACOSH#(v#)
ATANH#(v#)
Arcussinushyperbolicus, Arcuscosinushyperbolicus bzw. Arcustangenshyper-bolicus des Arguments v.

SINH# - Hyperbelfunktionen
SIN# - Winkelfunktionen

ASK!

Frage an den Nutzer
ASK!(question$)
ASK%(question$)
ASK!() stellt die Frage question in einer Box dar und erlaubt dem Nutzer, die YES- oder NO-Taste zu drücken. Der zurückgegebene Wert ist TRUE für YES, FALSE für NO.
Die Integer-Form ASK() (Ja - 1, Nein - 0) sollte zukünftig nicht mehr genutzt werden.
Beispiel:
neu (boolesch):

```
if ask!("Clear Screen") then
   cls 2
endif
```

alt (integer):

```
if ask("Clear Screen")>0 then
   cls 2
endif
```

ASK

Frage an den Nutzer
ASK%(question$)
Stellt die Frage question in einer Box dar und erlaubt dem Nutzer, die YES- oder NO-Taste zu drücken. Der zurückgegebene Wert ist 1 für YES, 0 für NO.

ATAN2#

Arcus Tangens von 2 Variablen
ATAN2#(s#, c#)
Die Funktion ist ähnlich dem Arcus-Tangens von (s/c). Sie liefert den Winkel, für den
s=r*SIN(phi)
c=r*COS(phi) .
phi ist ein Winkel im aktuellen Modus.

BATTSTAT

Batteriezustand
BATTSTAT%()
Die Funktion liefert einen Wert von 0 bis 1023, der ein Maß für die Batteriespannung ist.

BATTSTAT#

Batteriezustand
BATTSTAT#()
Die Funktion liefert die Batteriespannung in Volt.

BIT

Wert eines Bit
BIT%(ARR%[], nr%)
Die Funktion liefert den Wert des nr . Bits des Integerfeldes ARR . Mögliche Werte sind 0 und 1.

BITS

Bits und Bytes
Prozeduren für den Zugriff zu Bits und Bytes.

BYTES	Zerlegen einer Integerzahl in Bytes
SETBIT	Setzen eines Bits in einem Integerfeld
SETBYTE	Setzen eines Bytes in einem Integerfeld
BIT()	Wert eines Bits in einem Integerfeld
BYTE()	Wert eines Bytes in einem Integerfeld

BOX

Box zeichnen

BOX x1%, y1%, x2%, y2% [, z%=1]

Zeichnet einen Rahmen mit der linken oberen Ecke (x1,y1) und der rechten unteren Ecke (x2,y2) mit dem gegebenen Wert z.

Wert	Funktion
0	Box löschen (Weiß)
1	Box setzen (Schwarz)
2	Box löschen (Weiß)
4	Punktierte Linien
5	Box invertieren
10	Löschen des Box-Inneren
11	Invertieren des Box-Inneren

Um das Aussehen der Box besser beeinflussen zu können, kann mit Modi größer als 1000 jedes einzelne Boxelement spezifiziert werden. Diese Werte sind die Summe der entsprechenden Teile:

Boxelement	Wert	Darstellung
Innenfläche	0	unverändert
Innenfläche	1	Schwarz
Innenfläche	2	Weiß
Innenfläche	3	Schattiert
Innenfläche	5	Invers
Rand	0	unverändert
Rand	10	Schwarz
Rand	20	Weiß
Rand	30	Punktiert
Rand	50	Invers
Schatten	0	unverändert
Schatten	100	Schwarz
Schatten	200	Weiß
Schatten	300	Punktiert
Schatten	500	Invers

SHOW - Darstellung aktualisieren

BREAK

Schleifensteuerung

BREAK

CONTINUE

Diese Anweisungen ermöglichen das Verlassen einer Schleife (BREAK) bzw. das Springen zum nächsten Durchlauf (CONTINUE).
BREAK und CONTINUE sind für FOR..NEXT, WHILE..WEND und DO..LOOP-Schleifen implementiert.
Wie der GOTO-Befehl können auch BREAK und CONTINUE mit einzeiligen IF-Statements verwendet werden.

Beispiel:

```
n=0
FOR i=0 TO 100
 IF i=50 CONTINUE
 INC n,rnd(i)
 IF n>=800 BREAK
NEXT
PRINT n
```

BUZZER

Signalton

BUZZER n%

Signalton-Ausgabe

n	Reaktion
0	Dauerton
>0	n Töne pro Sekunde
<0	Ton aus

BYTE

Wert eines Bytes

BYTE%(bm%[], nr%)

Die Funktion liefert den Wert des nr. Bytes des Integerfeldes bm . Mögliche Werte liegen von 0..255.

BYTES

Zerlegen eines Integerwertes in Bytes

BYTES i%, BL%, BH%

Die Prozedur zerlegt den Integerwert i in die Bytes BL (niederwertiges Byte) und BH (höherwertiges Byte). Die Bytewerte werden in den Integervariablen BH und BL abgelegt und liegen im Bereich von 0..255 .

CALCINPUT#

Anzeige eines On-Screen-Taschenrechners

CALCINPUT#()

Ein Tastenfeld eines Taschenrechners wird in der unteren Bildschirmhälfte dargestellt. Es ist möglich, Zahlen einzugeben und Rechnungen auszuführen. Nach dem Drücken der '='-Taste oder von "NEXT" wird das Ergebnis zurückgegeben.

CARTX#

Umrechnung in kartesische Koordinaten

CARTX#(r#, phi#)
CARTY#(r#, phi#)

Berechnet die kartesischen Koordinaten x und y, die den Polarkoordinaten r und phi entsprechen. Der Winkel phi muß im aktuellen Modus angegeben sein.

CASE

Mehrfache Alternativen

```
CASE SELECT <Value>
CASE <Value1>:                          <Instructions1>
CASE <compasism-operator><Value2>:  <Instructions2>
CASE <Value3> TO <Value4>:          <Instructions3>
CASE ELSE:                              <DefaultInstructions>
ENDCASE
```

Mehrfache Alternativen von Anweisungen. Wenn der bei CASE SELECT gegebene Wert mit dem Wert eines CASE-Zweigs übereinstimmt, werden die dazugehörigen Anweisungen ausgeführt.
Nur genau ein CASE-Zweig, der erste, wird ausgeführt, auch wenn mehrere Werte übereinstimmen sollten. Der CASE ELSE-Zweig stellt eine Alternative dar, die abgearbeitet wird, wenn kein anderer Zweig zutrifft. Dieser Zweig muß, wenn vorhanden, der letzte definierte CASE-Zweig sein. Als Wert Value, Value1 usw. können auch Ausdrücke verwendet werden. Die Werte können vom Typ Integer, Char oder Boolesch sein.

Beispiel:

```
CASE SELECT a
CASE <1:      PRINT "below 1"
CASE 1:       PRINT "one"
CASE =2:      PRINT "two"
CASE 3:       PRINT "three"
CASE 4 TO 6:  PRINT "four, five or six"
CASE >=7:     PRINT "above or equal to seven"
ENDCASE
```

CHAR

Zeichen zu einem Code

```
CHAR&(i%)
CHAR(i%)
```

Gibt das Zeichen zurück, das dem ASCII-Wert i entspricht.

CHARACTER

Zeichen eines Strings

Auf die einzelnen Zeichen eines Strings kann (lesend) zugegriffen werden, indem an die Stringvariable (oder das String-Feldelement) in geschweiften Klammern der Index angehängt wird. Das Ergebnis ist ein aus einem Zeichen bestehender String.

Beispiel:

```
DIM b$[2]
a$="Test-String"
```

```
b$[0]="Wolfgang"
b$[1]="Ortmann"
? a${5} :! Ausgabe von 'S'
? b$[0]{6},b$[1]{5},b$[1]{6} :! Ausgabe von 3 'n'
```

CHR$

Zeichen zu einem Code
CHR$(i%)
Ergibt einen String, bestehend aus dem Zeichen mit dem Code i.

CHR&

Zeichen zu einem Code
CHR&(i%)
Ergibt einen Char, bestehend aus dem Zeichen mit dem Code i.

CIRCLE

Kreis zeichnen
CIRCLE xm%, ym%, r% [,z%=1]

Zeichnet einen Kreis um den Mittelpunkt (xm,ym) mit dem Radius r und dem gegebenen Wert z.
Es wird nur der sichtbare Teil des Kreises gezeichnet.

z	Funktion
0	Ellipse löschen (Weiß)
1	Ellipse setzen (Schwarz)
2	Ellipse löschen (Weiß)
5	Ellipse invertieren

CLEAR

Löschen eines Feldes
CLEAR A%
CLEARF A#
CLEARS A$
Löscht das Feld A, das heißt, alle Werte werden auf Null beziehungsweise den Leerstring gesetzt.

CLIPBOARD

System-Clipboard
Über die globale Variable CLIPBOARD kann auf den System-Zwischenspeicher zugegriffen werden. CLIPBOARD ist byteweise organisiert und kann nahezu wie ein normales Array abgefragt und zugewiesen werden; bezüglich der Übergabe als Variable gelten die gleichen Restriktionen wie bei FILEBUFFER.

FILEBUFFER – Interner Datei-Puffer

CLS

Löschen des Bildschirms
CLS [v%=1]
CLRSCR [v%=1]
Löschen des Bildschirms.

v	Funktion
1	(Default) Löschen des Textbildschirmes
2	Löschen des (grafischen) Bildschirms

CONST

Deklaration konstanter Variablen
CONST <identifier>=<constant>
Deklaration initialisierter Felder
CONST <identifier>=(<constant>, <constant> …)

Mit CONST können Integerkonstanten oder initialisierte Felder definiert werden.
In der ersten Form werden Integerkonstanten definiert, die später im Quelltext anstelle von Literalen verwendet werden können:

```
CONST adim=65
DIM a[#adim], b[#adim]
FOR i=0 to #adim : ....
```

Die Verwendung von Konstanten erleichtert das Lesen des Programms, wenn die Namen sinnvoll verwendet werden. Wenn im Beispiel eine Änderung von adim notwendig würde, würden durch die Änderung der Konstante gleich alle signifikanten Stellen angepaßt werden, also die DIM-Anweisungen und die Schleife.

Zur Compilezeit kann einer Integerkonstante ein anderer konstanter Wert zugewiesen werden:
#<constant>=<constant integer expression>

In der zweiten Form werden initialisierte Felder angelegt. Der Typ ergibt sich nach den üblichen Konventionen aus dem Namen. Die Feldgröße wird durch die Zahl der gegebenen Konstanten bestimmt. Konstante Felder können genauso wie andere Felder benutzt werden, sind jedoch beim Programmstart mit den gegebenen Werten initialisiert.

Beispiel:

```
CONST i=(8,8,426,384,384,21888) : ! Konstantes Integer-
      Feld
CONST s$=("one", "two", "three") :! Konstantes String-
      Feld
CONST f#=(0.2,0.4,0.6) :! Konstantes Float-Feld
CONST c&=("my char var") :! Konstantes Char-Feld
CONST &ps$=(&s) :! Konstantes String-Pointer-Feld
```

```
CONST b!=(true,false,1,0,0,1) :! Konstantes Bool-Feld
DRAWICON i,5,5
```

CONTROL

Steueranweisungen
Es gibt die folgenden Steueranweisungen:

IF	Bedingte Abarbeitung von Anweisungen
IF..GOTO	Bedingter Sprung
CASE	Mehrfache Alternative
FOR	Programm-Schleife
GOTO	Sprung zu einer Marke
GOSUB	Unterprogrammaufruf
PROC, FUNC	Anwender-Prozeduren- und Funktionendefinition
EXEC	Ein anderes Basic-Programm starten

COUNTC

Zählung von Zeichen
COUNTC%(c&, s$)
Die Funktion ermittelt, wie oft Zeichen c in String s vorkommt.

LEN - Stringlänge

DATATYPE

In OWBasic finden die fünf Datentypen Integer, Float, String, Char und der logische Typ Boolesch Anwendung. Der Typ von Variablen und Funktionen ergibt sich aus dem Defaulttyp oder einem angehängten Suffix. Bei Variablen kann nach einer ersten Verwendung das Suffix auch weggelassen werden, wenn nicht mit Variablen verschiedenen Types, aber gleichen Namens gearbeitet wird.

Suffix	Typ	Variable	Konstante	Zahlenbereich
%	Integer	A%	3107	-32768..32767
#	Float	B#	3.14159	-2147483648..2147483647
$	String	C$	"Hallo"	maximal 149 Zeichen
!	Bool	D!	TRUE,FALSE	TRUE,FALSE
&	Char	C&	'c'	CHR&(0..255)

Integerkonstanten können auch hexadezimal angegeben werden, indem '0x' vor den Wert gesetzt wird, wie z.B. 0x7ffe, 0x0D oder 0x123A.

DEFAULT

Definition eines Default-Datentyps
DEFAULT INTEGER
DEFAULT FLOAT
DEFAULT STRING

DEFAULT <bezeichner>

DEFAULT setzt den gegebenen Datentyp als Defaulttyp für Variablen und Funktionen ein. Dies wirkt im Quellfile bis zur Neudeklaration.Als Parameter sind INTEGER, FLOAT und STRING möglich. Es kann aber auch ein beliebiger Bezeichner mit Suffix verwendet werden, um einen Typ auszuwählen. Der Bezeichner selbst hat hier keine weitere Bedeutung.

VARIABLES - Datentypen und Variablen
DATATYPE - Datentypen
CONST - Deklaration konstanter Felder

DEG#

Umrechnung von Winkeln in Grad

DEG#(phi#)

Wenn phi ein Winkel im aktuellen Modus ist, liefert die Funktion DEG den entsprechenden Winkel in Grad zurück.

TRIGMODE - Winkelfunktions-Argumente
RAD# - Umrechnung von Winkeln in Bogenmaß
GRAD# - Umrechnung von Winkeln in Neugrad

DIRECTIVE

Direktiven

#IF <constant Boolean expression>
<code>
#ENDIF

Durch die Verwendung von Direktiven kann eine bedingte Kompilierung erzwungen werden; so kann z.B. abhängig vom PV-Modell verschiedener Code kompiliert werden.
Der Code zwischen #IF und #ENDIF wird nur kompiliert, wenn die Bedingung TRUE ergibt.

Mit Konstanten und Direktiven kann man z.B. eine Art Debug-Schalter implementieren, d.h., am Anfang eines Programms etwa die Konstante DEBUG deklarieren und mittels Direktiven-Abfrage, solange die Konstante definiert ist, Debug-Ausgaben machen.

Beispiel:

```
! directives
#if #pvmodel=1
pv$="PV-x50X"
#endif
#if #pvmodel=2
pv$="PV-S750(+)"
#endif
#if #pvmodel=3
pv$="PV-Sx50"
#endif
#if #pvmodel=4
pv$="PV-Sx60"
#endif
```

```
#if #pvmodel=5
pv$="PV-S1600"
#endif

CONST MAX=12
#if not vardef(MAX)
CONST MAX=13
#endif

#if not procfuncdef!(myproc)
proc myproc a,b
print a,b,a*b
endp
#endif

#if 5*7=35
print "ich kann kopfrechnen"
#endif
```

DO

Programm-Schleife

DO [UNTIL|WHILE] <boolean expression> <instructions>
LOOP [UNTIL|WHILE] <boolean expression>

Allgemeine Schleife. Eine Abbruchbedingung kann am Anfang oder am Ende der Schleife formuliert werden. Mit dem Schlüsselwort WHILE wird die Schleife abgearbeitet, solange der boolesche Ausdruck wahr (TRUE) ist. Mit UNTIL erfolgt die Abarbeitung solange, bis der boolesche Ausdruck wahr ist.

DRAWICON

Zeichnen eines Icons

DRAWICON A%[], x%, y% [, mode%=0] [, yoffs1%, yoffs2%]

Das durch das Integer-Feld a beschriebene Icon wird an der Position (x,y) dargestellt. mode beschreibt die Art der Darstellung:

mode	Darstellung
0	Normal (überschreiben des Bildschirminhalts)
1	OR-Verknüpfung mit dem Bildschirminhalt
2	AND-Verknüpfung mit dem Bildschirminhalt
3	Umkehren
4	Schattieren

Modus 3 und 4 sind nur auf den alten PVs verfügbar; werden sie auf dem PV-S1600 angewandt, so wird das Icon mit Mode 0 dargestellt.
Wenn yoffs1 und yoffs2 angegeben sind, so wird der Teil des Icons von yoffs1 bis yoffs2 dargestellt.

SHOW - Darstellung aktualisieren

IGRAPH - Interne Grafik darstellen
CONST - Deklaration konstanter Felder

DRAWSTRING

Text zeichnen
DRAWSTRING s$, x%, y% [,font%=0] [, xend%=159]
Der String s wird an die Position (x,y) gezeichnet. Dabei wird der Font font verwendet. Wenn der Text weiter als bis zu xend reichen würde, wird er begrenzt.

font	Font
0	Normal
1	Fett
2	FixedPitch
3	Groß
4	Klein
5	Sehr klein

Wenn xend <0 ist, wird der String rechtsbündig zu x dargestellt.

Der "kleine" Zeichensatz 4 ist nur für die Zeichen von CHR$(32) bis CHR$(127) mit den CASIO Zeichensätzen kompatibel.
Der "sehr kleine" Zeichensatz ist nur für die Darstellung von Zahlen und den Zeichen 'A', 'P' und 'M' brauchbar.

DRAWTEXT

Text zeichnen
DRAWTEXT s$, x1%, y1%, x2%, x2%[,font]
Der String s wird in die Mitte des Fensters von (x1,y1) bis (x2,y2) gezeichnet. Ist der Font nicht angegeben, wird er in Abhängigkeit von der Fenster-Größe gewählt.

DUP$

Zeichen vervielfachen
DUP$(c&,n%)
Gibt einen String zurück, der das Zeichen c n mal enthält.

ELLIPSE

Ellipse zeichnen
ELLIPSE xm%, ym%, rx%, ry% [,z%=1]
Zeichnet eine achsenparallele Ellipse um den Mittelpunkt (xm,ym) mit den Halbachsen rx und ry mit dem gegebenen Wert z.
Es wird nur der sichtbare Teil der Ellipse gezeichnet.

z	Funktion
0	Ellipse löschen (Weiß)

1	Ellipse setzen (Schwarz)
2	Ellipse löschen (Weiß)
5	Ellipse invertieren

END

Ende der Programm-Abarbeitung

END [n]

Beendet die Programm-Abarbeitung insgesamt. Der Parameter n steuert das weitere Verhalten:

n	Verhalten
0	Rückkehr zum PV-Menu
1	Warten auf Berührung
2	Zeige Meldung und warte auf Berührung

Das Standard-Verhalten bei Weglassen des Parameters n kann im Options-Menu definiert werden.

ENUM

Deklaration von Aufzählungstypen

ENUM <name>=(<constant 1>[=<value>], <constant 2>[=<value>], …)

Durch die ENUM-Direktive können eigene Aufzählungstypen deklariert werden. Variablen von diesem Typ kann nur eine der bei der Deklaration angegebenen Konstanten zugewiesen werden.
Soll einer ENUM-Variable ein Integerwert zugewiesen werden, so muß dieser explizit konvertiert werden:

<varname>=ENUM\<type>(<integer value>)

Die Werte der Konstanten werden durchnumeriert. Wird für eine Konstante ein Wert angegeben, so ist der Wert der nächsten Konstanten jeweils 1 höher.

Beispiel:

```
ENUM daytime_t=(MORNING=0, NOON, AFTERNOON, EVENING,
NIGHT)
CONST daytimes$=("morning", "noon", "afternoon",
"evening", "night")

PROC show_daytime daytime\daytime_t
 PRINT daytimes[INT(daytime)]
 IF daytime=#NIGHT THEN
  POWEROFF
 ENDIF
ENDP

show_daytime #NOON
```

ERROR

Einen Laufzeitfehler auslösen

ERROR procname$, number

Diese Prozedur löst einen Fehler der gleichen Art wie eingebaute Prozeduren aus. Dies kann in Nutzer definierten Prozeduren genutzt werden. ERROR erfordert als Parameter den Prozedurnamen und die Nummer der Fehlermeldung:

number	Meldung
0	Syntax error
1	Unbalanced parenthesis
2	No expression present
4	L-value expected
5	Symbol table full
6	Double defined label
8	Call stack underflow
9	Call stack overflow
10	Too many nested controls
11	Controls unbalanced
12	Type already exists
13	Last code reached
14	Expression delimiter expected
15	No open IF
16	Unknown type
17	Unknown function
18	No open FOR
19	Default argument missing
20	Undefined constant
21	Unclosed IF
22	Type mismatch
23	Unclosed FOR
24	Undefined variable
27	Not implemented
28	Global variable exist
29	2D-Array needed
30	Index out of range
31	Array variable used before DIM
32	String too long
33	Memory full
34	String stack underflow
35	String stack overflow
36	Negative argument
37	Label expected
38	Wrong parameter
39	Failed
40	Unknown procedure
41	Invalid code address
42	Include stack overflow
43	Include file not found
44	Append file not found
45	Error position found
46	Too many parameters
47	Too less parameters
48	Nested procedure
49	ENDP without PROC
50	PROC or FUNC not at begin
51	Wrong version

52	Procedure already exists
53	Interrupted
54	Undefined label
55	Expected constant expression
56	Pointer expected
57	Division by 0
58	Wrong mode
59	No valid icon
60	No more data
61	Not found
62	Wrong font
63	Missing ENDP
64	Arrays must be VAR parameter
65	Coordinates out of range
66	VAR parameters not allowed here
67	Not initialized
68	Can't save overlay
69	Can't load overlay
70	Same overlay
71	File too big

ERRORHANDLER

Fehlerbehandlung durch das Programm selbst

Dem globalen Array ERRORHANDLER kann mittels SETJUMP%() eine Codeposition zugewiesen werden, die nach Auftreten eines Laufzeitfehlers aufgerufen wird. Wenn wieder die normale Fehlerbehandlung angewendet werden soll, muß ERRORHANDLER 0 zugewiesen werden. Dieser Mechanismus ist dem try-catch-Mechanismus in C++ ähnlich.

