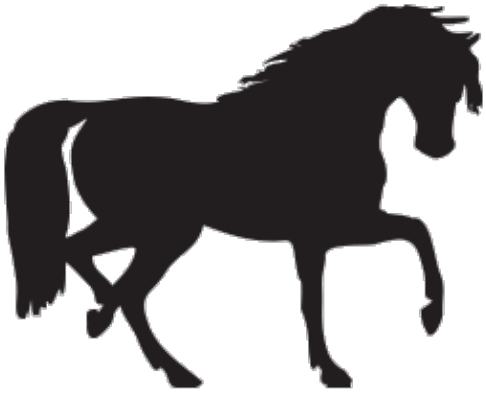

FB-Referenz - Startseite

Inhaltsverzeichnis

Inhaltsverzeichnis

Inhaltsverzeichnis

Inhaltsverzeichnis

Inhaltsverzeichnis

Inhaltsverzeichnis

Inhaltsverzeichnis

Inhaltsverzeichnis

Inhaltsverzeichnis

Inhaltsverzeichnis

Inhaltsverzeichnis

Inhaltsverzeichnis

Inhaltsverzeichnis

Inhaltsverzeichnis

Inhaltsverzeichnis

Inhaltsverzeichnis

Inhaltsverzeichnis

Inhaltsverzeichnis

Inhaltsverzeichnis

Inhaltsverzeichnis

Inhaltsverzeichnis

Inhaltsverzeichnis

Inhaltsverzeichnis

Inhaltsverzeichnis

Inhaltsverzeichnis

Über diese Referenz

FreeBASIC-Referenz » **Über diese Referenz**
Dies ist die aktuelle Fassung der deutschsprachigen FreeBASIC-Referenz.

Sie enthält ausführliche Erläuterungen und teilweise Code-Beispiele zu sämtlichen Schlüsselwörtern der Programmiersprache. Thematisch geordnete Befehlslisten erleichtern den Überblick.

Die deutsche Ausgabe der FreeBASIC-Referenz wird gemeinschaftlich von der Redaktion des FreeBASIC-Portals gepflegt. Inhaltlich orientiert sie sich an der jeweils aktuellen "stable"-Version des Compilers, auch wenn zum Teil Hinweise auf zukünftige Entwicklungen gegeben werden. Bei der deutschsprachigen Referenz handelt sich nicht um eine direkte Übersetzung des englischen Wikis; sie wird unabhängig davon fortgeschrieben. Den Grundstein für die deutsche Ausgabe der FreeBASIC-Referenz legte einst Stefan "Dusky_Joe" Hartinger.

Aus dem aktuellen Datenbestand der Online-Ausgabe wird automatisch jeden Tag eine Offline-Version zum Herunterladen generiert. Diese wird im CHM-Format bereitgestellt, sodass sie unter Windows komfortabel als Hilfedatei verwendet werden kann. Viewer für das CHM-Format stehen auch für Linux zur Verfügung.

Sollten Ihnen in der Referenz Fehler oder Unklarheiten auffallen, zögern Sie bitte nicht, uns einen Hinweis zu geben - zum Beispiel über das Forum oder per E-Mail.

Das Redaktionsteam

Letzte Bearbeitung des Eintrags am 30.05.12 um 23:29:36
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

ABS

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » A » **ABS**
Syntax: ABS (Zahl)
Typ: Funktion
Kategorie: Mathematik

ABS gibt den Absolutbetrag (den positiven bzw. vorzeichenlosen Wert) der angegebenen Zahl zurück.

'Zahl' ist ein beliebiger numerischer Ausdruck. Variablen, Konstanten, Operatoren und Funktionen sind erlaubt. Der Ausdruck darf von jedem Datentyp außer STRING, ZSTRING oder WSTRING sein.

Der Rückgabewert ist vom selben Typ wie 'Zahl'.

ABS kann mithilfe von OPERATOR überladen werden.

Beispiel:

```
PRINT ABS(-1), ABS(-3.1415), ABS(42)
```

Ausgabe:

```
1      3.1415      42
```

Siehe auch:
SGN, mathematische Funktionen

Letzte Bearbeitung des Eintrags am 16.06.12 um 21:08:11
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

ABSTRACT

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » A » **ABSTRACT**
Syntax:

```
TYPE Typname EXTENDS Elterntyp
  DECLARE ABSTRACT {SUB|FUNCTION|PROPERTY|OPERATOR} ...
END TYPE
```

Typ: Klausel
Kategorie: Klassen

ABSTRACT ist eine spezielle Form von VIRTUAL. Im Gegensatz zu virtuellen Methoden wird bei abstrakten kein Methodenkörper (body) geschrieben, also der eigentliche Code, der die Methode implementiert. Dadurch können sog. Interfaces erstellt werden, die von erbenden Klassen implementiert werden.
Damit eine abstrakte Methode also aufgerufen werden kann, muss sie durch die erbende Klasse überschrieben und implementiert werden, da es sonst zu einem Programmabsturz kommt.

Konstruktoren können nicht abstrakt sein, da sie auch nicht virtuell sein können. Siehe dazu VIRTUAL.
Weiterhin können auch Destruktoren nicht abstrakt sein, da es stets möglich sein muss, den Speicher einer Variablen freizugeben.

Beachte:
In einer mehrstufigen Vererbungshierarchie kann eine überschriebene Methode auf jeder Ebene als abstrakt, virtuell oder normal deklariert werden. Werden die Varianten gemischt, gilt folgende Reihenfolge von oben nach unten in der Hierarchie: Abstrakt -> Virtuell -> Normal.

Beispiel:

```
Type Hello Extends Object
  Declare Abstract Sub hi
End Type

Type HelloEnglish Extends Hello
  Declare Sub hi
End Type

Type HelloFrench Extends Hello
  Declare Sub hi
End Type

Type HelloGerman Extends Hello
  Declare Sub hi
End Type


Sub HelloEnglish.hi
  Print "Hello!"
End Sub

Sub HelloFrench.hi
    Print "Salut!"
End Sub
```

```
Sub HelloGerman.hi
  Print "Hallo!"
End Sub


Randomize

Dim As Hello Ptr h

For i As Integer = 0 To 9
  Select Case Int(Rnd*3) + 1
    Case 1
      h = New HelloFrench
    Case 2
      h = New HelloGerman
    Case Else
      h = New HelloEnglish
  End Select

  h->hi( )
  Delete h
Next

Sleep
```

Vergleich zu anderen Sprachen:
Abstrakte Methoden werden in C++ als "pure virtual" bezeichnet. Anders als in FreeBASIC erlaubt es C++, dass "pure virtuals" einen Methodenkörper besitzen.

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC: existiert seit FreeBASIC v0.90.0

Unterschiede unter den FB-Dialektformen: nur in der Dialektform -lang fb verfügbar

Siehe auch:
VIRTUAL, TYPE, EXTENDS, OBJECT, Objektorientierung

Letzte Bearbeitung des Eintrags am 26.06.13 um 20:43:31
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

ACCESS

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » A » **ACCESS**
Syntax: OPEN dateiname FOR dateimodus ACCESS { READ | WRITE | READ WRITE } [...] AS [#]nummer
Typ: Klausel
Kategorie: Dateien

ACCESS wird zusammen mit der OPEN-Anweisung verwendet.

Durch ACCESS werden die Zugriffsrechte auf die Datei festgelegt.

- READ ermöglicht Lesezugriff auf die Datei.
- WRITE ermöglicht Schreibzugriff auf die Datei.
- READ WRITE ermöglicht Lese- und Schreibzugriff auf die Datei.

ACCESS hat nur zusammen mit dem Dateimodus BINARY und RANDOM eine Auswirkung. Wird ACCESS nicht angegeben, dann wird die Datei mit Lese- und Schreibzugriff geöffnet.

Beispiel:

```
OPEN "data.raw" FOR BINARY ACCESS READ AS "reflinkicon"
href="temp0283.html">OPEN (Anweisung), CLOSE, Dateien (Files)
```

Letzte Bearbeitung des Eintrags am 26.12.12 um 23:41:10
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

ACOS

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » A » **ACOS**
Syntax: ACOS (Zahl)
Typ: Funktion
Kategorie: Mathematik

ACOS gibt den Arcuskosinus (oder auch Inverskosinus) einer Zahl zurück. ACOS bildet also die Umkehrfunktion zu COS.

- 'Zahl' ist ein beliebiger numerischer Ausdruck. Variablen, Konstanten, Operatoren und Funktionen sind erlaubt. Der Ausdruck darf von jedem Datentyp außer STRING, ZSTRING oder WSTRING sein. 'Zahl' muss zwischen -1 und 1 liegen. Ist 'Zahl' größer als 1 oder kleiner als -1, wird ein Fehler erzeugt.
- Der Rückgabewert ist ein DOUBLE, der eine Winkelangabe im Bogenmaß darstellt. Er liegt im Bereich von 0 bis Pi.

ACOS kann mithilfe von OPERATOR überladen werden.

Beispiel:

```
Const PI = Acos(0)*2
Dim As Double h, a
Input "Bitte gib die Länge der Hypotenuse ein: ", h
Input "Bitte gib die Länge der Ankathete ein:  ", a
Print ""
Print "Der Winkel zwischen den beiden Seiten beträgt " & Acos(a/h) / PI *
180 & "°"
Sleep
```

Ausgabebeispiel:

```
Bitte gib die Länge der Hypotenuse ein: 7
Bitte gib die Länge der Ankathete ein:  3.5

Der Winkel zwischen den beiden Seiten beträgt 60°
```

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC:
Die Überladung von ACOS für benutzerdefinierte Datentypen ist seit FreeBASIC v0.22 möglich.

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht ACOS nicht zur Verfügung und kann nur über **__ACOS** aufgerufen werden.

Siehe auch:
SIN, ASIN, COS, TAN, ATN, ATAN2, mathematische Funktionen

Letzte Bearbeitung des Eintrags am 24.12.12 um 14:13:36
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

ADD

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » A » **ADD**
Syntax: { PUT | DRAW STRING } [Puffer,] [STEP] (x, y), [weitere Angaben ...], ADD [, Faktor]
Typ: Schlüsselwort
Kategorie: Grafik

ADD ist ein Schlüsselwort, das im Zusammenhang mit PUT (Grafik) und DRAW STRING eingesetzt wird.

ADD bewirkt, dass die Farbnummer des gespeicherten Pixels mit 'Faktor' multipliziert und zur Sättigung des zu überzeichnenden Pixels addiert wird. 'Faktor' ist dabei ein Wert zwischen 0 und 255.

Das Ergebnis der ADD-Methode sind ebenso wie bei ALPHA durchscheinende Bildschirmausschnitte. Der Transparenzgrad des Ausschnitts ist jedoch nicht nur vom angegebenen Faktor abhängig, sondern auch von der Helligkeit des darunter liegenden Pixels. Beim Überzeichnen schwarzer Pixel verhält sich ADD wie ALPHA; mit zunehmender Helligkeit des zu überzeichnenden Pixels allerdings verschiebt sich das Gleichgewicht der Farbmischung hin zur Transparenz des zu zeichnenden Pixels.

Wird 'Faktor' ausgelassen, nimmt FreeBASIC automatisch Faktor = 255 an.

Ebenso wie bei den Methoden TRANS und ALPHA werden Flächen in der Maskenfarbe nicht gezeichnet (siehe dazu auch Interne Pixelformate).

Beispiel: Zeichnen von drei sich überlappenden Kreisen

```
SCREENRES 320, 200, 16

' Sprite mit einem Kreis erzeugen
CONST radius = 32
DIM AS ANY PTR img = IMAGECREATE(radius*2 + 1, radius*2 + 1, 0)
CIRCLE img, (radius, radius), radius, RGB(192, 192, 63), , , 1, f

' Kreis mit drei verschiedenen Faktoren zeichnen
PUT (146 - radius, 108 - radius), img, add, 64
PUT (174 - radius, 108 - radius), img, add, 128
PUT (160 - radius,  84 - radius), img, add, 192

' Speicher freigeben und auf Tastendruck warten
IMAGEDESTROY img
SLEEP
```

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC: existiert seit FreeBASIC v0.17

Siehe auch:
PUT (Grafik), DRAW STRING, SCREENRES, AND (Methode), OR (Methode), XOR (Methode), PSET (Methode), PRESET (Methode), ALPHA, TRANS, CUSTOM, Grafik

Letzte Bearbeitung des Eintrags am 26.08.12 um 13:32:15
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

ALIAS

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » A » **ALIAS**
Syntax: [DECLARE] { SUB | FUNCTION } Procname ALIAS "Aliasname" ...
Typ: Klausel
Kategorie: Bibliotheken

ALIAS gibt einer Prozedur in einer Library einen neuen Namen, mit dem man auf sie verweisen kann.

- 'Aliasname' ist der Name, den die Prozedur in der Library trägt.
- 'SubName ist der Name, den die Prozedur innerhalb des Programms trägt.

'Aliasname' kann im Programm nicht zum Aufruf der Prozedur verwendet werden. Er ist aber für den Linker sichtbar, wenn der Code zusammen mit Code in anderen Sprachen gelinkt wird.

ALIAS wird üblicherweise für Prozeduren verwendet, die in anderen Sprachen geschrieben sind und deren Name in FreeBASIC nicht erlaubt ist. Wird ALIAS mit DECLARE verwendet, dann wird nur 'Aliasname' vom Linker verwendet.

Anders als bei Prozedurnamen wird bei ALIAS nicht die Groß-/Kleinschreibung des Aliasnamens geändert. ALIAS ist daher nützlich, wenn ein externer Code eine spezielle Schreibweise des Namens benötigt.

Beispiel:

```
DECLARE SUB xClearScreen ALIAS "ClearVideoScreen" ()

SUB xClearScreen ALIAS "ClearVideoScreen" ()
  ' Inhalt der Prozedur
END SUB

xClearScreen
```

Unterschiede zu QB:
In QB funktioniert ALIAS nur zusammen mit DECLARE.

Siehe auch:
DECLARE, SUB, FUNCTION, EXPORT, EXTERN (Module), Module (Library / DLL)

Letzte Bearbeitung des Eintrags am 24.05.12 um 11:57:25
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

ALLOCATE

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » A » **ALLOCATE**
Syntax: ALLOCATE (Bytes)
Typ: Funktion
Kategorie: Speicher

ALLOCATE reserviert ("alloziert") eine beliebige Anzahl von Bytes im Speicher (Heap) und liefert einen Pointer zum Anfang dieses Speicherbereichs.

- 'Bytes' ist ein INTEGER-Wert, der angibt, wie viele Bytes reserviert werden sollen.
- Der Rückgabewert ist ein Pointer auf den Anfang des Speicherbereichs. Wenn beim Allozieren ein Fehler auftritt, ist der Rückgabewert 0 (Null-Pointer).
- Die reservierten Bytes werden nicht auf 0 gesetzt, dazu siehe CALLOCATE

Ein mit ALLOCATE reservierter Speicherbereich muss mit DEALLOCATE wieder freigegeben werden.

ALLOCATE ist kein Teil der FreeBASIC Runtime Library, sondern ein Alias von malloc der C-Lib.
Achtung: Es kann nicht garantiert werden, dass diese Funktion auf allen Plattformen Multithreading unterstützt, d.h. thread-safe ist. Unter Windows und Linux sind aktuell durch die verwendeten Implementationen der Betriebssysteme aber keine Probleme zu erwarten.

Beispiel:
Mit ALLOCATE wird ein dynamisches Array erstellt, dessen Elemente mit einer Fibonacci-Sequenz befüllt werden. Beachten Sie den DEALLOCATE-Aufruf am Programmende, der den Speicher wieder freigibt.

```
' Platz für 15 INTEGER-Speicherstellen reservieren
CONST IntegerCount AS INTEGER = 15
DIM AS INTEGER PTR buffer
buffer = ALLOCATE( IntegerCount * SIZEOF( INTEGER ) )

IF( buffer = 0 ) THEN
   ' Reservierung schlug fehl wenn PTR = 0
   PRINT "Fehler: Speicher konnte nicht reserviert werden"
   END -1
END IF

' Fibonacci-Sequenz einleiten
buffer&"hlzahl">0 ] = 0
buffer[ 1 ] = 1

DIM AS INTEGER i
' ..., vervollständigen...
FOR i = 2 TO IntegerCount - 1
   buffer&"hlzeichen">] = buffer[ i - 1 ] + buffer[ i - 2 ]
NEXT
' ... und ausgeben
FOR i = 0 TO IntegerCount - 1
   PRINT buffer&"hlzeichen">] ;
NEXT

' Speicher wieder freigeben
DEALLOCATE( buffer )
```

```
' auf Tastendruck warten und beenden
SLEEP
END 0
```

Ausgabe:

```
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377
```

Achtung: Wenn Sie einen Speicherbereich reservieren, MÜSSEN Sie diesen nach Gebrauch wieder freigeben, da er sonst reserviert bleibt. Sehen Sie sich dieses Beispiel an:

```
SUB AllocateExample2()
   DIM AS INTEGER PTR IntegerPtr = 0

   ' erste Reservierung
   ' Pointer auf neuen Speicherbereich setzen
   IntegerPtr = ALLOCATE( SIZEOF( INTEGER ) )

   ' zweite Reservierung
   ' Pointer auf neuen Bereich verlegen
   IntegerPtr = ALLOCATE( SIZEOF( INTEGER ) )
   DEALLOCATE( IntegerPtr )
END SUB

AllocateExample2()
SLEEP : END 0
```

In diesem Beispiel geht bei jedem Funktionsaufruf Speicherplatz verloren, da der von der ersten Reservierung bereitgestellte Speicherbereich nicht mehr dealloziert wird. Daher sollten Sie, wenn Sie einen größeren Speicherbereich benötigen, die Funktion REALLOCATE verwenden oder zumindest den alten Speicherbereich mit DEALLOCATE freigeben.

Hinweis:
ALLOCATE überschreibt im Gegensatz zu CALLOCATE nicht den bisherigen Inhalt des reservierten Speichers, weswegen zufällige Werte enthalten sein können. In Verbindung mit Strings sollte aufgrund der internen Verarbeitung unbedingt mit CALLOCATE reserviert werden.

Unterschiede zu QB: neu in FreeBASIC

Plattformbedingte Unterschiede:
Es kann nicht garantiert werden, dass die Prozedur auf allen Plattformen thread-safe ist.

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht ALLOCATE nicht zur Verfügung und kann nur über **__ALLOCATE** aufgerufen werden.

Siehe auch:
CALLOCATE, REALLOCATE, DEALLOCATE, Pointer, Speicher

Letzte Bearbeitung des Eintrags am 05.04.14 um 15:52:28
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

ALPHA

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » A » **ALPHA**
Syntax: { PUT | DRAW STRING } [Puffer,] [STEP] (x, y), [weitere Angaben ...], ALPHA [, alphawert]
Typ: Schlüsselwort
Kategorie: Grafik

ALPHA ist ein Schlüsselwort, das im Zusammenhang mit PUT (Grafik) und DRAW STRING eingesetzt wird.

Mit ALPHA lassen sich Transparenz-Effekte erzeugen. Die ALPHA-Methode wird verwendet, um Bildschirmausschnitte 'durchscheinend' auszugeben; das Ergebnis der Bildschirmausgabe ist eine 'Mischfarbe' aus dem Pixel, das überzeichnet wurde, und dem, das im Quellpuffer gespeichert war.

Das Aktionswort ALPHA ist nur in High-/Truecolor-Modi verfügbar, also in Modi ab 15bpp.

'alphawert' stellt den Transparenzgrad des zu zeichnenden Ausschnitts bzw. das Mischungsverhältnis der beiden Farben dar; 255 bedeutet dabei volle Überdeckung, 0 keine Überdeckung. 127 ist der exakte Mittelwert zwischen den beiden Farben. Ebenso wie bei TRANS und ADD werden Pixel in der Maskenfarbe nicht gezeichnet (siehe dazu auch Interne Pixelformate).

In 32bpp-Modi ist es auch zulässig, den Parameter 'alphawert' auszulassen; in diesem Fall benutzt FreeBASIC den Alphawert, der für jedes Pixel einzeln angegeben wurde. Dies ist nur in 32bpp-Modi möglich, da nur hier ein eingebetteter Alphawert für jedes Pixel möglich ist; siehe dazu auch Interne Pixelformate.

Wird 'alphawert' ausgelassen, jedoch ein Modus mit einer Farbtiefe unter 32bpp verwendet, so geht FreeBASIC von alphawert = 255 aus; dies entspricht völliger Überdeckung bzw. dem Aktionswort TRANS.

Beispiel:

```
' 32-bit-Bildschirm erstellen
ScreenRes 320, 200, 32

' schachbrettartigen Hintergrund zeichnen
For y As Integer = 0 To 199
    For x As Integer = 0 To 319
        PSet (x, y), IIf((x Shr 2 Xor y Shr 2) And 1, RGB(160, 160, 160),
RGB(128, 128, 128))
    Next x
Next y

' Sprite erstellen
Dim img As Any Ptr = ImageCreate(32, 32, RGBA(0, 0, 0, 0))
For y As Single = -15.5 To 15.5
    For x As Single = -15.5 To 15.5
        Dim As Integer r, g, b, a
        If y <= 0 Then
            If x <= 0 Then
                r = 255: g = 0: b = 0    ' rot
            Else
                r = 0: g = 0: b = 255    ' blau
            End If
        Else
            If x <= 0 Then
```

```
                r = 0: g = 255: b = 0   ' gruen
            Else
                r = 255: g = 0: b = 255 ' magenta (transparente
Maskenfarbe)
            End If
        End If
        a = 255 - (x ^ 2 + y ^ 2)
        If a < 0 Then a = 0: r = 255: g = 0: b = 255
        PSet img, (15.5 + x, 15.5 - y), RGBA(r, g, b, a)
    Next x
Next y

' mit verschiedenen Alphawerten zeichnen; TRANS dient zum Vergleich
Draw String (32, 10), "Single alpha"
Put (80 - 16,  50 - 16), img, Alpha, 64
Put (80 - 16, 100 - 16), img, Alpha, 192
Put (80 - 16, 150 - 16), img, Trans

' mit vollem Alphakanal zeichnen
Draw String (200, 10), "Full alpha"
Put (240 - 16, 100 - 16), img, Alpha

' Speicher freigeben und auf Tastendruck warten
ImageDestroy img
Sleep
```

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC: existiert seit FreeBASIC v0.14

Siehe auch:
PUT (Grafik), DRAW STRING, SCREENRES, AND (Methode), OR (Methode), XOR (Methode), PSET (Methode), PRESET (Methode), ADD, TRANS, CUSTOM, Grafik

Letzte Bearbeitung des Eintrags am 26.08.12 um 13:31:49
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

AND (Methode)

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » A » **AND (Methode)**
Syntax: { PUT | DRAW STRING } [Puffer,] [STEP] (x, y), [weitere Angaben ...], AND
Typ: Schlüsselwort
Kategorie: Grafik

AND ist ein Schlüsselwort, das im Zusammenhang mit PUT (Grafik) und DRAW STRING eingesetzt wird.

Die Farbe des gezeichneten Pixels ist das Ergebnis eines logischen AND des zu zeichnenden Pixels mit dem zu überschreibenden Pixel.

Berechnungsbeispiel: An der Position (100, 100) befindet sich ein Pixel mit dem Farbattribut 172. Dieses soll nach der AND-Methode mit einem Pixel des Farbattributs 47 überzeichnet werden. Das Ergebnis ist ein Pixel des Farbattributs 44. Dies ergibt sich folgendermaßen:

```
Dezimal    Binär
 172       10101100
  47       00101111
-AND---------AND---
  44       00101100
```

Die AND-Methode kann mit allen Farbtiefen angewandt werden, also sowohl in palettenindizierten Modi als auch in High-/Truecolor-Modi. Beachten Sie, dass in palettenindizierten Modi das sichtbare Ergebnis nicht nur von den Farbattributen, sondern auch von den zugeordneten Paletten-Einträgen abhängig ist. Siehe dazu PALETTE und Standardpaletten.

Beispiel:

```
' 32-bit-Bildschirm erstellen
ScreenRes 320, 200, 32
Line (0, 0)-(319, 199), RGB(255, 255, 255), bf

' drei Sprites mit Kreisen in zyan, magenta und gelb zeichnen
Const As Integer radius = 32
Dim As Any Ptr cc, cm, cy
cc = ImageCreate(radius * 2 + 1, radius * 2 + 1, RGBA(255, 255, 255,
255))
cm = ImageCreate(radius * 2 + 1, radius * 2 + 1, RGBA(255, 255, 255,
255))
cy = ImageCreate(radius * 2 + 1, radius * 2 + 1, RGBA(255, 255, 255,
255))
Circle cc, (radius, radius), radius, RGB(0, 255, 255), , , 1, f
Circle cm, (radius, radius), radius, RGB(255, 0, 255), , , 1, f
Circle cy, (radius, radius), radius, RGB(255, 255, 0), , , 1, f

' Kreise überlappend zeichnen
Put (146 - radius, 108 - radius), cc, And
Put (174 - radius, 108 - radius), cm, And
Put (160 - radius,  84 - radius), cy, And

' Speicher freigeben und auf Tastendruck warten
ImageDestroy cc
ImageDestroy cm
```

```
ImageDestroy cy
Sleep
```

Siehe auch:
AND (Operator), PUT (Grafik), DRAW STRING, SCREENRES, ALPHA, OR (Methode), XOR (Methode), PSET (Methode), PRESET (Methode), ADD, TRANS, CUSTOM, Grafik

Letzte Bearbeitung des Eintrags am 26.08.12 um 13:34:32
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

AND (Operator)

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » A » **AND (Operator)**
Syntax A: Ergebnis = Ausdruck1 AND Ausdruck2
Syntax B: Ausdruck1 AND= Ausdruck2
Typ: Operator
Kategorie: Operatoren

AND kann als einfacher (Syntax A) und kombinierter (Syntax B) Operator eingesetzt werden. Syntax B ist eine Kurzform von

```
Ausdruck1 = Ausdruck1 AND Ausdruck2
```

AND vergleicht zwei Werte Bit für Bit und setzt im Ergebnis nur dann ein Bit, wenn die entsprechenden Bits in beiden Ausdrücken gesetzt waren. AND wird in Bedingungen eingesetzt, wenn beide Aussagen erfüllt sein müssen.

AND kann mithilfe von OPERATOR überladen werden.

Beispiel 1: AND in einer IF-THEN-Bedingung:

```
IF (a = 1) AND (b = 7) THEN
PRINT "beides erfüllt"
ELSE
PRINT "Entweder a <> 1 oder b <> 7."
END IF
```

Beispiel 2: Konjunktion zweier Zahlen mit AND:

```
DIM AS INTEGER z1, z2

z1 =  6
z2 = 10

PRINT z1, BIN(z1, 4)
PRINT z2, BIN(z2, 4)
PRINT "---", "----"
PRINT z1 AND z2, BIN(z1 AND z2, 4)
SLEEP
```

Ausgabe:

```
  6           0110
 10           1010
---           ----
  2           0010
```

Beispiel 3: AND als kombinierter Operator

```
DIM AS INTEGER w, bnr

INPUT "Bitte geben Sie eine Zahl ein.", w
PRINT "Bitte geben Sie die Nummer des Bits ein,"
INPUT " auf das geprueft werden soll.", bnr
```

```
w AND= (1 SHL bnr)

IF w THEN
   PRINT "Dieses Bit wurde gesetzt."
ELSE
   PRINT "Dieses Bit wurde nicht gesetzt."
END IF
SLEEP
```

Ausgabebeispiel:

```
Bitte geben Sie eine Zahl ein. 5
Bitte geben Sie die Nummer des Bits ein,
 auf das geprueft werden soll. 2
Dieses Bit wurde gesetzt
```

Unterschiede zu QB:
Kombinierte Operatoren sind neu in FreeBASIC.

Siehe auch:
AND (Methode), NOT, OR (Operator), XOR (Operator), IMP, EQV, ANDALSO, Bit Operatoren / Manipulationen

Letzte Bearbeitung des Eintrags am 26.12.12 um 23:42:18
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

ANDALSO

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » A » **ANDALSO**
Syntax: Ausdruck1 ANDALSO Ausdruck2
Typ: Operator
Kategorie: Operatoren

ANDALSO prüft zwei Ausdrücke auf ihren Wahrheitsgehalt und gibt -1 zurück, wenn beide Ausdrücke wahr sind. Ansonsten wird 0 zurückgegeben.

Zunächst wird 'Ausdruck1' geprüft. Wenn dieser 0 (false) ergibt, wird mit diesem Ergebnis abgebrochen. Ansonsten wird 'Ausdruck2' ausgewertet. Ist dieser 0 (false), dann wird dies zurückgegeben; ansonsten lautet der Rückgabewert -1 (true). ANDALSO liefert also nur -1, wenn keiner der beiden Ausdrücke 0 ergibt; allerdings wird 'Ausdruck2' nur dann ausgewertet, wenn bereits 'Ausdruck1' nicht 0 war.

Beispiel: ANDALSO in einer IF-THEN-Bedingung:

```
Dim As Ubyte Ptr img = ImageCreate(breit, hoch)
IF img ANDALSO img&"hlzahl">0] = 7 THEN
   PRINT "Image mit neuem Header"
END IF
```

Erläuterung: Falls das Erstellen des Speicherbereichs fehlschlägt - img ist in diesem Fall 0 - dann wird die Auswertung img&"reflinkicon" href="temp0296.html">ORELSE, AND (Operator), Bit Operatoren / Manipulationen

Letzte Bearbeitung des Eintrags am 26.12.12 um 23:42:34
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

ANY

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » A » **ANY**
Syntax: ... AS ANY [PTR]
Typ: Datentyp
Kategorie: Datentypen

Dieser Eintrag behandelt den Datentyp ANY. Für Informationen zu ANY als Klausel siehe INSTR, TRIM, LTRIM, RTRIM und DIM.

Wird ANY in einer DECLARE-Anweisung verwendet, bewirkt es, dass Parameter jeden Typs an die Prozedur übergeben werden dürfen. Innerhalb der Prozedur werden sie dann so behandelt, wie im Prozedurheader angegeben.

Beispiel:

```
DECLARE SUB printFirstByte (byref x AS ANY)

DIM i AS INTEGER, z AS ZSTRING * 12, s AS STRING
i = -1
s = "Hallo Welt!"
z = "Hallo Welt!"

printFirstByte i
printFirstByte s
printFirstByte z
SLEEP

SUB printFirstByte (byref x AS UBYTE)
  PRINT x
END SUB
```

Ausgabe:

```
255
160
72
```

Diese Werte kommen wie folgt zustande: Der INTEGER 'i' hat den Wert -1, d.h. alle Bits sind gesetzt (siehe BIN). Da 'i' BYREF übergeben wird, erhält die SUB nur die Adresse von 'i'. Im Prozedurheader wurde festgelegt, dass der Parameter, dessen Adresse empfangen wurde, wie ein UBYTE behandelt werden soll. Von den 32 Bit der Integervariable 'i' werden also nur die ersten 8 ausgewertet. Diese sind alle gesetzt. In der Behandlung als UBYTE ergibt dies 255.
Vom STRING 's' wird ebenfalls die Adresse übermittelt; wie bei STRING (Datentyp) nachzulesen ist, steht an der Stelle, auf die verwiesen wird, nur der Bezeichner des Strings, nicht aber sein Inhalt. Das Ergebnis 160 ist also ein Teil der Adresse des Stringinhalts. Daher kann das Ergebnis auch von Rechner zu Rechner verschieden sein.
Da 'z' ein ZSTRING ist, hat diese Variable im Gegensatz zu 's' auch keinen Bezeichner. An der Adresse von 'z', die an die Prozedur übergeben wird, ist tatsächlich der Inhalt von 'z'. 72 ist der ASCII-Code von "H", dem ersten Zeichen des Strings.

Die Verwendung von ANY ohne PTR ist nur in DECLARE-Zeilen an BYREF-Argumenten erlaubt, um dort die Typenprüfung bei der Parameterübergabe zu deaktivieren.

Zusammen mit PTR wird ANY oft verwendet, um einen Pointer auf einen Speicherbereich zu setzen, dessen Inhalt kein spezielles Format hat; siehe dazu auch IMAGECREATE und ALLOCATE.

ANY als Startwert:
Bei der Deklaration von Variablen wird dessen Startwert automatisch auf 0 bzw. "" (Leer) gesetzt. Dies lässt sich mit ANY verhindern. Die Variable enthält dann den Wert, der sich bereits zuvor an der zugewiesenen Stelle im Speicher befunden hat.

```
Dim As Integer x1, x2 = 5, x3 = Any

Print "x1 wird    0     zugewiesen: ", x1
Print "x2 wird    5     zugewiesen: ", x2
Print "x3 wird  nichts  zugewiesen: ", x3
Sleep
```

Achtung: Der Typ ANY darf nicht mit VARIANT verwechselt werden, einem VB-Typ, der dazu in der Lage ist, Informationen aller Datentypen zu speichern.

Unterschiede zu QB:
Pointer und Variablen-Initiatoren sind neu in FreeBASIC.

Unterschiede unter den FB-Dialektformen:
Variablen-Initiatoren stehen unter -lang qb nicht zur Verfügung.

Siehe auch:
DECLARE, SUB, FUNCTION, ALLOCATE, PTR, IMAGECREATE, INSTR, TRIM, LTRIM, RTRIM, DIM, Datentypen, Prozeduren

Letzte Bearbeitung des Eintrags am 26.12.12 um 23:43:24
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

APPEND

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » A » **APPEND**
Syntax: OPEN Dateiname FOR APPEND [...]
Typ: Schlüsselwort
Kategorie: Dateien

Das Schlüsselwort APPEND wird mit der OPEN-Anweisung verwendet und öffnet die Datei im APPEND-Modus. Das heißt, die Datei wird mit sequentiellem Zugriff geöffnet und der Lese-/Schreibzugriff erfolgt am Ende der Datei.

Siehe auch:
OPEN (Anweisung), PRINT #, WRITE #, INPUT #, LINE INPUT #, Dateien (Files)

Weitere Informationen:
QB Express Issue #4: File Manipulation In QuickBasic: Sequential Files

Letzte Bearbeitung des Eintrags am 26.12.12 um 23:43:38
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

AS

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » A » **AS**
Syntax A: ... Parameter AS [CONST] Datentyp ...
Syntax B: OPEN Dateiname FOR Zugriffsart AS [#]Dateinummer
Syntax C: SELECT CASE Ausdruck AS CONST
Typ: Klausel
Kategorie: Deklaration

AS ist ein Sprachkonstrukt mit vielen verschiedenen Bedeutungen, die je nach Kontext variieren:

- In DECLARE, SUB und FUNCTION wird damit der Variablentyp in der Parameterliste angegeben.
- In DIM, REDIM, COMMON, STATIC und CONST wird der Datentyp der dimensionierten Variablen festgelegt.
- In TYPE und UNION wird AS verwendet, um den Variablentyp der Records innerhalb von UDTs anzugeben.
- Zusammen mit OPEN wird die Nummer des Dateipuffers beim Öffnen einer Datei angegeben.
- In SELECT CASE wird die Klausel 'AS CONST' verwendet, um die INTEGER-Optimierung zu aktivieren.

Unterschiede zu QB:

- AS wird in FreeBASIC nicht mehr mit dem Befehl NAME benutzt.
- FIELD hat in FreeBASIC eine andere Bedeutung; AS wird dort in QB für die Feldnamen bei RANDOM-Dateien verwendet. Verwenden Sie anstelle von RANDOM-Dateien lieber BINARY-Dateien.
- Die Verwendung von AS mit SELECT CASE ist neu in FreeBASIC.

Unterschiede zu früheren Versionen von FreeBASIC:
Die Verwendung von AS mit SELECT CASE existiert seit FreeBASIC v0.13.

Siehe auch:
Datentypen und Deklarationen

Letzte Bearbeitung des Eintrags am 26.12.12 um 23:45:50
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

ASC

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » A » **ASC**
Syntax: ASC (Ausdruck [,Stelle])
Typ: Funktion
Kategorie: Stringfunktionen

ASC liefert den ASCII-Code des Zeichens in einem Strings zurück.

- 'Ausdruck' ist ein STRING, ein WSTRING oder ein ZSTRING.
- 'Stelle' ist ein numerischer Ausdruck, der angibt, an welcher Stelle des Strings das zu analysierende Byte steht. Wenn 'Stelle' ausgelassen wird, nimmt FreeBASIC automatisch 1 an.
- Der Rückgabewert ist der ASCII-Code des angegebenen Zeichens. Ist 'Stelle' größer als die Länge von 'Ausdruck', so wird 0 zurückgegeben.

ASC ist die Umkehrung von CHR.

ASC kann durch die folgende Syntax ersetzt werden:

```
AsciiCode = Ausdruck[Stelle - 1]
```

Dieser direkte Zugriff ist wesentlich schneller, jedoch findet dabei keine Überprüfung der Stringlänge statt. Wenn 'Stelle' größer ist als die Länge von 'Ausdruck', kommt es daher zu einem Zugriff auf ungültigen Speicherbereich.

Beispiel:

```
PRINT "Der ASCII-Code von 'a' ist:"; ASC("a")
PRINT "ABC in ASCII: "; ASC("ABC", 1); ASC("ABC", 2); ASC("ABC", 3)
```

Unterschiede zu QB:

- Da unter QB ZSTRINGs und WSTRINGs nicht existieren, kann ASC auf solche natürlich nicht angewandt werden.
- Der Parameter 'Stelle' ist neu; unter QB liefert ASC immer den ASCII-Code des ersten Zeichens eines Strings zurück.

Plattformbedingte Unterschiede: DOS unterstützt nicht die WSTRING-Version von ASC.

Siehe auch:
CHR, WCHR, Eckige Klammern, String-Funktionen, Datentypen umwandeln

Weitere Informationen:
ASCII-Tabelle anzeigen lassen

Letzte Bearbeitung des Eintrags am 26.12.12 um 23:50:07
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

ASIN

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » A » **ASIN**
Syntax: ASIN (Zahl)
Typ: Funktion
Kategorie: Mathematik

ASIN gibt den Arcussinus (oder auch Inverssinus) einer Zahl zurück. ASIN bildet also die Umkehrfunktion zu SIN.

- 'Zahl' ist ein beliebiger numerischer Ausdruck. Variablen, Konstanten, Operatoren und Funktionen sind erlaubt. Der Ausdruck darf von jedem Datentyp außer STRING, ZSTRING oder WSTRING sein. 'Zahl' muss zwischen -1 und 1 liegen. Ist 'Zahl' größer als 1 oder kleiner als -1, wird ein Fehler erzeugt.
- Der Rückgabewert ist ein DOUBLE-Wert, der eine Winkelangabe im Bogenmaß darstellt. Er liegt im Bereich von -Pi/2 bis +Pi/2

ASIN kann mithilfe von OPERATOR überladen werden.

Beispiel:

```
DIM AS DOUBLE h, o

INPUT "Länge der Hypotenuse des Dreiecks: ", h
INPUT "Länge der Gegenkathete des Dreiecks: ", o
PRINT
PRINT "Der Winkel zwischen den Seiten ist"; ASIN (o/h)
SLEEP
```

Ausgabebeispiel:

```
Länge der Hypotenuse des Dreiecks: 5
Länge der Gegenkathete des Dreiecks: 3

Der Winkel zwischen den Seiten ist 0.6435011087932844
```

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC:
Die Überladung von ASIN für benutzerdefinierte Datentypen ist seit FreeBASIC v0.22 möglich.

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht ASIN nicht zur Verfügung und kann nur über **__ASIN** aufgerufen werden.

Siehe auch:
SIN, COS, ACOS, TAN, ATN, ATAN2, mathematische Funktionen

Letzte Bearbeitung des Eintrags am 24.12.12 um 14:12:09
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

ASM

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » A » **ASM**
Syntax A: ASM <ASSEMBLER-Code>
Syntax B:

```
ASM
<ASSEMBLER-Code>
END ASM
```

Typ: Anweisung
Kategorie: Programmablauf

ASM bindet Maschinensprache-Code ins Programm ein. Derzeit werden nur ASM-Anweisung für x86 Prozessoren unterstützt.
Der Assembler-Code unterliegt der INTEL-Syntax. Es wird der gesamte IA32-Befehlssatz unterstützt.

Kommentare in ASM-Blöcken verwenden dieselbe Syntax wie FreeBASIC-Kommentare. Verwenden Sie als Kommentarzeichen das Hochkomma " ' " und nicht das in ASM übliche Semikolon " ; "

Wenn Sie ASM-Code innerhalb einer FUNCTION einsetzen, können Sie das Symbol 'FUNCTION' als Pointer einsetzen (siehe Beispiel), der auf das Ergebnis der FUNCTION zeigt.

Beispiel: Multiplikation zweier Integer mit Assembler

```
Function Mal(ByVal x As Integer, ByVal y As Integer) As Integer
  Asm
    mov  eax, &"hlzeichen">] 'hole x nach eax
    imul eax, [y] 'Multipliziere mit y
    mov [Function], eax 'Ergebnis als Rückgabewert
  End Asm
End Function

Dim As Integer a = 45, b = 54
Print Mal(a, b)
Sleep
```

Weitere Hinweise und Erkärungen finden Sie hier unter BASIC-Grundlagen Inline-Assembler.

Unterschiede zu QB: neu in FreeBASIC

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht ASM nicht zur Verfügung und kann nur über **__ASM** aufgerufen werden.

Siehe auch:
Inline-Assembler, NAKED, Verschiedenes

Letzte Bearbeitung des Eintrags am 16.06.12 um 21:40:37
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

ASSERT

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » A » **ASSERT**
Syntax: ASSERT (Ausdruck)
Typ: Makro
Kategorie: Fehlerbehandlung

Das Makro ASSERT wurde zum Debugging entwickelt und funktioniert nur, wenn der Compiler mit der Option "-g" aufgerufen wird. Ist dies der Fall, wird das Programm beendet, sobald 'Ausdruck' gleich null ist. Zusätzlich wird eine Zeile ausgegeben:

```
Dateiname(Zeilennummer): assertion failed at PROZEDUR: Ausdruck
```

- 'Dateiname' ist der Dateiname des gerade aktiven Codes inklusive Pfad.
- 'Zeilennummer' ist die Zeile, in der abgebrochen wurde.
- 'PROZEDUR' ist die Prozedur, in der abgebrochen wurde. Wird das Programm auf Modulebene abgebrochen, wird __FB_MAINPROC__ ausgegeben.
- 'Ausdruck' ist der Ausdruck, der null geworden ist.

Wenn die Option "-g" nicht an fbc übergeben wird, erzeugt dieses Makro keinen Code; das Programm wird nicht beendet, wenn der Ausdruck gleich null wird.

Beispiel:

```
Declare Sub foo()

Sub foo()
  Dim a As Integer = 10
  For i As Integer = 1 To 20
    a -= 1
    Assert (a <> 0)
    Print a
  Next
End Sub

foo()

Sub Quit Destructor
  Sleep
End Sub
```

Wenn -g an den Compiler übergeben wird, wird das Programm mit dieser Meldung beendet:

```
test.bas(8): assertion failed at FOO: a <> 0
```

Die DESTRUCTOR-Sub bewirkt, dass auf einen Tastendruck gewartet wird; so kann die Meldung gelesen werden und verschwindet nicht sofort. Das Beispiel schlägt unter Umständen fehl, wenn Sie es z. B. mit der "Quick Run"-Option in FbEdit aufrufen. Eine fertig compilierte Debug-EXE wird aber wie erwartet bei 0 anhalten.

Unterschiede zu QB: neu in FreeBASIC

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht ASSERT nicht zur Verfügung und kann nur über **__ASSERT** aufgerufen

werden.

Siehe auch:
ASSERTWARN, Der Compiler, Fehlerbehandlung, Debugging

Letzte Bearbeitung des Eintrags am 19.01.13 um 16:06:53
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

ASSERT (Meta)

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » A » **ASSERT (Meta)**
Syntax: #ASSERT Bedingung
Typ: Metabefehl
Kategorie: Metabefehle

#ASSERT gibt zur Compile-Zeit einen Fehler aus und beendet die Compilierung, sofern die angegebene Bedingung falsch ist. Ist die Bedingung aber wahr, so wird kein Fehler ausgegeben und die Compilierung normal fortgeführt.
Im Gegensatz dazu wird ASSERT nicht zur Compile-Zeit, sondern zur Laufzeit des Programms überprüft.

Beispiel:

```
Const MIN = 5, MAX = 10
"hlzeichen">> MIN ' gibt einen Fehler aus, sobald MAX <= MIN
```

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC: existiert seit FreeBASIC v0.90

Siehe auch:
ASSERT, ASSERTWARN, ERROR (Meta), Präprozessor-Anweisungen

Letzte Bearbeitung des Eintrags am 02.07.13 um 23:53:52
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

ASSERTWARN

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » A » **ASSERTWARN**
Syntax: ASSERTWARN (Ausdruck)
Typ: Makro
Kategorie: Fehlerbehandlung

ASSERTWARN funktioniert genauso wie ASSERT, nur dass das Programm nicht beendet wird, sobald der Ausdruck gleich null wird. Es wird lediglich die Meldung ausgegeben.

Auch ASSERTWARN wird nur ausgeführt, wenn der Compiler mit der Option "-g" ausgeführt wird.

Unterschiede zu QB: neu in FreeBASIC

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht ASSERTWARN nicht zur Verfügung und kann nur über **__ASSERTWARN** aufgerufen werden.

Siehe auch:
ASSERT, Der Compiler, Fehlerbehandlung, Debugging

Letzte Bearbeitung des Eintrags am 26.12.12 um 23:51:28
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

ATAN2

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » A » **ATAN2**
Syntax: ATAN2 (Zahl1, Zahl2)
Typ: Funktion
Kategorie: Mathematik

ATAN2 gibt den Arcustangens (Inverstangens) des Quotienten zweier Zahlen zurück. Der Arcustangens ist der Winkel, dessen Tangens die angegebene Zahl ergeben würde.

- 'Zahl1' und 'Zahl2' sind beliebige numerische Ausdrücke. Variablen, Konstanten, Operatoren und Funktionen sind erlaubt. Die Ausdrücke dürfen von jedem Zahlendatentyp sein, jedoch kein STRING, ZSTRING, WSTRING oder UDT. Diese beiden Parameter stellen die Steigung einer Geraden dar. Dabei gilt
  m = Zahl1 / Zahl2
  'Zahl1' stellt also die vertikale Komponente, 'Zahl2' die horizontale Komponente der Steigung dar.
- Der Rückgabewert ist ein DOUBLE-Wert, der eine Winkelangabe im Bogenmaß darstellt. Er liegt im Bereich von -Pi bis +Pi.

ATAN2 liefert den Arcustangens des Quotienten zweier arithmetischer Ausdrücke, ohne dabei den Quotienten explizit zu berechnen, so dass keine Division durch Null auftreten kann. Im Gegensatz zur ATN-Funktion kann bei der ATAN2-Funktion das Ergebnis zwischen minus Pi und plus Pi liegen, also in allen 4 Quadranten des Koordinatensystems. Dies ist möglich, weil es eine Unterscheidungsmöglichkeit gibt zwischen x>0 und y<0 sowie x<0 und y>0, usw. Durch die Division zweier Zahlen geht die Information, welcher der beiden Parameter < 0 ist, verloren. ATN(x) ist eigentlich ATAN2(x,1) und nicht ATAN2(-x,-1).

Beispiel:

```
PRINT ATAN2(7, 24)
' ergibt das gleiche wie
PRINT ATN  (7 / 24)
' oder
PRINT ATAN2(7 / 24, 1)


PRINT ATAN2(24, 0)
' ergibt einen Wert, während ATN(24/0) nicht funktioniert
SLEEP
```

Unterschiede zu QB: neu in FreeBASIC

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht ATAN2 nicht zur Verfügung und kann nur über **__ATAN2** aufgerufen werden.

Siehe auch:
mathematische Funktionen, SIN, ASIN, COS, ACOS, TAN, ATN

Letzte Bearbeitung des Eintrags am 24.12.12 um 14:26:37
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

ATN

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » A » **ATN**
Syntax: ATN (Zahl)
Typ: Funktion
Kategorie: Mathematik

ATN gibt den Arcustangens (Inverstangens) einer Zahl zurück. Der Arcustangens ist der Winkel, dessen Tangens die angegebene Zahl ergeben würde.

- 'Zahl' ist ein beliebiger numerischer Ausdruck. Variablen, Konstanten, Operatoren und Funktionen sind erlaubt. Der Ausdruck darf von jedem Datentyp außer STRING, ZSTRING oder WSTRING sein.
- Der Rückgabewert ist ein DOUBLE-Wert, der eine Winkelangabe im Bogenmaß darstellt. Er liegt im Bereich zwischen -Pi/2 und +Pi/2

Die Umkehrfunktion zu ATN lautet TAN. In vielen Fällen ist es praktisch, eine besondere Version des Arcustangens zu benutzen: siehe ATAN2.

ATN kann mithilfe von OPERATOR überladen werden.

Beispiel:

```
PRINT "Pi ="; Atn ( 1.0 ) * 4
PRINT ATN ( 4 / 5 )
SLEEP
```

Ausgabe:

```
Pi = 3.141592653589793
 0.6747409422235527
```

Unterschiede zu früheren Versionen von FreeBASIC:
Die Überladung von ATN für benutzerdefinierte Datentypen ist seit FreeBASIC v0.22 möglich.

Siehe auch:
SIN, ASIN, COS, ACOS, TAN, ATAN2, mathematische Funktionen

Letzte Bearbeitung des Eintrags am 24.12.12 um 14:15:49
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

BASE

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » B » **BASE**
Syntax: OPTION BASE Startindex
Typ: Schlüsselwort
Kategorie: Programmoptionen

OPTION BASE setzt den standardmäßig kleinsten Index für Arrays. Die Option kann **nur bis FreeBASIC v0.16** eingesetzt werden, oder in entsprechend höheren Versionen, die mit der Kommandozeilenoption -lang deprecated compiliert wurden! Wird mit FreeBASIC v0.17 unter der Option -lang fb compiliert, so ist OPTION BASE nicht mehr zulässig! Geben Sie in diesem Fall bei der Dimensionierung eines Array immer explizit die untere Arraygrenze an.

'Startindex' ist ein INTEGER-Wert, der bei der Dimensionierung von Arrays verwendet wird, wenn eine Untergrenze nicht explizit angegeben wurde. Standardmäßig wird als untere Grenze 0 verwendet. 'Startindex' muss eine Zahl sein, es darf kein Ausdruck sein. Auch Variablen und einfache Ausdrücke, die nur aus Zahlen und mathematischen Operatoren bestehen, sind unzulässig. Aus diesem Grund können keine negativen Startindizes angegeben werden, da hierfür der Operator '-' zur Negation benötigt würde.

OPTION BASE ist ein Programmstandard, der mehrmals neu definiert werden darf.

Beispiel:

```
"hlstring">"fblite"
DIM AS INTEGER a(5)

OPTION BASE 4
DIM AS INTEGER b(5)

OPTION BASE 2
DIM AS INTEGER c(5)

DIM AS INTEGER d(0 TO 5)

PRINT LBOUND(a)
PRINT LBOUND(b)
PRINT LBOUND(c)
PRINT LBOUND(d)

SLEEP
```

Ausgabe:

```
 0
 4
 2
 0
```

Unterschiede zu QB:

- 'Startindex' darf in FreeBASIC auch andere Werte als 0 und 1 annehmen.
- Arrays müssen in FreeBASIC immer explizit erzeugt werden. QBASIC erzeugt implizit ein Array vom kleinsten Index bis 10, wenn es verwendet wird, ohne vorher definiert worden zu sein.

Unterschiede zu früheren Versionen von FreeBASIC:
Die Option ist nur bis FreeBASIC v0.16 erlaubt. Seit FreeBASIC v0.17 ist diese Option nur noch zulässig, wenn mit der Kommandozeilenoption -lang deprecated compiliert wurde.

Siehe auch:
BASE (Vererbung), DIM, OPTION, Verschiedenes

Letzte Bearbeitung des Eintrags am 19.01.13 um 16:07:38
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

BASE (Vererbung)

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » B » **BASE (Vererbung)**
Syntax: BASE
Typ: Datentyp
Kategorie: Klassen

BASE wird in Verbindung mit Vererbung verwendet. Dabei gibt BASE die Möglichkeit, auf ein Record der Eltern-Klasse zuzugreifen, selbst wenn die eigene Klasse ein Record mit gleichem Namen besitzt.

Auch auf die Methoden der Elternklasse kann zugegriffen werden; diese sprechen über THIS allerdings auch die Records und Methoden der Elternklasse an.

Beispiel:

```
Type ElternKlasse
  As Integer variable = 1
  Declare Sub ausgabe
End Type

Sub ElternKlasse.ausgabe
  Print "Variable der Klasse: " & This.variable
End Sub

Type KindKlasse Extends ElternKlasse
  As Integer variable = 2
  Declare Sub ausgabe
End Type

Sub KindKlasse.ausgabe
  Print "Variable der Eltern-Klasse: " & Base.variable
  Print "Variable der Kind-Klasse:   " & This.variable
  Print "================================"
  Print "Ausgabe-Methode der Elternklasse:"
  Base.ausgabe
End Sub

Dim As KindKlasse testKlasse

testKlasse.ausgabe
Sleep
```

Ausgabe:

```
Variable der Eltern-Klasse: 1
Variable der Kind-Klasse:   2
================================
Ausgabe-Methode der Elternklasse:
Variable der Klasse: 1
```

Hinweis: Mit der Syntax *'BASE()'* lässt sich innerhalb des CONSTRUCTORs einer Kind-Klasse der Constructor der Eltern-Klasse aufrufen. Um den Constructor auch aus anderen Methoden der Klasse aufzurufen, ist die Syntax *'BASE.CONSTRUCTOR()'* zu verwenden.

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC:
BASE in Verbindung mit Vererbung existiert seit FreeBASIC v0.24.

Unterschiede unter den FB-Dialektformen:
BASE in Verbindung mit Vererbung ist nur in -lang fb zulässig.

Siehe auch:
TYPE (UDT), OBJECT, EXTENDS, THIS, IS (Vererbung), Objektorientierung

Letzte Bearbeitung des Eintrags am 31.05.12 um 22:16:27
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

BEEP

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » B » **BEEP**
Syntax: BEEP
Typ: Anweisung
Kategorie: System

BEEP weist das System an, ein Tonsignal auszugeben (ein Signalton aus dem internen PC-Speaker, über die Soundkarte oder auch teilweise als Standard-Signal, das auch beim Auftauchen eines Dialogfeldes ertönt). Das ausgegebene akustische Signal variiert je nach Ausführungsplattform.
Wenn keine Soundkarte vorhanden ist, wird (sofern vorhanden) ein Signal über den PC-Speaker ausgegeben. Der ausgegebene Ton lässt sich weder in der Dauer noch in der Frequenz beeinflussen.

Beispiel: Eingabe von Ziffern; die Eingabe anderer Tasten erzeugt einen Signalton

```
dim as string taste
do
  taste = input(1)
  select case taste
    case "0" to "9" : print taste;
    case chr(27)    : exit do  ' Escape-Taste zum Beenden
    case else       : beep
  end select
loop
```

Unterschiede zu QB:
BEEP erzeugte einen einzelnen Ton über den PC-Speaker. Unter FreeBASIC variiert die Ausgabeform.

Siehe auch:
OUT, Verschiedenes

Letzte Bearbeitung des Eintrags am 01.06.12 um 18:44:06
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

BIN

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » B » **BIN**
Syntax: BIN[$] (Ausdruck [, Stellen])
Typ: Funktion
Kategorie: Stringfunktionen

BIN gibt den binären Wert eines beliebigen Ausdrucks als STRING zurück. Binärzahlen haben die Basis 2; sie bestehen aus den Ziffern 0 und 1.

- 'Ausdruck' ist eine Ganzzahl (eine Zahl ohne Nachkommastellen), die ins Binärformat übersetzt werden soll.
- 'Stellen' ist die Anzahl der Stellen, die dafür aufgewandt werden soll. Ist 'Stellen' größer als die benötigte Stellenzahl, wird der Rückgabewert mit führenden Nullen aufgefüllt; der zurückgegebene Wert ist jedoch nie länger, als maximal für den Datentyp von 'Ausdruck' benötigt wird.
  Ist 'Stellen' kleiner als die benötigte Stellenzahl, werden nur die hinteren Zeichen des Rückgabewerts ausgegeben.
  Wird 'Stellen' ausgelassen, besteht der Rückgabewert aus so vielen Zeichen, wie benötigt werden, um die Zahl korrekt darzustellen.
- Der Rückgabwert ist ein String, der den Wert von 'Ausdruck' im Binärformat enthält.

Das Dollarzeichen ($) als Suffix ist optional.

Beispiel:

```
PRINT BIN(54321)   ' Ausgabe: 1101010000110001
PRINT BIN(3, 3)    ' Ausgabe: 011
PRINT BIN(255, 4) ' Ausgabe: 1111
SLEEP
```

Um eine Binärzahl in ihre dezimale Form zurückzuverwandeln, wird VALINT verwendet:

```
DIM binaer AS STRING

binaer = "1001"
'Prefix &b zeigt an, dass der folgende String eine Binärzahl ist.
binaer = "&b" & binaer

PRINT VALINT(binaer)
SLEEP
```

gibt 9 aus.

Unterschiede zu QB: neu in FreeBASIC

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht BIN nicht zur Verfügung und kann nur über **__BIN** aufgerufen werden.

Siehe auch:
HEX, OCT, VAL, WBIN, BIT, BITSET, BITRESET, Datentypen umwandeln

Letzte Bearbeitung des Eintrags am 25.08.12 um 18:30:27
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

BINARY

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » B » **BINARY**
Syntax: OPEN Dateiname FOR BINARY [...]
Typ: Schlüsselwort
Kategorie: Dateien

Das Schlüsselwort BINARY wird mit der OPEN-Anweisung verwendet und öffnet die Datei im BINARY-Modus. Sowohl der Lese- als auch der Schreibzugriff kann in einer BINARY-Datei an jeder beliebigen Position (an jedem beliebigen Byte) erfolgen. Dazu werden GET # zum Lesen von Daten und PUT # zum Schreiben von Daten verwendet. Durch diese einfache Verwaltung ist der BINARY-Modus der Flexibelste.

Im Modus BINARY können auch die sequentiellen Lese- und Schreibzugriffe INPUT #, PRINT # und WRITE # verwendet werden.

Siehe auch:
OPEN (Anweisung), PUT #, GET #, SEEK (Anweisung), SEEK (Funktion), Dateien (Files)

Letzte Bearbeitung des Eintrags am 26.12.12 um 23:54:22
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

BIT

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » B » **BIT**
Syntax: BIT (Ausdruck, BitNr)
Typ: Funktion
Kategorie: Bitfelder

BIT wird benutzt, um zu prüfen, ob das Bit an der Stelle 'BitNr' gesetzt ist. Wenn ja, gibt BIT true (-1) aus, ansonsten false (0).

BIT erfüllt dieselbe Funktion wie

```
(Ausdruck AND (1 SHL BitNr)) <> 0
```

Beispiel:

```
DIM AS INTEGER foo = 1024
PRINT BIT( foo, 10 )
PRINT (foo AND (1 SHL 10)) <> 0
SLEEP
```

Ausgabe:

```
-1
-1
```

BIT wird intern folgendermaßen behandelt:

```
"cnf">__BIT in der Dialektform -lang qb existiert seit FreeBASIC v0.24.
```

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht BIT nicht zur Verfügung und kann nur über **__BIT** aufgerufen werden.

Siehe auch:
BITSET, BITRESET, BIN, Bit-Operatoren
Letzte Bearbeitung des Eintrags am 26.12.12 um 23:54:43
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

BITRESET

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » B » **BITRESET**
Syntax: BITRESET (Ausdruck, BitNr)
Typ: Funktion
Kategorie: Bitfelder

BITRESET gibt den Wert von 'Ausdruck' zurück, bei dem das Bit an der Stelle 'BitNr' gelöscht wurde.

BITRESET erfüllt dieselbe Funktion wie

```
Ausdruck AND NOT (1 SHL BitNr)
```

Beispiel:

```
DIM AS INTEGER foo = 192
PRINT BITRESET( foo, 6 )
PRINT foo AND NOT (1 SHL 6)
PRINT
PRINT BITRESET(33, 2)  ' keine Änderung
SLEEP
```

Ausgabe:

```
 128
 128

 33
```

Beachten Sie, dass die Nummerierung der Bits bei 0 beginnt.

BITRESET wird intern folgendermaßen behandelt:

```
"cnf">__BITRESET in der Dialektform -lang qb existiert seit FreeBASIC
v0.24.
```

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht BITRESET nicht zur Verfügung und kann nur über **__BITRESET** aufgerufen werden.

Siehe auch:
BIT, BITSET, BIN, Bit-Operatoren
Letzte Bearbeitung des Eintrags am 26.12.12 um 23:55:00
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

BITSET

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » B » **BITSET**
Syntax: BITSET (Ausdruck, BitNr)
Typ: Funktion
Kategorie: Bitfelder

BITSET gibt den Wert von 'Ausdruck' zurück, bei dem das Bit an der Stelle 'BitNr' gesetzt wurde.

BITSET erfüllt dieselbe Funktion wie

```
Ausdruck OR (1 SHL BitNr)
```

Beispiel:

```
DIM AS INTEGER foo = 128
PRINT BITSET( foo, 6 )
PRINT foo OR (1 SHL 6)
PRINT
PRINT BITSET(33, 5)  ' keine Änderung
SLEEP
```

Ausgabe:

```
 192
 192

 33
```

Beachten Sie, dass die Nummerierung der Bits bei 0 beginnt.

BITSET wird intern folgendermaßen behandelt:

```
"cnf">__BITSET in der Dialektform -lang qb existiert seit FreeBASIC
v0.24.
```

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht BITSET nicht zur Verfügung und kann nur über **__BITSET** aufgerufen werden.

Siehe auch:
BIT, BITRESET, BIN, Bit-Operatoren
Letzte Bearbeitung des Eintrags am 26.12.12 um 23:55:17
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

BLOAD

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » B » **BLOAD**
Syntax: BLOAD (Dateiname[, Adresse][, pal])
Typ: Funktion
Kategorie: Dateien

BLOAD lädt einen Block binärer Daten aus einer Datei. Der Befehl wird typischerweise benutzt, um Bilder aus externen Dateien (etwa BMPs) zu laden.

- 'Dateiname' ist der Name der Datei, aus der geladen werden soll.
- 'Adresse' ist ein ANY PTR auf die Adresse, auf welche die Daten geladen werden sollen. Wenn 'Adresse' ausgelassen oder mit 0 (null) angegeben wird, schreibt BLOAD die Daten auf die aktuelle Bildschirmseite.
- 'pal' ist die Adresse, auf welche die Farbpalette geladen werden soll. Wenn 'Adresse' ausgelassen oder mit 0 (null) angegeben wird, schreibt BLOAD die Daten in die aktuelle Farbpalette.
- Der Rückgabewert ist ein INTEGER mit einer der FreeBASIC-Fehlernummern. Mögliche Fehlermeldungen sind
  - 1 (Illegal function call): Die geladenen Daten können nicht verarbeitet werden.
  - 2 (File not found): Die Datei konnte nicht gefunden werden.
  - 3 (File I/O error): Die Datei besitzt keinen unterstützten Typ oder ein allgemeiner Lesefehler ist aufgetreten.

  Der Rückgabewert kann verworfen werden; BLOAD wird dann wie eine Anweisung eingesetzt.

BLOAD kann dazu verwendet werden, einen Block binärer Daten zu laden. Ebenso ist es möglich, einen gespeicherten Bildschirmausschnitt direkt auf den Bildschirm zu laden: Wenn Sie 'Adresse' auslassen oder 0 angeben, werden die Daten als Pixel behandelt und auf die aktuelle Bildschirmseite geladen.

BLOAD unterstützt folgende Dateiformate:

- Altes QB BSAVE-Format (Dateien, die mit QBs BSAVE gespeichert wurden), beginnend mit &hFD
- Neues FreeBASIC BSAVE-Format (Dateien, die mit FreeBASICs BSAVE gespeichert wurden), beginnend mit &hFE
- Diverse (Windows-)Bitmaps (.BMP), beginnend mit "BM", s.u.

Bilder werden beim Laden in ein FreeBASIC-kompatibles Bildformat konvertiert.

Das BSAVE-Format von QB erlaubt bis zu 64KB große Binärdaten. Es besitzt einen 7-Byte großen Header, dem die Rohdaten folgen:

Offset	Länge	Daten
1	1 Byte	ID (= &hFD)
2	2 Bytes	Datensegment (DS) zum Speicherzeitpunkt (von FreeBASIC ignoriert)
4	2 Bytes	Offset im Datensegment zum Speicherzeitpunkt (von FreeBASIC ignoriert)
6	2 Bytes	Länge der Binärdaten

Das BSAVE-Format von FreeBASIC erlaubt bis zu 4GB große Binärdaten. Es besitzt einen 5-Byte großen Header, dem die Rohdaten folgen:

Offset	Länge	Daten
1	1 Byte	ID (= &hFE)
2	4 Bytes	Länge der Binärdaten

Es ist auch möglich, mit BLOAD **Bitmaps** zu laden. Dazu muss lediglich als Dateiname eine Bitmapdatei (z. B. "MeinBild.bmp") angegeben werden. BLOAD erkennt anhand des Dateianfangs (magic number), dass es sich um eine Bitmap-Grafik handelt. Es werden palette-indizierte BMPs (8 Bit und weniger) und Truecolor-BMPs (15, 16, 24 oder 32 Bit) unterstützt. Die Palette von BMPs mit 8Bit oder weniger wird entweder an die durch den Parameter 'pal' festgelegte Adresse geladen oder, wenn dieser 0 oder unspezifiziert ist, als aktuelle Farbpalette eingesetzt.

Achten Sie beim Laden von Bildern darauf, dass diese die gleiche Farbtiefe aufweisen wie der gewählte Bildschirmmodus (siehe SCREENRES). Laden Sie beispielsweise kein 24-Bit-BMP in ein Grafikfenster mit Farbtiefe 4 Bit. Wenn Bilder in einem der BSAVE-Formate geladen werden sollen, müssen diese im selben Bildschirmmodus mittels BSAVE erstellt worden sein.

Wird die zu ladende Datei nicht gefunden oder ihr Format nicht unterstützt (dies ist z. B. bei RLE-komprimierten Bitmaps der Fall), verursacht BLOAD einen Laufzeitfehler. Ebenso wird ein Fehler verursacht, wenn versucht wird, eine Truecolor-BMP in einem palette-indizierten Bildschirmmodus zu laden.

Beispiel:
Ein 48x48 Pixel großes Bild in einen Puffer laden und auf dem Bildschirm ausgeben. Das Beispiel benötigt zur korrekten Ausführung ein 48x48 Pixel großes Bild namens "picture.bmp".

```
SCREENRES 640, 480, 32
Dim As Any Ptr bild
bild = ImageCreate(48, 48)  'Bildpuffer der Groesse 48x48 anlegen
BLoad "picture.bmp", bild   'Bilddatei in den Puffer laden
PUT (10, 10), bild          'Bildpuffer auf den Bildschirm zeichnen
ImageDestroy (bild)         'Bildpuffer wieder loeschen
Sleep                       'Auf Tastendruck warten
End
```

Um Speicherlecks zu vermeiden, sollte man erstellte Bildpuffer nach der Nutzung stets wieder löschen. Dies geschieht mit IMAGEDESTROY. Es empfiehlt sich weiterhin, den Pointer 'bild' auf Null zu setzen, um Zugriffsfehler zu vermeiden. Dies ist aber nicht zwingend nötig.

Die Methode, ein Bild in ein selbst erstelltes Byte-Array zu laden, ist **veraltet** und wird hier nur noch zu Dokumentationszwecken aufgeführt:

```
SCREENRES 640, 480, 32
DIM garray(4 * (48 * 48) + 4) AS Byte
BLOAD "picture.bmp", @garray(0)
PUT (10, 10), garray(0)
SLEEP
```

Unterschiede zu QB:

- BLOAD unterstützt auch das Laden von Datenblöcken größer als 64KB sowie von Bitmaps.
- BLOAD kann in FreeBASIC auch als Funktion eingesetzt werden.

Unterschiede zu früheren Versionen von FreeBASIC:

- Die Adresse der Farbpalette kann seit FreeBASIC v0.20.0 angegeben werden.
- Die Möglichkeit, BMPs höherer Farbtiefen als 8 Bit zu laden, existiert seit v0.14.
- Die Möglichkeit, Bitmaps zu laden, existiert seit FreeBASIC v0.12.

Siehe auch:

BSAVE, PUT (Grafik), GET (Grafik), IMAGECREATE, IMAGEDESTROY, Dateien (Files)

Letzte Bearbeitung des Eintrags am 26.12.12 um 23:55:54
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

BSAVE

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » B » **BSAVE**
Syntax: BSAVE (Dateiname, Adresse[,Größe][, pal][, BitsProPixel])
Typ: Funktion
Kategorie: Dateien

BSAVE speichert einen Block binärer Daten in eine Datei. Der Befehl kann auch benutzt werden, um Daten von der aktuellen Bildschirmseite in eine Datei zu speichern, wenn 0 als Adresse angegeben wird.

- 'Dateiname ist der Name der Datei, in welche die Daten geschrieben werden.
- 'Adresse' ist ein ANY PTR auf die Adresse, von der die zu schreibenden Daten stammen. Wenn 0 (Null) angegeben wird, kopiert BSAVE die Daten von der aktuellen Bildschirmseite. Üblicherweise ist dieser Parameter ein Pointer auf einen mit IMAGECREATE oder ALLOCATE erstellten Speicherbereich oder ein Pointer auf ein Array.
- 'Größe' ist die Größe des Datenblocks in Bytes, der gespeichert werden soll. Wird dieser Parameter ausgelassen, speichert BSAVE das gesamte Array. Ist 'Adresse' ein Pointer, so muss 'Größe' angegeben werden, da sonst gar nichts gespeichert wird.
- 'pal' die Adresse auf einen Speicherbereich, der die Werte der Farbpalette enthält, oder 0 (Null), wenn die Standard-Farbpalette gespeichert werden soll.
- 'BitsProPixel' gibt an, in welcher Bittiefe das Bild gespeichert werden soll.
- Der Rückgabewert ist eine der FreeBASIC Fehlernummern. Mögliche Fehlermeldungen sind
  - 2 (File not found): Die Datei konnte nicht erstellt werden, oder die Dateigröße ist kleiner 0, oder die Dateigröße ist 0 und 'Adresse' ist nicht 0.
  - 3 (File I/O error): Die Datei konnte nicht beschrieben werden.

  Der Rückgabewert kann verworfen werden; BSAVE wird dann wie eine Anweisung eingesetzt.

Wenn Sie Daten vom Bildschirm speichern, werden diese im selben Format gespeichert, das auch der aktuelle Bildschirmmodus verwendet; siehe Interne Pixelformate für Details. Beachten Sie besonders, dass ein Pixel möglicherweise mehr als ein Byte benötigt. Dementsprechend müssen Sie den Parameter 'Größe' anpassen.

BSAVE wird oft dazu verwendet, um Bildschirmausschnitte zu speichern, die zuvor mit GET (Grafik) eingelesen wurden. Diese können dann später mit PUT (Grafik) wieder ausgegeben werden.

Mit BSAVE können Pixeldaten auch im BMP-Format gespeichert werden; geben Sie dazu einfach einen Dateinamen mit Erweiterung '.bmp' an. In diesem Fall kann der Parameter 'Größe' ausgelassen werden, da die Größe direkt aus dem Bildpuffer entnommen wird.

Die Standard-Bittiefe für BMPs ist *8bit für 8bit*-Grafiken, *24bit für 16bit*-Grafiken und *32bit für 32bit*-Grafiken. Mit dem Parameter 'BitsProPixel' kann dieses Verhalten angepasst werden.

Beispiel:

```
' Grafikmodus initialisieren
SCREENRES 320, 200, 32

' Schwarzer Text auf weißem Grund
COLOR RGB(0, 0, 0), RGB(255, 255, 255)
CLS

LOCATE 13, 15: PRINT "Hello world!"

' Gesamten Bildschirm als Bitmap speichern
BSAVE "hello.bmp", 0
```

SLEEP

Unterschiede zu QB:

- Daten, die mit der FreeBASIC-Version von BSAVE gespeichert wurden, sind nicht kompatibel zu den Daten, die mit QBs BSAVE gespeichert wurden.
- Die binären Daten können in FreeBASIC auch im Bitmap-Format gespeichert werden.
- BSAVE kann in FreeBASIC als Funktion eingesetzt werden.
- Der Parameter 'BitsProPixel' ist neu in FreeBASIC.

Unterschiede zu früheren Versionen von FreeBASIC:

- Der Parameter 'BitsProPixel' existiert seit FreeBASIC v0.90.
- Die Adresse der Farbpalette kann seit FreeBASIC v0.20 angegeben werden.
- Die Möglichkeit, im BMP-Format zu speichern, existiert seit v0.14.
- Die Möglichkeit, Bitmaps zu speichern, existiert seit FreeBASIC v0.12.

Siehe auch:
BLOAD, GET (Grafik), PUT (Grafik), VARPTR, Dateien (Files)

Letzte Bearbeitung des Eintrags am 04.07.13 um 18:42:26
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

BYREF

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » B » **BYREF**
Syntax: [DECLARE] { SUB | FUNCTION } Name (BYREF parameter AS Typ, ...) [AS Typ]
Typ: Klausel
Kategorie: Deklaration

BYVAL legt in einer Parameterliste fest, dass ein Parameter 'by reference', also als direkte Referenz statt nur als Wert übergeben werden soll. Die Klausel kann bei der Deklaration bzw. Definition einer Prozedur angegeben werden; siehe dazu den Referenzeintrag Parameterübergabe. Zur Syntax siehe auch DECLARE, SUB und FUNCTION.

Bei einer BYREF übergebenen Variable wird beim Prozeduraufruf intern ein Zeiger auf den Speicherbereich der Variablen übergeben. Das hat zur Folge, dass (im Gegensatz zu BYVAL) alle Änderungen dieser Variablen in der bearbeitenden Prozedur direkt auf die ursprüngliche Variable einwirken, da ja direkt auf sie verwiesen wird.

In der aktuellen Version von FreeBASIC werden bei Zahlendatentypen (z. B. INTEGER, SINGLE) standardmäßig lediglich die Werte übergeben (by value). Um eine Variable als Referenz zu übergeben, muss die Klausel BYREF explizit angegeben werden.

STRINGs und UDTs werden standardmäßig BYREF übergeben; die Verwendung des Schlüsselwortes ist in diesem Fall überflüssig.

Arrays werden immer BYREF übergeben; die Verwendung des Schlüsselwortes BYREF oder BYVAL führt bei Arrays zu einer Fehlermeldung.

Beachten Sie aber auch die Unterschiede unter den Dialektformen und zu früheren Compiler-Versionen!

Beispiel:

```
DECLARE SUB MySub (BYREF a AS INTEGER, BYVAL b AS INTEGER)

DIM AS INTEGER a, b

a = 10
b = 10
PRINT "a = "; a
PRINT "b = "; b
PRINT

MySub a, b

PRINT "Wieder auf Modulebene:"
PRINT "a = "; a
PRINT "b = "; b
SLEEP

SUB MySub (BYREF a AS INTEGER, BYVAL b AS INTEGER)
   a += 10
   b += 10

   PRINT "in MySub:"
   PRINT "a = "; a
```

```
   PRINT "b = "; b
   PRINT
END SUB
```

Ausgabe:

```
a = 10
b = 10

in MySub:
a = 20
b = 20

Wieder auf Modulebene:
a = 20
b = 10
```

Unterschiede zu QB:
In QB werden Variablen standardmäßig BYREF übergeben.

Unterschiede zu früheren Versionen von FreeBASIC:
Bis einschließlich FreeBASIC v0.16 wurden Variablen standardmäßig BYREF übergeben.

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb, -lang deprecated und -lang fblite werden Variablen standardmäßig BYREF übergeben.

Siehe auch:
BYVAL, Parameterübergabe, BYREF (Rückgaben)

Letzte Bearbeitung des Eintrags am 04.07.13 um 23:03:32
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

BYREF (Rückgaben)

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » B » **BYREF (Rückgaben)**
Syntax: [DECLARE] FUNCTION Name (...) BYREF AS Typ
Typ: Klausel
Kategorie: Deklaration

BYREF definiert, dass die Rückgabe der Funktion nicht als reiner Wert (*by value*) sondern als Referenz (*by reference*) erfolgen soll. Eine solche Funktion übergibt die Adresse des Rückgabewerts, statt eine Kopie davon zu erstellen und diese zu übergeben. Dadurch kann der Rückgabewert direkt über die Funktion geändert werden.

Wird BYREF nicht angegeben, übergibt die Funktion standardmäßig nur den Wert.

Funktionen, die BYREF zurückgeben, dürfen keine lokalen Variablen der Funktion zurückgeben, da diese nach dem Verlassen der Funktion gelöscht werden. Der Versuch, diese dann zu manipulieren, kann zu Abstürzen des Programms führen. Um dies zu verhindern, wird der Compiler bei dem Versuch eine Fehlermeldung ausgeben.

Soll der Wert der Rückgabe über die Funktion geändert werden, sollte '=>' verwendet werden. Siehe dazu den Referenzeintrag Zuweisung. Alternativ kann die Funktion in eine zusätzliche Klammer gesetzt werden (vgl. Beispiel 2). Dies ist nötig, damit der Compiler den Aufruf richtig lesen kann und ihn nicht für einen Vergleich hält.

Auch globale und Member-Operatoren können mit dieser Syntax BYREF zurückgeben, sofern sie wie eine Funktion aufgerufen werden.

Beispiel 1:

```
Function f ByRef As Const ZString
  Return "abcd"
End Function

Print f
Sleep
```

Hinweis: Wenn Beispiel 1 in UTF mit BOM gespeichert wird, interpretiert der Compiler den String-Literal als WSTRING, was in diesem Fall eine Compiler-Warnung und eine falsche Rückgabe hervorruft.

Beispiel 2:

```
Dim Shared As String s

Function f1 ByRef As String
  Return s
End Function

Function f2(ByRef _s As String) ByRef As String
  Return _s
End Function

s = "abcd"
Print s
```

```
f1 &= "efgh"
Print s

(f2(s)) &= "ijkl"
Print s

Sleep
```

Beispiel 3:

```
Function power2(ByRef _I As Integer) ByRef As Integer
  _I *= _I
  Return _I
End Function

Dim As Integer I = 2
power2( power2( power2( I ) ) )
Print I
Sleep
```

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC: existiert seit FreeBASIC 0.90

Siehe auch:
BYVAL (Rückgaben), FUNCTION, BYREF, FUNCTIONs, Parameterübergabe

Letzte Bearbeitung des Eintrags am 06.07.13 um 00:20:57
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

BYTE

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » B » **BYTE**
Typ: Datentyp

Daten vom Typ BYTE sind vorzeichenbehaftete 8-bit-Ganzzahlen.
Der Wertebereich für Variablen vom Typ BYTE erstreckt sich von -128 bis 127.

Unterschiede zu QB: neu in FreeBASIC

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht BYTE nicht zur Verfügung und kann nur über **__BYTE** aufgerufen werden.

Siehe auch:
DIM, CAST, CBYTE, Datentypen, Datentypen und Deklarationen

Letzte Bearbeitung des Eintrags am 28.12.12 um 00:42:31
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

BYVAL (Klausel)

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » B » **BYVAL (Klausel)**
Syntax: [DECLARE] { SUB | FUNCTION } Name (BYVAL parameter AS Typ, ...) [AS Typ]
Typ: Klausel
Kategorie: Deklaration

BYVAL legt in einer Parameterliste fest, dass ein Parameter 'by value', also als Wert statt als direkte Referenz übergeben werden soll. Die Klausel kann bei der Deklaration bzw. Definition einer Prozedur angegeben werden; siehe dazu der Referenzeintrag Parameterübergabe. Zur Syntax siehe auch DECLARE, SUB und FUNCTION.

Bei einer BYVAL übergebenen Variable wird beim Prozeduraufruf eine Kopie der Variablen erstellt und nur diese an die Prozedur weitergegeben. Wird innerhalb der Prozedur der Wert geändert, hat das keine rückwirkenden Auswirkungen auf die ursprünglich übergebene Variable. Im Gegensatz dazu übergibt BYREF die Variable als Referenz; eine Änderung der übergebenen Variablen innerhalb der Prozedur bleibt dann auch nach Ende der Prozedur erhalten.

In der aktuellen Version von FreeBASIC werden Zahlendatentypen (z. B. INTEGER, SINGLE) standardmäßig BYVAL übergeben; die Verwendung des Schlüsselwortes ist in diesem Fall überflüssig.

STRINGs und UDTs werden BYREF übergeben. BYVAL hat bei Strings eine andere Bedeutung: statt einer Kopie des Strings wird ein ZSTRING PTR auf den Stringinhalt übergeben. Der String sollte innerhalb der Prozedur nicht verändert werden, da die Adresse des Strings nicht aktualisiert wird. Das Verhalten von BYVAL bei String-Variablen kann sich in zukünftigen Versionen ändern; deshalb sollte man die Verwendung vermeiden und lieber ZSTRING einsetzen.

Arrays werden immer BYREF übergeben; eine Verwendung des Schlüsselwortes BYVAL führt bei Arrays zu einer Fehlermeldung.

Beachten Sie aber auch die Unterschiede unter den Dialektformen und zu früheren Compiler-Versionen!

Beispiel:

```
DECLARE SUB MySub (BYVAL a AS INTEGER, BYREF b AS INTEGER)

DIM AS INTEGER a = 1, b = 1

PRINT "Vor Subaufruf:"
PRINT "a = "; a
PRINT "b = "; b
PRINT

MySub a, b

PRINT "Nach Subaufruf:"
PRINT "a = "; a
PRINT "b = "; b
SLEEP

SUB MySub (BYVAL a AS INTEGER, BYREF b AS INTEGER)
   a += 1
   b += 1
```

```
   PRINT "In der Sub:"
   PRINT "a = "; a
   PRINT "b = "; b
   PRINT
END SUB
```

Ausgabe:

```
Vor Subaufruf:
a = 1
b = 1

In der Sub:
a = 2
b = 2

Nach Subaufruf:
a = 1
b = 2
```

Beispiel mit ZStrings:

```
Declare Sub MySub (ByVal a As ZString Ptr)

Dim As ZString Ptr a = @"normal"

Print "Vor Subaufruf:"
Print "a = "; *a
Print

MySub a

Print "Nach Subaufruf:"
Print "a = "; *a
Sleep

Sub MySub (ByVal a As ZString Ptr)
   a = @"veraendert"

   Print "In der Sub:"
   Print "a = "; *a
   Print
End Sub
```

Ausgabe:

```
Vor Subaufruf:
a = normal

In der Sub:
a = veraendert

Nach Subaufruf:
a = normal
```

Hinweis:
Seit FreeBASIC v0.90 kann BYVAL auch bei der Rückgabe per RETURN und FUNCTION verwendet werden.

Unterschiede zu QB:
In QB werden Variablen standardmäßig BYREF übergeben.

Unterschiede zu früheren Versionen von FreeBASIC:

- Seit FreeBASIC v0.90 wird BYVAL auch zusammen mit RETURN und 'FUNCTION = ' verwendet.
- Bis einschließlich FreeBASIC v0.16 wurden Variablen standardmäßig BYREF übergeben.

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb, -lang deprecated und -lang fblite werden Variablen standardmäßig BYREF übergeben.

Siehe auch:
BYVAL (Schlüsselwort), BYREF, Parameterübergabe, BYVAL (Rückgaben)

Letzte Bearbeitung des Eintrags am 04.07.13 um 23:05:20
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

BYVAL (Rückgaben)

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » B » **BYVAL (Rückgaben)**
Syntax:

```
RETURN BYVAL wert
FUNCTION = BYVAL wert
FunctionName = BYVAL wert
```

Typ: Klausel
Kategorie: Rückgabe

BYVAL spielt in dieser Verwendung nur dann eine Rolle, wenn die Funktion, in der es verwendet wird, eine BYREF-Rückgabe besitzt. Dadurch kann eine Adresse (üblicherweise ein Pointer) direkt zurückgegeben werden, was die Funktion dazu zwingt, diese Adresse zu referenzieren.

Beispiel:

```
Dim Shared i As Integer = 123

Function f ByRef As Integer
  Dim pi As Integer Ptr = @i

  Function = ByVal pi

  'oder mit RETURN:
  Return ByVal pi
End Function

Print i, f
Sleep
```

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC: existiert seit FreeBASIC 0.90

Siehe auch:
BYREF (Rückgaben), RETURN, FUNCTION, BYREF, FUNCTIONs, Parameterübergabe

Letzte Bearbeitung des Eintrags am 04.07.13 um 22:36:50
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

BYVAL (Schlüsselwort)

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » B » **BYVAL (Schlüsselwort)**
Syntax: OPTION BYVAL
Typ: Klausel
Kategorie: Programmoptionen

Mit dieser Zeile wird festgelegt, dass die Parameter, die an SUBs und FUNCTIONs übergeben werden, standardmäßig BYVAL behandelt werden sollen. Die Option kann **nur bis FreeBASIC v0.16** eingesetzt werden, oder in entsprechend höheren Versionen, die mit der Kommandozeilenoption -lang deprecated compiliert wurden! Wird mit FreeBASIC v0.17 unter der Option -lang fb compiliert, so ist OPTION BYVAL nicht mehr zulässig!

Ist OPTION BYVAL aktiviert, muss explizit die BYREF-Klausel angewandt werden, um Variablen per Referenz zu übergeben. Wird OPTION BYVAL nicht bestimmt, geht FreeBASIC davon aus, dass alle Parameter standardmäßig BYREF übergeben werden sollen. In diesem Fall werden Variablen außer Arrays und UDTs standardmäßig BYVAL übergeben; benutzen Sie explizit die BYREF-Klausel, um dem entgegen zu wirken.

Achtung: Die Verwendung von BYVAL mit variablen STRINGs kann zu Problemen führen, wenn dieser String das ASCII-Zeichen '0' enthält! Bei der Übergabe des Strings an einen BYVAL-deklarierten String-Parameter wird der String ab dem ASCII-Zeichen '0' abgeschnitten. Für die Übergabe von variablen Strings an Funktionen sollten daher BYREF oder Strings fester Länge verwendet werden.

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC:
Die Option ist nur bis FreeBASIC v0.16 erlaubt. Seit FreeBASIC v0.17 ist diese Option nur noch zulässig, wenn mit der Kommandozeilenoption -lang deprecated compiliert wurde.

Siehe auch:
BYVAL (Klausel), BYREF, OPTION, __FB_OPTION_BYVAL__, DECLARE, SUB, FUNCTION, Prozeduren, Parameterübergabe

Letzte Bearbeitung des Eintrags am 19.01.13 um 16:08:33
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

CALL

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » C » **CALL**
Syntax: CALL Prozedurname ([Parameterliste])
Typ: Anweisung
Kategorie: Programmablauf

CALL ruft eine Prozedur (SUB oder FUNCTION) auf. Bei einer FUNCTION wird der Rückgabewert ignoriert.

Achtung: CALL existiert nur noch aus Gründen der Rückwärtskompatibilität. Der Befehl kann nur in den Dialektformen -lang fblite, -lang deprecated und -lang qb verwendet werden. Es ist besser, CALL wegzulassen und die Prozedur direkt aufzurufen.

Die folgenden Beispiele funktionieren nicht in der Dialektform -lang fb

Beispiel 1:

```
"hlstring">"fblite"

Declare Sub foobar(ByVal x As Integer, ByVal y As Integer)
Declare Function wert As Integer
Call foobar(35, 42)
Call wert   ' Funktion aufrufen, aber Rückgabewert ignorieren
Sleep

Sub foobar(ByVal x As Integer, ByVal y As Integer)
  Print x; y
End Sub

Function wert As Integer
  wert = 42
End Function
```

Beide Male hätte CALL weggelassen werden können; die 5. und 6. Zeile des Programms lauteten dann:

```
foobar(35, 42)
wert
```

Damit wäre das Programm auch mit -lang fb compilierbar.

Beispiel 2: Prozeduraufruf vor der Deklaration

```
"hlstring">"qb"

Call mysub(15, 16)  ' "mysub" ist im Code zuvor noch nicht aufgetreten

Sub mysub(ByRef a As Integer, ByRef b As Integer)
  Print a, b
End Sub
```

Hinweis: Dieser Code funktioniert nur in -lang qb. Solange die Prozedur nicht deklariert wurde, findet außerdem keine Typ-Überprüfung der übergebenen Parameter statt. Der Programmierer muss selbst sicherstellen, dass es sich um die korrekten Typen handelt.

Unterschiede zu QB:
CALL erstellt in QB eine Kopie aller Parameter. Wenn die Parameter innerhalb der aufgerufenen Prozedur verändert werden, hat dies keine Auswirkung auf den Wert der Parameter außerhalb der Prozedur.

Unterschiede unter den FB-Dialektformen:

- In der Dialektform -lang fb (Standard) existiert CALL nicht.
- Der Aufruf noch nicht deklarierter SUBs ist nur in der Dialektform -lang qb möglich.

Siehe auch:
DECLARE, SUB, FUNCTION, Prozeduren

Letzte Bearbeitung des Eintrags am 26.12.12 um 23:59:42
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

CALLOCATE

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » C » **CALLOCATE**
Syntax: CALLOCATE (Anzahl [, Größe])
Typ: Funktion
Kategorie: Speicher

CALLOCATE reserviert einen Bereich im Speicher (Heap), setzt alle seine Bytes auf 0 (im Gegensatz zu ALLOCATE!)

- 'Anzahl' ist die Anzahl der zu reservierenden Speicherstellen.
- 'Größe' ist die Größe der Speicherstelle in Bytes. Wird 'Größe' ausgelassen, nimmt FreeBASIC automatisch 1 an. CALLOCATE kann dann wie ALLOCATE verwendet werden, mit dem Unterschied, dass alle reservierten Bytes auf null gesetzt sind.
- Der Rückgabewert ist ein Pointer auf den Anfang des reservierten Bereichs. Wenn beim Allozieren ein Fehler auftritt, ist der Rückgabewert 0 (Null-Pointer).

CALLOCATE ist kein Teil der FreeBASIC Runtime Library, sondern ein Alias von calloc der C-Lib.
Achtung: Es kann nicht garantiert werden, dass diese Funktion auf allen Plattformen Multithreading unterstützt, d.h. thread-safe ist. Unter Windows und Linux sind aktuell durch die verwendeten Implementationen der Betriebssysteme aber keine Probleme zu erwarten.

Beispiel:

```
' Einen INTEGER Pointer erstellen
DIM p AS INTEGER PTR

' Speicherplatz für zehn INTEGER-Werte reservieren
p = CALLOCATE(10, LEN(INTEGER))

' Den Speicherbereich mit zehn Zahlen belegen
FOR fill_p AS INTEGER = 0 TO 9
  p&"hlzeichen">] = fill_p
NEXT

' Die Zahlen ausgeben
FOR show_p AS INTEGER = 0 TO 9
  PRINT p&"hlzeichen">]
NEXT

' Variable wieder freigeben
DEALLOCATE p

' auf Tastendruck warten und beenden
SLEEP
END
```

Unterschiede zu QB: neu in FreeBASIC

Plattformbedingte Unterschiede:
Es kann nicht garantiert werden, dass die Prozedur auf allen Plattformen thread-safe ist.

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht CALLOCATE nicht zur Verfügung und kann nur über **__CALLOCATE**

aufgerufen werden.

Siehe auch:
ALLOCATE, REALLOCATE, DEALLOCATE, POINTER, Speicher

Letzte Bearbeitung des Eintrags am 05.04.14 um 15:53:41
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

CASE

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » C » **CASE**
Siehe SELECT CASE

Letzte Bearbeitung des Eintrags am 28.08.11 um 18:06:49
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

CAST

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » C » **CAST**
Syntax: CAST (Datentyp, Ausdruck)
Typ: Operator
Kategorie: Typumwandlung

CAST konvertiert einen Ausdruck in einen beliebigen anderen Typ. Er löst damit alle C###-Befehle (CINT, CDBL, ...) ab.

'Datentyp' ist ein beliebiger Datentyp, inklusive Pointern. Auch ZSTRING- und WSTRING-Pointer sind erlaubt. STRINGs, ZSTRINGs oder WSTRINGs als Datentyp führen jedoch zu ungültigen Ergebnissen.

'Ausdruck' ist ein beliebiger Ausdruck. Soll ein String in einen Zahlendatentyp umgewandelt werden, wird dazu eine passende Funktion aufgerufen, z. B. VALINT zur Umwandlung in ein INTEGER.

CAST kann mithilfe von OPERATOR für verschiedene Datentypen überladen werden.

Beispiel:

```
DIM i AS INTEGER, ip AS INTEGER PTR
DIM b AS BYTE, bp AS BYTE PTR

i = &h0080
b = CAST(BYTE, i)

ip = @i
bp = CAST(BYTE PTR, ip)

PRINT i, b
PRINT *ip, *bp
```

Ausgabe:

```
 128     -128
 128     -128
```

Unterschiede zu QB: neu in FreeBASIC

Siehe auch:
CPTR, CSIGN, CUNSG, Datentypen umwandeln

Letzte Bearbeitung des Eintrags am 01.06.12 um 01:00:33
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

CBYTE

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » C » **CBYTE**
Syntax: CBYTE (Ausdruck)
Typ: Funktion
Kategorie: Typumwandlung

CBYTE konvertiert einen beliebigen numerischen Ausdruck zu einem BYTE. Es erfüllt dieselbe Funktion wie CAST(BYTE,Ausdruck).

Bei Bedarf wird die Zahl mathematisch gerundet: Ist der Nachkommawert größer als .5, dann wird aufgerundet; ist er kleiner als .5, dann wird abgerundet. Ist der Nachkommawert genau .5, dann wird zur nächstliegenden *geraden* Zahl gerundet.

Handelt es sich bei dem Ausdruck um einen STRING, dann wird dieser mit der Funktion VALINT umgewandelt. Dabei wird nicht gerundet, sondern die Nachkommastellen werden abgeschnitten.

Beispiel:

```
PRINT CBYTE(260)
```

Ausgabe:

```
4
```

Unterschiede zu QB: neu in FreeBASIC

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht CBYTE nicht zur Verfügung und kann nur über **__CBYTE** aufgerufen werden.

Siehe auch:
CAST, CUBYTE, CSHORT, CUSHORT, CINT, CUINT, CLNG, CLNGINT, CULNGINT, CSNG, CDBL, CSIGN, CUNSG, Datentypen umwandeln

Letzte Bearbeitung des Eintrags am 01.06.12 um 01:47:50
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

CDBL

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » C » **CDBL**
Syntax: CDBL(Ausdruck)
Typ: Funktion
Kategorie: Typumwandlung

CDBL konvertiert einen beliebigen numerischen Ausdruck in eine DOUBLE-Zahl (64 Bit).

Unterschiede zu QB:
In QB darf 'Ausdruck' kein String sein.

Siehe auch:
CAST, CBYTE, CUBYTE, CSHORT, CUSHORT, CINT, CUINT, CLNG, CLNGINT, CULNGINT, CSNG, CSIGN, CUNSG, Datentypen umwandeln

Letzte Bearbeitung des Eintrags am 01.06.12 um 01:37:12
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

CDECL

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » C » **CDECL**
Syntax: [DECLARE] {SUB | FUNCTION} prozedurname CDECL [evtl. weitere Angaben ...]
Typ: Schlüsselwort
Kategorie: Bibliotheken

Der Name der CDECL-Aufrufkonvention steht für C-DECLARE. Die Konvention wird von vielen C/C++ -Compilern verwendet, die auf der x86-Architektur laufen. Hierbei werden die Parameter nacheinander von rechts nach links (<-) auf den Stack gelegt.

Die aufgerufende Prozedur baut den Stack nicht selbst ab, sodass Prozeduren mit variabler Argumenten-Anzahl und -Länge realisiert werden können (siehe ...). Der Stack muss nach dem Aufruf von der aufrufenden Prozedur bereinigt werden.

CDECL ist die Aufrufkonvention, die standardmäßig unter Linux, BSD und DOS verwendet wird, sofern keine andere Konvention explizit festgelegt oder durch einen EXTERN-Block impliziert wird.

Beispiel: Vergleich der verschiedenen Aufrufmöglichkeiten

```
Sub s StdCall (s1 As Integer, s2 As Integer)
  Print "StdCall ", s1, s2
End Sub
Sub c Cdecl (c1 As Integer, c2 As Integer)
  Print "Cdecl ", c1, c2
End Sub
Sub p Pascal (p1 As Integer, p2 As Integer)
  Print "Pascal ", p1, p2
End Sub


Asm
  push 2 '2. Parameter - s2
  push 1 '1. Parameter - s1
  Call s 'rechts nach links

  push 2 '2. Parameter - c2
  push 1 '1. Parameter - c1
  Call c 'rechts nach links
  Add esp, 8 'der Stack muss durch die aufrufende Funktion aufgeräumt
werden

  push 1 '1. Parameter - p1
  push 2 '2. Parameter - p2
  Call p 'links nach rechts
End Asm

Sleep
```

Unterschiede zu QB: neu in FreeBASIC

Siehe auch:
DECLARE, STDCALL, PASCAL, Module (Library / DLL)

Letzte Bearbeitung des Eintrags am 29.08.12 um 22:24:20
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

CHAIN

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » C » **CHAIN**
Syntax: CHAIN (Programm)
Typ: Funktion
Kategorie: System

CHAIN übergibt die Kontrolle an ein anderes Programm und startet dieses. Nachdem das aufgerufene Programm beendet wurde, erhält das Originalprogramm die Kontrolle zurück.

- 'Programm' ist der Dateiname (eventuell mit Pfadangabe) des Programms, das aufgerufen werden soll.
- Der Rückgabewert ist der vom aufgerufenen Programm erzeugte Errorlevel (siehe END). Konnte kein Programm aufgerufen werden (z. B. weil die angegebene Datei nicht existiert), so wird -1 zurückgegeben. Beachten Sie, dass auch der vom Programm zurückgegebene Wert -1 sein könnte!

Beispiel:
Um dieses Beispiel zu starten, muss im aktuellen Arbeitsverzeichnis die Datei "program" bzw. "program.exe" vorhanden sein. Wenn sie nicht vorhanden ist, wird nichts aufgerufen, und CHAIN gibt -1 zurück.

```
"hlkw0">__FB_UNIX__
  Dim As String program = "./program"
"hlkw0">__FB_PCOS__
  Dim As String program = "program.exe"
"hlkw0">Print "Starte " & program & " ..."
If (Chain(program) = -1) Then
  Print program & " nicht gefunden!"
End If

Sleep
```

Unterschiede zu QB:
CHAIN kann in FreeBASIC als Funktion eingesetzt werden.

Plattformbedingte Unterschiede:

- Unter Linux muss der Dateiname 'case sensitive' erfolgen. Windows und DOS sind 'case insensitive'.
- Das Trennzeichen für den Dateipfad ist unter Linux der vorwärtsgerichtete Slash /. Windows verwendet den Backslash \, erlaubt aber auch den Slash. DOS verwendet den Backslash.

Siehe auch:
RUN, EXEC, SHELL, END, COMMAND, Betriebssystem-Anweisungen

Letzte Bearbeitung des Eintrags am 27.12.12 um 00:02:22
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

CHDIR

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » C » **CHDIR**
Syntax: CHDIR (Pfadangabe)
Typ: Funktion
Kategorie: System

CHDIR (CHange DIRectory) ändert das aktuelle Arbeitsverzeichnis oder -laufwerk.

- 'Pfadangabe' ist ein String, der das Verzeichnis identifiziert, in das gewechselt werden soll. Die Pfadangabe kann absolut oder relativ zum aktuellen Arbeitsverzeichnis sein (siehe CURDIR).
- Der Rückgabewert ist entweder 0, wenn der Wechsel erfolgreich war, oder -1, wenn ein Fehler aufgetreten ist.

Auch wenn die absolute Angabe ein anderes Laufwerk enthält als das Ausgangsverzeichnis, wird der Wechsel wie gewohnt durchgeführt. Ein expliziter Wechsel des Laufwerks ist nicht nötig.

Beispiel:

```
PRINT CURDIR

' Wechsel in das übergeordnete Verzeichnis
CHDIR ".."
PRINT CURDIR

' Wechsel mit absoluter Pfadangabe
"hlkw0">__FB_WIN32__
  CHDIR "C:\WINDOWS"
"hlkw0">__FB_UNIX__
  CHDIR "/usr/share"
"hlkw0">PRINT CURDIR

' zum Diskettenlaufwerk wechseln
' (Laufwerksbuchstaben existieren nicht in unixartigen Betriebssystemen)
"hlkw0">__FB_PCOS__
  IF CHDIR("A:\") THEN
     PRINT "Laufwerk A: nicht bereit"
  ELSE
     PRINT CURDIR
  END IF
"hlkw0">SLEEP
```

mögliche Ausgabe:

```
C:\BASIC\FreeBASIC
C:\BASIC
C:\WINDOWS
Laufwerk A: nicht bereit
```

Unterschiede zu QB:

- In QB kann CHDIR nicht zwischen zwei Laufwerken wechseln.
- In FreeBASIC kann CHDIR als Funktion eingesetzt werden.

Plattformbedingte Unterschiede:

- Unter Linux muss der Dateiname 'case sensitive' erfolgen. Windows und DOS sind 'case insensitive'.
- Das Trennzeichen für den Dateipfad ist unter Linux der vorwärtsgerichtete Slash /. Windows verwendet den Backslash \, erlaubt aber auch den Slash. DOS verwendet den Backslash.

Siehe auch:
RMDIR, MKDIR, CURDIR, SHELL, Betriebssystem-Anweisungen.

Letzte Bearbeitung des Eintrags am 15.06.12 um 22:24:28
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

CHR

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » C » **CHR**
Syntax: CHR[$] (ASCII_Wert [, ASCII_Wert_2 [...]])
Typ: Funktion
Kategorie: Stringfunktionen

CHR verwandelt einen ASCII-Code in seinen Character. Das Dollarzeichen ($) als Suffix ist optional.

CHR wird z. B. verwendet, um Zeichen darzustellen, die nicht auf der Tastatur sind. Wollen Sie dasselbe Zeichen mehrmals hintereinander ausgeben, sollten Sie stattdessen STRING (Funktion) benutzen.

Beispiel:

```
PRINT "ASCII-Character 127: "; CHR(127)
PRINT "Codes 65, 66 und 67: "; CHR(65, 66, 67)
```

Liste interessanter CHR-Codes:
Die Funktionstasten werden meist über zwei Byte angesprochen, von denen das erste den Wert 255 hat (im Gegensatz zu QBASIC, wo es der Wert 0 ist; vergleiche dazu INKEY)

```
F-Tasten:
CHR(255, 59)  = F1
CHR(255, 60)  = F2
CHR(255, 61)  = F3
CHR(255, 62)  = F4
CHR(255, 63)  = F5
CHR(255, 64)  = F6
CHR(255, 65)  = F7
CHR(255, 66)  = F8
CHR(255, 67)  = F9
CHR(255, 68)  = F10
CHR(255, 133) = F11
CHR(255, 134) = F12

Pfeiltasten:
CHR(255, 72)  = Rauf
CHR(255, 75)  = Links
CHR(255, 77)  = Rechts
CHR(255, 80)  = Runter

Weitere:
CHR(255, 71)  = Pos1 (oder auch Home)
CHR(255, 73)  = Bild auf
CHR(255, 79)  = Ende
CHR(255, 81)  = Bild ab
CHR(255, 82)  = Einfügen
CHR(255, 83)  = Entfernen
CHR(13,  10)  = Zeilenumbruch (unter Windows)
```

Unterschiede zu QB:
CHR kann in FreeBASIC mehr als ein Zeichen auf einmal zurückgeben.

Unterschiede unter den FB-Dialektformen:

- In der Dialektform -lang qb ist das Suffix $ verbindlich.
- In den Dialektformen -lang fblite und -lang fb ist das Suffix optional.

Siehe auch:
WCHR, ASC, STRING (Funktion), Datentypen umwandeln

Weitere Informationen zu CHR-Codes:
ASCII-Tabelle anzeigen lassen

Letzte Bearbeitung des Eintrags am 14.03.14 um 22:40:29
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

CINT

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » C » **CINT**
Syntax:

```
CINT (Ausdruck)
CINT<Bits> (Ausdruck)
```

Typ: Funktion
Kategorie: Typumwandlung

CINT konvertiert einen beliebigen numerischen Ausdruck zu einem INTEGER oder INTEGER<Bits>. Es erfüllt dieselbe Funktion wie CAST(INTEGER, Ausdruck) bzw. CAST(INTEGER<Bits>, Ausdruck).

Bei Bedarf wird die Zahl mathematisch gerundet: Ist der Nachkommawert größer als .5, dann wird aufgerundet; ist er kleiner als .5, dann wird abgerundet. Ist der Nachkommawert genau .5, dann wird zur nächstliegenden *geraden* Zahl gerundet.

Der Parameter 'Bits' gibt an, welche Größe der zurückgegebene INTEGER<Bits> besitzen soll. Die erlaubten Werte sind 8, 16, 32 und 64.

Handelt es sich bei dem Ausdruck um einen STRING, dann wird dieser mit der Funktion VALINT umgewandelt. Dabei wird nicht gerundet, sondern die Nachkommastellen werden abgeschnitten.

Beispiel:

```
PRINT CINT(123.45)      ' gibt 123 aus
PRINT CINT<8>(123.45)   ' gibt 123 aus
PRINT CINT(-56.78)      ' gibt -57 aus
PRINT CINT(457.5)       ' gibt 458 aus
PRINT CINT(456.5)       ' gibt 456 aus
PRINT CINT("457.5")     ' gibt 457 aus
SLEEP
```

Unterschiede zu QB:

- QB unterstützt den Parameter 'Bits' nicht. INTEGER sind dort immer 16 Bit lang.
- Die Übergabe eines STRINGs ist in QB nicht erlaubt.

Unterschiede zu früheren Versionen von FreeBASIC:
Der Parameter 'Bits' existiert seit FreeBASIC v0.90.

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb gibt CINT eine 16-Bit-Zahl zurück.

Siehe auch:
CAST, CBYTE, CUBYTE, CSHORT, CUSHORT, CUINT, CLNG, CLNGINT, CULNGINT, CSNG, CDBL, CSIGN, CUNSG, Datentypen umwandeln

Letzte Bearbeitung des Eintrags am 04.07.13 um 17:15:18
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

CIRCLE

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » C » **CIRCLE**
Syntax: CIRCLE [Puffer,] [STEP] (x,y), Radius[, [Farbe][, [Anfangswinkel][, [Endwinkel][, [Aspekt][, F]]]]]
Typ: Anweisung
Kategorie: Grafik

CIRCLE zeichnet Kreise, Ellipsen oder Bögen.

- 'Puffer' ist ein Speicherbereich wie ein mit IMAGECREATE erstellter Puffer oder ein Array. Beide können mit PUT (Grafik) angezeigt werden.
- STEP gibt an, dass die Koordinaten relativ zur aktuellen Grafikcursorposition sind.
- 'x' und 'y' sind die Koordinaten des Mittelpunkts des Kreises/Bogens/der Ellipse.
- 'Radius' gibt den Radius des Kreises/Bogens an, bzw. den größten Radius der Ellipse.
- 'Farbe' ist die Farbnummer. Sie ist modusspezifisch. Siehe dazu COLOR (Anweisung) und SCREENRES.
- 'Anfangswinkel' und 'Endwinkel' sind Winkel im Bogenmaß. Sie liegen in einem Bereich zwischen -2 * Pi und 2 * Pi. Wenn Sie einen negativen Winkel angeben, wird er in einen Positiven umgewandelt, und CIRCLE zeichnet eine Linie vom Kreismittelpunkt bis zum Bogenende (so können Sie z.B. Kuchendiagramme erstellen). 'Endwinkel kann kleiner sein als 'Anfangswinkel'.
  Wenn Sie keinen Start- und Endwinkel angeben, wird ein(e) ganze(r) Kreis/Ellipse gezeichnet. Wenn Sie nur den Anfangswinkel angeben, wird für den Endwinkel 2 * Pi angenommen; wenn Sie nur den Endwinkel angeben, wird für den Anfangswinkel 0 angenommen.
- 'Aspekt' ist das Verhältnis zwischen y-Radius und x-Radius. Der Standard-Aspekt ist der, der nötig ist, um einen perfekten Kreis auf dem Bildschirm zu zeichnen. Dieser Wert kann wie folgt berechnet werden:
  ratio = (y_radius / x_radius) * pixel_aspect_ratio
  pixel_aspect_ratio ist das Verhältnis der Breite des aktuellen Bildschirmmodus zu seiner Höhe (normalerweise 4:3).
- 'F' ist das Ausfüllen-Flag. Wenn Sie dieses Flag setzen, wird der Kreis bzw. die Ellipse in der gewählten Farbe ausgefüllt. Das Ausfüllen-Flag kann nicht genutzt werden, wenn 'Anfangswinkel' und/oder 'Endwinkel' angegeben wurden; 'Kuchendiagramme' können nicht alleine mit CIRCLE dargestellt werden.

Achtung: Bei der Verwendung von SCREEN und ohne Angabe von 'Aspekt=1.0' wird bei einigen Bildschirmmodi statt eines Kreises eine Ellipse angezeigt. Unter SCREENRES tritt dieses Problem nicht auf.

Benutzerdefinierte Koordinatensysteme, die mit WINDOW und/oder VIEW (Grafik) eingestellt wurden, wirken sich auf das Zeichnen des Kreises aus (das gilt auch für das Clipping; ein Kreis wird nur dann gezeichnet, wenn er sich innerhalb der logischen Bildschirmgrenzen befindet). Nachdem CIRCLE den Zeichenvorgang abgeschlossen hat, wird die Position des Grafikcursors auf den Kreismittelpunkt gesetzt.

Beispiel:

```
SCREENRES 640, 480                          ' Grafikfenster initialisieren
CIRCLE (320, 240), 200, 15                  ' zeichnet einen weißen Kreis
CIRCLE (320, 240), 200, 2, , , 0.2, F       ' zeichnet eine ausgefüllte
Ellipse
CIRCLE (320, 240), 200, 4, 0.83, 1.67, 3    ' zeichnet einen kleinen Bogen
SLEEP
```

Unterschiede zu QB:

- QB und FreeBASIC benutzen verschiedene Algorithmen, um Kreise, Bögen und Ellipsen zu zeichnen. Daher sind die von FreeBASIC erzeugten Bilder möglicherweise nicht pixelgenau identisch zu den von QB erzeugten.
- Die Möglichkeit, ausgefüllte Kreise und Ellipsen zu erzeugen, ist neu in FreeBASIC.
- In FreeBASIC ist es möglich, in einen Datenpuffer zu zeichnen.

Siehe auch:
SCREENRES, Grafik

Letzte Bearbeitung des Eintrags am 27.12.12 um 00:03:12
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

CLASS

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » C » **CLASS**
Syntax: CLASS Klassenname
Typ: Anweisung
Kategorie: Klassen

Aktuell gibt es CLASS noch nicht, befindet sich aber in Planung.
Die Verwendung wird grob der eines UDTs entsprechen, mit dem Unterschied, dass alle Records standardmäßig PRIVATE statt PUBLIC sein werden.

Beispiel:

```
' kein Beispiel möglich, da noch nicht implementiert
```

Siehe auch:
Objektorientierung

Letzte Bearbeitung des Eintrags am 27.12.12 um 00:04:59
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

CLEAR

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » C » **CLEAR**
Syntax: CLEAR Bezeichner, [Wert], Bytes
Typ: Anweisung
Kategorie: Speicher

CLEAR setzt eine Anzahl an Bytes ab einer bestimmten Adresse auf einen angegebenen Wert. Der Befehl kann z. B. dazu verwendet werden, um alle Elemente eines Arrays auf einen bestimmten Wert zu setzen.

- 'Bezeichner' ist der Bezeichner einer Variablen, eines Arrays oder das Ziel eines Pointers (*Pointer), dessen Werte reinitialisiert werden sollen. Dieser Parameter wird BYREF übergeben.
- 'Wert' ist der Wert, auf den alle Bytes des Speicherbereichs gesetzt werden sollen. Wird 'Wert' ausgelassen, nimmt FreeBASIC automatisch 0 an.
- 'Bytes' ist die Anzahl der Bytes, die reinitialisiert werden sollen. Soll ein Array komplett reinitialisiert werden, errechnet sich 'Bytes' nach dieser Formel:
  Bytes = LEN(Array(ersterIndex)) * (UBOUND(Array) - LBOUND(Array) + 1)
  (also die Größe des Datentyps multipliziert mit der Länge des Arrays)

Hinweis: Wird als 'Bezeichner' ein Pointer angegeben, so muss dieser zuerst dereferenziert werden. Ansonsten wird CLEAR versuchen, den Zeiger selbst zu überschreiben anstelle des Speichers, auf den der Zeiger zeigt.
Auch Strings fester Länge können übergeben werden. Bei Strings variabler Länge kann CLEAR nicht eingesetzt werden, da auch in diesem Fall der interne Zeiger auf die Daten überschrieben werden würde.

Beispiel:

```
DIM array(1 TO 100) AS INTEGER
DIM AllocArea AS BYTE PTR
DIM IntVar AS INTEGER
DIM text AS STRING*5 = "12345"

AllocArea = ALLOCATE(100)
IntVar = 5

CLEAR array(1), , LEN(array(1)) * (UBOUND(array) - LBOUND(array) + 1)
CLEAR *AllocArea, 1, 100
CLEAR IntVar, 255, LEN(INTEGER)
CLEAR text, 33, 3

PRINT array(1)
PRINT AllocArea&"hlzahl">0]
PRINT IntVar
PRINT text
PRINT
SLEEP
```

Ausgabe:

```
 0
 1
-1
!!!45
```

Erläuterung: Alle vier Bytes der INTEGER-Stelle 'IntVar' werden einzeln mit dem Wert 255 befüllt; deshalb hat die Variable nach dem CLEAR-Aufruf den Wert -1. Beim String werden die ersten drei Zeichen auf den ASCII-Wert 33 (Ausrufezeichen) gesetzt.

Unterschiede zu QB:
CLEAR hat in QB eine andere Bedeutung. Es wird dazu verwendet, alle Variablen auf null zu setzen und die Stack-Größe zu setzen.

Siehe auch:
DIM, ERASE, RESET, Speicher

Letzte Bearbeitung des Eintrags am 27.12.12 um 00:05:24
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

CLNG

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » C » **CLNG**
Syntax: CLNG (Ausdruck)
Typ: Funktion
Kategorie: Typumwandlung

CLNG verwandelt einen beliebigen numerischen Ausdruck zu einem LONG. Es erfüllt dieselbe Funktion wie CAST(LONG,Ausdruck).

Bei Bedarf wird die Zahl mathematisch gerundet: Ist der Nachkommawert größer als .5, dann wird aufgerundet; ist er kleiner als .5, dann wird abgerundet. Ist der Nachkommawert genau .5, dann wird zur nächstliegenden *geraden* Zahl gerundet.

Handelt es sich bei dem Ausdruck um einen STRING, dann wird dieser mit der Funktion VALINT umgewandelt. Dabei wird nicht gerundet, sondern die Nachkommastellen werden abgeschnitten.

Beispiel:

```
PRINT CLNG(123.45)   ' gibt 123 aus
PRINT CLNG(-56.78)   ' gibt -57 aus
PRINT CLNG(457.5)    ' gibt 458 aus
PRINT CLNG(456.5)    ' gibt 456 aus
PRINT CLNG("457.5") ' gibt 457 aus
```

Unterschiede zu QB:
Die Übergabe eines STRINGs ist in QB nicht erlaubt.

Siehe auch:
CAST, CBYTE, CUBYTE, CSHORT, CUSHORT, CINT, CUINT, CLNGINT, CULNGINT, CSNG, CDBL, CSIGN, CUNSG, Datentypen umwandeln

Letzte Bearbeitung des Eintrags am 01.02.14 um 01:14:43
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

CLNGINT

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » C » **CLNGINT**
Syntax: CLNGINT (Ausdruck)
Typ: Funktion
Kategorie: Typumwandlung

CLNGINT verwandelt einen beliebigen numerischen Ausdruck zu einem LONGINT. Es erfüllt dieselbe Funktion wie CAST(LONGINT,Ausdruck).

Bei Bedarf wird die Zahl mathematisch gerundet: Ist der Nachkommawert größer als .5, dann wird aufgerundet; ist er kleiner als .5, dann wird abgerundet. Ist der Nachkommawert genau .5, dann wird zur nächstliegenden *geraden* Zahl gerundet.

Handelt es sich bei dem Ausdruck um einen STRING, dann wird dieser mit der Funktion VALINT umgewandelt. Dabei wird nicht gerundet, sondern die Nachkommastellen werden abgeschnitten.

Beispiel:

```
PRINT CLNGINT(12345678.45)    ' gibt 12345678 aus
PRINT CLNGINT(-12345678.901) ' gibt -12345679 aus
PRINT CLNGINT(123457.5)       ' gibt 123458 aus
PRINT CLNGINT(123456.5)       ' gibt 123456 aus
PRINT CLNGINT("123457.5")     ' gibt 123457 aus
```

Unterschiede zu QB: neu in FreeBASIC

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht CLNGINT nicht zur Verfügung und kann nur über **__CLNGINT** aufgerufen werden.

Siehe auch:
CAST, CBYTE, CUBYTE, CSHORT, CUSHORT, CINT, CUINT, CLNG, CULNGINT, CSNG, CDBL, CSIGN, CUNSG, Datentypen umwandeln

Letzte Bearbeitung des Eintrags am 14.06.13 um 23:28:30
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

CLOSE

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » C » **CLOSE**
Syntax: CLOSE [([#]n [, [#]m [, [#]...]])]
Typ: Funktion
Kategorie: Dateien

CLOSE schließt eine, mehrere oder alle Dateien, die zuvor mit OPEN geöffnet wurden. Bei Programmende werden automatisch alle noch geöffneten Dateien geschlossen; dennoch gehört es zum guten Programmierstil, geöffnete Dateien wieder zu schließen.

- #n, #m, #... sind die Dateinummern, die beim Öffnen mit OPEN zugewiesen wurden.
- Um alle geöffneten Dateien zu schließen, geben Sie einfach keine Dateinummern an.
- Die Rauten (#) sind optional, sollten aber benutzt werden, um Verwechslungen auszuschließen.
- Der Rückgabewert ist ein INTEGER. Bei Erfolg wird 0 zurückgegeben, ansonsten die Fehlernummer.

Der Rückgabewert kann auch verworfen werden; CLOSE wird dann wie eine Anweisung eingesetzt.

Beispiel:

```
OPEN "file1" FOR INPUT AS "hlkw0">OPEN "file2" FOR INPUT AS "hlkw0">OPEN
"file3" FOR INPUT AS "hlkw0">OPEN "file4" FOR INPUT AS "hlkw0">OPEN
"file5" FOR INPUT AS "hlkommentar">' eine Datei schließen; Einsatz als
Funktion
IF CLOSE ("hlzahl">2) <> 0 THEN
  PRINT "Fehler: Datei "
END IF

' mehrere Dateien schließen; Rückgabewert wird verworfen
CLOSE "hlzeichen">, #1

' alle geöffneten Dateien (#3 und #5) schließen
CLOSE
SLEEP
```

Unterschiede zu QB:

- In QB kann CLOSE nicht als Funktion eingesetzt werden.
- In FreeBASIC ergibt CLOSE einen Fehler, wenn versucht wird, eine nicht verwendete Dateinummer zu schließen.

Unterschiede zu früheren Versionen von FreeBASIC:
Bis einschließlich Version 0.22 konnte auch 0 als Dateinummer verwendet werden, um alle geöffneten Dateien zu schließen. Der Versuch erzeugt nun eine Fehlermeldung.

Siehe auch:
OPEN (Anweisung), Dateien (Files)

Letzte Bearbeitung des Eintrags am 08.06.12 um 17:13:39
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

CLS

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » C » **CLS**
Syntax: CLS [Parameter]
Typ: Anweisung
Kategorie: Konsole

CLS löscht den Konsolen- bzw. Grafik-Bildschirm. Der gelöschte Bereich wird mit der Hintergrundfarbe aufgefüllt (siehe COLOR)

'Parameter' gibt an, welche Teile des Bildschirms gelöscht werden sollen. Er ist einer von diesen Werten:

- **0** löscht den gesamten Bildschirm.
- **1** löscht das durch VIEW (Grafik) festgelegte Darstellungsfeld, oder nichts im Konsolenmodus.
- **2** löscht den durch VIEW (Text) festgelegten Textanzeigebereich, sowohl im Grafik- als auch im Konsolenmodus.
- Wird der Parameter ausgelassen, dann wird das grafische Darstellungsfeld gelöscht, sofern zuvor VIEW (Grafik) verwendet wurde, ansonsten wird der durch VIEW (Text) festgelegte Textanzeigebereich gelöscht. Wurde keiner der beiden Bereiche festgelegt, dann löscht CLS den gesamten Bildschirm.

Beispiel 1:

```
Color 7, 1      ' hellgrau auf blau
Cls             ' Bildschirm löschen und auf die Hintergrundfarbe setzen
Locate 12, 30
Print "Hallo FreeBASIC-Welt!"
```

Beispiel 2:
Um im Grafikmodus den gesamten Bildschirm auf einen Farbwert zu setzen, kann es schneller sein, statt der Verwendung von CLS den Bildschirmspeicher mit CLEAR auf den gewünschten Farbwert zu setzen.

```
Dim scrbuf As Byte Ptr, scrsize As Integer
Dim As Integer scrhei, scrpitch
Dim As Integer radius = 0, richtung = 1

ScreenRes 640, 480, 8

scrbuf = ScreenPtr: Assert( scrbuf <> 0 )
ScreenInfo( , scrhei, , , scrpitch )
scrsize = scrpitch * scrhei

Do
  ScreenLock                ' Bildschirm sperren (nötig für
Direktzugriff)
  Clear *scrbuf, 0, scrsize  ' Bildschirm löschen
  Circle (320, 240), radius  ' Kreis zeichnen
  ScreenUnlock
  radius += richtung         ' Radius vergrößern oder verkleinern
  If radius <= 0 Then
    richtung = 1
  ElseIf radius >= 100 Then
    richtung = -1
  End If
```

```
  Sleep 1                     ' kleine Pause für den Prozessor
Loop Until Len(Inkey) > 0     ' läuft solange, bis eine Taste gedrückt
wird
```

Siehe auch:
VIEW (Grafik), VIEW (Text), Grafik, Konsole

Letzte Bearbeitung des Eintrags am 27.12.12 um 00:06:10
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

COLOR (Anweisung)

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » C » **COLOR (Anweisung)**
Syntax: COLOR [Vordergrund][,Hintergrund]
Typ: Anweisung
Kategorie: Konsole

COLOR setzt die Vorder- und Hintergrundfarbe für Ausgaben auf der Konsole oder auf dem Grafikfenster.

- 'Vordergrund' und 'Hintergrund' sind die neuen Farbnummern, die verwendet werden sollen. Welche Farbe welche Nummer hat, hängt vom aktuellen Bildschirmmodus ab (siehe Tabelle unten).
- 'Vordergrund' wird von allen Grafikanweisungen außer PRESET (Grafik) (siehe Liste unten) als Standard-Linienfarbe verwendet. Bei den Text-Anweisungen wird sie als Text-Vordergrundfarbe verwendet.
- 'Hintergrundfarbe' wird von PRESET (Grafik) als Standard-Linienfarbe verwendet. Bei den Text-Anweisungen wird sie als Hintergrundfarbe verwendet.
- CLS füllt den Bildschirm in 'Hintergrundfarbe' aus.
- Wird einer der beiden Parameter ausgelassen, behält FreeBASIC den zuvor eingestellten Farbwert bei. COLOR ohne Parameter hat keine Auswirkung.
- Im Textmodus ist standardmäßig 7 (helles grau) als Vordergrundfarbe, und 0 (schwarz) als Hintergrundfarbe eingestellt. In Grafikmodi ist standardmäßig ein helles weiß als Vordergrundfarbe und schwarz als Hintergrundfarbe eingestellt.

Diese Anweisungen benutzen die zuletzt mit COLOR gesetzten Farben, wenn kein anderer Wert angegeben wird:

Grafik-Anweisungen:

- PSET (Grafik)
- PRESET (Grafik)
- LINE (Grafik)
- CIRCLE
- DRAW (Grafik)
- DRAW STRING
- PAINT
- CLS

Text-Anweisungen:

- PRINT (Anweisung)
- INPUT (Anweisung)

Wurde das Grafikfenster mit SCREEN (Anweisung) initialisiert, dann sind die Farbnummern und die zugehörige Farbe abhängig vom gewählten Bildschirmmodus (gültig bis maximal 8bpp Farbtiefe).

Modus	Vordergrundfarbe	Hintergrundfarbe
0	Farbindex der aktuellen Palette zwischen 0 und 15	Farbindex der aktuellen Palette zwischen 0 und 15
1	Bildschirmhintergrundfarbe (zwischen 0 und 15)	Palette für Vordergrundfarben: (CGA-emulierte Palette) • 0: Grün, Rot und Braun • 1: Zyan, Magenta und Weiß

		• 2: wie 0, nur mit hellen Farben • 3: wie 1, nur mit hellen Farben
2, 10 und 11	Farbindex der aktuellen Palette zwischen 0 und 1	Farbindex der aktuellen Palette zwischen 0 und 1
7, 8, 9 und 12	Farbindex der aktuellen Palette zwischen 0 und 15	Farbindex der aktuellen Palette zwischen 0 und 15
13 und höher	Farbindex der aktuellen Palette zwischen 0 und 255	Farbindex der aktuellen Palette zwischen 0 und 255

SCREENRES ohne Angabe der Farbtiefe initialisiert ein Grafikfenster mit 256 Farben (8bbp).
Bei höheren Farbtiefen als 8bpp werden die Farben als RGB-Farbwerte behandelt. RGB-Farbwerte haben die Form &hRRGGBB, wobei RR, GG und BB die Rot-, Grün- und Blau-Werte sind. Sie liegen zwischen &h00 und &hFF (bzw 0 und 255 in dezimaler Schreibweise). Sie können aber auch die Funktionen RGB verwenden.

Im 32bit-Farbmodus werden die Farbwerte im Format &hAARRGGBB angegeben, wobei AA den Wert des Alphakanals angibt. Dazu steht die Funktion RGBA zur Verfügung. Um den Alphakanal für die Transparenz der *drawing primitives* nutzen zu können, muss bei der Initialisierung des Grafikfensters das Flag GFX_ALPHA_PRIMITIVES aktiviert sein (siehe SCREENRES).

Beispiel:

```
SCREENRES 800, 600, 32     ' Bildschirmmodus 800x600 bei 32bpp
COLOR &hFF8000, &h000040   ' Vordergrund orange, Hintergrund dunkelblau

' Hello World! ausgeben
CLS
LOCATE 19, 44: PRINT "Hello World!"
SLEEP
```

Die Farben im Vollbildmodus können sich von denen der Fenstermodi unterscheiden!

Unterschiede zu QB:

- Im Textmodus können in FreeBASIC sowohl Vordergrund- als auch Hintergrundfarben zwischen 0 und 15 liegen.
- Es gibt in FreeBASIC kein Rahmen-Argument mehr.
- Die FreeBASIC-Version akzeptiert auch RGB(A)-Farben in High-/Truecolor-Modi.
- Im SCREEN-Modus 1 lassen sich 4 verschiedene Farbpaletten anstelle von zweien benutzen.
- COLOR lässt sich in FreeBASIC auch als Funktion einsetzen; siehe COLOR (Funktion).

Siehe auch:
COLOR (Funktion), SCREENRES, PALETTE, PRINT (Anweisung), INPUT (Anweisung), Grafik, Konsole

Letzte Bearbeitung des Eintrags am 27.12.12 um 00:06:26
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

COLOR (Funktion)

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » C » **COLOR (Funktion)**
Syntax: COLOR
Typ: Funktion
Kategorie: Konsole

Die Funktion COLOR gibt Informationen über die verwendeten Textfarben zurück.
In allen 32/24-Bit Farbmodi wird nur die Vordergrundfarbe (&hAARRGGBB; siehe RGBA) zurückgegeben.
Für die 16-Bit (und kleineren) Farbmodi wird der zurückgegebene Wert folgendermaßen berechnet:

```
(Vordergrund OR (Hintergrund SHL 16))
```

Das obere Word ist also die Hintergrundfarbe, während das untere Word die Vordergrundfarbe angibt.

Beispiel:

```
Screen 0
Dim c As UInteger
Color 14, 1
Cls  ' erst nach CLS wird der Hintergrund gefärbt

c = Color()  ' Abfragen der benutzten Farben

' mit LOWORD und HOWORD werden die Werte getrennt
Print "Konsolenfarben:"
Print "Schreibfarbe: " & LoWord(c)
Print "Hintergrund : " & HiWord(c)

Sleep

Screen 18, 32
' Setze orangene Schreibfarbe und blauer Hintergrund
Color RGB(255, 128, 0), RGB(0, 0, 164)
Cls  ' erst nach CLS wird der Hintergrund gefärbt

' COLOR als Funktion gibt bei 32bit nur die Schreibfarbe an
Print "&h" & Hex( Color(), 6)
Print "&h" & Hex( RGB(255, 128, 0), 6)

Sleep
```

Unterschiede zu QB:
In QB kann COLOR nicht als Funktion eingesetzt werden.

Siehe auch:
COLOR (Anweisung), WIDTH (Funktion), LOCATE (Funktion), SCREEN (Funktion), Grafik, Konsole

Letzte Bearbeitung des Eintrags am 27.12.12 um 00:06:39
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

COM

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » C » **COM**
Siehe OPEN COM

Letzte Bearbeitung des Eintrags am 28.08.11 um 17:51:42
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

COMMAND

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » C » **COMMAND**
Syntax: COMMAND[$] [(index)]
Typ: Funktion
Kategorie: System

COMMAND enthält die Kommandozeilenparameter an das Programm.

Der optionale Parameter 'index' gibt an, dass nur der n-te Kommandozeilenparameter zurückgegeben werden soll. Wird 'index' ausgelassen oder ist kleiner als 0, so gibt FreeBASIC alle Kommandozeilenparameter zurück. Ist 'index' = 0, so gibt FreeBASIC Pfad und Dateiname des aufgerufenen Programms zurück. Ist der angegebene Index größer als die Anzahl der Parameter, so wird ein leerer String ("") zurückgegeben.

Beim Lesen der Parameter wird alles innerhalb von Anführungszeichen als ein Parameter betrachtet. So lassen sich mehrere Begriffe zusammengehörig als Parameter an ein Programm übergeben.

Das Dollarzeichen ($) als Suffix ist optional.

COMMAND sollte nicht in einem globalen Konstruktor aufgerufen werden. Aufgrund der Art, wie FreeBASIC arbeitet, erhält man in diesem Fall immer eine leere Liste. Dies könnte sich in Zukunft ändern.

Beispiel:

```
'Prog1.bas
SHELL "Prog2.exe /parameter1 /parameter2"

'Prog2.bas, liegt auch als compilierte Datei Prog2.exe vor:
PRINT COMMAND
PRINT COMMAND(2)
PRINT COMMAND(0)
```

Ausgabe:

```
/parameter1 /parameter2
/parameter2
C:\BASIC\freeBASIC\Prog2.exe
```

Wildcards (siehe DIR) werden standardmäßig aufgefüllt. So entsteht eine vollständige Liste aller passenden Treffer, die als Parameter übergeben werden. Wenn es keine passenden Treffer gibt oder der Parameter in Anführungszeichen gesetzt wird, werden die Wildcards nicht aufgefüllt. Es gibt weitere, plattformspezifische Möglichkeiten, das Auffüllen zu unterdrücken:

Unter Windows bis fbc 0.24 fügt man folgendes an den Anfang des Codes:

```
Extern _CRT_glob Alias "_CRT_glob" As Integer
Dim Shared _CRT_glob As Integer = 0
```

Unter Windows ab fbc 0.90:

```
Extern _dowildcard Alias "_dowildcard" As Integer
Dim Shared _dowildcard As Integer = 0
```

Unter DOS kann man folgenden Code verwenden:

```
Public Function __crt0_glob_function Alias "__crt0_glob_function" ( ByVal
arg As UByte Ptr ) As UByte Ptr Ptr
  Return 0
End Function
```

Unter Linux ist die Shell selbst dafür zuständig. Hier kann man das Auffüllen etwa mit dem Befehl set -f ausschalten.

Unterschiede zu QB:

- COMMAND hat in QB keinen Index-Parameter.
- In QB werden alle Buchstaben der Parameterliste großgeschrieben zurückgegeben.
- In QB werden Wildcards nicht aufgelöst.

Unterschiede unter den FB-Dialektformen:

- In der Dialektform -lang qb ist das Suffix $ verbindlich.
- In den Dialektformen -lang fblite und -lang fb ist das Suffix optional.

Siehe auch:
SHELL, EXEC, EXEPATH, RUN, __FB_ARGC__, __FB_ARGV__, Betriebssystem-Anweisungen

Letzte Bearbeitung des Eintrags am 12.08.13 um 11:47:27
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

COMMON

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » C » **COMMON**
Syntax A: COMMON [SHARED] Variable1[()] AS Typ [= Ausdruck] [, Variable2[()] AS Typ [= Ausdruck] [, ...]]
Syntax B: COMMON [SHARED] AS Typ Variable1[()] [= Ausdruck] [, Variable2[()] [= Ausdruck] [, ...]]
Typ: Anweisung
Kategorie: Bibliotheken

COMMON dimensioniert Variablen und Arrays (vgl. DIM) und macht sie mehreren Modulen zugänglich.

- 'Variable1' und 'Variable2' sind die Namen von neuen, noch nicht verwendeten Variablen oder Arrays.
- 'SHARED' bewirkt, dass die Variable auch Unterprogrammen zugänglich ist.
- 'AS Typ' gibt an, von welchem Typ die Variable sein soll. Auch FUNCTION oder SUB kann hier als Typ angegeben werden; siehe dazu DYLIBLOAD.
- 'Ausdruck' gibt den Wert an, welcher der dimensionierten Variable zugewiesen wird.

Seit FreeBASIC v.0.17 muss jeder Variablen mittels 'AS Typ' explizit ein Datentyp zugewiesen werden. Standarddatentypen (vgl. DEFxxx) stehen nur noch in Dialektformen wie -lang deprecated oder -lang qb zur Verfügung.

COMMON-Arrays sind immer dynamisch. Wenn ein Array allen Modulen zugänglich gemacht werden soll, muss eine leere Parameterliste übergeben werden. Die Dimensionierung findet später über DIM oder REDIM statt.

In allen Modulen muss derselbe Bezeichner für die Variable verwendet werden; die Reihenfolge der COMMON-Anweisungen ist damit gleichgültig.

Eine ähnliche Funktion bietet EXTERN, doch während COMMON die Variable in jedem Modul deklariert und Speicher reserviert, verweist EXTERN nur auf den Speicherbereich eines anderen Moduls und definiert den Variablennamen im eigenen Modul.

Beispiel:
Compilieren Sie dieses Beispiel mit der Kommandozeile:

```
<$fbc> Modul1.bas Modul2.bas -x modultest.exe
```

<*$fbc*> steht dabei für Pfad und Dateiname Ihres FB-Compilers.)

```
' Modul1.bas

COMMON SHARED a AS INTEGER
COMMON SHARED b() AS INTEGER


REDIM b(0 TO 2) AS INTEGER
DECLARE SUB PublicSub  ()

a = 1
b(0) = 2

PublicSub
```

```
SLEEP

'========================
' Modul2.bas
COMMON SHARED b() AS INTEGER
COMMON SHARED a AS INTEGER

DECLARE SUB PublicSub  ()

PUBLIC SUB PublicSub ()
  PRINT "PublicSub in Modul2.bas"
  PRINT "a = "; a, "b(0)= "; b(0)
END SUB
```

Ausgabe:

```
PublicSub in Modul2.bas
a = 1       b(0) = 2
```

Das komplette Einbinden eines Codes mittels INCLUDE ist oft vorteilhafter.

Unterschiede zu QB:

- In FreeBASIC sind über COMMON deklarierte Arrays immer dynamisch.
- Der Blockname wird in FreeBASIC nicht mehr benötigt und ist auch nicht mehr erlaubt, da die Reihenfolge der COMMON-Variablen keine Rolle mehr spielt, sondern nur die Bezeichner.

Siehe auch:
DIM, SHARED, EXTERN (Module), Module (Library / DLL), Gültigkeitsbereich von Variablen

Letzte Bearbeitung des Eintrags am 19.01.13 um 15:51:56
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

CONDBROADCAST

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » C » **CONDBROADCAST**
Syntax: CONDBROADCAST (handle)
Typ: Anweisung
Kategorie: Multithreading

COND steht für "conditional variable". Ebenso wie MUTEXe (siehe MUTEXCREATE) stellen diese eine Möglichkeit dar, Threads (siehe THREADCREATE) zu synchronisieren.

'handle' ist der Handle zu einem COND, also der Wert, der von CONDCREATE zur Identifizierung des COND zurückgegeben wird. Es ist ein ANY PTR.

CONDBROADCAST sendet ein Signal an alle Threads, die auf 'handle' warten, dass sie fortgesetzt werden dürfen. Dieses Signal wird quasi gleichzeitig an alle wartenden Threads gesandt.

Beispiel: siehe CONDCREATE

Unterschiede zu QB: neu in FreeBASIC

Plattformbedingte Unterschiede:
In der DOS-Version von FreeBASIC steht CONDBROADCAST nicht zur Verfügung, da Threads nicht unterstützt werden.

Unterschiede zu früheren Versionen von FreeBASIC:

- CONDBROADCAST existiert seit FreeBASIC v0.13.
- Seit FreeBASIC v0.17 verlangt CONDBROADCAST einen ANY PTR als Parameter 'handle'. Davor war es ein INTEGER.

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht CONDBROADCAST nicht zur Verfügung.

Siehe auch:
Multithreading

Letzte Bearbeitung des Eintrags am 01.06.12 um 20:26:49
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

CONDCREATE

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » C » **CONDCREATE**
Syntax: handle = CONDCREATE()
Typ: Funktion
Kategorie: Multithreading

COND steht für "conditional variable". Ebenso wie MUTEXe (siehe MUTEXCREATE) stellen diese eine Möglichkeit dar, Threads (siehe THREADCREATE) zu synchronisieren.

CONDCREATE gibt einen Handle zum COND zurück. Sobald CONDs erstellt und die Threads gestartet wurden, kann ein Thread angewiesen werden, auf ein COND zu warten (CONDWAIT), bis ein anderer Thread signalisiert, dass er fortfahren darf (CONDSIGNAL). Wenn alle wartenden Threads auf einmal fortgesetzt werden sollen, kann dies mittels CONDBROADCAST geschehen. Bevor das Programm beendet wird, müssen alle CONDs gelöscht werden, um den Speicherplatz freizugeben (CONDDESTROY).

Der Rückgabetyp ist ein ANY PTR.

Der Unterschied zwischen MUTEXen und CONDs besteht darin, dass ein MUTEX immer reagiert, sobald er entsperrt wird, während auf ein CONDSIGNAL nur reagiert wird, wenn das entsprechende CONDWAIT gerade ausgeführt wird. Ist der Thread noch mit anderen Anweisungen beschäftigt, geht das CONDSIGNAL verloren. Wird ein MUTEX entsperrt, muss der wartende Thread noch nicht einmal beim entsprechenden Zugriff auf den gerade entsperrten MUTEX angekommen sein, damit das MUTEXUNLOCK seine Wirkung zeigt. Das CONDSIGNAL zeigt seine Wirkung nur dann, wenn der wartende Thread auch wirklich gerade an der Stelle CONDWAIT ist. Es empfiehlt sich daher, vor dem CONDSIGNAL oder CONDBROADCAST ein SLEEP 1 einzubauen, um sicherzustellen, dass die Threads bereit sind, das Signal aufzunehmen.

Beispiel 1:

```
Dim Shared As Any Ptr hThread1, hThread2, mWait, hCond
Dim Shared As Integer q

Sub Thread (ByVal z As Any Ptr)
  Dim As Integer j

  MutexLock mWait
  Condwait(hCond, mWait)
  MutexUnlock mWait
  Print *Cast(ZString Ptr, z) & " gestartet"

  For j = 0 To 9
    MutexLock mWait
    Condwait(hCond, mWait)
    MutexUnlock mWait
    Print *Cast(ZString Ptr, z) & " sendet:"; j
  Next

  q += 1
End Sub

hCond = Condcreate()

hThread1 = ThreadCreate(@Thread, @"Thread 1")
hThread2 = ThreadCreate(@Thread, @"Thread 2")
```

```
mWait = MutexCreate

Print "Beliebige Taste druecken, um Threads zu starten"
Sleep
Condbroadcast(hCond)
Sleep 1
Print

Do
  Sleep 1
  CondSignal(hCond)
Loop Until q = 2

CondDestroy hCond
ThreadWait(hThread1)
ThreadWait(hThread2)
MutexDestroy mWait

Sleep
```

Wie man sehen kann, werden die Threads zeitgleich durch das CONDBROADCAST gestartet; das einzelne CONDSIGNAL wird immer abwechselnd an Thread 1 bzw. Thread 2 gesandt.

Beispiel 2:

```
Dim Shared hcondstart As Any Ptr
Dim Shared hmutexstart As Any Ptr
Dim Shared start As Integer = 0

Dim Shared threadcount As Integer
Dim Shared hmutexready As Any Ptr
Dim Shared hcondready As Any Ptr

Sub mythread(ByVal id_ptr As Any Ptr)
  Dim id As Integer = Cast(Integer, id_ptr)

  Print "Thread " & id & " is waiting..."

  ' Sende Signal, dass dieser Thread bereit ist
  MutexLock hmutexready
  threadcount += 1
  CondSignal hcondready
  MutexUnlock hmutexready

  ' Warte auf das Start-Signal
  MutexLock hmutexstart
  Do While start = 0
    CondWait hcondstart, hmutexstart
  Loop

  ' nun hält dieser Thread den Lock von hmutexstart

  MutexUnlock hmutexstart
```

```
  ' Die Nummer dieses Threads ausgeben
  For i As Integer = 1 To 40
    Print id;
    Sleep 1  ' Pause, um andere Threads arbeiten zu lassen
  Next i
End Sub

Dim threads(1 To 9) As Any Ptr

' Erstelle die CONDs
hcondstart = CondCreate()
hmutexstart = MutexCreate()

hcondready = CondCreate()
hmutexready = MutexCreate()

threadcount = 0

For i As Integer = 1 To 9
  threads(i) = ThreadCreate(@mythread, Cast(Any Ptr, i))
  If threads(i) = 0 Then
  Print "CONDs konnten nicht erstellt werden!"
  End If
Next i

Print "Warte, bis alle Threads bereit sind ..."

MutexLock(hmutexready)
Do Until threadcount = 9
  ' Warten auf das Signal des Threads, dass er bereit ist
  CondWait(hcondready, hmutexready)
Loop
MutexUnlock(hmutexready)

Print "Los!"

' Alle Threads gleichzeitig starten
MutexLock hmutexstart
start = 1
CondBroadcast hcondstart
MutexUnlock hmutexstart

' Warten, bis alle Threads abgeschlossen wurden
For i As Integer = 1 To 9
  If threads(i) <> 0 Then
    ThreadWait threads(i)
  End If
Next i

' Speicher freigeben
MutexDestroy hmutexready
CondDestroy hcondready

MutexDestroy hmutexstart
```

```
CondDestroy hcondstart
Sleep
```

Unterschiede zu QB: neu in FreeBASIC

Plattformbedingte Unterschiede:
In der DOS-Version von FreeBASIC steht CONDCREATE nicht zur Verfügung, da Threads nicht unterstützt werden.

Unterschiede zu früheren Versionen von FreeBASIC:

- CONDCREATE existiert seit FreeBASIC v0.13.
- Seit FreeBASIC v0.17 gibt CONDCREATE einen ANY PTR zurück. Davor war es ein INTEGER.

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht CONDCREATE nicht zur Verfügung.

Siehe auch:
Multithreading

Letzte Bearbeitung des Eintrags am 01.06.12 um 20:26:11
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

CONDDESTROY

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » C » **CONDDESTROY**
Syntax: CONDDESTROY handle
Typ: Anweisung
Kategorie: Multithreading

COND steht für "conditional variable". Ebenso wie MUTEXe (siehe MUTEXCREATE) stellen diese eine Möglichkeit dar, Threads (siehe THREADCREATE) zu synchronisieren.

CONDDESTROY löscht ein COND. 'handle' ist der Handle zum zu löschenden COND, also der Wert, der von CONDCREATE zur Identifizierung des COND zurückgegeben wurde. Es ist ein ANY PTR.

Beispiel: siehe CONDCREATE

Unterschiede zu QB: neu in FreeBASIC

Plattformbedingte Unterschiede:
In der DOS-Version von FreeBASIC steht CONDDESTROY nicht zur Verfügung, da Threads nicht unterstützt werden.

Unterschiede zu früheren Versionen von FreeBASIC:

- CONDDESTROY existiert seit FreeBASIC v0.13.
- Seit FreeBASIC v0.17 verlangt CONDDESTROY einen ANY PTR als Parameter. Davor war es ein INTEGER.

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht CONDCREATE nicht zur Verfügung.

Siehe auch:
Multithreading

Letzte Bearbeitung des Eintrags am 01.06.12 um 20:25:35
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

CONDSIGNAL

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » C » **CONDSIGNAL**
Syntax: CONDSIGNAL (handle)
Typ: Anweisung
Kategorie: Multithreading

COND steht für "conditional variable". Ebenso wie MUTEXe (siehe MUTEXCREATE) stellen diese eine Möglichkeit dar, Threads (siehe THREADCREATE) zu synchronisieren.

'handle' ist der Handle zu einem COND, also der Wert, der von CONDCREATE zurückgegeben wird und den Thread identifiziert. Es ist ein ANY PTR.

CONDSIGNAL sendet ein Signal an einen Thread, der auf 'handle' wartet, dass er fortgesetzt werden darf. Das Signal wird dabei in der Reihenfolge ausgegeben, wie die Threads gestartet wurden. Werden also die Threads in folgender Reihenfolge gestartet:

```
A, C, B
```

dann geht das erste CONDSIGNAL auch an A, das zweite an C und das dritte an B.
Achtung: Abhängig vom Betriebssystem werden Threads nicht unbedingt in der Reihenfolge gestartet, in der sie erstellt wurden.

Beispiel: siehe CONDCREATE

Unterschiede zu QB: neu in FreeBASIC

Plattformbedingte Unterschiede:
In der DOS-Version von FreeBASIC steht CONDSIGNAL nicht zur Verfügung, da Threads nicht unterstützt werden.

Unterschiede zu früheren Versionen von FreeBASIC:

- CONDSIGNAL existiert seit FreeBASIC v0.13.
- Seit FreeBASIC v0.17 verlangt CONDSIGNAL einen ANY PTR als Parameter. Davor war es ein INTEGER.

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht CONDSIGNAL nicht zur Verfügung.

Siehe auch:
Multithreading

Letzte Bearbeitung des Eintrags am 27.12.12 um 00:08:25
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

CONDWAIT

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » C » **CONDWAIT**
Syntax: CONDWAIT (handle, mutex)
Typ: Anweisung
Kategorie: Multithreading

COND steht für "conditional variable". Ebenso wie MUTEXe (siehe MUTEXCREATE) stellen diese eine Möglichkeit dar, Threads (siehe THREADCREATE) zu synchronisieren.

- 'handle' ist der Handle zu einem COND, also der Wert, der von CONDCREATE zur Identifizierung des COND zurückgegeben wird. Es ist ein ANY PTR.
- 'mutex' ist der Mutex, der mit diesem COND verknüpft ist und der während der Überprüfung des COND und während des Aufrufs von CONDWAIT gesperrt werden muss.

CONDWAIT wartet mit der Ausführung eines Threads, bis ein CONDSIGNAL oder ein CONDBROADCAST ein Signal für diesen Handle aussendet, sodass der Thread fortgesetzt wird.

Das mit CONDWAIT verwendete Mutex sollte vor dem Befehl mit MUTEXLOCK gesperrt, und gleich danach mit MUTEXUNLOCK entsperrt werden.

Beispiel: siehe CONDCREATE

Unterschiede zu QB: neu in FreeBASIC

Plattformbedingte Unterschiede:
In der DOS-Version von FreeBASIC steht CONDWAIT nicht zur Verfügung, da Threads nicht unterstützt werden.

Unterschiede zu früheren Versionen von FreeBASIC:

- CONDWAIT existiert seit FreeBASIC v0.13.
- Seit FreeBASIC v0.17 verlangt CONDWAIT einen ANY PTR als Parameter 'handle'.
- Seit FreeBASIC v0.18.3 verlangt CONDWAIT den zweiten Parameter 'mutex'.

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht CONDWAIT nicht zur Verfügung. Davor war es ein INTEGER.

Siehe auch:
Multithreading

Letzte Bearbeitung des Eintrags am 27.12.12 um 00:09:24
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

CONS

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » C » **CONS**
Siehe OPEN CONS

Letzte Bearbeitung des Eintrags am 28.08.11 um 17:34:51
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

CONST

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » C » **CONST**
Syntax A: CONST Symbol [AS Typ] = Ausdruck [, Symbol [AS Typ] = Ausdruck [, ...]]
Syntax B: CONST AS Typ Symbol = Ausdruck [, Symbol = Ausdruck [, ...]]
Typ: Anweisung
Kategorie: Deklaration

Dieser Eintrag behandelt die Anweisung CONST. Für die Klausel CONST bei der Parameterübergabe und der Deklaration von Memberfunktionen siehe CONST (Klausel); für die Klausel CONST in SELECT-Anweisungen siehe SELECT CASE.

CONST erzeugt ein Symbol, das als Konstante behandelt wird. Dieses Symbol kann von einem beliebigen Zahlendatentyp oder ein STRING sein. ZSTRING, WSTRING und UDTs sind nicht möglich. Konstanten sind in jeder Prozedur und in jedem Modul verfügbar und können nicht verändert werden.

DEFxxx-Anweisungen haben keine Auswirkungen auf CONST-Symbole.

Über die Klausel 'AS Typ' kann festgelegt werden, welchen Datentyp ein Symbol besitzen soll. Ansonsten verwendet FreeBASIC automatisch einen passenden Datentyp (in der Regel INTEGER für Ganzzahlen und DOUBLE für Gleitkommazahlen).

Bei 'AS STRING' wird geprüft, ob es sich bei dem folgenden Ausdruck tatsächlich um einen String handelt; WSTRINGs und ZSTRINGs sind als Konstanten nicht zugelassen.

Beispiel:

```
CONST blau = 1, rot = CAST(SHORT, 4)
CONST DoubleValue1 = 1"hlzeichen">, DoubleValue2 = 3.0
CONST SingleValue AS SINGLE = 3.0
CONST msg = "Hallo Welt!"

PRINT LEN(blau), LEN(rot)
PRINT LEN(DoubleValue1), LEN(DoubleValue2)
PRINT LEN(SingleValue)
PRINT msg
SLEEP
```

Ausgabe:

```
 4      2
 8      8
 4
Hallo Welt!
```

Da der Konstanten 'blau' eine Ganzzahl zugewiesen wurde, wird sie als INTEGER behandelt. Dieses Verhalten wurde bei 'rot' geändert, indem die zugewiesene Zahl zu einer SHORT-Variablen gecastet wurde. 'DoubleValue1' wird durch das Suffix als DOUBLE definiert, während bei 'DoubleValue2' die Schreibweise mit Dezimalpunkt ausschlaggebend ist. 'SingleValue' schließlich wurde explizit als SINGLE deklariert.

Unterschiede zu QB:
Unter QB kann der Datentyp von Konstanten nicht festgelegt werden.

Unterschiede zu früheren Versionen von FreeBASIC:

- Seit FreeBASIC v0.16 ist es möglich, den Datentyp einer Konstante explizit festzulegen. Davor wurde automatisch der am besten geeignete Datentyp gewählt.
- Die alternative Syntax CONST AS Typ Symbol (Syntax B) existiert seit FreeBASIC v0.16
- Seit FreeBASIC v0.13 können Strings als Konstanten definiert werden.

Siehe auch:
CONST (Klausel), DIM, VAR, DEFINE (Meta), SUB, FUNCTION, SELECT CASE, Datentypen

Letzte Bearbeitung des Eintrags am 27.12.12 um 00:11:07
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

CONST (Klausel)

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » C » **CONST (Klausel)**
Syntax A:

```
[DECLARE] {SUB | FUNCTION} _
        ( [BYVAL | BYREF] [Parameter] AS [CONST] Typ [= Wert]
          [, ...] )
```

Syntax B:

```
TYPE typename
  DECLARE CONST {SUB | FUNCTION} Prozedurname ...
END TYPE
```

Typ: Klausel
Kategorie: Programmablauf

Dieser Eintrag behandelt die Klausel CONST bei der Parameterübergabe bzw. der Deklaration von Memberfunktionen. Für die Definition von Konstanten siehe CONST; für die Klausel CONST in SELECT-Anweisungen siehe SELECT CASE.

CONST bewirkt, dass auf die damit bezeichnete Variable (Syntax A) bzw. auf die Prozedur innerhalb eines UDTs (Syntax B) nur lesend, aber nicht schreibend zugegriffen werden kann.

Als Klausel bei der Parameterübergabe in Syntax A garantiert CONST, dass die damit bezeichnete Variable innerhalb der Prozedur nicht verändert werden kann. Wird beim Compiliervorgang eine Operation entdeckt, welche den Wert des Parameters möglicherweise ändert, so wird der Compiliervorgang mit einem Fehler abgebrochen. Da die Überprüfung bereits beim Compilieren stattfindet, kommt es während der Programmausführung zu keinen Geschwindigkeitsnachteilen.

Beispiel:

```
Sub foo (x As Integer, y As Const Integer)
  Print x, y

  x = 5
  ' y = 7 ' wird versucht y zu ändern, gibt der Compiler einen Fehler aus

  Print x, y
End Sub

foo(1, 2)
Sleep
```

Als Klausel bei Memberfunktionen (Syntax B) bezieht sich der Schreibschutz auf die Instanz des UDTs, auf die mit THIS zugegriffen werden kann. Damit ist es der Prozedur nicht erlaubt, schreibend auf die Records des UDTs zuzugreifen oder eine nicht-konstante Memberfunktion aufzurufen. Wird beim Compiliervorgang eine solche nicht erlaubte Operation entdeckt, so wird der Compiliervorgang mit einem Fehler abgebrochen. Memberfunktionen können nicht gleichzeitig mit CONST und STATIC deklariert werden, da statische Memberfunktionen keinen THIS-Parameter besitzen.

Beispiel:

```
Type foo
  x As Integer
  c As Const Integer = 0
  Declare Const Sub Lesen1()
  Declare Const Sub Lesen2()
  Declare Sub Schreiben1()
  Declare Sub Schreiben2()
End Type

Sub foo.Schreiben1
  x = 1          ' ein nicht konstantes Record ändern
  ' c = 1        ' Compiler-Fehler: c ist konstant
End Sub

Sub foo.Schreiben2
  Lesen1         ' sowohl konstante Memberfunktionen als auch
  Schreiben1     ' nicht konstante können aufgerufen werden
End Sub

Sub foo.Lesen1
  ' auf Records kann lesend zugegriffen werden ...
  Dim y As Integer
  y = c + x

  ' ... sie können aber nicht verändert werden
  ' x = 10       ' Compiler-Fehler: Lesen1 ist konstant
End Sub

Sub foo.Lesen2
  ' konstante Memberfunktionen können aufgerufen werden ...
  Lesen1
  ' ... nicht konstante jedoch nicht
  ' Schreiben1  ' Compiler-Fehler
End Sub
```

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC: existiert seit FreeBASIC v0.18.3

Siehe auch:
CONST, DECLARE, SUB, FUNCTION, TYPE (UDT), SELECT CASE, DIM, Prozeduren, Objektorientierung

Letzte Bearbeitung des Eintrags am 27.12.12 um 00:11:29
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

CONSTRUCTOR (Klassen)

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » C » **CONSTRUCTOR (Klassen)**
Syntax A:

```
TYPE TypeName
    ' Feld-Deklarationen
    DECLARE CONSTRUCTOR ( )
    DECLARE CONSTRUCTOR ( &"hlzeichen">] )
End Type
```

Syntax B:

```
CONSTRUCTOR Typename ( &"hlzeichen">] )
   ' Anweisungen
END CONSTRUCTOR
```

Typ: Prozedur
Kategorie: Klassen

Ein Klassen-Konstruktor ist eine Prozedur, die aufgerufen wird, sobald eine UDT-Variable (user defined Type - siehe TYPE (UDT)) erstellt wird. Für einen alternativen Verwendungszusammenhang siehe CONSTRUCTOR (Module).

Dieses Schlüsselwort ist nur zulässig, wenn mit der Kommandozeilenoption -lang fb compiliert wird; siehe FB-Dialektformen.

- 'TypeName' ist der Name eines UDTs, wie er unter TYPE (UDT) erklärt wird.
- 'Feld-Deklarationen' sind die Deklarationen, die den UDT oder die Klasse ausmachen. Siehe dazu TYPE (UDT), PROPERTY, OPERATOR.
- 'Parameterliste' ist vom selben Format wie bei SUB. Auch optionale Argumente werden unterstützt.
- 'Anweisungen' ist ein Programmcode, der den Regeln einer SUB folgt.

Konstruktoren bei Klassen werden immer dann ausgeführt, wenn eine Variable eines Typs erstellt wird, der einen Konstruktor besitzt. In einem Konstruktor können die Records des UDTs bereits mit Standardwerten ausgefüllt werden sowie allgemeine Aufgaben erledigt werden, die bei der Erstellung der Variable fällig werden.

Befindet sich der TYPE innerhalb eines NAMESPACEs, so wird die Zuordnung genau so durchgeführt wie bei normalen Prozeduren innerhalb von NAMESPACEs; es ist möglich, den Bezeichner des TYPEs als Präfix vor den Bezeichner der Property zu hängen oder mittels USING das Präfix für den gesamten Code überflüssig machen.

Es ist möglich, einem Typen mehrere Konstruktoren zuzuordnen; diese müssen sich dann durch ihre Parameterliste unterscheiden. Siehe dazu auch OVERLOAD. Ein Konstruktor darf gar keinen, einen oder mehrere Parameter benutzen.

Welcher Konstruktor aufgerufen wird, hängt von den Parametern ab, die bei der Erstellung des Typs angegeben werden. Die Parameter werden dabei ähnlich übergeben, wie dies bei Variablen-Initiatoren geschieht:

Bei einem Parameter:

```
DIM VariablenName AS DatenTyp = Wert
```

Bei mehreren Parametern:

```
DIM VariablenName AS DatenTyp = DatenTyp( Wert1, Wert2, ... )
```

Statt DIM können natürlich auch COMMON und STATIC verwendet werden.

CONSTRUCTOR überschreibt den Standard-Konstruktor einer Klasse. Somit sind manche Deklarationsformen bei Variablen nur noch möglich, wenn der entsprechende Konstruktor auch definiert wurde. Einige Aufrufe wie REDIM oder ERASE benötigen einen Standard-Konstruktor, d. h. es darf entweder gar kein Konstruktor angegeben werden oder es muss ein parameterloser Konstruktor bereitgestellt werden (weitere Konstruktoren mit Parametern sind natürlich möglich).

Innerhalb der CONSTRUCTOR-Prozedur kann über das Schlüsselwort THIS auf die Records des UDTs zugegriffen werden.

Enthält der initiierte UDT selbst Felder, die Klassen mit eigenen Konstruktoren darstellen, so werden diese zuerst aufgerufen.

Klassen-Konstruktoren müssen PUBLIC sein; siehe PUBLIC (UDT).

Das Constructor-Schlüsselwort leitet einen eigenen SCOPE-Block ein. Siehe dazu auch SCOPE.

Beispiel:

```
Type sample

  Text As String

  Declare Constructor ()
  Declare Constructor ( a As Integer )
  Declare Constructor ( a As Single  )
  Declare Constructor ( a As String, b As Byte )

  Declare Operator Cast () As String

End Type

Constructor sample ()
  Print "Konstruktor sample ()"
  Print
  THIS.Text = "Leer"
End Constructor

Constructor sample ( a As Integer )
  Print "Konstruktor sample ( a As Integer )"
  Print "  a = "; a
  Print
  THIS.Text = Str(a)
End Constructor

Constructor sample ( a As Single )
  Print "Konstruktor sample ( a As Single )"
  Print "  a = "; a
  Print
```

```
  THIS.Text = Str(a)
End Constructor

Constructor sample ( a As String, b As Byte )
  Print "Konstruktor sample ( a As String, b As Byte )"
  Print "  a = "; a
  Print "  b = "; b
  Print
  THIS.Text = a & "," & b
End Constructor

Operator sample.CAST () As String
  Return THIS.Text
End Operator

Print "Erstelle x1"
Dim x1 As sample

Print "Erstelle x2"
Dim x2 As sample = 1

Print "Erstelle x3"
Dim x3 As sample = 99.9

Print "Erstelle x4"
Dim x4 As sample = sample( "aaa", 1 )

Print "Werte:"
Print "  x1 = "; x1
Print "  x2 = "; x2
Print "  x3 = "; x3
Print "  x4 = "; x4
Sleep
```

Ausgabe:

```
Erstelle x1
Konstruktor sample ()

Erstelle x2
Konstruktor sample ( a As Integer )
  a =  1

Erstelle x3
Konstruktor sample ( a As Single )
  a =  99.9

Erstelle x4
Konstruktor sample ( a As String, b As Byte )
  a = aaa
  b =  1

Werte:
  x1 = Leer
```

```
x2 = 1
x3 = 99.9
x4 = aaa,1
```

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC:

- CONSTRUCTOR für Klassen existiert seit FreeBASIC v0.17.

Unterschiede unter den FB-Dialektformen: nur in der Dialektform -lang fb verfügbar

Siehe auch:
DESTRUCTOR (Klassen), CONSTRUCTOR (Module), TYPE (UDT), NAMESPACE, Objektorientierung

Letzte Bearbeitung des Eintrags am 07.12.13 um 18:08:55
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

CONSTRUCTOR (Module)

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » C » **CONSTRUCTOR (Module)**
Syntax: [PUBLIC | PRIVATE] SUB name [ALIAS "externerName"] [()] CONSTRUCTOR [priorität] [STATIC]
Typ: Klausel
Kategorie: Programmablauf

Die Klausel CONSTRUCTOR bewirkt, dass eine SUB aufgerufen wird, bevor das Programm gestartet wird. Siehe auch SUB für Details zu Prozeduren.

- 'PRIVATE' und 'PUBLIC' legen fest, ob die SUB PRIVATE oder PUBLIC sein soll; siehe dazu die entsprechenden Einträge der Referenz.
- 'name' ist der Bezeichner der SUB. Er folgt den üblichen Regeln.
- Die 'CONSTRUCTOR'-Klausel markiert die SUB als Konstruktor.
- Ein Konstruktor darf keine Parameterliste besitzen, da kein Weg existiert, beim Programmstart Parameter für die Übergabe festzulegen.
- 'priorität' ist ein INTEGER-Wert zwischen 101 und 65535. Mit diesem Wert kann bei der Existenz mehrerer CONSTRUCTOR-SUBs im selben Modul die Reihenfolge festgelegt werden, in der die Konstruktoren abgearbeitet werden sollen. 101 ist dabei die höchste Priorität; dieser Konstruktor wird zuerst ausgeführt. Die Zahlen an sich haben keine spezielle Bedeutung; lediglich die Verhältnisse zueinander (größer als, kleiner als) wirken sich tatsächlich im Programmverlauf aus. Alle Konstruktor, die ohne eine Priorität festgelegt wurden, werden nach denen ausgeführt, die mit Priorität festgelegt wurden.

Werden in einem Modul mehrere Konstruktoren ohne Priorität verwendet, so werden diese im Code von oben nach unten abgearbeitet.

Ein Konstruktor muss nur dann durch eine DECLARE-Anweisung deklariert werden, wenn sie im Programmverlauf nochmals manuell aufgerufen werden soll. Sie kann dann - wie eine normale Prozedur - im Programm jederzeit aufgerufen werden und folgt den normalen Regeln.

Wird versucht, eine SUB mit der CONSTRUCTOR-Klausel zu belegen, die einen oder mehrere Parameter besitzt, so wird ein Compiler-Fehler erzeugt.

Innerhalb eines Modules können mehrere Konstruktoren verwendet werden. Innerhalb mehrerer Module können ebenfalls zusätzliche Konstruktoren eingefügt werden, unter der Voraussetzung, dass keine zwei PUBLIC-Konstruktoren mit demselben Bezeichner existieren.

Wenn mehrere Module compiliert werden, die Konstruktoren besitzen, kann die Reihenfolge der Ausführung der Konstruktoren nicht garantiert werden, solange keine Priorität angegeben wurde. Darauf muss besonders geachtet werden, wenn die Konstruktoren eines Moduls eine Prozedur eines anderen Moduls aufrufen, das ebenfalls einen Konstruktor besitzt. Es wird empfohlen, einen einzigen Konstruktor zu definieren, der die Prozeduren in den einzelnen Modulen manuell aufruft.

Beispiel:
Dieses Beispielprogramm enthält ein Set aus vier Konstruktoren und Destruktoren. Es wird gezeigt, in welcher Reihenfolge diese abgearbeitet werden.

```
Sub Constructor1 Constructor 101
    Print "Constructor1 aufgerufen - hoechste Prioritaet"
End Sub
Sub Constructor2 Constructor 201
    Print "Constructor2 aufgerufen - niedrigere Prioritaet"
```

```
End Sub
Sub Constructor3 Constructor
    Print "Constructor3 aufgerufen - ohne Prioritaet"
End Sub
Sub Constructor4 Constructor
    Print "Constructor4 aufgerufen - ohne Prioritaet"
End Sub

'-------------------------------------------------------

Sub Destructor1 Destructor 101
    Print "Destructor1 aufgerufen - niedrigste Prioritaet"
    Sleep 'Auf Tastendruck warten
End Sub
Sub Destructor2 Destructor 201
    Print "Destructor2 aufgerufen - hoehere Prioritaet"
End Sub
Sub Destructor3 Destructor
    Print "Destructor3 aufgerufen - ohne Prioritaet"
End Sub
Sub Destructor4 Destructor
    Print "Destructor4 aufgerufen - ohne Prioritaet"
End Sub

'-------------------------------------------------------

Print "Modul-Level Code"
```

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC:

- Die Möglichkeit, eine Priorität zuzuweisen, existiert seit FreeBASIC v0.17.
- Konstruktor-SUBs existieren seit FreeBASIC v0.14.

Siehe auch:
DESTRUCTOR (Module), SUB, CONSTRUCTOR (Klassen), Prozeduren

Letzte Bearbeitung des Eintrags am 19.01.13 um 16:13:08
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

CONTINUE

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » C » **CONTINUE**
Syntax: CONTINUE {DO | FOR | WHILE}
Typ: Anweisung
Kategorie: Programmablauf

CONTINUE springt in einer DO-, FOR- oder WHILE-Schleife an das Schleifen-Ende, wo dann die Abbruchbedingung geprüft wird. Bei einer FOR-Schleife wird der Schleifenzähler entsprechend der STEP-Anweisung erhöht (falls sie vorhanden ist; ansonsten findet regulär eine Erhöhung um 1 statt).

Beispiel 1:

```
DO
   PRINT "Diese Zeile wird immer angezeigt."
   SLEEP 100
   IF INKEY <> "" THEN EXIT DO
   CONTINUE DO
   PRINT "Diese Zeile wird NIE angezeigt."
LOOP

PRINT "Schleife verlassen"
SLEEP
```

Wenn mehrere Schleifen ineinander verschachtelt sind, dann wird die innerste Schleife der angegebenen Art fortgesetzt. Durch die mehrfache Angabe von Schleifentypen, durch Komma getrennt, kann auch eine weiter außen liegende Schleife angesprochen werden.

Beispiel 2:

```
' einfache Primzahlensuche

Print "Die Primzahlen zwischen 1 und 20 lauten:"
Print

For n As Integer = 2 To 20
  For t As Integer = 2 To Int(Sqr(n))
    If (n Mod t) = 0 Then  ' n ist durch t teilbar
      Continue For, For    ' n ist keine Primzahl; nächstes n versuchen
    End If
  Next t
  Print n
Next n
Sleep
```

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC:
Die Möglichkeit, eine Schleife höherer Ebene anzusprechen (z. B. CONTINUE DO, DO) existiert seit FreeBASIC v0.17.

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht CONTINUE nicht zur Verfügung und kann nur über **__CONTINUE** aufgerufen werden.

Siehe auch:
DO ... LOOP, FOR ... NEXT, WHILE ... WEND, EXIT, Schleifen

Letzte Bearbeitung des Eintrags am 27.12.12 um 00:13:24
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

COS

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » C » **COS**
Syntax: COS (winkel)
Typ: Funktion
Kategorie: Mathematik

COS gibt den Kosinus eines Winkels 'winkel' im Bogenmaß zurück. Der Kosinus zu einem Winkel ist die relative horizontale Entfernung innerhalb des Normkreises zu seinem Mittelpunkt (Wikipedia-Artikel zum Thema).

- 'winkel' ist ein beliebiger numerischer Ausdruck. Variablen, Konstanten, Operatoren und Funktionen sind erlaubt. Der Ausdruck darf von jedem Datentyp außer STRING, ZSTRING oder WSTRING sein.
- Der Rückgabewert ist ein DOUBLE-Wert, der den Kosinus-Wert zum Winkel enthält.

Die Umkehrfunktion zu COS lautet ACOS.

COS kann mithilfe von OPERATOR überladen werden.

Beispiel:

```
CONST PI AS DOUBLE = ACOS(0)*2
DIM a AS DOUBLE
DIM r AS DOUBLE

INPUT "Bitte geben Sie einen Winkel in Grad (DEG) ein: ", a
' Grad in Bogenmaß umrechnen:
r = a * PI / 180
PRINT ""
PRINT "Der Kosinus eines" ; a; "°-Winkels ist ";
PRINT USING ""; COS(r)
SLEEP
```

Ausgabebeispiel:

```
Bitte geben Sie einen Winkel in Grad (DEG) ein: 60

Der Kosinus eines 60°-Winkels ist 0.50
```

Unterschiede zu früheren Versionen von FreeBASIC:
Die Überladung von COS für benutzerdefinierte Datentypen ist seit FreeBASIC v0.22 möglich.

Siehe auch:
SIN, ASIN, ACOS, TAN, ATN, ATAN2, mathematische Funktionen

Letzte Bearbeitung des Eintrags am 24.12.12 um 14:12:50
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

CPTR

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » C » **CPTR**
Syntax: CPTR (Datentyp PTR, 32bit_Ausdruck)
Typ: Funktion
Kategorie: Typumwandlung

CPTR verwandelt einen beliebigen 32bit-Ausdruck in einen Pointer eines beliebigen Typs.

- 'Datentyp' ist ein beliebiger Datentyp, der jedoch ein Pointer sein muss. Erlaubt sind auch ZSTRING- und WSTRING-Pointer.
- '32bit_Ausdruck' ist entweder ein Pointer, eine INTEGER- oder eine UINTEGER-Variable.
- Der Rückgabewert ist ein Pointer auf eine Speicherstelle vom Typ 'Datentyp'.

Beispiel:

```
DIM i AS INTEGER
DIM ip AS INTEGER PTR, z AS ZSTRING PTR

i = &h0080
ip = @i
PRINT *CPTR(BYTE PTR, ip), *ip

i = 33
ip = @i
z = CPTR(ZSTRING PTR, ip)

PRINT i, *z
SLEEP
```

Ausgabe:

```
-128   128
33   !
```

'!' ist ASCII-Char 33 (siehe ASC, CHR).

Hinweis: Momentan gibt der Compiler keinen Fehler aus, wenn als 'Datentyp PTR' ein normaler Datentyp und kein Pointer angegeben wird. Dies kann zu Fehlern führen.

Unterschiede zu QB: neu in FreeBASIC

Unterschiede in den Dialektformen:
In der Dialektform -lang qb steht CPTR nicht zur Verfügung und kann nur über **__CPTR** aufgerufen werden.

Siehe auch:
CAST, Datentypen umwandeln

Letzte Bearbeitung des Eintrags am 03.06.12 um 19:41:21
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

CSHORT

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » C » **CSHORT**
Syntax: CSHORT (Ausdruck)
Typ: Funktion
Kategorie: Typumwandlung

CSHORT konvertiert einen beliebige numerischen Ausdruck zu einem SHORT (16-bit). Es erfüllt dieselbe Funktion wie CAST(SHORT,Ausdruck).

Bei Bedarf wird die Zahl mathematisch gerundet: Ist der Nachkommawert größer als .5, dann wird aufgerundet; ist er kleiner als .5, dann wird abgerundet. Ist der Nachkommawert genau .5, dann wird zur nächstliegenden *geraden* Zahl gerundet.

Handelt es sich bei dem Ausdruck um einen STRING, dann wird dieser mit der Funktion VALINT umgewandelt. Dabei wird nicht gerundet, sondern die Nachkommastellen werden abgeschnitten.

Beispiel:

```
DIM a AS INTEGER = 10
DIM b AS SHORT = CSHORT(a)
PRINT a, b
SLEEP
```

Ausgabe:

```
10            10
```

Unterschiede zu QB: neu in FreeBASIC

Unterschiede in den Dialektformen:
In der Dialektform -lang qb steht CSHORT nicht zur Verfügung und kann nur über **__CSHORT** aufgerufen werden.

Siehe auch:
CAST, CBYTE, CUBYTE, CUSHORT, CINT, CUINT, CLNG, CLNGINT, CULNGINT, CSNG, CDBL, CSIGN, CUNSG, Datentypen umwandeln

Letzte Bearbeitung des Eintrags am 01.06.12 um 01:46:52
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

CSIGN

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » C » **CSIGN**
Syntax: CSIGN (Ausdruck)
Typ: Funktion
Kategorie: Typumwandlung

CSIGN verwandelt eine vorzeichenlose Zahl in eine vorzeichenbehaftete Zahl desselben Datentyps.

Die Funktion wird dazu benutzt, bei Multiplikationen und Divisionen das Verhalten vorzeichenbehafteter Zahlen zu erzwingen (im Zusammenspiel mit SHL und SHR). CSIGN ist die Gegenfunktion zu CUNSG.

Beispiel:

```
DIM a AS USHORT
a = 65000
PRINT CSIGN(a)
SLEEP
```

Ausgabe:

```
-536
```

Unterschiede zu QB: neu in FreeBASIC

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht CSIGN nicht zur Verfügung und kann nur über **__CSIGN** aufgerufen werden.

Siehe auch:
CAST, CBYTE, CUBYTE, CUSHORT, CINT, CUINT, CLNG, CLNGINT, CULNGINT, CSNG, CDBL, CSHORT, CUNSG, Datentypen umwandeln

Letzte Bearbeitung des Eintrags am 01.06.12 um 01:32:45
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

CSNG

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » C » **CSNG**
Syntax: CSNG(Ausdruck)
Typ: Funktion
Kategorie: Typumwandlung

CSNG konvertiert einen beliebigen numerischen Ausdruck in eine SINGLE-Zahl (32-bit).

Beispiel:

```
Dim As Double a
Dim As Single b
a = 0.123456789999
b = CSng(a)

Print a; b
Sleep
```

Ausgabe:

```
0.123456789999              0.1234568
```

Unterschiede zu QB:
In QB darf 'Ausdruck' kein String sein.

Siehe auch:
CAST, CBYTE, CUBYTE, CUSHORT, CINT, CUINT, CLNG, CLNGINT, CULNGINT, CSIGN, CDBL, CSHORT, CUNSG, Datentypen umwandeln

Letzte Bearbeitung des Eintrags am 01.06.12 um 01:36:01
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

CSRLIN

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » C » **CSRLIN**
Syntax: CSRLIN
Typ: Funktion
Kategorie: Konsole

CSRLIN (CurSoR LINe) gibt die aktuelle Zeile (vertikale Position) des Cursors zurück. Die oberste Zeile besitzt den Wert 1.

Beispiel:

```
LOCATE 3, 10
PRINT CSRLIN
PRINT CSRLIN
SLEEP
```

Ausgabe:

```
           3
 4
```

Siehe auch:
POS, LOCATE (Anweisung), LOCATE (Funktion), Konsole

Letzte Bearbeitung des Eintrags am 01.02.12 um 21:02:06
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

CUBYTE

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » C » **CUBYTE**
Syntax: CUBYTE (Ausdruck)
Typ: Funktion
Kategorie: Typumwandlung

CUBYTE konvertiert einen beliebigen numerischen Ausdruck zu einemUBYTE. Es erfüllt dieselbe Funktion wie CAST(UBYTE,Ausdruck).

Bei Bedarf wird die Zahl mathematisch gerundet: Ist der Nachkommawert größer als .5, dann wird aufgerundet; ist er kleiner als .5, dann wird abgerundet. Ist der Nachkommawert genau .5, dann wird zur nächstliegenden *geraden* Zahl gerundet.

Handelt es sich bei dem Ausdruck um einen STRING, dann wird dieser mit der Funktion VALINT umgewandelt. Dabei wird nicht gerundet, sondern die Nachkommastellen werden abgeschnitten.

Beispiel:

```
PRINT CUBYTE(460)
```

Ausgabe:

```
204
```

Unterschiede zu QB: neu in FreeBASIC

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht CUBYTE nicht zur Verfügung und kann nur über **__CUBYTE** aufgerufen werden.

Siehe auch:
CAST, CBYTE, CSNG, CUSHORT, CINT, CUINT, CLNG, CLNGINT, CULNGINT, CSIGN, CDBL, CSHORT, CUNSG, Datentypen umwandeln

Letzte Bearbeitung des Eintrags am 01.06.12 um 01:47:21
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

CUINT

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » C » **CUINT**
Syntax:

```
CUINT (Ausdruck)
CUINT<Bits> (Ausdruck)
```

Typ: Funktion
Kategorie: Typumwandlung

CUINT konvertiert einen beliebigen numerischen Ausdruck zu einem UINTEGER oder UINTEGER<Bits>. Es erfüllt dieselbe Funktion wie CAST(UINTEGER, Ausdruck) bzw. CAST(UINTEGER<Bits>, Ausdruck).

Bei Bedarf wird die Zahl mathematisch gerundet: Ist der Nachkommawert größer als .5, dann wird aufgerundet; ist er kleiner als .5, dann wird abgerundet. Ist der Nachkommawert genau .5, dann wird zur nächstliegenden *geraden* Zahl gerundet.

Der Parameter 'Bits' gibt an, welche Größe der zurückgegebene INTEGER<Bits> besitzen soll. Die erlaubten Werte sind 8, 16, 32 und 64..

Handelt es sich bei dem Ausdruck um einen STRING, dann wird dieser mit der Funktion VALUINT umgewandelt. Dabei wird nicht gerundet, sondern die Nachkommastellen werden abgeschnitten.

Beispiel:

```
PRINT CUINT(123.45)      ' gibt 123 aus
PRINT CUINT<8>(123.45)   ' gibt 123 aus
PRINT CUINT(457.5)       ' gibt 458 aus
PRINT CUINT(456.5)       ' gibt 456 aus
PRINT CUINT("457.5")     ' gibt 457 aus
SLEEP
```

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC:
Der Parameter 'Bits' existiert seit FreeBASIC v0.90.

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht CUINT nicht zur Verfügung und kann nur über **__CUINT** aufgerufen werden.

Siehe auch:
CAST, CBYTE, CSNG, CUSHORT, CINT, CUBYTE, CLNG, CLNGINT, CULNGINT, CSIGN, CDBL, CSHORT, CUNSG, Datentypen umwandeln

Letzte Bearbeitung des Eintrags am 04.07.13 um 17:16:56
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

CULNG

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » C » **CULNG**
Syntax: CULNG (Ausdruck)
Typ: Funktion
Kategorie: Typumwandlung

CULNG konvertiert einen beliebigen numerischen Ausdruck zu einem ULONG. Es erfüllt dieselbe Funktion wie CAST(ULONG,Ausdruck).