ERRORHANDLING

Fehler-Behandlung

Bei den Fehlern ist zwischen Compilerfehlern und Laufzeitfehlern zu unterscheiden:
Fehlermeldungen, die bei der Übersetzung der Programms durch Syntax-Fehler enstehen sind Compiler-Fehler.
Fehler, die bei der Abarbeitung des Programmes auftreten, sind Laufzeitfehler.

ERROREXAMPLE

Beispiel zur Auwertung von Laufzeitfehlern

Das Beispiel zeigt, wie aufgetretene Laufzeitfehler im Programm ausgewertet werden können:

```
! alle MEMOs der Kategorie 5 lesen
i=0
$m1
```

```
SYS 7: ! Fehlermeldungen ausschalten
LOADMEMO a$,5,i: ! Memo lesen
 SYS 6: ! Fehlermeldungen wieder einschalten
 if error>0 GOTO ende: ! Kein Memo mehr da
 PRINT a$:  ! erste Zeile ausgeben
 INC i
GOTO m1
$ende
PRINT "Fertig"
```

ERROROFF

ERROROFF
ERRORON

Die normale Fehlerbehandlung wird ein- oder ausgeschaltet Ist die Fehlerbeahndlung ausgeschaltet, so wird kein Fehler gemeldet und das Programm bricht nicht ab. Es ist Aufgabe des Programmes auf Fehler zu korrigieren. Dazu wird die Variable ERROR gesetzt.

ERROROFF und ERRORON müssen unbedingt paarweise verwendet werden!

EXAMPLES

Beispiel-Basic-Programme

Diese Dokumentation enthält auch einige einfache Basic-Programme als Beispiele.

EXEC

Start eines anderen OWBasic-Programmes

EXEC progname$[, cat%=-1][, showmsg!=TRUE][, param$]

Start des angegebenen Programmes progname.
Dies ist ein Neustart von OWBasic. Variablen des alten Programmes sind im neu gestarteten Programm nicht verfügbar. Das Programm wird in der Programm-Kategorie gesucht oder in der Kategorie cat, wenn der Parameter angegeben ist. Mit showmsg=FALSE kann die Darstellung des Compiler-Bildschirms unterbunden werden.
Über den optionalen Parameter param kann dem aufgerufenen Programm ein String-Parameter übergeben werden, den es mittels PARAMETER$ abfragen kann.

EXITHANDLER

Abfangen des Programmabbruchs

Dem globalen Array EXITHANDLER kann mittels SETJUMP% eine Codeposition zugewiesen werden, die bei Abbruch des Programmes aufgerufen wird.
Soll das OWBasic-Programm bei Abbruch normal beendet werden, so muß EXITHANDLER 0 zugewiesen werden.

EXP#

Exponentialfunktionen
EXP#(f#)
EXP10(f#)
EXP2(f#)
Die Exponentialfunktionen zur Basis e (Eulersche Zahl), 10 und 2.

EXPRESSION

Ausdrücke und Zuweisungen
Ausdrücke sind syntaktische Konstruktionen, die einen Wert repräsentieren. Sie werden als Parameter von Prozeduren and Funktionen und als rechte Seite von Zuweisungen benötigt. Boolesche Ausdrücke dienen auch als Bedingung für bedingte Operationen (IF). Wie auch Variablen haben Ausdrücke einen Datentyp. Grundlegende Bestandteile von Ausdrücken sind:
- Konstanten, z.B. 3, 3.12, "a string", true
- Variablen, z.B. A, F#, valid!
- Operatoren, z.B. +, -, AND, OR
- Funktionen, z.B. SIN#(F), SQR()

Zuweisungen haben die Form:
<variable> = <ausdruck>
Der Wert, den der Ausdruck <ausdruck> repräsentiert, wird der Variablen zugewiesen. Das heißt, die Variable hat danach den Wert von <ausdruck> und der vorherige Wert der Variablen geht verloren.
Konstant ist ein Ausdruck, der zur Compilezeit berechnet werden kann, z.B. 5*(9-7) . Die Funktionen VARDEF und PROCFUNCDEF ergeben einen konstanten Ausdruck; ebenso die Funktionen INT, CHAR, ASC, vorausgesetzt, der Funktionsparameter ist ebenfalls konstant.
Konstante Ausdrücke können als Bedingung in Direktiven verwendet werden.

FASTTIMER

Schneller Kurzzeit-Timer
Ein schneller Timer kann benutzt werden, um kurze Zeiten zu messen. Wenn er mit SYS 10 eingeschaltet wurde, beginnt die Zählung bei Null. Die Zählung erfolgt mit 40 Schritten pro Sekunde. Die Abfrage des Wertes des Zählers erfolgt mit TIME(8). Mit SYS 11 sollte der Timer ausgeschaltet werden, wenn er nicht mehr benötigt wird. Die Prozeduren MESSAGEBOX und BUZZER benötigen den Timer ihrerseits und unterbrechen den normalen Zählvorgang.
Beispiel:

```
STARTTIMER :! Schnellen Timer starten
wait 10 :! ca 10 Sekunden warten
? timer#() :! Wirklich vergangene Zeit ausgeben
STOPTIMER :! Schnellen Timer stoppen
```

FBBBYTE

Lesen und Schreiben im Dateipuffer (binär)

FBBINIT
FBBSEEK nr%
FBBTELL%()
FBBREAD%()
FBBINT%()
FBBFLOAT#()
FBBSTRING$()
FBBPUTBYTE b%
FBBPUTINT i%
FBBPUTFLOAT f#
FBBPUTSTRING s$
FBBSIZE%

Routinen zum sequentiellen Lesen und Schreiben von Daten im Dateipuffer. Nach dem Rücksetzen (FBBINIT) oder Setzen (FBBSEEK) des internen Zeigers lesen oder schreiben die Prozeduren Daten und setzen den Zeiger entsprechend weiter.
FBBINIT - Rücksetzen des internen Zeigers auf den Anfang des binären Dateipuffers.
FBBSEEK nr% - Setzen des internen Zeigers auf das angegeben Byte des binären Dateipuffers.
FBBTELL%() – Abfragen der Position des internen Zeigers.
FBBSIZE% (Variable) - Aktuelle Größe des binären Dateipuffers

Schreibprozedur	Lesefunktion	Datentyp
FBBPUTBYTE b%	FBBREAD()	Byte
FBBPUTINT i%	FBBINT()	Integer
FBBPUTFLOAT f#	FBBFLOAT#()	Float
FBBPUTSTRING s$	FBBSTRING$()	String

Die schreibenden Prozeduren setzen den Wert für die Größe des binären Dateipuffers unter der Annahme, daß der geschriebene Wert der letzte Wert ist. Wird in die Mitte eines gefüllten Puffers geschrieben, so muß danach die Größe des binären Dateipuffers an FBBSIZE zugewiesen werden.

FBBHASH

Berechnung des Hashwertes des Dateipuffers (binär)

FBBHASH CRC1%, CRC2% [max%=-1] [,mode%]

Berechnet den CRC-Wert des aktuellen binären Dateipuffers und speichert ihn in CRC1 und CRC2. Mittels mode (1=default; 2=deprecated) kann der CRC-Algorithmus gewählt werden.
Ist max angegeben, so wird der Hashwert nur bis zur Position max des Filebuffers errechnet. Wenn max nicht oder als -1 angegeben wird, so wird der Hashwert für die aktuelle Größe des Filebuffers errechnet.

FBDATE

Lesen und Schreiben vonDatum und Zeit im Dateipuffer

FBDATE$(sel%)
FBPUTDATE(sel%,date$)
FBTIME$(sel%)
FBPUTTIME(sel%,time$)

In einigen Modes enthalten Dateien Daten und Zeiten. Ein Datum wird mit einer 8 Zeichen umfassenden Zeichenkette dargestellt, die Zeit erfordert 4 Zeichen. Die Form ist
YYYYMMDD - Y = Jahr, M = Monat, D = Tag
HHMM - H = Stunde (24h), M = Minute

*	Datum	Zeit
Lesen	FBDATE	FBTIME
Schreiben	FBPUTDATE	FBPUTTIME

CASIO definiert folgende Daten/Uhrzeiten:

sel	Datum	Zeit
0	Date/DueDate	Time/DueTime(From)
1	-	Time/DueTime(To)
2	CheckDate	CheckTime
3	AlarmDate	AlarmTime
4	SearchBufferDate	SearchBufferTime

FBHASH

Berechnung des Hashwertes des Dateipuffers (Text)

FBHASH CRC1%, CRC2% [,mode%]

Berechnet den CRC-Wert des aktuellen Text-Dateipuffers und speichert ihn in CRC1 und CRC2. Mittels mode (1=default; 2=deprecated) kann der CRC-Algorithmus gewählt werden.

FBSTRING

Schreiben und Lesen im Dateipuffer (Text)

FBINIT
FBSTRING$(RC%[, del%=1])
FBSTORE s$[, del%=1]

Prozeduren zum sequentiellen Zugriff auf den Dateipuffer für Textmode-Dateien.
FBINIT initialisiert den Zugriff und setzt den internen Zeiger auf den Anfang.
FBSTRING$ liefert die nächste Zeile des Textes im Dateipuffer. Der Variablenparameter RC ist nach der Rückkehr ungleich Null, falls keine Zeile mehr vorhanden war.
FBSTORE speichert den übergebenen String als Zeile (mit Zeilenvorschub) im Dateipuffer. Der optionale Parameter del steuert, welches Zeichen als Begrenzer der Zeilen/Zeichenketten zu verwenden ist:

del	Begrenzer	Anwendungsbeispiel

1	Zeilenvorschub	Memo
2	Next (0xfe)	Kontakte
3	Next und Zeilenvorschub	

FBSTRINGEXAMPLE - Beispiel für das Lesen von Zeichenketten aus dem Dateipuffer
FBBBYTE - Lesen und Schreiben im Dateipuffer (binär)

FBSTRINGEXAMPLE

Beispiel für das Lesen von Zeichenketten aus dem Dateipuffer
Das Beispiel zeigt die Verwendung von FBSTRING zum Lesen von Zeichenketten aus dem Dateipuffer:

```
! Lesen des ersten MEMO in Kategorie 2
fp=-1
FILENEXT 3,2,fp: ! fp auf den ersten Datensatz
FILEREAD 3,2,fp: ! Datensatz (=MEMO) lesen
FBINIT:          ! internen Zeiger auf Anfang des Puffers
$m
  s$=FBSTRING$(rc): ! String auslesen
  if rc>0 goto mend: ! Memoende -> fertig
  ? s$ :! String ausgeben
  goto m :          ! weiter
$mend
```

FFUNC

Prozeduren und Funktionen zur Arbeit mit reellen Werten

MIN#, MAX# - Extremwerte
RANDOM# - Zufallszahlen
ABS# - Der absolute Betrag
FRACT# - Der gebrochene Anteil
POW#, POWI# - Potenzen
SQR#, NRT# - Wurzeln
SIN#, ASIN# - Winkelfunktionen
SINH#, ASINH# - Hyperbelfunktionen
TRIGMODE, RAD#, DEG#, GRAD# - Winkeldarstellungen
POLR#, CARTX# - Koordinaten-Transformationen
LOG#, LN#, LOG2# - Logarithmen
EXP10#, EXP#, EXP2# - Exponentialfunktionen
FLOAT - Konvertierung in reelle Zahlen

FILE

Zugriff auf Dateien mittels Dateizeiger
Dateizeiger dienen zusammen mit Modus und Submodus zur Kennzeichnung eines Datensatzes einer Datei. Die Prozeduren mit Dateizeiger arbeiten sequentiell und sind beim Durchmustern großer

Datenmengen schneller als die mit einer Dateinummer arbeitenden Prozeduren (LOADFILE, SAVEFILE). Löschen und Verschieben von Datensätzen ist nur mittels Dateizeiger möglich.

FILENEXT, FILEPREV - Nächster/vorheriger Datensatz
FILEREAD, FILEWRITE - Datensatz lesen/schreiben
FILEXCHG - Datensatz verschieben
FILEREMOVE - Datensatz löschen

Diese Prozeduren arbeiten direkt mit den Daten im Dateipuffer.

FILEBUFFER

Interner Datei-Puffer

Es existiert in OWBasic nur ein Puffer für Dateien, so daß immer nur eine Datei geladen, bearbeitet und gespeichert werden kann. Alle Prozeduren die Daten aus dem Flash-RAM lesen oder schreiben, arbeiten mit diesem Speicher und zerstören damit den vorherigen Inhalt.
Der Zugriff auf diesen Speicher erfolgt über ein spezielles Array FILEBUFFER. Dies ist byteweise organisiert. Es kann beliebig gelesen werden als FILEBUFFER[i], aber nur in Zuweisungen beschrieben werden.
Beispiel:

```
c=FILEBUFFER[5]
PRINT FILEBUFFER[6]
FILEBUFFER[4]=125
```

aber nicht

```
INPUT FILEBUFFER[3] :! Syntaxfehler !
```

Ein vereinfachter sequentieller Zugriff auf den Dateipuffer ist mit den Prozeduren und Funktionen FBSTRING ... möglich.

FILEEXAMPLE

Beispiel für Zugriff auf Dateien mit Dateipointern

In diesem Beispiel werden die MEMOs eine Kategorie gelesen und die erste Zeile ausgegeben.

```
INPUT "Welche Kategorie: ",cat
fp=-1 :! Setze Filezeiger vor Anfang
$loop: FILENEXT 3,cat,fp :! Setze Zeiger auf nächsten
Satz
IF fp=-1 GOTO ready :! Kein weiterer Datensatz
FILEREAD 3,cat,fp : ! MEMO lesen nach FILEBUFFER
FOR i=0 TO 20 :     ! Erste 21 Zeichen
IF FILEBUFFER[i]=13 GOTO next
! Ende der Zeile?
PRINT CHR$(FILEBUFFER[i]);
! Zeichen ausgeben
NEXT
$next: PRINT:  ! Neue Zeile
GOTO loop
```

```
$ready: PRINT "Fertig!"
```

FILEFIND

Suchen eines Datensatzes nach dem Namen
FILEFIND%(mode%, submode%, name$)
Sucht nach der Datei mit dem Namen name im Modus mode und Submodus submode. Der Name ist gegeben durch die erste Zeile (Text) bzw. den ersten enthaltenen String (binär). Falls die Datei gefunden wurde, wird der zugehörige Dateizeiger zurückgeliefert; wurde sie nicht gefunden, gibt FILEFIND –1 zurück.

FILELABEL$

Titel einer Datei
FILELABEL$(mode%, submode%)
Gibt den Titel der durch mode und submode spezifizierten Datei zurück.

mode	Modus
1, 2	CONTACTS
3	MEMO

FILENEXT

Den nächsten Datensatz finden
FILENEXT mode%, submode%, FP%
Setzt den Dateizeiger FP auf den nächsten Dateisatz. Modus und Submodus wählen die Datei aus. Der Dateizeiger FP muß ein gültiger, von FILENEXT oder FILEPREV gelieferter Dateizeiger sein.
Um den ersten Datensatz zu finden, muß FP auf einen Wert von -1 gesetzt werden. Ein Wert von -1 für FP wird zurückgeliefert, wenn kein nächster Datensatz vorhanden ist.

FILEPREV

Den vorherigen Datensatz finden
FILEPREV mode%, submode%, FP%
Setzt den Dateizeiger FP auf den vorherigen Dateisatz. Modus und Submodus wählen die Datei aus. Der Dateizeiger FP muß ein gültiger, von FILENEXT oder FILEPREV gelieferter Dateizeiger sein.
Ein Wert von -1 für FP wird zurückgeliefert, wenn kein vorheriger Datensatz vorhanden ist.

FILEREAD

Lesen eines Datensatzes
FILEREAD mode%, submode%, fp% [, blocks%]

Der Datensatz, auf den fp zeigt, wird in den FILEBUFFER gelesen. Modus und Submodus wählen die Datei aus. Der Dateizeiger fp muß ein gültiger, von FILENEXT oder FILEPREV gelieferter Dateizeiger sein.
Wird blocks angegeben, so wird nur die gegebene Anzahl der Blocks (1 Block = 64 Bytes) in den FILEBUFFER eingelesen.

FILEREMOVE

Datensatz löschen
FILEREMOVE mode%, submode%, fp%
Der Datensatz, auf den fp zeigt, wird gelöscht. Modus und Submodus wählen die Datei aus. Der Dateizeiger fp muß ein gültiger, von FILENEXT oder FILEPREV gelieferter Dateizeiger sein.

FILEWRITE

Schreiben eines Datensatzes
FILEWRITE mode%, submode%, FP%
Der FILEBUFFER wird in den Datensatz, auf den FP zeigt, geschrieben. Modus und Submodus wählen die Datei aus. Der Dateizeiger FP muß ein gültiger, von FILENEXT oder FILEPREV gelieferter Dateizeiger sein.
Ist FP gleich -1 wird ein neuer Datensatz geschrieben. Nach dem Schreiben kann FP geändert sein, aber er zeigt immer auf den soeben geschriebenen Datensatz.
Nach dem Schreiben eines neuen Datensatzes werden alle vorher ermittelten Filezeiger ungültig.

FILEXCHG

Verschieben eines Datensatzes
FILEXCHG mode%, submode%, fp%, fpd%
Der Datensatz, auf den fp zeigt, wird auf die Position des Datensatzes verschoben, auf den fpd zeigt. Die vorher bei fpd gespeicherten Daten und alle folgenden Datensätze werden nach hinten verschoben.
Modus und Submodus wählen die Datei aus. Die Dateizeiger fp und fpd muß ein gültiger, von FILENEXT oder FILEPREV gelieferter Dateizeiger sein.

FINDSHARED

Suche nach Datensätzen des Shared File
FINDSHARED pattern, RECORDNAME$, FILEPOINTER [,application$]
Es wird im shared file nach einem Datensatz gesucht, dessen Name zu dem gegebenen Muster pattern paßt. Das Muster ist der Anfang des Namens oder leer für alle Datensätze.
Die Variable recordname$ gibt den gefundenen Namen zurück.
Die Variable filepointer muß für die erste Suche auf -1 gesetzt werden. Wenn ein Datensatz gefunden wurde gibt filepointer den zugehörigen Datensatzzeiger zurück, sonst den Wert -1. Für die zweite und weitere

Suche nach weiteren Datensätzen muß filepointer den unveränderten Wert beibehalten.
application ist ein Stringparameter, der den Namen der Anwendung angibt, nach deren Datensätzen zu suchen ist. Wenn dieser nicht angegeben wird, wird der Name des (BASIC-)Programmes selbst angenommen.
Warnung: Es ist nicht möglich die Suche fortzusetzen, indem der ermittelte Filepointer weiter verwendet wird, wenn zwischen den Aufrufen in den Flash-RAM geschrieben wurde, weil dadurch die Filepointer ungültig werden.

FLASH

Zugriff auf Dateien im FlashRAM

LOADFILE, SAVEFILE - Zugriff auf allgemeine Dateien
LOADMEMO, SAVEMEMO - Zugriff auf MEMOs
QUICKMEMO - Zugriff auf Quickmemos
FILE - Zugriff auf Dateien mit Dateizeigern
NAMEDFILE - Ermittle ADDIN-Datei Modus
SHARED - Zugriff auf Datensätze der gemeinsamen Datei
FILEBUFFER - Prozeduren zur Behandlung des Dateipuffers
FLASHSIZE - Gesamtgröße und freier Speicher
FLASHREMAKE - Flash-Speicherverwaltung

FLASHREMAKE

Flash-Speicherverwaltung
FLASHREMAKE
Führt den Speicherverwaltungs-Prozeß aus, der den Speicher reorganisiert und gelöschten Speicher zur Wiederverwendung freigibt.