Bei Bedarf wird die Zahl mathematisch gerundet: Ist der Nachkommawert größer als .5, dann wird aufgerundet; ist er kleiner als .5, dann wird abgerundet. Ist der Nachkommawert genau .5, dann wird zur nächstliegenden *geraden* Zahl gerundet.

Handelt es sich bei dem Ausdruck um einen STRING, dann wird dieser mit der Funktion VALINT umgewandelt. Dabei wird nicht gerundet, sondern die Nachkommastellen werden abgeschnitten.

Beispiel:

```
PRINT CULNG(123.45)   ' gibt 123 aus
PRINT CULNG(457.5)    ' gibt 458 aus
PRINT CULNG(456.5)    ' gibt 456 aus
PRINT CULNG("457.5") ' gibt 457 aus
SLEEP
```

Unterschiede zu QB: neu in FreeBASIC

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht CULNG nicht zur Verfügung und kann nur über **__CULNG** aufgerufen werden.

Siehe auch:
CAST, CBYTE, CUBYTE, CSHORT, CUSHORT, CINT, CUINT, CLNGINT, CULNGINT, CSNG, CDBL, CSIGN, CUNSG, Datentypen umwandeln

Letzte Bearbeitung des Eintrags am 15.06.13 um 00:30:28
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

CULNGINT

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » C » **CULNGINT**
Syntax: CULNGINT (Ausdruck)
Typ: Funktion
Kategorie: Typumwandlung

CULNGINT konvertiert einen beliebigen numerischen Ausdruck zu einem ULONGINT. Es erfüllt dieselbe Funktion wie CAST(ULONGINT,Ausdruck).

Bei Bedarf wird die Zahl mathematisch gerundet: Ist der Nachkommawert größer als .5, dann wird aufgerundet; ist er kleiner als .5, dann wird abgerundet. Ist der Nachkommawert genau .5, dann wird zur nächstliegenden *geraden* Zahl gerundet.

Handelt es sich bei dem Ausdruck um einen STRING, dann wird dieser mit der Funktion VALINT umgewandelt. Dabei wird nicht gerundet, sondern die Nachkommastellen werden abgeschnitten.

Beispiel:

```
PRINT CULNGINT(12345678.45)    ' gibt 12345678 aus
PRINT CULNGINT(123457.5)       ' gibt 123458 aus
PRINT CULNGINT(123456.5)       ' gibt 123456 aus
PRINT CULNGINT("123457.5")     ' gibt 123457 aus
SLEEP
```

Unterschiede zu QB: neu in FreeBASIC

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht CULNGINT nicht zur Verfügung und kann nur über **__CULNGINT** aufgerufen werden.

Siehe auch:
CAST, CBYTE, CSNG, CUSHORT, CINT, CUBYTE, CLNG, CLNGINT, CUINT, CSIGN, CDBL, CSHORT, CUNSG, Datentypen umwandeln

Letzte Bearbeitung des Eintrags am 15.06.13 um 00:31:59
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

CUNSG

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » C » **CUNSG**
Syntax: CUNSG (Ausdruck)
Typ: Funktion
Kategorie: Typumwandlung

CUNSG konvertiert eine vorzeichenbehaftete Zahl in eine vorzeichenlose Zahl desselben Datentyps.

Die Funktion wird benutzt, um bei Multiplikationen und Divisionen das Verhalten vorzeichenloser Zahlen zu erzwingen (im Zusammenspiel mit SHL und SHR). CUNSG ist die Gegenfunktion zu CSIGN.

Beispiel:

```
DIM a AS SHORT
a = -200
PRINT CUNSG(a)
SLEEP
```

Ausgabe:

```
65336
```

CUNSG ist die Gegenfunktion zu CSIGN.

Unterschiede zu QB: neu in FreeBASIC

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht CUNSG nicht zur Verfügung und kann nur über **__CUNSG** aufgerufen werden.

Siehe auch:
CAST, CBYTE, CSNG, CUSHORT, CINT, CUBYTE, CLNG, CLNGINT, CUINT, CSIGN, CDBL, CSHORT, CULNGINT, Datentypen umwandeln

Letzte Bearbeitung des Eintrags am 01.06.12 um 01:33:34
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

CURDIR

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » C » **CURDIR**
Syntax: CURDIR[$]
Typ: Funktion
Kategorie: System

CURDIR (CURrent DIRectory) gibt das aktuelle Arbeitsverzeichnis aus. Beim Programmstart handelt es sich dabei um das Verzeichnis, von dem aus das Programm gestartet wurde (dies ist *nicht* unbedingt identisch mit dem Verzeichnis, in dem das Programm liegt; siehe EXEPATH). Das Arbeitsverzeichnis kann mit CHDIR gewechselt werden.

Das Dollarzeichen ($) als Suffix ist optional.

Beispiel:

```
PRINT CURDIR
SLEEP
```

Unterschiede zu QB: neu in FreeBASIC

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht CURDIR nicht zur Verfügung und kann nur über **__CURDIR** aufgerufen werden.

Siehe auch:
EXEPATH, CHDIR, MKDIR, RMDIR, SHELL, OPEN, DIR, Betriebssystem-Anweisungen

Letzte Bearbeitung des Eintrags am 27.12.12 um 00:14:45
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

CUSHORT

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » C » **CUSHORT**
Syntax: CUSHORT (Ausdruck)
Typ: Funktion
Kategorie: Typumwandlung

CUSHORT konvertiert einen beliebigen numerischen Ausdruck zu einem USHORT. Es erfüllt dieselbe Funktion wie CAST(USHORT,Ausdruck).

Bei Bedarf wird die Zahl mathematisch gerundet: Ist der Nachkommawert größer als .5, dann wird aufgerundet; ist er kleiner als .5, dann wird abgerundet. Ist der Nachkommawert genau .5, dann wird zur nächstliegenden *geraden* Zahl gerundet.

Handelt es sich bei dem Ausdruck um einen STRING, dann wird dieser mit der Funktion VALINT umgewandelt. Dabei wird nicht gerundet, sondern die Nachkommastellen werden abgeschnitten.

Beispiel:

```
DIM a AS INTEGER = -10
DIM b AS USHORT = CUSHORT(a)
PRINT a, b
SLEEP
```

Ausgabe:

```
-10           65526
```

Unterschiede zu QB: neu in FreeBASIC

Unterschiede in den Dialektformen:
In der Dialektform -lang qb steht CUSHORT nicht zur Verfügung und kann nur über **__CUSHORT** aufgerufen werden.

Siehe auch:
CAST, CBYTE, CSNG, CUNSG, CINT, CUBYTE, CLNG, CLNGINT, CUINT, CSIGN, CDBL, CSHORT, CULNGINT, Datentypen umwandeln

Letzte Bearbeitung des Eintrags am 01.06.12 um 01:46:14
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

CUSTOM

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » C » **CUSTOM**
Syntax: { PUT | DRAW STRING } [Puffer,] [STEP] (x, y), [weitere Angaben ...], CUSTOM, blender [, parameter]
Typ: Schlüsselwort
Kategorie: Grafik

CUSTOM ist ein Schlüsselwort, das im Zusammenhang mit PUT (Grafik) und DRAW STRING eingesetzt wird.

- 'blender' ist ein Pointer auf eine Funktion (siehe Tutorial zum Thema **Callback-Funktionen**) der Form
  FUNCTION blender(BYVAL src AS UINTEGER, BYVAL dst AS UINTEGER, BYVAL parameter AS ANY PTR) AS UINTEGER
  Die Parameter der Funktion 'blender' haben folgende Bedeutung:
  - 'src' ist die Farbe des auszugebenden Pixels (Bildquelle).
  - 'dst' ist die Farbe des Pixels, das überschrieben werden soll (Hintergrund).
  - Der Rückgabewert ist die Farbe des neuen Pixels.
- 'parameter' ist ein Pointer, der als drittes Argument an die Funktion 'blender' übergeben wird. Wird er ausgelassen, nimmt FreeBASIC automatisch parameter = 0 an.

Durch CUSTOM wird vor Ausgabe jedes einzelnen Pixels die FUNCTION 'blender' aufgerufen, um zu errechnen, in welcher Farbe das Pixel ausgegeben werden soll.
Diese Funktion wird für jedes neu zu zeichnende Pixel aufgerufen; die Farbe wird jedes mal neu berechnet. Achten Sie beim Einsatz dieses Aktionsworts darauf, dass die Ausführungsgeschwindigkeit unter aufwändigen Berechnungen leiden wird!

Beispiel:

```
Function dither(ByVal quellpixel As UInteger, ByVal zielpixel As
UInteger, ByVal parameter As Any Ptr) As UInteger
  ' gibt zufallsbedingt entweder das Pixel der Bildquelle oder des
Hintergrunds zurück
  Dim grenzwert As Single = 0.5
  If parameter <> 0 Then grenzwert = *CPtr(Single Ptr, parameter)

  If Rnd() < grenzwert Then
    Return quellpixel
  Else
    Return zielpixel
  End If
End Function

Dim img As Any Ptr, grenzwert As Single

' Bildschirm setzen
ScreenRes 320, 200, 16, 2
ScreenSet 0, 1

' Bild erzeugen
img = ImageCreate(32, 32)
Line img, ( 0,  0)-(15,  15), RGB(255,   0,   0), bf
Line img, (16,  0)-(31,  15), RGB(  0,   0, 255), bf
```

```
Line img, ( 0, 16)-(15,  31), RGB(  0, 255,   0), bf
Line img, (16, 16)-(31,  31), RGB(255,   0, 255), bf

' Bild mit verschiedenen Dither-Werten ausgeben
Do Until Len(Inkey)
  Cls

  grenzwert = 0.2
  Put ( 80 - 16, 100 - 16), img, Custom, @dither, @grenzwert

  ' Standardwert = 0.5
  Put (160 - 16, 100 - 16), img, Custom, @dither

  grenzwert = 0.8
  Put (240 - 16, 100 - 16), img, Custom, @dither, @grenzwert

  ScreenCopy
  Sleep 25
Loop

' Speicher freigeben
ImageDestroy img
```

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC:

- CUSTOM existiert seit FreeBASIC v0.14.
- 'parameter' existiert seit FreeBASIC v0.17. In älteren Versionen konnte dieser Parameter nicht übergeben werden.

Siehe auch:
PUT (Grafik), DRAW STRING, SCREENRES, AND (Methode), OR (Methode), XOR (Methode), PSET (Methode), PRESET (Methode), ADD, ALPHA, TRANS, Grafik

Letzte Bearbeitung des Eintrags am 27.12.12 um 00:15:10
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

CVD

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » C » **CVD**
Syntax: CVD (8byte_String)
Typ: Funktion
Kategorie: Typumwandlung

CVD konvertiert einen 8-Byte-String (ein STRING aus 8 Zeichen) in eine DOUBLE-Zahl. CVD ist die Umkehrung von MKD.

Beispiel:

```
Dim a As Double, b As String
a = 4534.42186
b = MKD(a)
Print a, CVD(b)
Sleep
```

Ausgabe:

```
4534.42186     4534.42186
```

Siehe auch:
MKSHORT, MKI, MKL, MKLONGINT, MKS, MKD, CVSHORT, CVI, CVL, CVLONGINT, CVS, Datentypen umwandeln

Letzte Bearbeitung des Eintrags am 27.12.12 um 00:15:43
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

CVI

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » C » **CVI**
Syntax:

```
CVI (Stringausdruck)
CVI<Bits> (Stringausdruck)
```

Typ: Funktion
Kategorie: Typumwandlung

CVI konvertiert einen STRING in eine INTEGER-Zahl. CVI ist die Umkehrung von MKI.

- 'Bits' gibt an, wie groß der zurückgegebene INTEGER-Ausdruck sein soll. Entspricht 'Bits' dem Wert 16, so wird CVSHORT verwendet, bei 32 CVL und bei 64 CVLONGINT. Wird 'Bits' ausgelassen, so wird in Abhängigkeit von der verwendeten Plattform der Wert 32 (x86-Architektur) oder 64 (x64-Architektur) angenommen.
- 'Stringausdruck' ist, je nach Angabe von 'Bits' bzw. der verwendeten Plattform, ein String mit der Länge 2, 4 oder 8 Byte.
- Der Rückgabewert ist, je nach Angabe von 'Bits' bzw. der verwendeten Plattform, ein SHORT ('Bits'=2), LONG ('Bits'=4) oder LONGINT ('Bits'=8).

Beispiel:

```
Dim a As Integer
a = Cvi("RIFF")
Print a, CVI<32>("RIFF")
Print Mki(a)
Sleep
```

Ausgabe:

```
 1179011410    1179011410
RIFF
```

Unterschiede zu QB:
QB unterstützt den Parameter 'Bits' nicht. INTEGER sind dort immer 16 Bit lang und CVI erwartet einen 2-Byte-String.

Unterschiede zu früheren Versionen von FreeBASIC:
Der Parameter 'Bits' existiert seit FreeBASIC v0.90.

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb erwartet CVI, wenn 'Bits' nicht angegeben wird, ebenso wie in QB einen 2-Byte-String.

Siehe auch:
MKSHORT, MKI, MKL, MKLONGINT, MKS, MKD, CVSHORT, CVD, CVL, CVLONGINT, CVS, Datentypen umwandeln

Letzte Bearbeitung des Eintrags am 04.07.13 um 00:11:05
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

CVL

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » C » **CVL**
Syntax: CVL (4byte_String)
Typ: Funktion
Kategorie: Typumwandlung

CVL konvertiert einen 4-Byte-String (ein STRING aus 4 Zeichen) in eine LONG-Zahl. CVL ist die Umkehrung vonMKL.

Beispiel:

```
Dim a As Integer, b As String
a = 4534
b = MKL(a)
Print a; CVL(b)
Sleep
```

Ausgabe:

```
 4534 4534
```

Siehe auch:
MKSHORT, MKI, MKL, MKLONGINT, MKS, MKD, CVSHORT, CVI, CVD, CVLONGINT, CVS, Datentypen umwandeln

Letzte Bearbeitung des Eintrags am 27.12.12 um 00:16:13
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

CVLONGINT

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » C » **CVLONGINT**
Syntax: CVLONGINT (8byte_String)
Typ: Funktion
Kategorie: Typumwandlung

CVLONGINT konvertiert einen 8-Byte-String (einen STRING aus 8 Zeichen) in eine LONGINT-Zahl. CVLONGINT ist die Umkehrung von MKLONGINT.

Beispiel:

```
Dim a As  LongInt, b As String
a = 4534
b = MKLongInt(a)
Print a, CVLongInt(b)
Sleep
```

Ausgabe:

```
4534          4534
```

Unterschiede zu QB: neu in FreeBASIC

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht CVLONGINT nicht zur Verfügung und kann nur über **__CVLONGINT** aufgerufen werden.

Siehe auch:
MKSHORT, MKI, MKL, MKLONGINT, MKS, MKD, CVSHORT, CVI, CVD, CVL, CVS, Datentypen umwandeln

Letzte Bearbeitung des Eintrags am 27.12.12 um 00:16:24
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

CVS

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » C » **CVS**
Syntax: CVS (4byte_String)
Typ: Funktion
Kategorie: Typumwandlung

CVS konvertiert einen 4-Byte-String (ein STRING aus 4 Zeichen) in eine SINGLE-Zahl.

Beispiel:

```
Dim a As  Single, b As String
a = 4534.4243
b = MKS(a)
Print a, CVS(b)
Sleep
```

Ausgabe:

```
4534.424       4534.424
```

Siehe auch:
MKSHORT, MKI, MKL, MKLONGINT, MKS, MKD, CVSHORT, CVI, CVD, CVLONGINT, CVL, Datentypen umwandeln

Letzte Bearbeitung des Eintrags am 27.12.12 um 00:16:37
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

CVSHORT

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » C » **CVSHORT**
Syntax: CVSHORT (2byte_String)
Typ: Funktion
Kategorie: Typumwandlung

CVSHORT konvertiert einen 2-Byte-String (einen STRING aus 2 Zeichen) in eine SHORT-Zahl. CVSHORT ist die Umkehrung zu MKSHORT.

Beispiel:

```
Dim a As Short, b As String
a = 4534
b = MKShort(a)
Print a, CVShort(b)
Sleep
```

Ausgabe:

```
 4534           4534
```

Unterschiede zu QB: neu in FreeBASIC

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht CVSHORT nicht zur Verfügung und kann nur über **__CVSHORT** aufgerufen werden.

Siehe auch:
MKSHORT, MKI, MKL, MKLONGINT, MKS, MKD, CVS, CVI, CVD, CVLONGINT, CVL, Datentypen umwandeln

Letzte Bearbeitung des Eintrags am 27.12.12 um 00:16:49
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

DATA

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » D » **DATA**
Syntax: DATA Konstante [, Konstante [,...]]
Typ: Anweisung
Kategorie: Speicher

DATA speichert Konstanten im Programm, die mit READ eingelesen werden können. Als Werte lassen sich STRINGs und Zahlen, aber auch Konstanten-Namen und einfache Rechenausdrücke. verwenden.
Variablen-Namen können in DATA-Anweisungen nicht verwendet werden.
DATA-Anweisungen sind keine ausführbaren Anweisungen und stehen normalerweise am Ende vom Programm.

Beispiel 1: Werte in DATA-Zeilen speichern:

```
DATA 1, 2, 3, 4, 17, 21, 42, "Hello", "World"
DATA 3.1412, 4*ATN(1), ATAN2(0, -1), (17+4)*2, "Hello, " & " World!"
```

Der Zugriff auf die mit DATA gespeicherten Daten erfolgt über READ.
Dabei muss der Programmierer darauf achten, dass die Werte auch Variablen mit entsprechendem Datentyp zugewiesen werden. Werden Werte in nicht dazu passende Variablen eingelesen, so werden sie zuvor mittels STR, VALINT bzw. der passenden Entsprechung umgewandelt.

Beispiel 2: 9 gespeicherte Daten einlesen:

```
DIM height(1 to 7) AS INTEGER
DIM AS STRING s1, s2
FOR a AS INTEGER = 1 TO 7
  READ height(a)
NEXT
READ s1, s2
PRINT s1 & s2
DATA 1, 2, 3, 4, 17, 21, 42, "Hello,", " World"
SLEEP
```

Beispiel 3:

```
Dim As Integer h(4)
Dim As String hs
Dim As Integer readindex

' Daten in das Array einlesen
For readindex = 0 To 4
  Read h(readindex)  ' Integer einlesen
  Print "Number" ; readindex ; " = " ; h(readindex)
Next readindex
Print

Read hs              ' String einlesen
Print  "String = " & hs
Sleep

' Datenblock.
Data 3, 234, 435/4, 23+433, 87643, "Good" & "Bye!"
```

Achten Sie darauf, nicht mehr Daten einlesen zu wollen, als über DATA zur Verfügung stehen! Wenn Sie eine Zahl einlesen wollen, aber kein DATA-Eintrag mehr vorhanden ist, wird der Wert 0 eingelesen. Versuchen Sie jedoch über das DATA-Ende hinaus einen STRING einzulesen, bricht das Programm mit einem "Segmentation fault" ab.

Mit DATA gespeicherte Werte können nur einmal eingelesen werden. Durch jede READ-Anweisung wird der jeweils nächste Wert in der Liste der DATAs eingelesen. Durch RESTORE können Daten mehrfach eingelesen werden. Nach einem RESTORE-Aufruf liest die nächste READ-Anweisung wieder von der ersten DATA-Konstante, bzw. nach dem angegeben Label. Siehe dazu das Beispiel zu RESTORE.

DATA ist eine früher häufig verwendete Methode, um Konstanten im Programm zu speichern, heutzutage sind einfache Arrays allerdings viel üblicher.

Unterschiede zu QB:

- Strings müssen in FreeBASIC (außer in -lang qb) in *"Anführungszeichen"* stehen; sie werden sonst als Werte von Funktionen behandelt.
- Leere DATA-Elemente werden in QB als 0 bzw. als leerer String "" behandelt, in FreeBASIC führen sie zu einem Compilerfehler.

Unterschiede zu früheren Versionen von FreeBASIC:

- In den FreeBASIC-Versionen 0.15 und 0.16 war es möglich, DATA-Blöcke in Prozeduren zu verwenden. In den Versionen zuvor und danach ist dies nicht möglich.

Unterschiede unter den FB-Dialektformen:

- In den Dialektformen -lang fb und -lang fblite werden DATA-Zeilen wie Konstanten behandelt, die zur Compilezeit aufgelöst werden.
- In der Dialektform -lang qb wird jede Buchstaben- und Zeichenfolge als String behandelt, auch wenn keine Anführungszeichen gesetzt sind.

Siehe auch:
READ, RESTORE, Verschiedenes

Letzte Bearbeitung des Eintrags am 27.12.12 um 00:17:57
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

DATE

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » D » **DATE**
Syntax: DATE[$]
Typ: Funktion
Kategorie: Datum und Zeit

DATE gibt das aktuelle Datum im Format mm-dd-yyyy zurück.

Das Dollarzeichen ($) als Suffix ist optional.

Beispiel: Das aktuelle Datum ausgeben:

```
PRINT DATE
```

Ausgabebeispiel:

```
12-25-2005
```

Unterschiede zu QB:
Um in FreeBASIC ein neues Datum einzustellen, müssen Sie SETDATE verwenden.

Unterschiede unter den FB-Dialektformen:

- In der Dialektform -lang qb ist das Suffix $ verbindlich.
- In den Dialektformen -lang fblite und -lang fb ist das Suffix optional.

Siehe auch:
TIMER, SETDATE, TIME, SETTIME, Datum und Zeit

Letzte Bearbeitung des Eintrags am 27.12.12 um 00:18:15
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

DATEADD

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » D » **DATEADD**
Syntax: DATEADD (Intervall, n, Ausgangsserial)
Typ: Funktion
Kategorie: Datum und Zeit

Anmerkung:
Um diese Funktion nutzen zu können, muss die Datei datetime.bi in Ihren Quellcode eingebunden werden, z. B. mit INCLUDE. Alternativ können Sie auch die Datei vbcompat.bi einbinden, da dadurch auch automatisch datetime.bi eingebunden wird.

DATEADD gibt eine Serial Number aus, die eine Zeit symbolisiert, die 'n' 'Intervalle' hinter dem 'Ausgangsserial' liegt.

- 'Intervall' ist ein STRING, der angibt, wie groß die Intervalle sein sollen, die zum 'AusgangsSerial' hinzugezählt werden sollen. Die möglichen Angaben werden weiter unten aufgelistet.
- 'n' ist die Anzahl der Intervalle, die zum 'Ausgangsserial' hinzugezählt werden sollen. 'n' ist vom Typ INTEGER, kann also nur ganzzahlige Elemente enthalten. Es ist also nicht möglich, zu einer Serial Number ein *halbes* Jahr hinzuzuzählen. Es ist allerdings durchaus möglich, eine negative Zahl anzugeben; in diesem Fall wird ein Serial ausgegeben, das vor 'Ausgangsserial' liegt.
- 'Ausgangsserial' ist eine Serial Number, die einen beliebigen Zeitpunkt angibt, zu dem ein bestimmter Zeitraum hinzugezählt werden soll.
- Der Rückgabewert ist ein DOUBLE mit der Serial Number des errechneten Zeitpunkts.

Folgende Intervallangaben sind möglich:

Wert	Bedeutung
"yyyy"	Jahre
"q"	Quartale (drei Monate)
"m"	Monate
"ww"	Wochen
"d", "w" oder "y"	Tage
"h"	Stunden
"n"	Minuten
"s"	Sekunden

Beispiel:

```
"hlstring">"vbcompat.bi"
DIM AS DOUBLE SerialA, SerialB

SerialA = NOW
PRINT "Heute:"
PRINT FORMAT(SerialA, "dd.mm.yyyy, hh:mm:ss")
SerialB = DATEADD("ww", 3, serialA)
PRINT "In drei Wochen:"
PRINT FORMAT(SerialB, "dd.mm.yyyy, hh:mm:ss")
SLEEP
```

Ausgabebeispiel:

```
Heute:
13.12.2005, 15:10:41
In drei Wochen:
03.01.2006, 15:10:41
```

Unterschiede zu QB: existiert nur in QBX PDS und in VBWIN.

Unterschiede zu früheren Versionen von FreeBASIC: existiert seit FreeBASIC v0.15

Siehe auch:
NOW, DATESERIAL, DATEVALUE, TIMESERIAL, TIMEVALUE, YEAR, MONTH, DAY, WEEKDAY, HOUR, MINUTE, SECOND, MONTHNAME, WEEKDAYNAME, DATEDIFF, DATEPART, FORMAT, ISDATE, Serial Numbers, Datum und Zeit

Letzte Bearbeitung des Eintrags am 06.06.12 um 01:04:03
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

DATEDIFF

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » D » **DATEDIFF**
Syntax: DATEDIFF (Intervall, SerialA, SerialB [, erster_Tag_der_Woche [, erste_Woche_des_Jahres]])
Typ: Funktion
Kategorie: Datum und Zeit

Anmerkung:
Um diese Funktion nutzen zu können, muss die Datei datetime.bi in Ihren Quellcode eingebunden werden, z. B. mit INCLUDE. Alternativ können Sie auch die Datei vbcompat.bi einbinden, da dadurch auch automatisch datetime.bi eingebunden wird.

DATEDIFF gibt den zeitlichen Unterschied zwischen zwei Serial Numbers aus.

- 'SerialA' und 'SerialB' sind die Zeitpunkte, deren Unterschied ermittelt werden soll. Wenn 'SerialA' größer ist als 'SerialB', so ist das Ergebnis negativ.
- 'Intervall' ist ein String, der angibt, wie groß die Intervalle sein sollen, in denen der zeitliche Unterschied zwischen 'SerialA' und 'SerialB' gemessen werden soll. Die möglichen Angaben werden weiter unten aufgelistet.
- 'erster_Tag_der_Woche' wirkt sich auf das Ergebnis aus, wenn als Intervall "ww" angegeben wurde; siehe unten.
- 'erste_Woche_des_Jahres' wirkt sich auf das Ergebnis aus, wenn als Intervall "ww" angegeben wurde; siehe unten.

Folgende Intervallangaben sind möglich:

Wert	Bedeutung
"yyyy"	Jahre
"q"	Quartale (drei Monate)
"m"	Monate
"w"	Sieben-Tage-Einheiten (Wochen)
"ww"	Kalenderwochen (beginnen und enden bei einem bestimmten Wochentag)
"d" oder "y"	Tage
"h"	Stunden
"n"	Minuten
"s"	Sekunden

Wenn als 'Intervall' der Wert "ww" angegeben wird, können weitere Angaben zu 'erster_Tag_der_Woche' und 'erste_Woche_des_Jahres' gemacht werden.

'erster_Tag_der_Woche' ist einer von diesen Werten:

Wert	alternatives Symbol	Tag
ausgelassen	-	Sonntag
0	fbUseSystem	lokal eingestelltes System
1	fbSunday	Sonntag
2	fbMonday	Montag
3	fbTuesday	Dienstag
4	fbWednesday	Mittwoch
5	fbThursday	Donnerstag
6	fbFriday	Freitag

7 fbSaturday Samstag
'erste_Woche_des_Jahres' ist einer von diesen Werten:

Wert	alternatives Symbol	Bedeutung
0 oder ausgelassen	fbUseSystem	lokal eingestelltes System
1	fbFirstJan1	Beginne mit der Woche des ersten Januars als erste Kalenderwoche des Jahres
2	fbFirstFourDays	Beginne mit der ersten Woche, die vier Tage hat als erste Kalenderwoche des Jahres
3	fbFirstFullWeek	Beginne mit der ersten ganzen Woche des Jahres als erste Kalenderwoche des Jahres

Beispiel:

```
"hlstring">"vbcompat.bi"
DIM AS DOUBLE SerialA, SerialB

SerialA = NOW
SerialB = DATESERIAL(YEAR(NOW), 12, 25) ' Weihnachten dieses Jahr
PRINT "Heute ist der "; FORMAT(SerialA, "dd.mm.yyyy.")
PRINT "Noch";
PRINT DATEDIFF("d", SerialA, SerialB);
PRINT " Tage bis Weihnachten!"
SLEEP
```

Ausgabebeispiel:

```
Heute ist der 13.12.2005.
Noch 12 Tage bis Weihnachten!
```

Unterschiede zu QB: existiert nur in QBX PDS und in VBDOS

Unterschiede zu früheren Versionen von FreeBASIC: existiert seit FreeBASIC v0.15

Siehe auch:
NOW, DATESERIAL, DATEVALUE, TIMESERIAL, TIMEVALUE, YEAR, MONTH, DAY, WEEKDAY, HOUR, MINUTE, SECOND, MONTHNAME, WEEKDAYNAME, DATEADD, DATEPART, FORMAT, ISDATE, Serial Numbers, Datum und Zeit

Letzte Bearbeitung des Eintrags am 07.06.12 um 00:16:29
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

DATEPART

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » D » **DATEPART**
Syntax: DATEPART (Intervall, Serial [, erster_Tag_der_Woche [, erste_Woche_des_Jahres]])
Typ: Funktion
Kategorie: Datum und Zeit

Anmerkung:
Um diese Funktion nutzen zu können, muss die Datei datetime.bi in Ihren Quellcode eingebunden werden, z. B. mit INCLUDE. Alternativ können Sie auch die Datei vbcompat.bi einbinden, da dadurch auch automatisch datetime.bi eingebunden wird.

DATEPART extrahiert eine Teilangabe einer Serial Number.

- 'Intervall' ist ein String, der angibt, welcher Teil von 'Serial' zurückgegeben werden soll. Die möglichen Angaben werden weiter unten aufgelistet.
- 'Serial' ist eine Serial Number, von der eine bestimmte Teilangabe ausgegeben werden soll.
- 'erster_Tag_der_Woche' wirkt sich auf das Ergebnis aus, wenn als Intervall "ww" angegeben wurde; siehe unten.
- 'erste_Woche_des_Jahres' wirkt sich auf das Ergebnis aus, wenn als Intervall "ww" angegeben wurde; siehe unten.
- Der Rückgabewert ist ein INTEGER mit der extrahierten Teilangabe.

Folgende Intervallangaben sind möglich:

Wert	Bedeutung
"yyyy"	Jahre
"q"	Quartale (drei Monate)
"m"	Monate
"ww"	Wochen innerhalb des Jahres
"w"	Tage innerhalb der Woche
"d"	Tage innerhalb des Monats
"y"	Tage innerhalb des Jahres
"h"	Stunden
"n"	Minuten
"s"	Sekunden

Wenn als 'Intervall' der Wert "ww" angegeben wird, können weitere Angaben zu 'erster_Tag_der_Woche' und 'erste_Woche_des_Jahres' gemacht werden.

'erster_Tag_der_Woche' ist einer von diesen Werten:

Wert	alternatives Symbol	Tag
ausgelassen	-	Sonntag
0	fbUseSystem	lokal eingestelltes System
1	fbSunday	Sonntag
2	fbMonday	Montag
3	fbTuesday	Dienstag
4	fbWednesday	Mittwoch
5	fbThursday	Donnerstag

6	fbFriday	Freitag
7	fbSaturday	Samstag

'erste_Woche_des_Jahres' ist einer von diesen Werten:

Wert	alternatives Symbol	Bedeutung
0 oder ausgelassen	fbUseSystem	lokal eingestelltes System
1	fbFirstJan1	Beginne mit der Woche des ersten Januars als erste Kalenderwoche des Jahres
2	fbFirstFourDays	Beginne mit der ersten Woche, die vier Tage hat als erste Kalenderwoche des Jahres
3	fbFirstFullWeek	Beginne mit der ersten ganzen Woche des Jahres als erste Kalenderwoche des Jahres

Beispiel:

```
"hlstring">"vbcompat.bi"

PRINT "Wir befinden uns im ";
PRINT DATEPART("q", NOW); ". Quartal."
PRINT "Es ist die "; DATEPART("ww", NOW, , fbFirstFullWeek);
PRINT ". ganze Kalenderwoche."
PRINT "Heute ist der "; DATEPART("w", NOW, fbMonday);
PRINT ". Tag seit Montag."
SLEEP
```

Unterschiede zu QB: existiert nur in QBX PDS und in VBWIN.

Unterschiede zu früheren Versionen von FreeBASIC: existiert seit FreeBASIC v0.15

Siehe auch:
NOW, DATESERIAL, DATEVALUE, TIMESERIAL, TIMEVALUE, YEAR, MONTH, DAY, WEEKDAY, HOUR, MINUTE, SECOND, MONTHNAME, WEEKDAYNAME, DATEADD, DATEDIFF, FORMAT, ISDATE, Serial Numbers, Datum und Zeit

Letzte Bearbeitung des Eintrags am 07.06.12 um 00:21:12
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

DATESERIAL

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » D » **DATESERIAL**
Syntax: DATESERIAL (Jahr, Monat, Tag)
Typ: Funktion
Kategorie: Datum und Zeit

Anmerkung:
Um diese Funktion nutzen zu können, muss die Datei datetime.bi in Ihren Quellcode eingebunden werden, z. B. mit INCLUDE. Alternativ können Sie auch die Datei vbcompat.bi einbinden, da dadurch auch automatisch datetime.bi eingebunden wird.

DATESERIAL wandelt ein angegebenes Datum in eine Serial Number um. Diese Funktion wird vom Compiler solange ignoriert, bis die Datei datetime.bi (oder vbcompat.bi) in den Quellcode eingebunden wurde.

- 'Jahr' ist eine vierstellige Jahreszahl.
- 'Monat' ist eine Zahl zwischen 1 und 12, die den Monat angibt.
- 'Tag' ist der eine Zahl zwischen 1 und 31, die den Tag angibt.
- Der Rückgabetyp ist ein DOUBLE, der eine Serial Number repräsentiert. Durch die interne Darstellung von DOUBLE-Werten kann es zu Ungenauigkeiten bei großen Werten kommen.
- Wenn Sie "unsinnige" Daten angeben, wie z.B. den 32.14.2004, wird der Fehler automatisch umgerechnet; das Ergebnis wäre hier die Serial Number zum 4.3.2005.

Beispiel:

```
"hlstring">"vbcompat.bi"
Dim a As Double
a = DATESERIAL(2005, 11, 28) + TIMESERIAL(19, 3, 26)
PRINT FORMAT(a, "dd.mm.yyyy hh:mm:ss")
SLEEP
```

Ausgabe:

```
28.11.2005 19:03:26
```

Unterschiede zu QB: existiert nur in QBX PDS und in VBDOS

Unterschiede zu früheren Versionen von FreeBASIC: existiert seit FreeBASIC v0.15

Siehe auch:
NOW, DATEDIFF, DATEVALUE, TIMESERIAL, YEAR, MONTH, DAY, WEEKDAY, MONTHNAME, WEEKDAYNAME, DATEADD, DATEPART, FORMAT, ISDATE, Serial Numbers, Datum und Zeit

Letzte Bearbeitung des Eintrags am 07.06.12 um 00:41:04
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

DATEVALUE

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » D » **DATEVALUE**
Syntax: DATEVALUE (Datum)
Typ: Funktion
Kategorie: Datum und Zeit

Anmerkung:
Um diese Funktion nutzen zu können, muss die Datei datetime.bi in Ihren Quellcode eingebunden werden, z. B. mit INCLUDE. Alternativ können Sie auch die Datei vbcompat.bi einbinden, da dadurch auch automatisch datetime.bi eingebunden wird.

DATEVALUE verwandelt einen STRING mit einer Datumsangabe in eine Serial Number.

- 'Datum' ist der String, der in eine Serial Number verwandelt werden soll.
- Der Rückgabewert ist ein INTEGER, der den Datumsanteil der Serial Number enthält.

'Datum' muss in einem der folgenden Formate übergeben werden (dies ist jedoch abhängig von der Lokalisierung des Systems):

- dd.mm.yyyy
- dd-mm-yyyy
- dd/mm/yyyy
- dd.mm.yy
- dd-mm-yy
- dd/mm/yy
- d.m.yyyy
- d-m-yyyy
- d/m/yyyy
- d.m.yy
- d-m-yy
- d/m/yy

Wenn das System auf das amerikanische Datumsformat mm-dd-yyyy eingestellt ist, muss 'Datum' auch in einem entsprechenden Format übergeben werden.

Beispiel:
Das heutige Datum mit DATE ermitteln und in eine Serial Number verwandeln:

```
"hlstring">"datetime.bi"
Dim dv As Integer
Dim dt As String
' DATE gibt das aktuelle Datum im Format mm-dd-yyyy zurück.
dt = Date
' DATEVALUE(Datum) erwartet das Datum im Format dd-mm-yyyy
Swap dt&"hlzahl">0], dt[3]  ' nach dd-mm-yyyy wandeln
Swap dt&"hlzahl">1], dt[4]

dv = DateValue(dt)
Print dv
Sleep
```

Sind die Regionaleinstellungen des Systems nicht bekannt, kann die Serial Number des aktuellen Datums auch folgendermaßen ermittelt werden:

```
"hlstring">"datetime.bi"
Dim dv As Integer = Int(Now)
Print dv
Sleep
```

Unterschiede zu QB: existiert nur in QBX PDS und in VBDOS.

Unterschiede zu früheren Versionen von FreeBASIC: existiert seit FreeBASIC v0.15

Siehe auch:
NOW, DATESERIAL, DATEDIFF, TIMEVALUE, YEAR, MONTH, DAY, WEEKDAY, MONTHNAME, WEEKDAYNAME, DATEADD, DATEPART, FORMAT, ISDATE, Serial Numbers, Datum und Zeit

Letzte Bearbeitung des Eintrags am 13.06.13 um 21:30:08
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

DAY

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » D » **DAY**
Syntax: DAY (Serial)
Typ: Funktion
Kategorie: Datum und Zeit

Anmerkung:
Um diese Funktion nutzen zu können, muss die Datei datetime.bi in Ihren Quellcode eingebunden werden, z.B. mit INCLUDE. Alternativ können Sie auch die Datei vbcompat.bi einbinden, da dadurch auch automatisch datetime.bi eingebunden wird.

DAY extrahiert den Tag einer Serial Number.

- 'Serial' ist ein DOUBLE-Wert, der als Serial Number behandelt wird.
- Der Rückgabewert ist der Tag des Monats, der in der Serial Number gespeichert ist.

Beispiel:
Den heutigen Tag des Monats aus NOW extrahieren:

```
"hlstring">"vbcompat.bi"
DIM tag AS INTEGER
tag = DAY(NOW)
```

Unterschiede zu QB: existiert nur in QBX PDS und in VBDOS.

Unterschiede zu früheren Versionen von FreeBASIC: existiert seit FreeBASIC v0.15

Siehe auch:
NOW, DATESERIAL, DATEVALUE, YEAR, MONTH, DATEDIFF, WEEKDAY, HOUR, MINUTE, SECOND, MONTHNAME, WEEKDAYNAME, DATEADD, DATEPART, FORMAT, ISDATE, Serial Numbers, Datum und Zeit

Letzte Bearbeitung des Eintrags am 27.12.12 um 01:00:10
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

DEALLOCATE

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » D » **DEALLOCATE**
Syntax: DEALLOCATE Pointer
Typ: Anweisung
Kategorie: Speicher

DEALLOCATE gibt einen mit ALLOCATE reservierten Speicher wieder frei.

'Pointer' ist ein Pointer auf den Beginn des freizugebenden Speicherbereichs.

Der Bezeichner des Pointers muss nicht derselbe sein wie der, mit dem der Speicherbereich alloziert wurde; lediglich die Adresse muss dieselbe sein.

Es wird empfohlen, den Pointer sofort null zu setzen, sobald der Speicherbereich freigegeben wurde, um zu verhindern, dass immer noch darauf zugegriffen wird.

DEALLOCATE ist kein Teil der FreeBASIC Runtime Library, sondern ein Alias von free der C-Lib.
Achtung: Es kann nicht garantiert werden, dass diese Funktion auf allen Plattformen Multithreading unterstützt, d.h. thread-safe ist. Unter Windows und Linux sind aktuell durch die verwendeten Implementationen der Betriebssysteme aber keine Probleme zu erwarten.

Beispiel:

```
DIM AS INTEGER PTR p1, p2

' 4 Bytes reservieren
p1 = ALLOCATE(4)

' p2 soll auf denselben Speicherbereich zeigen
p2 = p1

' den Speicherbereich wieder freigeben; er wurde
' mit p1 erstellt, kann aber mit p2 freigegeben
' werden, da beide auf dieselbe Adresse zeigen.
DEALLOCATE p2

' p1 auf null setzen, um zu verhindern, dass auf
' den inzwischen freigegebenen Speicher
' zugegriffen wird
p1 = 0
```

Unterschiede zu QB: neu in FreeBASIC

Plattformbedingte Unterschiede:
Es kann nicht garantiert werden, dass die Prozedur auf allen Plattformen thread-safe ist.

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht DEALLOCATE nicht zur Verfügung und kann nur über **__DEALLOCATE** aufgerufen werden.

Siehe auch:
ALLOCATE, CALLOCATE, REALLOCATE, Pointer, Speicher

Letzte Bearbeitung des Eintrags am 05.04.14 um 15:53:47
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

DECLARE

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » D » **DECLARE**
Syntax A:

```
DECLARE SUB Name [Aufrufkonvention] [OVERLOAD] [[LIB "DLLName"] _
      ALIAS "AliasName"] [ (  _
      [{BYVAL | BYREF } ] [{Parameter1 | ()}] AS Typ [= Wert]   [, _
      [{BYVAL | BYREF } ] [{Parameter2 | ()}] AS Typ [= Wert] ] [, ...] )
]
```

Syntax B:

```
DECLARE FUNCTION Name [Aufrufkonvention] [OVERLOAD] [[LIB "DLLName"] _
      ALIAS "AliasName"] [ (  _
      [{ BYVAL | BYREF } ] [{Parameter1 | ()}] AS Typ [= Wert]   [, _
      [{ BYVAL | BYREF } ] [{Parameter2 | ()}] AS Typ [= Wert] ] [, ...]
) AS Rückgabetyp
```

Typ: Anweisung
Kategorie: Deklaration

Anmerkung zur Syntax: Unterstriche (_) am Zeilenende werden von FreeBASIC so interpretiert, als wäre die Zeile nicht unterbrochen; dies dient nur der besseren Übersichtlichkeit und hat letzt–end–lich keine Auswirkungen auf die Programmausführung.

DECLARE deklariert eine neue Prozedur. Dies ist notwendig, wenn die Prozedur aufgerufen werden soll, bevor sie im Quellcode definiert wird.

- 'Name' ist der Bezeichner, unter dem die SUB bzw. FUNCTION aufgerufen wird.
- 'Aufrufkonvention' ist ein Schlüsselwort, das angibt, in welcher Reihenfolge die Parameter übergeben werden sollen. Möglich sind: STDCALL, CDECL und PASCAL.
- 'OVERLOAD' erlaubt, Prozeduren mit unterschiedlicher Parameterliste, aber gleichem Prozedurnamen deklarieren. Siehe dazu den Referenzeintrag OVERLOAD.
- 'DLLName' ist der Name der LIB/DLL, in der sich die Prozedur befindet.
- 'AliasName' ist der Name, unter dem die Prozedur in der LIB/DLL aufgeführt ist. Da diese manchmal Zeichen enthalten, die unter FreeBASIC nicht erlaubt sind, ist es möglich, ihnen "einen anderen Namen zu geben".
- 'Parameter1', 'Parameter2', '...' stehen für Parameter, die an die Prozedur übergeben werden sollen (siehe auch Parameterübergabe). Ihre Namen sind hier eigentlich unbedeutend und können ausgelassen werden (anonyme Parameter). Eine Deklaration, in der die Parameter bezeichnet werden, ist jedoch einfacher nachzuvollziehen.
- Wird der Parametername eines Arrays ausgelassen, muss '()' verwendet werden, um dem Compiler zu signalisieren, dass es sich beim anonymen Parameter um ein Array handelt.
- 'Typ' ist der Datentyp der übergebenen Parameter; siehe Datentypen.
- Durch die Klausel '= Wert' wird ein Parameter als optional deklariert; er kann beim Aufruf der Prozedur ausgelassen werden. In diesem Fall wird an die Prozedur für diesen Parameter 'Wert' übergeben. 'Wert' darf ein einfacher Ausdruck sein, der Konstanten, Operatoren und Funktionen, jedoch keine Variablen enthält.
- 'BYVAL' bzw. 'BYREF' gibt an, ob der Parameter als Wert (by value) oder als Referenz (by reference) übergeben wird. Näheres dazu finden Sie unter BYVAL.
- 'Rückgabetyp' hinter der Parameterliste ist der Typ des Rückgabewertes der FUNCTION.

DECLARE wird auch bei UDTs verwendet, um einen CONSTRUCTOR bzw. DESTRUCTOR, eine PROPERTY oder die Überladung von OPERATORen zu deklarieren.

Werden Prozeduren vor ihrem ersten Aufruf definiert, dann ist keine vorherige Deklaration nötig. Jedoch kann eine kompakt zusammenstehende Deklaration aller im Programm auftretenden Prozeduren zur Lesbarkeit des Quellcodes beitragen.

Beispiel:

```
DECLARE FUNCTION twice (AS INTEGER) AS INTEGER
DECLARE SUB PrintTimes (AS STRING, BYVAL AS INTEGER = 1)

FUNCTION twice (x AS INTEGER) AS INTEGER
  twice = x * 2
END FUNCTION

SUB PrintTimes (ToPrint AS STRING, BYVAL times AS INTEGER = 1)
  FOR i AS INTEGER = 1 TO times
    PRINT ToPrint
  NEXT
END SUB

PRINT "Das Doppelte von 42 ist: "; twice(42)
PrintTimes "Einmal"
PrintTimes "Zehn mal", 10
Sleep
```

Beispiel 2 mit anonymen Array-Parameter:

```
Declare Sub foo(() As Integer)

Dim As Integer bar(0 To ...) = {0, 1, 2}
foo(bar())
Sleep

Sub foo(array() As Integer)
  For i As Integer = LBound(array) To UBound(array)
    Print array(i)
  Next
End Sub
```

DECLARE-Anweisungen können direkt an den Beginn eines BAS-Moduls gesetzt werden. Häufig werden sie jedoch in eine eigene Datei ausgelagert und dann über "reflinkicon" href="temp0573.html">-lang qb und -lang deprecated werden Parameter standardmäßig BYREF übergeben.

- Deklarationen innerhalb eines TYPE-Blocks sind nur in der Dialektform -lang fb erlaubt.

Unterschiede zu früheren Versionen von FreeBASIC:
Seit FreeBASIC v0.90 können in DECLARE-Zeilen die Namen von Array-Parametern ausgelassen werden.

Siehe auch:
SUB, FUNCTION, STATIC (Klausel), PASCAL, CDECL, STDCALL, BYVAL, BYREF, SHARED, LIB, ALIAS, OVERLOAD, OPERATOR, EXPORT, Prozeduren, Parameterübergabe

Letzte Bearbeitung des Eintrags am 02.07.13 um 21:50:52

FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

DEFBYTE

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » D » **DEFBYTE**
Syntax: DEFBYTE Buchstabenbereich [, Buchstabenbereich [, ...]]
Typ: Anweisung
Kategorie: Datentypen

DEFBYTE setzt den Standarddatentyp zu BYTE. Variablen, die mit einem Buchstaben beginnen, der im angegebenen Buchstabenbereich liegt, werden als BYTE dimensioniert.

Diese Anweisung kann **nur bis FreeBASIC v0.16** verwendet werden oder in entsprechend höheren Versionen, die mit der Kommandozeile -lang deprecated compiliert wurden. Wird seit FreeBASIC v0.17 mit -lang fb compiliert, dann müssen alle Datentypen explizit deklariert werden.

Der Ausdruck 'Buchstabenbereich' ist entweder der Form Startbuchstabe-Endbuchstabe, oder einfach nur ein Buchstabe.

Beispiel:

```
"hlstring">"deprecated"
DEFBYTE b, d-f
DIM bNumber
DIM eNumber
```

Dadurch werden 'bNumber' und 'eNumber' zu einem BYTE, da ihre Anfangsbuchstaben b sind bzw. im Bereich von d bis f liegen.

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC:
Seit FreeBASIC v0.17 können Standardtypen nur noch dann definiert werden, wenn die Kommandozeilenoption -lang deprecated angegeben wurde. Soll mit -lang fb compiliert werden, müssen alle Datentypen explizit deklariert werden.

Siehe auch:
DEFxxx, DIM, Datentypen und Deklarationen

Letzte Bearbeitung des Eintrags am 19.01.13 um 16:21:07
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

DEFDBL

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » D » **DEFDBL**
Syntax: DEFDBL Buchstabenbereich [, Buchstabenbereich [, ...]]
Typ: Anweisung
Kategorie: Datentypen

DEFDBL setzt den Standarddatentyp zu DOUBLE. Variablen, die mit einem Buchstaben beginnen, der im angegebenen Buchstabenbereich liegt, werden als DOUBLE dimensioniert.

Diese Anweisung kann **nur bis FreeBASIC v0.16** verwendet werden oder in entsprechend höheren Versionen, die mit der Kommandozeile -lang deprecated compiliert wurden. Wird seit FreeBASIC v0.17 mit -lang fb compiliert, dann müssen alle Datentypen explizit deklariert werden.

Der Ausdruck 'Buchstabenbereich' ist entweder der Form Startbuchstabe-Endbuchstabe, oder einfach nur ein Buchstabe.

Beispiel:

```
"hlstring">"deprecated"
DEFDBL d, e-g
DIM dNumber
DIM fNumber
```

Dadurch werden 'dNumber' und 'fNumber' zu einem DOUBLE, da ihre Anfangsbuchstaben d sind bzw. im Bereich von e bis g liegen.

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC:
Seit FreeBASIC v0.17 können Standardtypen nur noch dann definiert werden, wenn die Kommandozeilenoption -lang deprecated angegeben wurde. Soll mit -lang fb compiliert werden, müssen alle Datentypen explizit deklariert werden.

Siehe auch:
DEFxxx, DIM, Datentypen und Deklarationen

Letzte Bearbeitung des Eintrags am 19.01.13 um 16:22:26
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

DEFINE (Meta)

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » D » **DEFINE (Meta)**
Syntax A: #DEFINE Symbol [Ausdruck]
Syntax B: #DEFINE Symbol (Parameter [, Parameter] [...]) Makro
Typ: Metabefehl
Kategorie: Metabefehle

#DEFINE definiert ein Symbol mit benutzerdefinierter Bedeutung oder ein Makro.

- 'Symbol' ist ein Bezeichner, der direkt vor der Compilation durch seinen Ausdruck ersetzt wird. Selbst wenn es keinen Wert besitzt (wie in Syntax A ohne Angabe von 'Ausdruck') kann das Symbol bereits mit DEFINED verwendet werden.
- 'Parameter' sind Parameter, die im definierten Makro eingesetzt werden können.
- Mit der Ellipsis ... (Auslassung) kann eine variable Zahl an Parametern übergeben werden. Dann wird an der Stelle des letzten angegebenen Parameters die komplette restliche Parameterliste eingefügt. Siehe dazu Beispiel 2.
- Das Symbol ist nur in dem SCOPE-Block (auch SUB, FOR...NEXT, WHILE...WEND, DO...LOOP usw.) sichtbar, in dem es definiert wurde. Wenn es auf Modulebene definiert wurde, ist es im gesamten Modul sichtbar.
- NAMESPACEs haben keinen Effekt auf die Sichtbarkeit des Symbols.

Ein Makro ist im Grunde eine FUNCTION; direkt vor der Compilation wird der Makro-Bezeichner durch seinen Ausdruck ersetzt. Dabei können in diesem Ausdruck Parameter verwendet werden, die zuvor unter 'Parameter' festgelegt wurden. Makros werden meist für häufig auftretende mathematische Formeln verwendet.

Durch einfaches Definieren eines Symbols kann das Programm komplett anders laufen. Diese Funktion ermöglicht FreeBASIC u. a., ein Programm portabel zu verschiedenen Betriebssystemen zu halten, indem z. B. bestimmte Codeabschnitte nur unter Windows und andere nur unter Linux eingebunden werden. Einige Symbole und Makros sind bereits vordefiniert; siehe Präprozessoren.

Makros können auch als Parameter für andere Makros übergeben werden. In diesem Fall müssen auch dann Klammern für die Parameterliste angegeben werden, wenn es sich um ein parameterloses Makro handelt.

Beispiel 1:
Makros erstellen, die zwischen Altgrad DEG, Kreis mit 360°), Neugrad (GON, Kreis mit 400gon) und Bogenmaß (RAD, Kreis mit 2 PI) umrechnen

```
"hlzeichen">(4 * ATN(1))
"hlzeichen">(deg) ( (deg / 180) * PI)
"hlzeichen">(rad) ( (rad / PI) * 180 )
"hlzeichen">(gon) ( (gon / 200) * PI)
"hlzeichen">(rad) ( (rad / PI) * 200 )

PRINT "PI ="; PI
' Ausgabe: 3.141592654
PRINT "180 im Bogenmaß: " & deg2rad(180)
' Ausgabe: 3.141592654
PRINT "3.141592654 in Altgrad°(DEG): " & rad2deg(pi)
' Ausgabe: 180
PRINT "200gon im Bogenmaß: " & gon2rad(200)
' Ausgabe: 3.141592654
PRINT "3.141592654 in Neugrad(GON): " & rad2gon(pi)
```

```
' Ausgabe: 200
SLEEP
```

Beispiel 2 mit variabler Anzahl von Parametern:

```
"hlstring">"crt.bi"
"hlzeichen">(Format, args...) fprintf(stderr, Format, args)
eprintf(!"Hello from printf: %i %s %i\n", 5, "test", 123)

"hlzeichen">(a, b...) a
"hlzeichen">(a, b...) b
PRINT "car (1, 2, 3, 4) = "; car(1, 2, 3, 4)
PRINT "cdr (1, 2, 3, 4) = "; cdr(1, 2, 3, 4)
SLEEP
```

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC:

- Seit FreeBASIC v0.22 kann eine variable Parameterliste übergeben werden.
- Seit FreeBASIC v0.16 können Makros auch als Parameter für andere Makros übergeben werden. In diesem Fall müssen auch dann Klammern für die Parameterliste angegeben werden, wenn es sich um ein parameterloses Makro handelt.

Siehe auch:
DEFINED, UNDEF (Meta), MACRO (Meta), IF (Meta), IFDEF (Meta), IFNDEF (Meta), Präprozessoren, Präprozessor-Anweisungen

Letzte Bearbeitung des Eintrags am 19.01.13 um 16:23:24
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

DEFINED

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » D » **DEFINED**
Syntax: DEFINED (Symbol)
Typ: Funktion
Kategorie: Metabefehle

DEFINED gibt true (-1) zurück, wenn das angegebene Symbol bereits definiert wurde, oder false (0), wenn es bislang noch unbekannt ist. Bei dem Symbol kann es sich um ein FreeBASIC-Schlüsselwort, eine Variable (siehe DIM), eine Konstante, eine Prozedur (siehe SUB, FUNCTION) oder ein über #DEFINE oder MACRO definiert worden sein.

DEFINED wird zusammen mit #IF verwendet. Eine andere Verwendung ist nicht möglich. Dieselbe Funktion erfüllt #IFDEF; DEFINED ist aber flexibler, da es mehr als eine Überprüfung zur gleichen Zeit zulässt.

Beispiel:

```
"hlkw0">DEFINED(LogX)
  ' ... Anweisungen
"reflinkicon" href="temp0108.html">DEFINE (Meta), IFDEF (Meta), IFNDEF
(Meta), UNDEF (Metabefehl), Präprozessoren, Präprozessor-Anweisungen
```

Letzte Bearbeitung des Eintrags am 27.12.12 um 00:19:52
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

DEFINT

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » D » **DEFINT**
Syntax: DEFINT Buchstabenbereich [, Buchstabenbereich [, ...]]
Typ: Anweisung
Kategorie: Datentypen

DEFINT setzt den Standarddatentyp zu INTEGER. Variablen, die mit einem Buchstaben beginnen, der im angegebenen Buchstabenbereich liegt, werden als INTEGER dimensioniert.

Diese Anweisung kann **nur bis FreeBASIC v0.16** verwendet werden oder in entsprechend höheren Versionen, die mit der Kommandozeile -lang deprecated compiliert wurden. Wird seit FreeBASIC v0.17 mit -lang fb compiliert, dann müssen alle Datentypen explizit deklariert werden.

Der Ausdruck 'Buchstabenbereich' ist entweder der Form Startbuchstabe-Endbuchstabe, oder einfach nur ein Buchstabe.

Beispiel:

```
"hlstring">"deprecated"
DEFINT i, a-f
DIM iNumber
DIM eNumber
```

Dadurch werden 'iNumber' und 'eNumber' zu einem INTEGER, da ihre Anfangsbuchstaben i sind bzw. im Bereich von a bis f liegen.

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC:
Seit FreeBASIC v0.17 können Standardtypen nur noch dann definiert werden, wenn die Kommandozeilenoption -lang deprecated angegeben wurde. Soll mit -lang fb compiliert werden, müssen alle Datentypen explizit deklariert werden.

Siehe auch:
DEFxxx, DIM, Datentypen und Deklarationen

Letzte Bearbeitung des Eintrags am 19.01.13 um 16:24:01
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

DEFLNG

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » D » **DEFLNG**
Syntax: DEFLNG Buchstabenbereich [, Buchstabenbereich [, ...]]
Typ: Anweisung
Kategorie: Datentypen

DEFLNG setzt den Standarddatentyp zu LONG. Variablen, die mit einem Buchstaben beginnen, der im angegebenen Buchstabenbereich liegt, werden als LONG dimensioniert.

Diese Anweisung kann **nur bis FreeBASIC v0.16** verwendet werden oder in entsprechend höheren Versionen, die mit der Kommandozeile -lang deprecated compiliert wurden. Wird seit FreeBASIC v0.17 mit -lang fb compiliert, dann müssen alle Datentypen explizit deklariert werden.

Der Ausdruck 'Buchstabenbereich' ist entweder der Form Startbuchstabe-Endbuchstabe, oder einfach nur ein Buchstabe.

Beispiel:

```
"hlstring">"deprecated"
DEFLNG l, d-f
DIM lNumber
DIM eNumber
```

Dadurch werden 'lNumber' und 'eNumber' zu einem LONG, da ihre Anfangsbuchstaben l sind bzw. im Bereich von d bis f liegen.

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC:
Seit FreeBASIC v0.17 können Standardtypen nur noch dann definiert werden, wenn die Kommandozeilenoption -lang deprecated angegeben wurde. Soll mit -lang fb compiliert werden, müssen alle Datentypen explizit deklariert werden.

Siehe auch:
DEFxxx, DIM, Datentypen und Deklarationen

Letzte Bearbeitung des Eintrags am 19.01.13 um 16:24:56
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

DEFLONGINT

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » D » **DEFLONGINT**
Syntax: DEFLONGINT Buchstabenbereich [, Buchstabenbereich [, ...]]
Typ: Anweisung
Kategorie: Datentypen

DEFLONGINT setzt den Standarddatentyp zu LONGINT. Variablen, die mit einem Buchstaben beginnen, der im angegebenen Buchstabenbereich liegt, werden als LONGINT dimensioniert.

Diese Anweisung kann **nur bis FreeBASIC v0.16** verwendet werden oder in entsprechend höheren Versionen, die mit der Kommandozeile -lang deprecated compiliert wurden. Wird seit FreeBASIC v0.17 mit -lang fb compiliert, dann müssen alle Datentypen explizit deklariert werden.

Der Ausdruck 'Buchstabenbereich' ist entweder der Form Startbuchstabe-Endbuchstabe, oder einfach nur ein Buchstabe.

Beispiel:

```
"hlstring">"deprecated"
DEFLONGINT l, d-f
DIM lNumber
DIM eNumber
```

Dadurch werden 'lNumber' und 'eNumber' zu einem LONGINT, da ihre Anfangsbuchstaben l sind bzw. im Bereich von d bis f liegen.

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC:
Seit FreeBASIC v0.17 können Standardtypen nur noch dann definiert werden, wenn die Kommandozeilenoption -lang deprecated angegeben wurde. Soll mit -lang fb compiliert werden, müssen alle Datentypen explizit deklariert werden.

Siehe auch:
DEFxxx, DIM, Datentypen und Deklarationen

Letzte Bearbeitung des Eintrags am 19.01.13 um 16:25:27
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

DEFSHORT

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » D » **DEFSHORT**
Syntax: DEFSHORT Buchstabenbereich [, Buchstabenbereich [, ...]]
Typ: Anweisung
Kategorie: Datentypen

DEFSHORT setzt den Standarddatentyp zu SHORT. Variablen, die mit einem Buchstaben beginnen, der im angegebenen Buchstabenbereich liegt, werden als SHORT dimensioniert.

Diese Anweisung kann **nur bis FreeBASIC v0.16** verwendet werden oder in entsprechend höheren Versionen, die mit der Kommandozeile -lang deprecated compiliert wurden. Wird seit FreeBASIC v0.17 mit -lang fb compiliert, dann müssen alle Datentypen explizit deklariert werden.

Der Ausdruck 'Buchstabenbereich' ist entweder der Form Startbuchstabe-Endbuchstabe, oder einfach nur ein Buchstabe.

Beispiel:

```
"hlstring">"deprecated"
DEFSHORT s, a-f
DIM sNumber
DIM eNumber
```

Dadurch werden 'sNumber' und 'eNumber' zu einem SHORT, da ihre Anfangsbuchstaben s sind bzw. im Bereich von a bis f liegen.

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC:
Seit FreeBASIC v0.17 können Standardtypen nur noch dann definiert werden, wenn die Kommandozeilenoption -lang deprecated angegeben wurde. Soll mit -lang fb compiliert werden, müssen alle Datentypen explizit deklariert werden.

Siehe auch:
DEFxxx, DIM, Datentypen und Deklarationen

Letzte Bearbeitung des Eintrags am 19.01.13 um 16:25:53
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

DEFSNG

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » D » **DEFSNG**
Syntax: DEFSNG Buchstabenbereich [, Buchstabenbereich [, ...]]
Typ: Anweisung
Kategorie: Datentypen

DEFSNG setzt den Standarddatentyp zu SINGLE. Variablen, die mit einem Buchstaben beginnen, der im angegebenen Buchstabenbereich liegt, werden als SINGLE dimensioniert.

Diese Anweisung kann **nur bis FreeBASIC v0.16** verwendet werden oder in entsprechend höheren Versionen, die mit der Kommandozeile -lang deprecated compiliert wurden. Wird seit FreeBASIC v0.17 mit -lang fb compiliert, dann müssen alle Datentypen explizit deklariert werden.

Der Ausdruck 'Buchstabenbereich' ist entweder der Form Startbuchstabe-Endbuchstabe, oder einfach nur ein Buchstabe.

Beispiel:

```
"hlstring">"deprecated"
DEFSNG s, d-f
DIM sNumber
DIM eNumber
```

Dadurch werden 'sNumber' und 'eNumber' zu einem SINGLE, da ihre Anfangsbuchstaben s sind bzw. im Bereich von d bis f liegen.

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC:
Seit FreeBASIC v0.17 können Standardtypen nur noch dann definiert werden, wenn die Kommandozeilenoption -lang deprecated angegeben wurde. Soll mit -lang fb compiliert werden, müssen alle Datentypen explizit deklariert werden.

Siehe auch:
DEFxxx, DIM, Datentypen und Deklarationen

Letzte Bearbeitung des Eintrags am 19.01.13 um 16:26:36
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

DEFSTR

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » D » **DEFSTR**
Syntax: DEFSTR Buchstabenbereich [, Buchstabenbereich [, ...]]
Typ: Anweisung
Kategorie: Datentypen

DEFSTR setzt den Standarddatentyp zu STRING. Variablen, die mit einem Buchstaben beginnen, der im angegebenen Buchstabenbereich liegt, werden als STRING dimensioniert.

Diese Anweisung kann **nur bis FreeBASIC v0.16** verwendet werden oder in entsprechend höheren Versionen, die mit der Kommandozeile -lang deprecated compiliert wurden. Wird seit FreeBASIC v0.17 mit -lang fb compiliert, dann müssen alle Datentypen explizit deklariert werden.

Der Ausdruck 'Buchstabenbereich' ist entweder der Form Startbuchstabe-Endbuchstabe, oder einfach nur ein Buchstabe.

Beispiel:

```
"hlstring">"deprecated"
DEFSTR s, d-f
DIM sMessage
DIM eMessage
```

Dadurch werden 'sMessage' und 'eMessage' zu einem STRING, da ihre Anfangsbuchstabe s sind bzw. im Bereich von d bis f liegen.

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC:
Seit FreeBASIC v0.17 können Standardtypen nur noch dann definiert werden, wenn die Kommandozeilenoption -lang deprecated angegeben wurde. Soll mit -lang fb compiliert werden, müssen alle Datentypen explizit deklariert werden.

Siehe auch:
DEFxxx, DIM, Datentypen und Deklarationen

Letzte Bearbeitung des Eintrags am 19.01.13 um 16:27:07
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

DEFUBYTE

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » D » **DEFUBYTE**
Syntax: DEFUBYTE Buchstabenbereich [, Buchstabenbereich [, ...]]
Typ: Anweisung
Kategorie: Datentypen

DEFUBYTE setzt den Standarddatentyp zu UBYTE. Variablen, die mit einem Buchstaben beginnen, der im angegebenen Buchstabenbereich liegt, werden als UBYTE dimensioniert.

Diese Anweisung kann **nur bis FreeBASIC v0.16** verwendet werden oder in entsprechend höheren Versionen, die mit der Kommandozeile -lang deprecated compiliert wurden. Wird seit FreeBASIC v0.17 mit -lang fb compiliert, dann müssen alle Datentypen explizit deklariert werden.

Der Ausdruck 'Buchstabenbereich' ist entweder der Form Startbuchstabe-Endbuchstabe, oder einfach nur ein Buchstabe.

Beispiel:

```
"hlstring">"deprecated"
DEFUBYTE u, a-f
DIM uNumber
DIM eNumber
```

Dadurch werden 'uNumber' und 'eNumber' zu einem UBYTE, da ihre Anfangsbuchstaben u sind bzw. im Bereich von a bis f liegen.

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC:
Seit FreeBASIC v0.17 können Standardtypen nur noch dann definiert werden, wenn die Kommandozeilenoption -lang deprecated angegeben wurde. Soll mit -lang fb compiliert werden, müssen alle Datentypen explizit deklariert werden.

Siehe auch:
DEFxxx, DIM, Datentypen und Deklarationen

Letzte Bearbeitung des Eintrags am 19.01.13 um 16:27:40
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

DEFUINT

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » D » **DEFUINT**
Syntax: DEFUINT Buchstabenbereich [, Buchstabenbereich [, ...]]
Typ: Anweisung
Kategorie: Datentypen

DEFUINT setzt den Standarddatentyp zu UINTEGER. Variablen, die mit einem Buchstaben beginnen, der im angegebenen Buchstabenbereich liegt, werden als UINTEGER dimensioniert.

Diese Anweisung kann **nur bis FreeBASIC v0.16** verwendet werden oder in entsprechend höheren Versionen, die mit der Kommandozeile -lang deprecated compiliert wurden. Wird seit FreeBASIC v0.17 mit -lang fb compiliert, dann müssen alle Datentypen explizit deklariert werden.

Der Ausdruck 'Buchstabenbereich' ist entweder der Form Startbuchstabe-Endbuchstabe, oder einfach nur ein Buchstabe.

Beispiel:

```
"hlstring">"deprecated"
DEFUINT u, d-f
DIM uNumber
DIM eNumber
```

Dadurch werden 'uNumber' und 'eNumber' zu einem UINTEGER, da ihre Anfangsbuchstaben u sind bzw. im Bereich von d bis f liegen.

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC:
Seit FreeBASIC v0.17 können Standardtypen nur noch dann definiert werden, wenn die Kommandozeilenoption -lang deprecated angegeben wurde. Soll mit -lang fb compiliert werden, müssen alle Datentypen explizit deklariert werden.

Siehe auch:
DEFxxx, DIM, Datentypen und Deklarationen

Letzte Bearbeitung des Eintrags am 19.01.13 um 16:28:16
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

DEFULONGINT

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » D » **DEFULONGINT**
Syntax: DEFULONGINT Buchstabenbereich [, Buchstabenbereich [, ...]]
Typ: Anweisung
Kategorie: Datentypen

DEFULONGINT setzt den Standarddatentyp zu ULONGINT. Variablen, die mit einem Buchstaben beginnen, der im angegebenen Buchstabenbereich liegt, werden als ULONGINT dimensioniert.

Diese Anweisung kann **nur bis FreeBASIC v0.16** verwendet werden oder in entsprechend höheren Versionen, die mit der Kommandozeile -lang deprecated compiliert wurden. Wird seit FreeBASIC v0.17 mit -lang fb compiliert, dann müssen alle Datentypen explizit deklariert werden.

Der Ausdruck 'Buchstabenbereich' ist entweder der Form Startbuchstabe-Endbuchstabe, oder einfach nur ein Buchstabe.

Beispiel:

```
"hlstring">"deprecated"
DEFULONGINT u, d-f
DIM uNumber
DIM eNumber
```

Dadurch werden 'uNumber' und 'eNumber' zu einem ULONGINT, da ihre Anfangsbuchstaben u sind bzw. im Bereich von d bis f liegen.

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC:
Seit FreeBASIC v0.17 können Standardtypen nur noch dann definiert werden, wenn die Kommandozeilenoption -lang deprecated angegeben wurde. Soll mit -lang fb compiliert werden, müssen alle Datentypen explizit deklariert werden.

Siehe auch:
DEFxxx, DIM, Datentypen und Deklarationen

Letzte Bearbeitung des Eintrags am 19.01.13 um 16:28:42
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

DEFUSHORT

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » D » **DEFUSHORT**
Syntax: DEFUSHORT Buchstabenbereich [, Buchstabenbereich [, ...]]
Typ: Anweisung
Kategorie: Datentypen

DEFUSHORT setzt den Standarddatentyp zu USHORT. Variablen, die mit einem Buchstaben beginnen, der im angegebenen Buchstabenbereich liegt, werden als USHORT dimensioniert.

Diese Anweisung kann **nur bis FreeBASIC v0.16** verwendet werden oder in entsprechend höheren Versionen, die mit der Kommandozeile -lang deprecated compiliert wurden. Wird seit FreeBASIC v0.17 mit -lang fb compiliert, dann müssen alle Datentypen explizit deklariert werden.

Der Ausdruck 'Buchstabenbereich' ist entweder der Form Startbuchstabe-Endbuchstabe, oder einfach nur ein Buchstabe.

Beispiel:

```
"hlstring">"deprecated"
DEFUSHORT u, a-f
DIM uNumber
DIM eNumber
```

Dadurch werden 'uNumber' und 'eNumber' zu einem USHORT, da ihre Anfangsbuchstaben u sind bzw. im Bereich von d bis f liegen.

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC:
Seit FreeBASIC v0.17 können Standardtypen nur noch dann definiert werden, wenn die Kommandozeilenoption -lang deprecated angegeben wurde. Soll mit -lang fb compiliert werden, müssen alle Datentypen explizit deklariert werden.

Siehe auch:
DEFxxx, DIM, Datentypen und Deklarationen

Letzte Bearbeitung des Eintrags am 19.01.13 um 16:29:10
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

DEFxxx

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » D » **DEFxxx**
Unter DEFxxx werden alle DEF-Anweisungen zusammengefasst:

DEFBYTE, DEFUBYTE, DEFSHORT, DEFUSHORT, DEFINT, DEFLNG, DEFUINT, DEFLONGINT, DEFULONGINT, DEFSNG, DEFDBL, DEFSTR

Wenn Sie in ihrem gesamten Programm keine DEFxxx-Anweisung verwenden, wird automatisch INTEGER als Standarddatentyp festgelegt.

Diese Anweisungen können **nur bis FreeBASIC v0.16** verwendet werden oder in entsprechend höheren Versionen, die mit der Kommandozeile -lang deprecated compiliert wurden. Wird seit FreeBASIC ab v0.17 mit -lang fb compiliert, dann müssen alle Datentypen explizit deklariert werden.

Siehe auch:
DIM, Datentypen und Deklarationen

Letzte Bearbeitung des Eintrags am 19.01.13 um 16:29:41
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

DELETE

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » D » **DELETE**
Syntax A: DELETE Zeiger
Syntax B: DELETE [ ] Zeiger
Typ: Operator
Kategorie: Operatoren

DELETE löscht Daten, die mit dem Operator NEW zugewiesen wurden.
'Zeiger' gibt den Pointer auf den freizugebenden Speicherplatz an.

DELETE wird benutzt, um den Speicher eines Objektes, das mit NEW erzeugt wurde, freizugeben und zu löschen. Wenn ein TYPE gelöscht wird, dann wird sein DESTRUCTOR aufgerufen. DELETE sollte nur mit den Adressen benutzt werden, die von NEW zurückgegeben wurden.

DELETE [ ] ist die Array-Version von DELETE und wird benutzt, um ein Array von Objekten, die vorher mit NEW [ ] erzeugt wurden, freizugeben und zu löschen. Destruktoren werden hier ebenfalls aufgerufen.
DELETE muss mit den Adressen benutzt werden, die NEW zurückgibt, und DELETE [ ] mit denjenigen von NEW [ ]. Die verschiedenen Versionen der Operatoren dürfen nicht vermischt werden und 'passen' auch nicht zueinander.

DELETE kann mithilfe von OPERATOR überladen werden.

Beispiel:
Siehe NEW für ein Beispiel zu DELETE.

Unterschiede zu QB: neu in FreeBASIC

Unterschiede unter den FB-Dialektformen: nur in der Dialektform -lang fb verfügbar

Siehe auch:
NEW, Speicher

Letzte Bearbeitung des Eintrags am 27.12.12 um 00:20:43
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

DESTRUCTOR (Klassen)

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » D » **DESTRUCTOR (Klassen)**
Syntax A:

```
TYPE TypeName
    ' Feld-Deklarationen
    DECLARE DESTRUCTOR [ () ]
END TYPE
```

Synax B:

```
DESTRUCTOR Typename [ () ]
   ' Anweisungen
END DESTRUCTOR
```

Typ: Prozedur
Kategorie: Klassen

Ein Klassen-Destruktor ist eine Prozedur, die aufgerufen wird, sobald eine UDT-Variable (user definded type - siehe TYPE (UDT)) zerstört wird.

- 'TypeName' ist der Name eines UDTs, wie er unter TYPE (UDT) erklärt wird.
- 'Feld-Deklarationen' sind die Deklarationen, die den UDT oder die Klasse ausmachen. Siehe dazu TYPE (UDT), PROPERTY, OPERATOR.
- Die Parameterliste muss bei Destruktoren leer sein.
- 'Anweisungen' ist ein Programmcode, der den Regeln einer SUB folgt.

Die Destruktor-Prozedur wird aufgerufen, sobald eine UDT-Variable zerstört wird. Dies ist dann der Fall, wenn eine mit NEW erstellte Variable mit DELETE zerstört wird, wenn der SCOPE-Block, innerhalb dessen die Variable erstellt wurde, an sein Ende kommt oder wenn das Programm beendet wird.

Achtung: Wird das Programm mit END beendet, findet kein automatischer Aufruf der Klassen-Destruktoren statt! Die Destruktoren müssen gegebenenfalls zuvor explizit aufgerufen werden. Modul-Destruktoren werden dagegen bei der Verwendung von END vor Programmende aufgerufen.

Innerhalb der Destruktor-Prozedur kann über das Schlüsselwort THIS auf die Records des UDTs zugegriffen werden.

Enthält der zu zerstörende UDT selbst Felder, die Klassen mit eigenen Destruktoren darstellen, so wird der Destruktor des UDTs vor denen der Records aufgerufen.

Innerhalb eines Types kann nur ein einziger Destruktor erstellt werden.

Klassen-Destruktoren müssen PUBLIC sein.

Das Schlüsselwort DESTRUCTOR leitet einen eigenen SCOPE-Block ein. Siehe dazu auch SCOPE.

Beispiel 1: Praxisnaher Einsatz zur automatischen Speicherfreigabe mit DEALLOCATE:

```
TYPE T
  intPtr AS INTEGER PTR
```

```
  DECLARE CONSTRUCTOR ( value AS INTEGER )
  DECLARE DESTRUCTOR  ( )

  DECLARE OPERATOR CAST () AS STRING
END TYPE

CONSTRUCTOR T ( value AS INTEGER )
   THIS.intPtr = ALLOCATE( SIZEOF(INTEGER) )
  *THIS.intPtr = value
END CONSTRUCTOR

DESTRUCTOR T ()
  DEALLOCATE THIS.intPtr
END DESTRUCTOR

OPERATOR T.Cast () AS STRING
  RETURN STR(*THIS.intPtr)
END OPERATOR

DIM x AS T = 10

PRINT x
SLEEP
```

Ausgabe:

```
10
```

Beispiel 2: Zeitliche Abfolge des Erstellens und Zerstörens unter verschiedenen Bedingungen: Modulebene, Scope-Ebene, Prozedurebene.

```
TYPE T
  value AS ZSTRING * 32
  DECLARE CONSTRUCTOR ( init_value AS STRING )
  DECLARE DESTRUCTOR ()
END TYPE

CONSTRUCTOR T ( init_value AS STRING )
  value = init_value
  PRINT "Erstelle:  "; value
END CONSTRUCTOR

DESTRUCTOR T ()
  PRINT "Zerstoere: "; value
END DESTRUCTOR

SUB MySub
  DIM x AS T = ("MySub.x")
END SUB

DIM x AS T = ("main.x")

SCOPE
  DIM x AS T = ("main.scope.x")
```

```
END SCOPE

MySub

SUB Quit DESTRUCTOR
   SLEEP
END SUB
```

Ausgabe:

```
Erstelle:  main.x
Erstelle:  main.scope.x
Zerstoere: main.scope.x
Erstelle:  MySub.x
Zerstoere: MySub.x
Zerstoere: main.x
```

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC: DESTRUCTOR für Klassen existiert seit FreeBASIC v0.17.

Unterschiede unter den FB-Dialektformen: DESTRUCTOR ist nur in -lang fb zulässig.

Siehe auch:
DESTRUCTOR (Module), CONSTRUCTOR (Klassen), TYPE (UDT), NAMESPACE, Objektorientierung

Letzte Bearbeitung des Eintrags am 27.12.12 um 00:21:07
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

DESTRUCTOR (Module)

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » D » **DESTRUCTOR (Module)**
Syntax: [PUBLIC | PRIVATE] SUB name [ALIAS "externer_name"] [()] DESTRUCTOR [priorität] [STATIC]
Typ: Klausel
Kategorie: Programmablauf

Die Klausel DESTRUCTOR bewirkt, dass eine SUB aufgerufen wird, bevor das Programm beendet wird. Dies geschieht sowohl dann, wenn eine END-, SYSTEM-, oder STOP-Anweisung aufgerufen wird, als auch, nachdem die letzte Anweisung des Hauptmoduls ausgeführt wurde. Wurde per Kommandozeile die Fehlerbehandlung aktiviert, so werden die Destruktoren auch aufgerufen, wenn ein nicht-planmäßiges Programmende auftritt; siehe dazu Der Compiler.

Siehe auch SUB für Details zu Prozeduren.

- 'PRIVATE' und 'PUBLIC' legen fest, ob die SUB PRIVATE oder PUBLIC sein soll; siehe dazu die entsprechenden Einträge der Referenz.
- 'name' ist der Bezeichner der SUB. Er folgt den üblichen Regeln.
- Die Klausel 'DESTRUCTOR' markiert die SUB als Destruktor.
- Ein Destruktor darf keine Parameterliste besitzen, da kein Weg existiert, beim Programmende Parameter für die Übergabe festzulegen.
- 'priorität' ist ein INTEGER-Wert zwischen 101 und 65535. Mit diesem Wert kann bei der Existenz mehrerer Destruktoren im selben Modul die Reihenfolge festgelegt werden, in der die Destruktoren abgearbeitet werden sollen. 101 ist dabei die niedrigste Priorität; dieser Destruktor wird zuletzt ausgeführt. Die Zahlen an sich haben keine spezielle Bedeutung; lediglich die Verhältnisse zueinander (größer als, kleiner als) wirken sich tatsächlich im Programmverlauf aus. Alle Destruktoren, die ohne eine Priorität festgelegt wurden, werden vor denen ausgeführt, die mit Priorität festgelegt wurden.

Werden in einem Modul mehrere Destruktoren ohne Priorität verwendet, so werden diese im Code von oben nach unten abgearbeitet.

Eine solche SUB muss nur dann durch eine DECLARE-Anweisung deklariert werden, wenn sie im Programmverlauf nochmals manuell aufgerufen werden soll. Wird sie mit einer normalen DECLARE-Zeile deklariert, so kann sie - wie eine ganz normale SUB - im Programm jederzeit aufgerufen werden und folgt den Regeln einer normalen Prozedur.

Wird versucht, eine SUB mit der DESTRUCTOR-Klausel zu belegen, die einen oder mehrere Parameter besitzt, so wird ein Compiler-Fehler erzeugt.

Innerhalb eines Modules können mehrere Destruktoren verwendet werden. Innerhalb mehrerer Module können ebenfalls zusätzliche Destruktoren eingefügt werden, unter der Voraussetzung, dass keine zwei PUBLIC-Destruktoren mit demselben Bezeichner existieren.

Wenn mehrere Module compiliert werden, die Destruktoren besitzen, kann die Reihenfolge der Ausführung der Destruktoren nicht garantiert werden, solange keine Priorität angegeben wurde. Darauf muss besonders geachtet werden, wenn die Destruktoren eines Moduls eine Prozedur eines anderen Moduls aufrufen, das ebenfalls einen Destruktor besitzt. Es wird empfohlen, einen einzigen Destruktor zu definieren, der die Prozeduren in den einzelnen Modulen manuell aufruft.

Beispiel:
Dieses Beispielprogramm enthält ein Set aus vier Konstruktoren und Destruktoren. Es wird gezeigt, in

welcher Reihenfolge diese abgearbeitet werden.

```
Sub Constructor1 Constructor 101
Print "Constructor1 aufgerufen - hoechste Prioritaet"
End Sub
Sub Constructor2 Constructor 201
Print "Constructor2 aufgerufen - niedrigere Prioritaet"
End Sub
Sub Constructor3 Constructor
Print "Constructor3 aufgerufen - ohne Prioritaet"
End Sub
Sub Constructor4 Constructor
Print "Constructor4 aufgerufen - ohne Prioritaet"
End Sub

'-------------------------------------------------------

Sub Destructor1 Destructor 101
Print "Destructor1 aufgerufen - niedrigeste Prioritaet"
Sleep 'Auf Tastendruck warten
End Sub
Sub Destructor2 Destructor 201
Print "Destructor2 aufgerufen - hoehere Prioritaet"
End Sub
Sub Destructor3 Destructor
Print "Destructor3 aufgerufen - ohne Prioritaet"
End Sub
Sub Destructor4 Destructor
Print "Destructor4 aufgerufen - ohne Prioritaet"
End Sub

'-------------------------------------------------------

Print "Modul-Level Code"
```

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC:

- Die Möglichkeit, eine Priorität zuzuweisen, existiert seit FreeBASIC v0.17.
- Destruktor-SUBs existieren seit FreeBASIC v0.14.

Siehe auch:
CONSTRUCTOR (Module), SUB, DESTRUCTOR (Klassen), Prozeduren

Letzte Bearbeitung des Eintrags am 19.01.13 um 16:37:25
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

DIM

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » D » **DIM**
Syntax A:

```
DIM [SHARED] Variable1 AS Typ [= Ausdruck] [, _
             Variable2 AS Typ [= einfacherAusdruck] [, _
             ...] ]
```

Syntax B:

```
DIM [SHARED] AS Typ Variable1 [= Ausdruck] [, _
                    Variable2 [= Ausdruck] [, _
                    ...] ]
```

Syntax C:

```
DIM [SHARED] Array( [ [Startindex TO] Endindex  [, _
                      [Startindex TO] Endindex] [, ...] ] ] ) AS Typ [ _
                    = {Ausdruck [, Ausdruck [, _ ... ] ] } ]
```

Typ: Anweisung
Kategorie: Deklaration

Anmerkung zur Syntax: Unterstriche (_) am Zeilenende werden von FreeBASIC so interpretiert, als wäre die Zeile nicht unterbrochen; dies dient nur der besseren Übersichtlichkeit und hat letzt–end–lich keine Auswirkungen auf die Programmausführung.

DIM dimensioniert Variablen und Arrays einer angegebenen Größe und eines angegebenen Typs. Ein Array ist eine Gruppe aus mehreren Variablen desselben Typs, auf dessen einzelne Elemente über einen Index zugegriffen wird.

- 'Variable' und 'Array' sind die Namen von neuen, noch nicht verwendeten Variablen oder Arrays.
- 'SHARED' bewirkt, dass die Variable auch Unterprogrammen zugänglich ist.
- 'AS Typ' gibt an, von welchem Typ die Variable sein soll. Auch FUNCTION oder SUB kann hier als Typ angegeben werden; siehe dazu DYLIBLOAD.
- 'Ausdruck' ist ein Ausdruck, der aus Konstanten, Variablen, Operatoren, Funktionen und Symbolen bestehen darf. Der Wert von 'Ausdruck' wird der dimensionierten Variable zugewiesen. Um die Elemente eines Arrays zu befüllen, werden die einzelnen Werte in {geschweifte Klammern} gesetzt und durch Kommata getrennt. Bei mehrdimensionalen Arrays müssen innerhalb der Klammern noch einmal geschweifte Klammern gesetzt werden (siehe Beispiel 2 unten).
- 'Startindex' und 'Endindex' können alle Werte annehmen, wobei der Endindex größer oder gleich dem Startindex sein muss. Auch negative Indizes sind erlaubt, wenn auch nicht üblich. Dies könnte auch zu Problemen mit UBOUND führen, da der Befehl '-1' zurückgibt, falls das Array nicht initialisiert wurde.

Jeder Variablen muss mittels 'AS Typ' explizit ein Datentyp zugewiesen werden. Standarddatentypen (vgl. DEFxxx) stehen nur in Dialektformen wie -lang deprecated oder -lang qb zur Verfügung.

Für die Verwendung eines Initiators 'Ausdruck' kann neben dem Gleichheitszeichen = auch die Schreibweise => verwendet werden. Wird kein Initiator verwendet, so wird der Standardkonstruktor des Datentyps verwendet: Zahlen und Pointer werden auf 0 gesetzt, STRINGs auf den Leerstring "". Bei UDTs wird in diesem Fall der Konstruktor ohne Parameterliste aufgerufen. Wenn kein Konstruktor existiert, wird für jeden Record der Standardkonstruktor aufgerufen.

Es ist auch möglich, als 'Ausdruck' ANY anzugeben, um festzulegen, dass die Variable nicht initialisiert werden soll; an der Speicherstelle verbleibt der 'Datenmüll', der im Speicher zurückgeblieben ist.

Einfache Variablen werden folgendermaßen dimensioniert:

```
DIM i AS INTEGER, s AS SINGLE
```

Sollen mehrere Variablen desselben Typs dimensioniert werden, lässt sich das mit Syntax B einfacher bewerkstelligen:

```
DIM AS INTEGER a, b, c, d, e
```

Arrays
Sollen sehr viele Variablen desselben Typs gleichzeitig dimensioniert werden, dann bietet sich die Verwendung eines Arrays an. Mit folgendem Code wird der Speicherbereich für 1000 INTEGER-Werte zur Verfügung gestellt:

```
DIM hoehe(1 TO 1000) AS INTEGER
' oder auch
DIM AS INTEGER breite(1 TO 1000)
```

Dadurch werden 1000 Speicherstellen für INTEGER-Zahlen reserviert, wobei der Index des ersten Elements 1 ist und der des letzten Elements 1000.
Wird bei der DIM-Anweisung der Startindex ausgelassen, so wird automatisch angenommen, dass der Startindex 0 ist. Zur Reservierung von 1000 INTEGER-Werten kann daher auch folgendermaßen vorgegangen werden:

```
DIM hoehe(999) AS INTEGER
```

Hier reichen die Indizes von 0 bis 999.

Um mehrdimensionale Arrays anzulegen, werden die Grenzen der einzelnen Dimensionen nacheinander angegeben. Folgender Code legt ein zweidimensionales Array mit 100 Zeilen und 51 Spalten an:

```
DIM AS INTEGER zweidimensional (1 TO 100, 50)
```

Um nun auf die Einträge im Feld zuzugreifen, verwenden Sie die Syntax:

```
FeldName(Index &"hlkw0">DIM feld (3) AS BYTE
DIM tabelle (2, 4) AS INTEGER

feld(0) = 10
feld(1) = -3
feld(2) = feld(0) + feld(1)

tabelle(0, 0) = 10

PRINT feld(2), tabelle(0, 0)
SLEEP
```

Achtung: Die Verwendung eines Index, der außerhalb der mit DIM angegebenen Grenzen liegt, kann möglicherweise andere Programmdaten beeinflussen oder sogar zu einem Programmabsturz führen. Dieser Fehler tritt häufig nicht sofort auf, was eine Fehleranalyse erschwert. Der FreeBASIC-Compiler stellt den

Optionsschalter -exx zur Verfügung, um während des laufenden Programms die Gültigkeit der Array-Grenzen zu überprüfen.

Initialisierung
Soll bei der Dimensionierung der Variable sofort ein Wert zugewiesen werden, kann dies über die sogenannten Variablen-Initiatoren geschehen:

Beispiel 2: Variablen-Initiatoren

```
DIM a AS INTEGER = 5, b AS USHORT = 123
DIM AS STRING c = "hello world", d = "FreeBASIC" & CHR(13) & "free
Compiler"

'Alternativ:   Array(1 TO 3, 0 TO 3) ...
DIM AS INTEGER Array(1 TO 3, 3) = { {1, 2, 3, 4}, _
                                    {5, 6, 7, 8}, _
                                    {9,10,11,12} }
PRINT Array (3, 0)     'gibt 9 aus
SLEEP
```

Wird beim Anlegen eines Arrays die Werte sofort zugewiesen, dann kann der Endindex auch durch drei Punkte ... (Auslassung bzw. Ellipsis) ersetzt werden. Die obere Grenze des Arrays wird dann anhand der angegebenen Werte ermittelt.

Beispiel 3: Array-Dimensionierung mit Ellipsis

```
DIM AS INTEGER wert(3 TO ...) = {1, 2, 3, 4, 5, 6, 7, 8, 9}
PRINT LBOUND(wert), UBOUND(wert)  ' untere und obere Grenze anzeigen
SLEEP
```

Dynamische Arrays
Es können auch Felder unbekannter Größe erzeugt werden. Solche Felder können z.B. an Prozeduren übergeben werden, wo sie redimensioniert und befüllt werden. Ein dynamisches Array kann mit DIM ohne Angabe der Grenzen oder über REDIM angelegt werden.

Beispiel 4: Dynamisches Array

```
DECLARE SUB FileList(Path AS STRING, Array() AS STRING)
DIM AS STRING Files()  ' dynamisch

FileList ENVIRON("TMP"), Files()

SUB FileList(Path AS STRING, Array() AS STRING)
  DIM AS STRING filename
  DIM AS INTEGER i
  "hlkw0">__FB_PCOS__
    filename = DIR(Path & "\*.*")  ' Windows-Pfadangabe
  "hlzeichen">= DIR(Path & "/*")    ' Linux-Pfadangabe
  "hlkw0">IF LEN(filename) THEN
    REDIM Array(0) AS STRING  ' neu dimensionieren
    Array(0) = filename
    i = 1
  ELSE
```

```
    EXIT SUB
  END IF
  DO
    filename = DIR()
    IF LEN(filename) THEN
      REDIM PRESERVE Array(i) AS STRING
      Array(i) = filename
      i += 1
    ELSE
      EXIT DO
    END IF
  LOOP
END SUB

PRINT UBOUND(Files)
FOR i AS INTEGER = 0 TO UBOUND(Files)
   PRINT Files(i)
NEXT

SLEEP
```

Unterschiede zu QB:

- In FreeBASIC können als Indexzahl keine Variablen verwendet werden. Dies wird sich in Zukunft vielleicht ändern.
- FreeBASIC unterstützt Variablen-Initiatoren.
- Die alternative Syntax DIM AS Typ Variable (Syntax 2) ist neu in FreeBASIC.
- Bei der Speicherung mehrdimensionaler Arrays folgen in FreeBASIC die Werte aufeinander, deren erster Index gleich ist. In QB folgen die Werte aufeinander, deren letzter Index gleich ist.
- Die Auslassung (...; Ellipsis) des Endindex eines Arrays ist neu in FreeBASIC.

Unterschiede zu früheren Versionen von FreeBASIC:

- Seit FreeBASIC v0.21 kann bei Arrays mit Initiatoren der Endindex durch eine Auslassung ersetzt werden.
- Seit FreeBASIC v0.17 muss der Typ explizit deklariert werden, außer das Programm wird mit der Option *-lang deprecated* compiliert.
- Seit FreeBASIC v0.16 dürfen Felder unbekannter Größe an jedem Programmpunkt, auch in Prozeduren, erstellt werden.
- Variablen-Initiatoren existieren seit FreeBASIC v0.13.

Unterschiede unter den FB-Dialektformen:

- In der Dialektform -lang qb und -lang fblite sind Variablen, die in einer Prozedur dimensioniert wurden, in der ganzen Prozedur sichtbar. Variablen, die mit 'SHARED' dimensioniert wurden, sind im ganzen Modul sichtbar.
- In der Dialektform -lang fb und -lang deprecated sind Variablen, die in einem Block dimensioniert wurden (FOR ... NEXT, WHILE ... WEND, DO ... LOOP, SCOPE ... END SCOPE) nur in diesem Block sichtbar.
- In der Dialektform -lang fb sind OPTION-Anweisungen (z. B. OPTION BASE, OPTION DYNAMIC) nicht erlaubt.

Siehe auch:
REDIM, SHARED, SCOPE, STATIC (Anweisung), COMMON, DYNAMIC (Meta), STATIC (Meta),

OPTION, DYNAMIC (Schlüsselwort), STATIC (Schlüsselwort), ERASE, CLEAR, LBOUND, UBOUND, Datentypen, Pointer, Gültigkeitsbereich von Variablen

Letzte Bearbeitung des Eintrags am 11.01.14 um 23:01:28
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

DIR

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » D » **DIR**
Syntax: DIR[$] ([Dateiangabe] [, Attributnummer [, Rückgabeattribut]])
Typ: Funktion
Kategorie: System

DIR gibt die Dateien im aktuellen Arbeitsverzeichnis oder im angegebenen Pfad zurück.

- 'Dateiangabe' ist ein String, der das Muster (Pattern) zu den aufzulistenden Dateien oder Verzeichnissen enthält.
- 'Attributnummer' ist eine Zahl, die angibt, welche Attribute die zurückzugebenden Dateien haben sollen. Die Atrributnummern werden weiter unten erläutert.
- 'Rückgabeattribut' ist ein INTEGER PTR. Wird hier ein Wert angegeben, schreibt FreeBASIC an die Speicherstelle, auf die 'Rückgabeattribut' verweist, welche Attribute auf die gefundene Datei tatsächlich zutreffen.
- Der Rückgabewert ist der Name der ersten Datei (bzw. Verzeichnis), die passend zu 'Dateiangabe' gefunden wurde. Wird 'Dateiangabe' ausgelassen oder ein Leerstring übergeben, so wird die nächste Datei (bzw. Verzeichnis) zurückgegeben, die auf das zuletzt angegebene Pattern passt.

'Dateiangabe' darf auch Pfadangaben und Wildcards enthalten. Die Dateinamen bzw. Verzeichnisse werden mit dem angegebenen Muster verglichen. Durch Wildcards, also Platzhalter, kann es auf mehrere Dateien und/oder Verzeichnisse passen. Folgende Wildcards sind möglich:

- * - eine beliebige Zeichenfolge
- ? - ein beliebiges Zeichen, jedoch nur eines.

Beispielsweise passt die Angabe "a*.b??" auf alle Daten oder Verzeichnisse, die mit einem a beginnen und deren Endung mit einem b beginnt und 3 Zeichen lang ist. Mögliche Ergebnisse wären also z. B. allocate.bas und a.bin

Das Dollarzeichen ($) als Suffix ist optional.

Dateiattribute
'Attributnummer' ist eine Zahl, die angibt, welche Attribute die zurückzugebenden Dateien haben sollen. Sie ist eine Kombination aus folgenden Werten:

Wert	Bedeutung	Konstante *in der dir.bi*
&h00	*Normal*	- keines -
&h01	*Schreibgeschützt* Windows: Das Attribut 'Schreibgeschützt' ist gesetzt. Linux: Die Schreibberechtigung des aktuellen Benutzers und seiner Gruppenzugehörigkeit sowie die globale Schreibberechtigung wird geprüft; root-Berechtigungen des Benutzers werden ignoriert.	fbReadOnly
&h02	*Versteckt* Windows: Das Attribut 'Versteckt' ist gesetzt. Linux: Der Datei-/Verzeichnisname beginnt mit einem Punkt.	fbHidden
&h04	*System* Windows: Das Attribut 'System' ist gesetzt. Linux: Es handelt sich um ein zeichenorientiertes Gerät (character device), ein blockorientiertes Gerät (block device), ein Named Pipe (FIFO) oder ein Unix-Socket.	fbSystem

&h10	*Verzeichnis*	fbDirectory
	Archivierbar	
&h20	Windows: Das Attribut 'Archiv' ist gesetzt. Die unter Linux üblichen Dateisysteme unterstützen dies nicht.	fbArchive
&h21	*Dateien, die archivierbar oder schreibgeschützt sind*	fbNormal

Um die angegebenen Konstanten zu nutzen, müssen Sie mittels #INCLUDE die Datei dir.bi einbinden.
Die oben genannten Eigenschaften können durch eine OR-Verknüpfung kombiniert werden. Der Befehl gibt dann alle Dateien zurück, die in das Muster passen und mindestens ein Attributkriterium erfüllen. Will man also alle schreibgeschützten oder versteckten Dateien auflisten, muss die Attributnummer fbReadOnly OR fbHidden angegeben werden. Als Ergebnis werden hier schreibgeschützte Dateien, versteckte Dateien und schreibgeschützte, versteckte Dateien aufgelistet. Wird 'AttributNummer' ausgelassen, so nimmt FreeBASIC automatisch fbNormal an; dies entspricht einer Liste aller normalen Dateien, inklusive schreibgeschützter und archivierter Dateien.

Wird 'Rückgabeattribut' angegeben, schreibt FreeBASIC an die Speicherstelle, auf die 'Rückgabeattribut' verweist, welche Attribute auf die gefundene Datei oder Ordner tatsächlich zutreffen. Dadurch kann die Ausgabe selektiver gestaltet werden. Wollen Sie beispielsweise eine Liste aller Dateien in einem Verzeichnis erstellen, die schreibgeschützt sind und entweder archivierbar sind oder als versteckt markiert wurden, so können Sie zunächst mit 'Attributnummer' fbReadOnly OR fbArchive OR fbHidden eine Ausgabe von Dateien erreichen, bei denen mindestens eines der Kriterien erfüllt ist. Mit 'Rückgabeattribut' kann dann geprüft werden, ob die Datei tatsächlich mehrere Kriterien erfüllt; die entsprechende Zeile könnte so aussehen:

```
IF (rAttr AND fbReadOnly) ANDALSO _
   ((rAttr AND fbHidden) ORELSE (rAttr AND fbArchive)) THEN ...
```

Beispiel 1: Zwei Dateien anzeigen, die mit "a" beginnen

```
PRINT DIR("a*", 0)
PRINT DIR()
SLEEP
```

Beispiel 2: Verwendung der Attributnummer

```
"hlstring">"dir.bi"

DECLARE SUB listFiles (filespec AS STRING, attrib AS INTEGER)

SUB listFiles (filespec AS STRING, attrib AS INTEGER)
  ' alle Dateien mit den angegebenen Attributen auflisten
   DIM filename AS STRING

   filename = DIR(filespec, attrib)
   DO
      PRINT SPACE(4); filename
      filename = DIR("", attrib)
   LOOP WHILE LEN(filename)
END SUB

PRINT "Unterordner:"
listFiles "*", fbDirectory

PRINT "archive files:"
listFiles "*", fbArchive
```

```
PRINT "Versteckte und/oder Systemdateien, die mit 'a' beginnen:"
listFiles "a*", fbHidden OR fbSystem
SLEEP
```

Beispiel 3: Alle Dateien und Ordner, die nicht versteckt sind, in einem Array speichern und ausgeben:

```
"hlstring">"dir.bi"

sub GetFiles( Array() as string )
  dim as string  s
  dim as integer attr, i
  redim Array(0)

  s = dir("*",, @attr)
  do until s = ""
    if (attr and fbHidden) = 0 then
      i += 1
      redim preserve Array(i)
      Array(i) = s
    end if
    ' Da hier geprüft wird, ob das gewünschte Dateiattibut zu jeder
    ' Datei gesetzt ist, muss @attr bei jedem Aufruf übergeben werden.
    s = dir("",, @attr)
  loop
end sub

dim dateien() as string
GetFiles dateien()

for i as integer = 1 to ubound(dateien)
  print i, dateien(i)
next
sleep
```

Unterschiede zu QB:

- DIR existiert nicht in QB, sondern nur in Visual Basic.
- Das Rückgabeattribut existiert nur in FreeBASIC.

Plattformbedingte Unterschiede:

- Die Bedeutung der Dateiattribute ist unter Linux und Windows/DOS zum Teil verschieden (siehe Tabelle).
- Unter Linux muss der Dateiname 'case sensitive' erfolgen. Windows und DOS sind 'case insensitive'.
- Das Trennzeichen für den Dateipfad ist unter Linux der vorwärtsgerichtete Slash /. Windows verwendet den Backslash \, erlaubt aber auch der Slash. DOS verwendet den Backslash.
- Unter DOS gibt die Attributsmaske &h37 alle Dateien und Ordner, einschließlich "." und "..", zurück, jedoch keine Datenträger. Der Wert &h08 gibt den Datenträger zurück, auch wenn das aktuelle Arbeitsverzeichnis nicht das Hauptverzeichnis ist.

Unterschiede zu früheren Versionen von FreeBASIC:

- Der optionale Parameter 'Rückgabeattribut' existiert seit v0.20.

- Seit FreeBASIC v0.13 ist der erste Parameter 'Dateiangabe' optional.

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht DIR nicht zur Verfügung und kann nur über **__DIR** aufgerufen werden.

Siehe auch:
FILEEXISTS, OPEN, CURDIR, MKDIR, RMDIR, Betriebssystem-Anweisungen

Letzte Bearbeitung des Eintrags am 27.12.12 um 00:22:18
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

DO ... LOOP

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » D » **DO ... LOOP**
Syntax A:

```
DO [{UNTIL | WHILE} Bedingung]
   ' Programmcode
LOOP
```

Syntax B:

```
DO
   ' Programmcode
LOOP [{UNTIL | WHILE} Bedingung]
```

Typ: Anweisung
Kategorie: Programmablauf

Die DO-Schleife wiederholt einen Anweisungsblock, während bzw. bis eine Bedingung erfüllt ist. Wird keine Bedingung angegeben, so wird eine Endlosschleife erzeugt. Der Programmierer muss dann über Anweisungen wie EXIT selbst dafür sorgen, dass die Schleife wieder verlassen wird.

Bedingung ist ein numerischer Ausdruck, der entweder wahr/erfüllt (ungleich null) oder falsch/nicht erfüllt (gleich null) sein kann. Wird das Schlüsselwort UNTIL eingesetzt, so führt FreeBASIC die Schleife aus, *bis* (until) die Bedingung erfüllt ist, d.h. bis der Ausdruck einen Wert ungleich null annimmt. Wird das Schlüsselwort WHILE eingesetzt, so führt FreeBASIC die Schleife aus, *solange* (while) die Bedingung erfüllt ist, d.h. bis der Ausdruck gleich null ist.

Als Block-Anweisung initialisiert DO...LOOP einen SCOPE-Block, der mit der LOOP-Zeile endet. Variablen, die innerhalb einer solchen Schleife deklariert werden, existieren außerhalb nicht mehr.
Achtung: Eine innerhalb der DO-Schleife deklarierte Variable kann auch nicht in 'Bedingung' zur Überprüfung des Schleifenendes verwendet werden werden, da sie an dieser Stelle bereits nicht mehr existiert!

Taucht innerhalb des Schleifen-Codes die Anweisung CONTINUE DO auf, so ignoriert FreeBASIC den Code bis zum nächsten LOOP, prüft dort, ob die Bedingung (sofern angegeben) erfüllt wurde, und springt gegebenenfalls zurück zur DO-Zeile.

In Syntax A wird die Bedingung bereits geprüft, *bevor* die Schleife 'betreten' wird. Bewirkt der Wert der Bedingung dann ein Ende der Schleife, so wird der Schleifencode übersprungen. Bei Syntax B wird der Wert der Bedingung erst geprüft, wenn der Schleifencode schon einmal durchlaufen wurde. Die Schleife wird also auf jeden Fall mindestens einmal durchlaufen.

DO...LOOP-Blöcke können bis zu beliebiger Ebene ineinander und mit anderen Schleifen verschachtelt werden.

Beispiel 1: Endlosschleife mit CONTINUE und EXIT

```
Dim As Integer n = 1
Dim As Integer gerade = 0

Do
  n += 1
  If n > 10 Then Exit Do
```

```
  If n Mod 2 Then Continue Do
  gerade += 1
Loop
Print "Anzahl gerader Zahlen: "; gerade
Sleep
```

Beispiel 2: Datei zeilenweise auslesen

```
DIM f AS INTEGER = FREEFILE, zeile AS STRING
IF OPEN("datei.ext" FOR INPUT AS "hlzeichen">) = 0 THEN
  ' Datei existiert und wird ausgelesen
  DO UNTIL EOF(f)  ' bis Dateiende erreicht wurde
    LINE INPUT "hlzeichen">, zeile
    PRINT zeile
  LOOP
  CLOSE "hlkw0">ELSE
  PRINT "Datei konnte nicht geöffnet werden!"
END IF
SLEEP
```

Unterschiede zu QB:
Innerhalb einer Schleife dürfen in QB keine Variablen definiert werden.

Unterschiede zu früheren Versionen von FreeBASIC:
Seit FreeBASIC v0.16 wirkt eine DO-LOOP-Schleife wie ein SCOPE-Block.

Unterschiede unter den FB-Dialektformen:

- In der Dialektform -lang qb und -lang fblite sind Variablen, die in einer DO-Schleife dimensioniert werden, in der ganzen Prozedur sichtbar.
- In der Dialektform -lang fb und -lang deprecated sind Variablen, die in einer DO-Schleife dimensioniert wurden, nur innerhalb dieser Schleife gültig.

Siehe auch:
WHILE ... WEND, FOR ... NEXT, EXIT, CONTINUE, SCOPE, Bedingungsstrukturen, Schleifen

Letzte Bearbeitung des Eintrags am 27.12.12 um 00:22:46
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

DOUBLE

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » D » **DOUBLE**
Typ: Datentyp

DOUBLE bedeutet doppelte Genauigkeit. Variablen vom Typ DOUBLE sind vorzeichenbehaftete 64-bit-Gleitkommazahlen.
Damit lassen sich Zahlen mit bis zu 15 Stellen darstellen, der Wertebereich für DOUBLE-Zahlen liegt bei positiven Zahlen zwischen 4.940656458412465e-324 und 1.797693134862316e+308, bei negativen zwischen -4.940656458412465e-324 und -1.797693134862316e+308.

DOUBLE werden benutzt, um sehr genaue Dezimalzahlen zu speichern. Sie verhalten sich ähnlich zu SINGLE-Variablen, sind aber genauer. Dafür benötigt das Rechnen mit DOUBLE-Zahlen auch mehr Zeit.

Wie genau sie auch sein mögen, sie sind immer noch bestimmten Grenzen unterworfen, so dass große Genauigkeitsverluste auftreten können, wenn sie nicht richtig verwendet werden.

Beispiel:

```
DIM s AS SINGLE, d AS DOUBLE
s = ATN(1)*4  ' Kreiskonstante PI
d = ATN(1)*4

PRINT "PI als Single:"; s
PRINT "PI als Double:"; d
SLEEP
```

Ausgabe:

```
PI als Single: 3.141593
PI als Double: 3.141592653589793
```

Siehe auch:
DIM, CAST, CDBL, Datentypen, Datentypen und Deklarationen

Letzte Bearbeitung des Eintrags am 28.12.12 um 00:44:55
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

DRAW (Grafik)

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » D » **DRAW (Grafik)**
Syntax: DRAW [Puffer,] Stringbefehle
Typ: Anweisung
Kategorie: Grafik

DRAW kann für mehrere verschiedene Zeichenbefehle verwendet werden. Um kleinere Figuren zu zeichnen, ist es durchaus geeignet.

- 'Puffer' ist ein Speicherbereich wie ein mit IMAGECREATE erstellter Puffer oder ein Array. Beide können mit PUT angezeigt werden. Wird 'Puffer' ausgelassen, zeichnet FreeBASIC direkt auf den Bildschirm.
- 'Stringbefehle' ist eine Reihe aus Zeichen. Jedes Zeichen stellt einen eigenen Befehl dar. Die möglichen Befehle werden unten aufgelistet.

Grafikcursorbefehle

- **B** - Vor einem eigentlichen Befehl angehängt bewirkt es, dass der Grafikcursor zwar bewegt, aber nichts gezeichnet wird.
- **N** - Vor einem eigentlichen Befehl angehängt bewirkt es, dass zwar gezeichnet wird, der Grafikcursor aber nicht bewegt wird.
- **M**x,y - Zeichnet eine Strecke vom ursprünglichen Grafikcursor zu den Koordinaten (xly). Wenn dem x ein + oder - vorausgeht, sind die Koordinaten relativ zum ursprünglichen Grafikcursor.
- **U**[n] - Zeichnet eine Strecke n Pixel nach oben (up) und bewegt den Grafikcursor. Wenn n ausgelassen wird, nimmt FreeBASIC 1 an.
- **D**[n] - Zeichnet eine Strecke n Pixel nach unten (down) und bewegt den Grafikcursor. Wenn n ausgelassen wird, nimmt FreeBASIC 1 an.
- **L**[n] - Zeichnet eine Strecke n Pixel nach links (left) und bewegt den Grafikcursor. Wenn n ausgelassen wird, nimmt FreeBASIC 1 an.
- **R**[n] - Zeichnet eine Strecke n Pixel nach rechts (right) und bewegt den Grafikcursor. Wenn n ausgelassen wird, nimmt FreeBASIC 1 an.
- **E**[n] - Zeichnet eine Strecke n Pixel nach oben rechts und bewegt den Grafikcursor. Wenn n ausgelassen wird, nimmt FreeBASIC 1 an.
- **F**[n] - Zeichnet eine Strecke n Pixel nach unten rechts und bewegt den Grafikcursor. Wenn n ausgelassen wird, nimmt FreeBASIC 1 an.
- **G**[n] - Zeichnet eine Strecke n Pixel nach unten links und bewegt den Grafikcursor. Wenn n ausgelassen wird, nimmt FreeBASIC 1 an.
- **H**[n] - Zeichnet eine Strecke n Pixel nach oben links und bewegt den Grafikcursor. Wenn n ausgelassen wird, nimmt FreeBASIC 1 an.

Farb-Befehle:

- **C**n- Benutzt die Farbe mit der Nummer n als neue Vordergrundfarbe.
- **P**p,b - Füllt einen Bereich mit der Randfarbe p in der Farbe b aus.

Skalierung und Rotation:

- **S**n - Setzt die aktuelle Länge pro Einheit. Standardmäßig sind 4 Einheiten eingestellt.
- **A**n - Dreht um n * 90 Grad (n muss zwischen 0 und 3 liegen).
- **TA**n - Dreht um n Grad (n muss zwischen 0 und 359 liegen).

Extrabefehle:

Xp - Führt Befehle von Adresse p aus. Sie können das so benutzen:

```
DIM AS STRING down10 = "D10"
DRAW "U10R5X" & VARPTR(down10) & "L5"
```

Beispiel 1:

```
DIM Stern AS STRING, buffer As ANY PTR

SCREENRES 640, 480

Stern  = "BF25 C14 NU20 NR20 ND20 NL20 C4 NE10 NF10 NG10 NH10"
buffer = ImageCreate(200, 200, 1)

DRAW buffer, Stern
PUT (10, 10), buffer

IMAGEDESTROY buffer
SLEEP
```

Beispiel 2: Farbangabe im 32bit-Farbmodus

```
' Haus vom Nikolaus in gelb
SCREENRES 200, 200, 32
DRAW "BM50,180 C" & RGB(255, 255, 0) & " U100 R100 D100 L100 E100 H50 G50
F100"
SLEEP
```

Unterschiede zu QB:

- In FreeBASIC ist es möglich, in einen Datenpuffer zu zeichnen.
- QB verwendet mit dem Kommando "**X**p" das spezielle Schlüsselwort VARPTR$.
- FreeBASIC erlaubt zur Zeit keine Bewegung in Teil-Pixeln: alle Bewegungen werden zu den nächsten INTEGER-Koordinatenwerte gerundet. In Zusammenhang mit einer Skalierung ist daher die Ausgabe in FreeBASIC möglicherweise nicht pixelgenau gleich wie in QB.

Siehe auch:
DRAW STRING, SCREENRES, Grafik

Letzte Bearbeitung des Eintrags am 27.12.12 um 00:23:21
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

DRAW STRING

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » D » **DRAW STRING**
Syntax: DRAW STRING [Puffer,] [STEP] (x, y), Text [, [Farbe] [, [Font] [, Methode]]]
Typ: Anweisung
Kategorie: Grafik

DRAW STRING gibt einen Text im Grafikmodus an pixelgenauen Koordinaten aus. Der Text kann transparent ausgegeben werden, der Hintergrund bleibt dabei erhalten.
DRAW STRING beeinflusst den Grafikcursor (nicht den Textcursor).

- Wird für 'Puffer' ein Pointer auf einen Speicherbereich angegeben, z.B. ein mit IMAGECREATE erstellter Puffer oder ein Array, wird der Text in diesen Bereich gezeichnet. Wird 'Puffer' ausgelassen, zeichnet FreeBASIC direkt auf den Bildschirm.
- 'STEP' gibt an, dass die folgenden Koordinaten (x, y) relativ zur aktuellen Position des Grafikcursors sein sollen.
- 'x' und 'y' sind die Koordinaten der linken oberen Position, an der die Textausgabe beginnen soll.
- 'Text' ist ein beliebiger STRING oder eine Zeichenkette, die ausgegeben werden soll. Auch WSTRINGs werden unterstützt, allerdings können nur die normalen ASCII-Zeichen korrekt dargestellt werden; Symbole, deren Zeichencode über 255 liegt, werden als Fragezeichen '?' dargestellt.
- 'Farbe' ist die Schriftfarbe, in welcher der Text ausgegeben werden soll. Wird dieses Argument ausgelassen, verwendet FreeBASIC die Vordergrundfarbe (siehe COLOR,).
- 'Font' ist ein Pointer auf einen Fontpuffer, der einen benutzerdefinierten Font enthält. Mehr zum Format dieses Puffers siehe unten.
- 'Methode' ist ein Aktionswort (evtl. mit Parameter), wie es bei PUT (Grafik) verwendet wird. Wird das Aktionswort ausgelassen, verwendet FreeBASIC automatisch TRANS. Aktionswörter können nur mit benutzerdefinierten Fonts verwendet werden, nicht mit dem Standard-Schriftsatz.

Beispiel 1: DRAW STRING

```
SCREENRES 320, 240, 32
Width 320/8, 240/16 ' Einstellen des großen Fonts
PAINT (0, 0), &H0000FF ' blauer Hintergrund

' "Hello World" auf dem Bildschirm ausgeben; Standardfarbe ist weiß
DRAW STRING ( 10, 2), "Hello World "

' Standardfarbe auf grün setzen und "Hello World" in dieser Farbe
ausgeben.
' Die Hintergrundfarbe hat keine Auswirkungen auf die Textausgabe.
COLOR &h00FF00, &hFFFFFF
DRAW STRING (20, 20), "Hello World"

' Umleitung der Ausgabe in einen Puffer
DIM TextPuffer AS ANY PTR
TextPuffer = IMAGECREATE(100, 16)
DRAW STRING TextPuffer, (0, 0), "Hello World", &hFFFFFF
' Die Hintergrundfarbe des Puffer wird gezeichnet!
PUT (100, 100), TextPuffer, PSET

' 'STEP' Schreibposition relativ zum letzten PUT (100, 100)
' hier Schriftfarbe gelb; Methode bleibt wirkungslos beim
Standard-Schriftsatz
```

```
DRAW STRING Step ( -40, +40), "Hello World", &HFFFF00, , PSET
SLEEP
```

Benutzerdefinierte Schriftarten
Benutzerdefinierte Schriftarten werden Zeichen für Zeichen nebeneinander in einem Puffer (Image) angelegt. Die Farbtiefe des Puffers muss mit der mittels SCREENRES eingestellten Farbtiefe des Grafikfensters übereinstimmen. Andernfalls erzeugt DRAW STRING den Fehler "Unzulässiger Funktionsaufruf" (siehe Fehler-Behandlung in FreeBASIC). Der Puffer muss also so breit sein, wie alle Zeichen nebeneinander sind. Die erste Zeile des Puffers enthält den 'FontHeader' der Schriftart. Darin ist angegeben, welche Zeichen im Font dargestellt sind und wie breit diese sind. Folglich ist die Höhe des Puffers gleich der Höhe der Zeichen plus eins. Insgesamt ergibt sich dieser Speicheraufbau:

Anzahl Type	ImageHeader
UINTEGER	Version des ImageHeaders =7
INTEGER	Byte pro Pixel
UINTEGER	Breite des Image in Pixel
UINTEGER	Höhe des Image in Pixel
UINTEGER	Byte pro Zeile des Image (Pitch)
12 x UBYTE	reserviert
Anzahl Type	**FontHeader (1. Zeile des Image)**
BYTE	Version des FontHeaders =0
UBYTE	ASCII-Code des ersten Zeichens im Puffer
UBYTE	ASCII-Code des letzten Zeichens im Puffer
UBYTE	Breite des 1. Zeichens
UBYTE	Breite des 2. Zeichens
je 1 UBYTE	Breite nächstes Zeichen
"	"
UBYTE	Breite des letzten Zeichens
XX UBYTE	unbenutzt, bis Ende der 1. Zeile
	Image
XX	2. Zeile bis letzte Zeile

Beispiel 2: Erstellen und Speichern eines benutzerdefinierte Fonts

```
' CHR-Bereich festlegen
Const Erster = 32, Letzter = 127, Anzahl = (Letzter - Erster) + 1

Dim As UByte Ptr p, myFont
Dim As Integer farbe
' Einen 256 Farbgrafik-Screen (320*200) erzeugen
ScreenRes 320, 200, 8

' Erstellen eines Benutzerfonts in einem Image
' (Anzahl Zeichen * Zeichenbreite , Zeichenhöhe + 1(für Fontheader))
myFont = ImageCreate(Anzahl * 8, 8 +1)
' p = Zeiger auf den FontHeader
p = myFont + IIf(myFont&"hlzahl">0] = 7, 32, 4)
p[0] = 0       'Fontversion bisher immer 0
p[1] = Erster  'erster Buchstabe im Font
p[2] = Letzter 'letzter Buchstabe im Font
```

```
For i As Integer = Erster To Letzter
  p&"hlzahl">3 + i - Erster] = 8   'Zeichenbreite
  farbe = 60 + (i Mod 24) 'Zeichenfarbe
  'Zeichen in Font-Puffer kopieren (aus kleinem Standard Font)
  Draw String myFont, ((i - Erster) * 8, 1), Chr(i), farbe
Next i

' Den Font-Puffer können wir mit BSAVE abspeichern.
' Die Anzahl zu speichernde Byte berechnet sich aus
' (Anzahl * Zeichenbreite * (Zeichenhöhe + 1) * Byte_per_Pixel) +
ImageHeader
BSave "myfont.fbf", myFont, (Anzahl * 8 * 9 * 1)+32

' Als Bitmap können wir den Font so speichern:
BSave "myfont.bmp", myFont

' Hier zeichnen wir einen String mit unserem Font
Draw String (10, 10), "ABCDEFGHIJKLMNOPQRSTUVWXYZ",, myFont
Draw String (10, 26), "abcdefghijklmnopqrstuvwxyz",, myFont
Draw String (66, 58), "Hello world!",, myFont

ImageDestroy myFont 'Speicherbereich freigeben

Sleep
```

- Einmal erstellt, kann eine Schriftart wie im Beispiel mit BSAVE gespeichert und später mit BLOAD wieder geladen werden.
- Die Verwendung des Farb-Parameters ist mit eigenen Schriftarten nicht möglich, da die Schriftart selbst schon die Farbe beinhalten kann.

Beispiel 3: Der Parameter 'Methode' mit selbstdefinierten Fonts:

```
' Einen 256 Farbgrafik-Screen (320*200) erzeugen
ScreenRes 320, 200, 8

' Für dieses Beispiel brauchen wir den zuvor
' erstellten Font 'myfont.fbf', siehe Beispiel 2

' Dies brauche ich nur wenn Breite und Hoehe unbekannt sind
' sonst breit = Anzahl * 8 : hoch = 9
Dim As Integer breit, hoch, ff = FreeFile
If Open("myfont.fbf" For Binary Access Read As "hlzeichen">)<>0 Then
  Close "hlkw0">Print "keine Fontdatei gefunden!"
  Sleep
  End
End If
Get "hlzeichen">, 14, breit
Get "hlzeichen">, 18, hoch
Close "hlkommentar">' Erstellen eines Benutzerfonts im Image
Dim As UByte Ptr myFont = ImageCreate(breit, hoch)

' Jetzt können wir den Font mit BLoad laden
BLoad "myfont.fbf", myFont
```

```
' Ein blauer Hintergrund
Paint (0, 0), 1
' Hier zeichnen wir einen String mit unserem Font und 'Methode'
Draw String (10, 10), "PSET:   ABCDEFGHIJKLMNOPQRSTUVWXYZ",, myFont, PSet
Draw String (10, 26), "PRESET: ABCDEFGHIJKLMNOPQRSTUVWXYZ",, myFont,
Preset
Draw String (10, 42), "AND:    ABCDEFGHIJKLMNOPQRSTUVWXYZ",, myFont, And
Draw String (10, 58), "OR:     ABCDEFGHIJKLMNOPQRSTUVWXYZ",, myFont, Or
Draw String (10, 74), "XOR:    ABCDEFGHIJKLMNOPQRSTUVWXYZ",, myFont, Xor
Draw String (10, 90), "normal: abcdefghijklmnopqrstuvwxyz",, myFont
Draw String (66, 106), "Hello world!",, myFont
ImageDestroy myFont 'Speicherbereich freigeben

Sleep
```

Für die Verwendung des Aktionswortes CUSTOM siehe PUT (Grafik).

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC:

- DRAW STRING existiert seit FreeBASIC v0.16.
- Das Aktionswort ADD existiert seit FreeBASIC v0.17.

Siehe auch:
DRAW (Grafik), PRINT (Anweisung), LOCATE (Anweisung), WIDTH (Anweisung), Grafik

Letzte Bearbeitung des Eintrags am 27.12.12 um 00:23:41
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

DYLIBFREE

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » D » **DYLIBFREE**
Syntax: DYLIBFREE handle
Typ: Anweisung
Kategorie: Bibliotheken

DYLIBFREE gibt den Speicher wieder frei, der durch eine geladene dll/so belegt wurde.

'handle' ist die Nummer der dll/so (als Pointer), die durch DYLIBLOAD zurückgegeben wurde.

Beispiel: siehe DYLIBLOAD

Unterschiede zu QB: neu in FreeBASIC

Plattformbedingte Unterschiede: wird unter DOS nicht unterstützt

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht DYLIBFREE nicht zur Verfügung und kann nur über **__DYLIBFREE** aufgerufen werden.

Unterschiede zu früheren Versionen von FreeBASIC: existiert seit FreeBASIC v0.14

Siehe auch:
DYLIBLOAD, DYLIBSYMBOL, Module (Library / DLL)

Letzte Bearbeitung des Eintrags am 15.06.12 um 23:06:31
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

DYLIBLOAD

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » D » **DYLIBLOAD**
Syntax: DYLIBLOAD (LibName)
Typ: Funktion
Kategorie: Bibliotheken

DYLIBLOAD versucht, eine Dynamic Link Library (dll) bzw. ein Shared Object (so) zu laden, und gibt in einem Pointer einen Handle zu dieser dll/so aus.

- 'LibName' ist ein STRING mit dem Namen der dll/so, eventuell mit Pfadangabe. Wird keine Erweiterung angegeben, geht FreeBASIC unter Win32 automatisch von ".dll" und unter Linux von ".so" aus. Eventuell wird vom Compiler nicht geprüft, ob eine Erweiterung angegeben wurde; Sie sollten keine Erweiterung angeben.
- Der Rückgabewert ist ein ANY PTR mit dem Handle zur dll/so (eine Zahl, über die die dll/so identifiziert und aufgerufen werden kann), wenn sie geladen werden konnte, oder null (0), wenn der Ladeversuch fehlschlägt.

Durch DYLIBLOAD wird eine externe dll bzw. so zur Laufzeit in den Speicher geladen. Ihre Funktionen sind ab diesem Zeitpunkt theoretisch verfügbar, müssen aber erst durch DYLIBSYMBOL definiert werden, um aufgerufen werden zu können.

Nach der Nutzung der Funktionen der DLL sollte der belegte Speicher durch DYLIBFREE freigegeben werden.

Beispiel (aus dem Verzeichnis examples/dll in Ihrem FreeBASIC-Ordner):
In diesem Beispiel wird angenommen, dass sich im FreeBASIC-Verzeichnis die Datei "mydll.dll" befindet, die die Funktion "AddNumbers" enthält.

```
DIM library AS ANY PTR
DIM addnumbers AS FUNCTION( BYVAL operand1 AS INTEGER, _
    BYVAL operand2 AS INTEGER ) AS INTEGER

library = DYLIBLOAD( "mydll" )
IF( library = 0 ) THEN
  PRINT "Cannot load the mydll dynamic library, ";
  PRINT "aborting program..."
  END 1
END IF

addnumbers = DYLIBSYMBOL( library, "AddNumbers" )
IF( addnumbers = 0 ) THEN
  PRINT "Cannot get AddNumbers function address ";
  PRINT "from mydll library, aborting program..."
  END 1
END IF

RANDOMIZE TIMER

x = RND * 10
y = RND * 10

PRINT x; " +"; y; " ="; addnumbers( x, y )
```

```
DYLIBFREE library
```

Unterschiede zu QB: neu in FreeBASIC

Plattformbedingte Unterschiede: wird unter DOS nicht unterstützt

Unterschiede zu früheren Versionen von FreeBASIC: existiert seit FreeBASIC v0.13

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht DYLIBLOAD nicht zur Verfügung und kann nur über **__DYLIBLOAD** aufgerufen werden.

Siehe auch:
DYLIBFREE, DYLIBSYMBOL, Module (Library / DLL)

Letzte Bearbeitung des Eintrags am 15.06.12 um 23:11:25
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

DYLIBSYMBOL

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » D » **DYLIBSYMBOL**
Syntax: DYLIBSYMBOL (handle, Symbol)
Typ: Funktion
Kategorie: Bibliotheken

DYLIBSYMBOL gibt den Pointer auf eine Prozedur oder Variable innerhalb einer Dynamic Link Library (dll) bzw. einem Shared Object (so) zurück und macht sie so im Programm aufrufbar.

- 'handle' ist ein ANY PTR mit dem Handle einer dll/so, die durch DYLIBLOAD zurückgegeben wurde.
- 'Symbol' ist ein String mit dem Namen, den die Prozedur oder Variable innerhalb der dll/so trägt. Unter Windows kann auch die Ordnungsnummer der Prozedur bzw. Variablen angegeben werden.
- Der Rückgabewert ist ein ANY PTR, der die Adresse der Prozedur bzw. Variablen enthält. Schlägt der Aufruf fehl, so wird 0 zurückgegeben.

Beachten Sie, dass Sie die Prozedur nicht mit DECLARE, sondern mit

```
DIM name AS FUNCTION (Parameterliste) AS Type
'oder
DIM name AS SUB (Parameterliste)
```

deklarieren müssen.

Beispiel: siehe DYLIBLOAD

Unterschiede zu QB: neu in FreeBASIC

Plattformbedingte Unterschiede:

- Dynamic Link Libraries werden unter DOS nicht unterstützt.
- Unter Windows kann statt des Namens auch die Nummer der Prozedur bzw. Variablen angegeben werden.

Unterschiede zu früheren Versionen von FreeBASIC:

- Seit FreeBASIC v0.17 kann unter Windows auch die Nummer der Prozedur bzw. Variablen angegeben werden.
- DYLIBSYMBOL existiert seit FreeBASIC v0.13

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht DYLIBSYMBOL nicht zur Verfügung und kann nur über **__DYLIBSYMBOL** aufgerufen werden.

Siehe auch:
DYLIBLOAD, DYLIBFREE, Module (Library / DLL)

Letzte Bearbeitung des Eintrags am 26.10.12 um 11:06:09
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

DYNAMIC (Meta)

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » D » **DYNAMIC (Meta)**
Syntax: '$DYNAMIC
Typ: Metabefehl
Kategorie: Metabefehle

'$DYNAMIC ist ein Metabefehl, der sich auf die Verwaltung von Arrays im Speicher auswirkt. Durch ihn werden alle Arrays als dynamische Arrays definiert, so dass sie eine variable Größe besitzen und zur Laufzeit redimensioniert werden können. Der Metabefehl hat dieselbe Wirkung wie OPTION DYNAMIC. Der Metabefehl STATIC und OPTION STATIC setzen diesen Standard außer Kraft. Wird keiner von beiden Standards explizit gesetzt, geht FreeBASIC von OPTION STATIC aus.

'$DYNAMIC kann **nur bis FreeBASIC v0.16** eingesetzt werden, oder in entsprechend höheren Versionen, die mit der Kommandozeilenoption -lang deprecated compiliert wurden! Wird mit FreeBASIC v0.17 unter der Option -lang fb compiliert, so ist '$DYNAMIC nicht mehr zulässig! Um ein Array dynamischer Verwaltung zu erstellen, dimensionieren Sie es mit REDIM oder geben Sie bei der Erstellung keine Dimensionen an:

```
REDIM Dynamisch1 (200) AS INTEGER
DIM Dynamisch2 () AS INTEGER
```

Beispiel:

```
"hlstring">"deprecated"
'$DYNAMIC
DIM a(100)
ERASE a
REDIM a(200)
```

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC:
Der Metabefehl ist nur bis FreeBASIC v0.16 erlaubt. Seit FreeBASIC v0.17 ist der Befehl nur noch zulässig, wenn mit der Kommandozeilenoption -lang deprecated compiliert wurde.

Siehe auch:
DYNAMIC (Schlüsselwort), STATIC (Meta), DIM, REDIM, Präprozessor-Anweisungen, Verschiedenes

Letzte Bearbeitung des Eintrags am 19.01.13 um 16:38:13
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

DYNAMIC (Schlüsselwort)

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » D » **DYNAMIC (Schlüsselwort)**
Syntax: OPTION DYNAMIC
Typ: Schlüsselwort
Kategorie: Programmoptionen

OPTION DYNAMIC legt fest, dass Arrays standardmäßig DYNAMIC verwaltet werden sollen. Der Metabefehl DYNAMIC hat dieselbe Wirkung wie OPTION DYNAMIC. Der Metabefehl STATIC und OPTION STATIC setzen diesen Standard außer Kraft. Wird keiner von beiden Standards explizit gesetzt, geht FreeBASIC von OPTION STATIC aus.

OPTION DYNAMIC kann **nur bis FreeBASIC v0.16** eingesetzt werden, oder in entsprechend höheren Versionen, die mit der Kommandozeilenoption -lang deprecated compiliert wurden! Wird mit FreeBASIC v0.17 unter der Option -lang fb compiliert, so ist OPTION DYNAMIC nicht mehr zulässig! Um ein Array dynamischer Verwaltung zu erstellen, dimensionieren Sie es mit REDIM oder geben Sie bei der Erstellung keine Dimensionen an:

```
REDIM Dynamisch1 (200) AS INTEGER
DIM Dynamisch2 () AS INTEGER
```

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC:
Die Option ist nur bis FreeBASIC v0.16 erlaubt. Seit FreeBASIC v0.17 ist diese Option nur noch zulässig, wenn mit der Kommandozeilenoption -lang deprecated compiliert wurde.

Siehe auch:
DYNAMIC (Metabefehl), STATIC (Schlüsselwort), __FB_OPTION_DYNAMIC__, OPTION, DIM, REDIM

Letzte Bearbeitung des Eintrags am 19.01.13 um 16:39:14
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

ELSE

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » E » **ELSE**
Typ: Schlüsselwort
Kategorie: Programmablauf

ELSE wird im Zusammenhang mit IF ... THEN verwendet. Es beschreibt den Fall, wenn alle anderen definierten Fälle nicht zutreffen.

Beispiel:

```
"hlzahl">0
"hlzeichen">(Not FALSE)

Dim As Integer Ausdruck = TRUE

If Ausdruck = TRUE Then
    Print "Die Pruefung ob Ausdruck gleich TRUE ist, ist wahr."
ElseIf Ausdruck = FALSE Then
    Print "Die Pruefung ob Ausdruck gleich FALSE ist, ist wahr."
Else
    Print "Diesen Fall kann es nicht geben."
End If

Sleep
```

Verwendung als Metabefehl:

Typ: Metabefehl
Kategorie: Präprozessoren

"reflinkicon" href="temp0190.html">IF (Meta), IFDEF (Meta) und IFNDEF (Meta) verwendet.

Beispiel:

```
"hlkw0">Defined(FALSE)
    'nichts
"hlkw0">"hlzahl">0
"reflinkicon" href="temp0191.html">IF ... THEN, IF (Meta), IFDEF (Meta),
IFNDEF (Meta), Bedingungsstrukturen, Präprozessor-Anweisungen,
Programmablauf
```

Letzte Bearbeitung des Eintrags am 27.12.12 um 00:26:00
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

ELSEIF

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » E » **ELSEIF**
Typ: Schlüsselwort
Kategorie: Programmablauf

ELSEIF wird im Zusammenhang mit IF ... THEN verwendet. Es beschreibt einen vom ersten Prüffall unterschiedlichen Fall.

Beispiel:

```
"hlzahl">0
"hlzeichen">(Not FALSE)

Dim As Integer Ausdruck = TRUE

If Ausdruck = TRUE Then
    Print "Die Pruefung ob Ausdruck gleich TRUE ist, ist wahr."
ElseIf Ausdruck = FALSE Then
    Print "Die Pruefung ob Ausdruck gleich FALSE ist, ist wahr."
Else
    Print "Diesen Fall kann es nicht geben."
End If

Sleep
```

Verwendung als Metabefehl:

Typ: Metabefehl
Kategorie: Präprozessoren

"reflinkicon" href="temp0190.html">IF (Meta), IFDEF (Meta) und IFNDEF (Meta) verwendet.

Beispiel:

```
"hlkw0">Defined(FALSE)
    'nichts
"hlkw0">Not Defined(FALSE)
    "hlzahl">0
"reflinkicon" href="temp0191.html">IF ... THEN, IF (Meta), IFDEF (Meta),
IFNDEF (Meta), Bedingungsstrukturen, Präprozessor-Anweisungen,
Programmablauf
```

Letzte Bearbeitung des Eintrags am 27.12.12 um 00:26:12
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

ENCODING

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » E » **ENCODING**
Syntax: OPEN Datei FOR Dateimodus [...] ENCODING { "ASCII" | "UTF-8" | "UTF-16" | "UTF-32" } AS [#]Dateinummer [...]
Typ: Schlüsselwort
Kategorie: Dateien

ENCODING wird zusammen mit OPEN (Anweisung) verwendet, um festzulegen, mit welcher Zeichenkodierung die Daten behandelt werden sollen. Möglich sind:

- 'ENCODING "ASCII"' - Standard. Die Daten werden im ASCII-Format behandelt.
- 'ENCODING "UTF-8"' - gibt an, dass die Daten UTF-8-codiert sind.
- 'ENCODING "UTF-16"' - gibt an, dass die Daten UTF-16-codiert sind.
- 'ENCODING "UTF-32"' - gibt an, dass die Daten UTF-32-codiert sind.

Die ENCODING-Klausel kann nur in sequentiellen Modi verwendet werden (INPUT, OUTPUT, APPEND).

Achtung: ENCODING kann zusammen mit UTF-codierten Dateien nur dann erfolgreich eingesetzt werden, wenn das Byte Order Mark (BOM) gesetzt ist und mit der angegebenen Codierung übereinstimmt. Ansonsten wird der Laufzeitfehler 2 (File not found) zurückgegeben.

Unterschiede zu QB: Unicode wird unter QB nicht unterstützt.

Plattformbedingte Unterschiede: Unicode wird unter DOS nicht unterstützt.

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht ENCODING nicht zur Verfügung.

Siehe auch:
OPEN (Anweisung), PRINT (Datei), INPUT (Datei), WSTRING (Datentyp), Dateien (Files)

Letzte Bearbeitung des Eintrags am 15.06.13 um 13:43:22
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

END

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » E » **END**
Syntax A: END Block-Bezeichnung
Syntax B: END [Rückgabewert]
Typ: Anweisung
Kategorie: System

END beendet eine Blockstruktur (Syntax A) oder das Programm (Syntax B).

Mit Syntax A kann einer der folgenden Code-Blöcke abgeschlossen werden:

- SUB
- FUNCTION
- IF
- SELECT
- TYPE
- ENUM
- SCOPE
- WITH
- NAMESPACE
- EXTERN
- CONSTRUCTOR
- DESTRUCTOR
- OPERATOR
- PROPERTY

Syntax B beendet das laufende Programm und kehrt zum Betriebssystem zurück. 'Rückgabewert' ist ein INTEGER, das als sogenannter Errorlevel zurückgegeben wird und beispielsweise in BATCH-Programmen abgefragt werden kann. Eine 0 oder keine Angabe sind gleichbedeutend mit "Kein Fehler". Alles andere wird als Fehler interpretiert.

Wenn das Programm mit END beendet wird, dann werden Variablen und Speicher nicht automatisch zerstört. Die Destruktoren von Objekten werden nicht aufgerufen. Benötigte Objekt-Destruktoren müssen daher vor der END-Anweisung explizit aufgerufen werden.

Beispiel:

```
DIM AS INTEGER a
IF a = 2 THEN
   ' ...
ELSE
   ' ...
END IF              ' IF-Block beenden

SELECT CASE ERR
  CASE 0
    END             ' Programm normal beenden
  CASE 1, 2, 7
    END 1           ' Programm-Ende mit Errorlevel 1
  CASE 5, 23, 70
    END 2           ' Programm-Ende mit Errorlevel 2
  CASE ELSE
    END -1          ' Programm-Ende mit Errorlevel -1
```

```
END SELECT          ' SELECT-Block beenden
```

Unterschiede zu QB:
Die Rückgabe eines Errorlevels an das Betriebssystem ist neu in FreeBASIC.

Plattformbedingte Unterschiede:
Unter Linux wird für den Errorlevel eines Programmes nur ein Wert von -1 bis 254 verwendet. Zwar kann an END ein INTEGER-Wert übergeben werden, es wird vom System jedoch nur ein UBYTE weitergereicht (wobei 255 als -1 interpretiert wird).

Siehe auch:
SYSTEM, STOP, Betriebssystem-Anweisungen

Letzte Bearbeitung des Eintrags am 27.12.12 um 00:26:47
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

ENDIF

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » E » **ENDIF**
Typ: Schlüsselwort
Kategorie: Programmablauf

ENDIF wird im Zusammenhang mit IF ... THEN verwendet. Es schließt eine Bedingungsstruktur ab. Die Form "END IF" mit Leerzeichen kann ebenfalls verwendet werden.

Beispiel:

```
"hlzahl">0
"hlzeichen">(Not FALSE)

Dim As Integer Ausdruck = TRUE

If Ausdruck = TRUE Then
    Print "Die Pruefung ob Ausdruck gleich TRUE ist, ist wahr."
ElseIf Ausdruck = FALSE Then
    Print "Die Pruefung ob Ausdruck gleich FALSE ist, ist wahr."
Else
    Print "Diesen Fall kann es nicht geben."
EndIf

Sleep
```

Verwendung als Metabefehl:

Typ: Metabefehl
Kategorie: Präprozessoren

"reflinkicon" href="temp0190.html">IF (Meta), IFDEF (Meta) und IFNDEF (Meta) verwendet.

Beispiel:

```
"hlkw0">Defined(FALSE)
    'nichts
"hlkw0">"hlzahl">0
"reflinkicon" href="temp0191.html">IF ... THEN, IF (Meta), IFDEF (Meta),
IFNDEF (Meta), Bedingungsstrukturen, Präprozessor-Anweisungen,
Programmablauf
```

Letzte Bearbeitung des Eintrags am 27.12.12 um 00:27:23
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

ENDMACRO (Meta)

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » E » **ENDMACRO (Meta)**
Syntax:

```
"hlzeichen">[(Parameterliste)]
   ' Makro-Code
"reflinkicon" href="temp0248.html">#MACRO benutzt und beendet einen
Makro-Block.
```

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC: existiert seit FreeBASIC v0.17

Siehe auch:
#MACRO, Präprozessoren, Präprozessor-Anweisungen
Letzte Bearbeitung des Eintrags am 27.12.12 um 00:28:08
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

ENUM

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » E » **ENUM**
Syntax A:

```
ENUM &"hlzeichen">[EXPLICIT]]
   Element1 [= Integerwert]
   Element2 [= Integerwert]
   Element3 [= Integerwert]
   [...]
END ENUM
```

Syntax B:

```
ENUM &"hlzeichen">[EXPLICIT]]
   Element1 [= Integerwert], Element2 [= Integerwert] [, ...]
END ENUM
```

Typ: Anweisung
Kategorie: Klassen

ENUM (kurz für enumeration = Nummerierung) erzeugt eine Liste von Konstanten vom Typ INTEGER, deren Werte ihrer Position im ENUM-Block entsprechen. Begonnen wird dabei mit dem Wert 0.

- 'Listenname' ist eine Bezeichnung des ENUMs, mit der das ENUM im Zusammenhang mit DIM wie ein Datentyp verwendet werden kann.
- 'EXPLICIT' gibt an, dass der Aufruf eines Elements zwangsweise in der Form Listenname.Element verwendet werden muss.
- 'Element1', 'Element2', ... sind die Namen der angelegten Listen-Elemente.
- 'Integerwert' ist ein INTEGERs, das angibt, mit welcher Zahl bei der Nummerierung fortgefahren werden soll.

Da sich die Elemente wie Konstanten verhalten, ist es nicht möglich, ihnen im Laufe des Programms andere Werte zuzuweisen.

Beispiel:

```
ENUM meineListe
   a, b
   c = 4
   d
END ENUM

PRINT a, b, c, d
' oder auch
PRINT meineListe.b
DIM AS meineListe wert = c
PRINT wert
SLEEP
```

Ausgabe:

```
0 1 4 5
1
```

```
4
```

Unterschiede zu QB: neu in FreeBASIC

Unterschiede unter den FB-Dialektformen: In der Dialektform -lang qb steht EXPLICIT nicht zur Verfügung und kann nur über **__EXPLICIT** aufgerufen werden.

Siehe auch: DIM, CONST, DEFINE (Meta), Datentypen und Deklarationen

Letzte Bearbeitung des Eintrags am 04.07.12 um 21:11:10
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

ENVIRON

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » E » **ENVIRON**
Syntax: ENVIRON[$] (Systemumgebungsvariable)
Typ: Funktion
Kategorie: System

ENVIRON gibt den Wert einer Systemumgebungsvariablen zurück. Ist die Variable nicht belegt, so wird ein Leerstring "" zurückgegeben.

Das Dollarzeichen ($) als Suffix ist optional.

Beispiel: den Wert der Systemvariable PATH ausgeben:

```
PRINT ENVIRON("PATH")
```

Unterschiede zu QB:
In FreeBASIC kann ENVIRON nicht verwendet werden, um den Wert einer Variablen zu setzen; diese Funktion wurde von SETENVIRON übernommen.

Plattformbedingte Unterschiede:
Unter Linux muss die Systemvariable 'case sensitive' erfolgen. Windows und DOS sind 'case insensitive'.

Unterschiede unter den FB-Dialektformen:

- In der Dialektform -lang qb ist das Suffix $ verbindlich.
- In den Dialektformen -lang fblite und -lang fb ist das Suffix optional.

Siehe auch:
SETENVIRON, Betriebssystem-Anweisungen

Letzte Bearbeitung des Eintrags am 27.12.12 um 00:28:41
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

EOF

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » E » **EOF**
Syntax: EOF(f)
Typ: Funktion
Kategorie: Dateien

EOF steht für end-of-file (Ende der Datei) und gibt true (-1) zurück, wenn der Dateizeiger das Ende einer geöffneten Datei erreicht hat.

'f' ist die Nummer der Datei, die geprüft werden soll. Diese Nummer wird in der OPEN-Anweisung festgelegt.

Beispiel:

```
DIM AS INTEGER f
DIM AS STRING txt

' eine unbenutzte Dateinummer finden und Datei zum Einlesen öffnen
f = FREEFILE
OPEN "file.ext" FOR INPUT AS "hlkommentar">' Daten so lange einlesen, bis
das Ende erreicht wurde.
DO UNTIL EOF(f)
   ' Zeile aus der Datei einlesen und auf dem Bildschirm ausgeben
   INPUT "hlzeichen">, txt
   PRINT txt
LOOP

' Datei schließen
CLOSE "hlkommentar">'Auf Tastendruck vor dem Beenden warten
SLEEP
```

Wegen der plattformbedingten Unterschiede in den vom Compiler verwendeten Bibliotheken kann es zu Problemen kommen, wenn die EOF-Funktion verwendet wird, um in Linux erstellte Textdateien (mit LF als Zeilenende) in einem für Windows compilierten Programm zu verwenden. Die DOS- und Linux-Compiler haben dieses Problem nicht. Eine mögliche Lösung des Problems ist die Verwendung des Dateimodus BINARY statt INPUT. LINE INPUT"reflinkicon" href="temp0285.html">COM-Port ein EOF, wenn dort keine Zeichen zum Einlesen warten.

- In QB geben Dateien, die im Dateimodus RANDOM oder BINARY geöffnet werden, bei EOF nur dann einen Wert ungleich 0 zurück, nachdem versucht wurde, hinter dem Dateiende zu lesen. In FreeBASIC gibt EOF true (-1) zurück, sobald der letzte Eintrag gelesen wurde.

Siehe auch:
LOF, LOC, OPEN, Dateien (Files)

Letzte Bearbeitung des Eintrags am 27.12.12 um 00:28:53
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

EQV

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » E » **EQV**
Syntax A: Ergebnis = Ausdruck1 EQV Ausdruck2
Syntax B: Ausdruck1 EQV= Ausdruck2
Typ: Operator
Kategorie: Operatoren

EQV kann als einfacher (Syntax A) und kombinierter (Syntax B) Operator eingesetzt werden. Syntax B ist eine Kurzform von

```
Ausdruck1 = Ausdruck1 EQV Ausdruck2
```

EQV (EQuivalent Value) vergleicht zwei Werte Bit für Bit und setzt im Ergebnis nur dann ein Bit, wenn die entsprechenden Bits in beiden Ausdrücken gleichwertig waren. EQV wird in Bedingungen eingesetzt, wenn beide Aussagen denselben Status haben müssen, also entweder beide wahr oder beide falsch sein müssen.

Beispiel 1: EQV in einer IF-THEN-Bedingung:

```
IF (a = 1) EQV (b = 7) THEN
PRINT "beides erfüllt oder beides nicht erfüllt."
ELSE
PRINT "Entweder a <> 1 aber b = 7 oder a = 1 aber b <> 7."
END IF
```

Beispiel 2: Verknüpfung zweier Zahlen mit EQV:

```
DIM AS INTEGER z1, z2

z1 =  6
z2 = 10

PRINT z1, BIN(z1, 4)
PRINT z2, BIN(z2, 4)
PRINT "---", "----"
PRINT (z1 EQV z2) AND 15, BIN( (z1 EQV z2) AND 15, 4)
SLEEP
```

Ausgabe:

```
  6           0110
 10           1010
---           ----
  3           0011
```

Anmerkung dazu: Die angewandten AND 15 bewirken, dass nur die letzten vier Bits angezeigt werden. Der Grund dafür ist, dass INTEGER-Variablen aus 32bit bestehen. Die übrigen 28 Bits werden selbstverständlich auch mit EQV verglichen. Würden diese in dieses Beispiel mit einbezogen, dann würde das Beispiel einen Teil seiner Anschaulichkeit verlieren.

Beispiel 3: EQV als kombinierter Operator

```
DIM AS UBYTE a
```

```
a  = &b00110011
PRINT BIN(a, 8)
a EQV= &b01010101
PRINT "01010101"
PRINT "--------"

PRINT BIN(a, 8)
SLEEP
```

Ausgabe:

```
00110011
01010101
--------
10011001
```

Unterschiede zu QB:
Kombinierte Operatoren sind neu in FreeBASIC.

Siehe auch:
NOT, AND (Operator), OR (Operator), XOR (Operator), IMP, Bit Operatoren / Manipulationen

Letzte Bearbeitung des Eintrags am 04.07.12 um 21:17:04
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

ERASE

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » E » **ERASE**
Syntax: ERASE Array [, Array [,...]]
Typ: Anweisung
Kategorie: Speicher

ERASE löscht dynamische Arrays aus dem Speicher oder setzt alle Elemente eines statischen Arrays auf 0 bzw. auf den Initialisationswert. Es dürfen beliebig viele Array-Bezeichner angegeben werden.

Beispiel:

```
type MyUdt
  as integer x, y
  declare constructor
  declare constructor(xx as integer, yy as integer)
end type

constructor MyUdt
  this.CONSTRUCTOR(1, 2)
end constructor

constructor MyUdt(xx as integer, yy as integer)
  this.x = xx
  this.y = yy
end constructor

dim zahl(2) as integer = {0, 1, 2}
dim text(2) as string = {"a", "b", "c"}
redim dynamisch(2) as integer
dim udt(2) as MyUdt
udt(0) = MyUdt(9, 9)
udt(1) = MyUdt(8, 8)
udt(2) = MyUdt(7, 7)

' Arrays ausgeben
print "zahl()", "text()", "udt()"
for i as integer = 0 to 2
  print zahl(i), text(i), udt(i).x & "/" & udt(i).y
next
print "Adresse von dynamisch(): " & @dynamisch(0)
print

' Arrays löschen und erneut ausgeben
erase zahl, text, udt, dynamisch
print "zahl()", "text()", "udt()"
for i as integer = 0 to 2
  print zahl(i), text(i), udt(i).x & "/" & udt(i).y
next
print "Adresse von dynamisch(): " & @dynamisch(0)
sleep
```

Ausgabe:

```
zahl()         text()         udt()
 0             a              9/9
 1             b              8/8
 2             c              7/7
Adresse von dynamisch(): 143869584

zahl()         text()         udt()
 0                            1/2
 0                            1/2
 0                            1/2
Adresse von dynamisch(): 0
```

Achtung: Wird ein statisches Array als Parameter an eine Prozedur übergeben, so wird es dort als dynamisches Array angesehen. Die Verwendung von ERASE führt dann zu einem Speicherzugriffsfehler.

Unterschiede zu früheren Versionen von FreeBASIC:

- Seit FreeBASIC v0.24 muss ein UDT einen Standard-Konstruktor besitzen, wenn ein UDT-Array mit ERASE gelöscht werden soll.
- Vor FreeBASIC v0.11 war der Einsatz von ERASE nur für dynamische Arrays zulässig.

Siehe auch:
DIM, CLEAR, Arrays

Letzte Bearbeitung des Eintrags am 07.12.13 um 17:28:31
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

ERFN

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » E » **ERFN**
Syntax: ERFN
Typ: Funktion
Kategorie: Fehlerbehandlung

ERFN gibt einen ZSTRING PTR auf den Namen der Prozedur zurück, in der ein Fehler aufgetreten ist. Das Programm muss dabei mit der Kommandozeilenoption -exx compiliert werden. Ist dies nicht der Fall, so kann ERFN nicht eingesetzt werden; der Rückgabewert ist in diesem Fall immer null.

Beispiel:
Compilieren Sie dieses Beispiel mit der Kommandozeile

```
fbc -exx -lang deprecated ERFN_Test.bas
```

```
' ERFN_Test.bas

Declare Sub Generate_Error

Sub Generate_Error
   On Error Goto Handler
   Error 1000
   Exit Sub

   Handler:
      Print "Error Function: "; *Erfn()
      Print "Error Module  : "; *Ermn()
      Resume Next
End Sub

Generate_Error
Sleep
```

Ausgabe:

```
Error Function: GENERATE_ERROR
Error Module  : ERFN_Test.bas
```

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC: existiert seit FreeBASIC v0.16

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht ERFN nicht zur Verfügung und kann nur über **__ERFN** aufgerufen werden.

Siehe auch:
ERMN, ERROR, ON ERROR, Fehlerbehandlung, Debugging

Letzte Bearbeitung des Eintrags am 19.01.13 um 15:53:17
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

ERL

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » E » **ERL**
Syntax: ERL
Typ: Funktion
Kategorie: Fehlerbehandlung

ERL gibt eine INTEGER-Zahl zurück, welche die Zeilennummer des letzten aufgetretenen Fehlers enthält. Ist kein Fehler aufgetreten, dann gibt ERL 0 zurück.

ERL kann nur effektiv genutzt werden, wenn die QB-ähnliche Fehlerbehandlung aktiviert ist (siehe ON ERROR).

Beispiel:

```
"hlstring">"fblite"    ' notwendig für RESUME NEXT

' Hinweis: muss mit der Option -ex oder -exx
' compiliert werden (wegen RESUME NEXT)

On Error Goto ErrorHandler

' Fehler erzeugen
Error 100

Sleep
End

ErrorHandler:
  Dim num As Integer = Err
  Print "Fehler "; num; " in Zeile "; ERL
  Resume Next
```

Unterschiede zu QB:
FreeBASIC gibt die Zeilennummern des Quellcodes zurück und ignoriert die Werte explizierter Zeilennummern, während QB den Wert der letzten explizierten Zeilennummer zurück gibt.

Siehe auch:
ERROR (Anweisung), ERR (Funktion), __LINE__, Fehler-Behandlung in FreeBASIC, Übersicht: Fehlerbehandlung, Debugging

Letzte Bearbeitung des Eintrags am 15.06.13 um 14:26:15
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

ERMN

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » E » **ERMN**
Syntax: ERMN
Typ: Funktion
Kategorie: Fehlerbehandlung

ERMN gibt einen ZSTRING PTR auf den Namen des Moduls zurück, in dem ein Fehler aufgetreten ist.

Beispiel:
Um das im Beispiel auftretende RESUME und ERFN verwenden zu können, compilieren Sie es mit der Kommandozeile

```
fbc -exx -lang deprecated ERMN_Test.bas
```

```
' ERMN_Test.bas

Declare Sub Generate_Error

Sub Generate_Error
On Error Goto Handler
Error 1000
Exit Sub

Handler:
Print "Error Function: "; *Erfn()
Print "Error Module : "; *Ermn()
Resume Next
End Sub

Generate_Error
```

Ausgabe:

```
Error Function: GENERATE_ERROR
Error Module  : ERMN_Test.bas
```

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC: existiert seit FreeBASIC v0.16

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht ERMN nicht zur Verfügung und kann nur über **__ERMN** aufgerufen werden.

Siehe auch:
ERFN, ERROR, ON ERROR, Fehlerbehandlung, Debugging

Letzte Bearbeitung des Eintrags am 19.01.13 um 15:52:49
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

ERR (Anweisung)

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » E » **ERR (Anweisung)**
Syntax: ERR = Ausdruck
Typ: Anweisung
Kategorie: Fehlerbehandlung

ERR kann wie eine globale Variable verwendet werden; ein beliebiger INTEGER-Wert kann dieser Variable zugewiesen werden. Durch diese Anwendungsweise ist es möglich, benutzerdefinierte Fehler zu erzeugen. Dies wäre zwar auch mit ERROR möglich, allerdings wird durch ERROR das Programm beendet, wenn keine Umleitung auf eine Fehlerbehandlungsroutine mittels ON ERROR durchgeführt wird. ERR ermöglicht also die Verwendung benutzerdefinierter Fehler ohne die Kommandozeilenoption -ex.
Es wird empfohlen, möglichst hohe Zahlen als Fehlernummern für benutzerdefinierte Fehler zu verwenden, da die unteren Werte bereits von FreeBASIC verwendet werden und eventuell eine Überschneidung verursacht wird. Siehe auch die FreeBASIC-Fehlernummern.

Unterschiede zu QB:
Die Fehlernummern sind in FreeBASIC nicht die gleichen wie in QB.

Siehe auch:
ERROR (Anweisung), ON ERROR, ERR (Funktion), OPEN ERR, Fehler-Behandlung in FreeBASIC, Übersicht: Fehlerbehandlung, Debugging

Letzte Bearbeitung des Eintrags am 25.08.10 um 23:11:48
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

ERR (Funktion)

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » E » **ERR (Funktion)**
Syntax: ERR
Typ: Funktion
Kategorie: Fehlerbehandlung

ERR gibt die Fehlernummer des zuletzt aufgetretenen Fehlers zurück. Wenn kein Fehler aufgetreten ist, gibt ERR 0 zurück.

Achtung: Der Befehl PRINT setzt ERR auf 0 zurück. Wollen Sie die Fehlernummer durch PRINT ausgeben, dann sollten Sie sie zuerst in einer Variablen speichern und diese ausgeben.

Beispiel:

```
OPEN "datei.ext" FOR INPUT AS "hlkw0">IF ERR = 2 THEN PRINT "Fehler:
Datei nicht gefunden!"
' ...
CLOSE "reflinkicon" href="temp0149.html">ERR (Anweisung), ERROR
(Anweisung), ON ERROR, OPEN ERR, ERL, Fehler-Behandlung in FreeBASIC,
Übersicht: Fehlerbehandlung, Debugging
```

Letzte Bearbeitung des Eintrags am 25.08.10 um 23:30:29
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

ERROR (Anweisung)

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » E » **ERROR (Anweisung)**
Syntax: ERROR Fehlernummer
Typ: Anweisung
Kategorie: Fehlerbehandlung

ERROR simuliert einen Fehler (runtime error). Der Befehl kann dazu verwendet werden, benutzerdefinierte Fehler zu erzeugen.

Wenn keine Fehlerbehandlung aktiv ist (siehe ON ERROR), wird das Programm beendet.

Es wird empfohlen, möglichst hohe Fehlernummern für benutzerdefinierte Fehler zu wählen, damit sich diese nicht mit den FreeBASIC-Fehlernummern überschneiden; beachten Sie bitte auch, dass in späteren Versionen von FreeBASIC evtl. neue Fehlernummern hinzukommen.

Beispiel:
Fehler 150 erzeugen (einer von vielen möglichen Fehlern...)

```
ERROR 150
```

Unterschiede zu QB:
Die Fehlernummern sind in FreeBASIC nicht die gleichen wie in QB.

Siehe auch:
"reflinkicon" href="temp0281.html">ON ERROR, ERR (Funktion), ERL, ERR (Anweisung), Fehler-Behandlung in FreeBASIC, Übersicht: Fehlerbehandlung, Debugging

Letzte Bearbeitung des Eintrags am 04.07.12 um 21:33:16
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

ERROR (Meta)

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » E » **ERROR (Meta)**
Syntax: #ERROR Nachricht
Typ: Metabefehl
Kategorie: Metabefehle

#ERROR unterbricht die Compilierung und gibt eine benutzerdefinierte Nachricht aus. 'Nachricht' ist eine beliebige Zeichenfolge, die als Fehlermeldung vom Compiler zurückgegeben werden soll.

Dieser Metabefehl ist zusammen mit #IF sinnvoll, damit der Compiliervorgang nur abgebrochen wird, wenn eine bestimmte Bedingung eintritt.

Beispiel:

```
"hlkw0">__FB_WIN32__
   "hlstring">"OpenAL32"
"hlkw0">DEFINED(__FB_LINUX__)
   "hlstring">"openal"
"hlkw0">"hlkw0">not supported
"reflinkicon" href="temp0151.html">ERROR (Anweisung), ASSERT (Meta),
Präprozessor-Anweisungen
```

Letzte Bearbeitung des Eintrags am 02.09.13 um 17:09:42
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

ESCAPE

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » E » **ESCAPE**
Syntax: OPTION ESCAPE
Typ: Schlüsselwort
Kategorie: Programmoptionen

OPTION ESCAPE aktiviert die Verwendung von Escape-Characters in Strings. Diese Option kann **nur bis FreeBASIC v0.16** eingesetzt werden, oder in entsprechend höheren Versionen, die mit der Kommandozeilenoption -lang deprecated compiliert wurden! Für den Einsatz von ESCAPE-Sequenzen in Versionen ab v0.17 mit der Option -lang fb siehe unter 'Ab FreeBASIC v0.17'.

Durch diese Option werden in FreeBASIC Backslashes ("\", CHR(92)) als Escape-Characters verwendet. Der Ausdruck hinter dem Escape-Character wird nicht 1:1 ausgegeben, sondern zuerst interpretiert. Folgt dem Character eine Zahl, so funktioniert der Character wie ein CHR; anstelle der Zahl wird das zugehörige ASCII-Zeichen ausgegeben. Die Zahl kann in jedem beliebigen Zählsystem angegeben werden, wenn das Präfix &? verwendet wird (z.B. &h für hexadezimal).

Außerdem werden folgende Strings besonders behandelt::

- \r wie ein CHR(13) (carriage-return)
- \n und \l wie ein CHR(10) (line-feed). \r\n ergibt also ein CRLF, die EDV-Version eines Zeilenumbruchs unter Windows.
- \a wie ein CHR(7) (Bell)
- \b wie ein CHR(8) (Backspace)
- \t wie ein CHR(9) (Tab)
- \v wie ein CHR(11) (vtab)
- \f wie ein CHR(12) (formfeed)
- \" wie ein CHR(34) (Doppeltes Anführungszeichen ")
- \' wie ein CHR(39) (Einfaches Anführungszeichen ')
- Alle anderen Zeichen hinter dem Escape-Char werden so interpretiert, wie sie im Ausdruck stehen. \\ wird also zu \. Beachten Sie dies, wenn Sie OPTION ESCAPE in Zusammenhang mit Pfadangaben verwenden!

Es ist möglich, einem String ein Dollarzeichen $ voranzustellen. In diesem Fall wird für diesen String die Interpretation der Escape-Zeichen deaktiviert. Dies ist natürlich nicht nötig, wenn OPTION ESCAPE nicht verwendet wird.

Beispiel:

```
"hlstring">"deprecated"
OPTION ESCAPE

"hlstring">"\r\n"

DIM AS STRING message

message = "\45 Das ist die \"erste\" Zeile.\r\nDas "
message &= "ist die \"zweite\" Zeile. \45"
PRINT message

PRINT CRLF & "^-- CRLF --V" & CRLF;

PRINT "Der Pfad lautet \34" $"C:\FREEBASIC\33a" "\34"
```

```
SLEEP
```

Ausgabe:

```
- Das ist die "erste" Zeile.
Das ist die "zweite" Zeile. -

^-- CRLF --V
Der Pfad lautet "C:\FREEBASIC\33a
```

Ab FreeBASIC v0.17

Da mit FreeBASIC v0.17 die Verwendung von OPTION-Standards nicht mehr zulässig ist, müssen Strings, die Escape-Sequenzen enthalten, jetzt explizit markiert werden. Dies geschieht dadurch, dass ihnen ein Ausrufezeichen vorangestellt wird:

```
!"String, der Escape-Sequenzen enthält"
```

Das oben gezeigte Beispiel muss also folgendermaßen lauten, wenn in der Dialektform -lang fb (Standard) compiliert wird:

```
"hlstring">"\r\n"

DIM AS STRING message

message = !"\45 this is the \"first\" line\r\nthis "
message &= !"is the \"second\" line \45"
PRINT message

PRINT CRLF & !"^-- CRLF --V" & CRLF;

PRINT !"The Path is \34" "C:\FREEBASIC\33a" !"\34"
SLEEP
```

Die Markierung von Strings, die nicht geparsed werden sollen, mit einem Dollarzeichen $ ist also nicht mehr nötig; aus Kompatibilitätsgründen ist sie aber immer noch erlaubt. Die letzte Zeile dieses Beispiels dürfte also ebenso lauten:

```
PRINT !"The Path is \34" $"C:\FREEBASIC\33a" !"\34"
```

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC:

- Seit FreeBASIC v0.17 muss mit der Kommandozeilenoption -lang deprecated compiliert werden, um OPTION ESCAPE einsetzen zu können. Soll stattdessen nach '-lang fb' übersetzt werden, so werden Strings, die Escape-Sequenzen enthalten, mit einem Ausrufezeichen markiert: !"String, der Escape-Sequenzen enthält".
- Seit FreeBASIC v0.16 ist es möglich, Strings ein Dollarzeichen $ voranzustellen, um die Interpretation der Escape-Chars zu unterdrücken.
- Seit FreeBASIC v0.14 werden Escape-Chars in INCLUDE- und INCLIB-Anweisungen nicht mehr interpretiert.

Siehe auch:

OPTION, Ausrufezeichen, Dollarzeichen, __FB_OPTION_ESCAPE__, Verschiedenes

Letzte Bearbeitung des Eintrags am 19.01.13 um 16:39:55
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

EXEC

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » E » **EXEC**
Syntax: EXEC (Datei, Argumente)
Typ: Funktion
Kategorie: System

EXEC startet eine ausführbare Datei mit den übergebenen Argumenten. Nachdem das aufgerufene Programm beendet wurde, erhält das Originalprogramm die Kontrolle zurück.

- 'Datei' ist ein STRING, der den vollen Dateinamen (inklusive Erweiterung) der auszuführenden Datei enthält.
- 'Argumente' ist ein STRING, der die Argumente enthält, die an das Programm übergeben werden sollen. Wenn keine Argumente übergeben werden sollen, können Sie einen Leerstring "" angeben.
- Der Rückgabewert ist der vom Programm gesetzte Errorlevel (siehe END). Konnte kein Programm aufgerufen werden (z. B. weil die angegebene Datei nicht existiert), so wird -1 zurückgegeben. Beachten Sie, dass auch der vom Programm zurückgegebene Wert -1 sein könnte!

Beispiel:

```
"hlkw0">__FB_UNIX__
 Dim As String program = "./program"
"hlkw0">__FB_PCOS__
 Dim As String program = "program.exe"
"hlkw0">Print "Starte " & program & " ..."
dim as integer ret = exec(program, "")
Select Case ret
  Case 0    : Print "Das Programm wurde korrekt ausgeführt."
  Case -1   : Print program & " nicht gefunden!"
  Case Else : Print program & " beendete mit Errorlevel " & ret
End Select

Sleep
```

Unterschiede zu QB: neu in FreeBASIC

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht EXEC nicht zur Verfügung und kann nur über **__EXEC** aufgerufen werden.

Siehe auch:
RUN, CHAIN, SHELL, END, COMMAND, Betriebssystem-Anweisungen

Letzte Bearbeitung des Eintrags am 27.12.12 um 00:32:35
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

EXEPATH

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » E » **EXEPATH**
Syntax: EXEPATH[$]
Typ: Funktion
Kategorie: System

EXEPATH gibt das Verzeichnis zurück, in dem sich das gerade ausgeführte Programm befindet. Dieses ist *nicht* unbedingt identisch mit dem aktuellen Arbeitsverzeichnis (siehe CURDIR).

Das Dollarzeichen ($) als Suffix ist optional.

Beispiel:

```
CHDIR "C:"    ' nur sinnvoll unter Windows und DOS
PRINT "Das Programm befindet sich in: ";
PRINT EXEPATH
PRINT "Das aktuelle Arbeitsverzeichnis ist: ";
PRINT CURDIR
SLEEP
```

Unterschiede zu QB: neu in FreeBASIC

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht EXEPATH nicht zur Verfügung und kann nur über **__EXEPATH** aufgerufen werden.

Siehe auch: CURDIR, CHDIR, MKDIR, RMDIR, SHELL, OPEN, DIR, Betriebssystem-Anweisungen

Letzte Bearbeitung des Eintrags am 27.12.12 um 00:33:26
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

EXIT

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » E » **EXIT**
Syntax: EXIT { SUB | FUNCTION | CONSTRUCTOR | DESTRUCTOR | OPERATOR | PROPERTY | DO | FOR | WHILE | SELECT }
Typ: Anweisung
Kategorie: Programmablauf

EXIT verlässt einen Codeblock wie SUB, FUNCTION, OPERATOR, PROPERTY, CONSTRUCTOR, DESTRUCTOR oder Schleifen wie DO ... LOOP, FOR ... NEXT, WHILE ... WEND oder die Bedingungsstruktur SELECT CASE. Die Programmausführung wird am Ende der entsprechenden Blockstruktur fortgesetzt.

Beispiel 1: Der PRINT-Befehl wird nie ausgeführt.

```
Do
  Exit Do
  Print "Wird nie ausgeführt"
Loop
Sleep
```

Wenn mehrere Blöcke ineinander verschachtelt sind, dann wird der innerste Block der angegebenen Art verlassen. Durch die mehrfache Angabe von Blocktypen, durch Komma getrennt, kann auch ein weiter außen liegender Block angesprochen werden.

Beispiel 2: Verlassen mehrerer Schleifen gleichzeitig

```
Dim As Integer i, j
For i = 1 To 10
  For j = 1 To 10
    Exit For, For
  Next
  Print "Wird nie ausgeführt"
Next
Sleep
```

Unterschiede zu QB:

- Unter QB sind nur EXIT { DEF | DO | FOR | FUNCTION | SUB } möglich.
- EXIT DEF existiert unter FreeBASIC nicht mehr.

Unterschiede zu früheren Versionen von FreeBASIC:

- EXIT OPERATOR und EXIT PROPERTY sind seit FreeBASIC v0.17 möglich.
- Die Möglichkeit, mehrere Codeblock-Ebenen gleichzeitig zu verlassen (z. B. EXIT FOR, FOR) existiert seit FreeBASIC v0.17.
- EXIT SELECT und EXIT WHILE sind seit FreeBASIC v0.16 möglich.

Siehe auch:
SUB, FUNCTION, OPERATOR, PROPERTY, IF ... THEN, SELECT CASE, DO ... LOOP, WHILE ... WEND, FOR ... NEXT, CONTINUE, Programmablauf

Letzte Bearbeitung des Eintrags am 15.06.13 um 15:50:50
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

EXP

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » E » **EXP**
Syntax: EXP (Zahl)
Typ: Funktion
Kategorie: Mathematik

EXP gibt die Potenz einer angegebenen Zahl zur eulerschen Zahl e zurück. e ist näherungsweise 2.718281828459045.

- 'Zahl' ist eine beliebiger numerischer Ausdruck. Variablen, Konstanten, Operatoren und Funktionen sind erlaubt. Der Ausdruck darf von jeden Datentyp außer STRING, ZSTRING oder WSTRING sein.
- Der Rückgabewert ist ein DOUBLE, der das Ergebnis des Ausdrucks e^Zahl darstellt. Daraus folgt, dass EXP(1)=e ist.

Die Umkehrfunktion zu EXP ist LOG.

EXP kann mithilfe von OPERATOR überladen werden.

Beispiel:
Berechnung des Kapitals mit Zinseszins

```
"hlstring">"vbcompat.bi"
DIM r AS DOUBLE
DIM p AS DOUBLE
DIM t AS DOUBLE
DIM a AS DOUBLE

INPUT "Ursprünglich investiertes Kapital: ", p
INPUT "Jährliche Verzinsung (als Dezimale): ", r
INPUT "Verzinsungszeitraum in Jahren: ", t

a = p * Exp ( log(1+r) *t )
PRINT ""
PRINT "Nach"; t; " Jahren wird bei einer Verzinsung"
PRINT "von"; r * 100; "% ihr Ausgangskapital von"; p; "$ "
PRINT format(a, ""); "$ wert sein."
SLEEP
```

Ausgabebeispiel:

```
Ursprünglich investiertes Kapital: 100
Jährliche Verzinsung (als Dezimale): .08
Verzinsungszeitraum in Jahren: 20

Nach 20 Jahren wird bei einer Verzinsung
von 8% ihr Ausgangskapital von 100$
495.30$ Wert sein.
```

Hinweis: Die Einbindung von vbcompat.bi wird im obigen Beispiel nur für den Befehl FORMAT benötigt. Einfacher wäre es gewesen, in der Berechnung zu schreiben:

```
a = p * (1+r)^t
```

Dies wird intern jedoch ebenfalls zu a = p * EXP (LOG(1+r) *t) umgewandelt.

Unterschiede zu früheren Versionen von FreeBASIC:
Die Überladung von EXP für benutzerdefinierte Datentypen ist seit FreeBASIC v0.22 möglich.

Siehe auch:
LOG, Exponent ^, mathematische Funktionen

Letzte Bearbeitung des Eintrags am 17.08.12 um 23:17:40
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

EXPLICIT

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » E » **EXPLICIT**
Syntax: OPTION EXPLICIT
Typ: Schlüsselwort
Kategorie: Programmoptionen

Durch OPTION EXPLICIT müssen alle Variablen durch DIM, COMMON, STATIC oder in einem Prozedur-Header definiert werden; dadurch kann verhindert werden, dass versehentlich falsche Datentypen verwendet werden oder dass unerwünschte Effekte durch Tippfehler auftreten.

OPTION EXPLICIT kann **nur bis FreeBASIC v0.16** eingesetzt werden, oder in entsprechend höheren Versionen, die mit der Kommandozeilenoption -lang deprecated oder -lang fblite compiliert wurden! Wird ab FreeBASIC v0.17 unter der Option -lang fb compiliert, so ist OPTION EXPLICIT nicht mehr zulässig! Stattdessen müssen ab dieser Version alle Variablen immer explizit deklariert werden; eine implizite Deklarierung ist verboten.

Bei FreeBASIC v0.16 bzw. in der Dialektform -lang fb ist diese Funktion normalerweise deaktiviert. OPTION EXPLICIT kann auch in der Mitte des Codes aktiviert werden. Im Codeteil vor dem OPTION-EXPLICIT-Aufruf ist dann eine 'implizite Deklaration', also eine Verwendung von Variablen ohne vorherige Definition durch DIM, COMMON oder STATIC, erlaubt.

EXPLICIT wird auch im Zusammenhang mit ENUM verwendet.

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC:
Die Option ist nur bis FreeBASIC v0.16 erlaubt. Seit FreeBASIC v0.17 ist diese Option in der Dialektform -lang fb nicht mehr nötig.

Siehe auch:
DIM, COMMON, STATIC, OPTION, __FB_OPTION_EXPLICIT__, ENUM, Verschiedenes

Letzte Bearbeitung des Eintrags am 19.01.13 um 16:40:34
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

EXPORT

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » E » **EXPORT**
Syntax: {SUB | FUNCTION} foo (Argumente) [AS Datentyp] EXPORT
Typ: Klausel
Kategorie: Bibliotheken

Wenn eine SUB oder FUNCTION in einer dll/so mit der EXPORT-Klausel deklariert wurde, wird sie zur PUBLIC-EXPORT-Tabelle hinzugefügt, sodass sie von externen Programmen aus mit DYLIBSYMBOL verlinkt werden kann.

Beispiel: mydll.bas

```
' compiliert mit: fbc -dll mydll.bas (erstellt mydll.dll und
libmydll.dll.a unter Win32,
'                                    oder libmydll.so unter Linux)
DECLARE FUNCTION AddNumbers ALIAS "AddNumbers" ( _
   BYVAL operand1 AS INTEGER, BYVAL operand2 AS INTEGER ) _
   AS INTEGER

FUNCTION AddNumbers  ( _
    BYVAL operand1 AS INTEGER, BYVAL operand2 AS INTEGER ) _
    AS INTEGER EXPORT
  AddNumbers = operand1 + operand2
END FUNCTION
```

Beispiel: Testmydll.bas

```
'Hauptprogramm
' erstelle einen Funktionspointer
' Argumente sind dieselben wie in der DLL
DIM AddNumbers AS FUNCTION ( _
   BYVAL operand1 AS INTEGER, BYVAL operand2 AS INTEGER ) _
   AS INTEGER
DIM hndl AS ANY PTR
hndl = DYLIBLOAD("mydll.dll")
' Adresse des Programms finden
' (Groß-/Kleinschreibung beachten!)
AddNumbers = dylibsymbol( hndl, "AddNumbers" )
' Aufrufen
PRINT "1 + 2 ="; AddNumbers( 1, 2 )
DYLIBFREE  hndl
SLEEP
```

Wichtig: Wenn Sie die ALIAS-Klausel nicht benutzen, verwendet FreeBASIC den von Ihnen angegebenen Programmnamen in Großbuchstaben.

Unterschiede zu QB: neu in FreeBASIC

Plattformbedingte Unterschiede: wird unter DOS nicht unterstützt

Unterschiede zu früheren Versionen von FreeBASIC: existiert seit FreeBASIC v0.11

Unterschiede unter den FB-Dialektformen:

In der Dialektform -lang qb steht EXPORT nicht zur Verfügung und kann nur über **__EXPORT** aufgerufen werden.

Siehe auch:
DECLARE, DYLIBLOAD, DYLIBSYMBOL,DYLIBFREE,
Module (Library / DLL)

Letzte Bearbeitung des Eintrags am 19.01.13 um 16:41:05
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

EXTENDS

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » E » **EXTENDS**
Syntax: TYPE kindKlasse EXTENDS basisKlasse
Typ: Anweisung
Kategorie: Klassen

EXTENDS wird in Verbindung mit Vererbung bei TYPE und UNION verwendet. Dabei gibt das Schlüsselwort an, dass die eigene Klasse die Records und Methoden einer bereits bestehenden erbt, also erhält, ohne dass man diese zusätzlich angeben müsste.

Beispiel:

```
Type Haustier Extends Object
  As Integer beine = 4
  As Integer schwanz = 1
End Type

Type Hund Extends Haustier
  As Integer anhaenglich = 1
  Declare Sub gibLaut
End Type
Sub Hund.gibLaut
  Print "Wuff!"
End Sub

Type Chihuahua Extends Hund
  Declare Sub gibLaut
End Type
Sub Chihuahua.gibLaut
  Print "Klaeffklaeff!"
End Sub

Type Bernhardiner Extends Hund
  As Integer gutmuetig = 1
End Type

'"hlkw0">Dim benno As Bernhardiner, husky As Chihuahua
Print "Benno hat " & benno.beine & " Beine und " & benno.schwanz & "
Schwanz."
Print "Benno ist gutmuetig (Wert " & benno.gutmuetig & ")."
benno.gibLaut
Print
Print "Husky hat " & husky.beine & " Beine und " & husky.schwanz & "
Schwanz."
husky.gibLaut
Print

Sleep
```

Ausgabe:

```
Benno hat 4 Beine und 1 Schwanz.
Benno ist gutmuetig (Wert 1).
```

```
Wuff!

Husky hat 4 Beine und 1 Schwanz.
Klaeffklaeff!
```

Erläuterung: Sowohl Benno als auch Husky erben die Records 'beine' und 'schwanz' vom Elternobjekt. 'gutmuetig' steht nur Benno zur Verfügung. Dafür besitzt Husky eine eigene 'gibLaut'-Methode, während Benno auf die Methode des Elternobjekts zugreift.

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC: existiert seit FreeBASIC v0.24

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht EXTENDS nicht zur Verfügung und kann nur über **__EXTENDS** aufgerufen werden.