FLASHSIZE

Gesamtgröße des Flash-Speichers und freier Speicher
FLASHSIZE TOTAL%, FREE%[, MMBLOCKS%]
Die Gesamtgröße und der zur Zeit freie Speicher im Flash werden auf den Variablen TOTAL und FREE abgelegt.
Wird MMBLOCKS angegeben, so werden darin die Anzahl der bei einer Speicherverwaltung freiwerdenden Datenblöcke abgelegt. Auf dem PV-S1600 erhält die Variable momentan den Wert –1.

FLOAT

Umwandlung in eine Gleitkommazahl
FLOAT#(i%)
FLOAT#(s$)
FLOAT#(b!)
FLOAT(i%)
FLOAT(s$)

FLOAT(b!)
Umwandlung des Arguments in einen Gleitkommawert (Float).

FOR

Programm-Schleife
FOR Laufvariable% = Startwert% TO Endwert% [STEP step%]
Anweisungen : NEXT
Die Anweisungen werden in einer Schleife wiederholt durchlaufen. Dabei wird die Laufvariable beginnend beim Startwert jeweils um die Schrittweite step vergrößert, solange der Endwert nicht überschritten wird.
Laufvariable, Startwert und Endwert müssen vom Typ Integer sein; die Schrittweite muß eine positive Integer-Konstante sein. Ohne die Angabe von STEP ist die Schrittweite 1.

Beispiel:
```
summe=0
FOR i=0 to 9
  summe=summe+i
NEXT
PRINT "Die Summe der Zahlen von 0 bis 9 ist ",summe
```

FORMAT

Quelltext-Gestaltung
Quelltext kann frei formatiert werden. Wird mehr als eine Anweisung auf eine Zeile geschrieben, so sind die einzelnen Anweisungen durch ein Semikolon (':') zu trennen. Kommentare werden durch das Schlüsselwort REM oder ein Ausrufezeichen '!' eingeleitet und gehen bis zum Zeilenende. Es sind auch Kommentare möglich, die mitten in einer Zeile anfangen und sich über mehrere Zeilen erstrecken können; sie werden eingeleitet mit "/*" und beendet mit "*/". Eine Verschachtelung dieser Kommentare ist nicht erlaubt.
Im Programm-Text spielt die Groß- und Kleinschreibung keine Rolle.

FORMBUTTON

Button in Formularen
Ein Button in einem Formular gibt einfach den Wert des Selektors zurück, wenn er gedrückt wird. Die Typ-Nummer des Button ist 0. Der spezielle Wert von Null für den Selektor erzeugt eine inaktive Textbox, die nicht auf Druck reagiert.

Beispiel:
```
forminit
formentry "One",  5,30,65,44, 0, 10
formentry "Two",  5,50,65,64, 0, 20
formentry "Three",5,70,65,84, 0, 30

formdisplay
$m1
```

```
formtouch sel
messagebox "Selected:"+string(sel)
goto m1
```

FORMCHECKBOX

CheckBoxen in Formularen

Eine CheckBox in einem Formular erlaubt das Ein- und Ausschalten einer Option. Wird eine Checkbox geändert, so wird der Hauptwert des Selektors der Checkbox zurückgegeben, kombiniert mit einem Unterwert von 0 für eine ausgeschaltete Option und einer 1 für eine eingeschaltete Option.

Beispiel:

```
forminit

! several checkboxes
formentry "italic", 5,30,95,44, 1, 10
formentry "bold", 5,50,95,64, 1, 20
formentry "underlined", 5,70,95,84, 2, 30

formdisplay
$m1
formtouch sel
messagebox "Selected:"+string(sel)
goto m1
```

FORMDISPLAY

Darstellung eines Formulars

FORMDISPLAY

Das definierte Formular wird dargestellt. Ein expliziter Aufruf von SHOW ist nicht notwendig.

FORMENTRY

Definieren eines Steuerelements eines Formulars

FORMENTRY text$, x1%,y1%,x2%,y2%, type%, selector%

Ein neues Element eines Formulars (Button, CheckBox, RadioButton) wird festgelegt. Die Parameters x1,y1,x2,y2 legen das Feld auf dem Touchscreen fest, wo das Element dargestellt wird.
Die Zeichenkette text wird zentriert als Name des Elements dargestellt. Wenn das erste Zeichen von text ein '#' ist, wird eine nachfolgende Zahl als Nummer einer internen Graphik(IGRAPH) interpretiert, die als Darstellung des Elements verwendet wird.

Element	type	Bedeutung
Button	0	Auslösen einer Aktion
CheckBox	1	(De-)Aktivieren einer Option
RadioButton	2	Auswahl einer von mehreren Optionen

Der Selektor wird benutzt, um die Auswahl des Elements bei FORMTOUCH zu signalisieren.

FORMEXAMPLE

Beispiel-Formular

```
! formtest
cls 2
forminit
formentry "TestForm", 90, 2, 156, 24, 0, 0: ! "Title"
formsetstatus 0,4: ! doppelter Rahmen
! einige CheckButtons
formentry "One", 5,30, 65,44,1,10
formentry "Two", 5,50, 65,64,1,20
formentry "Three", 5,70, 65,84,1,30
! Gruppe von RadioButtons
formentry "",99,39, 156,85, 0,1: ! Rahmen um die
RadioButtons
formentry "alpha", 100,40,155,54,2,40
formsetstatus 40,3: ! Ausgewählt, kein Rahmen
formentry "beta", 100,55,155,69,2,41
formsetstatus 41,2: ! kein Rahmen
formentry "gamma", 100,70,155,84,2,42
formsetstatus 42,2: ! kein Rahmen
! Button mit interner Grafik
formentry "#22", 7,132, 25,149,0,50
formsetstatus 50,2: ! kein Rahmen
formentry "#24", 35,130, 63,159, 0,60
formentry "#25", 65,130, 93,159, 0,70
formentry "#257", 140,110,152,123,0,80
formentry "#184", 140,130,152,143,0,90

$m2
formdisplay : ! Formular anzeigen
formtouch sel : ! auf Eingabe warten
if sel=21 then : ! CheckBox 20 selektiert
 formsetstatus 60,8 : ! Button 60 inaktivieren
endif
if sel=20 then : ! CheckBox 20 deselektiert
 formresetstatus 60,8 : ! Button 60 aktivieren
endif
messagebox string(sel),3: ! anzeigen
goto m2
```

FORMINIT

Initialisieren eines Formulars

FORMINIT [defstatus%=0] [, deffont%=-1]

Ein neues Formular wird initialisiert. Da es nur einen Datenbereich für Formulare gibt, zerstört diese Prozedur vorher definierte Formulare. deffont bezeichnet den Standard-Schriftfont und defstatus den per Default gesetzten Status der Formularelemente.

FORMRADIOBUTTON

RadioButton in Formularen

Ein RadioButton in einem Formular erlaubt die Auswahl einer von mehreren Optionen. Dazu werden RadioButtons gruppiert. Alle RadioButtons einer Gruppe haben den selben Hauptwert für den Selektors , aber unterschiedliche Unterwerte. Die Typ-Nummer des RadioButton ist 2.

Beispiel:

```
forminit
! one group of radiobuttons
formentry "One",  5,30,65,44, 2, 10
formentry "Two",  5,50,65,64, 2, 11
formentry "Three",5,70,65,84, 2, 12

! a second group of radiobuttons
formentry "Alpha",  95,30,145,44, 2, 20
formentry "Beta",  95,50,145,64, 2, 21
formentry "Gamma",95,70,145,84, 2, 22

formdisplay
$m1
formtouch sel
messagebox "Selected:"+string(sel)
goto m1
```

FORMSELECTOR

Selektoren für Elemente von Formularen

Jedes Steuerelement eines Formulars wird über einen Selektor angesprochen. Ein Selektor ist eine ganze Zahl, die aus einem Hauptwert - einem Vielfachen von Zehn - und einem Unterwert - dem Rest - besteht.

Beispiel:

Selektor	Hauptwert	Unterwert
152	15	2
30	3	0

Jedes selbständige Element eines Formulars hat einen anderen Hauptwert. Der Unterwert dient dazu, den Status eines Elements zurückzumelden:

Element	Bedeutung des Unterwertes
CheckButtons	1 - ausgewählt
CheckButtons	0 - nicht ausgewählt
RadioButton	- Nummer der gewählten Option

FORMSETSTATUS

Status eines Formularelements ändern

FORMSETSTATUS sel%,bits%

FORMRESETSTATUS sel%,bits%

Der Zustand eines Steuerelement eines Formulars kann geändert werden, indem Bits des Statuswortes gesetzt oder rückgesetzt werden.
Aufgerufen mit dem Selektor und den Bitwerten setzt die Prozedur FORMSETSTATUS die entsprechenden Bits, FORMRESETSTATUS setzt sie zurück. Die Werte mehrerer Bits können auch kombiniert werden.

Bitwert	Bedeutung
1	Element ausgewählt
2	Kein Rahmen um das Element
4	Doppelter Rahmen
8	Element inaktiv (grau, nicht wählbar)

Vorsicht: Die Konsistenz der Steuerelemente darf nicht gestört werden. So sollten keine zwei RadioButton gleichzeitig "ausgewählt" werden.

FORMTOUCH

Formular-Eingaben

FORMTOUCH SEL%[, nowait!=FALSE]

Das definierte (FORMINIT, FORMENTRY) und dargestellte (FORMDISPLAY) Formular wird aktiviert. Die Prozedur wartet auf eine zulässige Auswahl (Touch), passt den internen Status an und setzt SEL auf den Wert des zugehörigen Selektors.
Wenn der optionale boolesche Parameter nowait mit TRUE angegeben wird, kehrt FORMTOUCH sofort zurück und meldet mit SEL=-1, daß keine Auswahl erfolgte.

FPOINTER

Funktionszeiger

In OWBasic können globale Variablen deklariert werden, die auf (nicht eingebaute) Funktionen oder Prozeduren zeigen können.

Bevor er verwendet werden kann, muß der Typ des Funktionszeigers zunächst deklariert werden:

FPTR <typename>=(PROC <paramlist>)
FPTR <typename>=(FUNC<return type> <paramlist>)

<paramlist> gibt die Parameter der Prozedur/Funktion an und hat folgenden Aufbau:

[VAR] <dummy variable> [[]], ...

Ein Funktionszeiger ist eine gewöhnliche Variable; dementsprechend muß er nicht deklariert (was aber mit LOCAL durchaus möglich ist), sondern kann einfach verwendet werden:

<varname>\<typename>=#NULL

Einem Funktionszeiger kann man einen Wert auf die gleiche Weise wie bei anderen Variablen zuweisen; die Adresse einer Prozedur läßt sich über

PROC(<procedure name>)

, die einer Funktion über

FUNC(<function name>)
abfragen.

Man kann einem Funktionszeiger auch den symbolischen Wert #NULL zuweisen und ihn in Integer konvertieren, um ihn darauf abzufragen. Dies ist sinnvoll bei optionalen Callback-Funktionen, die nur aufgerufen werden sollen, wenn sie auf etwas zeigen.

Aufgerufen wird ein Funktionszeiger durch die Direktive CALL:
CALL <procptrname> <parameters>
CALL <type suffix> (<funcptrname> <parameters>)

Beispiel:

```
FPTR myproc_t=(PROC)
FPTR myfunc_t=(FUNC$ i%)

DIM screeninit\myproc_t[0]
DIM bname\myfunc_t[0]

PROC drawscreen
IF INT(screeninit)<>0 THEN
 CALL screeninit
ENDIF
TEXTBOX CALL$(bname RND(2)),10,30,149,40,0
ENDP

FUNC name$ n
RETURN "Nr. "+STRING(n)

PROC init
CLS 2
BOX 0,0,159,159
ENDP

screeninit=PROC(init)
bname=FUNC(name$)

drawscreen
```

FRACT#

Gebrochener Anteil einer Gleitkommazahl
FRACT#(f#)
Der gebrochene Anteil der Zahl f#.
Es wird immer zur nächstkleineren Zahl gerechnet:

```
FRACT#(3.3)    0.3
FRACT#(-3.3)   0.7
```

FUNC

In OWBasic ist es möglich, Anwender-Funktionen zu definieren. Anwender-Funktionen und Prozeduren müssen am Anfang des Programmes definiert werden. Das Hauptprogramm folgt darauf.
Eine Funktionsdefinition beginnt mit dem Funktions-Kopf und endet mit der RETURN-Anweisung.

```
FUNC test A, B#, VAR C
 <statements>
RETURN <value>
```

Der Funktions-Kopf beginnt mit dem Schlüsselwort FUNC gefolgt vom Namen der Funktion und der Parameterliste. Der Typ der Funktion wird wie bei Variablen durch ein Suffix bestimmt, wenn der Typ vom Standardtyp abweicht. Der Typ kann auch ein Zeiger (dann hat das Schlüsselwort FUNC das Pointer-Präfix &), aber kein selbstdefinierter sein. Die Parameterliste beschreibt den Typ der Parameter der Funktion und gibt ihnen einen formalen Namen. Variablen, die innerhalb der Funktion eingeführt werden, sind lokal, das heißt, sie sind außerhalb der Funktion unbekannt.
Für die Parameter der Funktion können auch Default-Werte angegeben werden; wenn die Funktion dann ohne diesen Parameter aufgerufen wird, wird automatisch vom Compiler der Default-Wert eingesetzt. Als Default-Werte sind für Variablenparameter auch Zeiger erlaubt.

Beispiel:

```
PROC clearscreen mode%=1
 CLS mode
ENDP
PROC my_messagebox text$, caption$="", time#=0.0
 msgbox text, caption, time
ENDP
PROC memory_size var size%=#NULL
 IF &size THEN
  FLASHSIZE size, dummy
 ENDIF
ENDP
```

Wird für einen Parameter ein Default-Wert angegeben, so darf nach diesem kein Parameter ohne Default-Wert folgen.

Funktionen berechnen einen Wert und geben ihn an das aufrufende Programm zurück. Der Rückgabewert wird bei der RETURN-Anweisung festgelgt, welche die Funtkionsdefinition abschließt.
Die Verwendung von Anwender-Funktionen ist äquivalent der Anwendung der eingebauten Funktionen. Die Anwenderfunktion wird auch dann verwendet, wenn eine gleichnamige eingebaute Funktion existiert.
Falls es für den Rückgabewert einer Funktion keine Verwendung mehr gibt, kann die Funktion auch als Prozedur aufgerufen werden

Beispiel:

```
FUNC vlen x,y: ! length of 2d vector
PRINT "function vlen called"
RETURN sqr(x*x+y*y)

x=5: y=7: ! These variables do not have to do anything
          ! with the function parameters.
```

```
print vlen(x-5,y-2): ! Example-Call
vlen(1,2): ! call as procedure
```

GETFP

Dateizeiger zu einer Dateinummer ermitteln

GETFP%(mode%, submode%, nr%)

Ermittelt den Dateizeiger auf die Datei mit Modus mode, Untermodus submode und Nummer nr .

GETICON

Bildschirminhalt als Icon ablegen

GETICON A%[], x1%, y1%, x2%, y2%

Speichert den aktuellen Inhalt des Bilschirms im Rechteck x1,y1, x2,y2 als Icon im Integerfeld A.

GETSYSVAR

OWBasic-Optionen abfragen/verändern

GETSYSVAR nr%, VAL%
SETSYSVAR nr%, val%

Fragt eine OWBasic-Option ab bzw. ändert diese. nr spezifiziert die Option; VAL enthält deren Wert.

nr	Option	Wertebereich
1	Programmkategorie	0 (Alle), 1..5 (Kategorie 1..5), 15 (Alle – geheimer Bereich), 16..20 (Kategorie 1..5 - geheimer Bereich)
2	Include-Kategorie	0 (Alle), 1..5 (Kategorie 1..5), 15 (Alle – geheimer Bereich), 16..20 (Kategorie 1..5 - geheimer Bereich)
3	Statistiken anzeigen	0, 1
4	Startmenü	1 (Dateimenü), 3 (Programmenü)
5	Warten nach Ende	0, 1
6	Bei Abbruch fragen	0, 1
7	Overlays löschen	0, 1
8	Autorun	0, 1
9	Neu starten	0, 1
10	Caching	0, 1
11	Frage nach Fehlersuche	0, 1
12	Caching bei Aufruf durch externe Applikation	0, 1

Wird SETSYSVAR mit nr<0 aufgerufen, so werden die vorher geänderten Optionen abgespeichert.

GLOBAL

Globale Variablen und Konstanten

Durch globale Variablen können Systemwerte abgeftragt und das Verhalten mancher Prozeduren beeinflußt werden. Globale Konstanten ermöglichen die Abfrage systemspezifischer Werte, wie z.B. das verwendete PV-Modell. Sie können in Direktiven verwendet werden, was eine systemspezifisch unterschiedliche Lösungen einer Aufgabe ermöglicht.

Konstante	Bedeutung
#FLOATSIZE	Größe einer Float-Variablen in Integer-Größen (PV-S1600: 2, andere PV-Modelle: 4)
#INTSIZE	Größe einer Integer-Variablen in Byte (PV-S1600: 4, andere PV-Modelle: 2)
#JUMPBUFSIZE	Größe des Arrays, das für &JUMP/&SETJUMP benötigt wird (PV-S1600/andere PV-Modelle: 2)
#NULL	Symbolische Konstante mit dem Wert 0. Sollte bei Zeigeroperationen zwecks Übersichtlichkeit verwendet werden
#OWBVERSION	Aktuelle OWBasic-Version
#PVMODEL	Aktuelles PV-Modell

Variable	Bedeutung
TCHX	X-Koordinate der zuletzt berührten Bildschirmposition
TCHY	Y-Koordinate der zuletzt berührten Bildschirmposition
ERROR	Enthält die Fehlernummer des zuletzt aufgetretenen Fehlers. Wenn kein Fehler auftrat, ist ERROR 0.
INPUTRC	Gibt an, über welche Schaltfläche der INPUT-Dialog verlassen wurde (->INPUT)
PARAMETER$	Enthält den Parameter, mit welchem das Programm aufgerufen wurde
FORMSTRINGS$	Enthält die Namen der Schaltflächen in einem Formular. Der Hauptwert des Selektors entspricht dem zu verwendenden Index
ERRORHANDLER	Benutzerdefinierte Fehlerbehandlung
EXITHANDLER	Benutzerdefinierte Abbruchbehandlung
OWBVERSION	Aktuelle OWBasic-Version (veraltet; #OWBVERSION sollte stattdessen verwendet werden)
PVMODEL	Aktuelles PV-Modell (veraltet; #PVMODEL sollte stattdessen verwendet werden)

GOSUB

Unterprogramm-Aufruf

GOSUB marke

Die Programmabarbeitung wird an der gegebenen Marke fortgesetzt und beim nächsten RETURN wird zu der nach GOSUB folgenden Anweisung zurückgekehrt.

GOTO

Sprung zu einer Marke
GOTO marke
Die Programmabarbeitung wird an der angegebenen Marke fortgesetzt.

GRAD#

Umrechnung von Winkeln in Neugrad
GRAD#(phi#)
Wenn phi ein Winkel im aktuellen Modus ist, liefert die Funktion GRAD den entsprechenden Winkel in Neugrad zurück.

GRAPHIC

Grafikprozeduren
Die Darstellung gezeichneter Grafiken erfolgt nicht sofort, sondern muß durch SHOW veranlaßt werden. Eine sofortige Darstellung nach jeder einzelnen grafischen Operation würde die Abarbeitung des Programms stark verlangsamen.

Die Grafik-Prozeduren sollten normalerweise nicht mit den Text-orientierten Prozeduren vermischt werden.

GRAPHIO

Ein- und Ausgaben auf dem Grafik-Bildschirm
Zur Ein- und Ausgabe von Werten im Grafik-Modus dienen die folgenden Prozeduren und Funktionen:

DRAWTEXT - Text auf den Bildschirm zeichnen
TEXTBOX - Eine Text-Box
MESSAGEBOX - Eine Box mit einer Mitteilung
ASK! - Eine Ja-/Nein-Frage
CALCINPUT# - Zahlen-Eingabe auf einem Taschenrechner

ICON

Bitmaps in Integer-Felder speichern
Mit DRAWICON, SETICONPIX und ICONPIX kann man Bilder als Bitmap bearbeiten und darstellen. Auch wenn der Begriff ICON kleine Bilder nahelegt, lassen sich auch Bilder in Bildschirmgröße damit bearbeiten. Bilder werden in Integer-Feldern abgelegt. Die ersten zwei Elemente enthalten die Größe in X- und Y-Richtung. Die Daten danach sind byteweise organisiert. Von links nach rechts werden die Pixel bitweise, beginnend mit dem höchsten Bit, abgelegt. Jede neue Zeile beginnt mit einem neuen Byte. Die Feldgröße, die zur Speicherung eines Bildes erforderlich ist, errechnet sich folgendermaßen:

- Berechne die Zahl der Bytes pro Zeile: Bei 50 Pixeln in der Zeile werden 50 Bit oder 6 Byte und 2 Bit benötigt. Die verbleibenden 6 Bit des 7. Bytes können nicht genutzt werden, so daß 7 Byte pro Zeile benötigt werden.
- Berechne die Gesamtzahl von Bytes: Bei 45 Zeilen wären es 45 mal 7 Bytes oder 315 Bytes.
- Ist die Bytezahl ungerade, so muß ein Byte addiert werden, um eine gerade Zahl zu erhalten, da Integerzahlen jeweils zwei Byte enthalten. 316 Byte sind 158 Integerzahlen.
- Zwei Elemente werden für die Speicherung der Größe benötigt, so daß das Feld 160 Elemente haben muß.