Siehe auch:
TYPE (UDT), UNION, BASE, OBJECT, IS (Vererbung), Objektorientierung

Letzte Bearbeitung des Eintrags am 14.09.13 um 14:51:07
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

EXTERN (Module)

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » E » **EXTERN (Module)**
Syntax: EXTERN [IMPORT] Bezeichner [ALIAS "Aliasname"] [AS Typ]
Typ: Anweisung
Kategorie: Bibliotheken

EXTERN wird benutzt, um auf externe Variablen zuzugreifen, die in anderen Modulen oder DLLs deklariert sind. Im Gegensatz zu COMMON wird kein Speicherplatz reserviert, die Variable wird lediglich definiert und zeigt auf den Speicherbereich des anderen Moduls. Die Anweisung wurde hinzugefügt, um die C-Bibliotheken wie Allegro und DirectX zu unterstützen.

- 'IMPORT' muss verwendet werden, wenn auf Win32-DLLs zugegriffen wird, da hier Symbole auf eine andere Art verwaltet werden.
- 'Bezeichner' ist der Name, unter dem eine globale Variable der DLL oder des Moduls angesprochen werden soll.
- 'Aliasname' ist der Name, den die Variable im externen Programm trägt.
- 'Typ' ist ein beliebiger Datentyp wie z.B. INTEGER oder ein UDT.

Eine Variable kann mehrmals EXTERN deklariert werden, sofern bei dieser Deklaration derselbe Datentyp angegeben wird.

Eine als EXTERN definierte Variable benötigt kein SHARED (die Syntax erlaubt dies auch nicht), da diese Variablen automatisch global im Modul zur Verfügung stehen.

Beispiel:

```
' Modul1.bas
EXTERN foo ALIAS "foo" AS INTEGER

SUB SetFoo
  foo = 1234
END SUB

'------'

' Modul2.bas
DECLARE SUB SetFoo
EXTERN Foo ALIAS "foo" AS INTEGER
DIM foo AS INTEGER = 0

SetFoo
PRINT Foo
SLEEP
```

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC:

- Seit FreeBASIC v0.15 sind doppelte EXTERN-Deklarationen zulässig, sofern beide Male derselbe Datentyp angegeben wird.
- EXTERN existiert seit FreeBASIC v0.11.

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht EXTERN nicht zur Verfügung und kann nur über **__EXTERN** aufgerufen werden.

Siehe auch:
EXTERN ... END EXTERN, COMMON, DIM, REDIM, SHARED, ALIAS, Module (Library / DLL), Gültigkeitsbereich von Variablen

Letzte Bearbeitung des Eintrags am 27.12.12 um 00:35:14
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

EXTERN ... END EXTERN

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » E » **EXTERN ... END EXTERN**
Syntax:

```
EXTERN { "C" | "C++" | "Windows" | "Windows-MS" } [ LIB "LibName" ]
   ' Deklarationen
END EXTERN
```

Typ: Anweisung
Kategorie: Bibliotheken

'EXTERN "Mangling"' startet einen Block, innerhalb dessen alle Deklarationen andere interne Bezeichner erhalten, als sie von FreeBASIC bekommen würden. Dies hat für den Programmfluss keine direkten Auswirkungen. Bei der Rückgabe von Fehlermeldungen oder beim Debuggen mittels externer Programme kann dies allerdings von Vorteil sein.

- EXTERN "C": setzt alle Prozeduren auf CDECL und erhält die Groß-/Kleinschreibung von allen Namen. Dasselbe Verhalten erreicht man ohne EXTERN durch die Aufrufkonvention CDECL zusammen mit einem ALIAS-String, der exakt denselben Prozedurnamen enthält.
- EXTERN "C++": wie EXTERN "C", stellt die Namen zusätzlich aber auf die Konventionen von *g++-4.x*
- EXTERN "Windows": setzt alle Prozeduren auf STDCALL, erhält die Groß-/Kleinschreibung von allen Namen und setzt zusätzlich das Suffix *"@N"* an alle Prozedurnamen, wobei *N* die Größe aller Parameter der jeweiligen Prozedur in Bytes ist.
- EXTERN "Windows-MS": wie EXTERN "Windows", nur dass das Suffix unter DOS/Windows nicht angehängt wird.

LIB "LibName" kann verwendet werden, um eine Lib einzubinden. Zusätzlich werden alle Prozedurdeklarationen innerhalb des Blocks so behandelt, als wäre bei der Deklaration ein *'LIB "LibName"'* angegeben worden. Das kann aber durch explizite Angabe von LIB bei der Deklaration überschrieben werden.

Im EXTERN...END EXTERN-Block sind nur Deklarationen zulässig, sogenannte "ausführbare Anweisungen" dürfen innerhalb eines Blocks nicht auftreten. Im Detail sind die erlaubten Anweisungen:

- DIM
- STATIC
- COMMON
- DECLARE
- TYPE
- UNION
- ENUM
- CONST
- VAR
- NAMESPACE

Der Vorteil einer solchen Behandlung ist, dass der Linker einen leichter lesbaren Namen zurückgibt, wenn eine Referenz nicht gefunden werden konnte.

Als Block-Anweisung erstellt EXTERN einen eigenen SCOPE-Block.

Beispiel:

```
EXTERN "C++"
  ' Auch innerhalb des Namespace gilt die C++-Behandlung
  NAMESPACE Ns1
    DECLARE FUNCTION theFunction( BYVAL AS INTEGER ) AS UINTEGER
  END NAMESPACE
END EXTERN

' Innerhalb dieses Namespace gilt die FreeBASIC-Behandlung
NAMESPACE Ns2
  DECLARE FUNCTION theFunction( BYVAL AS INTEGER ) AS UINTEGER
END NAMESPACE

' Beide Funktionen existieren nicht; für beide wird also
' eine Fehlermeldung ausgegeben.
' Hier wird allerdings demonstriert, wie GNU C++ und
' FreeBASIC die Bezeichner intern handhaben; die
' Fehlermeldung EXTERN C++-Funktion ist wesentlich
' einfacher zu lesen:

Print Ns1.theFunction( 1 )
Print Ns2.theFunction( 1 )
```

Ausgabe:
Unter Windows erhält man in etwa folgende Fehlermeldung:

```
test.o:fake:(.text+0x54): undefined reference to `Ns1::theFunction(int)'
test.o:fake:(.text+0x68): undefined reference to
`_ZN3NS211THEFUNCTIONEi@4
```

Dagegen sieht die Fehlermeldung unter Linux im zweiten Fall freundlicher aus:

```
test.o: In function `main':
(.text+0x23): undefined reference to `Ns1::theFunction(int)'
test.o: In function `main':
(.text+0x3a): undefined reference to `NS2::THEFUNCTION(int)'
```

Unterschiede zu QB: neu in FreeBASIC

Plattformbedingte Unterschiede:
Unter Linux hängt EXTERN "Windows" nie das Suffix *"@N"* an

Unterschiede zu früheren Versionen von FreeBASIC:

- Die Angabe "Windows-MS" existiert seit FreeBASIC v0.18.2
- EXTERN als Blockanweisung existiert seit FreeBASIC v0.16

Unterschiede unter den FB-Dialektformen: nur zulässig in der Dialektform -lang fb

Siehe auch:
EXTERN (Module), NAMESPACE, DECLARE, CDECL, STDCALL, Module (Library / DLL)

Letzte Bearbeitung des Eintrags am 27.12.12 um 00:35:42
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

FIELD

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » F » **FIELD**
Syntax: TYPE UDTName FIELD = { 1 | 2 | 4 }
Typ: Schlüsselwort
Kategorie: Speicher

FIELD wird zusammen mit TYPE und UNION verwendet und legt dort das Padding-Verhalten fest. Dieses bestimmt, dass UDTs auf eine bestimmte Länge "ausgedehnt" werden sollen. Der Zweck liegt darin, dass auf solche ausgedehnten Felder z. T. schneller zugegriffen werden kann. Das Feld wird dabei so ausgedehnt, dass die Anzahl der Bytes, die das Feld im Speicher benötigt, ein ganzzahliges Vielfaches von 1, 2 oder 4 ist.

FIELD hat nur eine Auswirkung, wenn im UDT Records verwendet werden, die mindestens die angegebene Größe besitzen. Ansonsten wird auf ein Vielfaches des größten Record-Datentyps ausgeweitet. Wird FIELD ausgelassen, verwendet FreeBASIC unter Linux und DOS den Wert 4 und unter Windows den Wert 8.

Beispiel:

```
Type NormalPacked
  i As Integer
  b As Byte
End Type

Type BytePacked Field = 1
  i As Integer
  b As Byte
End Type

Type WordPacked Field = 2
  i As Integer
  b As Byte
End Type

Type DWordPacked Field = 4
  i As Integer
  b As Byte
End Type

' Im Folgenden hat FIELD keinen Effekt
Type ShortType Field = 4
  s1 As Short
  s2 As Short
  b  As Byte
End Type

Print "Groesse der einzelnen Datentypen"
Print "Integer     : " & SizeOf(     Integer)
Print "Byte        : " & SizeOf(        Byte)
Print

Print "Groesse der verschiedenen UDTs"
Print "Ohne Field  : " & SizeOf(NormalPacked)
Print "Field = 1   : " & SizeOf(  BytePacked)
Print "Field = 2   : " & SizeOf(  WordPacked)
```

```
Print "Field = 4    : " & SizeOf( DWordPacked)
Print "Ohne Integer: " & SizeOf( ShortType)
Sleep
```

Ausgabe:

```
Groesse der einzelnen Datentypen
Integer      : 4
Byte         : 1

Groesse der verschiedenen UDTs
Ohne Field   : 8
Field = 1    : 5
Field = 2    : 6
Field = 4    : 8
Ohne Integer: 6
```

Unterschiede zu QB:

- QB unterstützt kein Padding. Die Feldbreite ist immer 1. Beachten Sie dies, wenn Sie Dateien, die von QB generiert wurden in FreeBASIC einlesen.
- Unter QB wird FIELD verwendet, um die Puffergröße bei RANDOM-Files festzulegen. Diese Verwendung ist unter FreeBASIC nicht mehr möglich.

Plattformbedingte Unterschiede:

- Das Standardpadding unter Linux und DOS ist 4 Byte.
- Das Standardpadding unter Windows ist 8 Byte.

Unterschiede unter den FB-Dialektformen:
Das Standardpadding in -lang fb und -lang fblite hängt von der Plattform ab, während in -lang qb kein Padding durchgeführt wird.

Siehe auch:
TYPE (UDT), Datentypen und Deklarationen

Letzte Bearbeitung des Eintrags am 25.12.12 um 18:04:28
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

FILEATTR

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » F » **FILEATTR**
Syntax: FILEATTR (Dateinummer, [Ausgabe])
Typ: Funktion
Kategorie: Dateien

FILEATTR gibt Informationen über eine mit OPEN geöffnete Datei zurück.

Anmerkung:
Um diese Funktion nutzen zu können, muss die Datei file.bi in Ihren Quellcode eingebunden werden, z.B. mit INCLUDE. Alternativ können Sie auch die Datei vbcompat.bi einbinden, da diese die file.bi automatisch in Ihr Programm lädt.

- 'Dateinummer' ist die Nummer, die beim Öffnen mit OPEN zugewiesen wurde.
- 'Ausgabe' ist eine beliebige Zahl, die angibt, welche Information über die Datei zurückgegeben werden soll. Weiter unten ist aufgelistet, welcher Wert welche Information zurück gibt. Wird dieser Parameter ausgelassen, nimmt FreeBASIC automatisch 1 an.
- Der Rückgabewert ist eine INTEGER-Zahl, die von 'Ausgabe' abhängig ist und die Eigenschaften einer Datei beschreibt. Kann eine Information nicht abgefragt werden, z.B. weil die angegebene Dateinummer nicht belegt ist, so ist der Rückgabewert 0.

Abhängig von 'Ausgabe' werden verschiedene Werte ausgegeben. Um den Code besser lesbar zu gestalten, sind in der Datei 'file.bi' Symbole definiert worden, welche die Information hinter den Zahlenwerten in Worten wiedergeben. Folgende Werte können angegeben werden bzw. werden ausgegeben:

Wert für 'Ausgabe'	Symbol für 'Ausgabe'	Zurückgegebene Information	Rückgabewerte	Symbole für Rückgabewerte
1	fbFileAttrMode	Dateimodus (z.B. BINARY), in dem 'Dateinummer' geöffnet wurde	1 2 4 8 32	fbFileModeInput fbFileModeOutput fbFileModeRandom fbFileModeAppend fbFileModeBinary
2	fbFileAttrHandle	Der Handle, über den das System auf die Datei zugreift. Es ist derselbe Handle, den die C Runtime-Library verwendet. Nur unter Windows: Wenn an dieser Stelle der Handle eines geöffneten COM-Geräts abgefragt wird, gibt FreeBASIC das Ergebnis der API CreateFile() zurück. Der Rückgabewert für den Handle eines geöffneten LPT-Geräts ist das Ergebnis der API OpenPrinter(). Diese Handle-Werte können an andere Windows-API-Funktionen übergeben werden. Nur unter Linux: Der Rückgabewert für ein geöffnetes	- entfällt -	- entfällt -

		COM-Gerät ist der Datenbezeichner, der von OPEN() zurückgegeben wird.		
3	fbFileAttrEncoding	Die Codierungsvariante (z.B. ASCII, UTF-8), mit der die Datei behandelt wird. Siehe auch ENCODING.	0 1 2 3	fbFileEncodASCII fbFileEncodUTF8 fbFileEncodUTF16 fbFileEncodUTF32

Beispiel 1:

```
"hlstring">"file.bi"

DIM f AS INTEGER
f = FREEFILE

OPEN "C:\BASIC\FreeBASIC\fbc.exe" FOR INPUT AS "hlkw0">SELECT CASE
FILEATTR(f, fbFileAttrMode)
CASE fbFileModeInput
   PRINT "Die Datei wurde im INPUT-Modus geoeffnet"
CASE fbFileModeOutput
   PRINT "Die Datei wurde im OUTPUT-Modus geoeffnet"
CASE fbFileModeAppend
   PRINT "Die Datei wurde im APPEND-Modus geoeffnet"
CASE fbFileModeRandom
   PRINT "Die Datei wurde im RANDOM-Modus geoeffnet"
CASE fbFileModeBinary
   PRINT "Die Datei wurde im BINARY-Modus geoeffnet"
END SELECT

PRINT "Das System-Handle zur Datei ist"; FILEATTR(f, fbFileAttrHandle)

SELECT CASE FILEATTR(f, fbFileAttrEncoding)
CASE fbFileEncodASCII
   PRINT "Die Datei ist als ASCII-Datei geoeffnet"
CASE fbFileEncodUTF8
   PRINT "Die Datei ist als UTF-8-Datei geoeffnet"
CASE fbFileEncodUTF16
   PRINT "Die Datei ist als UTF-16-Datei geoeffnet"
CASE fbFileEncodUTF32
   PRINT "Die Datei ist als UTF-32-Datei geoeffnet"
END SELECT

CLOSE "hlkw0">SLEEP
```

Ausgabe:

```
Die Datei wurde im INPUT-Modus geoeffnet
Das System-Handle zur Datei ist 2009267424
Die Datei ist als ASCII-Datei geoeffnet
```

Beispiel 2:

```
"hlstring">"vbcompat.bi"
"hlstring">"crt.bi"

DIM f AS FILE PTR, i AS INTEGER
```

```
' Öffne eine Datei und schreibe einige Zeilen hinein
OPEN "test.txt" FOR OUTPUT AS "hlzeichen">= CAST( FILE PTR, FILEATTR( 1,
fbFileAttrHandle ))
FOR i = 1 TO 10
  fprintf( f, !"Line %i\n", i )
NEXT i
CLOSE "hlkommentar">' Öffne die Datei erneut, und lese den Text aus
dieser

OPEN "test.txt" FOR INPUT AS "hlzeichen">= CAST( FILE PTR, FILEATTR( 1,
fbFileAttrHandle ))
WHILE feof(f) = 0
  i = fgetc(f)
  PRINT CHR(i);
WEND
CLOSE "reflinkicon" href="temp0283.html">OPEN, ENCODING, Dateien (Files),
Betriebssystem-Anweisungen
```

Letzte Bearbeitung des Eintrags am 27.12.12 um 00:46:39

FILECOPY

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » F » **FILECOPY**
Syntax: FILECOPY (Quelle, Ziel)
Typ: Funktion
Kategorie: Dateien

FILECOPY kopiert die Datei 'Quelle' nach 'Ziel'. Diese Funktion entspricht der VB-Funktion FILECOPY.

Anmerkung:
Um diese Funktion nutzen zu können, muss die Datei file.bi in Ihren Quellcode eingebunden werden, z. B. mit INCLUDE. Alternativ können Sie auch die Datei vbcompat.bi einbinden, da diese die file.bi automatisch in Ihr Programm lädt.

- 'Quelle' ist ein STRING, der den Dateinamen der zu kopierenden Datei enthält. Diese Angabe darf eine Pfadangabe enthalten. Wird kein Pfad angegeben, so nimmt FreeBASIC an, dass sich die Datei im aktuellen Arbeitsverzeichnis befindet (siehe CURDIR).
- 'Ziel' ist ein STRING, der den Dateinamen enthält, den die Kopie erhalten soll. Diese Angabe darf eine Pfadangabe enthalten. Wird kein Pfad angegeben, so nimmt FreeBASIC an, dass sich die Datei im aktuellen Arbeitsverzeichnis befindet. Sind 'Quelle' und 'Ziel' identisch, so wird nichts kopiert.

Der Rückgabewert ist entweder 0, wenn der Kopiervorgang erfolgreich abgeschlossen wurde, oder 1, wenn ein Fehler aufgetreten ist. Mögliche Fehler sind:

- Die Datei in 'Quelle' existiert nicht.
- 'Ziel' und 'Quelle' sind identisch.
- Als 'Quelle' oder 'Ziel' wird ein unzulässiger Dateiname angegeben. Ein unzulässiger Dateiname enthält Zeichen wie \/|:<>*? oder ".
- Die Pfadangabe in 'Quelle' oder 'Ziel' existiert nicht.

Achtung: Existierende Dateien werden ohne Nachfrage überschrieben!

Beispiel:
Achtung: Dieses Beispiel erstellt selbständig die Dateien 'Src.txt" und 'Dst.txt' in 'C:\TMP\' sowie 'Dst.txt' in 'C:\'. Diese Dateien werden nach Programmende automatisch gelöscht. Passen Sie das Programm an, wenn sich in Ihrem Dateisystem in diesem Verzeichnis schon Daten mit den angegebenen Namen befinden, da diese sonst überschrieben werden. Wenn in Ihrem Dateisystem ein Ordner 'C:\TMP\' existiert, dieser aber keine Dateien 'Src.txt' oder 'Dst.txt' enthält, können Sie dieses Beispiel ohne Problem ausführen, da andere Daten nicht beeinflusst werden. Das Verzeichnis 'C:\TMP\' wird automatisch erstellt, wenn es noch nicht existiert, und automatisch aus dem System entfernt, wenn sich keine Daten darin befinden.

```
"hlstring">"file.bi"  ' Die Dateifunktionen aktivieren
CHDIR "C:\"  ' In das Verzeichnis C:\ wechseln
MKDIR "TMP"  ' Ein temporäres Verzeichnis anlegen

' Eine Datei erstellen und...
OPEN "TMP\Src.txt" FOR OUTPUT AS "hlkommentar">' ...mit Text befüllen:
PRINT "hlzeichen">, "Source-File"
CLOSE "hlkw0">IF FILECOPY("TMP\Src.txt", "TMP\Dst.txt") THEN
   PRINT "Fehler beim Kopieren von TMP\Src.txt nach TMP\Dst.txt."
ELSE
   PRINT "C:\TMP\Src.txt nach C:\TMP\Dst.txt kopiert."
END IF
```

```
IF FILECOPY("C:\TMP\Src.txt", "Dst.txt") THEN
   PRINT "Fehler beim Kopieren von C:\TMP\Src.txt nach C:\Dst.txt."
ELSE
   PRINT "C:\TMP\Src.txt nach C:\Dst.txt kopiert."
END IF

IF FILECOPY("C:\TMP\*.txt", "TMP\Dst.txt") THEN
   PRINT "Fehler beim Kopieren von C:\TMP\*.txt nach C:\TMP\Dst.txt."
ELSE
   PRINT "C:\TMP\*.txt nach C:\TMP\Dst.txt kopiert."
END IF

IF FILECOPY("C:\TMP\Src.txt", "C:\TMP\Src.txt") THEN
   PRINT "Fehler beim Kopieren von C:\TMP\Src.txt nach C:\TMP\Src.txt."
ELSE
   PRINT "C:\TMP\Src.txt nach C:\TMP\Src.txt kopiert."
END IF

IF FILECOPY("C:\ThisFolderDoesSurelyNotExist\Src.txt", "C:\TMP\Src.txt")
THEN
   PRINT "Fehler beim Kopieren von
C:\ThisFolderDoesSurelyNotExist\Src.txt"
   PRINT "nach C:\TMP\Src.txt."
ELSE
   PRINT "C:\ThisFolderDoesSurelyNotExist\Src.txt nach C:\TMP\Src.txt
kopiert."
END IF

' Die erstellten Daten löschen
KILL "TMP\Src.txt"
KILL "TMP\Dst.txt"
KILL "Dst.txt"

' Das Verzeichnis TMP löschen, wenn sich keine
' Daten oder Unterordner mehr darin befinden.
RMDIR "TMP"

SLEEP
```

Ausgabe:

```
C:\TMP\Src.txt nach C:\TMP\Dst.txt kopiert.
C:\TMP\Src.txt nach C:\Dst.txt kopiert.
Fehler beim Kopieren von C:\TMP\*.txt nach C:\TMP\Dst.txt.
Fehler beim Kopieren von C:\TMP\Src.txt nach C:\TMP\Src.txt.
Fehler beim Kopieren von C:\ThisFolderDoesSurelyNotExist\Src.txt
nach C:\TMP\Src.txt.
```

Unterschiede zu QB: existiert nur in Visual Basic

Plattformbedingte Unterschiede:

- Unter Linux muss der Dateiname 'case sensitive' erfolgen. Windows und DOS sind 'case insensitive'.
- Das Trennzeichen für den Dateipfad ist unter Linux der vorwärtsgerichtete Slash /. Windows

verwendet den Backslash \, erlaubt aber auch den Slash. DOS verwendet den Backslash.

Unterschiede zu früheren Versionen von FreeBASIC: existiert seit FreeBASIC v0.17

Siehe auch:
FILEEXISTS, KILL, Dateien (Files), Betriebssystem-Anweisungen

Letzte Bearbeitung des Eintrags am 27.12.12 um 00:46:51
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

FILEDATETIME

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » F » **FILEDATETIME**
Syntax: FILEDATETIME (Dateiname)
Typ: Funktion
Kategorie: Dateien

FILEDATETIME gibt den Zeitpunkt (Datum und Uhrzeit), an dem die Datei zuletzt bearbeitet wurde, als Serial Number zurück.

Anmerkung:
Um diese Funktion nutzen zu können, muss die Datei file.bi in Ihren Quellcode eingebunden werden, z.B. mit INCLUDE. Alternativ können Sie auch die Datei vbcompat.bi einbinden, da diese die file.bi automatisch in Ihr Programm lädt.

- 'Dateiname' ist ein STRING oder ein ZSTRING PTR, der angibt, welche Datei analysiert werden soll. Er kann eine Pfadangabe enthalten, muss aber nicht. Enthält 'Dateiname' keinen Pfad, geht FreeBASIC automatisch davon aus, dass sich die Datei im aktuellen Arbeitsverzeichnis befindet.
- Der Rückgabewert ist eine Serial Number, die Datum und Uhrzeit der letzten Bearbeitung enthält, oder 0, wenn ein Fehler aufgetreten ist. Die Serial Number wird als DOUBLE-Wert zurückgegeben.

Beispiel:

```
"hlstring">"vbcompat.bi"

DIM filename AS STRING, d AS DOUBLE

Print "Bitte geben Sie einen Dateinamen ein:"
LINE INPUT filename

IF FILEEXISTS( filename ) THEN
   d = FILEDATETIME( filename )
   PRINT "Datei wurde zuletzt geaendert am ";
   PRINT FORMAT( d, "dd.mm.yyyy, hh:mm:ss" )
ELSE
   PRINT "Datei wurde nicht gefunden"
END IF
SLEEP
```

Ausgabe:

```
Bitte geben Sie einen Dateinamen ein:
D:\BASIC\FreeBASIC\fbc.exe
Die Datei wurde zuletzt geändert am 16.04.2010, 00:15:02
```

Plattformbedingte Unterschiede:

- Unter Linux muss der Dateiname 'case sensitive' erfolgen. Windows und DOS sind 'case insensitive'.
- Das Trennzeichen für den Dateipfad ist unter Linux der vorwärtsgerichtete Slash /. Windows verwendet den Backslash \, erlaubt aber auch den Slash. DOS verwendet den Backslash.

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC: existiert seit FreeBASIC v0.17

Siehe auch:
FILEATTR, FILECOPY, FILEEXISTS, FILELEN, OPEN, CLOSE, Dateien (Files), Betriebssystem-Anweisungen

Letzte Bearbeitung des Eintrags am 27.12.12 um 00:47:03
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

FILEEXISTS

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » F » **FILEEXISTS**
Syntax: FILEEXISTS (Dateiname)
Typ: Funktion
Kategorie: Dateien

Anmerkung:
Um diese Funktion nutzen zu können, muss die Datei file.bi in Ihren Quellcode eingebunden werden, z. B. mit INCLUDE. Alternativ können Sie auch die Datei vbcompat.bi einbinden, da diese die file.bi automatisch in Ihr Programm lädt.

FILEEXISTS überprüft, ob eine Datei existiert oder nicht.
Intern wird versucht, die Datei zu öffnen. Dies kann bewirken, dass ein bestehender LOCK gelöst wird. Je nach Anforderung kann auf andere Methoden zur Überprüfung ausgewichen werden. Sie können z. B. DIR einsetzen (hierbei muss genau auf die Attribute und den Pfad geachtet werden) oder versuchen, mit OPEN die Datei zu öffnen, und anschließend den Rückgabewert prüfen.

- 'Dateiname' ist ein STRING oder ein ZSTRING PTR, der den Namen der Datei enthält, deren Existenz bestätigt werden soll. Er kann eine Pfadangabe enthalten, muss aber nicht. Enthält 'Dateiname' keinen Pfad, geht FreeBASIC automatisch davon aus, dass sich die Datei im aktuellen Arbeitsverzeichnis befindet (siehe CURDIR).
- Der Rückgabewert ist entweder -1, wenn die Datei existiert, oder 0, wenn dies nicht der Fall ist.

Beispiel:

```
"hlstring">"vbcompat.bi"

DIM filename AS STRING

PRINT "Bitte geben Sie einen Dateinamen ein: "
LINE INPUT filename

IF FILEEXISTS( filename ) THEN
  PRINT "Datei gefunden: " & filename
ELSE
  PRINT "Datei nicht gefunden: " & filename
END IF
SLEEP
```

Unterschiede zu QB: neu in FreeBASIC

Plattformbedingte Unterschiede:

- Unter Linux muss der Dateiname 'case sensitive' erfolgen. Windows und DOS sind 'case insensitive'.
- Das Trennzeichen für den Dateipfad ist unter Linux der vorwärtsgerichtete Slash /. Windows verwendet den Backslash \, erlaubt aber auch den Slash. DOS verwendet den Backslash.

Unterschiede zu früheren Versionen von FreeBASIC: existiert seit FreeBASIC v0.17

Siehe auch:
DIR, OPEN (Funktion), Dateien (Files), Betriebssystem-Anweisungen

Letzte Bearbeitung des Eintrags am 27.12.12 um 00:47:18
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

FILELEN

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » F » **FILELEN**
Syntax: FILELEN (Dateiname)
Typ: Funktion
Kategorie: Dateien

FILELEN gibt die Länge von einer Datei in Bytes zurück

Anmerkung:
Um diese Funktion nutzen zu können, muss die Datei file.bi in Ihren Quellcode eingebunden werden, z. B. mit INCLUDE. Alternativ können Sie auch die Datei vbcompat.bi einbinden, da diese die file.bi automatisch in Ihr Programm lädt.

- 'Dateiname' ist ein STRING, der den Dateinamen der zu analysierenden Datei enthält. Dieser kann eine Pfadangabe enthalten. Wird nur ein Dateiname ohne Pfad angegeben, geht FreeBASIC davon aus, dass sich die Datei im aktuellen Arbeitsverzeichnis befindet (siehe CURDIR). Wenn 'Dateiname' nicht existiert, wird keine Datei erstellt.
- Der Rückgabewert ist eine INTEGER-Zahl, welche die Größe der Datei in Bytes angibt, oder 0, wenn die Datei nicht existiert. Achtung: Null wird auch dann zurückgegeben, wenn die Datei tatsächlich 0 Bytes groß ist.

Beispiel:

```
"hlstring">"file.bi"

DIM laenge AS INTEGER
laenge = FILELEN("C:\BASIC\FreeBASIC\fbc.exe")

PRINT "Der fbc ist"; laenge; " Bytes gross, das sind:"
PRINT  laenge \ (1024   *  1024); " Megabytes,";
PRINT (laenge \  1024) MOD 1024 ; " Kilobytes und";
PRINT  laenge          MOD 1024 ; " Bytes."

SLEEP
```

Ausgabebeispiel:

```
Der fbc ist 568832 Bytes gross, das sind:
0 Megabytes, 555 Kilobytes und 512 Bytes.
```

Unterschiede zu QB: neu in FreeBASIC

Plattformbedingte Unterschiede:

- Unter Linux muss der Dateiname 'case sensitive' erfolgen. Windows und DOS sind 'case insensitive'.
- Das Trennzeichen für den Dateipfad ist unter Linux der vorwärtsgerichtete Slash /. Windows verwendet den Backslash \, erlaubt aber auch den Slash. DOS verwendet den Backslash.

Unterschiede zu früheren Versionen von FreeBASIC: existiert seit FreeBASIC v0.17

Siehe auch:
LOF, FILEEXISTS, FILEATTR, FILEDATETIME, Dateien (Files), Betriebssystem-Anweisungen

Letzte Bearbeitung des Eintrags am 27.12.12 um 00:47:29
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

FIX

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » F » **FIX**
Syntax: FIX (Variable)
Typ: Funktion
Kategorie: Mathematik

FIX verwandelt eine beliebige Zahl in eine INTEGER-Zahl, indem der Nachkommateil abgeschnitten wird. Wenn die Eingabezahl ein INTEGER ist, bleibt sie unverändert. Ansonsten wird in Richtung 0 gerundet. Beispielsweise gibt FIX(1.9) 1 und FIX(-1.9) -1 zurück.

Es bestehen Ähnlichkeiten zwischen FIX und INT. Sie unterscheiden sich jedoch bei negativen Zahlen, da INT immer abrundet. Während FIX(-1.9) -1 ausgibt, liefert INT(-1.9) -2. Bei positiven Zahlen besteht zwischen beiden Funktionen kein Unterschied.

FIX kann mithilfe von OPERATOR überladen werden.

Siehe auch:
INT, CINT, FRAC, mathematische Funktionen

Letzte Bearbeitung des Eintrags am 05.07.12 um 20:03:29
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

FLIP

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » F » **FLIP**
Syntax: FLIP [Quellseite] [, Zielseite]
Typ: Anweisung
Kategorie: Grafik

Im Grafikmodus hat der Befehl FLIP dieselbe Funktion wie die Befehle PCOPY und SCREENCOPY: Er kopiert den Inhalt einer Bildschirmseite auf eine andere.

Wenn ein Bildschirmmodus im OpenGL-Modus initiiert wurde, bewirkt FLIP eine Bildschirmaktualisierung. Die Verwendung mehrere Bildschirmseiten ist in diesem Modus nicht möglich.

Beispiel:

```
ScreenRes 320, 240, 32, 2   ' Fenster mit 320x240 Pixeln und 32-bit
Farbtiefe und zwei Bildseiten.

For n As Integer = 50 To 270

    ScreenSet 2, 1              ' stellt die zu bearbeitende Seite auf 2
und zeigt Seite 1 an.
    Cls
    Circle (n, 50), 50 , RGB(255, 255, 0) ' zeichnet einen gelben Kreis
und einem Radius von 50 Pixel auf Seite 2.
    ScreenSet 1, 1              ' stellt die zu bearbeitende und
anzuzeigende Seite auf Seite 1.
    ScreenSync                  ' wartet bis die Bildschirmaktualisierung
erfolgt ist.
    Flip 2, 1                   ' kopiert den Kreis von Seite 2 auf Seite
1.

    Sleep 25
Next

Print "Taste druecken, um zu beenden."
Sleep
```

Unterschiede zu QB: neu in FreeBASIC

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht FLIP nicht zur Verfügung und kann nur über **__FLIP** aufgerufen werden.

Siehe auch:
SCREENRES, SCREENCOPY, Grafik

Letzte Bearbeitung des Eintrags am 27.12.12 um 00:48:10
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

FOR

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » F » **FOR**
FOR wird in FOR-Schleifen, bei Datei- und Hardwarezugriffen, sowie bei Operatoren verwendet.

FOR-Schleife:
siehe FOR ... NEXT

Datei- und Hardwarezugriffe:
siehe OPEN

Operatoren:
Als Operator wird FOR deklariert, wenn man den Befehl für eigene Datentypen überladen möchte.
Das Tutorial zu Thema 'Overload' erklärt die Verwendung hierbei näher.

Letzte Bearbeitung des Eintrags am 15.06.13 um 23:43:33
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

FOR ... NEXT

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » F » **FOR ... NEXT**
Syntax:

```
FOR Variable [AS Typ] = Start TO Ende [STEP Schrittweite]
  ' Programmcode
NEXT [Variable]
```

Typ: Anweisung
Kategorie: Programmablauf

Die FOR-Schleife wiederholt den Codeblock zwischen FOR und NEXT, wobei eine Variable bei jedem Durchlauf um einen bestimmten Wert erhöht wird.

- 'Variable' ist die Zählvariable, deren Wert sich beim Schleifendurchlauf verändern soll. Es kann sich bei ihr um jeden Zahlen-Datentyp handeln.
- 'AS Typ' sorgt dafür, dass 'Variable' speziell für die Schleife neu angelegt wird. Diese Dimensionierung gehört dann zum SCOPE-Block der FOR-Schleife (vgl. Gültigkeitsbereich von Variablen).
- 'Start' ist der Startwert, welcher der Zählvariablen zu Beginn der Schleife zugewiesen wird.
- 'Schrittweite' gibt an, um welchen Wert die Zählvariable bei jedem Schleifendurchlauf verändert werden soll. Wird 'STEP Schrittweite' ausgelassen, dann wird die Zählvariable bei jedem Durchlauf um 1 erhöht.
- 'Ende' ist der Wert, bis zu dem die Zählvariable laufen soll. Wird dieser Wert erreicht, dann wird die Schleife noch ein letztes mal durchlaufen; wird der Wert überschritten (oder bei negativer Schrittweite unterschritten), dann wird die Schleife verlassen.

'Start' und 'Ende' werden zu Beginn der Schleife intern gespeichert und können innerhalb der Schleife nicht mehr geändert werden. Werden für 'Start' bzw. 'Ende' Variablen verwendet, dann können diese Variablen zwar in der Schleife geändert werden, dies beeinflusst jedoch nicht mehr den Start- bzw. Endwert der Schleife.

Achtung: Die Verwendung der Datentypen UBYTE und USHORT arbeiten zur Zeit nicht zusammen mit einer negativen Schrittweite! Wenn Sie mit STEP eine negative Schrittweite verwenden wollen, dann verwenden Sie die vorzeichenbehaftete Version der beiden Datentypen oder den wesentlich schneller berechenbaren Datentyp INTEGER.

Bei Ganzzahl-Datentypen ist es außerdem nicht möglich, bis zum höchstmöglichen Wert zu zählen (oder bis zum tiefstmöglichen bei negativer Schrittweite), da die Schleife nur verlassen wird, wenn die Zählvariable den Endwert überschreitet. Das tritt in diesem Fall jedoch nie ein. Wird beispielsweise versucht eine UBYTE-Variable von 0 bis 255 zählen zu lassen, wird die Schleife erst verlassen, wenn die Variable den Wert 256 oder höher erreicht. Den Wert 256 kann die UBYTE-Variable jedoch nicht speichern. Stattdessen wird sie nach 255 auf 0 zurückgesetzt und die Schleife läuft weiter.

Wenn Sie für 'Variable' einen Ganzzahlentyp und für 'Schrittweite' eine Fließkommazahl verwenden, dann wird 'Schrittweite' zuvor auf eine Ganzzahl gerundet. Ist 'Schrittweite' kleiner oder gleich 0.5, dann wird sie auf 0 abgerundet und damit eine Endlosschleife erzeugt.
Ist 'Schrittweite' positiv und 'Start' größer als 'Ende', dann wird die Schleife nicht durchlaufen. Dasselbe gilt für eine negative 'Schrittweite', wenn 'Start' kleiner ist als 'Ende'.

Als Block-Anweisung initialisiert die FOR-Schleife einen SCOPE-Block, der mit der Zeile NEXT endet. Wird die Zählvariable mit 'AS Typ' neu dimensioniert, dann ist sie ebenfalls außerhalb der Schleife nicht mehr gültig.

Eine FOR-Schleife kann 'manuell' mit der Anweisung EXIT FOR abgebrochen werden. Taucht innerhalb des Schleifen-Codes die Anweisung CONTINUE FOR auf, so ignoriert FreeBASIC den Code bis zum nächsten NEXT und springt zurück zur FOR-Zeile. Die Zählvariable wird dabei um 'Schrittweite' erhöht.

FOR, STEP und NEXT können mithilfe von OPERATOR überladen werden. Zum Einsatz von UDTs als Zählvariable siehe den Tutorial-Beitrag zum Überladen von Iteratoren.

Beispiel 1: gibt ein Dreieck aus Sternen aus

```
DIM AS INTEGER i, hoehe
hoehe = 10
FOR i = 1 TO hoehe
  PRINT STRING(i, "*")
NEXT
PRINT "Nach der Schleife hat i den Wert "; i
SLEEP
```

Ausgabe:

```
*
**
***
****
*****
******
*******
********
*********
**********
Nach der Schleife hat i den Wert 11
```

Erläuterung zu Beispiel 1: Nach dem zehnten Durchlauf wird die Zählvariable ein weiteres Mal erhöht und ihr Wert anhand der Schleifenbedingung überprüft. Da sie nun den Wert 11 besitzt, wird die Schleife verlassen.

Beispiel 2: Zählschleife mit automatischer Initialisierung

```
DIM i AS INTEGER = 1357
PRINT "Wert von i vor der Schleife: "; i

PRINT "Zähle von 3 bis 0 in Schritten von -0.2:"
FOR i AS SINGLE = 3 TO 0 STEP -0.2
  PRINT "i = "; i
NEXT

PRINT "Wert von i nach der Schleife: "; i
SLEEP
```

Erläuterung zu Beispiel 2: Nach Beendigung der Schleife wird die Zählvariable zerstört; die außerhalb der Schleife deklarierte Variable existiert weiterhin und es kann nun wieder auf sie zugegriffen werden.
Beachten Sie in diesem Beispiel auch die auftretenden Rundungsungenauigkeiten. Es ist besser, Zählvariablen vom Typ INTEGER einzusetzen und diese ggf. innerhalb der Schleife umzurechnen.

FOR in Prozeduren kann nicht mit Variablen als Zähler verwendet werden, die BYREF übergeben wurden.

Das liegt daran, dass bei BYREF intern ein Pointer übergeben wird und dieser nicht als Zählvariable eingesetzt werden kann.

Beispiel 3: ergibt einen Compilier-Fehler

```
DECLARE SUB foo (BYREF a As Integer)

DIM a AS INTEGER
foo a
SLEEP

SUB foo(BYREF a AS INTEGER)
  FOR a = 1 TO 100
  NEXT a
END SUB
```

Alternativ kann in der Prozedur eine neue Variable angelegt werden:

```
SUB foo (BYREF a AS INTEGER)
   DIM b AS INTEGER = a
   FOR b = 1 TO 100
   NEXT
   a = b
END SUB
```

Die hinter NEXT angegebene (optionale) Variable muss mit der bei FOR angegebenen Zählvariablen übereinstimmen. Es ist auch möglich, mehrere Zählvariablen hintereinander anzugeben, um mehrere FOR-Schleifen gleichzeitig abzuschließen.

Beispiel 4:

```
FOR i AS INTEGER = 1 TO 3 : FOR k AS INTEGER = 7 TO 9
  PRINT i, k
NEXT k, i
SLEEP
```

Unterschiede zu QB:

- FreeBASIC erlaubt als Zählvaribale keine mit BYREF übergebene Variable.
- In QB dürfen innerhalb einer Schleife keine Variablen definiert werden.
- Die Syntax FOR Variable AS Typ = ... ist neu in FreeBASIC.

Unterschiede zu früheren Versionen von FreeBASIC:

- Seit FreeBASIC v0.18.3 muss die Variable hinter NEXT (sofern angegeben) der Schleifenvariablen entsprechen.
- Seit FreeBASIC v0.17 kann FOR seine Zähler-Variable über die Syntax FOR Variable AS TYP = ... automatisch initialisieren.
- Seit FreeBASIC v0.16 wirkt eine FOR-Schleife wie ein SCOPE-Block.

Unterschiede unter den FB-Dialektformen:

- In der Dialektform -lang qb und -lang fblite sind Variablen, die innnerhalb einer FOR-Schleife dimensioniert werden, in der ganzen Funktion sichtbar.

- In der Dialektform -lang fb und -lang deprecated sind Variablen, die in einer FOR-Schleife dimensioniert wurden, nur innerhalb dieser Schleife gültig.

Siehe auch:
DO...LOOP, WHILE...WEND, CONTINUE, SCOPE, EXIT, Schleifen

Letzte Bearbeitung des Eintrags am 27.12.12 um 00:48:37
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

FORMAT

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » F » **FORMAT**
Syntax: FORMAT[$](numerischer_Ausdruck, FormatString)
Typ: Funktion
Kategorie: Stringfunktionen

Anmerkung:
Um diese Funktion nutzen zu können, muss die Datei vbcompat.bi in Ihren Quellcode eingebunden werden, z.B. mit INCLUDE.

FORMAT wandelt einen numerischen Ausdruck anhand der angegebenen Formatierung in einen STRING um.

- 'numerischer_Ausdruck' ist ein beliebiger Ausdruck, der als DOUBLE gehandhabt wird. Dies kann insbesondere auch eine Serial Number sein.
- 'FormatString' ist ein String, der festlegt, welche Form die Ausgabe haben soll. Er wird in den unten stehenden Tabellen näher erläutert.

Das Dollarzeichen ($) als Suffix ist optional.

Allgemeine Formatierungszeichen:

Symbol	Erklärung
Nullstring	keine besondere Formatierung
0	Platzhalter für eine Ziffer. Wenn die auszugebende Zahl weniger Ziffern hat, als Platzhalter reserviert wurden, so werden diese mit führenden Nullen aufgefüllt. Nach dem Dezimaltrennzeichen werden so viele Ziffern dargestellt wie angegeben. Die Ziffern vor dem Dezimaltrennzeichen werden vollständig angezeigt, auch wenn im Formatierungsstring weniger Stellen angegeben werden.
#	Platzhalter für eine Ziffer. Funktioniert genauso wie 0, erzeugt aber keine führenden Nullen.
.	Platzhalter für ein Dezimaltrennzeichen. Wenn der Formatierungsstring nur Rauten (#) links vom Punkt (.) enthält, werden Zahlen, die kleiner als eins sind, ohne Null vor dem Komma ausgegeben. Das Dezimaltrennzeichen kann in den Ländereinstellungen eingestellt werden.
%	Der Ausdruck wird mit 100 multipliziert und mit einem Prozent-Zeichen (%) ausgegeben.
,	Platzhalter für Tausendertrennzeichen. Zwei aufeinanderfolgende Kommata oder ein Komma direkt links vom Dezimaltrennzeichen (egal ob die Position eines solchen angegeben wurde oder nicht) bewirken, dass die drei Ziffern zwischen den Kommata bzw. dem Komma und dem Punkt ausgelassen werden; die Zahl wird dabei korrekt gerundet. Das Tausendertrennzeichen kann in den Ländereinstellungen eingestellt werden.
E- E+ e- e+	Wissenschaftliches Format: Die Zahl wird als Zehnerpotenz angegeben. Die Anzahl der Platzhalter für Ziffern (0 und #) links von E-, E+, e- oder e+ gibt dabei an, wie viele Stellen für die Darstellung der Zahl selbst reserviert werden; die Anzahl der Platzhalter rechts gibt an, wie viele Stellen für den Exponenten reserviert werden. E- und e- bewirken, dass das Vorzeichen des Exponenten nur angezeigt wird, wenn er negativ ist. E+ und e+ bewirken, dass das Vorzeichen des Exponenten sowohl angezeigt wird, wenn er positiv als auch wenn er negativ ist.
: ? + $ () Space	

	Literale: Diese Zeichen werden so ausgegeben, wie sie im Formatierungsstring stehen. Wenn Sie andere Zeichen als diese ausgeben wollen, müssen Sie einem einzelnen Zeichen einen Backslash (\) voranstellen oder die Zeichen in Anführungszeichen (") einschließen. Um einen Backslash auszugeben, müssen Sie also \\ oder "\" in Ihren Formatierungsstring einbauen.
\	Das nächste Zeichen im Formatierungsstring wird als Literal ausgegeben.
"Text zwischen Anführungszeichen"	Der Text innerhalb der Anführungszeichen wird so ausgegeben, wie er im Formatierungsstring steht.
:	Der Doppelpunkt wird verwendet, um Stunden, Minuten und Sekunden voneinander zu trennen, wenn Zeitwerte formatiert werden.
/	Datumstrennzeichen: Der Schrägstrich wird verwendet, um Tage, Monate und Jahre voneinander zu trennen, wenn Datumswerte formatiert werden. In den Ländereinstellungen lässt sich einstellen, wie dieses Zeichen interpretiert werden soll.

Datum/Zeit-Formatierungszeichen (bei Serial Numbers):

Symbol	Erklärung
d	Zeige den Tag als Zahl ohne führende Null an (0-31)
dd	Zeige den Tag als Zahl mit führender Null an (00-31)
ddd	Zeige den Tag als Abkürzung seines Namens an (So-Sa)*
dddd	Zeige den Tag als vollen Namen an (Sonntag - Samstag)*
ddddd	Zeige das Datum als vollständiges Datum an, einschließlich Tag, Monat und Jahr*
m	Zeige den Monat als Zahl ohne führende Null an (1-12). Wenn 'm' direkt nach 'h' oder 'hh' benutzt wird, wird die Minute ohne führende Null angezeigt (0-59).
mm	Zeige den Monat als Zahl mit führender Null an (01-12). Wenn 'mm' direkt nach 'h' oder 'hh' benutzt wird, wird die Minute mit führender Null angezeigt (00-59).
M, MM	Zeige den Monat als Zahl ohne bzw. mit führender Null an (wie m und mm), jedoch auch dann, wenn es direkt nach 'h' oder 'hh' benutzt wird.
mmm	Zeige den Monat als Abkürzung seines Namens an (Jan-Dez)*
mmmm	Zeige den Monat als vollen Namen an (Januar-Dezember)*
y oder yy	Zeige das Jahr als zweistellige Zahl an (00-99)
yyyy	Zeige das Jahr als vierstellige Zahl an (1900-2040)
h	Zeige die Stunde ohne führende Null an (0-23)
hh	Zeige die Stunde mit führender Null an (00-23)
n	Zeige die Minute ohne führender Null an (wie m), jedoch auch wenn es nicht direkt hinter h oder hh verwendet wird.
nn	Zeige die Minute mit führender Null an (wie mm), jedoch auch wenn es nicht direkt hinter h oder hh verwendet wird.
s	Zeige die Sekunde ohne führende Null an (0-59)
ss	Zeige die Sekunde mit führender Null an (00-59)
ttttt	Zeige die komplette Uhrzeit mit Stunde, Minute und Sekunde an.*
AM/PM am/pm	Gib die Zeit als 12-Stunden-Zeit (00:00-11:59) mit AM bzw. am aus, wenn die Uhrzeit vor 12:00 liegt, und mit PM bzw. pm, wenn sie nach 12:00 liegt.
A/P a/p	Gib die Zeit als 12-Stunden-Zeit (00:00-11:59) mit A bzw. a aus, wenn die Uhrzeit vor 12:00 liegt, und mit P bzw. p, wenn sie nach 12:00 liegt.

Hinweis: * Die Darstellung ist von den lokalen Ländereinstellungen abhängig.

Ausgabebeispiel:

```
Einfache Zahlenformatierung:
Formatstring            5         -5         .5         5000.5
Leerstring              5         -5         0,5        5000,5
0                       5         -5         1          5000
0.00                    5,00      -5,00      0,50       5000,50
#,##0                   5         -5         1          5.000
#,##0.00                5,00      -5,00      0,50       5.000,00
$#,##0;($#,##0)         $5        ($5)       $1         $5.000
$#,##0.00;($#,##0.00)   $5,00     ($5,00)    $0,50      $5.000,00
0%                      500%      -500%      50%        500000%
0.00%                   500,00%   -500,00%   50,00%     500000.00%
0.00E+00                5,00E+00  -5,00E+00  5,00E-01   5,00E+03
0.00E-00                5,00E00   -5,00E00   5,00E-01   5,00E03

Zeitformatierung:
Formatstring            Ausgabe
m/d/yy                  12/7/58
d-mmmm-yy               7-Dezember-58
d-mmmm                  7-Dezember
mmmm-yy                 Dezember-58
dddd                    Sonntag
ddddd                   07.12.1958
h:mm AM/PM              8:50 PM
h:mm:ss AM/PM           8:50:35 PM
h:mm                    20:50
h:mm:ss                 20:50:35
m/d/yy h:mm             12/7/58 20:50
```

Unterschiede zu QB:
Diese Funktion existiert nur in QBX PDS und in VBDOS.

Unterschiede zu früheren Versionen von FreeBASIC:

- FORMAT existiert seit FreeBASIC v0.15
- Die Formatierungsangabe 'n' und 'nn' existiert seit FreeBASIC v0.21

Siehe auch:
PRINT USING, STR, Serial Numbers, String-Funktionen

Letzte Bearbeitung des Eintrags am 27.12.12 um 00:49:11
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

FRAC

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » F » **FRAC**
Syntax: FRAC (Variable)
Typ: Funktion
Kategorie: Mathematik

FRAC gibt die Nachkommastellen inklusive Vorzeichen einer Zahl zurück.

Wird FRAC(10.625) aufgerufen, so wird 0.625 zurückgegeben.
Wird FRAC(-10.625) aufgerufen, so wird -0.625 zurückgegeben.

FRAC kann mithilfe von OPERATOR überladen werden.

Unterschiede zu QB: neu in FreeBASIC

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht FRAC nicht zur Verfügung und kann nur über **__FRAC** aufgerufen werden.

Siehe auch:
FIX, INT, mathematische Funktionen

Letzte Bearbeitung des Eintrags am 05.07.12 um 20:35:01
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

FRE

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » F » **FRE**
Syntax: FRE [(parameter)]
Typ: Funktion
Kategorie: Speicher

FRE gibt den verfügbaren RAM-Speicher in Bytes zurück.
Der Parameter kann lediglich aus Kompatibilitätsgründen angegeben werden, er erfüllt aber keinen Zweck und kann somit ignoriert werden.

Beispiel:

```
DIM AS INTEGER mem

mem = FRE
PRINT "Freier Speicher:"
PRINT
PRINT mem; " Bytes"
PRINT mem / 1024; " Kilobytes"
PRINT mem / (1024 * 1024); " Megabytes"
SLEEP
```

Unterschiede zu QB:
FRE braucht in FreeBASIC keinen Parameter mehr. Es gibt immer den verfügbaren physischen freien Speicher zurück.

Siehe auch:
DIM, ALLOCATE, Speicher

Letzte Bearbeitung des Eintrags am 27.12.12 um 00:49:27
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

FREEFILE

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » F » **FREEFILE**
Syntax: FREEFILE
Typ: Funktion
Kategorie: Dateien

FREEFILE gibt die nächste unbenutzte Dateinummer zurück.

Beispiel:

```
Dim buffer As String, ff As Integer
buffer = "Hello World in einer Datei."

' Die erste freie Dateinummer zuweisen
ff = FreeFile

' Die Datei "file.txt" mit der Nummer "ff" öffnen
Open "file.txt" For Binary As "hlkommentar">' Den String in die Datei
laden
Put "hlzeichen">, , buffer

' Die Datei schließen
Close "hlkommentar">' Das Programm beenden; die Ausgabe befindet sich in
der Datei "file.txt"
End
```

FREEFILE sollte immer vor jedem OPEN aufgerufen werden:

```
'So ist es richtig:
Dim fr As Integer, fs As Integer

fr = FreeFile
Open "File1" For Input As "hlzeichen">= FreeFile
Open "file2" For Input As "hlkw0">Close "hlkw0">Close "hlkommentar">'So
ist es falsch:
Dim fr As Integer, fs As Integer

fr = FreeFile
fs = FreeFile    ' fs enthält nun dieselbe Nummer wie fr
Open "File1" For Input As "hlkw0">Open "file2" For Input As "hlkw0">Close
"hlkw0">Close "reflinkicon" href="temp0058.html">CLOSE vergessen wurde,
so gibt FREEFILE 0 zurück. Dies zeigt an, dass keine weiteren Dateien
geöffnet werden können.
```

Hinweis: Wenn man FREEFILE aus einer DLL aus aufruft, ergibt das nicht die gleichen Nummern wie im Hauptprogramm. Dateinummern können also nicht beliebig zwischen Hauptprogramm und DLL übertragen werden.

Siehe auch:
OPEN, Dateien (Files)
Letzte Bearbeitung des Eintrags am 27.12.12 um 00:49:41
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

FUNCTION

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » F » **FUNCTION**
Syntax:

```
[{PRIVATE | PUBLIC }] FUNCTION Funktionsname [Aufrufkonvention] [ALIAS
"AliasName"] _
    [OVERLOAD] ([Parameterliste]) [BYREF] AS Rückgabetyp [STATIC]
[EXPORT]

     ' Anweisungen
     { RETURN | FUNCTION = | Funktionsname = } [BYVAL] wert
END FUNCTION
```

Typ: Prozedur
Kategorie: Programmablauf

Anmerkung zur Syntax: Unterstriche (_) am Zeilenende werden von FreeBASIC so interpretiert, als wäre die Zeile nicht unterbrochen; dies dient nur der besseren Übersichtlichkeit und hat letzt–end–lich keine Auswirkungen auf die Programmausführung.

Eine FUNCTION ist eine Prozedur, die einen Wert an das Hauptprogramm zurück gibt. FUNCTIONs werden dazu verwendet, häufig auftretende Berechnungen zusammenzufassen. Siehe auch SUB zum Thema Prozeduren.

- Die Klausel 'PRIVATE' bewirkt, dass die Funktion nur aus dem Modul heraus aufgerufen werden kann, in dem sie sich befindet. Wird nichts angegeben, so wird die Funktion standardmäßig als PUBLIC deklariert. Siehe dazu die Referenzeinträge PRIVATE und PUBLIC.
- 'Funktionsname' ist der Bezeichner, unter dem die Funktion aufgerufen wird.
- 'Aufrufkonvention' ist ein Schlüsselwort, das angibt, in welcher Reihenfolge die Parameter übergeben werden sollen. Möglich sind: STDCALL, CDECL und PASCAL.
- 'ALIAS' gibt einer Funktion in einer Library einen neuen Namen, mit dem man auf sie verweisen kann.
- 'OVERLOAD' erlaubt, Prozeduren mit unterschiedlicher Parameterliste, aber gleichem Funktionsnamen zu deklarieren. Siehe dazu den Referenzeintrag OVERLOAD.
- 'Parameterliste' gibt die Parameter an, die an die Funktion übergeben werden. Die Parameterübergabe wird weiter unten erläutert.
- Mit 'BYREF' kann angegeben werden, das die Rückgabe *by reference* erfolgt.
- 'Rückgabetyp' ist der Datentyp des Rückgabewerts. Auch UDTs und POINTER sind erlaubt.
- Die Klausel 'STATIC' bewirkt, dass alle Variablen innerhalb der Funktion zwischengespeichert werden. Das bedeutet, dass die Werte von Variablen, die in der Funktion verwendet werden, nach einem Aufruf nicht verloren gehen, sondern beim nächsten Aufruf der Funktion wieder verfügbar sind. Siehe auch STATIC (Klausel).
- Die Klausel 'EXPORT' bewirkt, dass die Funktion zur PUBLIC-EXPORT-Tabelle hinzugefügt wird, sodass sie von externen Programmen aus mit DYLIBSYMBOL verlinkt werden kann. Siehe dazu den Referenzeintrag EXPORT.

Um eine Funktion aufzurufen, bevor sie im Quellcode definiert wurde, muss sie zuvor deklariert werden:

```
DECLARE FUNCTION Funktionsname &"hlzeichen">] [OVERLOAD] [[LIB
"DLLName"&"hlzeichen">_
      ALIAS "AliasName"&"hlzeichen">[ ( Parameterliste ) ]  AS
Rückgabetyp
```

Siehe dazu den Referenzeintrag DECLARE.

Parameterübergabe
Der Wert der Funktion kann von übergebenen Parametern abhängig sein. Die Parameterliste besitzt die Form

```
&"hlkw2">BYVAL | BYREF } &"hlzeichen">[Parameter1] AS &"hlzeichen">] Typ
[= Wert]    [, _
   [ {BYVAL | BYREF } &"hlzeichen">[Parameter2] AS &"hlzeichen">] Typ [=
Wert] ] [, ...]
```

- 'Parameter1', 'Parameter2', ... sind die Parameter (auch Argumente genannt). Es sind Variablen (bzw. ihre Werte), die in der Funktion verwendet werden können.
- Wird 'CONST' angegeben, kann der Wert innerhalb der Funktion zwar gelesen, aber nicht mehr beschrieben werden.
- 'Typ' ist der Datentyp des übergebenen Parameters. Auch UDTs und Pointer können verwendet werden.
- Durch '= Wert' wird ein Parameter als optional deklariert; er kann beim Aufruf der Prozedur ausgelassen werden. In diesem Fall wird an die Funktion für diesen Parameter 'Wert' übergeben. Das VisualBASIC-Äquivalent heißt OPTIONAL.
- BYVAL und BYREF regeln die Art der Parameterübergabe an die FUNCTION.

Der Datentyp muss bei den Parametern immer angegeben werden.
Siehe auch Parameterübergabe.

Rückgabewert
Der Rückgabewert der Funktion kann auf dreierlei Weise gesetzt werden:

- über ihren Bezeichner: Funktionsname = [BYVAL] Ausdruck
- über das Symbol "FUNCTION": FUNCTION = [BYVAL] Ausdruck
- über RETURN: RETURN [BYVAL] Ausdruck
  Achtung: RETURN funktioniert wie FUNCTION = [BYVAL] Ausdruck : EXIT FUNCTION

Hinweis:
Die Angabe 'BYVAL' spielt nur bei Funktionen eine Rolle, die *by reference* mit BYREF arbeiten.

Der Rückgabewert einer Funktion kann beim Aufruf auch ignoriert werden. Dies kann dann sinnvoll sein, wenn die Funktion z.B. darüber informiert, ob ihr Code erfolgreich ausgeführt wurde. Um den Rückgabewert einer Funktion zu ignorieren, setzen Sie diese einfach wie eine SUB ein.

Beispiel 1:

```
DECLARE FUNCTION twice (x AS INTEGER) AS INTEGER

FUNCTION twice (x AS INTEGER) AS INTEGER
   RETURN x * 2
   PRINT "Diese Zeile wird nie ausgegeben"
END FUNCTION

PRINT twice (10) ' gibt 20 aus, das Doppelte von 10
SLEEP
```

Beispiel 2: Rückgabe eines ganzen UDTs:

```
TYPE udt
```

```
   v1 AS INTEGER
   v2 AS INTEGER
END TYPE

DECLARE FUNCTION bla(foo AS INTEGER, bar AS INTEGER) AS udt

DIM variable AS udt
variable = bla(1, 3)

PRINT "Variable: v1 ="; variable.v1; ", v2 ="; variable.v2
'Ausgabe: "Variable: v1 = 10, v2 = 30"
SLEEP

FUNCTION bla(foo AS INTEGER, bar AS INTEGER) AS udt
   DIM variable AS udt
   variable.v1 = foo*10
   variable.v2 = bar*10
   bla = variable
END FUNCTION
```

Beispiel 3: FUNCTION kann auch Teil eines UDTs sein.

```
Type MyUDT
   x As Integer
   y As Integer
   z As Integer
   Declare Function SumOfAll As Integer
End Type

Function MyUDT.SumOfAll As Integer
   Return This.x + This.y + This.z
End Function

Dim variable As MyUDT
With variable
   .x = 5
   .y = 10
   .z = -3
End With

Print variable.SumOfAll
Sleep
```

Ausgabe:

```
 12
```

Unterschiede zu QB:

- Parameter können in FreeBASIC optional sein.
- Der Rückgabewert kann in FreeBASIC auch mit FUNCTION = Ausdruck oder RETURN Ausdruck festgelegt werden.
- Funktionen können in FreeBASIC überladen werden.
- Der Rückgabewert kann in FreeBASIC beim Aufruf der Funktion ignoriert werden.

- Der Rückgabewert kann in FreeBASIC auch *by reference* erfolgen.

Unterschiede zu früheren Versionen von FreeBASIC:

- Seit FreeBASIC v0.90 kann die Rückgabe per BYREF auch *by reference* erfolgen.
- Bis FreeBASIC v0.17 war es auch zulässig, das Schlüsselwort STATIC links vom Prozedur-Header zu platzieren; seit v0.17 allerdings ist die Form STATIC FUNCTION Name ungültig.
- Seit FreeBASIC v0.17 ist es zulässig, auf Prozedurebene (z.B. innerhalb einer FUNCTION) TYPEs, UNIONs und ENUMs zu erstellen. Außerdem können FUNCTIONs auch innerhalb eines UDTs erstellt werden.
- Seit FreeBASIC v0.16 ist es zulässig, Arrays undefinierter Größe auch innerhalb von FUNCTIONs zu definieren (z.B. DIM AS INTEGER foo()).
- Seit FreeBASIV v0.16 kann GOSUB und ON ... GOSUB nicht mehr auf Prozedurebene eingesetzt werden.
- Das Überladen einer FUNCTION ist seit FreeBASIC v0.14 möglich; siehe OVERLOAD.
- Seit FreeBASIC v0.14 können FUNCTIONs ganze UDTs zurückgeben.
- Seit FreeBASIC v0.14 kann FUNCTION innerhalb von ASM-Blocks eingesetzt werden, um den Rückgabewert zu bestimmen.
- Seit FreeBASIC v0.13 kann RETURN als Shortcut für FUNCTION = Ausdruck : EXIT FUNCTION eingesetzt werden.
- Variable Argumente sind seit FreeBASIC v0.13 möglich; siehe VA_ARG.
- Seit FreeBASIC v0.13 können optionale Parameter auch bei Strings eingesetzt werden.
- Seit FreeBASIC v0.13 kann der Rückgabewert auch über FUNCTION = Ausdruck gesetzt werden.
- Seit FreeBASIC v0.12 müssen in den DECLARE-Zeilen keine Parameter-Namen mehr genannt werden; es ist jedoch immer noch zulässig.

Unterschiede unter den FB-Dialektformen:

- In der Dialektform -lang fb werden Parameter standardmäßig BYVAL übergeben. In den Dialektformen -lang qb und -lang fblite ist die Standardübergabe BYREF. Siehe die zugehörigen Referenzeinträge für genauere Informationen.
- In den Dialektformen -lang qb und -lang fblite kann RETURN nur eingesetzt werden, wenn OPTION GOSUB ausgeschaltet ist. In der Dialektform -lang qb muss dies explizit durch OPTION NOGOSUB festgelegt werden.

Siehe auch:
SUB, PROPERTY, DECLARE, OVERLOAD, EXIT, END, BYVAL, BYREF, BYREF (Rückgaben), BYVAL (Rückgaben), SHARED, PRIVATE (Klausel), PUBLIC (Klausel), VA_ARG, Prozeduren, FUNCTIONs, Parameterübergabe

Letzte Bearbeitung des Eintrags am 04.07.13 um 23:00:26
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

GET (Datei)

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » G » **GET (Datei)**
Syntax: GET #Dateinummer, [Position], Variable [, Menge [, Gelesen]]
Typ: Anweisung/Funktion
Kategorie: Dateien

GET liest Daten aus einer Datei, die im BINARY- oder RANDOM-Modus geöffnet wurde.

- '#Dateinummer' ist die Nummer, die beim Öffnen der Datei mit OPEN zugewiesen wurde.
- 'Position' ist die Position innerhalb der Datei, von der gelesen werden soll. Für BINARY-Dateien ist es die Nummer des Bytes, für RANDOM-Dateien ist es die Nummer des Datensatzes. Wird dieser Parameter ausgelassen, liest FreeBASIC von der Position des Dateizeigers. Diese ist abhängig von den letzten Lese-/Schreibzugriffen auf die Datei und von der SEEK-Anweisung. Direkt nach dem Öffnen ist der Dateizeiger immer auf der ersten Stelle in der Datei.
- 'Variable' ist ein Bezeichner, in dem der eingelesene Wert gespeichert werden soll. Dies kann eine einfache Variable, ein Array oder ein UDT sein.
- 'Menge' wird eingesetzt, sobald die Daten in einen ALLOCATE-Puffer eingelesen werden sollen. Dadurch wird GET mitgeteilt, wie viele Speicherstellen eingelesen werden sollen. Gelesen werden Menge*SIZEOF(Puffer_Datentyp) Bytes. Wird 'Menge' ausgelasen, nimmt FreeBASIC automatisch 1 an. Es wird also immer eine ganze Speicherstelle gelesen (entspricht SIZEOF(DatenPufferTyp) Bytes, also z. B. 4 Bytes bei einem INTEGER PTR). Bei anderen Datentypen als ALLOCATE-Datenpuffern wird der Parameter 'Menge' ignoriert.
- 'Gelesen' enthält die Anzahl der tatsächlich gelesenen Bytes. Damit kann überprüft werden, ob versucht wurde, hinter dem Dateiende zu lesen.
- Wird GET als Funktion eingesetzt, ist der Rückgabewert eine der FreeBASIC-Fehlernummern. In diesem Fall müssen die Parameter in Klammern gesetzt werden; die Syntax lautet also:
  Variable = GET (#Dateinummer, [Position], Variable [, Menge [, Gelesen]])

Da GET an jeder beliebigen Stelle einer beliebigen Datei ein einzelnes Byte oder einen beliebig langen String einlesen kann, ist es möglich, GET sehr vielseitig zu nutzen.

Im BINARY-Modus richtet sich die Anzahl der einzulesenden Bytes nach dem Datentyp der Variable, in die gelesen wird. Sie lässt sich - außer für Strings variabler Länge - über SIZEOF ermitteln:

- BYTE und UBYTE - 1 Byte (8 bit)
- SHORT und USHORT - 2 Byte (16 bit)
- INTEGER, UINTEGER, LONG und ULONG - 4 Bytes (32 bit)
- LONGINT und ULONGINT - 8 Bytes (64 bit)
- STRINGs (fester und variabler Länge) - lässt sich mit LEN ermitteln
- ZSTRINGs (fester Länge) - gelesen werden SIZEOF(ZstringVariable) - 1 Bytes. Ein Byte wird als Terminierungs-Byte benötigt.
- ZSTRING PTR - werden wie ALLOCATE-Puffer behandelt (siehe unten).
- WSTRINGs (fester Länge) - gelesen werden wie bei ZSTRINGs SIZEOF(WstringVariable) - 1 Bytes. Ein Byte wird als Terminierungs-Byte benötigt.
- ALLOCATE-Puffer - Der Parameter 'Menge' gibt an, wie viele Bytes gelesen werden. Achten Sie darauf, dass genügend Speicherplatz reserviert wurde!
- Arrays - Indexzahl * SIZEOF(Datentyp)
- UDTs - lässt sich mit SIZEOF ermitteln

Im RANDOM-Modus wird jeweils ein Datensatz eingelesen. Die Länge eines Satzes wird bereits beim Öffnen der Datei festgelegt. Siehe dazu OPEN. RANDOM-Daten werden kaum verwendet, da der BINARY-Modus vielseitigere Zugriffsmöglichkeiten bietet.

Wenn das Dateiende erreicht wird, bevor die gewünschte Datenmenge eingelesen wurde, wird der Lesevorgang einfach vorzeitig abgebrochen. Die Stellen des Lesepuffers ('Variable'), die nicht aus der Datei befüllt werden können, erhalten den Wert 0, unabhängig davon, welcher Wert zuvor gespeichert war.

Beispiel 1:
Drei mal vier Bytes aus der Datei 'file.ext' einlesen:

```
DIM FixedLenBuffer AS STRING * 4
DIM VarLenBuffer AS STRING
DIM IntegerBuffer AS INTEGER
VarLenBuffer = SPACE(4)

Dim As Integer f = Freefile
Open "file.ext" For Binary As "hlkw0">Get "hlzeichen">,  , FixedLenBuffer
Get "hlzeichen">, 1, VarLenBuffer
Get "hlzeichen">, 1, IntegerBuffer
Close "reflinkicon" href="temp0423.html">UDTs zu lesen, geben Sie einfach
nur den Bezeichner ohne Index bzw. Verweis auf einen Record an.
```

Beispiel 2:

```
Type UDT
   a As Integer
   b As Double
   c As String * 5
End Type

Dim inpArray(5) As Integer
Dim inpUDT As UDT

Dim As Integer f = Freefile
Open "file.ext" For Binary As "hlkw0">Get "hlzeichen">, , inpArray()
Get "hlzeichen">, , inpUDT

Close "reflinkicon" href="temp0423.html">TYPE (UDT)).

Bei der Verwendung von Datenpuffern, deren Pointer angegeben ist,
verwenden Sie die Pointer-Dereferenzierung
*Pointer
```

oder

```
Pointer[index]
```

Außerdem müssen Sie angeben, wie viele Bytes gelesen werden sollen. Dazu dient der Parameter 'Menge'.

Beispiel 3:

```
Dim As Integer f = Freefile
Open "file.ext" For Binary As "hlkw0">Dim x As Byte Ptr
x = Allocate(8)

Get "hlzeichen">, 1, *x  , 4  ' 4 Bytes in die ersten vier
Speicherstellen lesen
Get "hlzeichen">, 5, x[4], 4  ' 4 Bytes in die nächsten vier
```

```
Speicherstellen lesen

DeAllocate x
Close "hlkw0">Dim v1 As Byte, v2 As String * 2
Dim  f As integer

v1 = 33
v2 = "33"
f = FreeFile

Open "file.ext" For Binary As "hlkw0">Put "hlzeichen">, , v1
   Put "hlzeichen">, , v2
Close "hlkw0">Open "file.ext" For Binary As "hlkw0">Get "hlzeichen">, ,
v2
   Get "hlzeichen">, , v1
Close "hlkw0">Print v1, v2
Sleep
```

Ausgabe:

```
51     !3
```

Wie Sie sehen, wird in die Datei beim ersten Zugriff zuerst ein BYTE-Wert und **anschließend** ein 2-Byte-STRING geschrieben. Beim zweiten Zugriff wird zuerst ein 2-Byte-STRING und anschließend ein BYTE-Wert eingelesen. Ergebnis ist, dass nicht - wie anzunehmen - wieder zwei mal die Ausgabe '33' erfolgt, sondern nur Datenmüll auf dem Bildschirm erscheint. Obwohl die Informationen fehlerfrei gelesen wurden, kann mit den Informationen nicht gearbeitet werden, da sie nach dem Lesen auf die falsche Art und Weise behandelt werden. Werden beim Lesezugriff die beiden GET-Zeilen vertauscht, so erfolgt die korrekte Ausgabe:

```
33     33
```

Die Ursache hierfür liegt darin, dass im BINARY- und RANDOM-Modus (mit denen GET und PUT ja arbeiten) die Daten nicht in einem für Menschen lesbaren Format abgelegt werden, sondern binär behandelt werden, d. h. so geschrieben werden, wie sie vom Prozessor behandelt werden. Hierbei existieren Unterschiede zwischen der Behandlungsweise von Zeichenketten (des STRINGs) und Zahlen (des BYTE-Werts).

Unterschiede zu QB:

- GET kann in FreeBASIC auch als Funktion eingesetzt werden.
- In FreeBASIC können ganze Arrays und UDTs eingelesen werden. Auch die Verwendung von ALLLOCATE-Datenpuffern ist möglich.
- Mit dem Parameter 'Gelesen' kann in FreeBASIC die tatsächlich gelesene Datenmenge bestimmt werden.

Unterschiede zu früheren Versionen von FreeBASIC:

- Seit FreeBASIC v0.20 steht der fünfte Parameter 'Gelesen' zur Verfügung, um die Anzahl der tatsächlich gelesenen Byte zu bestimmen.
- Seit FreeBASIC v0.13 kann GET als Funktion eingesetzt werden
- Seit FreeBASIC v0.10 können ganze Arrays gelesen werden.

Siehe auch:
GET (Grafik), OPEN, BINARY, RANDOM, PUT "reflinkicon" href="temp0596.html">Dateien (Files)

Letzte Bearbeitung des Eintrags am 27.12.12 um 00:50:44
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

GET (Grafik)

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » G » **GET (Grafik)**
Syntax: GET [QuellPuffer], [STEP] (x1, y1) - [STEP] (x2, y2), ZielPuffer
Typ: Anweisung
Kategorie: Grafik

GET speichert einen Ausschnitt aus einem Grafikfenster (bzw. vom Bildschirm bei Vollbildmodi) in einem Bildpuffer.

- 'QuellPuffer' ist ein Speicherbereich wie ein mit IMAGECREATE erstellter Puffer oder ein Array. Beide können mit PUT (Grafik) angezeigt werden. Wird 'QuellPuffer' ausgelassen, liest FreeBASIC direkt vom Bildschirm.
- '(x1, y1)' und '(x2, y2)' sind die Koordinaten der Eckpunkte eines Rechtecks, dessen Inhalt kopiert werden soll. Dabei steht die erste Koordinate für die linke obere Ecke und die zweite für die rechte untere Ecke. Wenn der angegebene Bereich (teilweise) außerhalb der Bildschirmgrenzen liegt, wird der Fehler 'unzulässiger Funktionsaufruf' ('illegal function call') ausgelöst. Wird dieser Fehler nicht durch ON ERROR behandelt, führt dies zum Programmabsturz.
- 'STEP' bewirkt, dass die angegeben Koordinaten relativ zur Position des Grafik-Cursors sind. Dabei kann für jedes Koordinatenpaar eine eigene STEP-Klausel gesetzt werden.
- 'ZielPuffer' ist ein Bildpuffer, in dem der Bildschirmausschnitt gespeichert wird. Ebenso wie 'QuellPuffer' handelt es sich hierbei um einen Bildpuffer, wie er mit IMAGECREATE verwendet wird.

GET wird benutzt, um einen Bildschirmausschnitt zu speichern, um ihn später mit PUT (Grafik) zu verwenden. Die Koordinaten werden von den letzten WINDOW- und VIEW-Anweisungen beeinflusst und müssen beide innerhalb der aktuellen Fenstergrenzen (bzw. Bildschirmgrenzen) liegen.

Der angegebene Puffer muss groß genug sein, um den Bildschirmausschnitt speichern zu können. Die dafür benötigte Größe hängt von der Größe des Bildschirmausschnitts und der verwendeten Farbtiefe ab. IMAGECREATE erzeugt automatisch einen Puffer, der ein Bild angegebener Größe aufnehmen kann. Siehe auch Interne Pixelformate zur manuellen Berechnung des Speicherplatzbedarfs.

Beispiel:
Das folgende Beispiel zeichnet einen Spielstein, der mit der Maus über einen mehrfarbigen Hintergrund bewegt wird. GET wird verwendet, um den aktuellen Hintergrund unter der Spielstein-Position zu speichern, um ihn nach einer Bewegung des Steins wieder herstellen zu können.

```
"hlzahl">3.141592653589793
SCREENRES 300, 200, 32                        ' Grafikscreen mit 32bit
Farbtiefe
DIM AS ANY PTR bild, hg
DIM AS UINTEGER hell = RGB(255, 64, 64)       ' heller Farbwert des Steins
DIM AS UINTEGER dunkel = RGB(192, 0, 0)       ' dunkler Farbwert des Steins

' Bild in den Puffer schreiben
bild = IMAGECREATE(40, 40)
hg = IMAGECREATE(40, 40)
CIRCLE bild, (20, 25), 15, dunkel, PI, 0, .6
LINE bild, (5, 20)-step (0, 5), dunkel
LINE bild, (35, 20)-step (0, 5), dunkel
CIRCLE bild, (20, 20), 15, dunkel, , , .6
PAINT bild, (20, 30), dunkel, dunkel
PAINT bild, (20, 20), hell, dunkel
```

```
' Hintergrund erstellen
LINE (50, 50)-(250, 150), RGB(0,255,0), BF  ' gruenes Rechteck ...
LINE (80, 80)-(220, 120), RGB(0,0,255), BF  ' ... und darin ein blaues

DIM AS INTEGER mausX = 0, mausY = 0, mausB  ' Mausposition und
Buttonstatus
DIM AS INTEGER altX = 0, altY = 0           ' zuletzt gemerkte
Mausposition
SETMOUSE mausX, mausY, 0, 1                 ' Maus auf das Fenster
beschraenken
GET (altX, altY)-step(39, 39), hg           ' Hintergrund speichern
PUT (mausX, mausY), bild, TRANS
DO
  GETMOUSE mausX, mausY, , mausB            ' neue Position ermitteln ...
  IF mausX > 260 THEN mausX = 260           ' ... und an den Grenzen
anpassen
  IF mausy > 160 THEN mausY = 160
  IF mausX <> altX OR mausY <> altY THEN    ' Maus wurde bewegt
    SCREENLOCK
    PUT (altX, altY), hg, PSET              ' alte Position
wiederherstellen
    GET (mausX, mausY)-step(39, 39), hg     ' Hintergrund speichern
    PUT (mausX, mausY), bild, TRANS         ' neue Position zeichnen
    SCREENUNLOCK
    altX = mausX                            ' neue Position merken
    altY = mausY
  END IF
  SLEEP 1
LOOP UNTIL mausB > 0 OR INKEY = CHR(27)     ' bei Mausklick oder ESC
beenden
IMAGEDESTROY bild                           ' Bildpuffer freigeben
IMAGEDESTROY hg
```

Unterschiede zu QB:

- Im Gegensatz zur QB-Version verkleinert sich in FreeBASIC die benötigte Feldgröße nicht, wenn Sie eine geringere Farbtiefe als 8 bpp verwenden.
- Wenn Sie multidimensionale Arrays als Bildpuffer verwenden, achten Sie darauf, dass FreeBASIC die Daten in Zeilen-Reihenfolge speichert (QB speichert in Spalten-Reihenfolge). Was Sie also in QB mit PUT (100, 100), sprites(0, 7) implementiert haben, muss in FreeBASIC PUT (100, 100), sprites(7, 0) lauten.
- FreeBASIC erlaubt neben Arrays auch andere Formen von Bildpuffern.
- Der interne Aufbau eines Bildpuffers in FreeBASIC unterscheidet sich von dem, der in QB genutzt wird.

Unterschiede zu früheren Versionen von FreeBASIC:

- Seit FreeBASIC v0.17 benutzt GET das neue Bildpuffer-Format; siehe dazu Interne Pixelformate.
- Die Möglichkeit, einen Quellpuffer anzugeben, existiert seit FreeBASIC v0.15.
- Seit FreeBASIC v0.13 wird ein Laufzeitfehler ausgelöst, wenn versucht wird, Bereiche einzulesen, die außerhalb der Bildschirmgrenzen liegen.
- Die Möglichkeit, anstelle von Arrays auch Pointer als Puffer anzugeben, existiert seit FreeBASIC v0.12.

Unterschiede unter den FB-Dialektformen:

- In der Dialektform -lang fb wird als 'ZielPuffer' die neue Version des Bildpuffers verwendet, der mit einem 32-Byte Image-Header beginnt und dessen Zeilen auf Vielfache von 16 Bytes aufgestockt werden. Der benötigte Speicherplatz beträgt
  speicherplatz = 32 + (((Breite * bpp + &hF) AND NOT &hF) * Hoehe)
- In der Dialektform -lang qb und -lang fblite wird als 'ZielPuffer' die alte QB-Version des Bildpuffers verwendet, der mit einem 4-Byte Image-Header beginnt und dessen Zeilen nicht aufgestockt werden. Der benötigte Speicherplatz beträgt:
  speicherplatz = 4 + (Breite * Hoehe * bpp)

Siehe auch:
GET (Datei), PALETTE GET, PUT (Grafik), IMAGECREATE, SCREENRES, Grafik

Letzte Bearbeitung des Eintrags am 27.12.12 um 00:51:18
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

GETJOYSTICK

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » G » **GETJOYSTICK**
Syntax: GETJOYSTICK (id, buttons[, [x][, [y][, [z][, [r][, [u][, v]]]]]])
Typ: Funktion
Kategorie: Benutzereingabe

GETJOYSTICK gibt die Position des Joysticks oder des Gamepads und den Status seiner Buttons zurück.

- 'id' ist die Nummer des Joysticks. Die Joystick-IDs liegen zwischen 0 und 15.
- 'buttons' enthält ein Bitfeld, das angibt, welche Buttons gedrückt werden und welche nicht; ein gesetztes Bit repräsentiert dabei einen gedrückten Button:
  Bit 0 wird gesetzt, wenn der erste Button gedrückt wird;
  Bit 1 wird gesetzt, wenn der zweite Button gedrückt wird usw.
  Bis zu 32 Buttons werden unterstützt. Siehe dazu BIT.
- 'x' und 'y' geben die aktuelle Position des Joysticks relativ zum Maximalausschlag an. Sie liegen zwischen -1.0 und +1.0.
- 'z', 'r', 'u' und 'v' geben die Positionen zusätzlicher Joystick-Achsen zurück. Sie liegen zwischen -1.0 und +1.0.
- Der Rückgabewert (sofern abgefragt) gibt an, ob die Joystickabfrage erfolgreich war. Er ist entweder 0, wenn der Status erfolgreich abgefragt wurde, oder 1, wenn ein Fehler aufgetreten ist. Fehler können auftreten, wenn ungültige IDs oder Joysticks, die nicht existieren, angegeben werden, oder falls ein Fehler in der Joystick-API vorliegt.

Sobald GETJOYSTICK aufgerufen wird, speichert es den aktuellen Status der Joystick-Achsen und -Buttons in den Speicherstellen, deren Adressen übergeben wurden. Die Buttons-Speicherstelle muss dabei vom Typ INTEGER sein. Die Informationen zu den Achsen werden relativ zum Vollausschlag in SINGLE-Variablen gespeichert. Das Ergebnis -1.0 entspricht dabei einem Vollausschlag nach links (für x) oder nach vorne (für y); +1.0 bedeutet entsprechend einem Vollausschlag nach rechts oder hinten. 0.0 steht bei allen Achsen für die Ruheposition.

Ein Joystick wird immer zumindest die x- und y-Achsen unterstützen, manche Controller bieten aber noch zusätzliche Achsen. Wenn eine Achse nicht unterstützt wird, schreibt FreeBASIC an die entsprechende Speicherstelle den Wert -1000.00. Wenn der Joystick mit der angegebenen ID nicht verfügbar ist, gibt FreeBASIC -1 bei den Buttons zurück und alle Achsen liefern den Wert -1000.00.

Beispiel:

```
SCREENRES 640, 400

DIM AS SINGLE x, y
DIM AS INTEGER buttons, result, i

CONST JoystickID = 0

' Prüfe, ob die Joystick-Abfrage arbeitet:

IF GETJOYSTICK(JoystickID, buttons, x, y) THEN
   PRINT "Joystick existiert nicht, oder ein Fehler ist aufgetreten."
   PRINT
   PRINT "Beliebige Taste drücken um fortzusetzen"
   SLEEP
   END
END IF
```

```
DO
   GETJOYSTICK JoystickID, buttons, x, y
   LOCATE 1,1
   PRINT USING "x: +"; x; y; buttons
   ' Zeige an, welche Buttons gerade gedrückt werden:
   FOR i = 0 TO 26
      IF (buttons AND (1 SHL i)) THEN
         PRINT "Button ";i ;" gedrueckt.      "
      ELSE
         PRINT "Button ";i ;" nicht gedrueckt."
      END IF
   NEXT
' Bei Tastendruck beenden.
LOOP UNTIL LEN(INKEY)
```

Wie man sieht, muss der Funktionswert nicht abgefragt werden; GETJOYSTICK kann auch als Anweisung eingesetzt werden. In diesem Fall entfallen die Klammern, und der Rückgabewert geht verloren.

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC:

- Seit FreeBASIC v0.17 unterstützt GETJOYSTICK acht Achsten.
- GETJOYSTICK existiert seit FreeBASIC v0.14

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht GETJOYSTICK nicht zur Verfügung und kann nur über **__GETJOYSTICK** aufgerufen werden.

Siehe auch:
GETMOUSE, GETKEY, Benutzereingaben

Letzte Bearbeitung des Eintrags am 15.08.12 um 13:36:33
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

GETKEY

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » G » **GETKEY**
Syntax: GETKEY
Typ: Funktion
Kategorie: Benutzereingabe

GETKEY wartet mit der Programmausführung, bis eine Taste gedrückt wird. GETKEY arbeitet also ähnlich wie SLEEP, löscht jedoch die abgefragte Taste nach der Abfrage aus dem Tastaturpuffer.

Der Rückgabewert von GETKEY ist ein INTEGER mit dem Ascii-Code der gedrückten Taste (vgl. INKEY). Bei Sondertasten wie den Funktionstasten (F1, F2, etc.), Pfeiltasten usw. wird ein kombinierter Wert aus dem Erweiterungscode 255 und dem regulären Tastencode zurückgegeben (siehe zweites Beispiel).

Für eine Tastaturabfrage ohne Unterbrechung des Programmablaufs siehe INKEY und MULTIKEY.

Beispiel 1:

```
PRINT "beliebige Taste drücken, um fortzusetzen"
GETKEY
PRINT LEN(INKEY)
SLEEP
```

Beispiel 2: Illustration der Sondertasten

```
Dim As UShort Result
Dim As UByte a, b

Print "Taste druecken."

Result = GetKey

Print "+------++------+"
Print Bin(Result,16) & " (dezimal: " & Result & ")"

a = (Result And &hFF)
b = (Result And &hFF00) Shr 8

Print
Print "Beide Bytes einzeln:"
Print Bin(A,8) & " = " & a
Print Bin(B,8) & " = " & b

Sleep
```

Unterschiede zu QB: neu in FreeBASIC

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht GETKEY nicht zur Verfügung und kann nur über **__GETKEY** aufgerufen werden.

Siehe auch:
SLEEP, INKEY, INPUT (Funktion), MULTIKEY, Benutzereingaben

Letzte Bearbeitung des Eintrags am 15.08.12 um 13:44:33
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

GETMOUSE

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » G » **GETMOUSE**
Syntax: GETMOUSE x, y[, [Rad][, [Buttons][, [Clip]]]]
Typ: Funktion
Kategorie: Benutzereingabe

GETMOUSE liefert die Position der Maus und den Status der Buttons, des Mausrads und des Clipping-Status zurück.

- Die Position wird in den Variablen 'x' und 'y' zurückgegeben; beide müssen vom Typ INTEGER sein. Die Koordinaten richten sich nach dem Anzeigemodus: In Grafikmodi geben sie die Position in Pixeln an. In Textmodi wird die Position des Cursors in Zeilen und Spalten ausgegeben. In beiden Fällen entspricht die linke obere Ecke den Koordinaten 0, 0. Änderungen am Koordinatensystem durch VIEW und WINDOW haben keinen Einfluss auf die von GETMOUSE zurückgelieferten Koordinaten.
- 'Rad' gibt den Status des Mausrads zurück: Wenn Sie das Mausrad von sich wegdrehen, wird diese Variable jeweils um den Wert 1 erhöht. Drehen Sie es auf sich zu, wird 'Rad' um 1 vermindert. 'Rad' startet bei jeder Programmausführung mit 0. Möglicherweise unterstützt FreeBASIC nicht immer Mausräder; in diesem Fall wird immer 0 zurückgegeben.
- 'Buttons' gibt an, welche Maustasten gerade gedrückt sind:
  Bit 0 wird gesetzt, wenn die LINKE Maustaste gedrückt wird.
  Bit 1 wird gesetzt, wenn die RECHTE Maustaste gedrückt wird.
  Bit 2 wird gesetzt, wenn die MITTLERE Maustaste gedrückt wird.
  Bit 3 wird gesetzt, wenn die 1. Zusatztaste gedrückt wird.
  Bit 4 wird gesetzt, wenn die 2. Zusatztaste gedrückt wird.
  Siehe dazu BIT.
- 'Clip' speichert den Maus-Clipping-Status. Bei 1 ist die Maus auf das Grafik-Fenster beschränkt; bei 0 ist die Maus nicht beschränkt.
- Wenn sich die Maus außerhalb des Fensters befindet oder keine Maus angeschlossen ist, werden im Grafikmodus alle übergeben Variablen auf -1 gesetzt. Im Konsolenmodus werden die letzten Koordinaten vor dem Verlassen des Fensters zurückgegeben
- Der Rückgabewert der Funktion ist 0 bei Erfolg oder 1 beim Auftreten eines Fehlers (z. B. weil sich die Maus außerhalb des Fensters befindet).

Da beim Bewegen der Maus aus dem Grafikfenster heraus alle Werte auf -1 gesetzt werden und dies im Zusammenhang mit der Button- und Mausrad-Abfrage zu Fehlinterpretationen führen kann, sollte immer der Rückgabewert der Funktion geprüft werden.