Beispiel für das Anlegen einer Bitmap:

```
DIM bitmap[159]: ! Felddefinition 160 Elemente
clear bitmap
bitmap[0]=50:    ! Größe x
bitmap[1]=45;    ! Größe y
```

Jetzt kann der Inhalt der Bitmap durch direktes Belegen des Feldes oder mit SETICONPIX festgelegt werden.

ICONHASH

Berechnen eines Hashwertes eines Icons

ICONHASH BM%[], CRC1%, CRC2%

Für das ICON in BM wird ein Hashwert nach CRC32 berechnet und in crc1 und crc2 zurückgegeben. Dieser kann dazu verwendet werden, dieses Icon aus einem Quickmemo mittels LOADQMHASH zu laden.

ICONPIX

Wert des Pixel im Icon

ICONPIX%(A%[], x%, y%)

Die Funktion liefert den Wert des Pixels mit den Koordinaten x und y im Icon A.
Die Größe des Icons muß unbedingt vorher festgelegt werden!

IF

Bedingte Ausführung von Anweisungen

IF <Bedingung> THEN <Anweisungen>: ENDIF
IF <Bedingung> THEN <Anweisungen1>: ELSE <Anweisungen2>: ENDIF
IF <Bedingung> GOTO <Marke>

Bedingte Ausführung von Anweisungen. Ist die Bedingung erfüllt, so werden die Anweisungen zwischen THEN und ENDIF bzw. THEN und ELSE ausgeführt. In der zweiten Form werden die Anweisungen zwischen ELSE und ENDIF ausgeführt, wenn die Bedingung nicht erfüllt ist.
Eine Bedingung ist irgend ein boolescher (logischer) Ausdruck.
Beispiel:

```
IF a>b THEN
   ? "A ist größer B"
ELSE
   ? "A ist kleiner gleich B"
ENDIF

IF a % 2=0 THEN PRINT "A ist gerade": ENDIF
```

In der Form IF <anweisung> GOTO <marke> wird bei erfüllter Bedingung zur angegebenen Marke gesprungen. Dies ist eine Kurzform für die Anweisungen:
IF <bedingung> THEN GOTO <marke>: ENDIF

IFUNC

Prozeduren und Funktionen zur Arbeit mit Integerwerten

MULDIV - Kombinierte Multiplikation und Division
SIN, COS - Ganzzahlige Winkelfunktionen
RANDOM, RANDOMIZE - Zufallszahlen
INT - Konvertierung in ganze Zahlen
INC, DEC - Zählen, Inkrementieren/Dekrementieren

IGRAPH

Interne Grafik darstellen
IGRAPH nr%, x%, y%
Darstellung der internen Grafik mit der Nummer nr an der Stelle (x,y) des Bildschirms.

IN

Port-Eingabe
IN%(port%)
Einlesen des Ports port (Byte).

INC

Inkrementieren/Dekrementieren
INC A% [,i%=1]
DEC A% [,i%=1]
Die Integer-Variable A wird um den Betrag von i erhöht/erniedrigt.

INCF

Inkrementieren/Dekrementieren
INCF A#[, i#=1.0]
DECF A#[, i#=1.0]

Die Float-Variable A wird um den Betrag von i erhöht/erniedrigt.

INCLUDE

Verwendung von mehr als einem MEMO

APPEND "<name>"
INLCUDE "<name>"

Reicht die Größe eines MEMOs nicht aus, so können weitere MEMOs angehängt werden. Durch Angabe von APPEND zwei wird zum Beispiel das MEMO mit dem Namen zwei angehängt. Dies bedeutet, daß die Übersetzung des ersten MEMOs beendet wird und danach der Quelltext aus einem MEMO angehängt wird, das mit "!+" beginnt, und dessen Rest der ersten Zeile dem Namen "zwei" entspricht.
Groß- und Kleinschreibung wird hier nicht unterschieden. Wenn der Name des Memos Leerzeichen oder andere nicht alphanumerische Zeichen enthält, muß der Name in der APPEND-Anweisung in Anführungsstriche gesetzt werden:

APPEND "Memo 2"

APPEND-Memos müssen in der Kategorie des Programmes stehen.
Mehr auf die Wiederverwendung von Quelltext (quasi als Bibliothek) zielt die Möglichkeit der Verwendung von INCLUDE. Hiermit wird ein anderer Quelltext mit Prozeduren oder Subroutinen an der Stelle des INCLUDES eingefügt. INCLUDE-Memos beginnen mit "!*". Das darauffolgende Wort wird wie bei APPEND als Name interpretiert.
INCLUDE-Memos werden zunächst in der Programm-Kategorie, dann in dem mittels der Options als INCLUDE-Kategorie festgelegten Kategorie gesucht. Dies erlaubt die Anlage von wiederverwendbaren INCLUDEs mit Prozeduren in einer separaten Kategorie. Jedes Memo wird immer nur einmal eingefügt, auch wenn die INCLUDE-Anweisung mehrfach im Quelltext erscheint. Dies verhindert Compiler-Fehler durch mehrfach definierte Prozeduren, wenn unterschiedliche
INCLUDE-Files ihrerseits jeweils das gleiche Memo mit INCLUDE einschließen.

INFLOCATE

Abfrage der Kursorposition

INFLOCATE X%, Y%

Die Variablen X und Y werden mit der aktuellen Kursorposition belegt.

INPUT

Eingabe von Werten

INPUT [prompt$], variable

Veranlaßt die Eingabe eines Wertes durch den Benutzer. Wird die Stringkonstante prompt angegeben, so wird diese als Eingabe-Anforderung ausgegeben. Die Eingabe wird mit der NEXT-Taste, der ENTER-Taste oder mit ESC beendet. Der Grund für den Abbruch kann nach INPUT über die Variable INPUTRC abgefragt werden.

```
inputrc  ReturnCode
0        ENTER
1        NEXT
2        ESC
```

Im normalen Modus führt Drücken von ESC zum Programmabbruch. Um das zu vermeiden und die ESC-Taste im Programm auszuwerten müssen die Fehlermeldungen ausgeschaltet werden (ERROROFF/ERRORON).

Beispiel:

```
...
SYS 7
INPUT "Ihre Eingabe bitte :", s$
? "Ihre Eingabe war: ",s$," ReturnCode:",inputrc
SYS 6
...
```

INPUTI

Bearbeiten eines Variablenwertes

INPUTI VAL%
INPUTF VAL#
INPUTS VAL$ [,prompt$]
INPUTTEXT VAL$, x1%, y1%, x2%, lines%, maxchar%

Diese Befehle stellen eine Möglichkeit dar, eine Variable vom Benutzer komfortabel editieren zu lassen. Für Zahlentypen wird dazu ein numerisches Eingabefeld, für Strings eine Tastatur angezeigt, über die der Benutzer den Wert ändern kann.
Über den optionalen Parameter prompt kann eine Überschrift (etwa: "Name: ") bei der Stringeingabe angegeben werden.
Die Prozedur INPUTTEXT funktioniert ähnlich wie INPUTS, nur kann das Eingabefeld über die Koordinaten x1, y1 und x2 selbst definiert werden. lines gibt die Zeilenanzahl, die zur Verfügung steht, maxchar die maximal zulässige Zeichenzahl an.

INRANGE!

Test, ob ein Wert in einem vorgegebenen Bereich liegt

INRANGE!(v%,v1%,v2%)

Testet, ob der Wert v innerhalb des Bereichs von v1 bis v2 liegt. Gibt TRUE zurück, wenn der Wert im Bereich liegt, sonst FALSE. Die Funktion ist äquivalent zum logischen Ausdruck (v>=v1) AND (v<=v2).

INSIDE!

Test ob Koordinatenwerte in einem Fenster liegen

INSIDE!(x%, y%, x1%, y1%, x2%, y2%)

Testet, ob die Koordinaten x und y in dem durch die linke obere Ecke (x1,y1) und die rechte untere Ecke (x2,y2) definierten Fenster liegen. Gibt TRUE zurück, wenn die Koordinaten im Fenster liegen, sonst FALSE. Die

Funktion ist äquivalent zum logischen Ausdruck (x>=x1) AND (x<=x2) AND (y>=y1) AND (y<=y2) .

INSIDEG!

Test von Koordinatenwerten
INSIDEG!(x%, y%)
Testet, ob x und y gültige Koordinatenwerte sind und nicht die zulässigen Werte überschreiten. Gibt TRUE zurück, wenn die Koordinaten gültig sind, sonst FALSE.

INT

Umwandlung in eine Integer-Zahl
INT(f#)
INT(s$)
INT(b!)
INT(c&)
INT(<function pointer>)
INT(<enum variable>)
INT(&p)
Umwandlung des Arguments in Integer.
Zeiger jedes beliebigen Typs können ebenfalls in Integer umgewandelt werden.

INTSIZE

Größe eines Integers in Byte
#INTSIZE
Die globale Konstante INTSIZE gibt die Größe einer Integer-Variable in Byte an (alte PVs: 2, PV-S1600: 4).

JUMP

Sprung über Prozeduren hinweg
JUMP ADDR%[], statval%
SETJUMP%(ADDR%[])
Die Funktion SETJUMP speichert Daten zur aktuellen Codeposition in das Array ADDR, dessen Mindestgröße durch #JUMPBUFSIZE angegeben wird. Mit JUMP kann später die Position von überall aus angesprungen werden. Wenn SETJUMP einen Wert ungleich 0 zurückgibt, wurde es mittels JUMP angesprungen; der zurückgegebene Wert entspricht statval. Wird als statval 0 angegeben, so ist der Rückgabewert 1.
Die Funktionalität und die Beschränkungen von JUMP und SETJUMP lassen sich mit denen von setjmp () und longjmp () in ANSI C vergleichen.
ACHTUNG: Eine innerhalb einer Funktion gespeicherte Codeposition darf nicht von außerhalb der Funktion aufgerufen werden!

KBDRAW

Tastatur darstellen
KBDRAW
Die Prozedur stellt eine Tastatur auf dem Bildschirm dar. Diese Prozedur muß vor der Verwendung von KBWAIT! aufgerufen werden.

KBWAIT!

Auf Tasteneingabe warten
KBWAIT!(KEYCHAR&)
Mittels KBWAIT! kann eine zuvor mit KBDRAW dargestellte Tastatur abgefragt werden. Gibt die Funktion TRUE zurück, so wurde eine Taste gedrückt, andernfalls wurde der Bildschirm außerhalb des Tastaturbereichs berührt. Die gedrückte Taste wird in KEYCHAR abgelegt.
Für KEYCHAR existieren einige spezielle Werte, die besondere Tasten repräsentieren:

KEYCHAR	Taste
CHR&(0xF0)	Enter
CHR&(0xF7)	NEXT

LABEL

Marken
Im Programm können Marken gesetzt werden. Eine Marke ist entweder eine ganze Zahl, die am Anfang einer Anweisung steht, (entsprechend "Zeilennummern" in alten Basic-Interpretern) oder ein mit '$' beginnender String. Marken werden als Ziel von GOTO- bzw. GOSUB-Anweisungen benötigt.

GOTO - Sprung zu einer Marke
GOSUB - Unterprogramm-Aufruf

LEFT$

Linker Teilstring
LEFT$(s$,i%)
Ein String der Länge i, linksbündig mit s$ gefüllt:

```
LEFT$("Test",2)"Te"
LEFT$("Test",6)"Test  "
```

RIGHT$ - Rechter Teilstring
MID$ - Teilstring

LEN

Stringlänge
LEN%(String$)

Länge des gegebenen Strings (Zahl der Zeichen).

LEVERPUSH

Status des Action-Reglers

LEVERPUSH%()

Die Funktion gibt den aktuellen Status des Action-Reglers zurück:

Wert	Status
0	Inaktiv
1	Oben
2	Gedrückt
3	Unten
6	Links
7	Rechts

LIGHT

Hintergrund-Beleuchtung schalten und abfragen

LIGHT n%

LIGHT!()

LIGHT!() gibt den Status der Hintergrund-Beleuchtung zurück (FALSE – aus, TRUE – ein). LIGHT schaltet die Hintergrund-Beleuchtung

n	Funktion
0	Aus
1	Ein für 15s
2	Ein

LINE

Linie zeichnen

LINE x1%, y1%, x2%, y2% [, z%=1]

Zeichnet eine Linie vom Punkt (x1,y1) nach dem Punkt (x2,y2) mit dem gegebenen Wert z.

Wert	Funktion
0	Linie löschen (Weiß)
1	Linie setzen (Schwarz)
2	Linie löschen (Weiß)
4	Punktierte Linie
5	Linie invertieren

LOADFILE

Datei laden

LOADFILE mode%, submode%, nr%

Laden einer Datei aus dem FlashRAM. Die Datei ist gegeben durch den Modus, den Untermodus (Kategorie) und die laufende Nummer. Die Art der Datei (Text oder binär) wird durch den Modus bestimmt. Das Laden

erfolgt in einen internen Puffer (FILEBUFFER), aus dem die Daten ausgelesen werden müssen.

LOADMEMO

Memo lesen
LOADMEMO ARRAY$[], submode%, nr%[, LNR%]
LOADMEMOFP ARRAY$[], submode%, fp%[, LNR%]
Ein Memo des Bereichs submode wird in das Stringarray ARRAY geladen. Das Memo wird selektiert über seine Nummer nr bzw. einen gültigen Dateizeiger fp . Die Zahl der wirklich im Memo vorhandenen Zeilen wird auf LNR zurückgeliefert.

LOADMEMONAME

Laden eines Memos nach seinem Namen
LOADMEMONAME ARRAY$[], submode%, name$, sig$[, LNR%][, FP%]
Liest ein Memo der Kategorie submode ein und speichert es im Stringarray array$. Das Memo wird über seinen Namen ausgewählt (erste Zeile). Der Name besteht aus einer Signatur sig$ und einem Namen. Um keine Signatur zu nennen, ist sig als Leerstring Anzugeben, führende oder nachfolgende Leerzeichen werden ignoriert. Die Zahl der wirklich im Memo vorhandenen Zeilen wird auf LNR zurückgeliefert. Wenn FP angegeben ist, wird auf der Integer-Variablen der Filepointer des Memos zurückgegeben, so daß zum Beispiel nach Änderungen das Memo zurückgeschrieben werden kann.

LOADQM

Einlesen eines Quickmemos
LOADQM BM%[], cat%, nr%
LOADQMFP BM%[], cat%, fp%
Ein Quickmemo aus der Kategorie cat wird eingelesen und in dem Integerfeld BM abgelegt. In BM liegt die Grafik als ICON vor.
Die Auswahl eines Quickmemos erfolgt über die Nummer nr oder den Dateizeiger fp.

LOADQMHASH

Lesen eines Quickmemos selektiert über einen Hashwert
LOADQMHASH BM%[], cat%, crc1%, crc2% [,FP%]

Ein Quickmemo aus der Kategorie cat wird eingelesen und in dem Integerfeld BM abgelegt. In BM liegt die Grafik als ICON vor. Die Auswahl des Quickmemos erfolgt über eine Hashwert, repräsentiert durch die Integerwerte crc1 und crc2. Die Berechnung erfolgt mittels CRC32. Auch wenn der Hashwert nicht eindeutig ein Quickmemo beschreibt, ist die Wahrscheinlichkeit gering, daß zwei vorhandene Quickmemos den gleichen Hashwert haben. In diesem Fall muß ein Quickmemo leicht verändert werden.

Zur Berechnung des Hashwertes dient ICONHASH.

LOADSHARED

Zugriff auf die gemeinsame Datei

LOADSHARED recordname$[, application$]
SAVESHARED recordname$[, application$]

Die Prozedur SAVESHARED erwartet einen gefüllten binären Dateipuffer und schreibt dessen Inhalt als Datensatz mit der gegebenen Signatur in die gemeinsame Datei. LOADSHARED lädt den Datensatz mit der gegebenen Signatur in den Dateipuffer.

Beispiel:

```
! MyApp
  LOADSHARED "settings": ! Einstellungen laden
  FBBINIT
  prompt$=FBBSTRING$():       ! Ersten Wert lesen
..
  FBBINIT:       ! Geänderte Einstellungen speichern
  FBBPUTSTRING prompt$
  SAVESHARED "settings": ! In die Datei schreiben
```

Achtung: Die übergebenen Daten dürfen den binären Dateipuffer (3072 Byte) nicht vollständig füllen, da die Signatur und ein Trennzeichen ebenfalls dort noch Platz finden müssen.

LOCAL

Variablendeklaration

LOCAL 0
LOCAL 1
LOCAL I%,F#,S$,C&,B!, &PI%,&PF#,&PS$,&PC&,&PB!

Mit der Direktive LOCAL kann man die bei BASIC-Sprachen übliche implizite Variablendeklaration deaktivieren; dazu muß sie mit dem Parameter 1 benutzt werden. Eine Benutzung mit Parameter 0 aktiviert die implizite Deklaration wieder.
Um bei deaktivierter impliziter Variablendeklaration lokale Variablen benutzen zu können, muß man sie mit der Direktive LOCAL deklarieren.

Beispiel:

```
LOCAL 1
LOCAL alter,name$
input "Alter: ",alter
input "Name:,name
```

LOCAL ist vor allem beim Aufspüren von Tippfehlern hilfreich, da es bei nicht deklarierten Variablen einen Fehler auslöst.
Auf globale Variablen, die mittels CONST oder DIM deklariert wurden, und auf Prozedur-/Funktionsparameter hat LOCAL keinen Einfluß.

LOCATE

Setzen des Kursors
LOCATE x%, y%
GOTOXY x%, y%
Setzt den alphanumerischen Kursor auf die angegebene Position.

LOG#

Logarithmen
LOG#(f#)
LN#(f#)
LOG2#(f#)
Dekadischer Logarithmus, natürlicher Logarithmus und duadischer Logarithmus.

LOWCASE$

Zeichenkette in Klein-/Großbuchstaben umwandeln
LOWCASE$(s$)
UPCASE$(s$)
Alle Zeichen der Zeichenkette s$ werden in Klein-/Großbuchstaben umgewandelt.
Achtung: Funktioniert nur für ASCII-Codes (<128)!

MESSAGEBOX

Darstellung einer Meldung
MESSAGEBOX mess$, n%
Stellt die gegebene Meldung mess in einer Box im Zentrum des Bilschirms dar. Der Parameter nr steuert die Rückkehr der Prozedur und das Löschen der Meldung:

n	
0	Warten auf Antippen
1	Sofort Schließen
2	0.125 Sekunden warten
3	0.5 Sekunden warten
4	1 Sekunde warten

MID$

Teilstring
MID$(s$, i%[, l%=1])
Ein String der Länge l ab Position i aus dem String s wird zurückgegeben:

```
MID$("Test",1)"e"
MID$("Test",1,2)"es"
```

RIGHT$ - Rechter Teilstring
LEFT$ - Linker Teilstring

MIN#

Minimum und Maximum
MIN#(f1#, f2#)
MAX#(f1#, f2#)
Minimaler bzw. maximaler Wert der zwei gegebenen Argumente

MIN

Minimum und Maximum
MIN%(i1%, i2%)
MAX%(i1%, i2%)
Minimaler bzw. maximaler Wert der zwei gegebenen Argumente

MODE

Zugriff auf andere Anwendungen
Anwendungen werden auf dem Pocketviewer "Modus" genannt. Mit den OWBasic-Prozeduren ist es möglich, den Namen eines Modus abzufragen (MODENAME), einen Modus mit einem bestimmten Namen zu finden (MODEFIND) und einen Modus zu starten (MODEJUMP).
Die Modusnummern außer der 8 sind den eingebauten Anwendungen zugeordnet. Alle Addins haben die Modusnummer 8 und eine Submodus-Nummer zwischen 0 und 15.

MODEFIND

Findet ein Addin nach dem Namen
MODEFIND name$, MODE%, SUBMODE%
Ermittelt MODE und SUBMODE für das Addin name$.

MODEICON

Menü-Icon eines Modus laden
MODEICON mode%,submode%,ICON%[]
Lädt das Menü-Icon des durch mode und submode gegebenen Modus nach ICON. ICON kann mit DRAWICON auf dem Bildschirm dargestellt werden.

MODEJUMP

Modus starten
MODEJUMP mode%,submode%[,status%][,param$][,cat%,cursorpos%]
Startet den durch mode und submode gegebenen Modus. Optional kann der Status und ein Parameter vorgegeben werden.
Werden cat und cursorpos angegeben, so werden die Parameter nach ModeCall-Konvention zusammengefügt.

MODELISTICON

List-Icon eines Modus laden
MODELISTICON mode%,submode%,ICON%[]
Lädt das List-Icon des durch mode und submode gegebenen Modus nach ICON. ICON kann mit DRAWICON auf dem Bildschirm dargestellt werden.

MODENAME

Namen eines Modus bestimmen
MODENAME mode%, submode%, NAME$
Bestimmt den Namen des durch mode und submode gegebenen Modus.

MODEVER$

Version eines Modus bestimmen
MODEVER$(mode%,submode%,ver%)
Bestimmt die Version des durch mode und submode gegebenen Modus. Wenn ver 0 ist, wird die Version des Modus bestimmt, ist ver 1, so wird die Version der verlinkten Bibliothek zurückgegeben.
Der zurückgegebene String hat den folgenden Aufbau:
"yyyymmddhhmmvvvv"
yyyymmdd = Jahr, Monat, Tag
hhmm = Stunden, Minuten
vvvv = Version

```
Beispiel:
"1999101815300100"
19991018 – Datum: 18.10.1999
1530 – Zeit: 15:30
0100 – Version: 1.00
```

MODES

Modi für Dateiarbeit
Bei der Arbeit mit LOADFILE, SAVEFILE und den Dateizeiger-orientierten Prozeduren müssen Modi und Submodi für die Dateien angegeben werden. Modi bezeichnen eine Art von Dateien, was normalerweise mit jeweils einer eingebauten Anwendung korrespondiert.
CASIO dokumentiert im PV-SDK folgende Modes:

Modus	Applikation	Typ
1	TEL	Text
2	TEL(Company)	Text
3	Memo	Text
4	Schedule	Text
5	Spreadsheet	Binär
6	Mail	Binär
7	EXPENSE	Text
8	Clock (City setting)	Text
9	Dual window	Text

10	ADD IN	Binär
11	Quick Memo	Binär

submode	Kategorie
<TEL>	
0	Common
1	Personal
2	Company
3	Untitled1
4	Untitled2
5	Untitled3
6	Untitled4
7	Untitled5
15	Owner
<Memo>	
0	Common
1	Untitled1
2	Untitled2
3	Untitled3
4	Untitled4
5	Untitled5
<Schedule>	
0	Common
1	Schedule
2	Schedule (Term registration)
3	Reminder
4	Holiday setting
6	Schedule (Common)
8	TODO (Common)
9	TODO (CATEGORY A)
10	TODO (CATEGORY B)
11	TODO (CATEGORY C)
12	TODO (CATEGORY D)
13	TODO (CATEGORY E)
14	TODO (CATEGORY E)
15	TODO (CATEGORY F)
<EXPENSE>	
1	Account
2	Transaction
3	Payment type
<Dual-window>	
1	Clip
<Quick Memo>	
0	Common
1	Untitled1
2	Untitled2
3	Untitled3

Die Submodi korrespondieren mit den Kategorien in den meisten Anwendungen.

Das Pocketviewer Betriebssystem benutzt Submodi von 0 bis 15. In OWBasic werden die Modi von 16 bis 31 dafür verwendet, die Kategorien 0 bis 15 des geheimen Bereichs anzusprechen. Zugriff zu diesen ist jedoch nur möglich, wenn das Passwort einmal eingegeben wurde.

MSGBOX

Darstellung einer Message-Box auf dem Bildschirm

MSGBOX message$ [, caption$=""] [, time#=0.0] [, type%=2]

Dieser Befehl stellt eine Box mit dem Text message in der Mitte des Bildschirmes dar. Wenn caption angegeben und kein Leerstring ist, wird auch ein Titel in Fettschrift angegeben. time gibt an, nach wie vielen Sekunden die Funktion zurückkehrt.
type spezifiziert das Verhalten der Message-Box:

type	
0	Die Message-Box wird nur dargestellt
1	Die Prozedur kehrt nach time Sekunden zurück
2	Die Prozedur kehrt nach dem Berühren des Bildschirms zurück

MULDIV

Kombinierte Multiplikation und Division

MULDIV%(a%, b%, c%)

Berechnet den Wert a*b/c, wobei der Wert von a*b den Integerbereich überschreiten darf. Dies ist eine häufig genutzte Methode, um quasi Integerzahlen mit einer gebrochenen Zahl (a) zu multiplizieren (b/c).

Beispiel:
Umfang eines Kreises mit dem Radius r:

```
U= r * 355 / 113
```

Diese im ganzzahligen Bereich sehr gute Näherung führt bereits für kleine r zu einem Überlauf. Mit Muldiv wird diese vermieden, solange nur u den Integer-Bereich nicht überschreitet:

```
U=MULDIV(r,355,113)
```

NAMEDFILE

Ermitteln von Modus und Submodus von Addin-Dateien

NAMEDFILE name$, MODE%, SUBMODE%

Dateien der Add-Ins besitzen Namen. NAMEDFILE liefert zu diesen Dateien den Modus und Submodus, die zum Zugriff benötigt werden. Wenn die Datei noch nicht existiert, wird sie angelegt.
Erstaunlicherweise ist NAMEDFILE sehr zeitaufwendig. Rufe NAMEDFILE nur einmal in deinem Programm auf und speichere das Ergebnis zur weiteren Verwendung ab.

NEW25

Neu in Version 2.5

Wesentliche neue Merkmale der Version 2.5 von OWBasic sind:
- Der logische Datentyp Boolean und entsprechende Operatoren.
- Einfachere Behandlung von Arrays als Pozedurparameter.
- Benutzung von Overlays für sehr große Programme
- Formulare
- Ein Fernbedienungsmodus zur Entwicklung von Programmen auf dem PC.
- Status des Action-Reglers
- STEP in FOR-Schleifen kann negative Argumente haben.

NEW30

Neu in Version 3.0

Wesentliche neue Merkmale der Version 3.0 von OWBasic sind:
- Neue Steueranweisungen CASE, LOOP
- TEXTBOX, DRAWTEXT haben Parameter für Zeichensatz
- Neue Zeichenprozeduren für Kreis und Ellipse: CIRCLE, ELLIPSE
- Zusätzliche neue Bedeutung von VERSION als "Erforderliche Version"

NEW32

Neu in Version 3.2

Wesentliche neue Merkmale der Version 3.2 von OWBasic sind:
- Zeitmessung mit TIMER
- Anwendungen synchronisieren mit einer inneren Uhr mit SYNC
- Quickmemos mittels Hashwert auswählen (LOADQMHASH)
- Variablen die die OWBasic-Version und das PV-Modell angeben (OWBVERSION, PVMODEL)
- Neuer Datentyp Zeichen (Character)

NEW40

Neu in Version 4.0

Wesentliche neue Merkmale der Version 4.0 von OWBasic sind:
- Zweidimensionale Felder
- Vom Nutzer definierbare Funktionen
- "Echte" Konstanten mit CONST
- Lesen von Pixeln mittels der Funktion PIXEL

NEW42

Neu in Version 4.2

Neue Merkmale der Version 4.2 von OWBasic sind:
- Es ist möglich, Zeigervariablen und –arrays zu deklarieren und zu setzen

- Befehle zum Setzen eines Array-Bereiches auf einen Wert (SETI, SETF, SETS, SETB, SETC)
- TIMER%() als Integer-Funktion
- Inkrement-/Dekrement-Funktionen für Float-Variablen (INCF, DECF)
- FLASHSIZE nimmt als optionalen dritten Parameter eine Variable an, in der es die Anzahl der bei einer Speicherverwaltung freiwerdenden Datenblöcke ablegt (nur alte PVs)
- SETTIME setzt die Systemzeit.
- SYSDLG zeigt systemverwandte Dialoge an.
- FBHASH und FBBHASH zur Errechnung des Hashwertes des Dateipuffers
- GETFP gibt den Filepointer einer Dateinummer zurück
- optionaler Parameter mode bei DRAWICON steuert die Art der Icon-Darstellung
- SCROLL verschiebt einen Bildschirmausschnitt
- EXEC nimmt zwei zusätzliche optionale Parameter zur Steuerung der Compiler-Anzeige und zur Übergabe eines Parameters an das zu startende OWBasic-Programm an
- PARAMETER$ enthält den Parameter, mit dem ein Programm aufgerufen wurde
- Die Funktion SGN() gibt das Vorzeichen einer Integerzahl zurück.
- Konstanten dürfen negative Werte haben
- Über die globale Variable CLIPBOARD kann der Zwischenspeicher ausgelesen und verändert werden
- Initialisierte CHAR- und BOOL- Konstanten erlaubt:

```
CONST C&=("my char array")
CONST B!=(true,false,1,0)
```

- Die Steueranweisung CASE erlaubt Bereichsüberprüfungen (CASE <=5, CASE 5 TO 8)
- Explizite Umwandlung von BOOL und CHAR nach INT ist erlaubt

NEW44

Neu in Version 4.4

Neue Merkmale der Version 4.4 von OWBasic sind:

- Bedingte Kompilierung mittels Direktiven (->DIRECTIVE)
- LOCAL zur Variablendeklaration
- Funktions- und Prozedurzeiger (->FPOINTER)
- konstante Ausdrücke werden nun zur Compilezeit ausgewertet
- FILEFIND%() sucht eine Datei nach ihrem Namen
- FILELABEL$() gibt den Titel einer Datei zurück
- FBBTELL%() gibt die aktuelle Position des internen Filebuffer-Zeigers zurück
- DRAWSTRING kann einen String rechtsbündig darstellen
- DUP$() dupliziert ein Char
- Mittels INPUTI, INPUTF und INPUTS kann der Benutzer den Wert einer Variablen komfortabel verändern
- TOUCH!() nimmt vier optionale Parameter als Koordinatenfenster entgegen
- TCHINSIDE!() überprüft, ob TCHX und TCHY in einem Koordinatenfenster liegen
- TCHSET setzt den Status des Touch-Panel

- die globale Variable FBBSIZE enthält die Größe des binären Dateipuffers
- über die konstanten Funktionen VARDEF und PROCFUNCDEF läßt sich abfragen, ob eine Variable/Prozedur/Funktion definiert ist
- NOT ist ein unärer Operator und existiert für Ausdrücke der Datentypen Integer (bitweise Negation) und Boolean (logische Negation)
- Konstanten kann zur Compilezeit ein neuer Wert zugewiesen werden
- Die Integerkonstante #PVMODEL enthält einen Wert, der das PV-Modell repräsentiert
- Initialisierte STRING-Konstanten erlaubt:
  ```
  CONST S$=("one", "two", "three")
  ```
- In Stringliteralen können einzelne Zeichen hexadezimal angegeben werden (->STRINGCONSTANT)
- BREAK und CONTINUE in Schleifen erlaubt
- Funktionen können als Prozeduren (ohne Verwendung des Rückgabewertes) aufgerufen werden (->FUNC)
- mehrzeilige Kommentare mit "/*" und "*/" möglich (->FORMAT)
- die Anzahl der maximal möglichen Programmcodes wurde geringfügig erhöht (von 5136 auf 5392)

NEW50

Neu in Version 5.0

Neue Merkmale der Version 5.0 von OWBasic sind:

- Die graphische Oberfläche (GUI) von OWBasic wurde erneuert; OWBasic erscheint jetzt in dem für Add-Ins üblichen Aussehen. Außerdem lädt OWBasic jetzt deutlich schneller.
- Der Dialog zum Ändern der Optionen ist nicht mehr in OWBasic enthalten; zum Konfigurieren der Optionen ist das Programm "OWBasic Settings", das in der OWBasic Standard Library enthalten ist, nötig.
- Das Zwischenspeichern eines kompilierten Programmes in die SHARED-Datei (Caching) ist möglich. (->OPTIONS)
  ACHTUNG: Damit OWBasic bei dem Start eines Programmes nach einer zwischengespeicherten Version sucht, muß in den Optionen "Caching" aktiviert sein!
- Wenn ein Laufzeitfehler auftritt, fragt OWBasic, ob es nach der Fehlerposition suchen soll. Wurde das Programm von IEdit gestartet, so sucht IEdit, nachdem der Fehler gefunden wurde, die Fehlerposition auf.
- Einige Fehlermeldungen enthalten nun zusätzliche Informationen; z.B. wird bei Auftreten des Fehlers "Index out of range" auch der fehlerhafte Index angezeigt.
- Der Compiler optimiert Variablen- und Arrayzugriffe, so daß weniger Programmcodes entstehen. Konstante Ausdrücke wie z.B. der Zugriff auf ein globales Array mit einem konstanten Index (z.B. PRINT arr[1]) werden zur Kompilierzeit ausgewertet, so daß der gleiche Code wie für den Zugriff auf eine normale Variable erzeugt wird.
- Anwenderprozeduren sind nicht mehr auf 250 (PV-S1600: 1000) limitiert, da sie nun in die Symboltabelle eingetragen werden.
- Die Kapazität für Symbole, Daten und Programmcode und die maximale Stringlänge wurden vergrößert (nicht auf dem PV-S1600). Für Daten stehen nun nicht 12288, sondern 16128 Words zur

Verfügung; anstatt 5392 sind nun maximal 5632 Programmcodes möglich. Die maximale Stringlänge beträgt nicht mehr 150, sondern 220 Zeichen. Außerdem stieg die maximale Anzahl verschachtelter Kontrollstrukturen von 15 auf 19 und die maximale Bezeichnerlänge von 17 auf 30.

- Die Syntax für die Deklaration von Funktionszeigern wurde geändert (->FPOINTER)
- Das Typsystem des Compilers wurde komplett überarbeitet (->VARIABLES); Zeiger sind jetzt typsicher, außerdem können eigene Typen mit ENUM und FPTR erstellt werden.
- Umwandlungen zwischen Zeigern und Integern sind möglich. (->INT, ->POINTER)
- Mit dem Operator '*' ist das explizite Dereferenzieren von Zeigern möglich. (->POINTER)
- Funktionen können Pointer zurückgeben (-> FUNC)
- Die globale Konstante #INTSIZE gibt die Größe eines Integers in Byte an (alte PVs: 2, PV-S1600: 4)
- Der Button "Menu Bar" in der Hardicon-Leiste spricht nun nicht mehr den Abort-Dialog an, sondern kann von OWBasic-Programmen abgefragt werden.
- Die Funktion SETJUMP%() speichert den Code-Pointer des Interpreters und zusätzliche Daten in einem Array der Größe #JUMPBUFSIZE. Die darin gespeicherte Programmstelle kann später von überall aus mittels JUMP angesprungen werden (vgl. setjmp () und longjmp () in ANSI C).
- Das globale Array ERRORHANDLER kann durch die Funktion SETJUMP%() einen Code-Pointer aufnehmen, der – sofern ungleich 0 – bei Auftreten eines Laufzeitfehlers angesprungen wird.
- Die Funktion APOTIME%() fragt die Zeit bis zum automatischen Abschalten des PV in 500ms-Einheiten ab. Mit der Prozedur SETAPOTIME kann diese verändert werden.
- Die Buttons "Light" und "Off" in der Hardicon-Leiste werden von OWBasic abgefragt. Ist dies nicht erwünscht, so kann die Funktion mit der Prozedur SYS abgeschaltet werden.
- Wenn durch SYS aktiviert, schaltet sich der PV nach einer gegebenen Zeit ohne Betätigung des Touchscreens automatisch ab (AutoPowerOff).
- Der PV kann auf Befehl ausgeschaltet werden (->POWEROFF).
- Über SYSDLG können auch Systemeinstellungsdialoge (Kontrast, Format etc.) aufgerufen werden.
- FILEREAD nimmt als optionalen vierten Parameter die Anzahl der Blocks an, die in den Filebuffer geladen werden soll (1 Block = 64 Bytes).
- POLYGON stellt ein Polygon dar.
- MSGBOX stellt eine Message-Box auf dem Bildschirm dar.
- Die Funktion TOUCHED!() kann auch ohne Parameter aufgerufen werden; sie bewirkt dann das gleiche wie NEWTOUCH!().
- Über die Funktion LIGHT!() kann der Status der EL-Hintergrundleuchte abgefragt werden.
- DRAWICON kann ein Icon auch teilweise darstellen.
- Mittels CONST können auch initialisierte Arrays von Daten- und Funktionszeigern erstellt werden.
- Prozeduren und Funktionen können Default-Parameter besitzen (->PROC; ->FUNC)

- Mittels ENUM können Aufzählungstypen deklariert werden
- Über die Funktion SYSLANG%() kann die Systemsprache abgefragt werden
- Der Name des im Autorun-Modus gestarteten Programmes kann mit OWBasic Settings angepaßt werden
- Über SYSDLG sind zwei neue Dialoge verfügbar
- Mit MODELISTICON kann das zu einem Modus gehörende List-Icon geladen werden
- Version und Dateidetails eines Modus werden über MODEVER$() bereitgestellt
- Mit INPUTTEXT kann eine Stringvariable in einem definierbaren Bereich editiert werden
- Mittels KBDRAW und KBWAIT! kann eine Tastatur dargestellt und abgefragt werden

NEWTOUCH!

Test auf Berührung
NEWTOUCH!()
NEWTOUCH%()
Die logische Funktion NEWTOUCH!() liefert TRUE, wenn der Touchscreen neu berührt wurde, sonst FALSE. Dies ermöglicht die einmalige Auslösung bei Berührung.
Die Integer-Funktion NEWTOUCH() (nicht mehr empfohlen) liefert einen Wert größer Null, wenn der Touchscreen neu berührt wurde, sonst Null.
Wird eine Berührung gemeldet, sind in den Integer-Variablen TCHX und TCHY die Koordinaten der Berührung abgelegt.

NOT!

Logische Negation
NOT b!
Negiert den logischen (booleschen) Wert von b!.
NOT TRUE -> FALSE; NOT FALSE -> TRUE
Der Operator NOT hat die höchste Priorität.

NOT

Bitweise Negation
NOT i%
Gibt den bitweise negierten Wert von i zurück.
0xFFFF -> 0x0000; 0x0000 -> 0xFFFF
Der Operator NOT hat die höchste Priorität.

NRT#

Wurzel
NRT#(v#, e#)
Die e. Wurzel aus v. v muß positiv sein.

OLDER25

Kleine Probleme mit älteren Programmen

Ein paar kleinere Probleme können mit älteren Programmen auftreten: Koordinaten werden strikter getestet. Zeichenfunktionen (außer LINE) melden einen Fehler, wenn die Koordinaten den (ansonsten unveränderten) Wertebereich von 0 bis 159 überschreiten. Ein Test von Koordinaten ist sehr leicht mit INSIDEG!() möglich.

- Die booleschen Konstanten TRUE und FALSE sind neue Schlüsselwörter, die nicht als Namen von Variablen und Prozeduren benutzt werden können.
- Die Code-Größe von Programmen ist geringfügig gestiegen. Alte Programme, die bisher fast den ganzen Speicherbereich nutzten, können für die Übersetzung zu groß sein.
- Es gibt die Integer-Operatoren AND, OR und XOR, die die Integer-Operanden bitweise logisch verknüpfen und die Booleschen Operatoren AND, OR und XOR, die logische Operanden verknüpfen. Um sicherzustellen, daß der jeweils richtige Operator genutzt wird, sollten die Integer-Operatoren mit Klammern verwendet werden:

```
IF (a AND 1)>0 THEN .. : ! ist das Bit 1 der Variable a gesetzt
```

Bei Verwendung ohne Klammern würde das so interpretiert:

```
IF a AND (1>0) THEN ..
```

was einen Typfehler liefert: a ist integer und (1>0) ist boolesch.

```
IF A>0 AND A<10 THEN .. : ! liegt a im Bereich 1..9
```

Hier werden keine Klammern benötigt. (Allerdings würde man hier besser INRANGE!() verwenden)

OPERATOR

Arithmetische Operatoren

Operatoren und ihre Verfügbarkeit in der Arithmetik:

Stufe	Operator	Integer	Float	String	Char	Bool	Funktion
0	^	yes	yes*	no	no	no	Potenzierung
1	*	yes	yes	no	no	no	Multiplikation
1	/	yes	yes	no	no	no	Division
1	%	yes	no	no	no	no	Modulo
2	+	yes	yes	yes	yes	no	Addition
2	-	yes	yes	no	no	no	Substraktion
3	<	yes	yes	yes	yes	no	Kleiner als
3	<=	yes	yes	yes	yes	no	Kleiner oder gleich
3	=	yes	yes	yes	yes	no	Gleich
3	>	yes	yes	yes	yes	no	Größer oder gleich
3	>=	yes	yes	yes	yes	no	Größer als
3	<>	yes	yes	yes	yes	no	Ungleich
4	AND	yes	no	no	no	yes	UND
5	OR	yes	no	no	no	yes	ODER
5	XOR	yes	no	no	no	yes	Exklusiv-ODER

Operatoren niedrigerer Stufe werden zuerst ausgewertet. Operatoren der gleichen Stufe werden von links nach rechts ausgewertet.
Das Ergebnis der Vergleichsoperatoren ist vom Typ Boolesch.
Die logische (boolesche) Negation wird mittels der Funktion NOT!() realisiert.
Anmerkung: Der Modulo-Operator ist zugleich Typkennzeichen (Integer). Deshalb muß bei Verwendung als Modulo-Operator immer ein Leerzeichen nach einer vorhergehenden Variable stehen.
* - Float-Variablen sind nur als Basis (linker Operand) erlaubt. Falls ein Gleitkomma-Exponent benötigt wird, muß POW#() verwendet werden.