Beispiel:

```
Dim As Integer x, y, buttons, Ergebnis

' Grafikfenster 320x200x8 setzen
Screenres 320, 200

Do
  ' Lade Mauskoordinaten ohne Mausrad.
    Ergebnis = GetMouse (x, y, , buttons)
    Locate 1, 1
    If Ergebnis <> 0 Then
      Print "Maus nicht vorhanden/nicht im Fenster"
    Else
      Print Using "Maus-Position: "; x; y;
```

```
      If buttons And 1 Then Print "L";
      If buttons And 2 Then Print "R";
      If buttons And 4 Then Print "M";
      If buttons And 8 Then Print "X1";
      If buttons And 16 Then Print "X2";
      Print Space(5)
    End If
Loop While Inkey = ""
End
```

Unterschiede zu QB: neu in FreeBASIC

Plattformbedingte Unterschiede:

- Unter Win32 kann im Vollbild-Konsolenmodus nicht garantiert werden, dass die Änderungen des Mausrads korrekt erkannt werden.
- Unter DOS hat das Maus-Clipping keine Auswirkung. Außerdem wird das Mausrad und die mittlere Maustaste nicht von jedem Maustreiber unterstützt.

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht GETMOUSE nicht zur Verfügung und kann nur über **__GETMOUSE** aufgerufen werden.

Unterschiede zu früheren Versionen von FreeBASIC:

- Maus-Clipping wird seit FreeBASIC v0.18 unterstützt.
- Seit FreeBASIC 0.14 wird GETMOUSE im Konsolenfenster unterstützt.

Siehe auch:
SCREENRES, SETMOUSE, Benutzereingaben

Letzte Bearbeitung des Eintrags am 13.05.12 um 23:16:43
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

GOSUB

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » G » **GOSUB**
Syntax: GOSUB label
Typ: Anweisung
Kategorie: Programmablauf

GOSUB springt zu einem Label und speichert, von wo der Sprung ausgeführt wurde, um später dorthin zurückzukehren. Um zur gespeicherten Stelle zurückzukehren, wird der Befehl RETURN verwendet.

GOSUB ist der Vorgänger von SUB und besitzt eine ähnliche Funktionsweise. Jedoch ist es mit SUB leichter, strukturierten Code zu verfassen. Daher steht GOSUB **nur in den Dialektformen -lang fblite oder -lang qb** zur Verfügung. Unter -lang fblite muss die Unterstützung von GOSUB zudem über OPTION GOSUB aktiviert werden.

Beispiel:

```
"hlstring">"fblite"
OPTION GOSUB
GOSUB message
END

message:
PRINT "Welcome!"
RETURN
```

Unterschiede zu QB:
In der Dialektform -lang qb verhält sich GOSUB wie in QB. Zu den anderen Dialektformen siehe 'Unterschiede unter den FB-Dialektformen'.

Unterschiede unter den FB-Dialektformen:

- GOSUB steht nur in den Dialektformen -lang fblite und -lang qb zur Verfügung.
- In der Dialektform -lang fblite muss GOSUB mittels OPTION GOSUB aktiviert werden.

Siehe auch:
ON ... GOSUB, RETURN, GOTO, SUB, FUNCTION, Prozeduren, Programmablauf

Letzte Bearbeitung des Eintrags am 27.12.12 um 00:52:12
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

GOSUB (Schlüsselwort)

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » G » **GOSUB (Schlüsselwort)**
Syntax: OPTION GOSUB
Typ: Schlüsselwort
Kategorie: Programmoptionen

OPTION GOSUB aktiviert die Unterstützung von GOSUB und ON ... GOSUB. Die Option kann **nur bis FreeBASIC v0.16** eingesetzt werden, oder in entsprechend höheren Versionen, die mit der Kommandozeilenoption -lang deprecated compiliert wurden! Wird mit FreeBASIC v0.17 unter der Option -lang fb compiliert, so ist OPTION GOSUB nicht mehr zulässig!

Da RETURN sowohl eine Rückkehr von GOSUB als auch von einer Prozedur bedeuten kann, können OPTION GOSUB und OPTION NOGOSUB verwendet werden, um GOSUB zu aktivieren bzw. deaktivieren. Wenn GOSUB deaktiviert wurde, wird RETURN nur als Rückkehr aus einer Prozedur bzw. Funktion aufgefasst. Ansonsten wird RETURN nur als Rückkehr von GOSUB aufgefasst.

Beispiel:

```
"hlstring">"fblite"

' GOSUB aktivieren
Option GoSub

GoSub unten
oben:
    Print "wieder zurueck"
    End

unten:
    Print "unten"
    Return
```

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC:
Die Option ist nur bis FreeBASIC v0.16 erlaubt. Seit FreeBASIC v0.17 ist diese Option nur noch zulässig, wenn mit der Kommandozeilenoption -lang deprecated compiliert wurde.

Siehe auch:
NOGOSUB (Schlüsselwort), GOSUB, RETURN, __FB_OPTION_GOSUB__, OPTION, Verschiedenes

Letzte Bearbeitung des Eintrags am 19.01.13 um 16:42:05
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

GOTO

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » G » **GOTO**
Syntax: GOTO label
Typ: Anweisung
Kategorie: Programmablauf

GOTO springt zu einem beliebigen Label. Ein Label ist eine Zeichenfolge mit einem abschließenden Doppelpunkt. Die Zeichenfolge darf kein Befehl, Name einer SUB/FUNCTION oder einer bereits bestehenden Variable sein. In der Dialektform -lang qb und -lang deprecated darf als Label auch eine Zahl verwendet werden, der kein Doppelpunkt folgen muss.

Für den Namen von Labels gelten dieselben Regeln wie für die Namen von Variablen. Jedes Label darf nur einmal vergeben werden. Es ist ebenfalls unzulässig, denselben Namen innerhalb zweier Module doppelt zu verwenden. Eine GOTO-Anweisung bezieht sich immer auf ein Label innerhalb derselben SUB/FUNCTION desselben Moduls; es ist also nicht erlaubt, mit GOTO in eine Prozedur hinein- oder aus ihr herauszuspringen.

Beispiel:

```
"hlstring">"deprecated"
1 GOTO 4
2 PRINT "Tschüss!"
3 GOTO ende
4 PRINT "Willkommen!"
5 GOTO 2

ende:
END
```

Mit GOTO ist es sehr leicht, sogenannten Spaghetticode zu erzeugen, der sich nur noch schwer warten lässt. Daher sollte man in der Regel versuchen, den Befehl zu vermeiden. Ersatz dafür bieten Strukturen wie Schleifen, SUBs und FUNCTIONs.

Siehe auch:
GOSUB, DO ... LOOP, FOR ... NEXT, SUB, FUNCTION, EXIT, Prozeduren, Programmablauf

Letzte Bearbeitung des Eintrags am 27.12.12 um 00:53:04
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

HEX

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » H » **HEX**
Syntax: HEX[$] (Ausdruck [, Stellen])
Typ: Funktion
Kategorie: Stringfunktionen

HEX gibt den hexadezimalen Wert eines beliebigen numerischen Ausdrucks als STRING zurück. Hexadezimale Zahlen enthalten Ziffern aus dem Bereich 0-F (0123456789ABCDEF).

- 'Ausdruck' ist eine positive Ganzzahl (eine Zahl ohne Nachkommastellen und ohne Vorzeichen), die ins Hexadezimalformat übersetzt werden soll. Negative Zahlen werden intern umgewandelt, sodass scheinbar "unerwartete" Ergebnisse auftreten.
- 'Stellen' ist die Anzahl der Stellen, die dafür aufgewandt werden soll. Ist 'Stellen' größer als die benötigte Stellenzahl, wird der Rückgabewert mit führenden Nullen aufgefüllt; der zurückgegebene Wert ist jedoch nie länger, als maximal für den Datentyp von 'Ausdruck' benötigt wird.
  Ist 'Stellen' kleiner als die benötigte Stellenzahl, werden nur die hinteren Zeichen des Rückgabewerts ausgegeben.
  Wird 'Stellen' ausgelassen, besteht der Rückgabewert aus so vielen Zeichen, wie benötigt werden, um die Zahl korrekt darzustellen.
- Der Rückgabwert ist ein String, der den Wert von 'Ausdruck' im Hexadezimalformat enthält.

Das Dollarzeichen ($) als Suffix ist optional.

Beispiel:

```
PRINT HEX(54321)     ' Ausgabe: D431
PRINT HEX(255, 4)    ' Ausgabe: 00FF
PRINT HEX(70000, 3) ' Ausgabe: 170
```

Um einen hexadezimalen Wert in einen dezimalen zurückzuverwandeln, benutzen Sie VALINT. Damit VALINT den nachfolgenden STRING als Hexadezimalwert behandelt, muss ihm ein "&h" vorausgehen:

```
PRINT VALINT("&hC2")
```

gibt 194 aus.

Unterschiede zu QB:

- In QB kann die Anzahl der ausgegebenen Stellen nicht festgelegt werden.
- Die Größe des zurückgegebenen Strings ist in QB auf 32 Bits bzw. 8 Hexadezimal-Stellen begrenzt.

Unterschiede unter den FB-Dialektformen:

- In der Dialektform -lang qb ist das Suffix $ verbindlich.
- In den Dialektformen -lang fblite und -lang fb ist das Suffix optional.

Siehe auch:
BIN, OCT, VAL, WHEX, Datentypen umwandeln

Letzte Bearbeitung des Eintrags am 27.12.12 um 00:54:22
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

HIBYTE

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » H » **HIBYTE**
Syntax: HIBYTE (Ausdruck)
Typ: Funktion
Kategorie: Speicher

HIBYTE gibt das obere Byte eines Ausdrucks als UINTEGER zurück. HIBYTE hat dieselbe Funktion wie

```
CUNSG(Ausdruck) SHR 8 AND &hFF
```

Beispiel:

```
Dim As Integer foo = &b10000100000 ' = 1056 dezimal
PRINT foo
PRINT Hex(foo, 4)
PRINT HIBYTE(foo)
PRINT CUNSG(foo) SHR 8 AND &hFF
SLEEP
```

Ausgabe:

```
1056
0420
4
4
```

Intern wird HIBYTE folgendermaßen behandelt:

```
"cnf">__HIBYTE in der Dialektform -lang qb existiert seit FreeBASIC
v0.24.
```

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht HIBYTE nicht zur Verfügung und kann nur über **__HIBYTE** aufgerufen werden.

Siehe auch:
HIWORD, LOBYTE, LOWORD, Bit-Operatoren
Letzte Bearbeitung des Eintrags am 27.12.12 um 00:54:57
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

HIWORD

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » H » **HIWORD**
Syntax: HIWORD (Ausdruck)
Typ: Funktion
Kategorie: Speicher

HIWORD gibt das obere Word eines Ausdrucks als UINTEGER zurück. HIWORD hat dieselbe Funktion wie

```
CUNSG(Ausdruck) SHR 16 AND &hFFFF
```

Beispiel:

```
Dim As Integer foo = &b11000000000000000 ' = 98304 dezimal
PRINT HIWORD(foo)
PRINT CUNSG(foo) SHR 16 AND &hFFFF
SLEEP
```

Ausgabe:

```
1
1
```

Intern wird HIWORD folgendermaßen behandelt:

```
"cnf">__HIWORD in der Dialektform -lang qb existiert seit FreeBASIC
v0.24.
```

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht HIWORD nicht zur Verfügung und kann nur über **__HIWORD** aufgerufen werden.

Siehe auch:
HIBYTE, LOBYTE, LOWORD, Bit-Operatoren
Letzte Bearbeitung des Eintrags am 27.12.12 um 00:55:21
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

HOUR

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » H » **HOUR**
Syntax: HOUR (Serial)
Typ: Funktion
Kategorie: Datum und Zeit

Anmerkung:
Um diese Funktion nutzen zu können, muss die Datei datetime.bi in Ihren Quellcode eingebunden werden, z.B. mit INCLUDE. Alternativ können Sie auch die Datei vbcompat.bi einbinden, da dadurch auch automatisch datetime.bi eingebunden wird.

HOUR extrahiert die Stunde einer Serial Number.

- 'Serial' ist ein DOUBLE-Wert, der als Serial Number behandelt wird.
- Der Rückgabewert ist die Stunde, die in der Serial Number gespeichert ist.

Beispiel:
Die aktuelle Stunde aus NOW extrahieren:

```
"hlstring">"vbcompat.bi"
DIM Stunde AS INTEGER
Stunde = HOUR(NOW)
```

Unterschiede zu QB: existiert nur in QBX PDS und in VBDOS.

Unterschiede zu früheren Versionen von FreeBASIC: existiert seit FreeBASIC v0.15.

Siehe auch:
NOW, TIMESERIAL, TIMEVALUE, MINUTE, SECOND, FORMAT, Serial Numbers, Datum und Zeit

Letzte Bearbeitung des Eintrags am 27.12.12 um 00:59:44
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

IF (Meta)

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » I » **IF (Meta)**
Syntax:

```
"hlzeichen">(Bedingung)
   '...Programmcode
["hlzeichen">(Bedingung)
   ...]
["hlzahl">...]
"reflinkicon" href="temp0191.html">IF...THEN. Der Unterschied liegt
darin, dass IF...THEN den Code lediglich nicht ausführt, wenn die
Bedingung nicht erfüllt wurde, während er durch #IF gar nicht erst
compiliert wird.
Die Bedingung darf nur aus solchen Ausdrücken und Befehlen (z. B.
LBOUND/UBOUND auf fixe Arrays) bestehen, die zur Compile-Zeit bekannt
sind. Ebenso eingesetzt werden können #DEFINE-Symbole sowie die
FB-eigenen vordefinierten Symbole.

Dieser Metabefehl ermöglicht es z. B., je nach Betriebssystem passende
Header einzubinden (siehe #INCLUDE) oder Variablen zu setzen. Dadurch
wird die Portabilität zu verschiedenen Systemen gewährleistet.

Wenn die Bedingung nach #IF erfüllt (wahr) ist, wird der folgende Code
compiliert. Die Befehle nach #ELSEIF und #ELSE werden übergangen.
Andernfalls werden nacheinander die Bedingungen hinter den #ELSEIFs
geprüft. Ist keine Bedingung wahr, wird der Code hinter #ELSE compiliert.
```

Beispiel:

```
"hlkw0">DEFINED(__FB_WIN32__)
  "hlstring">"ProjectWin.bi"
"hlkw0">DEFINED(__FB_LINUX__)
  "hlstring">"ProjectLinux.bi"
"hlkw0">"hlstring">"Plattform wird nicht unterstützt"
"reflinkicon" href="temp0192.html">#IFDEF, #IFNDEF, IF...THEN,
Präprozessoren, Bedingungsstrukturen, Präprozessor-Anweisungen
```

Letzte Bearbeitung des Eintrags am 03.07.13 um 00:09:35
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

IF ... THEN

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » I » **IF ... THEN**
Syntax A:

```
IF Bedingung THEN [:] Anweisungen [ELSE Anweisungen] [:] [END IF]
```

Syntax B:

```
IF Bedingung_1 THEN
   ' Anweisungen
[ELSEIF Bedingung_2 THEN
   ' Anweisungen]
[ ... ]
[ELSE
   ' Anweisungen]
END IF
```

Typ: Anweisung
Kategorie: Programmablauf

IF...THEN führt einen Codeteil nur dann aus, wenn eine Bedingung erfüllt ist.

- 'Bedingung' ist ein numerischer Ausdruck, der entweder wahr/erfüllt (ungleich null) oder falsch/nicht erfüllt (gleich null) sein kann.
- 'Anweisungen' steht für einen beliebigen FreeBASIC-Programmcode, der ausgeführt werden soll, wenn die Bedingung erfüllt wurde.

Mit IF...THEN werden einfache Entscheidungen getroffen. Um eine Variable auf viele verschiedene Möglichkeiten hin zu prüfen, wird bevorzugt SELECT CASE verwendet.

Die Bedingungen werden in der Reihenfolge abgearbeitet, in der sie im Code stehen. Der Code hinter ELSE wird abgearbeitet, wenn keine der vorhergehenden Bedingungen wahr ist.

Folgende Bedingungsstrukturen sind möglich:

- ein einfacher Vergleich wie 'x = 5'
- mehrfache Vergleiche, verbunden durch logische Operatoren, wie 'y <> 0 ANDALSO x \ y = 1'
- jede Variable eines numerischen Datentyps oder jede Zahl, wobei der Wert 0 (null) als *FALSE* interpretiert wird, alles andere als *TRUE*.

Der IF...THEN-Block muss mit END IF abgeschlossen werden.

Wenn IF nur als einzeiliger Befehl eingesetzt wird, ist END IF optional; das soll das Portieren von C-Code erleichtern. Alternativ zu END IF kann man auch ENDIF schreiben, also ohne ein Leerzeichen.
Nur in der einzeiligen Version dürfen hinter THEN und ELSE Anweisungen folgen.
Als Block-Anweisung (Syntax B) initialisiert IF...THEN seit FreeBASIC v0.16 einen SCOPE-Block, der mit der Zeile END IF endet. Variablen, die innerhalb eines solchen Blocks deklariert werden, existieren außerhalb nicht mehr.

Beispiel: Einfaches Ratespiel

```
RANDOMIZE
DIM AS INTEGE x = INT(RND*10)
```

```
PRINT "Rate die Zahl von 0 und 9)"

DO
   INPUT "Rate..."; y
   IF x = y THEN
      PRINT "richtig!"
      EXIT DO
   ELSEIF x > y THEN
      PRINT "zu klein!"
   ELSE
      PRINT "zu groß!"
   END IF
LOOP
SLEEP
```

Unterschiede zu QB:

- Innerhalb eines IF...THEN-Blocks dürfen in QB keine Variablen definiert werden.
- In FreeBASIC muss in einem einzeiligen IF (Syntax A) END IF benutzt werden, wenn direkt hinter dem THEN ein Doppelpunkt (:) folgt.
- Ein einzeiliges IF darf in QB nicht mit END IF enden.

Unterschiede zu früheren Versionen von FreeBASIC:

- Seit FreeBASIC v0.16 wirkt ein IF...THEN-Block wie ein SCOPE-Block
- Seit FreeBASIC v0.15 muss END IF benutzt werden, wenn direkt hinter dem THEN ein Doppelpunkt (:) folgt.

Unterschiede unter den FB-Dialektformen:

- In -lang fb und -lang fblite können Doppelpunkte (:) statt Zeilenumbrüchen verwendet werden, um mehrzeilige Bedingungsstrukturen zu erzeugen.
- Wird in diesen Dialekten nach THEN ein Doppelpunkt oder ein Kommentarzeichen (REM oder ') gesetzt, so wird es als mehrzeilige Bedingungsstruktur interpretiert, alles andere wird als einzeiliges IF behandelt.

Siehe auch:
IF (Meta), SELECT CASE, IIF, DO ... LOOP, Ausdrücke und Operatoren, Bedingungsstrukturen

Letzte Bearbeitung des Eintrags am 27.12.12 um 13:20:27
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

IFDEF (Meta)

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » I » **IFDEF (Meta)**
Syntax: #IFDEF Symbol
Typ: Metabefehl
Kategorie: Metabefehle

#IFDEF Symbol ist die verkürzte Form von

```
#IF DEFINED(Symbol)
```

Symbol darf jede Art von Symbol sein, auch eines, das mit CONST erstellt wurde.

Unterschiede zu QB: neu in FreeBASIC

Siehe auch:
#IFNDEF, #IF, Präprozessoren, Präprozessor-Anweisungen

Letzte Bearbeitung des Eintrags am 27.12.12 um 13:20:57
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

IFNDEF (Meta)

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » I » **IFNDEF (Meta)**
Syntax: #IFNDEF Symbol
Typ: Metabefehl
Kategorie: Metabefehle

#IFNDEF Symbol ist die verkürzte Form von

```
#IF NOT DEFINED(Symbol)
```

Symbol darf jede Art von Symbol sein, auch eines, das mit CONST erstellt wurde.

Unterschiede zu QB: neu in FreeBASIC

Siehe auch:
#IFDEF, #IF, Präprozessoren, Präprozessor-Anweisungen

Letzte Bearbeitung des Eintrags am 27.12.12 um 13:21:17
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

IIF

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » I » **IIF**
Syntax: IIF (Bedingung, Ausdruck_wenn_wahr, Ausdruck_wenn_falsch)
Typ: Funktion
Kategorie: Programmablauf

IIF liefert einen von zwei Werten zurück, abhängig davon, ob die Bedingung erfüllt ist oder nicht.

- 'Bedingung' ist ein Ausdruck oder Vergleich, der entweder gleich oder ungleich Null sein kann. Ist der Wert des Ausdrucks gleich Null, so ist die Bedingung nicht erfüllt.
- 'Ausdruck_wenn_wahr' wird als Ergebnis der Funktion zurückgegeben, wenn die Bedingung erfüllt wurde.
- 'Ausdruck_wenn_falsch' wird als Ergebnis der Funktion zurückgegeben, wenn die Bedingung nicht erfüllt wurde.

IIF liefert einen Wert, abhängig vom Ergebnis der Bedingungsauswertung. Die typische Anwendung ist mitten in einem Ausdruck; so wird vermieden, dass ein Bedingungsausdruck unnötig aufgeteilt werden muss.

IIF berechnet den Ausdruck nur, bis das Ergebnis feststeht. Das spart Zeit und kann auch verhindern, dass Ausdrücke ausgewertet werden, die ungültige Ergebnisse nach Auswertung von Bedingung hätten.

Beispiel 1:

```
DIM AS INTEGER a, b, x, y, z
a = (x + y + IIF(b > 0, 4, 7)) \ z
```

Dieser Code ist äquivalent zu:

```
DIM AS INTEGER a, b, x, y, z, temp
IF b > 0 THEN temp = 4 ELSE temp = 7
a = (x + y + temp) \ z
```

Es obliegt dem Programmierer, zu entscheiden, welche Form die günstigere ist.

Beispiel 2: Zufallsbedingt a oder b ausgeben

```
DIM AS INTEGER a, b
PRINT IIF (RND < .5, a, b)
```

Beispiel 3:
Bis einschließlich FreeBASIC v0.24 mussten alle Parameter numerischer Natur sein. Dies schließt auch Pointer ein, sodass über einen Umweg auch mit STRINGs gearbeitet werden konnte.

```
' Verwendung von Strings bis fbc 0.24
DIM AS INTEGER a, b
PRINT "Ist a > b? Antwort: "; *IIF(a > b, @"Ja", @"Nein")
SLEEP
```

Seit FreeBASIC v0.90 sind STRINGs und UDTs problemlos nutzbar.

```
' Verwendung von Strings ab fbc 0.90
DIM AS INTEGER a, b
PRINT "Ist a > b? Antwort: "; IIF(a > b, "Ja", "Nein")
```

SLEEP

Unterschiede zu QB: neu in FreeBASIC

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht IIF nicht zur Verfügung und kann nur über **__IIF** aufgerufen werden.

Unterschiede zu früheren Versionen von FreeBASIC:

- Seit FreeBASIC v0.90 können auch STRINGs und UDTs verwendet werden; davor mussten alle Ausdrücke numerisch sein.
- IIF existiert seit FreeBASIC v0.12

Siehe auch:
IF...THEN, Ausdrücke und Operatoren, Bedingungsstrukturen

Letzte Bearbeitung des Eintrags am 01.07.13 um 23:33:40
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

IMAGECONVERTROW

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » I » **IMAGECONVERTROW**
Syntax: IMAGECONVERTROW Quelle, Quell_bpp, Ziel, Ziel_bpp, Breite [, IstRGB]
Typ: Anweisung
Kategorie: Grafik

IMAGECONVERTROW kopiert eine bestimmte Anzahl von Pixeln von einem Grafikpuffer in einen anderen und konvertiert dabei die Anzahl der Bits pro Pixel in der Kopie auf einen gewünschten Wert.

- 'Quelle' ist ein Pointer vom Typ ANY PTR auf die Adresse, an der die Bildzeile beginnt, die konvertiert werden soll. Die Quelle kann entweder ein Bildpuffer im High-/Truecolor-Format sein (mit 24 oder 32 bpp), oder ein Bildpuffer im palettenindizierten Format mit ein bis acht bpp. Die Konversion eines palettenindizierten Bildes wird nur dann korrekte Ergebnisse erzielen, wenn Sie in einem Bildschirmmodus arbeiten, der die für das Bild korrekte Palette benutzt.
- 'Quell_bpp' ist die Anzahl der Bits pro Pixel in der Quelle. Zulässig sind die Werte 1-8, 24 und 32.
- 'Ziel' ist ein Pointer vom Typ ANY PTR auf den Beginn der Zeile, in die das konvertierte Bild geschrieben werden soll. Der Pointer darf auf einen Bereich innerhalb eines Bildpuffers für ein 16bpp oder einen 32bpp-Modus zeigen. Nur wenn das Quellbild ein palettenindiziertes Bild ist, kann diese Adresse auch auf einen Bildpuffer für ein Bild mit 1-8bpp verweisen.
- 'Ziel_bpp' ist die Anzahl der Bits pro Pixel im Zielpuffer. Zulässig sind die Werte 1-8, 16 und 32.
- 'Breite' ist die Breite einer Zeile in Pixel, oder auch die Anzahl der Pixel des gesamten Bildes, berechnet nach Breite * Höhe. Auf diese Art kann ein ganzes Bild in einem Schritt konvertiert werden; dies ist jedoch nur möglich, wenn die Bildinformation in einem Stück abgespeichert ist und nicht durch Padding-Bytes unterbrochen wird. Während frühere FreeBASIC-Versionen einen QB-artigen Bildpuffer ohne Padding verwendeten, werden die Zeilen in einem aktuellen Bildpuffer auf ein Vielfaches von 16 Byte gepaddet; vgl. dazu GET (Grafik), "Unterschiede unter den FB-Dialektformen".
- 'IstRGB' wird verwendet, um bei High-/Truecolor-Bildern (ab einer Farbtiefe von 15bpp) die Reihenfolge der Rot-, Grün- und Blauanteil-Bytes festzulegen. Wird hier null angegeben, so vertauscht FreeBASIC während der Konversion die Position des Rot- und Blaukanals; die Reihenfolge der Teilfarbinformationen wird also umgekehrt. Wird hier ein Wert ungleich null angegeben oder wird der Parameter ausgelassen, so behält FreeBASIC die Reihenfolge der Farbkanäle bei, so wie es in 'Quelle' vorgegeben wurde.

Beispiel:

```
"hlstring">"fbgfx.bi"
"hlkw0">__FB_LANG__ = "fb"
Using FB
"hlkw2">Const As Integer w = 64, h = 64
Dim As IMAGE Ptr img8, img32
Dim As Integer x, y

' erstelle ein 32-bit-Image in der Größe w*h:
ScreenRes 1, 1, 32, , GFX_NULL
img32 = ImageCreate(w, h)
If img32 = 0 Then Print "IMAGECREATE ist bei img32 fehlgeschlagen!":
Sleep: End

' erstelle ein 8-bit-Image in der Größe w*h:
ScreenRes 1, 1, 8, , GFX_NULL
img8 = ImageCreate(w, h)
If img8 = 0 Then Print "IMAGECREATE ist bei img8 fehlgeschlagen!": Sleep:
```

```
End

' fülle das 8-bit-Image mit einem Muster
For y = 0 To h - 1
    For x = 0 To w - 1
        PSet img8, (x, y), 56 + (x + y) Mod 24
    Next x
Next y

' öffne ein Grafikfenster im 8-bit-Modus und gib dort das Bild aus
ScreenRes 320, 200, 8
WindowTitle "8-bit-Farbmodus"
Put (10, 10), img8
Sleep

' kopiere die Daten in ein 32-bit-Image
Dim As Byte Ptr p8, p32
Dim As Integer pitch8, pitch32

"hlkw0">ImageInfo ' ältere Versionen von FreeBASIC besitzen keine
Funktion IMAGEINFO
"hlzeichen">(img_)
IIf(img_->type=PUT_HEADER_NEW,img_->pitch,img_->old.width*img_->old.bpp)
"hlzeichen">(img_) CPtr(Byte
Ptr,img_)+IIf(img_->type=PUT_HEADER_NEW,SizeOf(PUT_HEADER),SizeOf(_OLD_HEADER))
pitch8 = GETPITCH(img8): p8 = GETP(img8)
pitch32 = GETPITCH(img32): p32 = GETP(img32)
"hlkw0">ImageInfo( img8,  , , , pitch8,  p8  )
ImageInfo( img32, , , , pitch32, p32 )
"hlkw0">For y = 0 To h - 1
    ImageConvertRow(@p8 &"hlzeichen">* pitch8 ],  8, _
                    @p32[ y * pitch32], 32, _
                    w)
Next y

' öffne ein Grafikfenster im 32-bit-Modus und gib dort das Bild aus:
ScreenRes 320, 200, 32
WindowTitle "32-bit-Farbmodus"
Put (10, 10), img32

Sleep

' Speicher freigeben
ImageDestroy img8
ImageDestroy img32
```

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC: existiert erst seit FreeBASIC v0.16

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht IMAGECONVERTROW nicht zur Verfügung und kann nur über **__IMAGECONVERTROW** aufgerufen werden.

Siehe auch:
IMAGECREATE, IMAGEDESTROY, IMAGEINFO, GET (Grafik), PUT (Grafik), SCREENRES, Grafik

Letzte Bearbeitung des Eintrags am 27.12.12 um 13:22:30
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

IMAGECREATE

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » I » **IMAGECREATE**
Syntax: IMAGECREATE (Breite, Höhe[, [Farbe] [, Farbtiefe]])
Typ: Funktion
Kategorie: Grafik

IMAGECREATE reserviert einen Speicherbereich als Datenpuffer für ein Bild.

- 'Breite' und 'Höhe' sind die Maße des Bildes in Pixel.
- 'Farbe' ist ein optionaler Parameter, der eine Farbnummer angibt, mit der der Puffer bei seiner Erstellung gefüllt sein soll. Wenn dieser Parameter ausgelassen wird, setzt FreeBASIC die Transparenzfarbe des jeweiligen Modus ein.
- 'Farbtiefe' gibt die Farbtiefe des erstellten Puffers in Bits pro Pixel an. Wird sie ausgelassen, dann wird die mit SCREENRES eingestellte Farbtiefe des Grafikfensters verwendet.
- Der Rückgabewert ist ein Pointer auf den Beginn des reservierten Speicherbereichs.

Diese Funktion reserviert einen Speicherbereich, in dem Pixeldaten gepuffert werden können. Sobald dieser Puffer erstellt wurde, können alle Drawing Primitives (einfachste Grafikfunktionen) darauf zugreifen. Bei seiner Erstellung wird der Speicherbereich mit der angegebenen Farbe oder der Transparenzfarbe des jeweiligen Modus ausgefüllt (für indizierte Grafikmodi ist die Transparenzfarbe 0, für alle anderen ist es RGB(255, 0, 255)). Siehe auch SCREENCONTROL.

IMAGECREATE gibt, ähnlich wie ALLOCATE, einen Pointer auf den reservierten Speicherbereich zurück. Wenn IMAGECREATE verwendet werden soll, um Grafiken zwischen Bildschirm und Puffer auszutauschen, dann funktioniert dies nur, wenn bereits ein Grafikmodus mit SCREENRES oder SCREEN initiiert wurde. Falls IMAGECREATE fehlschlägt, ist das Ergebnis 0.
Bildpuffer, die Sie mit IMAGECREATE erstellt haben, sollten unbedingt mit IMAGEDESTROY entfernt werden, sobald sie nicht mehr benötigt werden, um den Speicherplatz freizugeben.

Der Bildpuffer ist vom Typ Image, der in der Datei fbgfx.bi definiert wird.

```
TYPE Image FIELD = 1
  UNION
    old             AS _OLD_HEADER
    type            AS UINTEGER
  END UNION
  bpp                AS INTEGER
  width              AS UINTEGER
  height             AS UINTEGER
  pitch              AS UINTEGER
  _reserved(1 to 12) AS UBYTE
END TYPE
```

Wenn Sie fbgfx.bi mit "reflinkicon" href="temp0269.html">NAMESPACE FB gehört.
Näheres zum Speicheraufbau des Bildpuffers siehe unter Interne Pixelformate: Struktur eines Bildpuffers für die Drawing Primitives.

Beispiel:

```
"hlkw0">Once "fbgfx.bi"
Dim As FB.Image Ptr img, cut

Screenres 400, 300, 32
```

```
img = Imagecreate(64, 64, RGBA(64, 160, 0, 255))
cut = Imagecreate(32, 32)

If img = 0 OR cut = 0 Then
  Print "Speicher konnte nicht reserviert werden!"
  If img Then ImageDestroy img
  If cut Then ImageDestroy cut
End If

Circle img, (32, 32), 28, Rgba(255, 0, 0, 128), , , , F
Get img, (0, 0)-(31, 31), Cut

Put img, (32, 32), cut, Pset
Put (160, 120), img, Pset
Put (180, 140), img, Alpha

Imagedestroy img
Imagedestroy cut
Sleep
```

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC:

- Seit v0.24 setzt IMAGECREATE eine Fehlernummer, die man per ERR abfragen kann.
- Seit v0.17 stehen zwei Speicherformate für Bildpuffer zur Verfügung. Siehe Interne Pixelformate für Details.
- IMAGECREATE existiert seit FreeBASIC v0.14.

Siehe auch:
IMAGEDESTROY, IMAGEINFO, IMAGECONVERTROW, GET (Grafik), PUT (Grafik), PSET (Grafik), SCREENRES, Interne Pixelformate, Grafik

Letzte Bearbeitung des Eintrags am 20.08.14 um 21:12:02
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

IMAGEDESTROY

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » I » **IMAGEDESTROY**
Syntax: IMAGEDESTROY image
Typ: Anweisung
Kategorie: Grafik

IMAGEDESTROY gibt einen mit IMAGECREATE reservierten Speicher wieder frei. 'image' ist ein Pointer, der auf den freizugebenden Speicherbereich zeigt.

Jeder Speicher, der mit IMAGECREATE erstellt wurde, sollte auch unbedingt wieder mit IMAGEDESTROY freigegeben werden, sobald er nicht mehr benötigt wird.

Beispiel: siehe IMAGECREATE

Unterschiede zu QB: neu in FreeBASIC

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht IMAGEDESTROY nicht zur Verfügung und kann nur über **__IMAGEDESTROY** aufgerufen werden.

Siehe auch:
IMAGECREATE, Grafik

Letzte Bearbeitung des Eintrags am 27.12.12 um 13:25:23
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

IMAGEINFO

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » I » **IMAGEINFO**
Syntax: IMAGEINFO (ImagePtr, [Breite] [, [Höhe] [, [bpp] [, [Pitch] [, [Pixdata] [, [Größe]]]]]])
Typ: Funktion
Kategorie: Grafik

IMAGEINFO gibt Informationen über das mit 'ImagePtr' angesprochene Image zurück.

- 'ImagePtr' ist ein Speicherbereich wie ein mit IMAGECREATE erstellter Puffer, der das abzufragende Image enthält.
- 'Breite' und 'Höhe' geben die Breite und die Höhe des Images in Pixeln zurück.
- 'bpp' enthält die Anzahl der Byte pro Pixel.
- 'Pitch' liefert die Größe einer Image-Zeile in Byte.
- 'Pixdata gibt einen Pointer auf den Anfang des Pixelbereichs zurück.
- 'Größe' liefert die Größe des Images in Byte.
- Der Rückgabewert ist ein INTEGER, das angibt, ob ein Image angelegt ist (Rückgabewert=0) oder nicht (Rückgabewert<>0).

Beispiel:

```
Dim img As Any Ptr, pixdata As Any Ptr, pitch As Integer

' 32bit-Screen und Bildpuffer erzeugen
ScreenRes 320, 200, 32
img = ImageCreate(64, 64)

' Zeilengröße und Anfang des Pixelbereichs ermitteln
imageinfo img, ,,, pitch, pixdata

' Muster direkt in den Datenpuffer schreiben
For y As Integer = 0 To 63
    Dim As UInteger Ptr p = pixdata + y * pitch
    For x As Integer = 0 To 63
        p&"hlzeichen">] = RGB(x * 4, y * 4, (x Xor y) * 4)
    Next x
Next y

' Bild zeichnen und Datenpuffer freigeben
Put (10, 10), img
ImageDestroy img
Sleep
```

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC: existiert seit FreeBASIC v0.20.0

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht IMAGEINFO nicht zur Verfügung und kann nur über **__IMAGEINFO** aufgerufen werden.

Siehe auch:
IMAGECREATE, SCREENINFO, Interne Pixelformate, Grafik

Letzte Bearbeitung des Eintrags am 16.06.13 um 00:02:09
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

IMP

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » I » **IMP**
Syntax A: Ergebnis = Ausdruck1 IMP Ausdruck2
Syntax B: Ausdruck1 IMP= Ausdruck2
Typ: Operator
Kategorie: Operatoren

IMP kann als einfacher (Syntax A) und kombinierter (Syntax B) Operator eingesetzt werden. Syntax B ist eine Kurzform von

```
Ausdruck1 = Ausdruck1 IMP Ausdruck2
```

IMP ist ein bitweiser Operator, der dieselbe Funktion hat wie

```
(NOT a) OR b.
```

Er wird benutzt, um logische Folgerungen auszuwerten; das Ergebnisbit wird nur dann nicht gesetzt, wenn das zweite Operanden-Bit nicht gesetzt ist, während das Erste gesetzt ist.

IMP kann mithilfe von OPERATOR überladen werden.

Beispiel 1: IMP in einer IF-THEN-Bedingung:

```
IF (a = 1) IMP (b = 7) THEN
PRINT "Moeglich sind:"
PRINT "a = 1 und b = 7"
PRINT "a <> 1 aber b = 7"
PRINT "a <> 1 und b <> 7"
ELSE
PRINT "a = 1 aber b <> 7."
END IF
```

Beispiel 2: Verknüpfung zweier Zahlen mit IMP:

```
DIM AS INTEGER z1, z2

z1 = 6
z2 = 10

PRINT z1, BIN(z1, 4)
PRINT z2, BIN(z2, 4)
PRINT "---", "----"
PRINT (z1 IMP z2) AND 15, BIN( (z1 IMP z2) AND 15, 4)
GETKEY
```

Ausgabe:

```
  6           0110
 10           1010
---           ----
 11           1011
```

Anmerkung dazu: Die angewandten AND 15 bewirken, dass nur die letzten vier Bits angezeigt werden. Der Grund dafür ist, dass INTEGER-Variablen aus 32bit bestehen. Die übrigen 28 Bits werden selbstverständlich auch mit IMP verglichen. Würden diese in dieses Beispiel miteinbezogen, würde das Beispiel einen Teil seiner Anschaulichkeit verlieren.

Beispiel 3: IMP als kombinierter Operator

```
DIM AS UBYTE a

a = &B00110011
PRINT BIN(a, 8)
a IMP= &B01010101
PRINT "01010101"
PRINT "--------"

PRINT BIN(a, 8)
GETKEY
```

Ausgabe:

```
00110011
01010101
--------
11011101
```

Unterschiede zu QB:
Kombinierte Operatoren sind neu in FreeBASIC.

Siehe auch:
NOT, AND (Operator), OR (Operator), XOR (Operator), EQV, Bit Operatoren / Manipulationen

Letzte Bearbeitung des Eintrags am 27.12.12 um 13:26:44
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

IMPLEMENTS

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » I » **IMPLEMENTS**
Syntax: TYPE typeName IMPLEMENTS interface
Typ: Anweisung
Kategorie: Klassen

IMPLEMENTS hat bisher keine Funktion, ist als Schlüsselwort aber bereits geschützt. Zukünfig soll der Befehl eine Klasse angeben, die ein Interface (eine Schnittstelle) implementiert.

Beispiel:

```
'noch nichts
```

Unterschiede zu QB: neu in FreeBASIC.

Unterschiede zu früheren Versionen von FreeBASIC: existiert seit FreeBASIC v0.24.

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht IMPLEMENTS nicht zur Verfügung und kann nur über **__IMPLEMENTS** aufgerufen werden.

Siehe auch:
TYPE (UDT), BASE, OBJECT, IS (Vererbung), EXTENDS, Objektorientierung

Letzte Bearbeitung des Eintrags am 27.12.12 um 13:27:35
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

IMPORT

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » I » **IMPORT**
Syntax: EXTERN IMPORT Bezeichner [ALIAS "Aliasname"] [AS Typ]
Typ: Schlüsselwort
Kategorie: Bibliotheken

IMPORT wird unter Win32 zusammen mit EXTERN (Module) verwendet, wenn auf globale Variablen aus DLLs zugegriffen werden muss, da in solchen ein Zugriff über eine implizite Pointer-Dereferenzierung geschieht.

```
' mydll.c: compilieren mit
' gcc -shared -Wl,--strip-all -o mydll.dll mydll.c
__declspec( dllexport ) int MyDll_Data = 0x1234;

' import.bas:
"hlstring">"mydll"
Extern Import MyDll_Data Alias "MyDll_Data" As Integer

Print "&h" + Hex( MyDll_Data )
```

Ausgabe:

```
&h1234
```

Unterschiede zu QB: neu in FreeBASIC

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht IMPORT nicht zur Verfügung und kann nur über **__IMPORT** aufgerufen werden.

Siehe auch:
EXTERN (Module), Module (Library / DLL)

Letzte Bearbeitung des Eintrags am 26.05.12 um 00:52:51
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

INCLIB (Meta)

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » I » **INCLIB (Meta)**
Syntax A: #INCLIB "LibName.a"
Syntax B: '$INCLIB: "LibName.a"
Typ: Metabefehl
Kategorie: Metabefehle

#INCLIB lädt eine Bibliothek (*.a-Datei) in den Link-Vorgang, wodurch dem Programmierer das Compilieren vereinfacht wird (man erspart sich die Angabe der Compiler-Option -l LibName).

- 'LibName.a' ist der Dateiname der einzubindenden Bibliothek, eventuell mit Pfadangabe. Wenn kein Pfad angegeben wird, sucht FreeBASIC zuerst im aktuellen Arbeitsverzeichnis, dann im Verzeichnis, in dem sich der Quellcode befindet, und schließlich in den System-Include-Verzeichnissen nach der einzubindenden Datei.
- Sowohl Slashes (/) als auch Backslashes (\) werden als Pfad-Trennzeichen akzeptiert.

Beispiel:

```
"hlstring">"mylib"
'Dadurch wird mylib.a in den Link-Vorgang eingebunden.
```

Achtung:
Seit FreeBASIC v0.14 werden ESCAPE-Chars in #INCLIB-Anweisungen nicht mehr interpretiert.

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC:

- Die Variante '$INCLIB ist seit FreeBASIC v0.17 nicht mehr zulässig.
- Escape-Chars werden seit FreeBASIC v0.14 in INCLIB-Anweisungen nicht mehr interpretiert.

Siehe auch:
INCLUDE, Präprozessor-Anweisungen

Letzte Bearbeitung des Eintrags am 27.12.12 um 13:28:33
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

INCLUDE (Meta)

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » I » **INCLUDE (Meta)**
Syntax A: #INCLUDE [ONCE] "Datei"
Syntax B: '$INCLUDE [ONCE]: 'Datei'
Typ: Metabefehl
Kategorie: Metabefehle

#INCLUDE bindet Programmcode aus einer anderen Quellcodedatei ein. Dies hat zur Folge, dass der eingebundene Code beim Compilieren so behandelt wird, als stünde er an der Stelle, an der sich im echten Code #INCLUDE befindet. Dadurch lässt sich ein Programm in mehrere Teile zerlegen, wodurch es übersichtlicher wird.

- 'Datei' ist der Dateiname des einzubindenden Codes, eventuell mit Pfadangabe. Wenn kein Pfad angegeben wird, sucht FreeBASIC zuerst im aktuellen Arbeitsverzeichnis, dann im Verzeichnis, in dem sich der Quellcode befindet, und schließlich in den System-Include-Verzeichnissen nach der einzubindenden Datei.
- Sowohl Slashes (/) als auch Backslashes (\) werden als Pfad-Trennzeichen akzeptiert.
- 'ONCE' verhindert, dass Daten doppelt eingebunden werden. Dies kann geschehen, wenn eingebundene Dateien ebenfalls INCLUDE-Befehle enthalten.

Beispiel:

```
"hlkw0">ONCE "vbcompat.bi"
```

Im Gegensatz zu QB können in per "reflinkicon" href="temp0064.html">COMMON/DIM [SHARED]-, DECLARE-, oder TYPE-Anweisungen enthalten) mit der Erweiterung BI zu kennzeichnen. BIs werden oft eingesetzt, um die Prozeduren einer LIB/DLL zu deklarieren.

Das maximale Rekursionslevel liegt bei 16, d. h. dass bei einem Code, der einen weiteren einbindet, der einen weiteren einbindet etc. nach 16 Dateien die Grenze erreicht ist, bei dem der Compiler aufhört zu suchen. Eine solch tiefe Verschachtelung ist allerdings unüblich.

Achtung:
Seit FreeBASIC v0.14 werden ESCAPE-Chars in #INCLUDE-Anweisungen nicht mehr interpretiert.

Unterschiede zu QB:

- Wenn der Pfadname einen Unterstrich enthält, müssen doppelte Anführungsstriche statt einfacher verwendet werden.
- Es können auch 'echte' BAS-Files eingebunden werden, die auch 'ausführbare' Anweisungen enthalten.
- Das Schlüsselwort 'ONCE' steht zur Verfügung.

Unterschiede zu früheren Versionen von FreeBASIC:

- Die Variante '$INCLUDE ist seit FreeBASIC v0.17 nicht mehr zulässig.
- Escape-Chars werden seit FreeBASIC v0.14 in INCLUDE-Anweisungen nicht mehr interpretiert.

Siehe auch:
INCLIB, Präprozessor-Anweisungen

Letzte Bearbeitung des Eintrags am 19.01.13 um 18:10:06
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

INKEY

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » I » **INKEY**
Syntax: INKEY[$]
Typ: Funktion
Kategorie: Benutzereingabe

INKEY gibt einen STRING zurück, der die erste Taste im Tastaturpuffer enthält. Bei Funktionstasten sind meist zwei Zeichen enthalten, z.B. F1: CHR(255) & CHR(59) bzw. CHR(255, 59). Ist der Tastaturpuffer leer, wird ein Leerstring zurückgegeben.

Das Dollarzeichen ($) als Suffix ist optional.

Beispiel 1:

```
PRINT "Druecken Sie 'q', um zu beenden."
DO
  SLEEP 1 'Prozessorauslastung auf ein Minimum senken
LOOP UNTIL INKEY = "q"
```

Beispiel 2:
Bei einer Tastaturabfrage über MULTIKEY oder der Programmunterbrechung durch SLEEP wird der Tastaturpuffer nicht geleert. In solchen Fällen kann es sinnvoll sein, den Puffer an geeigneter Stelle mittels INKEY zu leeren. Der folgende Code fragt solange den Tastaturpuffer ab, bis der Rückgabestring leer ist, also die Länge 0 besitzt.

```
DO : LOOP WHILE LEN(INKEY)
```

Unterschiede zu QB:

- In QB ist das Suffix $ verbindlich.
- Bei erweiterten Zeichen (z.{nbsp&"reflinkicon" href="temp0051.html">CHR(255). In QB ist es CHR(0).

Unterschiede unter den FB-Dialektformen:

- In der Dialektform -lang qb ist das Suffix $ verbindlich. Das erste Zeichen bei erweiterten Zeichen ist, wie in QB, CHR(0) statt CHR(255).
- In den Dialektformen -lang fblite und -lang fb ist das Suffix optional.

Siehe auch:
INPUT (Funktion), GETKEY, MULTIKEY, Benutzereingaben

Letzte Bearbeitung des Eintrags am 27.12.12 um 13:32:01
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

INP

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » I » **INP**
Syntax: INP (Portnummer)
Typ: Funktion
Kategorie: Hardware

INP liest ein Byte von einem Port.

- 'Portnummer' ist ein USHORT und gibt die Adresse des zu lesenden Ports an.
- Der Rückgabewert ist ein INTEGER mit dem Wert, der am abgefragten Port anliegt.

Direkte Portzugriffe sind unter Windows NT, 2000, XP und Vista ohne einen speziellen Systemtreiber nicht möglich. Ab FreeBASIC v0.15b wird vom FBCompiler ein 3KB großer Systemtreiber in die erstellte EXE integriert.
Dieser Systemtreiber wird nur ausgeführt, wenn das Programm unter Administrator-Rechten gestartet wurde. Danach können auch Programme, die nicht unter Administrator-Rechten laufen, auf diesen Treiber zugreifen. Nach jedem Neustart des Betriebssystems ist ein erneuter Aufruf des Systemtreibers unter Administrator-Rechten erforderlich.

Unterschiede zu früheren Versionen von FreeBASIC:

- Bis FreeBASIC v0.14 emuliert INP den Zugriff auf den VGA-Port. Nur der Port &H3C9 funktioniert wirklich; alle anderen Ports liefern 0 zurück. Im Normalfall benötigen Sie diese Funktion nicht; benutzen Sie stattdessen PALETTE.
- Ab FreeBASIC v0.15b werden Zugriffe auf den VGA-Port immer noch emuliert, wie in FreeBASIC v0.14. Wenn der Zugriff auf den Port fehlschlägt, wird ein Laufzeit-Fehler erzeugt.

Siehe auch:
OUT, WAIT, PALETTE, PALETTE GET, Hardware-Zugriffe

Letzte Bearbeitung des Eintrags am 27.12.12 um 13:33:01
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

INPUT (Anweisung)

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » I » **INPUT (Anweisung)**
Syntax: INPUT [;] ["Frage" {; | , }] Variable [, Variable [...]]
Typ: Anweisung
Kategorie: Benutzereingabe

INPUT gibt dem Benutzer die Möglichkeit, über Tastatur Werte einzugeben.

- 'Frage' ist ein String-Literal, der auf dem Bildschirm an der aktuellen Cursorposition angezeigt wird, bevor der Benutzer seine Eingabe tätigen kann. Es darf sich nicht um eine Variable oder einen zusammengesetzten Stringausdruck handeln. Wird 'Frage' ausgelassen, so wird stattdessen ein Fragezeichen ausgegeben.
- Wird als Trennzeichen zwischen 'Frage' und 'Variable' ein Strichpunkt statt eines Kommas gesetzt, so wird an das Ende von 'Frage' ein Fragezeichen angehängt.
- Bei 'Variable' handelt es sich um Variablen beliebigen Typs, in denen die Eingabe des Benutzers gespeichert wird.

Nachdem die Ausgabe erfolgt ist, hat der Benutzer die Möglichkeit, einen oder mehrere Werte einzugeben. Die Werte werden dann in den angegebenen Variablen gespeichert. Dabei müssen die einzelnen Werte durch Kommata getrennt werden; dies bedeutet, dass die Eingaben selbst keine Kommata enthalten dürfen. Bei der Eingabe von Zahlen dienen auch Leerzeichen zur Trennung der einzelnen Werte.

Um dennoch Strings mit Kommata eingeben zu können, setzen Sie Ihre Eingabe in "Anführungszeichen". Wenn ein Eingabewert mit einem Anführungszeichen beginnt, wird alles bis zum nächsten Anführungszeichen als zusammenhängender Wert interpretiert. Ein anschließendes Komma zur Trennung vom nächsten Eingabewert ist weiterhin möglich, jedoch nicht erforderlich.

Gibt der Benutzer mehr Werte an, als in der Anweisung gefordert werden, so werden die überflüssigen Werte ignoriert. Gibt der Benutzer zu wenige Daten an, befüllt FreeBASIC die fehlenden Variablen mit 0 bzw. Nullstrings. Wird versucht, einer numerischen Variablen einen String zuzuweisen, wird dieser mittels VAL umgewandelt und gegebenenfalls mathematisch gerundet (vgl. CINT).

Beispiel:

```
DIM wert1 AS INTEGER
DIM wert2 AS SINGLE
DIM wert3 AS STRING
INPUT "Gib 3 Werte an: ", wert1, wert2, wert3

PRINT wert1, wert2, wert3
SLEEP
```

Auswertungsbeispiel A:

```
Gib 3 Werte an: 10, 2.5, String
10       2.5       String
```

Auswertungsbeispiel B:

```
Gib 3 Werte an: String, 1
0       1
```

Hinweis: 'wert3' hat hier den Wert "", ist also ein Leerstring.

INPUT kann durchaus für einfache Eingaben verwendet werden. Fortgeschrittene Programmierer verwenden jedoch oft eigene Ersatz-Routinen, die erweiterte Fähigkeiten besitzen oder von der LOCATE-Anweisung unabhängig sind.

Unterschiede zu QB:
In FreeBASIC ist es möglich, Werte einzugeben, die länger sind als 256 Zeichen. Die Länge der Eingabe wird nur durch die maximale Länge eines Strings begrenzt, die derzeit bei 2GB liegt.

Unterschiede zu früheren Versionen von FreeBASIC:

- Seit FreeBASIC v0.18 wird beim Auslassen der Frage ein Fragezeichen ausgegeben.
- Seit v0.16 rundet FreeBASIC eine eingegebene Gleitkommazahl automatisch zu einem INTEGER, wenn keine Gleitkommavariable zur Verfügung steht.

Siehe auch:
INPUT (Funktion), INPUT "reflinkicon" href="temp0317.html">PRINT (Anweisung), LINE INPUT, Benutzereingaben

Letzte Bearbeitung des Eintrags am 06.04.12 um 23:21:43
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

INPUT (Datei)

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » I » **INPUT (Datei)**
Syntax: INPUT #f, Variable [, Variable [, ...]]
Typ: Anweisung
Kategorie: Dateien

INPUT liest Daten aus einer Datei, die im Dateimodus INPUT oder BINARY geöffnet wurde. Dabei wird Datensatz für Datensatz aus einer sequentiellen Datei gelesen. Die eingelesenen Daten werden in den angegebenen Variable gespeichert.

- 'f' ist die Dateinummer, die durch die OPEN-Anweisung zugewiesen wurde.
- 'Variable' ist eine beliebige Variable beliebigen Typs.

Beispiel:
Datei liste.dat öffnen und alle Datensätze in der Datei auf dem Bildschirm ausgeben:

```
DIM satznr AS INTEGER
DIM satz   AS STRING
satznr = 0

DIM f AS INTEGER = FREEFILE
OPEN "liste.dat" FOR INPUT AS "hlkw0">DO
   satznr = satznr + 1
   INPUT "hlzeichen">, satz

   PRINT satznr, satz
LOOP UNTIL EOF(f)
GETKEY
```

Die Länge eines Datensatzes wird beim Schreiben in die Datei mit PRINT "reflinkicon" href="temp0469.html">WRITE # festgelegt. Bei unpassenden Datentypen gelten dieselben Regeln wie bei INPUT (Anweisung).

Siehe auch:
INPUT (Anweisung), INPUT (Funktion), OPEN, INPUT (Dateimodus), LINE INPUT #, PRINT #, WRITE #, Dateien (Files)

Letzte Bearbeitung des Eintrags am 27.12.12 um 13:33:59
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

INPUT (Dateimodus)

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » I » **INPUT (Dateimodus)**
Syntax: OPEN Dateiname FOR INPUT [...]
Typ: Schlüsselwort
Kategorie: Dateien

Das Schlüsselwort INPUT wird mit der OPEN-Anweisung verwendet und öffnet die Datei im INPUT-Modus. Das heißt, die Datei wird mit sequentiellem Zugriff zum Lesen geöffnet und der Lesezugriff erfolgt am Anfang der Datei.

Siehe auch:
INPUT #, INPUT (Anweisung), INPUT (Funktion), LINE INPUT #, OPEN (Anweisung), Dateien (Files)

Weitere Informationen:
QB Express Issue #4: File Manipulation In QuickBasic: Sequential Files

Letzte Bearbeitung des Eintrags am 27.12.12 um 13:34:16
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

INPUT (Funktion)

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » I » **INPUT (Funktion)**
Syntax: INPUT[$] (Anzahl [, [#]Dateinummer])
Typ: Funktion
Kategorie: Benutzereingabe

INPUT liest eine bestimmte Menge an Zeichen von der Tastatur oder aus einer Datei und gibt sie als STRING zurück.

- 'Anzahl' ist die Zahl der Zeichen, die eingelesen werden sollen.
- 'Dateinummer' ist die Nummer einer im INPUT-Modus geöffneten Datei.
- Der Rückgabewert ist ein String, der die eingegebenen Zeichen enthält.

Wird eine Dateinummer angegeben, dann liest der Befehl 'Anzahl' Zeichen aus einer Datei und aktualisiert den Dateicursor. Ansonsten wartet er auf die Eingabe von 'Anzahl' Zeichen von der Tastatur und gibt diese als STRING zurück. Erweiterte Zeichen (wie z.B. die Pfeiltasten) werden nicht eingelesen. Die Zeichen werden nicht auf dem Bildschirm ausgegeben.

Das Dollarzeichen ($) als Suffix ist optional.

Beispiel 1: Von der Tastatur lesen

```
DIM k AS STRING

PRINT "1) laden"
PRINT "2) speichern"
PRINT "3) neu"
PRINT
PRINT "0) Beenden"
PRINT

k = INPUT(1)

PRINT "gewählt: ";
SELECT CASE k
   CASE "1"
      PRINT "laden"
   CASE "2"
      PRINT "speichern"
   CASE "3"
      PRINT "neu"
   CASE "0"
      END
   CASE ELSE
      PRINT "ungültige Auswahl!"
END SELECT
SLEEP
```

Beispiel 2: Die ersten drei Zeichen aus der Datei "file.ext" lesen

```
DIM AS INTEGER f = FREEFILE
OPEN "file.ext" FOR INPUT AS "hlkw0">PRINT INPUT(3, "hlzeichen">)
CLOSE "hlkw0">SLEEP
```

Unterschiede zu QB:
In QB ist das Suffix $ verbindlich.

Unterschiede unter den FB-Dialektformen:

- In der Dialektform -lang qb ist das Suffix $ verbindlich.
- In den Dialektformen -lang fblite und -lang fb ist das Suffix optional.

Siehe auch:
INPUT (Anweisung), INPUT (Datei), INPUT (Dateimodus), LINE INPUT, INKEY, GETKEY, MULTIKEY, OPEN, WINPUT, Benutzereingaben

Letzte Bearbeitung des Eintrags am 27.12.12 um 13:34:48
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

INSTR

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » I » **INSTR**
Syntax: INSTR ([Start,] String1, [ANY] String2)
Typ: Funktion
Kategorie: Stringfunktionen

INSTR prüft, ob 'String2' oder Teile davon in 'String1' vorkommt, und liefert die Position innerhalb 'String1' zurück.

- 'Start' ist eine INTEGER-Zahl, die angibt, von welchem Zeichen an 'String1' durchsucht werden soll. Wenn 'Start' ausgelassen wird, beginnt die Suche beim ersten Zeichen.
- 'String1' und 'String2' sind zwei Zeichenketten (STRING, ZSTRING oder WSTRING).
- INSTR prüft, ob der gesamte 'String2' in 'String1' vorkommt. Wird die ANY-Klausel verwendet, prüft INSTR nur, ob irgendeines der Zeichen aus 'String2' vorkommt.
- Der Rückgabewert ist ein INTEGER, der die erste Position angibt, an der 'String2' bzw. Teile davon in 'String1' gefunden wurden. Falls keine Treffer erzielt werden, liefert INSTR den Wert 0 zurück.

Beispiel:

```
PRINT INSTR("abcdefg", "def")
PRINT INSTR(4, "Das Leben ist ein Hund", ANY "Wasser")
SLEEP
```

Ausgabe:

```
 4
 6
```

Beispiel 2: alle vorkommenden 'b' suchen

```
Dim teststring As String
Dim idx As Integer

teststring = "abababab"
idx = InStr(teststring, "b")

Do While idx > 0 'Falls nichts gefunden wird, wird die Schleife
übersprungen
  Print """b"" an Stelle " & idx
  idx = InStr(idx + 1, teststring, "b")
Loop
Sleep
```

Hinweis: Die ANY-Klausel hat nichts mit dem Datentyp ANY zu tun

Unterschiede zu QB:

- Die ANY-Klausel ist neu in FreeBASIC.
- QB gibt *'Start'* zurück, falls der Suchstring leer ist.

Plattformbedingte Unterschiede:
Unicode wird in der DOS-Portierung von FreeBASIC nicht unterstützt, deshalb können dort auch keine Unicode-Strings umgewandelt werden.

Unterschiede zu früheren Versionen von FreeBASIC:
Die ANY-Klausel kann seit FreeBASIC v0.15 eingesetzt werden.

Siehe auch:
INSTRREV, String-Funktionen

Letzte Bearbeitung des Eintrags am 27.12.12 um 13:36:05
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

INSTRREV

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » I » **INSTRREV**
Syntax: INSTRREV (String1, [ANY] String2, [Start])
Typ: Funktion
Kategorie: Stringfunktionen

INSTRREV prüft, ob 'String2' oder Teile davon in 'String1' vorkommt, und liefert die letzte auftretende Position zurück.

- 'String1' und 'String2' sind zwei Zeichenketten (STRING, ZSTRING oder WSTRING).
- INSTRREV prüft, ob der gesamte 'String2' in 'String1' vorkommt. Wird die ANY-Klausel verwendet, prüft INSTRREV nur, ob irgendeines der Zeichen aus 'String2' vorkommt.
- 'Start' ist eine INTEGER-Zahl, die angibt, von welchem Zeichen an 'String1' durchsucht werden soll. Wenn 'Start' ausgelassen wird, beginnt die Suche beim letzten Zeichen.
- Der Rückgabewert ist ein INTEGER, der die Position angibt, an der 'String2' bzw. Teile davon das letzte Mal in 'String1' auftreten. Falls keine Treffer erzielt werden, liefert INSTRREV den Wert 0 zurück.

Beispiele:

```
Print InstrRev("abcdefg", "de") ' gibt 4 zurück

Print InstrRev("abcdefg", "h")  ' gibt 0 zurück

' alle auftretenden 'b' suchen:
Dim test As String = "abababab"
Dim idx As Integer

idx = InstrRev(test, "b")

Do While idx > 0 'falls nicht gefunden, wird die Schleife übersprungen
  Print "b bei " & idx
  idx = InstrRev(Test, "b", idx - 1)
Loop
Sleep
```

Hinweis:
Bis einschließlich Version 0.20.0b ist ein Fehler in INSTRREV enthalten, der dafür sorgt, dass manche Treffer nicht angezeigt werden. Dieser Fehler ist ab Version 0.21.0 SVN behoben.

Unterschiede zu QB: neu in FreeBASIC

Plattformbedingte Unterschiede:
Unicode wird in der DOS-Portierung von FreeBASIC nicht unterstützt, deshalb können dort auch keine Unicode-Strings umgewandelt werden.

Unterschiede zu früheren Versionen von FreeBASIC: existiert seit FreeBASIC v0.18.4

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht INSTRREV nicht zur Verfügung und kann nur über **__INSTRREV** aufgerufen werden.

Siehe auch:

INSTR, String-Funktionen

Letzte Bearbeitung des Eintrags am 27.12.12 um 13:36:29
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

INT

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » I » **INT**
Syntax: INT (Ausdruck)
Typ: Funktion
Kategorie: Typumwandlung

INT verwandelt einen numerischen Ausdruck in die nächstkleinere oder gleiche INTEGER-Zahl. INT(4.9) wird beispielsweise 4 zurückgeben, während INT(-1.3) -2 ausgibt.

INT kann mithilfe von OPERATOR überladen werden.

Siehe auch:
FIX, CINT, FRAC, mathematische Funktionen

Letzte Bearbeitung des Eintrags am 27.12.12 um 13:38:46
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

INTEGER

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » I » **INTEGER**
Syntax:

```
DIM AS INTEGER variable
DIM AS INTEGER<Bits> variable
```

Typ: Datentyp

Bei der x86-Version des Compilers ist ein INTEGER eine vorzeichenbehaftete 32-bit-Ganzzahl (vgl. LONG). Sie liegt im Bereich von -(2^31) bis (2^31)-1, bzw. von -2'147'483'648 bis 2'147'483'647.
Bei der x64-Version des Compilers ist ein INTEGER eine vorzeichenbehaftete 64-bit-Ganzzahl (vgl. LONGINT). Sie liegt im Bereich von -(2^63) bis (2^63)-1, bzw. von -9'223'372'036'854'775'808 bis 9'223'372'036'854'775'807.

Wird der Parameter 'Bits' angegeben, so kann die exakte Größe des Datentyps definiert werden. Erlaubte Werte sind dabei 8, 16, 32 und 64.

Unterschiede zu QB:
In QB sind INTEGER immer 16-bit-Zahlen.

Unterschiede zu früheren Versionen von FreeBASIC:
Der 'Bits'-Parameter existiert seit FreeBASIC v0.90.

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb werden INTEGER standardmäßig als 16-bit-Ganzzahlen behandelt.

Siehe auch:
DIM, CAST, CINT, Datentypen, Datentypen und Deklarationen

Letzte Bearbeitung des Eintrags am 04.07.13 um 17:45:53
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

IS

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » I » **IS**
Syntax: CASE IS Bedingung
Typ: Schlüsselwort
Kategorie: Programmablauf

IS wird im Zusammenhang mit SELECT CASE verwendet.

Beispiel:

```
Dim As Integer x = 5

Select Case x
    Case Is < 0
        Print "x ist kleiner als 0"
    Case Is <= 5
        Print "x ist ein Wert von 0 bis 5"
    Case Is > 5
        Print "x ist groeßer als 5"
End Select

Sleep
```

Seit FreeBASIC v0.24 ist IS auch ein Syntaxbestandteil in der Vererbung.

Siehe auch:
SELECT CASE, IS (Vererbung), Bedingungsstrukturen

Letzte Bearbeitung des Eintrags am 27.12.12 um 13:42:55
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

IS (Vererbung)

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » I » **IS (Vererbung)**
Syntax: Variable IS Klasse
Typ: Operator
Kategorie: Klassen

IS wird in Verbindung mit Vererbung dazu genutzt, um den Typen einer Variable zu prüfen. Der Operator kann nur genutzt werden, wenn die Basis-Klasse von OBJECT erbt und somit die RTTI-Funktionalität zur Verfügung steht.

Der Operator prüft nicht, ob die Variable exakt zur gefragten Klasse gehört. Er liefert vielmehr auch dann ein wahres Ergebnis, wenn die Variablen-Klasse von der angegebenen Klasse erbt. Das bedeutet, dass eine Variable einer Kind-Klasse auch in die Basis-Klasse fällt. Ein Beispiel dazu finden Sie im Artikel zu OBJECT. Wenn Sie die exakte Zugehörigkeit zu einer Klasse prüfen wollen, sollten Sie aus diesem Grund die abgefragten Klassen immer in zur Erbreihenfolge entgegengesetzen Reihenfolge prüfen.

Beispiel 1:

```
Type Haustier Extends Object
  As Integer beine   = 4
  As Integer schwanz = 1
End Type

Type Hund Extends Haustier
  Declare Sub gibLaut
End Type
Sub Hund.gibLaut
  Print "Wuff!"
End Sub

Type Chihuahua Extends Hund
  Declare Sub gibLaut
End Type
Sub Chihuahua.gibLaut
  Print "Klaeffklaeff!"
End Sub

Dim As Haustier ptr balto = New Hund
If *balto Is Chihuahua Then
  Print "Balto ist sehr klein."
ElseIf *balto Is Hund Then
  Print "Balto ist ein normaler Hund."
Else
  Print "Balto ist ein Haustier."
End If

Delete balto
Sleep
```

IS arbeitet nur mit der Klasse, die direkt von OBJECT erbt. Andererseits kann diese Klasse natürlich nicht auf die Records und Methoden seiner Kindklassen zugreifen; balto.gibLaut ist in oben stehenden Beispiel nicht möglich. Als Lösung kann die Variable entsprechend geCASTet werden. Damit ist es z. B. auch möglich, an eine Prozedur eine Variable zu übergeben, die je nach zugehöriger Klasse anders behandelt wird.

Beispiel 2:

```
' "hlkw0">Type Haustier Extends Object
  As Integer beine   = 4
  As Integer schwanz = 1
End Type

Type Hund Extends Haustier
  As Integer anhaenglich = 1
  Declare Sub gibLaut ()
End Type
Sub Hund.gibLaut ()
  Print "Wuff!"
End Sub

Type Katze Extends Haustier
  As Integer verspielt = 1
  Declare Sub gibLaut ()
End Type
Sub Katze.gibLaut ()
  Print "Miau!"
End Sub

' "hlkw0">Sub haustiertest(tier As Haustier)
  If tier Is Hund Then
    Print "Dieser Hund hat einen Anhaenglichkeitswert von " & Cast(Hund
ptr, @tier)->anhaenglich & "."
    Cast(Hund ptr, @tier)->gibLaut
  Elseif tier Is Katze Then
    ' dasselbe mit temporaerer Variable
    Dim As Katze ptr k = Cast(Katze ptr, @tier)
    Print "Diese Katze hat einen Verspieltheitswert von " & k->verspielt
& "."
    k->gibLaut
  Else
    Print "Ueber dieses Tier weiss ich nichts."
  End If
End Sub

' "hlkw0">Dim mautzi As Katze, balu As Hund
haustiertest(balu)
Print
haustiertest(mautzi)
Sleep
```

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC:
IS in Verbindung mit Vererbung existiert seit FreeBASIC v0.24

Siehe auch:
TYPE (UDT), BASE, OBJECT, EXTENDS, Objektorientierung

Letzte Bearbeitung des Eintrags am 18.08.12 um 18:44:14
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

ISDATE

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » I » **ISDATE**
Syntax: ISDATE (Datumsstring)
Typ: Funktion
Kategorie: Datum und Zeit

ISDATE überprüft, ob ein String einem korrekten Datumsformat entspricht.

Anmerkung:
Um diese Funktion nutzen zu können, muss die Datei datetime.bi in Ihren Quellcode eingebunden werden, z.B. mit INCLUDE. Alternativ können Sie auch die Datei vbcompat.bi einbinden, da dadurch auch automatisch datetime.bi eingebunden wird.

'Datumsstring' ist ein STRING, der analysiert werden soll. Der Rückgabewert ist entweder gleich -1, wenn der String ein Datum enthält, oder gleich 0, wenn dies nicht der Fall ist.

Damit FreeBASIC einen String als Datum anerkennt, muss er im FB-Datumsformat gegeben sein. Die einzelnen Segmente (Tag, Monat, Jahr) dürfen durch die Zeichen Minus (-), Slash (/) und Punkt (.) voneinander abgetrennt werden. Alle drei Segmente müssen angegeben werden, ansonsten ist das Ergebnis dieser Funktion gleich 0. Werden in einem String zwei verschiedene Trennzeichen benutzt, so wird ebenfalls kein Datum erkannt. Vor und nach einem Trennzeichen sowie vor und nach dem Datum an sich dürfen beliebig viele Leerzeichen eingefügt werden; jedoch müssen die Ziffern eines Datumssegments direkt beisammen stehen und dürfen nicht durch Leerzeichen getrennt werden. Enthält ein korrekt formatierter Datumsstring neben dem Datum und Leerzeichen noch andere Zeichen, so wird 0 ausgegeben. ISDATE arbeitet auch mit ZSTRINGs und WSTRINGs.

Achtung: Je nach Benutzereinstellung bzw. Betriebssystem variiert die Reihenfolge der Segmente (z. B. dd.mm.yy oder mm.dd.yy).

Beispiel 1:

```
"hlstring">"vbcompat.bi"

DIM s AS String, d AS Integer

DO
Print
Print "Bitte geben Sie ein Datum ein: "

Line Input s

If s = "" Then Exit Do

If IsDate( s ) Then
d = DateValue( s )
Print "Jahr : "; Year ( d )
Print "Monat : "; Month( d )
Print "Tag : "; Day ( d )
Else
Print "'"; s; "' ist kein zulaessiges Datum!"
End If

Loop
```

Beispiel 2:

```
"hlstring">"vbcompat.bi"

"hlkw0">Chr(34)       ' Das Anführungszeichen (")

Data "01.01.01"
Data "01/01/01"
Data "01-01-01"
Data "01,01,01"
Data "01.01/01"
Data "01.01"
Data " 01 . 01 . 01 "
Data "01. 01. 0 1"
Data "01.01.01 a"
Data "End"

Dim s As String

Do
   Read s
   If s = "End" Then Exit Do

   Color 0
   PRINT FB_QM; s; FB_QM; Tab(40)

   If IsDate(s) Then
      Color 10
      PRINT "ist ein gueltiges Datum"
   Else
      Color 12
      PRINT "ist kein gueltiges Datum"
   End If
Loop

Sleep
```

Ausgabe:

```
"01.01.01"              ist ein gueltiges Datum
"01/01/01"              ist ein gueltiges Datum
"01-01-01"              ist ein gueltiges Datum
"01,01,01"              ist kein gueltiges Datum
"01.01/01"              ist kein gueltiges Datum
"01.01"                 ist kein gueltiges Datum
" 01 . 01 . 01 "        ist ein gueltiges Datum
"01. 01. 0 1"           ist kein gueltiges Datum
"01.01.01 1"            ist kein gueltiges Datum
```

Sie können in dieses Beispiel gerne weitere Test-Daten einbauen: Fügen Sie einfach eine weitere DATA-Zeile vor der Zeile DATA "End" ein.

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC: existiert seit FreeBASIC v0.17

Siehe auch:
NOW, DATESERIAL, DATEVALUE, YEAR, MONTH, DAY, WEEKDAY, MONTHNAME, WEEKDAYNAME, DATEDIFF, DATEPART, DATEADD, FORMAT, Serial Numbers, Datum und Zeit

Letzte Bearbeitung des Eintrags am 27.12.12 um 13:46:23
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

KILL

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » K » **KILL**
Syntax: KILL (Dateiname)
Typ: Funktion
Kategorie: System

KILL löscht eine Datei von einem Datenträger.

- 'Dateiname' ist ein STRING, der den Namen einer Datei enthält, die gelöscht werden soll. Er muss ihre Erweiterung enthalten und kann mit einer Pfadangabe versehen sein. Wird kein Arbeitspfad angegeben, so sucht KILL die Datei im aktuellen Arbeitsverzeichnis; siehe CURDIR. Die Verwendung von Wildcards ist nicht möglich (siehe DIR zu Wildcards).
- Der Rückgabewert der Funktion ist entweder 0, wenn der Löschvorgang erfolgreich abgeschlossen wurde, oder die Fehlernummer 2, wenn die Datei nicht existiert, bzw. 3, wenn die Datei nicht gelöscht werden kann.

Der Rückgabewert kann verworfen werden; KILL wird dann wie eine Anweisung eingesetzt.

Beispiel:

```
SELECT CASE KILL("file.ext")
  CASE 0 : PRINT "Die Datei wurde gelöscht."
  CASE 2 : PRINT "Fehler: Die Datei existiert nicht!"
  CASE 3 : PRINT "Fehler: Unzureichende Zugriffsrechte, oder es handelt
sich um einen Ordner!"
END SELECT
SLEEP
```

Achtung: Die zu löschende Datei wird nicht in den Papierkorb verschoben; sie wird unwiederbringlich gelöscht!

Unterschiede zu QB:
KILL kann in FreeBASIC auch als Funktion eingesetzt werden.

Plattformbedingte Unterschiede:

- Auf manchen Plattformen kann KILL möglicherweise Ordner und schreibgeschützte Dateien löschen. Ob dies möglich ist oder nicht, ist zur Zeit nicht festgelegt. Überprüfen Sie ggf. zuvor die Attribute der Datei und entscheiden Sie dementsprechend, ob Sie die Datei löschen wollen.
- Unter Linux muss der Dateiname 'case sensitive' erfolgen. Windows und DOS sind 'case insensitive'.
- Das Trennzeichen für den Dateipfad ist unter Linux der vorwärtsgerichtete Slash /. Windows verwendet den Backslash \, erlaubt aber auch den Slash. DOS verwendet den Backslash.

Siehe auch:
MKDIR, RMDIR, CHDIR, CURDIR, SHELL, Betriebssystem-Anweisungen

Letzte Bearbeitung des Eintrags am 27.12.12 um 13:47:18
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

LANG (Meta)

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » L » **LANG (Meta)**
Syntax A: #LANG "Sprachversion"
Syntax B: '$LANG: "Sprachversion"
Syntax C: REM $LANG: "Sprachversion"
Typ: Metabefehl
Kategorie: Metabefehle

#LANG legt den Dialekt des Programms fest. "Sprachversion" kann einer der Begriffe "fb", "fblite", "qb" oder "deprecated" sein und wird in Anführungszeichen angegeben.

Wird die -lang-Option in der Befehlszeile nicht angegeben, kann #LANG verwendet werden, um für den Code des aktuellen Moduls einen bestimmten Dialekt einzustellen. Meistens gibt es zwei Compiler-Durchläufe für ein Quellcode-Modul. Im ersten Durchlauf wird, falls der angegebene Dialekt nicht identisch mit dem als Compileroption festgelegten Dialekt ist, der Parser für einen neuen Durchlauf zurückgesetzt, sodass die Übersetzung neustartet. Taucht im zweiten Durchlauf wieder ein LANG-Befehl auf, der nicht dem ersten entspricht, wird eine Warnung ausgegeben und die Übersetzung fortgesetzt. Sind im ersten Durchlauf irgendwelche Fehler aufgetreten, bricht der Compiler ab.

#LANG darf nicht in einer Blockanweisung (z. B. FOR ... NEXT, WHILE ... WEND), einem Programmbereich (SCOPE) oder in einem Unterprogramm auftauchen. Jedoch darf es innerhalb anderer Präprozessor-Anweisungen (z.B. #IF) oder in einer Include-Datei benutzt werden.

Es gibt aktuell keine Einschränkung für die Platzierung dieser Anweisung im Quellcode. Das kann sich jedoch in zukünftigen Versionen ändern. Die Anweisung sollte besser nur am Anfang einer Quellcode-Datei verwendet werden. Sie überschreibt die Option -lang, falls diese in der Befehlszeile übergeben wurde. Jedoch kann der Befehl durch die Option -forcelang wiederum überschrieben werden.

Beispiel:

```
"hlstring">"qb"
'Der Quellcode wird wie mit fbc -lang qb übersetzt.
```

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC: existiert seit FB 0.20

Siehe auch:
FB-Dialektformen, Der Compiler, __FB_LANG__

Letzte Bearbeitung des Eintrags am 27.12.12 um 13:49:27
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

LBOUND

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » L » **LBOUND**
Syntax: LBOUND (Array[, Dimension])
Typ: Funktion
Kategorie: Speicher

LBOUND gibt den kleinsten Index des angegebenen Arrays zurück.

- 'Array' ist der Name des Arrays, dessen unterster Index zurückgegeben werden soll.
- 'Dimension' ist die Nummer der Dimension, deren unterster Index zurückgegeben werden soll. Wird dieser Wert ausgelassen, so nimmt FreeBASIC automatisch 1 an. Ist 'Dimension' gleich 0, so wird 1 ausgegeben, da die Dimension 1 immer die erste ist. Ist 'Dimension' kleiner als 0, so werden ungültige Werte erzeugt. Ist 'Dimension' größer als die Anzahl der Dimensionen, so wird der Wert 0 ausgegeben.

Ist das Array bisher nur deklariert, hat aber noch keine Dimensionen, so gibt LBOUND 0 aus. Ob das Array nun den geringsten Index 0 besitzt oder einfach keine Dimensionen, kann man über die Adresse des ersten Elements erfahren, welche 0 ist sofern es keine Dimensionen gibt. Alternativ kann zusätzlich UBOUND abgefragt werden, was in einem solchen Fall -1 zurück gibt. Eine weitere Möglichkeit besteht darin, LBOUND und UBOUND mit Dimension 0 abzufragen, wobei bei ersterem der Wert 1 und bei letzterem der Wert 0 (keine Dimensionen) zurückgegeben wird.

Beispiel:

```
DIM Array(-10 TO 10, 5 TO 15, 1 TO 2) AS INTEGER
DIM unArray() AS INTEGER

PRINT LBOUND(Array, 1)
PRINT LBOUND(Array, 2)
PRINT LBOUND(Array, 3)
PRINT LBOUND(Array, 4)
PRINT

' Überprüfung, ob das Array dimensioniert wurde
PRINT LBOUND(unArray)
PRINT @unArray(0)
IF UBOUND(unArray) < LBOUND(unArray) THEN
  PRINT "Das Array wurde noch nicht dimensioniert."
END IF
SLEEP
```

Ausgabe:

```
-10
 5
 1
 0

 0
0
Das Array wurde noch nicht dimensioniert.
```

Unterschiede zu früheren Versionen von FreeBASIC:

- Bis einschließlich FreeBASIC v0.24 führten sowohl der Wert 0 als auch 1 bei 'Dimension' dazu, dass die Grenzen der ersten Dimension abgefragt werden.
- Bis einschließlich FreeBASIC v0.15 ergaben leere Dimensionen statt 0 den Wert -1.

Siehe auch:
UBOUND, DIM, REDIM, Arrays

Letzte Bearbeitung des Eintrags am 04.07.13 um 19:17:24
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

LCASE

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » L » **LCASE**
Syntax: LCASE[$] (Stringausdruck [, Modus])
Typ: Funktion
Kategorie: Stringfunktionen

LCASE wandelt einen Stringausdruck in Kleinbuchstaben.

- 'Stringausdruck' ist ein STRING, ZSTRING oder WSTRING, der in Kleinbuchstaben zurückgegeben werden soll.
- 'Modus' gibt den ASCII-Modus an, wobei der Wert 1 den Modus aktiviert. Wird 0 angegeben oder 'Modus' ausgelassen, so wird der ASCII-Modus deaktiviert. Ist der ASCII-Modus aktiv, so werden nur ASCII-Zeichen unterstützt statt der durch die Konsole eingestellte Lokalisierung.
- Der Rückgabewert ist der in Kleinbuchstaben gewandelte STRING bzw. WSTRING.

Achtung: LCASE wandelt die ASCII-Zeichen "A" - "Z" um. Welche weiteren Zeichen umgewandelt werden, hängt vom Parameter 'Modus' sowie von der systeminternen Lokalisierung ab. Wenn Sie Unicode-Dateien verwenden, ist es außerdem nötig, das Byte Order Mark (BOM) zu setzen, damit Sonderzeichen wie z. B. Umlaute korrekt behandelt werden.

Das Dollarzeichen ($) als Suffix ist optional.

Beispiel:

```
PRINT LCASE("Hello World abCDefGHäÄ", 1)
SLEEP
```

Ausgabe:

```
hello world abcdefghäÄ
```

Hinweis: Im Grafikfenster wird eine andere Codepage verwendet als in der Konsole, weshalb Umlaute andere Codenummern besitzen und nicht mit LCASE bearbeitet werden können. Sie müssen bei Bedarf gesondert umgewandelt werden.

Unterschiede zu QB:

- In QB ist das Suffix $ verbindlich.
- Da QB kein Unicode unterstützt, existiert dort auch nicht die Möglichkeit, WSTRING zu verwenden.

Unterschiede zu früheren Versionen von FreeBASIC:
Der Modus-Parameter existiert seit FreeBASIC v0.90.

Plattformbedingte Unterschiede:
Unicode wird in der DOS-Portierung von FreeBASIC nicht unterstützt, deshalb kann dort auch kein WSTRING verwendet werden.

Unterschiede unter den FB-Dialektformen:

- In der Dialektform -lang qb ist das Suffix $ verbindlich.
- In den Dialektformen -lang fblite und -lang fb ist das Suffix optional.

Siehe auch:
UCASE, String-Funktionen

Letzte Bearbeitung des Eintrags am 03.07.13 um 01:06:03
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

LEFT

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » L » **LEFT**
Syntax: LEFT[$] (Text, Anzahl)
Typ: Funktion
Kategorie: Stringfunktionen

LEFT gibt die ersten 'Anzahl' Zeichen von 'Text' zurück.

- 'Text' ist ein STRING, ZSTRING oder WSTRING, dessen Teilstring zurückgegeben werden soll.
- 'Anzahl' ist ein INTEGER mit der Anzahl der zurückgegebenen Zeichen. Ist 'Anzahl' kleiner als 0, dann wird ein Leerstring zurückgegeben. Ist 'Anzahl' größer als die Länge von 'Text', dann wird der gesamte 'Text' zurückgegeben.
- Der Rückgabewert ist ein STRING bzw. WSTRING, der die ersten 'Anzahl' Zeichen von 'Text' enthält.

Das Dollarzeichen ($) als Suffix ist optional.

Beispiel:

```
Dim Text As String
Text = "hello world"
PRINT LEFT(Text, 5) ' Ausgabe: "hello"
SLEEP
```

Unterschiede zu QB:

- In QB ist das Suffix $ verbindlich.
- Da QB kein Unicode unterstützt, existiert dort auch nicht die Möglichkeit, WSTRING zu verwenden.

Plattformbedingte Unterschiede:
Unicode wird in der DOS-Portierung von FreeBASIC nicht unterstützt, deshalb kann dort auch kein WSTRING verwendet werden.

Unterschiede unter den FB-Dialektformen:

- In der Dialektform -lang qb ist das Suffix $ verbindlich.
- In den Dialektformen -lang fblite und -lang fb ist das Suffix optional.

Siehe auch:
RIGHT, MID (Funktion), String-Funktionen

Letzte Bearbeitung des Eintrags am 27.12.12 um 13:50:26
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

LEN

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » L » **LEN**
Syntax: LEN({Variable | Datentyp})
Typ: Funktion
Kategorie: Stringfunktionen

LEN gibt die Größe eines Stringausdrucks oder eines Datentyps zurück.

- Ist 'Variable' ein STRING, ZSTRING oder WSTRING, so wird die Anzahl der Zeichen zurückgegeben.
- Bei Strings fester Länge wird immer die Gesamtgröße des Strings zurückgegeben. Dies betrifft nicht ZSTRING und WSTRING.
- Ist 'Variable' eine Zahl oder ein UDT, so wird die Größe des zugrunde liegenden Datentyps in Byte zurückgegeben. Dies ist auch der Fall, wenn stattdessen 'Datentyp' angegeben wird.

Beispiel 1:

```
PRINT LEN("hello world")   ' Ausgabe: 11
PRINT LEN(INTEGER)         ' Ausgabe: 4
SLEEP
```

Aufgrund der konstanten Länge von fixed-length-Strings gibt LEN in diesem Fall immer die Gesamtgröße des Strings zurück. Bei ZSTRING und WSTRING ist das jedoch nicht der Fall.

Beispiel 2:

```
DIM festerText AS STRING  * 15, zstr AS ZSTRING * 15
festerText = "TestA"
zstr       = "TestB"
PRINT LEN(festerText)               ' Ausgabe: 15
PRINT LEN(zstr)                     ' Ausgabe: 5
PRINT LEN(festerText & "")          ' Ausgabe: 5
PRINT INSTR(festerText, CHR(0)) - 1 ' Ausgabe: 5
SLEEP
```

Unterschiede zu QB:
In FreeBASIC funktioniert LEN mit allen Datentypen und mit benutzerdefinierten Typen.

Unterschiede unter den FB-Dialektformen:

- In der Dialektform -lang qb ist nur die Übergabe von 'Variable' erlaubt.
- In den Dialektformen -lang fblite und -lang fb kann auch die Länge von Datentypen abgefragt werden.

Siehe auch:
SIZEOF, STRING (Datentyp), OPEN, String-Funktionen, Verschiedenes

Letzte Bearbeitung des Eintrags am 27.12.12 um 13:50:51
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

LET

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » L » **LET**
Syntax A: [LET] Variable = Ausdruck
Syntax B: LET (Variablenliste) = UDT_Variable
Typ: Anweisung
Kategorie: Operatoren

LET hat in der Verwendung einer einfachen Variablenzuweisung (Syntax A) keine praktische Funktion mehr; es wurde nur beibehalten, um die Kompatibilität zu älteren BASIC-Dialekten zu wahren. Stattdessen sollte LET hier einfach ausgelassen werden.

Mit Syntax B ist eine mehrfache Variablenzuweisung möglich. Dabei wird einer Liste von Variablen die Werte der einzelnen Records eines UDTs zugewiesen. 'Variablenliste' muss aus Variablen derselben (oder kompatiblen) Datentypen wie die Records bestehen. Es können auch weniger Variablen aufgelistet werden, als 'UDT_Variable' Records besitzt.

LET kann mithilfe von OPERATOR überladen werden.

Beispiel 1 (funktioniert nicht mit -lang fb):

```
DIM AS INTEGER a, b

LET a = 15 + 7
b = 3 * 6

PRINT a, b
SLEEP
```

Beispiel 2:

```
Type Vector3D
    x As Double
    y As Double
    z As Double
End Type

Dim a As Vector3D = ( 5, 7, 9 )

Dim x As Double, z As Double

' nur den ersten und letzten Eintrag holen
Let( x, , z ) = a

Print "x ="; x
Print "z ="; z
Sleep
```

Ausgabe:

```
x = 5
z = 9
```

Unterschiede zu QB:
Mehrfache Variablenzuweisung (Syntax B) ist neu in FreeBASIC.

Unterschiede unter den FB-Dialektformen:

- In der Dialektform -lang fb ist LET in der einfachen Variablenzuweisung (Syntax A) nicht erlaubt.
- Mehrfache Variablenzuweisung (Syntax B) ist nur in der Dialektform -lang fb erlaubt.

Letzte Bearbeitung des Eintrags am 27.12.12 um 13:51:46
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

LIB

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » L » **LIB**
Syntax A:

```
DECLARE { SUB | FUNCTION } Name LIB "LibName" ALIAS "Aliasname"
&"hlzahl">...]
```

Syntax B:

```
Type T
     As Integer dummy
     Declare Constructor Lib "LibName" ALIAS "Aliasname" &"hlzahl">...]
End Type
```

Typ: Klausel
Kategorie: Bibliotheken

In Zusammenhang mit LIB wird eine SUB/FUNCTION aus einer Lib/DLL eingebunden.
In UDTs können so auch SUBs/FUNCTIONs als normale Methode eingebunden werden, aber auch als CONSTRUCTOR oder DESTRUCTOR.