OPTIONS

Menüpunkt Options
Das Menü "Options" im Startmenü von OWBasic erlaubt einige erweiterte Aktionen:
- Aufruf des OWBasic-Programmes "OWBasic Settings" zum Einstellen der Optionen
- Finden von Laufzeitfehlern
- Aufruf des Remote Mode (nicht PV-S1600)
- Zwischenspeichern eines vorkompilierten Programmes (Caching)

OUT

Port-Ausgabe
OUT port%, nr%
Ausgabe des Wertes nr (Byte) auf den Port port.

OVERLAY

Prozeduren als Overlay übersetzen
OVERLAY 1
OVERLAY 0
OVERLAY 1 veranlaßt den Compiler, die nachfolgenden Prozeduren als eine Overlay-Gruppe zu übersetzen, bis eine neues OVERLAY-Kommando auftritt. Ein neues OVERLAY 1 startet eine neue Overlay-Gruppe, OVERLAY 0 beendet die Übersetzung als Overlay.
Overlays können als Mittel zur Reduktion des Speicherbedarfs genutzt werden. Prozeduren, die als Overlay übersetzt werden, belegen zur Laufzeit den gleichen Speicherplatz im RAM, natürlich nicht gleichzeitig. Während der Übersetzung werden diese Prozeduren im Flash-RAM abgelegt und zur Laufzeit bei Bedarf geladen. Daraus resultieren die folgenden Eigenschaften:
- Der Speicherbedarf aller Prozeduren einer Overlay-Gruppe entspricht dem Speicherbedarf der größten Prozedur.
- Zur Übersetzung des Programmes muß freier Flash-RAM verfügbar sein.
- Die Übersetzung benötigt mehr Zeit, da das Ablegen im Flash-RAM zeitaufwendig ist.

Weniger zeitaufwendig, aber oft nicht vernachlässigbar, ist das Laden der Prozeduren zur Laufzeit (ca. 15 ms).
Prozeduren einer Overlay-Gruppe können einander nicht aufrufen.

PARAMETER$

Aufrufparameter des Programmes
PARAMETER$
Die globale Variable parameter enthält den Aufrufparameter des aktuellen Programms. Wurde beim Start mittels EXEC kein Parameter angegeben oder das Programm aus dem Menü gestartet, so enthält parameter einen leeren String.

PEEK

Lesen von Bytes aus dem Hauptspeicher
PEEK%(seg%, offs%)
Die Funktion liefert den Wert des Bytes, das im Segment seg an der Stelle offs im Speicher steht. Die absolute Adresse dieses Bytes ist seg*16+offs. Dabei werden seg und offs als vorzeichenlose Zahlen interpretiert, ein (Basic-)Wert von -1 zum Beispiel entspricht dem vorzeichenlosen Wert 65535.

POKE - Schreiben von Bytes in den Hauptspeicher
IN - Port Eingabe

PIXEL

Pixel lesen
PIXEL%(x%, y%)
PIXEL!(x%, y%)

Die Funktion PIXEL gibt den aktuellen Wert des Pixels (x,y) zurück.

Wert (int)	Wert (bool)	Pixel
0	FALSE	Weißes Pixel
1	TRUE	Schwarzes Pixel

PLAY

Spielen einer Melodie
PLAY mel$, dur%, vol%
Spielt eine Melodie, die in mel kodiert ist. Die Zeichen in mel haben die folgende Bedeutung.

Zeichen	Bedeutung
'a'..'h'	Spielen des entsprechenden Tones
'#'	Erhöhe die nächste Note um eine halbe Note (z.B. c -> cis)
'+'	Wechsel zur nächst höheren Oktave
'-'	Wechsel zur nächst niedrigeren Oktave
1	Auswahl einer ganzen Note

```
2           Auswahl einer halben Note
4           Auswahl einer Viertel-Note
8           Auswahl einer Achtel-Note
6           Auswahl einer Sechzehntel-Note
' ' oder 'p' Pause
'r'         Rücksetzen auf die Startwerte
```

Die Wechsel der Oktave und der Notenlänge bleiben zwischen den Aufrufen erhalten, bis sie explizit duch das Zeichen 'r' zurückgesetzt werden. Der Parameter dur ist die Länge der ganzen Note. vol ist die Lautstärke analog zu SOUND.

POINTER

Zeigeroperationen

In OWBasic ist es möglich, Zeigervariablen zu deklarieren und zu verwenden. Vor Version 4.2 existierten Zeiger ausschließlich als VAR-Parameter in Prozeduren und konnten nicht verändert werden.
Um die Abwärtskompatibilität zu gewährleisten, werden Zeiger bei ihrer Verwendung automatisch dereferenziert (d.h., es wird auf das, worauf der Zeiger zeigt, zugegriffen); um ihren Wert abzufragen oder zu verändern, muß die spezielle Zeigersyntax verwendet werden.
Es ist auch möglich, Arrays von Zeigern zu deklarieren, für deren Benutzung wiederum eine spezielle Syntax existiert.

Um eine Zeigervariable zu deklarieren, weisen Sie ihr einen Wert zu:
&<pointer><type-suffix> = <int-expression>

Wenn Sie einer Zeigervariablen noch keine bestimmte Adresse zuweisen wollen, ist es ratsam, den Zeiger auf die symbolische Konstante NULL zeigen zu lassen:
&<pointer><type-suffix> = #NULL

Normalerweise weist man einem Zeiger die Adresse einer anderen Variable zu. Um die Adresse zu erhalten, wird der &-Operator angewendet:
&<pointer><type-suffix> = &<variable>

Ein Zeiger kann auch auf ein Element eines Arrays zeigen:
&<pointer><type-suffix> = &<variable>[<int-expression>]

Um die Adresse einer Variable, auf die bereits ein anderer Zeiger zeigt, zu übernehmen, kann diese wie bei einer normalen Variable abgefragt werden:
&<pointer2><type-suffix> = &<pointer>

Um die flexible Handhabung von Arrays zu ermöglichen, ist es erlaubt, Pointer zu indizieren:
<variable> = <pointer>[< int-expression>]
Dieser Code setzt <variable> auf den Wert der Variablen, die im Speicher um <int-expression> Einheiten weiter als die Variable, auf die <pointer> zeigt, steht. Das Ergebnis ist nur definiert, wenn <pointer> auf ein

Element eines Arrays zeigt und das Array größer ist als <pointer> + <int-expression> .

Zeigerarrays lassen sich mit DIM deklarieren:
```
DIM &<pointer-array><type-suffix>[<constant expression>]
```

Um auf ihre Elemente zuzugreifen, muß wieder eine spezielle Syntax verwendet werden:
```
&~<pointer-array>[<int-expression>] = <int-expression>
```
oder alternativ in der vereinfachten, aber inkonsequenten Syntax:
```
&<pointer-array>[<int-expression>] = <int-expression>
```

Dementsprechend wird auch auf ein Pointer-Array zugegriffen:
```
&<pointer> = &~<pointer-array>[<int-expression>]
```

Um auf den von einem Element eines Zeiger-Arrays bezeigten Wert zuzugreifen, bedarf es folgender Syntax:
```
~<pointer>[<int-expression>] = <expression>
<variable> = ~<pointer>[<int-expression>]
```

Bei Bedarf kann auch ein solcher Pointer indiziert werden:
```
~<pointer>[<int-expression>][<int-expression2>] = <expression>
<variable> = ~<pointer>[<int-expression>][<int-expression2>]
```

In einigen Fällen möchte an auf die Adresse eines Pointers zugreifen. Beispielsweise kann das nützlich sein, wenn man einen Pointer über ein Array iteritieren läßt:
```
INC &&<pointer>
INC &&<pointer>[<int-expression>]
```

ACHTUNG: Dies funktioniert nicht bei Zeigern auf Float-Arrays, da Float-Zahlen mehr Speicherplatz benötigen als Integerzahlen (bei den anderen Datentypen ergeben sich keine Probleme, da Char- und Bool-Variablen intern als Integervariablen dargestellt und bei Strings wiederum nur Integer-große Zeiger verwaltet werden). Um über ein Float-Array zu iteritieren, sollte die folgende Vorgehensweise angewandt werden:
```
INC &&<float-pointer>,#FLOATSIZE
INC &&<float-pointer>[<int-expression>],#FLOATSIZE
```
FLOATSIZE ist eine globale Konstante und gibt die Größe von Float in Einheiten von Integer an.

Um einem Zeiger einen Integerwert zuzuweisen, kann man einen solchen mittels `POINTER@(i%)` in einen Zeiger beliebigen Typs umwandeln (das `@` steht hier für das Typ-Suffix bzw. die Angabe eines eigenen Typs).

Ein Zeiger, der nicht automatisch dereferenziert wird (z.B., wenn er von einer Funktion zurückgeliefert wird oder Ergebnis eines komplexen Ausdrucks ist), kann mit dem Operator '*' explizit dereferenziert werden:
```
i=*(&array+2) !: äquivalent zu i=array[2]
```

Pointer und Pointer-Arrays können nur eindimensional indiziert werden.

Zeiger sind bei unachtsamer Verwendung sehr gefährlich; z.B. findet bei der Indizierung von Zeigern keine Bereichsüberprüfung statt. Auch kann

nicht überprüft werden, ob ein Zeiger auf die korrekte Speicherstelle zeigt; dies kann vor allem im Zusammenhang mit der String-Verwaltung in OWBasic fatale Folgen haben. Aus diesem Grunde wird empfohlen, die Zeiger wenn überhaupt unter äußerster Vorsicht anzuwenden.

POINTEREXAMPLE

Beispiel zur Verwendung von Pointern

Da Zeiger in BASIC-Sprachen recht selten sind, sei hier noch ein einfaches Beispiel im Vergleich zur Programmiersprache C gegeben, das die Verwendung der Pointer verdeutlicht:

C-Quelltext:

```c
int main (void)
{
  int a; /* integer variable */
  int b[3]; /* integer array */
  int* p1; /* pointer to integer variable */
  int* p2[3] /* array of integer pointers */

  a = 0; /* a is set to 0 */
  p1 = &a; /* p1 points to a */
  *p1 = 5; /* a is set to 5 */
  b[0] = *p1; /* b[0] is set to the value of the variable p1 is
                 pointing at (a) */
  p2[0] = &b[0]; /* p2[0] contains the address of b[0] */
  *p2[0] = 7; /* the value p2[0] is pointing at (b[0]) is set to 7 */
  p2[0][1] = 21; /* the value after the value p2[0] is pointing at
                    (b[1]) is set to 21 */
  *p1 = p2[0][2]; /* a is set to the value of b[2] */
  p1 = p2[0]; /* p2 is assigned the address p2[0] contains (b[0]) */
  ++p1; /* p1 points now to the value after b[0] (b[1]) */

  return (0);
}
```

OWBasic-Quelltext:

```
DIM b[2] :! integer array
DIM &p2[2] :! array of integer pointers

a = 0 :! a is set to 0
&p1 = &a :! p1 points to a
p1 = 5 :! a is set to 5
b[0] = p1 :! b[0] is set to the value of the variable p1 is pointing
          ! at (a)
&~p2[0] = &b[0] :! p2[0] contains the address of b[0]
~p2[0] = 7 :! the value p2[0] is pointing at (b[0]) is set to 7
~p2[0][1] = 21 :! the value after the value p2[0] is pointing at
               ! (b[1]) is set
                 to 21
p1 = ~p2[0][2] :! a is set to the value of b[2]
&p1 = &~p2[0] :! p2 is assigned the address p2[0] contains (b[0])
INC &&p1 :! p1 points now to the value after b[0] (b[1])

END 0
```

POKE

Schreiben von Bytes in den Hauptspeicher
POKE seg%, offs%, val%
Schreibt den Wert val auf das Byte, das im Segment seg an der Stelle offs im Speicher steht. Die absolute Adresse dieses Bytes ist seg*16+offs. Dabei werden seg und offs als vorzeichenlose Zahlen interpretiert, ein (Basic-)Wert von -1 zum Beispiel entspricht dem vorzeichenlosen Wert von 65535.

POLR#

Umrechnung in Polarkoordinaten
POLR#(x#, y#)
POLPHI#(x#, y#)
Berechnet die Komponenten r (Radius, Betrag) und phi (Winkel) der Polarkoordinaten, die den kartesischen Koordinaten x und y entsprechen. Die Angabe des Winkels phi erfolgt im aktuellen Modus.

POLYGON

Darstellung eines Polygons
POLYGON COORDS%[] [, style%=1]
Diese Funktion stellt ein Polygon dar; die Koordinaten werden aus COORDS gelesen. Das erste Element in COORDS gibt die Anzahl der x-/y-Koordinatenpaare, die jeweils nächsten zwei die x- und y-Koordinaten des Eckpunktes an. Der optionale Parameter style definiert Attribute und Linientyp (verschiedene Attribute können kombiniert werden):

style%	Attribut
100	geschlossenes Polygon
000	offenes Polygon
1	schwarzer Rand
2	weißer Rand
4	punktierter Rand
5	inverser Rand

POS

Suche nach Teilstring
POS%(teilstring$,string$ [,case_sensitive!=TRUE])
Liefert die Position des Teilstrings teilstring im String string (Werte >=0) bzw. den Wert -1, falls teilstring nicht in string enthalten ist.
Wenn case_sensitive true ist, wird Groß- und Kleinschreibung nicht unterschieden.

POW#

Potenz
POW#(b#,e#)
Potenz e zur Basis b. b muß positiv sein.

POWEROFF

Ausschalten des Systems
POWEROFF
Schaltet das System aus. Das Programm wird nach dem Einschalten an der gleichen Stelle fortgesetzt.

POWI#

Potenz mit ganzzahligem Exponenten
POWI#(b#,e%)
Potenz e zur Basis b für positive, ganzzahlige Exponenten e.

PRINT

Ausgabe von Werten
PRINT [Wert] [Trennzeichen] ...
? [Wert] [Trennzeichen] ...
PRINT (oder ?) gibt Werte aus. Für Wert kann eine Variable, eine Konstante oder ein Ausdruck beliebigen Types stehen. Die einzelnen Werte werden durch die Trennzeichen ';' oder ',' getrennt. Diese haben außerdem folgende Auswirkungen auf die Formatierung:

; Die Werte werden ohne weitere Trennzeichen aneinandergefügt.
, Zwischen die Werte wird ein Tabulator eingefügt.

Steht am Ende der Ausgabeliste eines der beiden Trennzeichen, so erfolgt kein Zeilenvorschub. Die Darstellung von Gleitkommazahlen kann mittels SETFFORMAT geändert werden.

PROCDESC

Format der Prozedur- und Funktions-Beschreibungen
Bei den Beschreibungen der Prozeduren und Funktionen werden folgende Konventionen verwendet:
Prozedur- und Funktionsnamen werden groß geschrieben. Dies ist im Quelltext nicht erforderlich.
Parameter werden immer mit einem Typ-Kennzeichen versehen, um den Typ deutlich zu kennzeichnen.
Wahlfreie Parameter werden in eckigen Klammern geschrieben. Das spezielle Verhalten beim Weglassen dieser Parameter wird entweder im Text beschrieben, oder es wird der Wert angenommen, der beim Parameter nach einem Gleichheitszeichen steht.
Groß geschriebene Parameter bezeichnen Variablen-Parameter.
Beispiel:
PSET x%, y% [, z%=1]
Hier kann der Parameter z als Wert des Pixels weggelassen werden, wobei dann der Wert 1 für "Setzen" angenommen wird.

PROCEDURE

Prozeduren und Funktionen
Es gibt Prozeduren und Funktionen zu folgenden Bereichen:

- Standard-Text-Ein- und Ausgabe
- Grafik
- Touch-Screen und Action-Regler
- Formulare
- Ein-/Ausgabe auf dem Grafik-Bildschirm
- System-Prozeduren
- Flash-Speicherzugriff
- Kommunikation über die serielle Schnittstelle
- Bits und Bytes
- Integer-Funktionen
- String-Operationen
- Float-Funktionen

Es gelten besondere Konventionen in den Beschreibungen der Prozeduren und ihrer Parameter.

PROCFUNCDEF

Test, ob Variable definiert ist
PROCFUNCDEF!(<procfunc>)
Die konstante Funktion PROCFUNCDEF gibt TRUE zurück, wenn die Prozedur/Funktion <procfunc> definiert ist, ansonsten FALSE.

PSET

Pixel setzen
PSET x%, y% [, z%]
Setzt das Pixel (x,y) mit dem angebenen Wert z.

Wert	Funktion
0	Pixel löschen (Weiß)
1	Pixel setzen (Schwarz)
2	Pixel löschen (Weiß)
5	Pixel invertieren

PVMODEL

Informationen über das System
#PVMODEL
#OWBVERSION
PVMODEL%
OWBVERSION%

Die globale Variable PVMODEL% spezifiziert das PV-Modell, auf dem die Anwendung läuft. Für alle alten PVs hat sie den Wert 450, für den PV-S1600 den Wert 1600.
Die globale Variable OWBVERSION% und die Integerkonstante #OWBVERSION enthalten die aktuelle OWBasic-Version, also 440 für Version 4.40 .

Die Integerkonstante #PVMODEL enthält einen Wert, der das aktuelle PV-Modell spezifiziert:

Wert	Modell
1	PV-x50X
2	PV-750(+)
3	PV-Sx50
4	PV-Sx60, PV-Sx00Plus
5	PV-S1600

QUICKMEMO

Lesen und Schreiben von Quickmemos
Quickmemos sind das Grafikformat auf dem PV. Die OWBasic-Prozeduren zum Lesen und Schreiben von Quickmemos transferieren die Daten zwischen dem Flash-Speicher und Icons. Quickmemos lassen sich auf zwei Arten auswählen: Über ihre Nummer oder über einen Dateizeiger.

*	Nummer	Dateizeiger
Lesen	LOADQM	LOADQMFP
Schreiben	SAVEQM	SAVEQMFP

RAD#

Umrechnung von Winkeln in Bogenmaß
RAD#(phi#)
Wenn phi ein Winkel im aktuellen Modus ist, liefert die Funktion RAD den entsprechenden Winkel im Bogenmaß zurück.

RANDOM#

Zufallszahl
RND#(i%)
RANDOM#(i%)
Zufälliger Wert. Der Integerwert von i bestimmt die Art der Zufallszahl.

i	Zufallszahl
1	gleichverteilte Zufallszahl zwischen 0 und 1

RANDOM

Zufallszahl
RND%(i%)

RANDOM%(i%)
Zufälliger Wert zwischen 0 und i.

RANDOM# - Zufallszahl
RANDOMIZE - Neustart des Zufallszahlen-Generators

RANDOMIZE

Neustart des Zufallszahlen-Generators
RANDOMIZE [i%]
Jedesmal, wenn OWBasic gestartet wird, beginnt der Zufallszahlen-Generator mit der gleichen Zahlenfolge. Um eine echt zufälligen Zahlenfolge zu erzeugen, muß der Zufallszahlen-Generator neu initialisiert werden. Beim Aufruf von RANDOMIZE ohne Parameter wird die aktuelle Zeit als Initialisierungswert verwendet. Mit dem Parameter i wird der Zufallszahlengenerator mit diesem Wert initialisiert und liefert nach jedem RANDOMIZE mit demselben i dieselbe Zufallszahlenfolge.

RCMODE

Fernbedienungsmodus
Die Entwicklung von Basic-Programmen wird komfortabler, wenn man einen Editor auf dem PC nutzen kann.
Verbinde dazu deinen PV mit dem PC. Starte OWBasic und schalte über das Options-Menü in den Remote Control Mode.
Schreibe alle Teile deines Programmes im Editor auf dem PC. Speichere das Hauptprogramm in einer Datei mit der Endung .bas, Include-Dateien mit der Endung .inc und Append-Dateien mit der Endung .app in einem Verzeichnis.
Rufe owbrc mit dem Namen des Programms (ohne Erweiterung) auf:
owbrc myprog
Wenn du einen anderen seriellen Port als COM2 benutzt, gib die Portnummer als zweiten Parameter an (0 = COM1, 1 = COM2, ...)
owbrc myprog 0
owbrc sendet alle notwendigen Teile an den PV und OWBasic übersetzt sie. Nach erfolgreicher Übersetzung wird das Programm gestartet. Fehlermeldungen werden auf dem PC von owbrc ausgegeben.
Mit emacs als Editor/Entwicklungsumgebung kann man owbrc direkt aufrufen (META-x compile) und direkt zu aufgetretenen Fehlern springen (META-x next-error).
Dies funktioniert auf PCs mit Linux und Windows.

Eine sehr komfortable Alternative zu emacs unter Windows stellt der Editor ConText (www.context.cx) dar, mit dem sich ebenfalls OWBasic-Programme vom PC aus entwickeln und debuggen lassen, zudem ist es möglich, daß der Editor mittels einer speziell angepaßten Syntax-Datei Schlüsselwörter, Prozeduren und Funktionen farblich hervorhebt. Weitere Informationen und die Syntax-Datei von Joachim Kromm können unter www.wie-geht.de/viewtopic.php?id=223 gefunden werden.

REMOVESHARED

Löschen eines Datensatzes der gemeinsamen Datei
REMOVESHARED recordname$[,application$]
Löscht den Datensatz mit dem Namen recordname in der in der gemeinsamen Datei .

RESIZEICON

Größe eines Icons ändern
RESIZEICON OLD%[], NEW%[], sx%, sy%, [x1%, y1%, x2%, y2%]
Erzeugt ein neues Icon NEW der Größe sx*sy durch Größenänderung des Icons OLD. Wird ein Fenster von (x1,y1)(linke obere Ecke) bis (x2,y2) (rechte untere Ecke) angegeben, so wird nur dieser Teil des Icons old neu skaliert.
old und new dürfen nicht dasselbe Feld sein.

RESTART

Neustart von OWBasic
RESTART status%
Durch diese Prozedur wird OWBasic neugestartet. Durch den Parameter status kann das Verhalten angepaßt werden:

status	Bedeutung
0	Normal
1	Remote Mode
2	Laufzeitfehler suchen
4	Autorun-Programm nicht starten
5	Programm in den Cache schreiben

RETURN

Ende der Subroutine
RETURN
Beendet die Abarbeitung eines durch GOSUB angesprungenen Unterprogramms.
Unterprogramme (Subroutinen) dürfen nie durch den normalen Programmablauf erreicht werden, bzw. durch GOTO angesprungen werden.
Eine im Hauptprogramm erreichte RETURN-Anweisung führt zu einem Laufzeitfehler. Stehen Subroutinen am Ende eines Programmes, so sollte das Hauptprogramm vorher mit END abgeschlossen werden.
Stehen Subroutinen vor dem Hauptprogramm, so ist ein Sprungbefehl nötig, um diese zu umgehen:

```
GOTO MAIN: !Sprung zum Hauptprogramm
! es folgt eine Subroutine
$SUB
 PRINT "Unterprogramm"
RETURN
```

```
! das Hauptprogramm
$MAIN
...
```

RIGHT$

Rechter Teilstring
RIGHT$(s$, i%)
Ein String der Länge i, rechtsbündig mit s$ gefüllt:

```
RIGHT$("Test",2)"st"
RIGHT$("Test",6)" Test"
```

LEFT$ - Linker Teilstring
MID$ - Teilstring

ROUND

Für die Umwandlung von gebrochenen Zahlen vom Typ Float in ganze Zahlen vom Typ Integer gibt es verschiedene Funktionen, die sich in der Behandlung des gebrochenen Anteils unterscheiden.

INT(); FLOOR()	Abrunden auf die nächst niedrigere ganze Zahl
CEIL()	Aufrunden auf die nächst höhere ganze Zahl
ROUND()	Runden auf die nächstgelegene ganze Zahl

Die Gleitkomma-Versionen CEIL#, FLOOR# und ROUND# berechnen den ganzzahligen Wert, geben ihn aber als Gleitkommawert zurück. Dies ist nützlich, falls die Integerwerte den Zahlen-Bereich des Types Integer überschreiten würden.

RUNTIMEERROR

Laufzeit-Fehler
Laufzeitfehler sind Fehler, die erst zur Laufzeit des Programmes festgestellt werden. Das können Fehler wie "Division durch Null", negative Operanden beim Wurzelziehen oder generell falsche Parameter bei Prozeduren und Funktionen sein. Normalerweise wird dieser Fehler unter Angabe der Prozedur, bei der dies aufgetreten ist, gemeldet und das Programm abgebrochen. Die Fehlermeldung enthält auch den aktuellen Wert des Programmzählers. Mit diesem Wert kann man den Fehler im Quelltext lokalisieren. Die Funktion "Find runtime error" im Options-Menü fragt zunächst nach dem Programm-Zählerwert. Dann muß das Programm gewählt werden. Die folgende Übersetzung des Programms stoppt an der Fehlerposition.
Manchmal möchte man die Fehlerbehandlung zur Laufzeit auch selbst vornehmen. Zum Beispiel könnte man solange alle MEMOs lesen, bis ein Fehler auftritt, weil kein weiteres mehr da ist. Hier ist der Fehler nur eine Meldung, daß man das letzte MEMO erreicht hat. Dazu ist es möglich, mit ERROROFF die normale Fehlerbehandlung auszuschalten. Fehler führen

nicht mehr zu einer Fehlermeldung und Abbruch, sondern es wird die Integer-Variable ERROR% belegt. Nunmehr kann man selbst abfragen, ob ein Fehler aufgetreten ist. Weil jedoch auch unerwartete Fehler nun nicht mehr gemeldet werden, empfiehlt es sich, die normale Fehlerbehandlung mit ERRORON sofort wieder einzuschalten. ERRORON und ERROROFF wirken akkumulativ, damit das Prinzip auch funktioniert, wenn solche Prozeduren verschachtelt werden. Wird die Fehlerbehandlung 2 mal ausgeschaltet, so muß sie auch zweimal wieder eingeschaltet werden. Es ist deshalb streng auf die Paarung von ERRORON und ERROROFF zu achten.
Während die normale Fehlerbehandlung ausgeschaltet ist, muß nach Auswertung von ERROR% diese Variable wieder auf 0 gesetzt werden.
Anwenderprozeduren können mittels ERROR Fehler auslösen.

SAVEFILE

Datei schreiben
SAVEFILE mode%,submode%,nr%
Speichern des internen Puffers FILEBUFFER in eine Datei im FlashRAM.
Die Datei ist gegeben durch den Modus, den Untermodus (Kategorie) und die laufende Nummer. Die Art der Datei (Text oder binär) wird durch den Modus bestimmt.

SAVEMEMO

Schreiben eines MEMO
SAVEMEMO ARRAY$[], submode%, nr% [,lnr%]
SAVEMEMOFP ARRAY$[], submode%, FP% [,lnr%]
Speichert den Inhalt des Stringarrays ARRAY in einem Memo im Bereich submode.
Das MEMO wird selektiert über die Nummer des Memos oder einen filepointer.
Ist der Parameter nr oder der Filepointer FP gleich -1, so wird das MEMO neu angelegt und steht dann als erstes MEMO in der Kategorie.
Ist nr eine nichtnegative Nummer oder FP ein gültiger Zeiger auf ein Memo, so wird das entsprechende MEMO überschrieben. Sollen nicht alle Feldelemente von array geschrieben werden, so ist die Zeilenzahl als Parameter lnr anzugeben.

SAVEQM

Speichern eines Quickmemos
SAVEQM BM%[], cat%, nr%
SAVEQMFP BM%[], cat%, FP%
Das Integer-Feld BM muß ein gültiges ICON enthalten. Die Größe dieses Icons muß kleiner oder gleich der Quickmemo-Größe von 156x130 Pixeln sein. Das Icon wird als Quickmemo in der Kategorie cat gespeichert.
Das zu schreibende Quickmemo kann durch seine Nummer nr oder einen Dateizeiger fp ausgewählt werden. Ist die Nummer nr oder der Dateizeiger FP gleich -1, so wird ein neues Quickmemo angelegt.

SAVESHAREDFLOAT

Speichern/Laden eines Gleitkomma-Arrays im Shared File

SAVESHAREDFLOAT name$,A#[][, size%][, application$]
LOADSHAREDFLOAT name$,A#[][, size%][, application$]

Die Prozeduren speichern bzw. laden Daten des Rekord name des Shared File in das Gleitkomma-Array A. Wenn der Parameter size angegeben ist, wird nur diese Anzahl Array-Elemente gespeichert/geladen. Der Parameter application gibt den Namen des Programmes an, auf dessen Rekord zugegriffen wird. Standard ist der Name des BASIC-Programmes.

SAVESHAREDINT - Speichern/Laden eines Integer-Arrays im Shared File

SAVESHAREDINT

Speichern/Laden eines Integer-Arrays im Shared File

SAVESHAREDINT name$,A%[][, size%][, application$]
LOADSHAREDINT name$,A%[][, size%][, application$]

Die Prozeduren speichern bzw. laden Daten des Rekord name des Shared File in das Integer-Array A. Wenn der Parameter size angegeben ist, wird nur diese Anzahl Array-Elemente gespeichert/geladen. Der Parameter application gibt den Namen des Programmes an, auf dessen Rekord zugegriffen wird. Standard ist der Name des Basic-Programmes.

SCROLL

Verschieben eines Bildschirmausschnittes

SCROLL x1%, y1%, x2%, y2%, diff%

Der Befehl verschiebt den Inhalt des Rechtecks mit der linken oberen Ecke (x1,y1) und der rechten unteren Ecke (x2,y2) um diff Pixel nach unten. Mit negativen Werten für diff kann der Bereich nach oben verschoben werden. Der freiwerdende Bereich wird weiß gefüllt.

SERIAL

Kommunikation über die serielle Schnittstelle

SRLOPEN - Öffnen der seriellen Schnittstelle
SRLSTAT, SRLRBUF, SRLTBUF - Status der seriellen Schnittstelle
SRLRBYTE, SRLSBYTE - Empfangen und Senden von Daten
SRLCLOSE - Schließen der seriellen Schnittstelle

SETBIT

Setzen eines Bit

SETBIT BM%[], nr%, value%

Die Prozedur setzt das nr . Bit des Integerfeldes BM auf den Wert value. Gültige Werte sind 0 und 1.

SETBYTE

Setzen eines Bytes

```
SETBYTE BM%[],nr%,value%
```

Die Prozedur setzt das nr . Byte des Integerfeldes BM auf den Wert value. Gültig für value sind Werte von 0 bis 255.

SETI

Setzen des Bereiches eines Arrays

```
SETI ARRAY%[], first%, last%[, val%=0]
SETF ARRAY#[], first%, last%[, val#=0.0]
SETS ARRAY$[], first%, last%[, val$=""]
SETB ARRAY![], first%, last%[, val!=FALSE]
SETC ARRAY&[], first%, last%[, val&=0]
```

Diese Befehle setzen alle Felder vom Index first bis last des Arrays ARRAY auf den Wert val .

SETFFORMAT

Format für Gleitkommazahlen festlegen

```
SETFFORMAT width, format, prec
SETFFORMAT width, format
SETFFORMAT width
SETFFORMAT
```

Die Prozedur steuert, wie Gleitkomma-Werte in Strings konvertiert werden. Dies beeinflußt die explizite Konvertierung mittel STRING und die implizite Konvertierung bei Ausgabe mittels PRINT. Der Parameter width gibt die Breite des zu verwendenden Zeichenfeldes an (Zahl der Zeichen). Für format können 3 verschiedene Werte benutzt werden:

1 Dynamische Auswahl
2 Gleitkomma-Darstellung
3 Exponential-Darstellung

Der Parameter prec steuert die Genauigkeit (Stellenzahl).
Bei Aufruf ohne Parameter wird auf den Standard zurückgeschaltet.

FORMAT	V1	V2	V3
SETFFORMAT	3.14159	314159	3.14159e+17
SETFFORMAT 15,1,4	3.142	3142e-05	3.142e+17
SETFFORMAT 15,2,4	3.1416	314159.2813	31415926551635600.0000
SETFFORMAT 15,3,4	3.1416e+00	3.1416e+05	3.1416e+17

SETICONPIX

Setze ein Pixel im Icon

SETICONPIX A%[], x%, y% [, v%=1]

Setzt das Pixel mit den Koordinaten x und y im Icon A.
Die Größe des Icons muß unbedingt vorher festgelegt werden!

SETTIME

Setzen der Systemzeit

SETTIME i%, val%

Setzt die Komponente i der Systemzeit auf den Wert val. i kann die folgenden Werte haben:

i	Aktion
1	setzt die Sekunden auf 0
2	setzt die Minute auf val
3	setzt die Stunde auf val
4	setzt den Tag auf val
5	setzt den Monat auf val
6	setzt das Jahr auf val

SGN

Vorzeichen eines Integerwertes

SGN%(val%)

Gibt das Vorzeichen des Integerwertes val zurück:

val	Rückgabewert
<0	-1
0	0
>0	1

SHAREDFILE

Shared File

Weil die Zahl der Addin-Dateien auf dem PV begrenzt ist, ist es empfehlenswert, Daten verschiedener Anwendungen in einer gemeinsamen Datei SHARED zu speichern. Jeder Datensatz der Datei enthält eine Signatur/Namen, der es erlaubt, daß jede Anwendung ihre Daten wiederfindet. Die Signatur enthält dazu den Namen der Anwendung und einen für den Datensatz spezifischen
Namen.
Der Shared File Standard wurde durch Anton Poluektov und Wolfgang Ortmann vorgeschlagen.
Bei den Prozeduren für die Arbeit mit Rekords des Shared File wird der Name des Basic-Programmes als Anwendungsname verwendet, wenn kein anderer Anwendungsname angegeben wird.
Prozeduren zur Arbeit mit Rekords des Shared File:

LOADSHARED, SAVESHARED - Prozeduren, die mit dem Filebuffer arbeiten

REMOVESHARED - Einen Rekord löschen
SAVESHAREDINT, LOADSHAREDINT - Speicherung von Integer-Arrays
SAVESHAREDFLOAT, LOADSHAREDFLOAT - Speicherung von Gleitkomma-Arrays

SHL

Bitverschiebung nach links
SHL(i%[, n%=1])
Alle Bits der Zahl i werden um n Positionen nach links verschoben.

SHOW

Darstellung aktualisieren
SHOW [x1%, y1%, x2%, y2%]
Erzwingt die neue Darstellung einer geänderten Grafik.
Dies ist um gezeichnete grafischen Darstellungen sichtbar zu machen. Nur bei einigen grafischen Darstellungen (z.B. MESSAGEBOX) geschieht dies automatisch.
Bei Aufruf mit 4 Parametern wird nur das Fenster von (x1,y1) (linke obere Ecke) bis (x2,y2) (rechte untere Ecke) aktualisiert.

SHR

Bitverschiebung nach rechts
SHR%(i%[, n%=1])
Alle Bits der Zahl i werden um n Positionen nach rechts verschoben.

SIN#

Winkelfunktionen
SIN#(phi#)
COS#(phi#)
TAN#(phi#)
Sinus, Cosinus bzw. Tangens des Arguments phi. Phi ist ein Winkel im aktuellen Modus.
SIN - Winkelfunktionen
ASIN# - Umkehrung der Winkelfunktionen
ACOS# - Umkehrung der Winkelfunktionen
ATAN# - Umkehrung der Winkelfunktionen

SOUND

Ton-Ausgabe
SOUND freq%, dur%, vol%
Es wird ein Ton mit der Frequenz freq erzeugt (willkürliche Einheiten). Die Länge des Tones ist dur (in Millisekunden). Die Lautstärke und der Klang kann mit Werten von vol zwischen 1 und 100 von leise bis laut gesteuert werden.
Für eine Pause zwischen den Tönen ist SOUND mit freq=0 aufzurufen.

SPLIT

Zerlegung eines String mit Trennzeichen
SPLIT S1$, s2$[, S3$]
s2 wird als Trennstring aufgefaßt, und der Teil des Strings S1 vor dem Trennstring wird auf S3 abgelegt. Auf S1 verbleibt der Rest des urprünglichen Strings nach dem Trennstring. Ist der Trennstring nicht in S1 enthalten, wird der ganze String S1 der Variablen S3 zugewiesen und S1 ist leer.
Wird S3 weggelassen, wird der abgespaltete Teil nicht gespeichert.

SPLITN

Abspalten eine Anzahl von Zeichen von einem String
SPLITN S1$, nr%[, S2$]
Von dem String S1 werden nr Zeichen abgespalten und als String S2 abgespeichert. Auf S1 verbleibt der Rest des urprünglichen Strings. Wird S2 weggelassen, wird der abgespaltete Teil nicht gespeichert.

SQR#

Quadratwurzel
SQR#(f#)
Die Quadratwurzel von f.

SQR

Quadratwurzel
SQR(i%)
Die Quadratwurzel von i.

SQUARE

Das Quadrat einer Zahl
SQUARE(i%)
Das Quadrat von i.

SRLCLOSE

Schließen der seriellen Schnittstelle
SRLCLOSE
Die serielle Schnittstelle wird geschlossen.

SRLOPEN

Serielle Schnittstelle öffnen

SRLOPEN [rate%=384, bits%=8, par%=0, stop%=1 ,fctrl%=2, port=1]
Die serielle Schnittstelle wird für Lesen oder Schreiben geöffnet. Dabei werden die übergebenen Kommunikationsparameter eingestellt:

rate
Die Datenrate der seriellen Schnittstelle wird auf 100*rate Bit pro Sekunde eingestellt. Zulässig sind folgende Datenraten: 300bps, 600bps, 1200bps, 2400bps, 4800bps, 9600bps, 19200bps, 38400bps, 57600bps und 115200bps

bits
Die Zahl der zu übertragenden Bits pro Zeichen wird durch bits (7 oder 8) festgelegt.

par
Der Parameter bestimmt die zu übertragende Parität:

par	Parität
0	keine
1	ungerade
2	gerade

stop
Die Zahl der Stopbits nach jedem Zeichen (1 oder 2).

fctrl
Die Datenfluß-Steuerung erfolgt durch:

fctrl	Datenfluß-Steuerung
0	Keine
1	Software XON/XOFF
2	Hardware RS/CS
3	Hard- und Software

port
Die benutzte Schnittstelle

port	Schnittstelle
1	Serielle Schnittstelle
2	USB (wenn vorhanden)

Die Standardwerte entsprechen einer Datenrate von 38400 bps und dem üblichen Protokoll 8N1 bei Hardware-Datenflußsteuerung.

SRLRBUF

Status des Empfangspuffers der seriellen Schnittstelle
SRLRBUF%()
Die Funktion gibt die Zahl der Zeichen im Empfangs-Puffer zurück. Ein Wert > 0 bedeutet, daß mit SRLRBYTE ein Wert empfangen werden kann.

SRLRBYTE

Empfang eines Zeichens von der seriellen Schnittstelle
SRLRBYTE%()
Empfang eines Zeichens von der seriellen Schnittstelle. Ist noch kein Zeichen empfangen worden, so wartet diese Funktion und das Programm blockiert. Soll das vermieden werden, so muß vorher mit SRLRBUF abgefragt werden, ob ein Zeichen vorliegt.

SRLSBYTE

Senden eines Zeichens über die seriellen Schnittstelle
SRLSBYTE byte%
Sendet ein Zeichen an die serielle Schnittstelle. Vorher ist zu testen, ob im Sendepuffer noch Platz ist, da es ansonsten zu einer Fehlermeldung kommt.

SRLSTAT

Status der seriellen Schnittstelle
SRLSTAT%()
Die Funktion gibt den Status der Schnittstelle zurück.
Bedeutung der Bits nach CASIO:

IX_SRL_	Hex-Wert	Binärer Wert	Bedeutung
TXEMP	0x0020	00000000 00100000	sending register empty
FE	0x0008	00000000 00001000	flaming error
PE	0x0004	00000000 00000100	parity error
OE	0x0002	00000000 00000010	overrun error
RXRDY	0x0001	00000000 00000001	receive data ready
CB	0x0200	00000010 00000000	DCE busy
TB	0x0400	00000100 00000000	DTE busy
OV	0x0100	00000001 00000000	receive buffer overflow error

SRLTBUF

Status des Sendepuffers der seriellen Schnittstelle
SRLTBUF%()
Die Funktion gibt die Zahl der freien Plätze im Sende-Puffer zurück. Ein Wert >0 bedeutet, daß mit SRLSBYTE ein Wert gesendet werden kann.

STRING

Umwandlung in einen String
STRING$(i%)
STRING$(f#)
STRING$(b!)

STRING(i%)
STRING(f#)
STRING(b!)
Liefert eine Stringrepräsentation des Wertes i% bzw. f#.