- 'LibName' ist der Dateiname (evtl. mit Pfad) der Lib/DLL.
- 'Aliasname' gibt den Namen an, unter dem die Prozedur dem Programm bekannt sein soll.
- Die anderen Parameter entsprechen denen von DECLARE

Beispiel:

```
' mydll.bas: compilieren mit
' fbc -dll mydll.bas

Public Function GetValue() As Integer Export
  Function = &h1234
End Function

Declare Function GetValue Lib "mydll" ALIAS "GetValue" As Integer

Print "GetValue = &h"; Hex(GetValue())
```

Ausgabe :

```
GetValue = &h1234
```

LIB kann auch in EXTERN ... END EXTERN Blöcken verwendet werden.

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC:
Seit FreeBASIC v0.15 muss bei der Verwendung von LIB auch ALIAS angegeben werden.

Siehe auch:
DECLARE, ALIAS, Prozeduren, Module (Library / DLL)

Letzte Bearbeitung des Eintrags am 26.05.12 um 01:27:01
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

LIBPATH (Meta)

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » L » **LIBPATH (Meta)**
Syntax: #LIBPATH Pfad
Typ: Metabefehl
Kategorie: Metabefehle

#LIBPATH fügt einen weiteren Pfad zu der Liste der Verzeichnisse hinzu, in welcher der Compiler nach einzubindenden Bibliotheken suchen soll.

'Pfad' ist ein String, der den Pfad enthält, der zur Liste hinzugefügt werden soll.

Dieser Metabefehl wirkt genau so, als ob der Pfad über die Kommandozeilenoption '-p' hinzugefügt worden wäre.

Relative Pfade werden ausgehend vom Arbeitsverzeichnis des fbc berechnet, nicht relativ zum Verzeichnis des Quellcodes. Kann der Pfad nicht gefunden werden, so wird KEIN Fehler erzeugt.

Beispiel: Verzeichnis der aktuell umgesetzten Quellcodedatei hinzufügen

```
"hlkw0">__PATH__
```

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC: existiert seit FreeBASIC v0.17

Siehe auch:
"reflinkicon" href="temp0203.html">#INCLUDE, Präprozessor-Anweisungen

Letzte Bearbeitung des Eintrags am 27.12.12 um 13:52:16
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

LINE (Grafik)

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » L » **LINE (Grafik)**
Syntax: LINE [Puffer,] [[STEP] (x1, y1)]-[STEP] (x2, y2)[, [Farbe][, [B|BF][, Stil]]]
Typ: Anweisung
Kategorie: Grafik

LINE zeichnet eine Strecke von einem angegebenen Punkt zu einem zweiten, oder ein Rechteck, dessen Eckpunkte die beiden angegebenen Punkte sind.

- 'Puffer' ist ein Speicherbereich wie ein mit IMAGECREATE erstellter Puffer oder ein Array. Beide können mit PUT angezeigt werden. Wird 'Puffer' ausgelassen, zeichnet FreeBASIC direkt auf den Bildschirm.
- '(x1, y1)' und '(x2, y2)' sind die Koordinaten, zwischen denen sich die Strecke bzw. das Rechteck befinden soll. Die Koordinaten werden durch die letzten WINDOW- und VIEW-Befehle beeinflusst.
- 'STEP' gibt an, dass die darauf folgenden Koordinaten relativ zur aktuellen Position des Grafikcursors sind.
- Wird '[STEP] (x1, y1)' ausgelassen - wird also der Befehl in der Form LINE -(x2, y2) verwendet - dann wird für die Startkoordinaten die aktuelle Position des Grafikcursors verwendet.
- 'Farbe' ist die Nummer einer Farbe. Welche Nummer für welche Farbe steht, ist von der letzten SCREENRES-Anweisung sowie den letzten PALETTE-Befehlen abhängig.
- Ist das Flag 'B' gesetzt, wird ein Rechteck anstatt einer Strecke gezeichnet.
- Ist das Flag 'BF' gesetzt, wird ein ausgefülltes Rechteck gezeichnet.
- 'Stil' ist eine 16-bit-Maske, die den Stil der Strecke festlegt. "Stil der Strecke" bedeutet, auf welche Art die Strecke gestrichelt gezeichnet werden soll. Für jedes gesetzte Bit der Maske wird ein Pixel gezeichnet. Wird Stil ausgelassen, nimmt FreeBASIC 65535 (entspricht &b1111111111111111) an, was eine Volllinie zeichnet.
- Ist das Flag 'B' oder 'BF' gesetzt, hat 'Stil' keine Wirkung.

Beispiel:
Zeichnet eine rote Strichlinie und ein weißes Rechteck und zeigt beides 3 Sekunden lang an.

```
SCREENRES 320, 200, , ,1                ' Vollbildmodus 320x200x8

LINE (20, 20)-(300, 180), 4,  , &hFF00  ' gestrichelte Strecke
LINE (140, 80)-(180, 120), 15, B        ' weißes Rechteck

SLEEP 3000
```

Unterschiede zu QB:
In FreeBASIC ist es möglich, in einen Datenpuffer zu zeichnen.

Siehe auch:
LINE (Meta), __LINE__, LINE INPUT, SCREENRES, DRAW (Grafik), Grafik

Letzte Bearbeitung des Eintrags am 27.12.12 um 13:52:32
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

LINE (Meta)

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » L » **LINE (Meta)**
Syntax: #LINE Zeilennummer ["Modulname"]
Typ: Metabefehl
Kategorie: Metabefehle

#LINE setzt die aktuelle Zeilennummer und den Dateinamen des Moduls. Diese Festlegungen haben keinen Einfluss auf die eigentliche Programmausführung, sie dienen lediglich dem Programmierer, der auf diese Art Teile seines Programms 'markieren' kann.

- 'Zeilennummer' ist ein positiver Zahlenwert oder null, der festlegt, welche Zeilennummer der #LINE-Anweisung zugewiesen werden soll. Nachfolgende Compiler-Warnungen und Fehlermeldungen, die sich auf Zeilen hinter der #LINE-Anweisung beziehen, werden davon beeinflusst.
- 'Modulname' ist ein String, der den Namen des Moduls enthält, wie er in Fehlermeldungen ausgegeben werden soll. Wird dieser Parameter ausgelassen, so behält FreeBASIC den ursprünglichen Namen bei.

Die FB-eigenen vordefinierten Symbole __LINE__ und __FILE__ werden durch #LINE automatisch aktualisiert. Ebenso beachtet ERMN den hier bestimmten Modulnamen. Daneben ändern sich auch die Compiler- und Laufzeit-Fehlermeldungen.

Durch diese Direktive können Programme andere Quellcodes einbinden und für diese Warnungen und Fehlermeldungen ausgegeben, die auf die richtigen Stellen im Fremd-Programm verweisen.

Beispiel:

```
"hlzahl">155 "outside.src"

Error 1000
```

Ausgabe:

```
Aborting due to runtime error 1000 at line 157 of outside.src::()
```

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC: existiert seit FreeBASIC v0.17

Siehe auch:
__LINE__, __FILE__, ERMN, Präprozessor-Anweisungen

Letzte Bearbeitung des Eintrags am 27.12.12 um 13:53:04
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

LINE INPUT

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » L » **LINE INPUT**
Syntax: LINE INPUT [;] [Frage { , | ; }] Variable
Typ: Anweisung
Kategorie: Benutzereingabe

LINE INPUT gibt dem Benutzer die Möglichkeit, über Tastatur eine Textzeile einzugeben.

- Wird vor 'Frage' ein Strichpunkt angegeben, so bleibt der Cursor nach der Benutzereingabe an der Stelle stehen, an der die Eingabe endet. Ansonsten wird nach der Eingabe ein Zeilenumbruch durchgeführt.
- 'Frage' ist ein STRING, ZSTRING oder WSTRING, der auf dem Bildschirm an der aktuellen Cursorposition angezeigt wird, bevor der Benutzer seine Eingabe tätigen kann..
- Wird als Trennzeichen zwischen 'Frage' und 'Variable' ein Strichpunkt statt eines Kommas gesetzt, so wird an das Ende von 'Frage' ein Fragezeichen angehängt.
- 'Variable' ist eine Variable, die vom Typ STRING, ZSTRING oder WSTRING sein muss. Die Eingabe des Benutzers wird in dieser Variablen gespeichert.

Im Gegensatz zu INPUT gibt es bei LINE INPUT nur die Möglichkeit, eine einzelne Variable einzugeben. Diese muss zudem von einem String-Datentyp sein. Dafür treten keine Probleme mit Sonderzeichen wie Kommata (,) oder "Anführungszeichen" auf. Außerdem darf es sich bei 'Frage' auch um eine Variable oder einen zusammengesetzten Stringausdruck handeln. Wird 'Frage' ausgelassen, so wird vor der Benutzereingabe kein Text (auch kein Fragezeichen wie bei INPUT) ausgegeben.

Beispiel:

```
DIM AS STRING benutzername, eingabe
LINE INPUT "Gib deinen Namen ein: ", benutzername
LINE INPUT "Hallo, " & benutzername & ". Willst du noch etwas sagen";
eingabe
```

Unterschiede zu QB:
In QB kann für 'Frage' nur ein String-Literal angegeben werden. In FreeBASIC sind auch Variablen und zusammengesetzte Stringausdrücke erlaubt.

Siehe auch:
LINE INPUT "reflinkicon" href="temp0206.html">INPUT (Anweisung), Benutzereingaben

Letzte Bearbeitung des Eintrags am 06.04.12 um 22:20:46
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

LINE INPUT (Datei)

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » L » **LINE INPUT (Datei)**
Syntax: LINE INPUT #Dateinummer, Stringvariable
Typ: Anweisung
Kategorie: Dateien

LINE INPUT # liest eine ganze Zeile aus einer Datei, die im Dateimodus INPUT oder BINARY geöffnet wurde.

- 'Dateinummer' ist die Nummer, die beim Öffnen der Datei mit OPEN zugewiesen wurde.
- 'Stringvariable' ist eine Variable, die als STRING deklariert sein muss. In dieser Variablen wird die ausgelesene Zeile gespeichert.

Beispiel: Datei zeilenweise auslesen und ausgeben

```
DIM zeile AS STRING, f AS INTEGER = FREEFILE
OPEN "Datei.txt" FOR INPUT AS "hlkw0">DO UNTIL EOF(f)
   LINE INPUT "hlzeichen">, zeile
   PRINT zeile
LOOP

CLOSE "hlkw0">SLEEP
```

Siehe auch:
LINE INPUT, INPUT "reflinkicon" href="temp0318.html">PRINT #, WRITE #, OPEN (Anweisung), INPUT (Dateimodus), BINARY, Dateien (Files)

Letzte Bearbeitung des Eintrags am 27.12.12 um 13:53:21
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

LOBYTE

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » L » **LOBYTE**
Syntax: LOBYTE (Ausdruck)
Typ: Funktion
Kategorie: Speicher

LOBYTE gibt das niedere Byte eines Ausdrucks als UINTEGER zurück. LOBYTE hat dieselbe Funktion wie

```
Ausdruck AND 255
```

Beispiel:

```
Dim As Integer foo = &b10000100000 ' = 1056 dezimal
PRINT LOBYTE(foo)
PRINT foo AND 255
SLEEP
```

Ausgabe:

```
32
32
```

Intern wird LOBYTE folgendermaßen behandelt:

```
"cnf">__LOBYTE in der Dialektform -lang qb existiert seit FreeBASIC
v0.24.
```

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht LOBYTE nicht zur Verfügung und kann nur über **__LOBYTE** aufgerufen werden.

Siehe auch:
HIBYTE, HIWORD, LOWORD, Bit-Operatoren
Letzte Bearbeitung des Eintrags am 27.12.12 um 13:53:44
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

LOC

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » L » **LOC**
Syntax: LOC (Dateinummer)
Typ: Funktion
Kategorie: Dateien

LOC gibt die Position des Zeigers innerhalb einer mit OPEN geöffneten Datei zurück.

- Wurde die Datei im Dateimodus BINARY geöffnet, dann wird die Position des Bytes zurückgegeben, das zuletzt gelesen wurde. Zu Beginn steht LOC also auf Position 0, nach dem Lesen eines Bytes auf Position 1 usw. LOC gibt damit immer einen Wert zurück, der um 1 kleiner ist als SEEK.
- Im Dateimodus RANDOM wird die Nummer des zuletzt gelesenen Datensatzes zurückgegeben.
- Bei den sequentiellen Dateimodi INPUT, OUTPUT und APPEND wird eine Datensatzlänge von 128 Byte angenommen. LOC(Dateinummer) verhält sich damit wie (SEEK(Dateinummer) - 1) \ 128.

Beispiel:

```
Dim As UByte dummy, nr=FreeFile
DIM AS INTEGER firstPos
OPEN "mydata.ext" FOR BINARY AS "hlkw0">DO UNTIL LOC(nr) = LOF(nr)
   GET "hlzeichen">, , dummy
   IF dummy = ASC("A") THEN EXIT DO
LOOP
firstPos = LOC(nr)
If firstPos = LOF(nr) THEN firstPos = 0
CLOSE "hlkw0">IF firstPos THEN
   PRINT "In dieser Datei kommt 'A' zum ersten ";
   PRINT "Mal an Byte " & firstPos &" vor."
ELSE
   PRINT "In dieser Datei kommt 'A' nie vor."
END IF
SLEEP
```

Unterschiede zu QB:
Die Berechnung bei sequentiellen Dateimodi unterscheidet sich in QB etwas.

Siehe auch:
OPEN, LOF, EOF, SEEK (Funktion), SEEK (Anweisung), Dateien (Files)

Letzte Bearbeitung des Eintrags am 27.12.12 um 13:53:59
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

LOCAL

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » L » **LOCAL**
Syntax: ON LOCAL ERROR GOTO { label | 0 }
Typ: Schlüsselwort
Kategorie: Fehlerbehandlung

LOCAL wird zusammen mit ON ERROR benutzt. LOCAL bewirkt, dass die Fehlerbehandlungsroutine nur für die gerade aktive Prozedur gilt und nicht für das gesamte Modul.

Beispiel:

```
ON ERROR GOTO errorhandler              ' globale Fehlerroutine setzen

SUB Test
  ON LOCAL ERROR GOTO suberrorhandler  ' Fehlerroutine lokal ersetzen
  ERROR 24

  suberrorhandler:
  PRINT "In der SUB 'Test' ist der Fehler " & ERR & " aufgetreten!"
  PRINT "Beliebige Taste zum Beenden"
  SLEEP
  END
END SUB

Test
END

errorhandler:
' Diese Fehlerroutine kommt nicht zum Einsatz
PRINT "Fehler " & ERR & " ist aufgetreten!"   'Fehlernummer anzeigen
PRINT "Beliebige Taste zum Beenden"
SLEEP
END
```

Unterschiede zu QB: neu in FreeBASIC

Siehe auch:
ON ERROR, Fehlerbehandlung, Debugging

Letzte Bearbeitung des Eintrags am 17.08.12 um 22:55:03
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

LOCATE (Anweisung)

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » L » **LOCATE (Anweisung)**
Syntax: LOCATE [Zeile] [, Spalte] [, sichtbar] [, Start] [, Ende]
Typ: Anweisung
Kategorie: Konsole

LOCATE setzt die Position des Cursors in die angegebene Zeile und Spalte.

- 'Zeile' ist die Y-Koordinate, also der Hochwert. Die erste Zeile trägt den Wert 1. Wird dieser Parameter ausgelassen, verwendet FreeBASIC die aktuelle Cursorzeile.
- 'Spalte' ist die X-Koordinate, also der Rechtswert. Die erste Spalte trägt den Wert 1. Wird dieser Parameter ausgelassen, verwendet FreeBASIC die aktuelle Cursorspalte.
- 'sichtbar' gibt an, ob ein Cursor angezeigt werden soll oder nicht. Wenn ein Cursor angezeigt werden soll, muss 'sichtbar' ungleich null sein. Wird dieser Parameter ausgelassen, so wird die Sichtbarkeit des Cursors nicht verändert.
- 'Start' und 'Ende' sind nur in der Dialektform -lang qb erlaubt und existieren aus Kompatibilitätsgründen zu QB. Der Wert dieser Parameter wird ignoriert.

Bei den Werten für Zeile und Spalte handelt es sich um die Y bzw. X-Koordinate des Textrasters mit Ursprung (1,1) in der linken oberen Ecke.

Beispiel:

```
LOCATE 5, 10, 0
PRINT "Zeile 5, Spalte 10"
PRINT "Zeile 6, Spalte  1"
PRINT "Kein Cursor."
PRINT
GETKEY
LOCATE , , 1
PRINT "Cursor aktiv"
SLEEP
```

Hinweis:
Der Cursor ist nur im Textmodus sichtbar, nicht dagegen im Grafikmodus (siehe SCREENRES). Will man im Grafikmodus dennoch einen Cursor anzeigen, muss er selbst auf den Bildschirm gezeichnet werden.

Unterschiede zu QB:
Die optionalen Parameter 'Start' und 'Ende' haben in FreeBASIC keine Auswirkung.

Unterschiede zu früheren Versionen von FreeBASIC:
Die optionalen Parameter 'Start' und 'Ende' existieren seit FreeBASIC v0.17

Unterschiede unter den FB-Dialektformen:
Die Parameter 'Start' und 'Ende' sind nur in der Dialektform -lang qb erlaubt.

Siehe auch:
LOCATE (Funktion), SPC, CSRLIN (Funktion), POS (Funktion), WIDTH (Anweisung), Konsole

Letzte Bearbeitung des Eintrags am 01.10.13 um 13:27:59
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

LOCATE (Funktion)

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » L » **LOCATE (Funktion)**
Syntax: LOCATE
Typ: Funktion
Kategorie: Konsole

LOCATE gibt Informationen über die aktuelle Cursorposition und die Sichtbarkeit des Cursors zurück.

Der zurückgegebene Wert wird folgendermaßen berechnet:

```
(Spalte OR (Zeile SHL 8) OR (sichtbar SHL 16))
```

Das untere Byte ist also die Spalte, das obere Byte die Zeile, und im oberen Word wird das Sichtbarkeits-Flag angegeben.
Bei den Werten für Zeile und Spalte handelt es sich um die Y bzw. X-Koordinate des Textrasters mit Ursprung (1,1) in der linken oberen Ecke.

Beispiel:

```
DIM AS INTEGER pst, row, lin, vis

pst = LOCATE
row = LOBYTE(pst)
lin = HIBYTE(pst)
vis = HIWORD(pst)

PRINT "Cursorposition:"
PRINT "Line: "; lin
PRINT "Row:  "; row
Print "visible: "; vis

PRINT

LOCATE 10, 2, 0

pst = LOCATE
row = LOBYTE(pst)
lin = HIBYTE(pst)
vis = HIWORD(pst)

PRINT "Cursorposition:"
PRINT "Line: "; lin
PRINT "Row:  "; row
Print "visible: "; vis

SLEEP
```

Unterschiede zu QB:
Der Einsatz von LOCATE als Funktion ist neu in FreeBASIC.

Siehe auch:
LOCATE (Anweisung), CSRLIN (Funktion), POS (Funktion), WIDTH (Funktion), Konsole

Letzte Bearbeitung des Eintrags am 01.10.13 um 13:27:15
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

LOCK

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » L » **LOCK**
Syntax: LOCK Dateinummer, { Satznummer | Start TO Ende }
Typ: Anweisung
Kategorie: Dateien

Achtung: LOCK arbeitet zur Zeit noch nicht wie vorgesehen. Siehe dazu auch den Diskussionsbeitrag im englischen Forum.

LOCK sperrt den Zugriff auf eine Datei. Dieser Befehl ist nützlich, wenn mehrere Programme, Benutzer oder Prozesse auf dieselbe Datei zuzugreifen versuchen.

- 'Dateinummer' ist die Nummer, die beim Öffnen der Datei mit OPEN zugewiesen wurde.
- Im Dateimodus RANDOM ist 'Satznummer' die Nummer eines Datensatzes, auf den der Zugriff gesperrt werden soll. Im BINARY-Modus gibt 'Start' und 'Ende' den zu sperrenden Datenbereich an.

Achtung: Durch LOCK wird zur Zeit immer die gesamte Datei gesperrt; trotzdem ist die Angabe von 'Satznummer' bzw. von 'Start' und 'Ende' verbindlich!

Eine Datei kann auch bereits beim Öffnen für Lese- und/oder Schreibzugriffe gesperrt werden; siehe dazu OPEN (Anweisung)

UNLOCK entsperrt eine Datei wieder. Für UNLOCK gilt dieselbe Syntax wie für LOCK.

Dateien, die noch nicht mit UNLOCK freigegeben wurden, sollten nicht ein zweites Mal mit LOCK gesperrt werden!

Beispiel:
Eine Datei sperren, 100 Bytes lesen, und anschließend wieder entsperren. Um dieses Programm ausführen zu können, sollten Sie sicherstellen, dass die Datei 'file.ext' existiert, sich im aktuellen Arbeitsverzeichnis befindet und mindestens 100 Bytes lang ist.

```
DIM AS UBYTE array(1 to 100), f = FREEFILE
OPEN "file.ext" FOR BINARY AS "hlkw0">LOCK "hlzeichen">, 1 TO 100
   FOR i AS INTEGER = 1 TO 100
      GET "hlzeichen">, i, array(i)
   NEXT
   UNLOCK "hlzeichen">, 1 TO 100
CLOSE "reflinkicon" href="temp0433.html">UNLOCK, OPEN, Dateien (Files)
```

Letzte Bearbeitung des Eintrags am 27.12.12 um 13:54:39
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

LOF

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » L » **LOF**
Syntax: LOF (Dateinummer)
Typ: Funktion
Kategorie: Dateien

LOF (**L**en **O**f **F**ile) gibt die Länge einer geöffneten Datei in Bytes zurück. Wurde OPEN COM verwendet, dann gibt LOF die Länge der Daten zurück, die im Buffer zum Lesen bereitstehen.

Beispiel:

```
DIM AS INTEGER nr = FREEFILE
OPEN "file.ext" FOR BINARY AS "hlkw0">PRINT LOF(nr)
CLOSE "hlkw0">SLEEP
```

Siehe auch:
OPEN, LOC, SEEK (Funktion), EOF, Dateien (Files)

Letzte Bearbeitung des Eintrags am 27.12.12 um 13:54:55
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

LOG

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » L » **LOG**
Syntax: LOG (Zahl)
Typ: Funktion
Kategorie: Mathematik

LOG gibt den Logarithmus zur Eulerschen Zahl e zurück. e ist näherungsweise 2.718281828459045 und kann durch EXP(1) zurückgegeben werden.

- 'Zahl' ist eine beliebiger numerischer Ausdruck. Variablen, Konstanten, Operatoren und Funktionen sind erlaubt. Der Ausdruck darf von jeden Datentyp außer STRING, ZSTRING oder WSTRING sein. Dieser Wert muss größer als null sein, andernfalls wird ein Fehler erzeugt.
- Der Rückgabewert ist ein DOUBLE, der den Logarithmus von 'Zahl' zur Basis e enthält.

LOG ist die Gegenfunktion zu EXP.

Es kann zu Unklarheiten kommen, da in der Mathematik der natürliche Logarithmus (zur Basis e) in der Regel mit **LN** bezeichnet wird, während der dekadische Logarithmus (zur Basis 10) meist mit LOG bezeichnet wird. In FreeBASIC bezeichnet LOG - wie in den meisten Programmiersprachen - den natürlichen Logarithmus.

Der Logarithmus ist die Gegenfunktion zu Potenzen; betrachtet man die Gleichung

```
x = e ^ k
```

so gilt:

```
k = LOG(x)
```

LOG kann mithilfe von OPERATOR überladen werden.

Beispiel:
Um den Exponenten 'k' zu einer beliebigen Basis 'b' zu berechnen, der die Zahl 'x' ergibt, benutzen Sie folgenden Code:

```
' x = b ^ k <=> k = LogX(x, b)
DECLARE FUNCTION LogX (x AS DOUBLE, basis AS DOUBLE) AS DOUBLE

FUNCTION LogX (x AS DOUBLE, basis AS DOUBLE)
   LogX = LOG(x) / LOG(basis)
END FUNCTION

PRINT LogX(100, 10) 'Ergebnis: 2, denn 10 ^ 2 = 100
SLEEP
```

Unterschiede zu früheren Versionen von FreeBASIC:
Die Überladung von LOG für benutzerdefinierte Datentypen ist seit FreeBASIC v0.22 möglich.

Siehe auch:
EXP, mathematische Funktionen

Letzte Bearbeitung des Eintrags am 27.12.12 um 13:55:24
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

LONG

Typ: Datentyp

LONG-Zahlen sind vorzeichenbehaftete 32-bit-Ganzzahlen. Sie liegen im Bereich von -(2^31) bis (2^31)-1, bzw. von -2'147'483'648 bis 2'147'483'647.

Siehe auch:
SIZEOF, ANY, PTR, DIM, CAST, CLNG, Datentypen, Datentypen und Deklarationen

Letzte Bearbeitung des Eintrags am 30.06.13 um 14:15:13
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

LONGINT

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » L » **LONGINT**
Typ: Datentyp

LONGINT-Zahlen sind vorzeichenbehaftete 64-bit-Ganzzahlen. Sie liegen im Bereich von -(2^63) bis (2^63)-1, bzw. von -9'223'372'036'854'775'808 bis 9'223'372'036'854'775'807.

Unterschiede zu QB: neu in FreeBASIC

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht LONGINT nicht zur Verfügung und kann nur über **__LONGINT** aufgerufen werden.

Siehe auch:
DIM, CAST, CLNGINT, Datentypen, Datentypen und Deklarationen

Letzte Bearbeitung des Eintrags am 28.12.12 um 00:44:24
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

LOOP

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » L » **LOOP**
Siehe DO ... LOOP

Letzte Bearbeitung des Eintrags am 28.08.11 um 18:07:39
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

LOWORD

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » L » **LOWORD**
Syntax: LOWORD (Ausdruck)
Typ: Funktion
Kategorie: Speicher

LOWORD gibt das niedere Word eines Ausdrucks als UINTEGER zurück. LOWORD hat dieselbe Funktion wie

```
Ausdruck AND 65535
```

Beispiel:

```
Dim As Integer foo = &b11000000000000000 ' = 98304 dezimal
PRINT LOWORD(foo)
PRINT foo AND 65535
SLEEP
```

Ausgabe:

```
32768
32768
```

Intern wird LOWORD folgendermaßen behandelt:

```
"cnf">__LOWORD in der Dialektform -lang qb existiert seit FreeBASIC
v0.24.
```

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht LOWORD nicht zur Verfügung und kann nur über **__LOWORD** aufgerufen werden.

Siehe auch:
HIWORD, HIBYTE, LOBYTE, Bit-Operatoren
Letzte Bearbeitung des Eintrags am 27.12.12 um 13:57:15
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

LPOS

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » L » **LPOS**
Syntax: LPOS (n)
Typ: Funktion
Kategorie: Hardware

LPOS gibt die Anzahl der Zeichen zurück, die seit dem letzten Zeilenumbruch an den Drucker gesendet wurden.

'n' ist die Nummer des LPT-Ports, an dem der Drucker angeschlossen wurde. Dieser Wert liegt zwischen 0 und 3. 0 und 1 stehen für LPT1, 2 für LPT2 und 3 für LPT3.

Beispiel:

```
"hlstring">"fblite"

Dim test As String = "LPRINT-Beispiel"

Print "Sende '" & test & "' to LPT1 (Standarddrucker)"
LPrint test
Print "LPT1 hat zuletzt "  & Lpos(1) & " Zeichen empfangen."
Print "Der gesendete String war " & Len(test) & " Zeichen lang."

Sleep
```

Siehe auch:
LPRINT (Anweisung), POS, Hardware-Zugriffe

Letzte Bearbeitung des Eintrags am 17.08.12 um 23:22:01
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

LPRINT (Anweisung)

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » L » **LPRINT (Anweisung)**
Syntax: LPRINT Printausdruck
Typ: Anweisung
Kategorie: Hardware

LPRINT sendet Daten an den Standarddrucker.

'Printausdruck' ist ein Ausdruck, der wie bei PRINT (Anweisung) formatiert sein muss; er unterliegt den dort genannten Regeln. LPRINT schreibt immer auf den Drucker an LPT1. Um an einen anderen Drucker zu senden, verwenden Sie OPEN LPT.

Einige Drucker beginnen mit dem Druckauftrag erst, wenn ein CHR(12) (End of Page) gedruckt wird.

Beispiel:

```
"hlstring">"fblite"

LPrint "Hallo Welt!"          ' mit Zeilenumbruch
LPrint "Hallo"; "Welt"; "!"; ' ohne Zeilenumbruch
LPrint                        ' Zeilenumbruch
LPrint "Hallo!", "Welt!"      ' mit Tabulator
LPrint Chr(12)                ' Seitenende
```

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang fb steht LPRINT nicht zur Verfügung.

Siehe auch:
LPRINT USING, LPOS, PRINT (Anweisung), OPEN LPT, PRINT "reflinkicon" href="temp0593.html">Hardware-Zugriffe

Letzte Bearbeitung des Eintrags am 27.05.12 um 21:23:51
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

LPRINT USING

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » L » **LPRINT USING**
Syntax: LPRINT USING formatstring, printausdruck
Typ: Anweisung
Kategorie: Hardware

LPRINT sendet Daten an den Standarddrucker.

- 'formatstring' ist ein STRING, der bestimmte Formatierungszeichen enthält. Siehe PRINT USING für weitere Informationen.
- 'printausdruck' ein Ausdruck, der wie bei PRINT (Anweisung) formatiert sein muss; er unterliegt den dort genannten Regeln.

Einige Drucker beginnen mit dem Druckauftrag erst, wenn ein CHR(12) (End of Page) gedruckt wird.

LPRINT schreibt immer auf den Drucker an LPT1. Um an einen anderen Drucker zu senden, verwenden Sie OPEN LPT.

Beispiele: siehe LPRINT und PRINT USING

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang fb steht LPRINT nicht zur Verfügung.

Siehe auch:
LPRINT (Anweisung), LPOS, OPEN LPT, PRINT #, PRINT USING, Hardware-Zugriffe

Letzte Bearbeitung des Eintrags am 27.05.12 um 21:33:43
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

LPT

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » L » **LPT**
Siehe OPEN LPT

Letzte Bearbeitung des Eintrags am 28.08.11 um 17:50:57
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

LSET

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » L » **LSET**
Syntax: LSET Ziel, Quelle
Typ: Anweisung
Kategorie: Stringfunktionen

LSET befüllt das 'Ziel' mit 'Quelle', behält aber die Länge von 'Ziel' bei. Es wird benutzt, um Strings linksbündig zu formatieren.

- 'Ziel' ist ein STRING oder ein UDT, der befüllt werden soll.
- 'Quelle' enthält die Daten, mit denen 'Ziel' befüllt wird. Ist 'Ziel' ein String, so muss auch 'Quelle' ein String sein. Ist 'Ziel' ein UDT, so muss 'Quelle' ein UDT sein, jedoch nicht notwendigerweise desselben Typs.

War 'Ziel' vor der Bearbeitung mit LSET z. B. ein fünf Zeichen langer String, so ist er auch nach der Bearbeitung noch fünf Zeichen lang, unabhängig von der Länge von 'Quelle'. Ist 'Quelle' länger als 'Ziel', so werden die vorderen Zeichen des Strings nach 'Ziel' kopiert, der Rest von 'Quelle' wird ignoriert. Ist 'Quelle' kürzer als 'Ziel', so werden die restlichen Zeichen von 'Ziel' mit Leerzeichen aufgefüllt. In allen Fällen bleibt 'Quelle' unverändert.

Beispiel:

```
DIM Quelle AS STRING
DIM Ziel AS STRING

Ziel = "0123456789"
Quelle = "***"
PRINT Ziel, Quelle

LSET  Ziel, Quelle
PRINT Ziel, Quelle
SLEEP
```

Ausgabe:

```
0123456789    ***
***           ***
```

Bei der Verwendung von UDTs werden die Daten Byte für Byte kopiert; der Programmierer muss selbst darauf achten, dass das Ergebnis valide ist.

Beispiel:

```
Type mytype1
  x As byte
  y As short
End Type

Type mytype2
  z As Integer
End Type

Dim a As mytype1 , b As mytype2
```

```
b.z = &h01020304

LSet a, b
Print "&h" & hex(a.x, 2), "&h" & hex(a.y, 4)
Sleep
```

Ausgabe:

```
&h04    &h0102
```

Das hinterste Byte des Integers 'b.z' wurde also nach 'a.x' kopiert, die beiden vorderen nach 'a.y' - das dritte Byte ging wegen des Paddings verloren.

Unterschiede zu QB:
In QB lautet die Syntax LSET Ziel=Quelle.

Siehe auch:
RSET, SPACE, String-Funktionen

Letzte Bearbeitung des Eintrags am 27.12.12 um 13:58:37
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

LTRIM

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » L » **LTRIM**
Syntax: LTRIM[$] (Stringausdruck [, [ANY] zuEntfernen])
Typ: Funktion
Kategorie: Stringfunktionen

LTRIM (Left TRIM) entfernt bestimmte führende Zeichen aus einem String.

- 'Stringausdruck' ist der STRING, ZSTRING oder WSTRING, der gekürzt werden soll.
- 'zuEntfernen' ist ein Ausdruck, der angibt, welche Zeichen entfernt werden sollen. Wird dieser Parameter ausgelassen, entfernt FreeBASIC automatisch alle Leerzeichen am Anfang des Strings. 'zuEntfernen' darf aus mehreren Zeichen bestehen.
- Wird die ANY-Klausel verwendet, entfernt FreeBASIC jedes Zeichen aus 'zuEntfernen' am Anfang von 'Stringausdruck'.
- Der Rückgabewert ist der um die angegebenen Zeichen gekürzte String.

LTRIM arbeitet *case-sensitive*, d. h. die Groß-/Kleinschreibung ist ausschlaggebend.

Das Dollarzeichen ($) als Suffix ist optional.

Beispiel:

```
PRINT LTRIM(" hello")
PRINT LTRIM("hello World", "he")
PRINT LTRIM("hello World", "eh")
PRINT LTRIM("hello World", ANY "eh")
SLEEP
```

Ausgabe:

```
hello
llo World
hello World
llo World
```

Unterschiede zu QB:

- Der Parameter 'zuEntfernen' und die ANY-Klausel sind neu in FreeBASIC.
- In QB ist das Suffix $ verbindlich.
- Da QB kein Unicode unterstützt, existiert dort auch nicht die Möglichkeit, WSTRING zu verwenden.

Plattformbedingte Unterschiede:
Unicode wird in der DOS-Portierung von FreeBASIC nicht unterstützt, deshalb kann dort auch kein WSTRING verwendet werden.

Unterschiede zu früheren Versionen von FreeBASIC:
Der Parameter 'zuEntfernen' und die ANY-Klausel existieren seit FreeBASIC v0.15

Unterschiede unter den FB-Dialektformen:

- In der Dialektform -lang qb ist das Suffix $ verbindlich.
- In den Dialektformen -lang fblite und -lang fb ist das Suffix optional.

Siehe auch:
RTRIM, TRIM, INSTR, String-Funktionen

Letzte Bearbeitung des Eintrags am 27.12.12 um 13:58:53
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

MACRO (Meta)

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » M » **MACRO (Meta)**
Syntax:

```
"hlzeichen">( [Parameterliste [...]] )
' Makro-Code
"reflinkicon" href="temp0108.html">#DEFINE ein Makro, ermöglicht aber die
Erstellung mehrzeiliger Makros.
```

- 'Bezeichner' ist der Name des zu erstellenden Makros.
- 'Parameterliste' ist eine Liste von Parametern, von denen das Makro abhängig ist. Dafür müssen Variablen verwendet werden, die bis dahin noch nicht verwendet wurden. Die Parameterliste ist optional.
- Die Verwendung der Ellipsis ... (Auslassung) hinter dem letzten Parameter eines Makros erlaubt es, Makros mit variabler Anzahl von Parametern zu erzeugen. Siehe dazu #DEFINE.
- 'Makro-Code' ist ein normaler FreeBASIC-Programmcode. In ihm dürfen die in 'Parameterliste' spezifizierten Variablen vorkommen.

Taucht nach der Definition eines Makros sein Bezeichner im Code auf, so wird der Bezeichner durch den Makro-Code ersetzt. Ebenso ersetzt FreeBASIC die Parameter der Parameterliste durch die beim Aufruf angegebenen Parameter. Das Auslassen von Parametern ist bei Makros nicht möglich.

Mit #IFDEF, #IFNDEF und DEFINED kann überprüft werden, ob ein Makro definiert wurde. Mit #UNDEF kann ein Makro gelöscht werden, z. B. um den Bezeichner als Variable zu benutzen oder um ihn mit einem neuen Makro zu belegen.

Beispiel 1:

```
"hlkw0">Add(a, b)
  a + b
"hlkw0">PRINT Add("Hello", " World")
PRINT Add(1, 2)
```

Ausgabe:

```
Hello World
 3
```

Beispiel 2:

```
"hlzeichen">(a, b)
  Print a;
  Print " ";
  Print b;
  Print "!"
"hlzeichen">("Hello", "World")
```

Ausgabe:

```
Hello World!
```

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC:

- `Seit der SVN-Version 0.22 kann man eine variable Anzahl von Parametern verwenden.`
- `MACRO existiert seit FreeBASIC v0.17.`

Siehe auch:
#ENDMACRO, DEFINE (Meta), UNDEF (Metabefehl), IFDEF (Meta), IFNDEF (Meta), DEFINED, Präprozessoren, Präprozessor-Anweisungen

Letzte Bearbeitung des Eintrags am 01.01.13 um 00:36:08
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

MID (Anweisung)

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » M » **MID (Anweisung)**
Syntax: MID[$](Text, Start[, Anzahl]) = Ausdruck
Typ: Anweisung
Kategorie: Stringfunktionen

MID ersetzt einen Teil einer Zeichenkette durch eine andere.

- 'Text' ist ein STRING, ZSTRING oder WSTRING, der verändert werden soll.
- 'Start' ist die Nummer des Zeichens, ab dem die Änderung beginnen soll. Enthält 'start' einen größeren Wert, als 'Text' lang ist, oder einen Wert kleiner als 1, dann wird nichts geändert.
- 'Anzahl' gibt an, wie viele Zeichen geändert werden sollen. Wird 'Anzahl' ausgelassen, richtet sich die Anzahl der zu ändernden Zeichen nach der Länge des nachfolgenden 'Ausdruck'. Ist 'Anzahl' größer, als 'Ausdruck' lang ist, wird automatisch die Länge von 'Ausdruck' verwendet. Ist 'Ausdruck' länger, als 'Anzahl' groß ist, dann werden nur die ersten 'Anzahl' Zeichen aus 'Ausdruck' verwendet.

In jedem Fall bleibt die Länge von 'Text' erhalten. Sollen mehr Zeichen ersetzt werden, als 'Text' aufnehmen kann, dann findet nur eine Ersetzung bis zum Ende von 'Text' statt.

Das Dollarzeichen ($) als Suffix ist optional.

Beispiel:

```
DIM text AS STRING
text = "abc 123"
PRINT text                    ' Ausgabe: "abc 123"
MID(text, 5, 3) = "4" & "56"
PRINT text                    ' Ausgabe: "abc 456"
MID(text, 1, 2) = "def"
PRINT text                    ' Ausgabe: "dec 456"
SLEEP
```

Wenn man eine Anzahl von Zeichen durch eine *andere* Anzahl von Zeichen ersetzen will, kann diese Anweisung nicht eingesetzt werden. Stattdessen muss man den neue String selbst zusammensetzen.

Beispiel:

```
DIM AS STRING text = "Schönen Abend!"
DIM AS INTEGER suche

' Da Umlaute u. U. nicht korrekt dargestellt werden, wird 'ö' durch 'oe'
ersetzt.
suche = INSTR(text, "ö")
IF suche > 0 THEN       ' 'ö' gefunden
  ' Umlaut ersetzen
  text = LEFT(text, suche-1) & "oe" & MID(text, suche+1)
END IF
PRINT text
SLEEP
```

Hinweis: Das Beispiel funktioniert nur einwandfrei, wenn es in einem Format wie ISO 8859-1 o. ä. gespeichert wurde, bei dem der Buchstabe "ö" nur ein Byte Speicherplatz benötigt.

Unterschiede zu QB:

- In QB ist das Suffix $ verbindlich.
- Da QB kein Unicode unterstützt, existiert dort auch nicht die Möglichkeit, WSTRING zu verwenden.

Plattformbedingte Unterschiede:
Unicode wird in der DOS-Portierung von FreeBASIC nicht unterstützt, deshalb kann dort auch kein WSTRING verwendet werden.

Unterschiede unter den FB-Dialektformen:

- In der Dialektform -lang qb ist das Suffix $ verbindlich.
- In den Dialektformen -lang fblite und -lang fb ist das Suffix optional.

Siehe auch:
MID (Funktion), String-Funktionen

Letzte Bearbeitung des Eintrags am 27.12.12 um 14:41:37
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

MID (Funktion)

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » M » **MID (Funktion)**
Syntax: MID[$](Ausdruck, Start[, Länge])
Typ: Funktion
Kategorie: Stringfunktionen

MID gibt einen Ausschnitt einer Zeichenkette zurück.

- 'Ausdruck' ist der STRING, ZSTRING oder WSTRING, der analysiert werden soll.
- 'Start' ist die Position des ersten Zeichens innerhalb des Strings, das zurückgegeben werden soll.
- 'Länge' ist die Anzahl der Zeichen, die zurückgegeben werden sollen. Wenn 'Länge' ausgelassen wird oder negativ ist, gibt MID alle Zeichen des STRINGs ab 'Start' zurück; MID funktioniert dann so ähnlich wie RIGHT.
- Der Rückgabewert ist ein STRING bzw. WSTRING, der den angeforderten Teilstring enthält.

Wird für einen der Parameter 0 (bzw. ein Leerstring) angegeben, ist der Rückgabewert ein Leerstring. Ist die Anzahl der zurückzugebenden Zeichen größer als der verbliebene Rest des Strings, dann werden nur die Zeichen bis zum Stringende zurückgegeben.

Das Dollarzeichen ($) als Suffix ist optional.

Beispiel:

```
PRINT MID("abcdefg"     , 3, 2) 'Ausgabe: "cd"
PRINT MID("abcdefg"     , 3)    'Ausgabe: "cdefg"
PRINT MID("abc" & "defg", 2, 1) 'Ausgabe: "b"
Print Mid("Test!", 2, 20) 'Ausgabe: "est!"
```

Unterschiede zu QB:

- In QB ist das Suffix $ verbindlich.
- Da QB kein Unicode unterstützt, existiert dort auch nicht die Möglichkeit, WSTRING zu verwenden.

Plattformbedingte Unterschiede:
Unicode wird in der DOS-Portierung von FreeBASIC nicht unterstützt, deshalb kann dort auch kein WSTRING verwendet werden.

Unterschiede unter den FB-Dialektformen:

- In der Dialektform -lang qb ist das Suffix $ verbindlich.
- In den Dialektformen -lang fblite und -lang fb ist das Suffix optional.

Siehe auch:
MID (Anweisung), INSTR, LEFT, RIGHT, String-Funktionen

Letzte Bearbeitung des Eintrags am 27.12.12 um 14:41:51
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

MINUTE

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » M » **MINUTE**
Syntax: MINUTE (Serial)
Typ: Funktion
Kategorie: Datum und Zeit

Anmerkung:
Um diese Funktion nutzen zu können, muss die Datei datetime.bi in Ihren Quellcode eingebunden werden, z. B. mit INCLUDE. Alternativ können Sie auch die Datei vbcompat.bi einbinden, da dadurch auch automatisch datetime.bi eingebunden wird.

MINUTE extrahiert die Minute einer Serial Number.

- 'Serial' ist ein DOUBLE-Wert, der als Serial Number behandelt wird.
- Der Rückgabewert ist die Minute, die in der Serial Number gespeichert ist.

Beispiel: Die aktuelle Minute aus NOW extrahieren:

```
"hlstring">"vbcompat.bi"
DIM min AS INTEGER
min = MINUTE(NOW)
```

Unterschiede zu QB: existiert nur in QBX PDS und in VBDOS

Unterschiede zu früheren Versionen von FreeBASIC: existiert seit FreeBASIC v0.15

Siehe auch:
NOW, TIMESERIAL, TIMEVALUE, HOUR, SECOND, FORMAT, Serial Numbers, Datum und Zeit

Letzte Bearbeitung des Eintrags am 27.12.12 um 14:42:21
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

MKD

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » M » **MKD**
Syntax: MKD[$](DOUBLE-Ausdruck)
Typ: Funktion
Kategorie: Typumwandlung

MKD verwandelt eine DOUBLE-Zahl in einen 8-Byte-STRING. Die Funktion wurde in älteren BASIC-Dialekten oft mit FIELD benutzt. Das Dollarzeichen ($) als Suffix ist optional. MKD ist die Umkehrung von CVD.

Der von MKD zurückgegebene String stellt eine binäre Kopie der übergebenen Zahl dar. Siehe dazu die Beispiele zu CVD und zu MKL.

Unterschiede unter den FB-Dialektformen:

- In der Dialektform -lang qb ist das Suffix $ verbindlich.
- In den Dialektformen -lang fblite und -lang fb ist das Suffix optional.

Siehe auch:
CHR, MKSHORT, MKI, MKL, MKLONGINT, MKS, ASC, CVSHORT, CVI, CVL, CVLONGINT, CVS, CVD, Datentypen umwandeln

Letzte Bearbeitung des Eintrags am 27.12.12 um 14:42:50
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

MKDIR

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » M » **MKDIR**
Syntax: MKDIR (Ordnername)
Typ: Funktion
Kategorie: System

MKDIR erstellt einen Ordner mit dem angegebenen Namen.

- 'Ordnername' ist ein STRING, der den Namen des Ordners enthält, der erstellt werden soll. Er kann mit einer Pfadangabe versehen sein. Wird kein Arbeitspfad angegeben, so erstellt MKDIR den Ordner im aktuellen Arbeitsverzeichnis; siehe CURDIR.
- Der Rückgabewert ist entweder 0, wenn der Ordner erstellt werden konnte, oder -1, wenn ein Fehler aufgetreten ist. Dies kann z.B. bedeuten, dass ein Ordner mit angegebenen Namen schon existiert, dass 'Ordnername' unzulässige Zeichen enthält oder dass auf den angegebenen Pfad nicht zugegriffen werden kann (z. B. auf eine CD oder einen schreibgeschützten Ordner).

Der Rückgabewert kann verworfen werden; MKDIR wird dann wie eine Anweisung eingesetzt.

Beispiel:

```
MKDIR "data"
IF MKDIR("C:\BASIC\Testfolder") THEN
  PRINT "'Testfolder' konnte nicht erstellt werden!"
END IF
SLEEP
```

Das Beispiel erstellt, wenn möglich, den Ordner 'data' im aktuellen Arbeitsverzeichnis und den Ordner 'Testfolder' in C:\BASIC. Wenn 'Testfolder' nicht erstellt werden kann, wird eine Meldung ausgegeben.

Unterschiede zu QB:
In FreeBASIC kann MKDIR auch als Funktion eingesetzt werden.

Plattformbedingte Unterschiede:

- Unter Linux muss der Ordnername 'case sensitive' erfolgen. Windows und DOS sind 'case insensitive'.
- Das Trennzeichen für den Dateipfad ist unter Linux der vorwärtsgerichtete Slash /. Windows verwendet den Backslash \, erlaubt aber auch den Slash. DOS verwendet den Backslash.

Siehe auch:
RMDIR, CHDIR, CURDIR, SHELL, Betriebssystem-Anweisungen

Letzte Bearbeitung des Eintrags am 27.12.12 um 14:43:35
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

MKI

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » M » **MKI**
Syntax:

```
MKI[$] (INTEGER-Ausdruck)
MKI[$]<Bits> (INTEGER<Bits>-Ausdruck)
```

Typ: Funktion
Kategorie: Typumwandlung

MKI verwandelt eine INTEGER-Zahl in einen STRING. Der zurückgegebene String stellt eine binäre Kopie der übergebenen Zahl dar. Die Funktion wurde in älteren BASIC-Dialekten oft mit FIELD benutzt. MKI ist die Umkehrung von CVI.

- 'Bits' gibt an, wie groß der übergebene INTEGER-Ausdruck ist und muss sich folglich nach diesem richten. Entspricht 'Bits' dem Wert 16, so wird MKSHORT verwendet, bei 32 MKL und bei 64 MKLONGINT. Wird 'Bits' ausgelassen, so wird in Abhängigkeit von der verwendeten Plattform der Wert 32 (x86-Architektur) oder 64 (x64-Architektur) angenommen.
- 'INTEGER-Ausdruck' ist die Zahl, von der eine binäre Kopie als String erstellt werden soll.
- Der Rückgabewert ist, je nach Angabe von 'Bits' bzw. der verwendeten Plattform, ein String mit der Länge 2, 4 oder 8 Byte.

Das Dollarzeichen ($) als Suffix ist optional.

Beispiel:

```
DIM n AS  INTEGER
n = &h21504246
PRINT MKI(n)
PRINT MKI<32>(n)
PRINT CHR(&h46, &h42, &h50, &h21)
SLEEP
```

Ausgabe:

```
FBP!
FBP!
FBP!
```

Unterschiede zu QB:
QB unterstützt den Parameter 'Bits' nicht. INTEGER sind dort immer 16 Bit lang und MKI erzeugt einen 2-Byte-String.

Unterschiede zu früheren Versionen von FreeBASIC:
Der Parameter 'Bits' existiert seit FreeBASIC v0.90.

Unterschiede unter den FB-Dialektformen:

- In der Dialektform -lang qb ist das Suffix $ verbindlich. Ohne Angabe von 'Bits' erzeugt MKI ebenso wie QB einen 2-Byte-String.
- In den Dialektformen -lang fblite und -lang fb ist das Suffix optional.

Siehe auch:
CHR, MKSHORT, MKL, MKLONGINT, MKS, MKD, ASC, CVSHORT, CVI, CVL, CVLONGINT, CVS, CVD, Datentypen umwandeln

Letzte Bearbeitung des Eintrags am 04.07.13 um 17:34:18
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

MKL

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » M » **MKL**
Syntax: MKL[$](LONG-Ausdruck)
Typ: Funktion
Kategorie: Typumwandlung

MKL verwandelt eine LONG-Zahl in einen 4-Byte-STRING. Die Funktion wurde in älteren BASIC-Dialekten oft mit FIELD benutzt. Das Dollarzeichen ($) als Suffix ist optional. MKL ist die Umkehrung von CVL.

Der von MKL zurückgegebene String stellt eine binäre Kopie der übergebenen Zahl dar.

Beispiel:

```
DIM AS LONG n
n = &h21504246
PRINT MKL(n)
PRINT chr(&h46, &h42, &h50, &h21)
SLEEP
```

Ausgabe:

```
FBP!
FBP!
```

Unterschiede unter den FB-Dialektformen:

- In der Dialektform -lang qb ist das Suffix $ verbindlich.
- In den Dialektformen -lang fblite und -lang fb ist das Suffix optional.

Siehe auch:
CHR, MKSHORT, MKI, MKLONGINT, MKS, MKD, ASC, CVSHORT, CVI, CVL, CVLONGINT, CVS, CVD, Datentypen umwandeln

Letzte Bearbeitung des Eintrags am 27.12.12 um 14:43:55
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

MKLONGINT

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » M » **MKLONGINT**
Syntax: MKLONGINT[$](LONGINT-Ausdruck)
Typ: Funktion
Kategorie: Typumwandlung

MKLONGINT verwandelt eine LONGINT-Zahl in einen 8-Byte-STRING. Das Dollarzeichen ($) als Suffix ist optional. MKLONGINT ist die Umkehrung von CVLONGINT.

Der von MKLONGINT zurückgegebene String stellt eine binäre Kopie der übergebenen Zahl dar.

Beispiel:

```
DIM n AS LONGINT
n = &h21656D6F636C6557
PRINT MKLONGINT(n)
PRINT chr(&h57, &h65, &h6C, &h63, &h6F, &h6D, &h65, &h21)
SLEEP
```

Ausgabe:

```
Welcome!
Welcome!
```

Unterschiede zu QB: neu in FreeBASIC

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht MKLONGINT nicht zur Verfügung und kann nur über **__MKLONGINT** aufgerufen werden.

Siehe auch:
CHR, MKSHORT, MKI, MKL, MKS, MKD, ASC, CVSHORT, CVI, CVL, CVLONGINT, CVS, CVD, Datentypen umwandeln

Letzte Bearbeitung des Eintrags am 27.12.12 um 14:44:07
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

MKS

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » M » **MKS**
Syntax: MKS[$](SINGLE-Ausdruck)
Typ: Funktion
Kategorie: Typumwandlung

MKS verwandelt eine SINGLE-Zahl in einen 4-Byte-STRING. Die Funktion wurde in älteren BASIC-Dialekten oft mit FIELD benutzt. Das Dollarzeichen ($) als Suffix ist optional. MKS ist die Umkehrung von CVS.

Der von MKS zurückgegebene String stellt eine binäre Kopie der übergebenen Zahl dar. Siehe dazu die Beispiele zu CVS und zu MKL.

Unterschiede unter den FB-Dialektformen:

- In der Dialektform -lang qb ist das Suffix $ verbindlich.
- In den Dialektformen -lang fblite und -lang fb ist das Suffix optional.

Siehe auch:
CHR, MKSHORT, MKI, MKL, MKLONGINT, MKD, ASC, CVSHORT, CVI, CVL, CVLONGINT, CVS, CVD, Datentypen umwandeln

Letzte Bearbeitung des Eintrags am 27.12.12 um 14:44:22
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

MKSHORT

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » M » **MKSHORT**
Syntax: MKSHORT[$](SHORT-Ausdruck)
Typ: Funktion
Kategorie: Typumwandlung

MKSHORT verwandelt eine SHORT-Zahl in einen 2-Byte-STRING. Das Dollarzeichen ($) als Suffix ist optional. MKSHORT ist die Umkehrung von CVSHORT.

Der von MKL zurückgegebene String stellt eine binäre Kopie der übergebenen Zahl dar.

Beispiel:

```
Dim As Short a = &h6968
Print mkshort(a); chr(&h68, &h69)
Sleep
```

Ausgabe:

```
hihi
```

Unterschiede zu QB:
Die Funktion ist neu in FreeBASIC. In QB entspricht sie der Funktion MKI.

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht MKSHORT nicht zur Verfügung und kann nur über **__MKSHORT** aufgerufen werden.

Siehe auch:
CHR, MKI, MKL, MKLONGINT, MKS, MKD, ASC, CVSHORT, CVI, CVL, CVLONGINT, CVS, CVD, Datentypen umwandeln

Letzte Bearbeitung des Eintrags am 27.12.12 um 14:44:34
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

MOD

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » M » **MOD**
Syntax A: Ergebnis = Zahl1 MOD Zahl2
Syntax B: Ergebnis MOD= Zahl
Typ: Operator
Kategorie: Operatoren

MOD gibt den Rest der Division 'Zahl1' / 'Zahl2' zurück (Modulo).

- 'Zahl1', 'Zahl2' und 'Zahl' sind INTEGER-Werte. Wird MOD mit Gleitkommazahlen (SINGLE oder DOUBLE) verwendet, so werden diese zuerst mithilfe von CINT mathematisch zu INTEGER-Werten gerundet.
- Beim kombinierten MOD (Syntax B) wird in 'Ergebnis' der Rest der Division aus 'Ergebnis' und 'Zahl' zurückgegeben; es ist die Kurzform für Ergebnis = Ergebnis MOD Zahl

MOD kann mithilfe von OPERATOR überladen werden.

Beispiel:

```
DIM n AS INTEGER
PRINT 47 MOD 7
PRINT 5.6 MOD 2.1
PRINT 5.1 MOD 2.8
n = 11
n MOD= 3
PRINT n
```

Ausgabe:

```
5
0
2
2
```

47 geteilt durch 7 ergibt einen Rest von 5.
5.6 wird gerundet auf 6, während 2.1 auf 2 gerundet wird. Dadurch wird die Aufgabe zu 6 MOD 2. Da sich 6 ohne Rest durch 2 teilen lässt, ist das Ergebnis 0.
5.1 wird gerundet auf 5, während 2.8 auf 3 gerundet wird. Dadurch wird die Aufgabe zu 5 MOD 3. Beim Teilen von 5 durch 3 bleibt ein Rest von 2.

Siehe auch:
SHL, SHR, Backslash, mathematische Funktionen

Letzte Bearbeitung des Eintrags am 16.06.13 um 21:11:23
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

MONTH

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » M » **MONTH**
Syntax: MONTH (Serial)
Typ: Funktion
Kategorie: Datum und Zeit

Anmerkung:
Um diese Funktion nutzen zu können, muss die Datei datetime.bi in Ihren Quellcode eingebunden werden, z. B. mit INCLUDE. Alternativ können Sie auch die Datei vbcompat.bi einbinden, da dadurch auch automatisch datetime.bi eingebunden wird.

MONTH extrahiert den Monat einer Serial Number.

- 'Serial' ist ein DOUBLE-Wert, der als Serial Number behandelt wird.
- Der Rückgabewert ist der Monat des Jahres, der in der Serial Number gespeichert ist.

Beispiel: den heutigen Monat des Jahres aus NOW extrahieren:

```
"hlstring">"vbcompat.bi"
DIM monat AS INTEGER
monat = MONTH(NOW)
```

Unterschiede zu QB: existiert nur in QBX PDS und in VBDOS

Unterschiede zu früheren Versionen von FreeBASIC: existiert seit FreeBASIC v0.15

Siehe auch:
NOW, DATESERIAL, DATEVALUE, YEAR, DAY, WEEKDAY, HOUR, MINUTE, SECOND, MONTHNAME, WEEKDAYNAME, DATEDIFF, DATEPART, DATEADD, FORMAT, ISDATE, Serial Numbers, Datum und Zeit

Letzte Bearbeitung des Eintrags am 27.12.12 um 14:45:25
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

MONTHNAME

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » M » **MONTHNAME**
Syntax: MONTHNAME (Monat [, Kurzform])
Typ: Funktion
Kategorie: Datum und Zeit

Anmerkung:
Um diese Funktion nutzen zu können, muss die Datei datetime.bi in Ihren Quellcode eingebunden werden, z. B. mit INCLUDE. Alternativ können Sie auch die Datei vbcompat.bi einbinden, da dadurch auch automatisch datetime.bi eingebunden wird.

MONTHNAME gibt den Namen eines Monats zurück.

- 'Monat' ist die Nummer des Monats, also 1 für Januar, 2 für Februar und so weiter.
- Der Rückgabewert ist ein String, der den Namen des Monats enthält.
- 'Kurzform' ist ein Wert, der entweder gleich oder ungleich null ist. Wenn er ungleich null ist, wird ein abgekürzter Monatsname ausgegeben, der aus drei Zeichen besteht, ansonsten wird der volle Name ausgegeben.

Beispiel: den Namen des aktuellen Monats ausgeben:

```
"hlstring">"vbcompat.bi"
PRINT "Wir befinden uns im Monat " & MONTHNAME(MONTH(NOW)) & "."
SLEEP
```

Unterschiede zu QB: existiert nur in QBX PDS und in VBDOS

Unterschiede zu früheren Versionen von FreeBASIC: existiert seit FreeBASIC v0.15

Siehe auch:
NOW, DATESERIAL, DATEVALUE, YEAR, MONTH, DAY, WEEKDAY, WEEKDAYNAME, FORMAT, ISDATE, Serial Numbers, Datum und Zeit

Letzte Bearbeitung des Eintrags am 27.12.12 um 14:45:46
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

MULTIKEY

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » M » **MULTIKEY**
Syntax: MULTIKEY (Scancode)
Typ: Funktion
Kategorie: Benutzereingabe

MULTIKEY reagiert auf (multiple) Tastendrücke. Diese Funktion zeigt an, ob die angegebene Taste gerade gedrückt wird.

- 'Scancode' ist der DOS-Hardware Scancode der Taste, auf die die Anfrage stattfinden soll.
- Der Rückgabewert (vom Typ INTEGER) ist -1, wenn die angegebene Taste gerade gedrückt wird. Ansonsten ist das Ergebnis 0.
- Der Tastaturpuffer wird nicht geleert, wenn MULTIKEY verwendet wird; jede gedrückte Taste wird gespeichert, bis sie z. B. durch INKEY abgefragt wird. Damit kein Überlauf im Tastaturpuffer stattfindet, müssen Sie diesen selbständig leeren; es empfiehlt sich daher, MULTIKEY in Kombination mit INKEY zu verwenden.

Das Arbeiten mit MULTIKEY ist komfortabler als der alleinige Einsatz von INKEY; da das gleichzeitige Drücken zweier Tasten mit einer einfachen AND-Verknüpfung geschehen kann:

```
IF MULTIKEY(&h01) AND _
(MULTIKEY(&h2A) OR MULTIKEY(&h36)) THEN ...
```

Der Code hinter THEN würde ausgeführt, wenn Escape (Code &h01) und gleichzeitig (AND) die linke Shifttaste (Code &h2A) oder (OR) die rechte Shifttaste (Code &h36) gedrückt sind. Mit INKEY dagegen wäre eine Unterscheidung zwischen linker und rechter Shifttaste gar nicht möglich. Da MULTIKEY den Tastaturpuffer nicht leert, bietet es sich an, in Schleifen die Bedingung zum Verlassen der Schleife mit INKEY zu prüfen, während alle anderen Tastatureingaben bequem mit MULTIKEY ausgewertet werden. Ist dies nicht möglich, kann der Tastaturpuffer mit diesem Code leer gehalten werden:

```
WHILE LEN(INKEY) : WEND
```

Beispiel:

```
DIM AS INTEGER work_page, x, y
 'Grafikfenster 640x480x8 mit zwei Seiten initialisieren
SCREENRES 640, 480, ,2
COLOR 2, 15
work_page = 0
x = 320
y = 240
DO
    SLEEP 1
    ' Eine Seite beschreiben, während die andere angezeigt wird
    SCREENSET work_page, work_page XOR 1
     ' Überprüfe die Pfeiltasten und aktualisiere die Koordinaten
    ' x, y entsprechend.
    IF MULTIKEY(&h4D) AND x < 639 THEN x += 1
    IF MULTIKEY(&h4B) AND x >   0 THEN x -= 1
    IF MULTIKEY(&h50) AND y < 479 THEN y += 1
    IF MULTIKEY(&h48) AND y >   0 THEN y -= 1
```

```
    CLS
    CIRCLE(x, y), 30, , , , ,F
    ' Seite wechseln
    work_page = work_page XOR 1
LOOP UNTIL INKEY = CHR(27) ' Bis ESC gedrückt wird weitermachen
' Setze aktive und sichtbare Bildschirmseite wieder auf 0
SCREENSET
PRINT "Beliebige Taste zum Beenden druecken."
SLEEP
```

Anmerkung: Sämtliche Keyboard-Scancodes sind in der Datei fbgfx.bi als Symbole mit verständlichen Namen definiert. Indem Sie diese Datei mit INCLUDE einbinden, ersparen Sie sich beim Programmieren das Nachsehen, welcher Code für welche Taste steht.

Beispiel:

```
"hlstring">"fbgfx.bi"
IF MULTIKEY(FB.SC_F1) THEN PRINT "F1 wurde gedrückt."
IF MULTIKEY(FB.SC_LEFT) THEN PRINT "Linke Pfeiltaste wurde gedrückt."
IF MULTIKEY(FB.SC_RIGHT) THEN PRINT "Rechte Peiltaste wurde gedrückt."
'...
```

Sollte die Datei *fbgfx.bi* sich nicht in Ihrem Include-Verzeichnis befinden, können Sie den Code verwenden, der unter DOS Keyboard Scancodes der Referenz hinterlegt ist.

Unterschiede zu QB: neu in FreeBASIC

Plattformbedingte Unterschiede:
Die Unterstützung von MULTIKEY im Konsolenfenster ist möglicherweise nicht unter allen Plattformen gewährleistet.

Unterschiede zu früheren Versionen von FreeBASIC:

- Seit Version v0.15 können die erweiterten Scancodes auch in der DOS-Version abgefragt werden.
- Seit Version v0.14 kann MULTIKEY auch im Konsolen-Modus eingesetzt werden.

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht MUKTIKEY nicht zur Verfügung und kann nur über **__MULTIKEY** aufgerufen werden.

Siehe auch:
INKEY, DOS Keyboard Scancodes, Benutzereingaben

Letzte Bearbeitung des Eintrags am 19.08.12 um 23:47:46
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

MUTEXCREATE

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » M » **MUTEXCREATE**
Syntax: MUTEXCREATE
Typ: Funktion
Kategorie: Multithreading

MUTEXCREATE erstellt einen Mutex und gibt einen Handle darauf zurück.

Der Rückgabewert ist ein ANY PTR, der einen Mutex eindeutig identifiziert. Über diesen wird später auf den Mutex zugegriffen. In früheren Versionen von FreeBASIC (vor v0.17) wurde ein INTEGER zurückgegeben.

Mutex (englisch: Mutual Exclusion, deutsch: gegenseitiger Ausschluss) bezeichnet Verfahren, mit denen verhindert wird, dass nebenläufige Prozesse gleichzeitig auf Daten zugreifen und so unter Umständen inkonsistente Zustände herbeiführen (Auszug aus Wikipedia; siehe http://de.wikipedia.org/wiki/Mutex). Wenn zwei Threads (siehe THREADCREATE) gleichzeitig versuchen, auf eine globale Variable oder eine Datei zuzugreifen, kann dies zu Problemen führen. Durch Mutexe kann dies verhindert werden, indem ein Thread dazu gezwungen wird, darauf zu warten, dass ein anderer Thread seinen Zugriff beendet hat. Ein Mutex kann entweder gesperrt oder entsperrt sein. Wenn ein Thread versucht, auf den Mutex zuzugreifen, muss er warten, bis dieser entsperrt wurde. Dazu verwendet man MUTEXLOCK und MUTEXUNLOCK. Auch das Sperren eines Mutexes gilt als Zugriff; bei der Durchführung dieses Vorgangs wird also auf die Entsperrung gewartet.

Wenn der Mutex nicht mehr benötigt wird, sollte er mit MUTEXDESTROY zerstört werden.

Beispiel:

```
Dim Shared produced As Any Ptr
Dim Shared consumed As Any Ptr
Dim consumer_id As Any Ptr
Dim producer_id As Any Ptr


Sub consumer ( ByVal param As Any Ptr )
  For i As Integer = 0 To 9
    MutexLock produced
    Print ", consumer gets:", i
    MutexUnlock consumed
  Next i
End Sub

Sub producer ( ByVal param As Any Ptr )
  For i As Integer = 0 To 9
    Print "Producer puts:", i;
    MutexUnlock produced
    MutexLock consumed
  Next i
End Sub


produced = MutexCreate
consumed = MutexCreate
If (produced = 0) Or (consumed = 0) Then
  Print "Error creating mutexes! Exiting..."
```

```
  Sleep
  End
End If

MutexLock produced
MutexLock consumed

consumer_id = ThreadCreate(@consumer)
producer_id = ThreadCreate(@producer)
If (producer_id = 0) Or (consumer_id = 0) Then
  Print "Error creating threads! Exiting..."
  Sleep
  End
End If

ThreadWait consumer_id
ThreadWait producer_id

MutexDestroy consumed
MutexDestroy produced
Sleep
```

Zunächst sind beide Mutexe gesperrt. Wenn der Prozessor zuerst versucht, den Thread 'consumer' auszuführen, stößt er auf das Hindernis, dass 'produced' bereits gesperrt ist. Es kann erst dann gesperrt werden, wenn es entsperrt wurde. Der Prozessor wartet also mit der Ausführung des Threads 'consumer' und setzt mit der Ausführung der anderen Threads in der Warteliste fort (hier: Thread 'producer'). 'producer' greift zuerst auf keinen Mutex zu und wird sofort durchgeführt. Dann wird 'produced' entsperrt - 'consumer' kann also ausgeführt werden. Wenn der Prozessor dennoch zuerst versucht, die Schleife in 'producer' einmal zu durchlaufen, stößt er auf das Hindernis, dass 'consumed' bereits gesperrt ist. Er muss also darauf warten, dass dieser Mutex von 'consumer' entsperrt wird.

Unterschiede zu QB: neu in FreeBASIC

Plattformbedingte Unterschiede:
In der DOS-Version von FreeBASIC stehen Mutexe nicht zur Verfügung, da Threads nicht unterstützt werden.

Unterschiede zu früheren Versionen von FreeBASIC:

- Seit FreeBASIC v0.17 gibt MUTEXCREATE einen ANY PTR zurück.
- MUTEXCREATE existiert seit FreeBASIC v0.13

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht MUTEXCREATE nicht zur Verfügung.

Siehe auch:
Multithreading

Letzte Bearbeitung des Eintrags am 08.01.13 um 01:01:19
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

MUTEXDESTROY

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » M » **MUTEXDESTROY**
Syntax: MUTEXDESTROY handle
Typ: Anweisung
Kategorie: Multithreading

MUTEXDESTROY löscht einen Mutex. Dieser kann sowohl gesperrt als auch entsperrt sein; es wird nicht auf seine Entsperrung gewartet.

'handle' ist ein ANY PTR. Er stellt den Handle zu einem mit MUTEXCREATE erstellten Mutex dar. In früheren Versionen von FreeBASIC (vor v0.17) wurde ein INTEGER verwendet.

MUTEXDESTROY sollte immer aufgerufen werden, nachdem alle Threads, die auf den Mutex zugreifen, beendet wurden.

Beispiel: siehe MUTEXCREATE.

Unterschiede zu QB: neu in FreeBASIC

Plattformbedingte Unterschiede:
In der DOS-Version von FreeBASIC stehen Mutexe nicht zur Verfügung, da Threads nicht unterstützt werden.

Unterschiede zu früheren Versionen von FreeBASIC:

- Seit FreeBASIC v0.17 verlangt MUTEXDESTROY einen ANY PTR als Parameter.
- MUTEXDESTROY existiert seit FreeBASIC v0.13

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht MUTEXDESTROY nicht zur Verfügung.

Siehe auch:
Multithreading

Letzte Bearbeitung des Eintrags am 27.12.12 um 14:46:28
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

MUTEXLOCK

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » M » **MUTEXLOCK**
Syntax: MUTEXLOCK handle
Typ: Anweisung
Kategorie: Multithreading

MUTEXLOCK sperrt den Zugriff auf einen Mutex.

'handle' ist ein ANY PTR. Er stellt den Handle zu einem mit MUTEXCREATE erstellten Mutex dar. In früheren Versionen von FreeBASIC (vor v0.17) wurde ein INTEGER verwendet.
Ein Mutex kann nicht zweimal gesperrt werden. Ist ein Mutex bereits gesperrt, wartet der Prozessor mit der Ausführung des Threads, bis der Mutex von einem anderen Thread entsperrt wurde.

Beispiel: siehe MUTEXCREATE.

Unterschiede zu QB: neu in FreeBASIC

Plattformbedingte Unterschiede:
In der DOS-Version von FreeBASIC stehen Mutexe nicht zur Verfügung, da Threads nicht unterstützt werden.

Unterschiede zu früheren Versionen von FreeBASIC:

- Seit FreeBASIC v0.17 verlangt MUTEXLOCK einen ANY PTR als Parameter.
- MUTEXLOCK existiert seit FreeBASIC v0.13

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht MUTEXLOCK nicht zur Verfügung.

Siehe auch:
Multithreading

Letzte Bearbeitung des Eintrags am 27.12.12 um 14:46:48
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

MUTEXUNLOCK

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » M » **MUTEXUNLOCK**
Syntax: MUTEXUNLOCK handle
Typ: Funktion
Kategorie: Multithreading

MUTEXUNLOCK entsperrt einen Mutex. Der Mutex wird in jedem Fall entsperrt, egal ob er zuvor gesperrt oder entsperrt war.

'handle' ist ein ANY PTR. Er stellt den Handle zu einem mit MUTEXCREATE erstellten Mutex dar. In früheren Versionen von FreeBASIC (vor v0.17) wurde ein INTEGER verwendet.

Beispiel: siehe MUTEXCREATE.

Unterschiede zu QB: neu in FreeBASIC

Plattformbedingte Unterschiede:
In der DOS-Version von FreeBASIC stehen Mutexe nicht zur Verfügung, da Threads nicht unterstützt werden.

Unterschiede zu früheren Versionen von FreeBASIC:

- Seit FreeBASIC v0.17 verlangt MUTEXUNLOCK einen ANY PTR als Parameter.
- MUTEXUNLOCK existiert seit FreeBASIC v0.13.

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht MUTEXUNLOCK nicht zur Verfügung.

Siehe auch:
Multithreading

Letzte Bearbeitung des Eintrags am 27.12.12 um 14:47:15
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

NAKED

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » N » **NAKED**
Syntax:

```
Sub SubName Naked &"hlzeichen">] ([Parameterliste])
    Assemblerbefehle
End Sub

Function FunctionName Naked &"hlzeichen">] ([Parameterliste]) As Typ
    Assemblerbefehle
End Function
```

Typ: Schlüsselwort
Kategorie: Deklaration

NAKED erstellt Funktionen ohne Handlingcode.

- 'SubName' bzw. 'FunctionName' ist der Bezeichner, unter dem die SUB/FUNCTION aufgerufen wird.
- 'Konvention' enthält die Aufrufkonventionen wie CDECL, PASCAL oder STDCALL.
- 'Parameterliste' gibt die Parameter an, die an das Unterprogramm übergeben werden sollen.
- 'Typ' ist der Datentyp des übergebenen Parameters; siehe Datentypen. Auch UDTs können verwendet werden.

NAKED ermöglicht dem Programmierer, ein Unterprogramm ohne Handlingcode zu erstellen. Das ist vor allem dann nützlich, wenn schnelle Funktionen in ASM benötigt werden.
NAKED ist mit -gen gcc nicht kompatibel, da GCC solche Funktionen nicht kennt, und deswegen nicht verwenden kann.
Den Aufrufkonventionen des C-ABI entsprechend, dürfen die Register EAX, ECX und EDX beliebig verändert werden - alle anderen Register dürfen entweder nicht verändert werden oder müssen gesichert und am Ende des NAKED Unterprogramms auf ihren ursprünglichen Wert zurückgesetzt werden.

Beispiel:

```
' Naked Cdecl Function
Function addieren Naked Cdecl (ByVal a As Integer, ByVal b As Integer) As
Integer
    Asm
        mov eax,&"hlzeichen">+4] ' a
        add eax,&"hlzeichen">+8] ' + b
        ret                       ' gibt das Ergebnis in eax zurück
    End Asm
End Function

' Naked StdCall Function
Function addieren2 Naked (ByVal a As Integer, ByVal b As Integer) As
Integer
    Asm
        mov eax,&"hlzeichen">+4]  ' a
        add eax,&"hlzeichen">+8]  ' + b
        ret 8                        ' 8 Byte auf dem Stack freigeben,
Ergebnis in eax
    End Asm
End Function
```

```
Print addieren(3, 5)
Print addieren2(7, 9)
sleep
```

Plattformbedingte Unterschiede:
Die Standardaufrufkonvention kann von der verwendeten Plattform abhängen. Zusätzlich hierzu verhält sich STDCALL nicht auf allen Plattformen gleich - unter Linux verhält es sich wie CDECL. Dadurch könnte eine Prüfung der vordefinierten Symbole (wie __FB_WIN32__) nötig sein, um den Code speziell auf die verwendete Plattform auszurichten.

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC:

- Vor FreeBASIC v0.24 war NAKED nicht kompatibel mit der Compiler-Option -exx. Die Funktionsaufrufe dieser Option benötigten den fehlenden Handlingcode.
- NAKED existiert seit FreeBASIC v0.21.

Siehe auch:
ASM, FUNCTION, SUB, CDECL, PASCAL, STDCALL, Prozeduren

Letzte Bearbeitung des Eintrags am 27.12.12 um 14:48:15
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

NAME

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » N » **NAME**
Syntax: NAME (NameAlt, NameNeu)
Typ: Funktion
Kategorie: System

NAME benennt eine Datei um.

- 'NameAlt' ist ein STRING, der den Namen der Datei enthält, die umbenannt werden soll.
- 'NameNeu' ist ein STRING, der den neuen Dateinamen enthält, den die Datei erhalten soll.
- Der Rückgabewert ist 0, wenn die Datei erfolgreich umbenannt werden konnte, oder -1, wenn ein Fehler aufgetreten ist (z. B. weil die Datei 'NameAlt' nicht existiert oder weil bereits eine Datei namens 'NameNeu' existiert).

FreeBASIC geht davon aus, dass sich die Datei 'NameAlt' im aktuellen Arbeitsverzeichnis befindet (siehe CURDIR). Ist dies nicht der Fall, muss 'NameAlt' eine Pfadangabe enthalten. Diese darf absolut oder relativ sein.
Enthält 'NameNeu' eine andere Pfadangabe als 'NameAlt', so wird die Datei in den bei 'NameNeu' angegeben Ordner verschoben. Das bedeutet auch, dass bei einer Pfadangabe in 'NameAlt' auch 'NameNeu' dieselbe Pfadangabe enthalten muss, wenn die Datei nicht verschoben werden soll.

Beispiel:

```
' Benenne die Datei "dsc001.jpg" zu "landscape.jpg" um:
NAME "dsc001.jpg", "landscape.jpg"

' Verschiebe die Datei Dialog.bas von C:\BASIC\Projekt\ nach
C:\BASIC\Beispiele\
' ohne sie umzubenennen
IF NAME ("C:\BASIC\Projekt\Dialog.bas", "C:\BASIC\Beispiele\Dialog.bas")
<> 0 THEN
  PRINT "Fehler: Die Datei konnte nicht verschoben werden."
END IF
SLEEP
```

Unterschiede zu QB:

- In QB lautet die Syntax NAME NameAlt AS NameNeu.
- In FreeBASIC kann NAME auch als Funktion verwendet werden.

Da FreeBASIC ein 32bit-Compiler ist, werden auch lange Dateinamen bis zu 255 Zeichen unterstützt.

Plattformbedingte Unterschiede:

- Unter Linux muss der Dateiname 'case sensitive' erfolgen. Windows und DOS sind 'case insensitive'.
- Das Trennzeichen für den Dateipfad ist unter Linux der vorwärtsgerichtete Slash /. Windows verwendet den Backslash \, erlaubt aber auch den Slash. DOS verwendet den Backslash.

Siehe auch:
KILL, CURDIR, Betriebssystem-Anweisungen

Letzte Bearbeitung des Eintrags am 27.12.12 um 14:48:32
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

NAMESPACE

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » N » **NAMESPACE**
Syntax:

```
NAMESPACE Bereichsname
  ' Programmcode
END NAMESPACE
```

Typ: Anweisung
Kategorie: Deklaration

NAMESPACE definiert einen Codeteil als Namespace. Innerhalb eines Namespaces dürfen Symbole benutzt werden, die in einem anderem Namespace bereits benutzt wurden. Namespaces werden üblicherweise in Libraries verwendet, um keine Symbolbezeichner benutzen zu müssen, die der Anwender der Lib eventuell ebenfalls benutzen möchte. Wird z.B. in der Lib ein UDT 'foobar' verwendet, kann der User einen eigenen UDT 'foobar' definieren, wenn der UDT in der Lib innerhalb eines Namespaces definiert wurde.

Außerhalb eines Namespaces kann über die Syntax 'Bereichsname.Bezeichner' auf ein Element des Namespaces zugegriffen werden, oder indem der Namespace via USING in den *'Globalen Namespace'* eingebunden wurde. Über WITH kann (noch) nicht auf die Elemente eines Namespaces zugegriffen werden. Innerhalb eines Namespaces dürfen ausschließlich Deklarationen stehen; sogenannte *'ausführbare'* Anweisungen sind unzulässig.

FreeBASIC-Namespaces sind kompatibel zu GCC C++.

Folgende Anweisungen sind innerhalb eines Namespaces erlaubt:

- DIM
- STATIC
- COMMON
- DECLARE
- TYPE
- UNION
- ENUM
- CONST
- VAR

Beispiel:

```
Namespace Forms
  Type Dot ' Ein 2D-Punkt
    x As Integer
    y As Integer
  End Type

  /' Da sich die folgenden Zeilen
  innerhalb des Namespaces befinden,
  beziehen sich alle Aufrufe von 'Dot'
  auf Forms.Dot. '/
  Sub AdjustDot( ByRef pt As Dot, ByVal newx As Integer, ByVal newy As
Integer )
    pt.x = newx
```

```
    pt.y = newy
  End Sub
End Namespace

Type Dot ' Ein 3D-Punkt
  x As Integer
  y As Integer
  z As Integer
End Type

Sub AdjustDot( ByRef PT As Dot, ByVal newx As Integer, ByVal newy As
Integer, ByVal newz As Integer )
  pt.x = newx
  pt.y = newy
  pt.z = newz
End Sub

Dim d1 As Dot
AdjustDot( d1, 1, 1, 1 )

Dim d2 As Forms.Dot
Forms.AdjustDot( d2, 1, 1 )

Sleep
```

Namespace kann man auch beliebig verschachteln, was durchaus nützlich sein kann:

```
Namespace erste
  Dim As Integer variable = 1
End Namespace

Namespace zweite
  Dim As Integer variable = 2
End Namespace

Namespace dritte
  Dim As Integer variable = 3

  Namespace verschachtelte
    Dim As Integer variable = 4
  End Namespace
End Namespace

Print erste.variable
Print zweite.variable
Print dritte.variable
Print dritte.verschachtelte.variable

Sleep
```

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC: existiert seit FreeBASIC v0.16

Siehe auch:
USING (Namespace), EXTERN...END EXTERN, Datentypen und Deklarationen

Letzte Bearbeitung des Eintrags am 27.12.12 um 14:49:13
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

NEW

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » N » **NEW**
Syntax A: NEW [(Adresse)] Datentyp [(Parameterliste | ANY)]
Syntax B: NEW [(Adresse)] Datentyp [Anzahl] [{ ANY }]
Typ: Operator
Kategorie: Operatoren

NEW weist dynamisch Speicher zu und erzeugt Daten/Objekte des angegebenen Typs.

- 'Parameterliste' gibt den/die Anfangswert(e) der Variablen an. Sie kann auch ausgelassen werden.
- 'Datentyp' ist der Name des zu erzeugenden Datentyps. Es kann sich um einen Standarddatentyp oder ein UDT handeln. Auch Pointer sind möglich. STRINGs sind allerdings nicht erlaubt.
- 'Anzahl' gibt die Anzahl der zu erzeugenden Elemente an.
- 'Adresse' ist ein bereits alloziierter Pointer, an den das Objekt geschrieben werden soll.
- Wird der Anfangswert 'ANY' benutzt, wie z. B. in NEW Datentyp (ANY) oder NEW Datentyp [Anzahl] {ANY}, wird zwar Speicher für 'Datentyp' reserviert, die Daten werden jedoch nicht initialisiert. Das gilt nur bei Datentypen, die keine Konstruktoren besitzen.
- Der Rückgabewert ist ein Pointer des Typs 'Datentyp' auf den neu erzeugten Datenbereich.

NEW weist dynamisch Speicher zu und konstruiert den angegebenen Datentyp. Bei einfachen Typen wie INTEGERs kann ein Anfangswert angegeben werden. Bei Typen ohne Konstruktoren können Anfangswerte für jedes Feld angegeben werden. Bei Typen, die Konstruktoren besitzen, können diese durch NEW auch aufgerufen werden. Wird 'Parameterliste' nicht angegeben, werden die Standardwerte für diese Typen benutzt.

NEW ['Anzahl'] ist die Array-Version von NEW und weist genügend Speicher für die angegebene Anzahl der Objekte zu. Der Standard-Konstruktor des Typs wird dann aufgerufen, um die Anfangswerte jedes Objekts zu setzen.

Objekte, die mit NEW erzeugt wurden, müssen mit DELETE freigegeben werden. Speicher, der mittels NEW [Anzahl] reserviert wurde, muss jedoch mit DELETE [ ], der Array-Version von DELETE, freigegeben werden. Die verschiedenen Versionen der Operatoren dürfen nicht vermischt werden und 'passen' auch nicht zueinander.

Wird NEW mit dem Parameter 'Adresse' verwendet, so reserviert NEW keinen Speicher, sondern erzeugt das Objekt an der vorgegebenen Adresse. Der Speicher muss also schon vorher reserviert werden. Da der Speicher nicht durch NEW reserviert wird, sollte er auch nicht mit DELETE freigegeben werden. Stattdessen sollte der Destruktor des Objekts explizit aufgerufen und der Speicher hinterher mit der zur Reservierung entsprechenden Freigabemethode freigegeben werden.

NEW kann mithilfe von OPERATOR überladen werden.

Beispiel 1:

```
Type Rational
    As Integer Zaehler, Nenner
End Type

Scope

    ' Erzeugen und Initialisieren eines Rationalbruchs
    ' und Speichern seiner Adresse
    Dim p As Rational Ptr = New Rational(3, 4)
```

```
    Print p->Zaehler & "/" & p->Nenner

    ' Löschen des Bruchs und Rückgabe des Speichers
    ' an das System
    Delete p

End Scope

Scope

    ' Speicherreservierung für 100 Integers, Speichern
    ' der Adresse des ersten
    Dim p As Integer Ptr = New Integer&"hlzahl">100]

    ' Zuweisen von Werten an die Integers im Array
    For i As Integer = 0 To 99
        p&"hlzeichen">] = i
    Next

    ' Freigeben des gesamten Integer-Arrays
    Delete&"hlzeichen">] p

End Scope

Sleep
```

Beispiel 2:

```
Type Rational
    As Integer zaehler, nenner
End Type

' Pointer anlegen und Größe des Types reservieren
Dim As Any Ptr ap = Callocate(SizeOf(Rational))

' Pointer des Types anlegen und neues Objekt an die Stelle
' des bereits reservierten Speichers schreiben
Dim As Rational Ptr r = New (ap) Rational( 3, 4 )

' Beide Pointer haben nun dieselbe Adresse im Speicher
Print ap, r
Print r->zaehler & "/" & r->nenner

' Statt mit DELETE sollte das Objekt durch explizites
' Aufrufen des Destruktors zerstört werden, da DELETE
' den Speicher danach freigibt, was zu Speicherproblemen
' und Programmabstürzen führen kann
r->Destructor( )

' Den Speicher letztlich freigeben
DeAllocate ap

' Auf Tastendruck vor Programmende warten
Sleep
```

Unterschiede zu QB: neu in FreeBASIC

Unterschiede unter den FB-Dialektformen: nur in der Dialektform -lang fb verfügbar

Siehe auch:
DELETE, Speicher

Letzte Bearbeitung des Eintrags am 27.12.12 um 14:49:45
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

NEXT

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » N » **NEXT**
Syntax A: NEXT [Variable [, Variable ...]]
Syntax B: RESUME [NEXT]
Typ: Schlüsselwort
Kategorie: Programmablauf

Syntax A: FOR-Schleife
NEXT ist der schließende Bestandteil der Zählschleife FOR ... NEXT.

Auf NEXT folgt die Zählvariable, die am Anfang des Schleifenblocks in der FOR-Zeile verwendet wurde. Die Angabe der Zählvariablen hinter NEXT kann auch ausgelassen werden.

```
DIM i AS INTEGER
FOR i = 1 TO 5
  PRINT "Hallo Welt!"
NEXT i
SLEEP : END
```

Es ist auch möglich, mehrere Zählvariablen anzugeben und damit mehrere FOR-Schleifen gleichzeitig zu schließen.
Beispiel:

```
FOR a as integer = 1 TO 5 : FOR b as integer = 1 TO 5
  PRINT a & " * " & b & " = " & a*b
NEXT b, a
SLEEP
```

Unterschiede zu früheren Versionen von FreeBASIC:
War es vor FreeBASIC v0.18.3 noch möglich,

```
For i = 0 To 3
  Print "i = " & i
Next irgendwasesmussnurzusammengeschriebensein
```

zu schreiben, also dem NEXT eine beliebige Zeichenfolge folgen zu lassen, ist seit v0.18.3 die Zählvariable (Laufindex) anzufügen, in diesem Fall 'i', oder nur ein NEXT ohne Variable. Dabei wird dann automatisch die Variable des dazugehörigen FOR-Befehls verwendet. Bei einer anderen beliebigen Zeichenfolge gibt der Compiler eine Fehlermeldung aus.

Unterschiede zu QB:
In FreeBASIC werden auch mehrere Zähler unterstützt.

Syntax B: Fehlerbehandlung
NEXT ist auch Teil der RESUME-Anweisung. RESUME &"reflinkicon" href="temp0573.html">-lang qb und -lang fblite zur Verfügung.

Letzte Bearbeitung des Eintrags am 27.12.12 um 14:50:11
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

NOGOSUB (Schlüsselwort)

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » N » **NOGOSUB (Schlüsselwort)**
Syntax: OPTION NOGOSUB
Typ: Schlüsselwort
Kategorie: Programmoptionen

OPTION NOGOSUB deaktiviert die Unterstützung von GOSUB und ON ... GOSUB. Die Option kann **nur bis FreeBASIC v0.16** eingesetzt werden, oder in entsprechend höheren Versionen, die mit der Kommandozeilenoption -lang deprecated compiliert wurden! Wird mit FreeBASIC v0.17 unter der Option -lang fb compiliert, so ist OPTION NOGOSUB nicht mehr zulässig!

Da RETURN sowohl eine Rückkehr von GOSUB als auch von einer Prozedur bedeuten kann, können OPTION GOSUB und OPTION NOGOSUB verwendet werden, um GOSUB zu aktivieren bzw. deaktivieren. Wenn GOSUB deaktiviert wurde, wird RETURN nur als Rückkehr aus einer Prozedur bzw. Funktion aufgefasst. Ansonsten wird RETURN nur als Rückkehr von GOSUB aufgefasst.

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC:
Die Option ist nur bis FreeBASIC v0.16 erlaubt. Seit FreeBASIC v0.17 ist diese Option nur noch zulässig, wenn mit der Kommandozeilenoption -lang deprecated compiliert wurde.