STRINGCONSTANTS

Zeichenketten-Konstanten
Zeichenketten-Konstanten werden in Programmen benutzt, um feste Texte darzustellen. Diese können in Variablen gespeichert werden oder direkt Prozeduren als Parameter übergeben werden.

```
b$="World"
PRINT "Hello, "; b$
```

Zeichenketten-Konstanten werden mit dem Anführungszeichen (") begrenzt. Sie können nicht die Länge einer Zeile überschreiten und damit im Quelltext keinen Zeilenvorschub enthalten. In Zeichenketten-Konstanten repräsentiert mit wenigen Ausnahmen jedes Zeichen sich selbst.

Ausnahmen (Escape-Sequenzen):

Geschriebenes Zeichen	Dargestelltes Zeichen
¶	Zeilenvorschub
\¶	¶
\"	"
\'	'
\\	\
\x??	Zeichen mit dem durch ?? hexadezimal angegebenen ASCII-Code

Zeichen-Konstanten (Typ Character) werden durch einfache Anführungszeichen ' begrenzt. Sie müssen genau ein Zeichen beinhalten. Im Fall der in der Tabelle genannten Sonderzeichen können das zwei geschriebene Zeichen im Programm sein, wie in folgendem Beispiel:

```
nl&='¶':     ! newline character
cc&='\¶':    ! the character ¶
sq&='\'':    ! single quota
sp&='\x20': ! space
```

STRINGHEIGHT

Texthöhe für Darstellung ermitteln
STRINGHEIGHT(s$,font%)
Ermittelt die Höhe (Y-Ausdehnung) des Strings s bei der Darstellung mittels DRAWSTRING mit dem Zeichensatz font.

font	Font
0	Normal
1	Fett
2	FixedPitch
3	Groß
4	Klein
5	Sehr klein

STRINGSIZE

Textlänge für Darstellung ermitteln

STRINGSIZE(s$,font%)

Ermittelt die Breite (X-Ausdehnung) des Strings s bei der Darstellung mittels DRAWSTRING mit dem Font font.

font	Font
0	Normal
1	Fett
2	FixedPitch
3	Groß
4	Klein
5	Sehr klein

Der "kleine" Zeichensatz 4 ist nur für die Zeichen von CHR$(32) bis CHR$(127) mit den CASIO Zeichensätzen kompatibel.
Der "sehr kleine" Zeichensatz 5 ist nur für Zahlen und einige wenige Buchstaben brauchbar.

SYNC

Mit einer internen Uhr synchronisieren

SYNC cycl#

SYNC

Synchronisieren mit einer internen Uhr bedeutet, bestimmte Dinge nur in vordefinierten Zeitabständen zu tun. Eine Anwendung könnte ein Spiel sein, bei dem in festen Zeitabständen ein Bild aufzubauen ist, unabhängig davon, wieviel Zeit die Interaktion mit dem Nutzer oder andere Berechnungen erfordern.
Der Aufruf von SYNC mit einer Zykluszeit cycl als Parameter startet die interne Uhr.
Der Aufruf ohne Parameter wartet, bis der jeweils nächste Zyklus/Takt der internen Uhr abgelaufen ist.

Beispiel:

```
SYNC 1.0
FOR I=0 TO 50
 IF I % 2= 0 THEN
   ? "tick ";
 ELSE
   ? "tack"
 ENDIF
SYNC
NEXT
```

Das Programm gibt sekündlich abwechselnd "tick" und "tack" aus, unabhängig davon, wielange PRINT benötigt - kürzer für "tick", länger für "tack" wegen des Zeilenvorschubes.
TIMER und SYNC benutzen den gleichen internen Timer. Der initiale Aufruf von SYNC setzt den Timer zurück. Während der Benutzung dürfen STARTTIMER und STOPTIMER nicht benutzt werden, da sie den Timer anhalten oder zurücksetzen.

SYS

Aufruf spezieller System-Funktionen

SYS wert%

Spezielle Prozedur zum Aufruf von System-Funktionen.

Wert	Funktion
0	Keine Abfrage auf Programmunterbrechung und Belegung der Touch-Variablen
1	Abfrage auf Programmunterbrechung und Belegung der Touch-Variablen
2	Erzwungene sofortige Touchscreen-Abfrage
3	Anzeige aller gespeicherten String
4	Anzeige des freien Speichers für Strings
5	Erzwungene GarbageCollection
6	Normale Laufzeit-Fehlermeldungen mit Programmabbruch
7	Keine Laufzeit-Fehlermeldungen
10	Starte Kurzzeit-Timer (Fasttimer)
11	Stoppe Kurzzeit-Timer (Fasttimer)
12	Die Buttons "Light" und "Off" werden von OWBasic abgefragt (Default)
13	"Light" und "Off" werden von OWBasic nicht abgefragt
14	AutoPowerOff (Ausschalten des PV nach einer gegebenen Zeit ohne Betätigung des Touch-Screen) aktivieren (Default)
15	AutoPowerOff deaktivieren

SYSDLG

Aufruf eines System-Dialoges

SYSDLG n%

Ruft einen durch n spezifizierten Systemdialog auf. n kann dabei folgende Werte haben:

n	Dialog
0	PopUp-Tools
1	Calculator
2	Speicherverwaltungs-Dialog
3	Datums-/Zeit-Dialog
4	Ton-Dialog
5	Format-Dialog
6	Kapazität
7	Kontrast-Dialog
8	Touch-Panel-Kalibrierung
9	Add-In-Download-Modus
10	PC-Sync-Modus (kehrt nicht zurück)

SYSLANG

Systemsprache

SYSLANG%()

Gibt einen Wert zurück, der die Systemsprache repräsentiert. Die Zuordnung kann der folgenden Tabelle entnommen werden.

Rückgabewert	Sprache
0	Deutsch
1	English
2	Español
3	Français
4	Italiano

SYSTEM

Systemnahe Prozeduren

SYSBASIC - System-Funktionen
VERSION - Versions-Kontrolle
IN, OUT - Port-Ein- und -Ausgaben
PEEK, POKE - Manipulation des Haupt-Speichers
BUZZER - Akustisches Signal
SOUND, PLAY - Tonausgabe
BATTSTAT - Batteriezustand
WAIT - Programmausführung verzögern
TIME - Datums- und Zeitabfrage
FASTTIMER - Schneller Kurzzeit-Timer
LIGHT - Schalte Hintergrund-Beleuchtung

Systeminformationen

Die Integer-Variable OWBVERSION enthält die aktuelle Version, zum Beispiel 310 für die Version 3.10. Die Integer-Variable PVMODEL enthält das PV-Modell, auf dem das Programm gerade läuft. Der Wert ist zur Zeit 450 für alle Modelle vor dem PV1600.

TCHINSIDE!

Test, ob Touch-Koordinaten in einem Fenster liegen
`TCHINSIDE!(x1%,y1%,x2%,y2%)`
Testet, ob TCHX und TCHY im Bereich des gegebenen Koordinatenfensters liegen.
Diese Funktion ist äquivalent zu INSIDE!(TCHX, TCHY, x1, y1, x2, y2) .

TCHSET

Setzen des Touchscreen-Status
`TCHSET status!`
Setzt den Status des Touchscreen auf berührt (TRUE) bzw. nicht berührt (FALSE).

TEXT

Standard-Text-Ein- und Ausgabe

Die Standard-Text-Ein- und -Ausgabe erfolgen in der Art eines Terminals. Wenn das Bildschirm-Ende erreicht ist, rollt der Text nach oben. Eine Mischung mit graphischen Ausgaben ist meist nicht zu empfehlen.
Folgende Prozeduren sind verfügbar:

- PRINT - Ausgabe
- INPUT - Eingabe
- CLS - Bildschirm löschen
- LOCATE - Kursor setzen
- INFLOCATE - Kursorposition abfragen

TEXTBOX

Textbox

TEXTBOX s$, x1%, y1%, x2%, x2% [,font%]

Der String s wird in die Mitte der Box (Rechteck) von (x1,y1) bis (x2,y2) gezeichnet. Ist der Font nicht angegeben, wird er in Abhängigkeit von der Box-Größe gewählt.

TIME

Zeit und Datum

TIME(i%)

Bestimmung der aktuellen Zeit und Datum:

i	Rückgabewert
0	Fortlaufender Sekundenwert, wiederholt sich alle 4 Stunden
1	Sekunden
2	Minuten
3	Stunden
4	Tag
5	Monat
6	Jahr
7	Wochentag
8	Wert des Fasttimer

TIMER

Timer

STARTTIMER
STOPTIMER
TIMER%()
TIMER#()

Der Timer mißt die Zeit in Sekunden, wobei auch Bruchteile von Sekunden möglich sind.
STARTTIMER startet diesen Timer und setzt ihn auf Null. STOPTIMER stoppt den Timer.
TIMER#() fragt die aktuelle Zeit ab. Es handelt sich um einen

Gleitkommawert, der die Sekunden seit dem Start des Timers angibt. Der Wert wird mit der jeweils möglichen Genauigkeit zurückgegeben.
TIMER%() fragt die aktuelle Zeit in 1/40-Sekunden ab und gibt einen Integerwert zurück.
Probleme bei der Verwendung in Zusammenhang mit SYNC siehe dort.

TOUCH!

Status des Touchscreen

TOUCH!()
TOUCH%()
TOUCH!(x1%, y1, x2%%, y2%)
TOUCH%(x1%, y1, x2%%, y2%)

Die logische Funktion TOUCH!() liefert TRUE, wenn der Touchscreen zur Zeit berührt wird, sonst FALSE.
Die Integer-Funktion (nicht mehr empfohlen) liefert einen Wert größer Null, wenn der Touchscreen zur Zeit berührt wird, sonst Null.
Wird eine Berührung gemeldet, sind in den Integer-Variablen TCHX und TCHY die Koordinaten der Berührung abgelegt.
Wenn x1, y1, x2 und y2 angegeben sind, wird zusätzlich geprüft, ob die Berührung in diesem Koordinatenfenster stattgefunden hat.

TOUCHED!

Test auf Berührung einer definierten Fläche

TOUCHED!()
TOUCHED%()
TOUCHED!(x1%,y1%,x2%,y2%)
TOUCHED%(x1%,y1%,x2%,y2%)

Die logische Funktion TOUCHED!() funktioniert analog NEWTOUCH!, aber nur für eine Berührung im Rechteck mit der linken oberen Ecke (x1,y1) und der rechten unteren Ecke (x2,y2).
Die Benutzung der Integer-Funktion TOUCHED%() ist nicht mehr zu empfehlen.

Beispiel:

```
...
BOX 5,5,25,25,4
IF TOUCHED!(5,5,25,25) THEN
  DRAWSTRING "Touched", 5, 55
ENDIF
...
```

TOUCHSCREEN

Touch-Screen-Abfrage

Der Touchscreen kann über die Funktionen TOUCH!, NEWTOUCH! und TOUCHED! und die Integer-Variablen TCHX und TCHY abgefragt werden. Diese werden beim Interpreterlauf durch wiederholte Abfrage des Touchstatus (BIOS-Funktion) belegt. Die Abfrage erfolgt (zeitlich)

unregelmäßig. Häufungen länger dauernder Operationen können gelegentlich die Abfrage verzögern. Mit SYS 2 kann dann die sofortige Abfrage erzwungen werden. Im Normalfall ist das aber nicht nötig. Bei der Touch-Screen-Abfrage wird auch die ESC-"Taste" ausgewertet und - wenn gedrückt - das Basic-Programm abgebrochen. Soll die Abfrage (aus Laufzeitgründen o.ä.) nicht durchgeführt werden, so kann sie mit SYS 0 ausgeschaltet werden, und mit SYS 1 wieder ein.
Ein einfaches Warten auf irgendeine Berührung kann mit WAITTOUCH erfolgen.

TRIGMODE

Winkelfunktions-Argumente
TRIGMODE i%
Die Argumente (Winkel) der Winkelfunktionen werden wahlweise im Bogenmaß, Gradmaß oder in Neugrad angegeben. Standard ist die Angabe in Bogenmaß.
Die Umschaltung erfolgt mit der Prozedur TRIGMODE.

i		
1	Bogenmaß	rad
2	Grad	deg
3	Neugrad	grad

TRIM$

Entfernen führender und nachfolgender Leerzeichen
TRIM$(s$)
Die Funktion entfernt von der Zeichenkette s die Leerzeichen am Anfang und am Ende und gibt das Ergebnis als Funktionswert zurück.

USERFUNC

Definition einer Anwender-Funktion
In OWBasic ist es möglich, Anwender-Funktionen zu definieren. Anwender-Funktionen und Prozeduren müssen am Anfang des Programmes definiert werden. Das Hauptprogramm folgt darauf.
Eine Funktionsdefinition beginnt mit dem Funktions-Kopf und endet mit der RETURN Anweisung.

```
FUNC test A, B#, VAR C
 <statements>
RETURN <value>
```

Der Funktions-Kopf beginnt mit dem Schlüsselwort FUNC gefolgt vom Namen der Funktion und der Parameterliste. Der Typ der Funktion wird wie bei Variablen durch ein Suffix bestimmt, wenn der Typ vom Standardtyp abweicht.
Die Parameterliste beschreibt den Typ der Parameter der Funktion und gibt ihnen einen formalen Namen.

Variablen, die innerhalb der Funktion eingeführt werden, sind lokal, das heißt, sie sind außerhalb der Funktion unbekannt. Funktionen berechnen einen Wert und geben ihn an das aufrufende Programm zurück. Der Rückgabewert wird bei der RETURN-Anweisung festgelgt, welche die Funtkionsdefinition abschließt.
Die Verwendung von Anwender-Funktionen ist äquivalent der Anwendung der eingebautenm Funktionen. Die Anwenderfunktion wird auch dann verwendet, wenn eine gleichnamige eingebaut Funktion existiert.

Beispiel:

```
FUNC vlen x,y: ! length of 2d vector
RETURN sqr(x*x+y*y)

x=5: y=7: ! These variables do not have to do anything
          ! with the function parameters.
print vlen(x-5,y-2): ! Example-Call
```

USERPROC

Definition von Anwenderprozeduren

In OWBasic ist es möglich, Anwenderprozeduren zu definieren. Anwenderprozeduren müssen immer am Anfang des Programmes stehen. Die Abarbeitung des Hauptprogrammes beginnt immer nach der letzten Prozedurdefinition. Eine Prozedur-Definition beginnt mit dem Prozedurkopf und endet mit dem Schlüsselwort ENDP.

```
PROC test A, B#, VAR C
<Anweisungen>
ENDP
```

Der Prozedurkopf beginnt mit dem Schlüsselwort PROC, dem Namen der Prozedur und der Parameterliste.
Die Parameterliste beschreibt die beim Aufruf zu übergebenden Parameter und gibt ihnen einen formalen Namen. Dieser Name wird innerhalb der Prozedur für den Parameter verwendet. Der Typ wird analog zu Variablen aus dem Suffix ermittelt. Ohne Suffix findet der Standardtyp Anwendung.
Ohne weitere Kennzeichnung sind die Parameter Werte-Parameter (im Beispiel die Parameter A und B). Das heißt, daß beim Aufruf Werte bereitgestellt werden, die in der Prozedur verwendet werden können. Änderungen dieser Werte haben keine Auswirkung auf die Variablen des Hauptprogramms.
Variablen-Parameter werden mit dem Schlüsselwort VAR gekennzeichnet (Parameter C im Beispiel). Hier wird beim Aufruf eine Variable des rufenden Programmes übergeben. Änderungen an diesem Parameter ändern hier wirklich die übergebene Variable.
Innerhalb der Prozedur verwendete Variablen sind lokal, das heißt, außerhalb der Prozedur sind sie nicht vorhanden.
Der Aufruf von Anwenderprozeduren erfolgt wie bei normalen Prozeduren über den Namen. Auch die Parameter werden analog zu anderen Prozeduren übergeben.

Beispiel 1: Werteparameter

```
PROC printxy x,y: ! Prozedur zur Ausgabe von Koordinaten
 PRINT "("; x; ","; y; ")"; : ! Formatierte Ausgabe
```

```
ENDP
x=5: y=7: !
printxy x-4,y+6: ! Beispiel-Aufruf
```

Beispiel 2: Variablen-Parameter

```
PROC myinc VAR n: ! Prozedur analog zum eingebauten INC
 n+1
ENDP

i=4
myinc i : ! Beispiel-Aufruf
PRINT i : ! es wird 5 ausgegeben
```

Besonderer Beachtung bedürfen Felder als Prozedur-Parameter.

VARDEF

Test, ob Variable definiert ist

VARDEF!(<variable>)

Die konstante Funktion VARDEF gibt TRUE zurück, wenn <variable> definiert ist, ansonsten FALSE.

VARIABLES

Datentypen und Variablen

Es gibt in OWBasic Variablen der Typen Integer, Float, Bool, Char und String. Ohne weitere Kennung ist der Standardtyp Integer. Dies kann mit dem Schlüsselwort DEFAULT im Programm geändert werden.
Der Typ von Variablen kann durch ein Suffix gekennzeichnet werden. Dadurch können vom Defaulttyp abweichende Variablen angelegt werden. Anstelle der eingebauten können auch mittels ENUM oder FPTR selbst definierte Typen verwendet werden; sie werden durch den Operator '\' und den Namen des Typs angegeben.
Ist eine Variable einmal verwendet worden, so wird bei nachfolgender Verwendung ohne explizite Typkennzeichnung der als erstes verwendete Typ für diese Variable verwendet.
Achtung: Diese Regelung wurde geschaffen, um die Schreibarbeit abzukürzen. Typkennzeichen bei Variablen sind dadurch überflüssig, nachdem die Variable einmal verwendet wurde.
Andererseits ist dies bei unvorsichtiger Verwendung fehleranfällig. Wird zum Beispiel von der Annahme ausgegangen, daß der Standardtyp Anwendung findet, aber ist durch vorherige Verwendung ein anderer Typ festgelegt, ensteht ein schwer zu diagnostizierender Fehler.
Noch undurchsichtiger wird die Situation, wenn zwei Variablen gleichen Namens, aber verschiedenen Typs existieren. Hier ist der Typ der zuerst verwendeten Variable ausschlaggebend.

Beispiel:

```
 alpha=4
 beta#=5
 alpha#=3.14
 PRINT alpha :!alpha ist die Integervariable der Zeile 1
```

```
PRINT alpha# :!alpha# ist die Float-Variable der Zeile 3
PRINT beta :!beta ist die Float-Variable der Zeile 2
```

VERSION

Version setzen

VERSION const%

Setzt grundlegende System-Eigenschaften auf den Stand der durch const angegebenen Version zurück. const muß dazu in der Hunderter-Stelle die Haupt-Versionsnummer, in den folgenden zwei Ziffern die Unterversionsnummer enthalten, z.B. 110 für die Version 1.10.
Ist die aktuelle Version kleiner als const, wird ein entsprechender Fehler gemeldet.
Sinnvolle Werte:

Wert	Auswirkungen
100	Zuordnung der Suffixe '#' und '%' vertauscht gegenüber der jetzigen Version:'#' - Integer, '%' - Float
<250	Konvertierungen liefern keinen Fehler, wenn das Argument bereits vom gewünschten Typ ist.

WAIT

Pause in der Programm-Abarbeitung

WAIT time%

Warten/Unterbrechen der Programm-Abarbeitung für time Sekunden. Die wirkliche Zeit kann bis zu knapp einer Sekunde länger sein.

WAITTOUCH

Warten auf eine Berührung des Touchscreen

WAITTOUCH

Die Prozedur wartet bis zur Berührung des Touchscreens.

WHILE

Programm-Schleife

WHILE <boolean expression>
<instructions>
WEND

Die Anweisungen werden in einer Schleife wiederholt durchaufen, solange der logische Wert <boolean expression> erfüllt ist (TRUE).

XCHGS

Austauschen zweier Strings

XCHGS S1$, S2$

Die Prozedur vertauscht die Strings in S1 und S2.
Da die Strings in OWBasic dynamisch verwaltet werden, beschränkt sich die Tätigkeit der Prozedur auf einen einfachen Zeigertausch, der um

einiges effizienter ist als die gewöhnliche Verfahrensweise (Dreieckstausch).

ZEICHENSATZ

Zeichensätze für Textdarstellung

Die grafischen Prozeduren zur Textdarstellung nutzen verschiedene Zeichensätze. Diese werden mit einer ganzen Zahl als Parameter spezifiziert.

font	Font
0	Normal
1	Fett
2	FixedPitch
3	Groß
4	Klein
5	Sehr klein

Der "kleine" Zeichensatz 4 ist nur für die Zeichen von CHR$(32) bis CHR$(127) mit den CASIO Zeichensätzen kompatibel. Der "sehr kleine" Font enthält nur die Ziffern und wenige Zeichen ('A','P','M'); Die Größe der Zeichen kann mit den Funktionen STRINGSIZE und STRINGHEIGHT ermittelt werden.