Siehe auch:
NOGOSUB (Schlüsselwort), GOSUB, RETURN, __FB_OPTION_GOSUB__, OPTION, Verschiedenes

Letzte Bearbeitung des Eintrags am 19.01.13 um 16:44:25
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

NOKEYWORD

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » N » **NOKEYWORD**
Syntax: OPTION NOKEYWORD keyword
Typ: Schlüsselwort
Kategorie: Programmoptionen

Mit OPTION NOKEYWORD kann ein reserviertes Schlüsselwort freigeben und anschließend als Variable, Objekt, Prozedur oder anderes Symbol verwendet werden. 'keyword' ist das Schlüsselwort, das freigegeben wird.

Die Option kann **nur bis FreeBASIC v0.16** eingesetzt werden, oder in entsprechend höheren Versionen, die mit der Kommandozeilenoption -lang deprecated compiliert wurden! Seit FreeBASIC v0.17 wird dieser Befehl durch den Metabefehl UNDEF ersetzt.

Beispiel:

```
"hlstring">"fblite"

Option NoKeyword Int ' gibt das Schlüsselwort INT frei
Dim Int As Integer ' INT wird ab sofort als Variable verwendet
```

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC:
Die Option ist nur bis FreeBASIC v0.16 erlaubt. Seit FreeBASIC v0.17 ist diese Option nur noch zulässig, wenn mit der Kommandozeilenoption -lang deprecated compiliert wurde. Ansonsten kann ab FreeBASIC v0.17 der Metabefehl UNDEF verwendet werden.

Siehe auch:
UNDEF (Metabefehl), OPTION, Verschiedenes

Letzte Bearbeitung des Eintrags am 19.01.13 um 16:45:09
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

NOT

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » N » **NOT**
Syntax: NOT Ausdruck
Typ: Operator
Kategorie: Operatoren

NOT vertauscht die Bits im Quellausdruck; aus 1 wird 0 und aus 0 wird 1. NOT wird in Bedingungen eingesetzt, um eine Aussage ins Gegenteil zu verkehren.

NOT kann mithilfe von OPERATOR überladen werden.

Beispiel 1: NOT in einer IF-THEN-Bedingung:

```
IF (a = 1) AND NOT (b = 7) THEN
    PRINT "a = 1 aber b <> 7."
ELSE
    PRINT "Moeglich sind..."
    PRINT "a <> 1 aber b = 7"
    PRINT "a <> 1 und b <> 7"
    PRINT "a = 1 aber b = 7"
END IF
```

Beispiel 2: Logische Inversion einer Zahl mit NOT

```
DIM AS UBYTE zahl = 100
PRINT zahl, BIN(zahl, 8)
PRINT "---", "--------"
PRINT (NOT zahl) AND 255, BIN(NOT zahl, 8)
GETKEY
```

Ausgabe:

```
100 01100100
--- --------
155 10011011
```

Anmerkung: Das AND 255 bewirkt hier, dass die Zahl wie ein UBYTE behandelt wird; ohne diesen Operator würde FreeBASIC den Ausdruck 'NOT zahl' zu einem INTEGER konvertieren, was das Beispiel weniger anschaulich gestalten würde.

Hinweis: NOT ist keine Funktion. Die Verwendung der Form

```
Not(Variable)
```

kann zu Problemen führen. Dazu ein kleines Beispiel:
NOT(0) + 1 wird behandelt wie NOT ((0) + 1), obwohl man diese Interpretation erwarten würde: (NOT 0) + 1

Siehe auch:
AND (Operator), OR (Operator), XOR (Operator), IMP, EQV, Bit Operatoren / Manipulationen

Letzte Bearbeitung des Eintrags am 27.12.12 um 14:51:01
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

NOW

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » N » **NOW**
Syntax: NOW
Typ: Funktion
Kategorie: Datum und Zeit

Um diese Funktion nutzen zu können, muss die Datei datetime.bi in Ihren Quellcode eingebunden werden, z. B. mit INCLUDE. Alternativ können Sie auch die Datei vbcompat.bi einbinden, da dadurch auch automatisch datetime.bi eingebunden wird. NOW wird vom Compiler solange ignoriert, bis die Datei datetime.bi (bzw. vbcompat.bi) in den Quellcode eingebunden wurde.

Diese Funktion gibt die aktuelle Systemzeit als Serial Number aus. Der Rückgabewert ist ein DOUBLE. Das Datum wird bei einer Serial Number im ganzzahligen Bereich der Zahl gespeichert, die Uhrzeit im Nachkommabereich. Das bedeutet: Wenn Sie zur Speicherung eine INTEGER-Zahl verwenden, erhalten Sie eine Serial Number, welche den Zeitpunkt der nächstliegenden Mitternacht enthält.

Beispiel:

```
"hlstring">"vbcompat.bi"
Dim a as Double
a = NOW
PRINT FORMAT( a, "dd.mm.yyyy, hh:mm:ss" )
SLEEP
```

Ausgabebeispiel:

```
06.11.2007, 18:40:14
```

Unterschiede zu QB: existiert nur in QBX PDS und in VBDOS.

Unterschiede zu früheren Versionen von FreeBASIC: existiert seit FreeBASIC v0.15

Siehe auch:
DATESERIAL, DATEVALUE, TIMESERIAL, TIMEVALUE, YEAR, MONTH, DAY, WEEKDAY, HOUR, MINUTE, SECOND, MONTHNAME, WEEKDAYNAME, DATEPART, DATEADD, FORMAT, ISDATE, Serial Numbers, Datum und Zeit

Letzte Bearbeitung des Eintrags am 22.08.12 um 23:46:27
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

OBJECT

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » O » **OBJECT**
Syntax: TYPE basisklasse EXTENDS OBJECT
Typ: Datentyp
Kategorie: Klassen

OBJECT wird in Verbindung mit Vererbung genutzt. Will man über IS den Typ einer Variablen erfahren, muss die Basis-Klasse von OBJECT erben, da nur so die RTTI-Funktionalität zur Verfügung steht, die dies ermöglicht.

Beispiel:

```
Type Haustier Extends Object
  As Integer beine   = 4
  As Integer schwanz = 1
End Type

Type Hund Extends Haustier
  ' Deklarationen für Hunde
End Type

Type Dackel Extends Hund
  ' Deklarationen für Dackel
End Type

Dim As Haustier Ptr waldi = New Dackel
If *waldi Is Hund Then Print "Waldi ist ein Hund."
If *waldi Is Dackel Then Print "Waldi ist ein Dackel."

Delete waldi
Sleep
```

Ausgabe:

```
Waldi ist ein Hund.
Waldi ist ein Dackel.
```

Erläuterung: Da 'waldi' sowohl ein Hund als auch ein Dackel ist, spricht er auf beide Abfragen an. Wenn Sie die exakte Zugehörigkeit zu einer Klasse prüfen wollen, sollten Sie aus diesem Grund die abgefragten Klassen immer in zur Erbreihenfolge entgegengesetzer Reihenfolge prüfen; siehe dazu IS (Vererbung).

IS arbeitet nur mit der Klasse, die direkt von OBJECT erbt. Andererseits kann diese Klasse nicht auf die Records und Methoden seiner Kindklassen zugreifen. Um diese Records ansprechen zu können, muss die Variable entsprechend geCASTet werden; siehe dazu IS (Vererbung).

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC: existiert seit FreeBASIC v0.24

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht OBJECT nicht zur Verfügung und kann nur über **__OBJECT** aufgerufen werden.

Siehe auch:
TYPE (UDT), BASE, EXTENDS, IS (Vererbung), Objektorientierung

Letzte Bearbeitung des Eintrags am 25.08.12 um 18:28:23
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

OCT

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » O » **OCT**
Syntax: OCT[$] (Ausdruck [, Stellen])
Typ: Funktion
Kategorie: Stringfunktionen

OCT gibt den oktalen Wert eines beliebigen Ausdrucks als STRING zurück. Oktalzahlen haben die Basis 8; die Ziffern reichen von 0 bis 7.

- 'Ausdruck' ist eine Ganzzahl (eine Zahl ohne Nachkommastellen), die ins Oktalformat übersetzt werden soll.
- 'Stellen' ist die Anzahl der Stellen, die dafür aufgewandt werden soll. Ist 'Stellen' größer als die benötigte Stellenzahl, wird der Rückgabewert mit führenden Nullen aufgefüllt; der zurückgegebene Wert ist jedoch nie länger, als maximal für den Datentyp von 'Ausdruck' benötigt wird.
  Ist 'Stellen' kleiner als die benötigte Stellenzahl, werden nur die hinteren Zeichen des Rückgabewerts ausgegeben.
  Wird 'Stellen' ausgelassen, besteht der Rückgabewert aus so vielen Zeichen, wie benötigt werden, um die Zahl korrekt darzustellen.
- Der Rückgabwert ist ein String, der den Wert von 'Ausdruck' im Oktalformat enthält.

Das Dollarzeichen ($) als Suffix ist optional.

Beispiel:

```
PRINT OCT(8)       ' Ausgabe: 10
PRINT OCT(20, 4)   ' Ausgabe: 0024
PRINT OCT(100, 2) ' Ausgabe: 44
```

Um eine Oktalzahl in ihre dezimale Form zurückzuverwandeln, wird VALINT verwendet:

```
DIM oktal AS STRING
oktal = "100"
/'Kennung &o zeigt an,
dass der folgende String eine Oktalzahl ist. '/
oktal = "&o" & oktal
PRINT VALINT(oktal)
SLEEP
```

gibt 64 aus.

Unterschiede zu QB:

- In QB kann die Anzahl der ausgegebenen Stellen nicht festgelegt werden.
- Die Größe des zurückgegebenen Strings ist in QB auf 32 Bits bzw. 11 Oktal-Stellen begrenzt.

Unterschiede unter den FB-Dialektformen:

- In der Dialektform -lang qb ist das Suffix $ verbindlich.
- In den Dialektformen -lang fblite und -lang fb ist das Suffix optional.

Siehe auch:
HEX, BIN, VAL, WOCT, Datentypen umwandeln

Letzte Bearbeitung des Eintrags am 27.12.12 um 20:04:28
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

OFFSETOF

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » O » **OFFSETOF**
Syntax: OFFSETOF (TypeName, RecordName)
Typ: Funktion
Kategorie: Klassen

OFFSETOF gibt den Offset (Abstand in Byte zur Adresse des UDTs) eines Records innerhalb eines UDTs relativ zu seinem Anfang zurück.

- 'TypeName' ist der Name des UDTs, auf den sich der Rückgabewert beziehen soll.
- 'RecordName' ist der Name des Records, dessen Offset zurückgegeben werden soll.

Beispiel:

```
TYPE MyType
  x AS SINGLE
  y AS SINGLE
  UNION
    b AS BYTE
    i AS INTEGER
  END UNION
END TYPE

PRINT "Offset von x = "; OFFSETOF(MyType, x)
PRINT "Offset von y = "; OFFSETOF(MyType, y)
PRINT "Offset von b = "; OFFSETOF(MyType, b)
PRINT "Offset von i = "; OFFSETOF(MyType, i)
```

Ausgabe:

```
Offset von x =  0
Offset von y =  4
Offset von b =  8
Offset von i =  8
```

Interne Darstellung:

"cnf">__OFFSETOF in der Dialektform -lang qb existiert seit FreeBASIC v0.24.

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht OFFSETOF nicht zur Verfügung und kann nur über **__OFFSETOF** aufgerufen werden.

Siehe auch:
TYPE, SIZEOF, VARPTR, Verschiedenes
Letzte Bearbeitung des Eintrags am 27.12.12 um 20:04:57
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

ON ... GOSUB

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » O » **ON ... GOSUB**
Syntax: ON Ausdruck GOSUB label1[, label2 [, ...]]
Typ: Anweisung
Kategorie: Programmablauf

ON...GOSUB verzweigt zu verschiedenen Labels, abhängig vom Wert des Ausdrucks. Die Anweisung kann **nur bis FreeBASIC v0.16** eingesetzt werden, oder in entsprechend höheren Versionen, die mit der Kommandozeilenoption -lang deprecated compiliert wurden! Wird mit FreeBASIC v0.17 unter der Option -lang fb compiliert, so ist ON...GOSUB nicht mehr zulässig!

- 'Ausdruck' ist ein beliebiger numerischer Ausdruck, dessen Wert zu einem INTEGER gerundet wird. Nicht zulässig sind STRINGs, ZSTRINGs und WSTRINGs.
- 'label1', 'label2', ... sind Labels, wie sie mit GOTO oder GOSUB verwendet werden.

Ist der Wert von 'Ausdruck' gleich 1, wird zum ersten Label verzweigt, ist er gleich 2, zum zweiten, und so fort. Ist der Wert von 'Ausdruck' kleiner als 1 oder größer als die Anzahl der Labels, so wird das Programm mit der Anweisung nach ON...GOSUB fortgesetzt. Da der Sprung durch GOSUB durchgeführt wurde, kann anschließend mit RETURN wieder zur Zeile nach dem ON...GOSUB zurückgesprungen werden.

ON...GOSUB kann durch andere Mechanismen, wie z. B. SELECT CASE in Verbindung mit SUB, ersetzt werden.

Beispiel:

```
"hlstring">"fblite"
OPTION GOSUB
DIM AS INTEGER wahl = 3
ON wahl GOSUB labelA, labelB, labelC
PRINT "Good bye."
SLEEP
END

labelA:
PRINT "Wahl A"
RETURN

labelB:
PRINT "Wahl B"
RETURN

labelC:
PRINT "Wahl C"
RETURN
```

Ausgabe:

```
Wahl C
Good bye.
```

Unterschiede zu QB:
FreeBASIC erzeugt keinen Laufzeitfehler, wenn 'Ausdruck' negativ oder größer als 255 ist.

Unterschiede unter den FB-Dialektformen:

- ON...GOSUB steht nur in den Dialektformen -lang fblite und -lang qb zur Verfügung.
- In der Dialektform -lang fblite muss ON...GOSUB mittels GOSUB (Schlüsselwort) aktiviert werden.

Siehe auch:
ON...GOTO, GOSUB, RETURN, GOTO, SELECT CASE, Programmablauf

Letzte Bearbeitung des Eintrags am 19.01.13 um 16:47:46
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

ON ... GOTO

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » O » **ON ... GOTO**
Syntax: ON Ausdruck GOTO label1[, label2 [, ...]]
Typ: Anweisung
Kategorie: Programmablauf

ON...GOTO verzweigt zu verschiedenen Labels, abhängig vom Wert des Ausdrucks.

- 'Ausdruck' ist ein beliebiger numerischer Ausdruck, dessen Wert zu einem INTEGER gerundet wird. Nicht zulässig sind STRINGs, ZSTRINGs und WSTRINGs.
- 'label1', 'label2', ... sind Labels, wie sie mit GOTO oder GOSUB verwendet werden.

Ist der Wert von 'Ausdruck' gleich 1, wird zum ersten Label verzweigt, ist er gleich 2, zum zweiten, und so fort. Ist der Wert von 'Ausdruck' kleiner als 1 oder größer als die Anzahl der Labels, so wird das Programm mit der Anweisung nach ON...GOTO fortgesetzt. Dieser Befehl kann durch andere Mechanismen, wie z. B. SELECT CASE, ersetzt werden.

Beispiel:

```
DIM AS INTEGER wahl = 3
ON wahl GOTO labelA, labelB, labelC

labelA:
PRINT "Wahl A"
SLEEP
END

labelB:
PRINT "Wahl B"
SLEEP
END

labelC:
PRINT "Wahl C"
SLEEP
END
```

Unterschiede zu QB:
FreeBASIC erzeugt keinen Laufzeitfehler, wenn 'Ausdruck' negativ oder größer als 255 ist.

Siehe auch:
ON...GOSUB, GOTO, GOSUB, RETURN, SELECT CASE, Programmablauf

Letzte Bearbeitung des Eintrags am 27.12.12 um 20:05:41
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

ON ERROR

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » O » **ON ERROR**
Syntax: ON [LOCAL] ERROR GOTO { label | 0 }
Typ: Anweisung
Kategorie: Fehlerbehandlung

ON ERROR bewirkt einen Programmsprung an ein angegebenes Label, sobald ein Fehler auftritt. Um ON ERROR in Ihrem Programm verwenden zu können, müssen Sie den Compiler mit der Kommandozeilenoption -e, -ex oder -exx aufrufen. Anderenfalls wird kein Sprung ausgeführt, der Befehl verbraucht aber dennoch Prozessorzeit.

- Das Label muss sich auf Modulebene befinden, darf sich also nicht in einer SUB oder FUNCTION befinden; ansonsten muss zusätzlich das Schlüsselwort LOCAL verwendet werden.
- Wird das optionale Schlüsselwort LOCAL verwendet, gilt die Fehlerbehandlung nur für die aktuelle Prozedur. In diesem Fall sucht FreeBASIC nach dem Label innerhalb der aktuellen Prozedur.
- Wird 0 als Label angegeben, werden aktivierte Fehlerbehandlungsroutinen ignoriert; wenn ein Fehler auftritt, führt FreeBASIC keinen Sprung aus.

Beispiel 1:

```
ON ERROR GOTO errorhandler  ' legt fest, zu welchem Label verzweigt
werden soll.
ERROR 24                    ' Fehler 24 simulieren
PRINT "Diese Zeile wird nicht angezeigt."

errorhandler:
PRINT "Fehler " & ERR & " ist aufgetreten!"   'Fehlernummer anzeigen
PRINT "Beliebige Taste zum Beenden"
SLEEP
END
```

Beispiel 2: Fehlerroutine in einer SUB

```
ON ERROR GOTO errorhandler             ' globale Fehlerroutine setzen

SUB Test
  ON LOCAL ERROR GOTO suberrorhandler  ' Fehlerroutine lokal ersetzen
  ERROR 24

  suberrorhandler:
  PRINT "In der SUB 'Test' ist der Fehler " & ERR & " aufgetreten!"
  PRINT "Beliebige Taste zum Beenden"
  SLEEP
  END
END SUB

Test
END

errorhandler:
' Diese Fehlerroutine kommt nicht zum Einsatz
PRINT "Fehler " & ERR & " ist aufgetreten!"   'Fehlernummer anzeigen
```

```
PRINT "Beliebige Taste zum Beenden"
SLEEP
END
```

Unterschiede zu QB:
In QB existiert das Schlüsselwort LOCAL nicht. Das Label muss sich auf Modulebene befinden.

Siehe auch:
RESUME, ERROR (Anweisung), ERR (Funktion), __FB_ERR__, Der Compiler, Fehler-Behandlung in FreeBASIC, Übersicht: Fehlerbehandlung, Debugging

Letzte Bearbeitung des Eintrags am 25.08.12 um 20:36:10
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

ONCE

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » O » **ONCE**
Syntax: #INCLUDE ONCE "Datei"
Typ: Schlüsselwort
Kategorie: Metabefehle

#INCLUDE ONCE bindet eine Datei in den Quellcode ein, verhindert dabei aber, dass die Datei mehrfach eingebunden wird.

Beispiel:

```
"hlkw0">ONCE "windows.bi"
```

Unterschiede zu QB: ONCE ist neu in FreeBASIC

Siehe auch:
INCLUDE, Präprozessor-Anweisungen

Letzte Bearbeitung des Eintrags am 27.12.12 um 20:06:26
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

OPEN (Anweisung)

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » O » **OPEN (Anweisung)**
Syntax:

```
OPEN Datei FOR Dateimodus [ACCESS Zugriffsart] [LOCK Sperrart] _
     [ENCODING "Format"] AS [#]Dateinummer [LEN = Recordlänge]
```

Typ: Anweisung
Kategorie: Dateien

Anmerkung zur Syntax: Unterstriche (_) am Zeilenende werden von FreeBASIC so interpretiert, als wäre die Zeile nicht unterbrochen; dies dient nur der besseren Übersichtlichkeit und hat letzt–end–lich keine Auswirkungen auf die Programmausführung.

OPEN bereitet eine Datei auf einen Lese- und/oder Schreibzugriff vor und weist der Datei eine Dateinummer zu. OPEN kann auch zum Zugriff auf verschiedene Geräte verwendet werden; siehe die OPEN-Varianten am Ende dieses Eintrags.

- 'Datei' ist der Name einer Datei, auf die zugegriffen werden soll. Wird kein Pfad angegeben, geht FreeBASIC davon aus, dass sie sich im aktuellen Arbeitsverzeichnis befindet. Wenn die Datei nicht existiert, wird sie neu erstellt. (Ausnahme: INPUT (Dateimodus))
- 'Dateimodus' gibt an, in welchem Modus die Datei geöffnet werden soll. Möglich sind:
  - RANDOM
  - INPUT
  - OUTPUT
  - APPEND
  - BINARY

  Abhängig vom Dateimodus stehen verschiedene Dateioperationen zur Verfügung.
- '#Dateinummer' ist eine Nummer, über die später auf die Datei bzw. das Gerät zugegriffen wird. Diese Nummer darf nicht doppelt vergeben werden. Möglich sind die Nummern 1 bis 255. Die führende Raute ist optional, sollte aber verwendet werden, um sicherzustellen, dass eine Dateinummer bezeichnet wird.
- Mit 'ENCODING "Format"' kann bei sequentiellen Modi (INPUT, OUTPUT und APPEND) angegeben werden, wie die Datei kodiert werden soll. '"Format"' ist dabei einer der folgenden STRINGs, der angibt, wie die Datei kodiert sein soll:
  - ASCII
  - UTF-8
  - UTF-16
  - UTF-32

  Dadurch ist es möglich, ohne großen Aufwand Unicode-Dateien nach derzeit gültigen Kodierungsregeln zu erstellen. Wird der Parameter ENCODING ausgelassen, nimmt FreeBASIC automatisch 'ENCODING "ASCII"' an. Im BINARY- und RANDOM-Modus kann die ENCODING-Klausel nicht eingesetzt werden.
- 'Zugriffsart' gibt an, welche Zugriffe gestattet sind:
  - READ: Nur Lesezugriff
  - WRITE: Nur Schreibzugriff
  - READ WRITE: Lese- und Schreibzugriff (Standard)

  Sinnvollerweise wird diese Klausel nur mit dem RANDOM oder dem BINARY-Modus verwendet.
- 'Sperrart' gibt an, wie andere Prozesse auf die gerade geöffnete Datei zugreifen dürfen:
  - READ: Andere Prozesse dürfen auf die Datei zugreifen, sie aber nicht lesen.
  - WRITE: Andere Prozesse dürfen auf die Datei zugreifen, nicht aber in sie schreiben.
  - READ WRITE: Andere Prozesse dürfen NICHT auf die Datei zugreifen.

- 'Recordlänge' legt fest, wie viele Bytes ein Record lang sein soll; siehe RANDOM

Achtung: LOCK arbeitet zur Zeit noch nicht wie vorgesehen. Siehe dazu auch den Diskussionsbeitrag im englischen Forum.

OPEN kann auch als Funktion eingesetzt werden. In diesem Fall müssen die übergebenen Parameter von Klammern umschlossen werden. Der Rückgabewert ist ggf. eine Fehlernummer. Wird 0 zurückgeliefert, konnte die Datei geöffnet werden.

Achtung: ENCODING kann zusammen mit UTF-codierten Dateien nur dann erfolgreich eingesetzt werden, wenn das Byte Order Mark (BOM) gesetzt ist und mit der angegebenen Codierung übereinstimmt. Ansonsten wird der Laufzeitfehler 2 (File not found) zurückgegeben.

Beispiel 1:
Öffnen der UTF-8-Datei "file.ext" im aktuellen Arbeitsverzeichnis und Ausgabe ihrer ersten Zeile auf dem Bildschirm

```
Dim Zeile As String, DNr As Integer = FREEFILE
OPEN "file.ext" FOR INPUT ENCODING "UTF-8" AS "hlkw0">LINE INPUT
"hlzeichen">, Zeile
  PRINT Zeile
CLOSE "hlkw0">SLEEP
```

Beispiel 2:
Dasselbe Beispiel wie oben, mit Einsatz von OPEN als Funktion:

```
Dim Zeile As String, DNr As Integer = FREEFILE
IF OPEN("file.ext" FOR INPUT ENCODING "UTF-8" AS "hlzeichen">) = 0 THEN
  LINE INPUT "hlzeichen">, Zeile
  PRINT Zeile
  CLOSE "hlkw0">END IF
SLEEP
```

Zugriff auf Geräte
Neben Dateien lassen sich auch 'Geräte' öffnen. Geräte sind entweder von FreeBASIC verwaltete Speicherbereiche, in denen verschiedene, interne Informationen gespeichert werden, oder externe Anschlüsse wie die COM-Ports oder der LTP-Druckerport. Um mit OPEN ein Gerät zu öffnen, wird einfach einer dieser Bezeichner bei 'Gerät' angegeben.
Beispiel **LPT**: Dieses Gerät ermöglicht die Kommunikation mit dem LPT-Port und dadurch mit dem Drucker.
Weitere Informationen zum Thema finden sich bei den Spezialseiten zum Thema:

- OPEN COM
- OPEN CONS
- OPEN ERR
- OPEN LPT
- OPEN PIPE
- OPEN SCRN

Unterschiede zu QB:

- Die Verwendung von MS-DOS Gerätenamen zum Öffnen eines Datenstroms zu Hardwaregeräten ("LPT:", "SCR:", etc.) wird nur in der Dialektform -lang qb unterstützt. Verwenden Sie ansonsten die

neuen Schlüsselwörter, die oben im Abschnitt 'Zugriff auf Geräte' aufgelistet sind.
- OPEN kann in FreeBASIC als Funktion eingesetzt werden, die einen Fehlercode zurückgibt.

Plattformbedingte Unterschiede:

- Unter Linux muss der Dateiname 'case sensitive' erfolgen. Windows und DOS sind 'case insensitive'.
- Das Trennzeichen für den Dateipfad ist unter Linux der vorwärtsgerichtete Slash /. Windows verwendet den Backslash \, erlaubt aber auch den Slash. DOS verwendet den Backslash.
- Unter Windows ist die Dateinummer, die in einer DLL verwendet wird, nicht dieselbe wie eine identische Dateinummer, die vom Hauptprogramm verwendet wird. Dateinummern können nicht einfach zwischen DLL und Hauptprogramm ausgetauscht werden.

Unterschiede zu früheren Versionen von FreeBASIC:
Der Zugriff auf Geräte mittels OPEN COM usw. (s. o.) existiert seit FreeBASIC v0.15

Unterschiede unter den FB-Dialektformen:
Die Dialektform -lang qb unterstützt auch die alte GW-BASIC-Syntax

```
OPEN Dateimodus, &"reflinkicon" href="temp0284.html">OPEN (Funktion),
ENCODING, FREEFILE, LPRINT (Anweisung), Dateien (Files)
```

Letzte Bearbeitung des Eintrags am 27.12.12 um 20:08:08
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

OPEN (Funktion)

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » O » **OPEN (Funktion)**
Syntax:

```
Ergebnis = OPEN (Datei FOR Dateimodus [ACCESS Zugriffsart] [LOCK
Sperrart] _
                 [ENCODING "Format"] AS [#]Dateinummer [LEN =
Recordlänge])
```

Typ: Funktion
Kategorie: Dateien

Anmerkung zur Syntax: Unterstriche (_) am Zeilenende werden von FreeBASIC so interpretiert, als wäre die Zeile nicht unterbrochen; dies dient nur der besseren Übersichtlichkeit und hat letztendlich keine Auswirkungen auf die Programmausführung.

OPEN bereitet eine Datei oder ein Gerät auf einen Lese- und/oder Schreibzugriff vor, und weist der Datei oder dem Gerät eine Dateinummer zu. Der Rückgabewert ist ggf. eine Fehlernummer. Wird 0 zurückgeliefert, konnte die Datei geöffnet werden.

Siehe OPEN (Anweisung) für eine ausführliche Beschreibung des OPEN-Befehls.

Letzte Bearbeitung des Eintrags am 27.12.12 um 20:08:30
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

OPEN COM

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » O » **OPEN COM**
Syntax: OPEN COM "COMn:Optionen, erweiterte_Optionen" AS [#]Dateinummer
Typ: Anweisung
Kategorie: Hardware

OPEN COM bereitet den Zugriff auf einen COM-Port vor.

- 'n' ist die Nummer des Ports, auf den zugegriffen werden soll.
- 'Optionen' ist eine Zeile dieses Formats:
  [Geschwindigkeit] [,[Parität] [,[Datenbits] [,[Stoppbits]]]]
  Die einzelnen Elemente werden weiter unten erklärt.
- 'Dateinummer' ist eine Ganzzahl von 1 bis 255, über die später auf den Port zugegriffen wird. Sie darf nicht doppelt vergeben werden und verhält sich wie eine Dateinummer der OPEN-Anweisung.

OPEN COM kann auch als Funktion eingesetzt werden. In diesem Fall müssen die übergebenen Parameter von Klammern umschlossen werden. Wurde der Port erfolgreich geöffnet, wird 0 zurückgeliefert. Bei einem Fehler wird -1 zurückgegeben.

Die Parameter in 'Optionen' haben die folgenden Bedeutungen:

Option	Beschreibung	Bereich (Standard fettgedruckt)
Geschwindigkeit	gibt die Geschwindigkeit der Datenübermittlung in Baud (Bits pro Sekunde) an.	75, 110, 150, **300**,.., 115200
Parität	gibt die Parität an:	
	none	N
	even	**E**
	odd	O
	space	S
	mark	M
	error checking	PE
Datenbits	Anzahl der Datenbits pro Byte	5, 6, **7**, oder 8
Stoppbits	Anzahl der Stoppbits	**1**, 1.5, oder 2 (der Standardwert ist abhängig von Baudrate und Datenbits)

Die Reihenfolge innerhalb dieser Liste muss eingehalten werden. Benutzen Sie Kommata, um Platzhalter zu verwenden:

```
OPEN COM "COM1: ,N,8," AS "hlkw0">OPEN COM "COM1:9600,N,8,1" AS
"hlkommentar">' Zugriff auf COM1 mit 9600 Baud, ohne
' Paritäts-Bit, acht Datenbits, ein Stoppbit

DIM AS INTEGER filenumber = FREEFILE
OPEN COM "COM1:115200" AS "hlkommentar">' Zugriff auf COM1 mit 115200
Baud, gerader Parität, 7 Datenbits und 1 Stoppbit
```

Unterschiede zu QB:

- Neben der geänderten Syntax lässt sich OPEN COM in FreeBASIC auch als Funktion einsetzen.
- In QB werden nur "COM1:" und "COM2:" unterstützt. In FreeBASIC kann jede korrekt konfigurierte serielle Schnittstelle verwendet werden.

Plattformbedingte Unterschiede:

- Unter Windows verweist "COM:" auf "COM1:"
- Unter Linux verweist
  - "COM:" auf "/dev/modem"
  - "COM1:" auf "/dev/ttyS0"
  - "COM2:" auf "/dev/ttyS1", usw.
  - "/dev/xyz:" auf "/dev/xyz", usw.
- Die seriellen Schnittstellen unter DOS sind experimentell, und es kann auf die Ports 1 bis 4 zugegriffen werden. Standardmäßig werden für die Ports folgende Basisadressen und IRQs verwendet:
  - COM1 - &h3F8 - IRQ4
  - COM2 - &h2F8 - IRQ3
  - COM3 - &h3E8 - IRQ4
  - COM4 - &h2E8 - IRQ3

  Seit fbc v0.18.4 kann mit der Protokoll-Option "IRn" ein alternatives IRQ festgelegt werden, wobei "n" eine Zahl von 3 bis 7 ist.
  Im Augenblick nicht unterstützt werden IRQ am Slave-PIC, alternative Basisadressen für I/O, Timeouts und die meisten in QB entdeckten Fehler, hardware flow control sowie FIFOs.

Siehe auch:
OPEN (Anweisung), INP, OUT, WAIT, LPRINT (Anweisung), LPRINT USING, Hardware-Zugriffe

Letzte Bearbeitung des Eintrags am 25.08.12 um 21:46:18
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

OPEN CONS

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » O » **OPEN CONS**
Syntax: OPEN CONS FOR Dateimodus AS #Dateinummer
Typ: Anweisung
Kategorie: System

OPEN CONS öffnet die Standardeingabe *stdin* sowie die Standardausgabe *stdout*. Dies ist im Normalfall das Konsole-Fenster. Im Konsolenmodus kann die Ausgabe aber auch umgeleitet werden: Wird das Programm mit Umleitungszeichen (<, >, |) aufgerufen, interpretiert sie OPEN CONS entsprechend.

OPEN CONS funktioniert also genauso wie OPEN SCRN, mit dem Unterschied, dass mit OPEN CONS auch gelesen werden kann.

- 'Dateimodus' gibt einen der sequentiellen Dateimodi INPUT oder OUTPUT an.
- 'Dateinummer' ist eine Ganzzahl von 1 bis 255, über die später auf den Puffer zugegriffen wird. Sie darf nicht doppelt vergeben werden und verhält sich wie eine Dateinummer der OPEN-Anweisung.

LOCATE (Anweisung) und COLOR (Anweisung) haben keine Auswirkung auf die Ausgaben, die mit OPEN CONS gemacht werden.

Beispiel:

```
' Quelltext zu ConsTest.exe
Dim DateiNr As Integer
DateiNr = FREEFILE
OPEN CONS FOR OUTPUT AS "hlkw0">PRINT "hlzeichen">, "HELLO"
CLOSE "hlkw0">SLEEP
```

Wird dieses Programm normal aufgerufen, gibt FreeBASIC "HELLO" auf dem Bildschirm aus, jedoch führt die Zeile

```
ConsTest >>ConTestOut.txt
```

zu einer Ausgabe in die Datei ConTestOut.txt. Um die Standardeingabe bzw. -ausgabe zurückzusetzen, muss RESET verwendet werden.

Unterschiede zu QB:
In QB lautet der Befehl OPEN "CONS:" ...

Plattformbedingte Unterschiede:
Unter Linux kann ein mit OPEN CONS geöffneter Datenstrom während des Programmlaufs nicht mehr korrekt geschlossen werden. Andere Betriebssysteme als Linux und Windows wurden nicht getestet.

Unterschiede zu früheren Versionen von FreeBASIC:

- OPEN CONS existiert seit FreeBASIC v0.15
- Ab v0.12 bis v0.14 existierte der analoge Befehl OPEN "CONS:" ...

Siehe auch:
OPEN (Anweisung), OPEN ERR, PRINT "reflinkicon" href="temp0469.html">WRITE #, INPUT #, LINE INPUT #, CLOSE, RESET, Dateien (Files)

Weitere Informationen:

Wikipedia-Artikel zu den Standard-Datenströmen

Letzte Bearbeitung des Eintrags am 16.06.13 um 22:22:56
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

OPEN ERR

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » O » **OPEN ERR**
Syntax: OPEN ERR [FOR Dateimodus] AS [#]Dateinummer
Typ: Anweisung
Kategorie: System

Mit OPEN ERR wird eine Eingabe von der Standardeingabe *stdin* sowie eine Ausgabe auf die Standardfehlerausgabe *stderr* geöffnet. Standardausgabe (*stdout*) und Standardfehlerausgabe können sich unterscheiden; in der Regel leitet *stderr* auf die Konsole weiter. Damit kann OPEN ERR z. B. verwendet werden, wenn die Standardausgabe mit OPEN CONS umgeleitet wird und dennoch eine Meldung im Konsolenfenster erscheinen soll, oder wenn ein Grafikmodus verwendet wird (siehe SCREENRES) und zu Debugging-Zwecken eine Statusmeldungen auf der Konsole ausgegeben werden soll.

- 'Dateimodus' wurde nur aus Kompatibilitätsgründen beibehalten und wird ignoriert.
- 'Dateinummer' ist eine Ganzzahl von 1 bis 255, über die später auf den Puffer zugegriffen wird. Sie darf nicht doppelt vergeben werden und verhält sich wie eine Dateinummer der OPEN-Anweisung.

LOCATE (Anweisung) und COLOR (Anweisung) haben keine Auswirkung auf die Ausgaben, die mit CONS oder ERR gemacht werden.

Beispiel 1: Verwendung zusammen mit OPEN CONS

```
' Quelltext ConsErrTest.exe
Dim As Integer stdout, stderr
stdout = FREEFILE
OPEN CONS FOR OUTPUT AS "hlzeichen">= FREEFILE
OPEN ERR  FOR OUTPUT AS "hlkw0">PRINT "hlzeichen">, "HELLO"
PRINT "hlzeichen">, "FINISHED"
CLOSE "hlzeichen">, #stderr
```

Wird dieses Programm mit der Zeile

```
ConsErrTest>>ConErrTestOut.txt
```

aufgerufen, gibt FreeBASIC die Zeile "HELLO" in die Datei aus, während "FINISHED" auf dem Bildschirm erscheint. Ohne die Umleitungszeichen erscheinen beide Zeilen auf dem Bildschirm.

Beispiel 2: Verwendung zusammen mit SCREENRES

```
DIM AS INTEGER stderr = FREEFILE
DIM AS STRING text
OPEN ERR  FOR OUTPUT AS "hlkw0">SCREENRES 640, 480
LINE INPUT "Geben Sie einen beliebigen Text ein: ", text
IF text = "" THEN PRINT "hlzeichen">, "Warnung: Es wurde kein Text
eingegeben!"
PRINT "Sie haben folgendes geschrieben:"
PRINT text
SLEEP
```

Die Ausgabe der Warnung wird auf der Konsole ausgegeben, um damit den weiteren Programmablauf nicht zu stören.

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC:

- OPEN ERR existiert seit FreeBASIC v0.15
- Ab v0.12 bis v0.14 existierte der analoge Befehl OPEN "ERR:" ...

Siehe auch:
OPEN (Anweisung), OPEN CONS, PRINT "reflinkicon" href="temp0469.html">WRITE #, INPUT #, LINE INPUT #, CLOSE, Dateien (Files)

Weitere Informationen:
Wikipedia-Artikel zu den Standard-Datenströmen

Letzte Bearbeitung des Eintrags am 27.12.12 um 20:09:23
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

OPEN LPT

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » O » **OPEN LPT**
Syntax: OPEN LPT "LPT:Printer_Name,TITLE=Doc_Title,EMU=TTY" [FOR Dateimodus] AS #Dateinummer
Typ: Anweisung
Kategorie: Hardware

OPEN LPT bereitet einen Drucker darauf vor, Daten vom Befehl PRINT # bzw. PUT # auszugeben. Anders als bei LPRINT kann jeder Drucker angesprochen werden, der im System registriert ist. Auch USB-Drucker können verwendet werden.

- 'x' gibt eine Port-Nummer an. Wird die Zahl ausgelassen, dann werden die Daten an den Drucker-Spooler gesendet.
- 'Printer_Name' ist der Name des Druckers, wie er im System eingetragen ist. Wird dieser Parameter ausgelassen, benutzt FreeBASIC automatisch den Standarddrucker.
- 'Doc_Title' ist der Titel, der im Viewer des Drucker-Spoolers angezeigt wird. Wird dieser Parameter ausgelassen, verwendet FreeBASIC einen Standardtitel, der den Namen der Anwendung enthält, die den Druckauftrag ausgesandt hat.
- Das Argument 'EMU=TTY" ermöglicht die Benutzung der Sonderzeichnen CR (CHR(13), Carriage Return), LF (CHR(10), Line Feed), BS (CHR(8), Backspace), TAB (CHR(9), Tabulator), etc. Dies funktioniert sogar dann, wenn der Drucker ein GDI-Drucker ist und diese Sonderzeichen selbst nicht interpretieren kann. Wenn 'EMU=TTY' ausgelassen wird, müssen die Daten in der Druckersprache gesendet werden (ESC/P, HPGL, PostScript, ...). Andere Emulationen werden noch nicht unterstützt. "EMU=TTY" steht nur unter Windows zur Verfügung.
- 'FOR Dateimodus' wurde nur aus Kompatibilitätsgründen beibehalten und wird ignoriert. [
- 'Dateinummer' ist eine Ganzzahl von 1 bis 255, über die später auf den Port zugegriffen wird. Sie darf nicht doppelt vergeben werden und verhält sich wie eine Dateinummer der OPEN-Anweisung.

OPEN LPT kann auch als Funktion eingesetzt werden. In diesem Fall müssen die übergebenen Parameter von Klammern umschlossen werden. Der Rückgabewert ist ggf. eine Fehlernummer. Wird 0 zurückgeliefert, konnte der Drucker geöffnet werden.

Beispiel 1:
"Hello World" drucken

```
Dim As Integer Drucker = FREEFILE
OPEN LPT "LPT:" FOR OUTPUT AS "hlkw0">PRINT "hlzeichen">,"Hello World!"
PRINT "hlzeichen">, CHR(12)    'Neue Seite: die meisten Druckertreiber
machen nichts
                            'bis eine neue Seite empfangen wird
CLOSE "hlkw0">DIM RptInput AS STRING
Dim as Integer RptFilehandle = FREEFILE

OPEN "test.txt" FOR INPUT AS "hlkw0">Dim as Integer Printhandle =
FREEFILE
OPEN LPT "LPT:ReceiptPrinter,TITLE=ReceiptWinTitle,EMU=TTY" AS
"hlkw0">WHILE NOT EOF(RptFilehandle)
   LINE INPUT "hlzeichen">, RptInput
   PRINT "hlzeichen">, RptInput
WEND

CLOSE "hlkommentar">' Ein CHR(12) zum Bestätigen des Druckauftrages ist
nicht
```

```
                    ' notwendig. Wenn allerdings eine neue Seite begonnen
werden
                    ' soll, muss dennoch CHR(12) gesendet werden.
CLOSE "hlkw0">PRINT "Beliebige Taste zum Beenden druecken..."
SLEEP
```

Plattformbedingte Unterschiede:

- Das Argument "EMU=TTY" wird unter Linux und DOS ignoriert.
- "Printer_Name" und "TITLE=Doc_Title" werden unter DOS ignoriert.
- Unter Linux muss ein Drucker-Spooler installiert sein, der über lp verfügbar ist. Der Zugriff auf den Spooler wurde nur mit CUPS getestet, aber es können auch andere Spooler funktionieren, die durch lp aufgerufen werden. Bei der Angabe des Ports entspricht "LPT1:" "/dev/lp0" usw.
  Die Daten müssen in einer Druckersprache (ESC/P, HPGL, PostScript etc.) gesendet werden.
  Emulationsmodi werden unter Linux noch nicht unterstützt.
- Unter DOS werden Drucker-Spooler nicht unterstützt. Drucker müssen über "LPTx:" geöffnet werden.
  Die Daten müssen in einer Druckersprache (ESC/P, HPGL, PostScript etc.) gesendet werden.
  Emulationsmodi werden unter DOS noch nicht unterstützt.

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb wird die alte Syntax OPEN "LPT:..." unterstützt. Wird diese Syntax in einer anderen Dialektform eingesetzt, dann wird eine normale Datei geöffnet.

Siehe auch:
OPEN (Anweisung), LPRINT (Anweisung), Hardware-Zugriffe

Letzte Bearbeitung des Eintrags am 25.08.12 um 22:05:19
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

OPEN PIPE

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » O » **OPEN PIPE**
Syntax: OPEN PIPE "Shellbefehl" FOR Dateimodus [ACCESS Zugriffsart] AS [#]Dateinummer
Typ: Anweisung
Kategorie: System

PIPE ist ein Gerät, auf das man mit INPUT # zugreifen kann. In PIPE wird die Ausgabe eines SHELL-Befehls gespeichert. Die Ausgabe dieses Befehls wird aber nicht auf den Bildschirm geleitet, sondern in einen Dateipuffer umgeleitet, auf den mit INPUT # und LINE INPUT # zugegriffen werden kann.

- '"Shellbefehl"' ist eine beliebige Kommandozeile, wie sie mit SHELL aufgerufen werden kann. Diese Kommandozeile wird normal ausgeführt.
- 'Dateimodus' gibt einen der Dateimodi INPUT, OUTPUT oder BINARY an. Bei BINARY wird zudem die Angabe der Zugriffsart erwartet. Der Zugriff auf den aufgerufenen Befehl erfolgt, je nach Angabe, über die Standardeingabe (*stdin*) oder die Standardausgabe (*stdout*) des Befehls.
- 'Zugriffsart' wird zusammen mit dem Dateimodus BINARY benötigt und lautet entweder READ oder WRITE.
- 'Dateinummer' ist eine Ganzzahl von 1 bis 255, über die später auf den Puffer zugegriffen wird. Sie darf nicht doppelt vergeben werden und verhält sich wie eine Dateinummer der OPEN-Anweisung.

Beispiel:

```
"hlkw0">__FB_UNIX__
  CONST TEST_COMMAND = "ls *"
"hlkw2">CONST TEST_COMMAND = "dir *.*"
"hlkw0">DIM AS INTEGER nr = FREEFILE
DIM s AS STRING

OPEN PIPE TEST_COMMAND FOR INPUT AS "hlkw0">DO UNTIL EOF(nr)
   LINE INPUT "hlzeichen">, s
   PRINT s
LOOP
CLOSE "hlkw0">SLEEP
```

Ein umfangreicheres Beispiel zu OPEN PIPE findet sich in der Code-Beispiel-Rubrik des FreeBASIC-Portals.

FreeBASIC unterstützt keine bidirektionalen Pipes. Diese müssen über die API-Funktionen des Betriebssystems initialisiert werden.

Unterschiede zu QB: neu in FreeBASIC

Plattformbedingte Unterschiede:
Der Dateimodus BINARY wird nicht von allen Plattformen unterstützt.

Unterschiede zu früheren Versionen von FreeBASIC:

- OPEN PIPE existiert seit FreeBASIC v0.15
- Ab v0.13 bis v0.14 existierte der analoge Befehl OPEN "PIPE:Shellbefehl" ...

Siehe auch:
OPEN (Anweisung), CLOSE, INPUT "reflinkicon" href="temp0229.html">LINE INPUT #, EOF, SHELL, Dateien (Files)

Weitere Informationen:
Wikipedia-Artikel zu Pipe

Letzte Bearbeitung des Eintrags am 27.12.12 um 20:11:03
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

OPEN SCRN

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » O » **OPEN SCRN**
Syntax: OPEN SCRN [FOR Dateimodus] AS #Dateinummer
Typ: Anweisung
Kategorie: System

Über SCRN kann auf das Standardausgabegerät im Schreibmodus zugegriffen werden. Das Standardausgabegerät ist im Normalfall das Konsole-Fenster. Durch die Umleitungszeichen kann die Ausgabe aber auch umgeleitet werden: Wird das Programm mit Umleitungszeichen (<, >, |) aufgerufen, interpretiert sie SCRN entsprechend. Auf SCRN können nur Ausgaben erfolgen; ein Versuch, von SCRN zu lesen, führt zu einer Endlosschleife.

- 'Dateimodus' wurde nur aus Kompatibilitätsgründen beibehalten und wird ignoriert.
- 'Dateinummer' ist eine Ganzzahl von 1 bis 255, über die später auf den Puffer zugegriffen wird. Sie darf nicht doppelt vergeben werden und verhält sich wie eine Dateinummer der OPEN-Anweisung.

LOCATE (Anweisung) und COLOR (Anweisung) haben keine Auswirkung auf die Ausgaben, die mit SCRN gemacht werden.

Beispiel:

```
' Quelltext zu ScrnTest.exe
DIM AS INTEGER nr = FREEFILE
OPEN SCRN AS "hlkw0">PRINT "hlzeichen">, "Test"
CLOSE "cnf">OPEN "SCRN:" ...
```

Unterschiede zu früheren Versionen von FreeBASIC: existiert seit FreeBASIC v0.15

Siehe auch:
OPEN (Anweisung), OPEN CONS, PRINT #, WRITE #, INPUT #, LINE INPUT #, CLOSE, Dateien (Files)

Weitere Informationen:
Wikipedia-Artikel zu den Standard-Datenströmen
Letzte Bearbeitung des Eintrags am 27.12.12 um 20:11:42
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

OPERATOR

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » O » **OPERATOR**
Syntax A:

```
{ TYPE | CLASS | UNION } Typenname
  ' Typdeklarationen
  DECLARE &"hlzeichen">] OPERATOR Operatorname (&"hlzeichen">]) [AS
Datentyp&"hlkw0">END { TYPE | CLASS | UNION }
```

Syntax B:

```
&"hlzeichen">] OPERATOR Typename.Operatorname (&"hlzeichen">]) [AS
Datentyp&"reflinkicon" href="temp0423.html">TYPE, CLASS, UNION und (mit
Einschränkungen) ENUM eingesetzt werden.
```

- 'Typenname' gibt den Namen von TYPE, CLASS, UNION oder ENUM an.
- 'Operatorname' ist eine Zuweisung, ein Iterator oder ein unärer oder binärer Operator.

Als 'Operatorname' sind folgende Operatoren erlaubt:

Zuweisung:

- LET
- +=
- -=
- *=
- &=
- /=
- \=
- MOD=
- SHL=
- SHR=
- AND=
- OR=
- XOR=
- IMP=
- EQV=
- ^=

unärer Operator:

- - (Negation)
- NOT
- @
- []
- * (Pointer-Dereferenzierung)
- ->
- NEW
- DELETE
- ABS
- SGN
- FIX
- FRAC
- INT
- EXP
- LOG
- SIN
- ASIN
- COS
- ACOS
- TAN
- ATN
- CAST

binärer Operator:

- + (Addition)
- - (Subtraktion)
- * (Multiplikation)
- & (Stringverknüpfung)
- / (Division)
- \ (Integerdivision)
- MOD
- SHL
- SHR
- AND
- OR
- XOR
- IMP
- EQV
- ^ (Exponent)
- = (Vergleich)
- <>
- <
- >
- <=
- >=

Iterator:

- FOR
- STEP
- NEXT

Eine Überladung der unären Operatoren zur Pointer-Dereferenzierung * und der Pointer-Referenzierung -> arbeitet zur Zeit nicht korrekt.

Hinweis: LET verweist auf den Zuweisungsoperator wie in LET a=b (das Schlüsselwort LET wird im

üblichen Gebrauch weggelassen)

Ein Beispiel zur Überladung von Iteratoren wird in diesem Tutorial behandelt.

Beschreibung:
Mit OPERATOR kann festgelegt werden, wie die oben genannten Operatoren arbeiten sollen, wenn mindestens eines der Argumente dem Datentyp des TYPE-, CLASS-, ENUM- oder UNION-Blocks entspricht.

Operatoren sind Funktionen. Der Operator '+' arbeitet wie
FUNCTION plus(a AS datentyp, b AS datentyp) AS datentyp
Operatoren können überladen werden, um verschiedene Datentypen als Parameter zu akzeptieren. Nur der CAST-Operator kann auch dazu überladen werden, um verschiedene Datentypen zurückzugeben.

Manche Operatoren werden innerhalb des Typs oder der Klasse deklariert, manche außerhalb. Dabei gilt:

- Zuweisungen und Iteratoren müssen innerhalb von TYPE oder CLASS (usw.) deklariert werden.
- NEW, DELETE, @ und CAST müssen ebenfalls innerhalb von TYPE oder CLASS deklariert werden. Bei NEW und @ ist der Rückgabewert ein Pointer. DELETE besitzt keinen Rückgabewert. CAST kann mehrmals mit verschiedenen Rückgabewerten definiert sein, darf jedoch nicht den Datentyp des TYPE bzw. CLASS zurückgeben.
- Unäre und binäre Operatoren (außer @ und CAST) müssen außerhalb von TYPE oder CLASS deklariert werden. Ihr Rückgabewert wird explizit angegeben. Sie können jeden erlaubten Datentyp zurückgeben, außer -> (Zeiger auf Element-Zugriff), das den Datentyp von TYPE bzw. CLASS zurückgeben muss. Vergleichsoperatoren können nur dann sinnvoll eingesetzt werden, wenn der Rückgabewert als Zahl interpretiert werden kann.

Sämtliche Definitionen der Operatoren stehen außerhalb der TYPE- bzw. CLASS-Definition.

Da innerhalb eines ENUM keine Deklarationen stehen dürfen, ist hier auch nur die Überladung von unären und binären Operatoren möglich.

Beispiel:

```
TYPE Vector
  AS SINGLE x, y
  ' gibt einen String zurück, der die Vektordaten enthält.
  DECLARE OPERATOR CAST AS STRING
End Type

' erlaubt die Addition zweier Vektoren
DECLARE OPERATOR + (v1 AS Vector, v2 AS Vector) AS Vector

OPERATOR Vector.CAST () AS STRING
  RETURN "(" + STR(x) + ", " + STR(y) + ")"
END OPERATOR

OPERATOR + (v1 AS Vector, v2 AS Vector) AS Vector
  RETURN TYPE<Vector>( v1.x + v2.x, v1.y + v2.y )
END OPERATOR
```

```
DIM a AS Vector = TYPE<Vector>( 1.2, 3.4 )
DIM b AS Vector = TYPE<Vector>( 8.9, 6.7 )

PRINT "a = "; a
PRINT "b = "; b
PRINT "a + b = "; a + b
SLEEP
```

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC:

- Der Indexoperator &"reflinkicon" href="temp0423.html">TYPE (UDT), UNION, Ausdrücke und Operatoren, Objektorientierung

Letzte Bearbeitung des Eintrags am 23.11.13 um 18:01:56
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

OPTION

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » O » **OPTION**
Syntax: OPTION Programmstandard
Typ: Anweisung
Kategorie: Programmoptionen

OPTION setzt verschiedene Standards in Ihrem Programm. Die Anweisung kann **nur bis FreeBASIC v0.16** eingesetzt werden, oder in entsprechend höheren Versionen, die mit der Kommandozeilenoption -lang deprecated compiliert wurden! Wird mit FreeBASIC v0.17 unter der Option -lang fb compiliert, so ist OPTION nicht mehr zulässig!

'Programmstandard' ist eines der folgenden Schlüsselwörter:

- BASE
- PRIVATE (Schlüsselwort)
- BYVAL (Schluesselwort)
- DYNAMIC (Schlüsselwort) oder STATIC (Schlüsselwort)
- EXPLICIT
- ESCAPE
- GOSUB
- NOGOSUB
- NOKEYWORD

Dieser Befehl sollte nicht mit Option() verwechselt werden, siehe Option()

OPTION-Zeilen *sollten* die ersten Zeilen Ihres Programms darstellen, da eine Änderung eines Standards während des Programmverlaufs zu Problemen führen kann. Dieses Beispiel verdeutlicht die Problematik:

```
DECLARE SUB s1 (a AS INTEGER)

OPTION BYVAL

DECLARE SUB s2 (b AS INTEGER)

SUB s1(a AS INTEGER)
END SUB

SUB s2(a AS INTEGER)
END SUB
```

Der Versuch, dieses Programm zu compilieren würde fehlschlagen. Während der Zeile

```
DECLARE SUB s1 (a AS INTEGER)
```

gilt OPTION BYVAL noch nicht; in dieser Zeile wird also noch angenommen, dass der Parameter 'a' BYREF übergeben wird. Der zugehörige Sub-Header

```
SUB s1(a AS INTEGER)
```

wird allerdings schon durch OPTION BYVAL beeinflusst; hier wird angenommen, dass 'a' BYVAL übergeben werden soll. Aufgrund des Unterschieds zwischen Deklarationszeile und Prozedurkopf gibt der Compiler die Fehlermeldung

```
error 55: Type mismatch, at parameter 1 (a) of s1()
SUB s1 (a As Integer)
```

aus.

Unterschiede zu QB:
Unter QB ist nur OPTION BASE möglich.

Unterschiede zu früheren Versionen von FreeBASIC:

- Rufen Sie bitte die einzelnen Einträge auf, um die Versionsunterschiede für die verschiedenen Standards im Speziellen einzusehen.
- Seit FreeBASIC v0.17 können Programmstandards nur noch dann mit OPTION festgelegt werden, wenn die Kommandozeilenoption -lang deprecated angegeben wurde. Soll mit -lang fb compiliert werden, ist es nicht mehr erlaubt, OPTION zu benutzen. Sehen Sie bitte die einzelnen Schlüsselwörter ein, um entsprechende Walkarounds nachzulesen.
- Seit FreeBASIC v0.14 gelten Programmstandards auch, wenn sie in Dateien festgelegt wurden, die per "reflinkicon" href="temp0293.html">Option(), Verschiedenes

Letzte Bearbeitung des Eintrags am 19.01.13 um 15:52:33
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

Option()

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » O » **Option()**
Syntax: [...] Option("Text")
Typ: Schlüsselwort
Kategorie: Programmoptionen

Option() ermöglicht es dem Programmierer, zusätzliche Attribute oder Merkmale zu setzen.
Der Übergabeparameter 'Text' ist ein STRING und muss in Anführungszeichen und Klammern stehen. Nicht erkannte Optionen werden ignoriert.

Die möglichen Optionen sind:

SSE
Option("SSE") bewirkt, dass eine Gleitkommazahl (SINGLE oder DOUBLE), die von einer Funktion zurückgegeben wird, im Register *xmm0* gespeichert wird. Der Befehl wird ignoriert, sollte nicht mit -fpu SSE compiliert worden ist.
Der Befehl muss sich direkt hinter dem Rückgabetypen einer Funktionsdeklaration oder -definition befinden.
Es handelt sich hierbei nur um eine Optimierung und wird nicht dazu benötigt, Programme mit -fpu SSE zu compilieren.

```
Declare Function ValueInXmm0 () As Double Option("SSE")
```

FPU
Option("FPU") bewirkt, dass eine Gleitkommazahl (SINGLE oder DOUBLE), die von einer Funktion zurückgegeben wird, im Register *st(0)* gespeichert wird. Der Befehl muss sich direkt hinter dem Rückgabetypen einer Funktionsdeklaration oder -definition befinden.

```
Declare Function ValueInStZero () As Double Option("FPU")
```

OPTION diente in früheren Versionen von FreeBASIC auch als Befehl zum Setzen von Compilieroptionen.

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC: existiert seit FreeBASIC v0.20

Siehe auch:
Compileroptionen, OPTION, Verschiedenes

Letzte Bearbeitung des Eintrags am 26.08.12 um 13:28:53
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

OR (Methode)

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » O » **OR (Methode)**
Syntax: { PUT | DRAW STRING } [Puffer,] [STEP] (x, y), [weitere Angaben ...], OR
Typ: Schlüsselwort
Kategorie: Grafik

OR ist ein Schlüsselwort, das im Zusammenhang mit PUT (Grafik) und DRAW STRING eingesetzt wird. Die Farbe des gezeichneten Pixels ist das Ergebnis eines logischen OR des zu zeichnenden Pixels mit dem zu überschreibenden Pixel.

Beispiel:
An der Position (100, 100) befindet sich ein Pixel mit dem Farbattribut 172. Dieses soll nach der OR-Methode mit einem Pixel des Farbattributs 15 überzeichnet werden. Das Ergebnis ist ein Pixel des Farbattributs 175. Dies ergibt sich folgendermaßen:

```
Dezimal     Binär
 172        10101100
  15        00001111
--OR----------OR---
 175        10101111
```

Die OR-Methode kann mit allen Farbtiefen angewandt werden, also sowohl in palettenindizierten Modi als auch in High-/Truecolor-Modi. Beachten Sie, dass in palettenindizierten Modi das sichtbare Ergebnis nicht nur von den Farbattributen, sondern auch von den zugeordneten Paletten-Einträgen abhängig ist. Siehe dazu PALETTE und Standardpaletten.

Beispiel: drei überlappende Kreise in verschiedenen Farben zeichnen

```
ScreenRes 320, 200, 16

' 3 Sprite mit je einem Kreis in den Farben rot, grün und blau erzeugen
Const As Integer radius = 32
Dim As Any Ptr rotkreis, gruenkreis, blaukreis
rotkreis   = ImageCreate(radius * 2 + 1, radius * 2 + 1, RGBA(0, 0, 0,
0))
gruenkreis = ImageCreate(radius * 2 + 1, radius * 2 + 1, RGBA(0, 0, 0,
0))
blaukreis  = ImageCreate(radius * 2 + 1, radius * 2 + 1, RGBA(0, 0, 0,
0))
Circle rotkreis,   (radius, radius), radius, RGB(255, 0, 0), , , 1, f
Circle gruenkreis, (radius, radius), radius, RGB(0, 255, 0), , , 1, f
Circle blaukreis,  (radius, radius), radius, RGB(0, 0, 255), , , 1, f

' Kreise überlappend zeichnen
Put (146 - radius, 108 - radius), rotkreis,   Or
Put (174 - radius, 108 - radius), gruenkreis, Or
Put (160 - radius,  84 - radius), blaukreis,  Or

' Speicher freigeben und auf Tastendruck warten
ImageDestroy rotkreis
ImageDestroy gruenkreis
ImageDestroy blaukreis
Sleep
```

Siehe auch:
OR (Operator), PUT (Grafik), DRAW STRING, SCREENRES, AND (Methode), PSET (Methode), PRESET (Methode), ALPHA, ADD, TRANS, CUSTOM, Grafik

Letzte Bearbeitung des Eintrags am 26.08.12 um 13:31:02
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

OR (Operator)

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » O » **OR (Operator)**
Syntax A: Ergebnis = Ausdruck1 OR Ausdruck2
Syntax B: Ausdruck1 OR= Ausdruck2
Typ: Operator
Kategorie: Operatoren

OR kann als einfacher (Syntax A) und kombinierter (Syntax B) Operator eingesetzt werden. Syntax B ist eine Kurzform von

```
Ausdruck1 = Ausdruck1 OR Ausdruck2
```

OR vergleicht zwei Werte Bit für Bit und setzt im Ergebnis nur dann ein Bit, wenn mindestens ein Bit der entsprechenden Stelle in den Ausdrücken gesetzt waren. OR wird in Bedingungen eingesetzt, wenn mindestens eine Aussage erfüllt sein muss.

OR kann mithilfe von OPERATOR überladen werden.

Beispiel 1: OR in einer IF-THEN-Bedingung:

```
IF (a = 1) OR (b = 7) THEN
PRINT "Entweder a = 1 oder b = 7."
ELSE
PRINT "a <> 1 und b <> 7."
END IF
```

Beispiel 2: Adjunktion zweier Zahlen mit OR:

```
DIM AS INTEGER z1, z2
z1 = 6
z2 = 10
PRINT z1, BIN(z1, 4)
PRINT z2, BIN(z2, 4)
PRINT "---", "----"
PRINT z1 OR z2, BIN(z1 OR z2, 4)
GETKEY
```

Ausgabe:

```
  6           0110
 10           1010
---           ----
 14           1110
```

Beispiel 3: OR als kombinierter Operator

```
DIM AS INTEGER w, bnr

PRINT "Bitte geben Sie die Nummer des Bits ein, ";
INPUT "das gesetzt werden soll.", bnr

w OR= (1 SHL bnr)
```

```
PRINT "2 ^ "; bnr; " = "; w
GETKEY
```

Ausgabebeispiel:

```
Bitte geben Sie die Nummer des Bits ein,
das gesetzt werden soll. 5
2 ^ 5 = 32
```

Unterschiede zu QB:
Kombinierte Operatoren sind neu in FreeBASIC.

Siehe auch:
OR (Methode), NOT, AND (Operator), XOR (Operator), IMP, EQV, ORELSE, Bit Operatoren / Manipulationen

Letzte Bearbeitung des Eintrags am 27.12.12 um 20:12:48
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

ORELSE

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » O » **ORELSE**
Syntax: Ausdruck1 ORELSE Ausdruck2
Typ: Operator
Kategorie: Operatoren

ORELSE prüft zwei Ausdrücke auf ihren Wahrheitsgehalt und gibt -1 zurück, wenn mindestens einer der beiden Ausdrücke wahr ist. Ansonsten wird 0 zurückgegeben.

Zunächst wird 'Ausdruck1' geprüft. Wenn dieser 0 (false) ergibt, wird auch 'Ausdruck2' ausgewertet, sonst wird -1 (true) zurückgegeben. ORELSE liefert also nur dann 0 (false), wenn beiden Ausdrücke 0 ergeben; allerdings wird 'Ausdruck2' nur dann ausgewertet, wenn 'Ausdruck1' 0 war.

Beispiel: ORELSE in einer IF-THEN-Bedingung:

```
IF (x < 0) ORELSE (x > 639) THEN
PRINT "x ist ausserhalb des sichtbaren Bereiches"
END IF
```

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC: existiert seit FreeBASIC v0.20

Siehe auch:
ANDALSO, OR (Operator), Bit Operatoren / Manipulationen

Letzte Bearbeitung des Eintrags am 27.12.12 um 20:13:27
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

OUT

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » O » **OUT**
Syntax: OUT Port, Wert
Typ: Anweisung
Kategorie: Hardware

OUT schreibt ein Byte an einen Ausgabeport.

- 'Port' ist die Adresse (16 Bit) des Ausgabeports.
- 'Wert' ist der Ausgabewert (8 Bit)

Direkte Portzugriffe sind unter Windows NT, 2000, XP und Vista ohne einen speziellen Systemtreiber nicht möglich. Vom FreeBASIC-Compiler wird ein 3KB großer Systemtreiber in die erstellte EXE integriert. Dieser Systemtreiber wird nur ausgeführt, wenn das Programm unter Administrator-Rechten gestartet wurde. Danach können auch Programme, die nicht unter Administrator-Rechten laufen, auf diesen Treiber zugreifen. Nach jedem Neustart des Betriebssystems ist ein erneuter Aufruf des Systemtreibers unter Administrator-Rechten erforderlich.

Windows NT und dessen Nachfolger verwenden dies als Sicherheitsmaßnahme, da der Büronutzer selbstverständlich nicht im schlimmsten Fall die Hardware zerstören soll. Man sollte daher wissen was man tut, wenn man OUT anwendet.

Der vom Compiler aktuell integrierte Treiber wird nicht auf Linux angewandt. Daher muss beim Zugriff unter Linux mit Root-Rechten gearbeitet werden, andernfalls findet keine Ausführung statt. Unter allen anderen Betriebssytemen, welche FreeBASIC unterstützt, ist dies nicht nötig.

Unterschiede zu früheren Versionen von FreeBASIC:
Seit FreeBASIC v0.15 wird ein Laufzeit-Fehler erzeugt, wenn der Zugriff auf den VGA-Port fehlschlägt.

Unterschiede zu QB:
OUT ist auf die Emulation des VGA-Ports begrenzt; nur die Ports &h3C7, &h3C8 und &h3C9 funktionieren.

Siehe auch: PALETTE, PALETTE GET, INP, WAIT, Hardware-Zugriffe

Letzte Bearbeitung des Eintrags am 16.06.13 um 22:37:08
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

OUTPUT

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » O » **OUTPUT**
Syntax: OPEN Dateiname FOR OUTPUT [...]
Typ: Schlüsselwort
Kategorie: Dateien

Das Schlüsselwort OUTPUT wird mit der OPEN-Anweisung verwendet und öffnet die Datei im OUTPUT-Modus. Das heißt, die Datei wird mit sequentiellem Schreibzugriff geöffnet und der Schreibzugriff erfolgt am Anfang der Datei.

Vorsicht: Wenn Sie eine bereits bestehende Datei FOR OUTPUT öffnen, wird sie gelöscht und neu angelegt! Wenn stattdessen neue Daten an eine bestehende Datei *angehängt* werden sollen, sodass vorhandene Daten erhalten bleiben, muss statt OUTPUT APPEND verwendet werden.

Siehe auch:
OPEN (Anweisung), PRINT #, WRITE #, Dateien (Files)

Weitere Informationen:
QB Express Issue #4: File Manipulation In QuickBasic: Sequential Files

Letzte Bearbeitung des Eintrags am 27.12.12 um 20:13:44
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

OVERLOAD

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » O » **OVERLOAD**
Syntax: DECLARE { SUB | FUNCTION } Funktionsname OVERLOAD (Parameterliste) [AS Typ]
Typ: Klausel
Kategorie: Deklaration

Mit der Klausel OVERLOAD können Sie Prozeduren mit unterschiedlicher Parameterliste, aber gleichem Prozedurnamen deklarieren, ohne die Fehlermeldung "Symbol already defined" zu erhalten. Sobald die Prozedur zum ersten Mal deklariert wird, muss OVERLOAD angegeben werden; die folgenden Deklarationen müssen die OVERLOAD-Klausel nicht tragen, auch wenn es keinen Fehler ausgibt, wenn sie angegeben wird.

Funktionen, die in UDTs deklariert sind, müssen die OVERLOAD-Klausel nicht erhalten, da innerhalb des UDTs standardmäßig alle Funktionen überladen sind.

Eine überladene Funktion muss sich durch mindestens einen Übergabeparameter von den anderen unterscheiden.

Beispiel:

```
DECLARE FUNCTION SUM OVERLOAD (a AS INTEGER, b AS INTEGER) AS INTEGER
DECLARE FUNCTION SUM          (a AS SINGLE, b AS SINGLE)  AS SINGLE

FUNCTION SUM  (a AS INTEGER, b AS INTEGER) AS INTEGER
   FUNCTION = a + b
END FUNCTION

FUNCTION SUM  (a AS SINGLE, b AS SINGLE) AS SINGLE
   FUNCTION = a + b
END FUNCTION

DIM AS INTEGER a , b
DIM AS SINGLE  a1, b1
a = 2
b = 3
a1 = 2.0
b1 = 3.0
PRINT SUM (a , b )
PRINT SUM (a1, b1)
SLEEP
```

Unterschiede zu QB: neu in FreeBASIC

Siehe auch:
DECLARE, FUNCTION, SUB, Prozeduren

Letzte Bearbeitung des Eintrags am 27.12.12 um 20:14:19
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

OVERRIDE

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » O » **OVERRIDE**
Syntax:

```
TYPE Typname EXTENDS Elterntyp
  DECLARE {SUB|FUNCTION|PROPERTY|OPERATOR} ... (Parameterliste) [AS
Datentyp] OVERRIDE
END TYPE
```

Typ: Klausel
Kategorie: Klassen

OVERRIDE wird bei der Deklaration von Methoden einer Klasse verwendet und gibt an, dass die dazugehörige Methode eine virtuelle oder abstrakte Methode seiner Elternklasse überschreiben **muss**. Ist dies nicht der Fall, da in der Elternklasse keine derartige Methode existiert, wird der Compiler einen Fehler ausgeben.

Beachte:
Nur nicht-statische Methoden können virtuelle oder abstrakte Methoden überschreiben.

OVERRIDE muss nicht zum Überschreiben von Methoden der Elternklasse angegeben werden, es hilft allerdings, Fehler durch falsche Methoden-Signaturen (Parameter und Name der Methode) vorzubeugen.

OVERRIDE kann nur bei der Deklaration im UDT angegeben werden, nicht aber beim Methodenrumpf, da es sich dabei um eine reine Prüfung zur Compile-Zeit handelt, die sich nicht weiter auf die Methode auswirkt.

Beispiel:

```
Type A Extends Object
  Declare Virtual Sub f1
  Declare Virtual Function f2 As Integer
End Type

Type B Extends A
  Declare Sub f1 Override
  Declare Function f2 As Integer Override
End Type

Sub A.f1
End Sub

Function A.f2 As Integer
  Return 0
End Function

Sub B.f1
End Sub

Function B.f2 As Integer
  Return 0
End Function
```

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC: existiert seit FreeBASIC v0.90.0

Unterschiede unter den FB-Dialektformen: nur in der Dialektform -lang fb verfügbar

Siehe auch:
VIRTUAL, ABSTRACT, TYPE, EXTENDS, OBJECT, Objektorientierung

Letzte Bearbeitung des Eintrags am 02.07.13 um 22:27:24
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

PAINT

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » P » **PAINT**
Syntax: PAINT [Puffer] [, STEP] (x, y)[, [Füllfarbe][, Randfarbe]]
Typ: Anweisung
Kategorie: Grafik

PAINT füllt in einem Grafikfenster einen Bereich mit einer Farbe. Das Verfahren ist auch bekannt als 'flood-fill' oder 'paint bucket'.

- 'Puffer' ist ein Speicherbereich wie ein mit IMAGECREATE erstellter Puffer oder ein Array. Beide können mit PUT angezeigt werden. Wird 'Puffer' ausgelassen, zeichnet FreeBASIC direkt auf den Bildschirm.
- 'x' und 'y' sind Koordinaten innerhalb der auszufüllenden Fläche.
- Wenn 'Füllfarbe' eine Zahl oder ein numerischer Ausdruck ist, wird die Fläche in der Farbe mit der angegebenen Farbnummer ausgefüllt. Für die Farbnummern gelten dann dieselben Regeln wie bei COLOR. Wenn hier ein STRING angegeben wird, behandelt FreeBASIC diesen als Muster, in dem die Fläche ausgefüllt werden soll.
- 'Randfarbe' ist die Farbnummer der Bereichsgrenze. PAINT füllt nur den Bereich innerhalb dieser Grenzen mit der Füllfarbe aus.

Ein Füllmuster beschreibt immer einen 8x8 Pixel großen Bereich. Jedes Zeichen innerhalb des Muster-Strings beschreibt dabei die Farbe eines Pixels. Je nach Farbtiefe können die Muster-Strings unterschiedlich lang sein; die maximale Länge berechnet sich aus den Formeln:

- Für Farbtiefen 1, 2, 4, 8: size = 8 * 8 = 64
- Für Farbtiefen 15 und 16: size = (8 * 8) * 2 = 128
- Für Farbtiefen 24 und 32: size = (8 * 8) * 4 = 256

size gibt dabei die benötigte Größe in Bytes an.
Wenn der übergebene String kleiner ist (weniger Zeichen besitzt), werden die fehlenden Bytes durch CHR(0) ersetzt.

Beispiel 1: Einen weißen, innen rot ausgefüllten Kreis zeichnen und 3 Sekunden warten

```
SCREENRES 320, 200
CIRCLE (160, 100), 30, 15
PAINT  (160, 100), 1 , 15

SLEEP 3000
```

Beispiel 2: Verwendung eines Füllmusters

```
Const bit_tiefe = 8 ' Farbtiefe wählen - möglich ist 8, 16 oder 32

' Funktion zur Umwandlung eines Farbwertes in den für das Füllmuster
benötigten String
Function paint_pixel( ByVal c As UInteger, ByVal tiefe As Integer ) As
String
  If tiefe <= 8 Then          ' 8-bit:
    Return Chr( CUByte(c) )
  ElseIf tiefe <= 16 Then     ' 16-bit:
    Return MKShort( c Shr 3 And &h1f Or _
                    c Shr 5 And &h7e0 Or _
```

```
                    c Shr 8 And &hf800 )
  ElseIf tiefe <= 32 Then     ' 32-bit:
    Return MKL(c)
  End If
End Function

ScreenRes 320, 200, bit_tiefe   ' Grafikfenster in gewünschter Bit-Tiefe
öffnen

Dim As UInteger c, c1, c2       ' Variablen für die Farbwerte
Dim As UInteger randfarbe
Dim As String fuellmuster = ""  ' Variable für das Füllmuster

' Farben setzen
If bit_tiefe <= 8 Then
  c1        = 7  ' Füllfarbe 1
  c2        = 8  ' Füllfarbe 2
  randfarbe = 15 ' Randfarbe
Else
  c1        = RGB(192, 192, 192) ' Füllfarbe 1
  c2        = RGB(128, 128, 128) ' Füllfarbe 2
  randfarbe = RGB(255, 255, 255) ' Randfarbe
End If

' Karomuster erstellen
For y As UInteger = 0 To 7
  For x As UInteger = 0 To 7
    ' Farbe des Pixels wählen (c)
    If (x \ 4 + y \ 4) Mod 2 > 0 Then
      c = c1
    Else
      c = c2
    End If
    ' Pixel zum Muster hinzufügen
    fuellmuster &= paint_pixel(c, bit_tiefe)
  Next x
Next y

' Kreis zeichnen und mit dem Muster füllen
Circle (160, 100), 50, randfarbe
Paint (160, 100), fuellmuster, randfarbe
Sleep
```

Unterschiede zu QB:

- Der optionale Parameter 'Hintergrund' existiert unter FreeBASIC nicht mehr.
- In FreeBASIC ist es möglich, in einen Datenpuffer zu zeichnen.

Siehe auch:
SCREENRES, COLOR (Anweisung), Grafik

Letzte Bearbeitung des Eintrags am 28.08.12 um 01:22:24
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

PALETTE

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » P » **PALETTE**
Syntax A: PALETTE [Index, Farbe]
Syntax B: PALETTE Index, Rotwert, Grünwert, Blauwert
Syntax C: PALETTE USING ArrayName[(idx)]
Typ: Anweisung
Kategorie: Grafik

PALETTE bearbeitet die aktuelle Farbpalette.

- 'Index' ist die Nummer der Farbe, die bearbeitet werden soll.
- 'Farbe' ist der neue **BGR**-Farbwert, welcher der angegebenen Farbnummer zugewiesen werden soll.
- 'Rotwert', 'Grünwert' und 'Blauwert' sind die einzelnen Farbanteile.
- 'ArrayName(idx)' ist ein Array, in dem sich alle neuen Farbwerte befinden. 'idx' ist dabei der Index innerhalb von ArrayName, ab dem die Farbeinträge beginnen. Ein mit ALLOCATE reservierter Speicherbereich kann nicht verwendet werden; PALETTE kann mit einem solchen Speicherbereich (noch) nicht umgehen.
- Wird PALETTE ohne Parameter aufgerufen, so wird die Standard-Palette wieder hergestellt.

PALETTE ändert die aktuellen Einträge in der Farbpalette (nur in Grafik-Modi). Die Anweisung wird nur bei Modi bis 8 bpp eingesetzt; bei höheren Farbtiefen hat PALETTE keine Auswirkung.

'Index' und 'Farbe' sind abhängig von der aktuellen Farbtiefe des Grafikfensters. Wird das Grafikfenster mit SCREEN initialisiert, dann gelten folgende Farbbereiche:

Bildschirmmodus	Indexbereich	Farbbereich
1	0-3	0-15
2	0-1	0-15
7, 8	0-15	0-15
9	0-15	0-63
11	0-1	siehe unten
12	0-15	siehe unten
13 - 21	0-255	siehe unten

In den Bildschirmmodi 1, 2, 7, 8 und 9 können Sie jeder Farbnummer einen Index aus einer vordefinierten Palette zuweisen.
In den anderen Bildschirmmodi sowie bei der Verwendung von SCREENRES müssen Sie die Farbe als BGR-Farbwerte angeben. Diese haben das Format &hBBGGRR. BB, GG und RR sind jeweils der Blau-, Grün- und Rot-Wert der Farbe. Jede Teilfarbe kann dabei einen Wert von &h0 bis &h3F annehmen (0-63 als dezimale Werte).
Eine alternative Methode der Angabe der Farbe ist diese Methode:

```
clr = rot OR (grün SHL 8) OR (blau SHL 16)
```

Auch hier müssen rot, grün und blau im Bereich von 0 bis 63 liegen.

Der nächste SCREENRES- oder SCREEN-Befehl stellt die Standardpalette wieder her (siehe auch Standard-Paletten).

Alternativ zum Befehl PALETTE können Sie OUT verwenden, um die Palette zu bearbeiten; diese Methode ist allerdings veraltet und sollte nicht mehr verwendet werden.

Es ist auch möglich, die Teilfarben als einzelne Parameter anzugeben (Syntax B). Hier kann jeder Farbanteil im Bereich von 0 bis 255 liegen.

PALETTE USING (Syntax C) ändert alle Einträge der Palette auf einmal; die dazu nötigen Farbwerte werden dabei aus einem Array gelesen. Dazu muss ein Array übergeben werden, das groß genug ist, um Farbwerte für jeden Index des aktuellen Bildschirmmodus zu enthalten. Für 1 bpp müssen das 2 Elemente, für 2 bpp 4, für 4 bpp 16 und für 8 bpp 256 Elemente sein. Es müssen INTEGER-Werte sein, die das oben angegebene Format besitzen.
Die Farben, die in 'ArrayName' ab Index 'Index' gespeichert sind, werden dann den Farbnummern zugewiesen, wobei bei den Palette-Einträgen mit Farbe 0 begonnen wird. Die Verwendung von PALETTE USING mit Teilfarben ist nicht möglich; jedes Element des Arrays muss einen BGR-Wert enthalten.

Jede Veränderung der Palette wird sofort auf dem Bildschirm sichtbar.

Beispiel:

```
SCREENRES 640, 480, 4     ' 16 indizierte Farben
DIM AS INTEGER rot, gruen, blau, farbe

' Vertikale Streifen in allen 16 Farben zeichnen
FOR i AS INTEGER = 0 TO 15
  LINE (i * 40, 0)-((i + 1) * 40, 479), i, BF
NEXT

' Auf Tastendruck warten
GETKEY

' Farbverlauf von schwarz nach blau
FOR i AS INTEGER = 0 TO 15
  rot   = 0
  gruen = 0
  blau  = i * 4
  farbe = rot OR (gruen SHL 8) OR (blau SHL 16)
  PALETTE i, farbe
NEXT

GETKEY

' Farbverlauf von schwarz nach grün
FOR i AS INTEGER = 0 TO 15
  PALETTE i, 0, (i+1)*16 - 1, 0
NEXT

GETKEY

' Farbverlauf von schwarz nach rot
DIM Farben(15) AS INTEGER
FOR i AS INTEGER = 0 TO 15
  Farben(i) = i * 4
NEXT
PALETTE USING Farben
GETKEY

' Standardpalette wiederherstellen
```

```
PALETTE
SLEEP
```

Unterschiede zu früheren Versionen von FreeBASIC:
Seit Version 0.13 ist es auch möglich, die Teilfarben als einzelne Parameter anzugeben (Syntax B).

Siehe auch:
PALETTE GET, SCREENRES, COLOR (Anweisung), Standardpaletten, Grafik

Letzte Bearbeitung des Eintrags am 28.08.12 um 01:38:29
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

PALETTE GET

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » P » **PALETTE GET**
Syntax A: PALETTE GET Index, Farbe
Syntax B: PALETTE GET Index, Rotwert, Grünwert, Blauwert
Syntax C: PALETTE GET USING ArrayName(idx)
Typ: Anweisung
Kategorie: Grafik

PALETTE GET speichert den Farbwert eines Palette-Eintrags in einer Variable, einem Array oder in drei einzelnen Speicherstellen.

- 'Index' ist der Farbindex, dessen Wert zurückgegeben werden soll.
- 'Farbe' ist eine Variable, in welcher der **BGR**-Wert gespeichert werden soll.
- 'Rotwert', 'Grünwert' und 'Blauwert' sind Variablen, in denen die einzelnen Teilfarben gespeichert werden sollen.
- 'ArrayName' ist der Name eines Arrays, in dem alle Palette-Einträge gespeichert werden sollen. 'idx' ist dabei der Index des Arrays, ab dem das Speichern beginnen soll.

Um die Palette-Einträge auszulesen, mussten bisher OUT und INP verwendet werden. PALETTE GET soll diese beiden Anweisungen ersetzen.
Indem ein Index angegeben wird, liefert PALETTE GET den zugehörigen BGR-Wert zurück und speichert ihn in einer Variable, oder es teilt ihn in seine Einzelfarben auf und speichert diese in drei Variablen (dann allerdings in der Reihenfolge rot, grün, blau). Wenn die komplette Palette kopiert werden soll, können ihre Elemente über PALETTE GET USING in ein Array kopiert werden.

PALETTE GET [USING] ist die Gegenfunktion zu PALETTE [USING]. Dort erhalten Sie weitere Informationen zum Aufbau der Farbpalette.

Beispiel:

```
DIM farbe AS INTEGER, rot AS INTEGER, gruen AS INTEGER, blau AS INTEGER,
i AS INTEGER
DIM farben(15) AS INTEGER

SCREENRES 400, 300, 4   ' 16 indizierte Farben

PALETTE GET 1, farbe
PRINT "Der BGR-Farbwert von Farbe 1 ist &h"; HEX(farbe)
PRINT

PALETTE GET 2, rot, gruen, blau
PRINT "Farbe 2 setzt sich aus diesen Farben zusammen:"
PRINT "Rot:"; rot; " Gruen:"; gruen; " Blau:"; blau
PRINT

PALETTE GET USING farben(0)
PRINT "Alle Farben:"
FOR i = 0 TO 15
  IF i = 0 THEN
    COLOR 0, 15         ' schwarz auf weiß, da man es sonst nicht sieht
  ELSE
    COLOR i, 0          ' farbig auf schwarzem Hintergrund
  END IF
```

```
  PRINT "Farbe"; i; ": ";
  IF i < 10 THEN PRINT " ";
  PRINT "&h"; HEX(farben(i), 6)
NEXT

SLEEP
```

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC: existiert seit FreeBASIC v0.14

Siehe auch:
PALETTE, SCREENRES, Grafik

Letzte Bearbeitung des Eintrags am 28.08.12 um 01:50:49
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

PASCAL

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » P » **PASCAL**
Syntax: [DECLARE] {SUB | FUNCTION} prozedurname PASCAL [evtl. weitere Angaben ...]
Typ: Schlüsselwort
Kategorie: Bibliotheken

Das Schlüsselwort PASCAL sorgt dafür, dass die Parameter bei einem Prozeduraufruf nach PASCAL-Konventionen übergeben werden. Das Schlüsselwort gewährleistet die Kompatibilität zu PASCAL-Libs mit FreeBASIC.

Die PASCAL-Aufrufkonvention ist das Gegenteil der CDECL-Konvention. Die Parameter werden in der Reihenfolge von links nach rechts (->) auf dem Stack abgelegt, und die aufgerufene Funktion muss den Stack vor dem Rücksprung mit RET XX (Return, Anzahl Bytes) ausgleichen.

PASCAL ist die Aufrufkonvention, die standardmäßig in Microsoft QuickBASIC und in der API für Windows 3.1 verwendet wird.

Beispiel: Vergleich der verschiedenen Aufrufmöglichkeiten

```
Sub s StdCall (s1 As Integer, s2 As Integer)
  Print "StdCall ", s1, s2
End Sub
Sub c Cdecl (c1 As Integer, c2 As Integer)
  Print "Cdecl ", c1, c2
End Sub
Sub p Pascal (p1 As Integer, p2 As Integer)
  Print "Pascal ", p1, p2
End Sub


Asm
  push 2 '2. Parameter - s2
  push 1 '1. Parameter - s1
  Call s 'rechts nach links

  push 2 '2. Parameter - c2
  push 1 '1. Parameter - c1
  Call c 'rechts nach links
  Add esp, 8 'der Stack muss durch die aufrufende Funktion aufgeräumt
werden

  push 1 '1. Parameter - p1
  push 2 '2. Parameter - p2
  Call p 'links nach rechts
End Asm

Sleep
```

Unterschiede zu QB: neu in FreeBASIC

Siehe auch:
DECLARE, CDECL, STDCALL, Module (Library / DLL)

Letzte Bearbeitung des Eintrags am 29.08.12 um 22:23:17
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

PCOPY

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » P » **PCOPY**
Syntax: PCOPY Quellseite, Zielseite
Typ: Anweisung
Kategorie: Grafik

PCOPY kopiert den Inhalt einer Bildschirmseite auf eine andere.

- 'Quellseite' ist die Nummer einer Bildschirmseite, die in einem vorhergehenden SCREENRES-Befehl erstellt wurde. Wird dieser Parameter ausgelassen, nimmt FreeBASIC automatisch die Nummer der aktiven Bildschirmseite an.
- 'Zielseite' ist ebenfalls die Nummer einer Bildschirmseite. Wird dieser Parameter ausgelassen, nimmt FreeBASIC automatisch die Nummer der sichtbaren Bildschirmseite an.

Der Befehl wird benutzt, um weiche Animationen zu erzeugen, indem auf eine unsichtbare Arbeitsseite gezeichnet wird, deren Inhalt per PCOPY auf die aktive Seite übertragen wird. Diese Technik ist auch als 'double buffering' oder 'page flipping' bekannt.
Die 'Zielseite' wird mit dem Inhalt der 'Quellseite' überschrieben, wenn PCOPY aufgerufen wird.

Beispiel 1:

```
ScreenRes 320, 240, 32, 2
Dim As Integer x = 50, max_x = 270

ScreenSet 0, 1        ' aktive Seite auf 0 und die sichtbare Seite auf 1
setzen

Do While x < max_x
  ' aktive Seite löschen und Kreis zeichnen
  Cls
  Sleep 25            ' Diese Pause würde ohne double buffering starkes
Flackern hervorrufen
  Circle (x, 50), 50
  x += 1              ' Kreis im nächsten Durchgang um 1 nach rechts
verschieben
  ScreenSync          ' auf Bildschirm-Aktualisierung warten
  PCopy               ' Kreis von der aktiven Seite auf die sichtbare
Seite kopieren
Loop

Sleep
```

Im Grafikmodus macht PCOPY dasselbe wie FLIP und SCREENCOPY, PCOPY ist aber der einzige Befehl, der auch im Konsolen-Modus funktioniert.

Beispiel 2: PCOPY im Konsolen-Modus (SCREEN 0)
Dieses Beispiel funktioniert nur unter Windows und DOS. Unter Linux ist der Einsatz von PCOPY im Konsolenmodus nicht möglich.

```
' aktive Seite auf 0 und die sichtbare Seite auf 1 setzen
"hlkw0">__FB_LANG__ = "QB"
Screen ,, 0, 1
"hlkw0">Screen , 0, 1
```

```
"hlkw0">Dim As Integer i, frames, fps
Dim As Double t

t = Timer

Do
  ' aktive Seite beschreiben
  Cls
  Locate 1, 1
  Color (i And 15), 0
  Print STRING$(80 * 25, Hex$(i, 1));
  i += 1

  ' Frames pro Sekunde ausgeben
  Color 15, 0
  Locate 1, 1
  Print "fps: " & fps,
  If Int(t) <> Int(Timer) Then
    t = Timer
    fps = frames
    frames = 0
  End If
  frames += 1

  ' aktive Seite auf die sichtbare Seite kopieren
  PCopy

  ' 50ms pro Frame warten, um CPU-Zeit freizugeben
  Sleep 50, 1
Loop Until Len(Inkey$)
```

Plattformbedingte Unterschiede:
PCOPY kann im SCREEN-Modus 0 nur unter Windows und DOS eingesetzt werden.

Unterschiede zu früheren Versionen von FreeBASIC:
Die Möglichkeit, PCOPY im SCREEN-Modus 0 einzusetzen (unter Windows und DOS), besteht seit FreeBASIC v18.2

Siehe auch:
FLIP, SCREENCOPY, SCREENRES, Grafik

Letzte Bearbeitung des Eintrags am 29.08.12 um 22:52:18
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

PEEK

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » P » **PEEK**
Syntax: PEEK ([Datentyp,] Adresse)
Typ: Funktion
Kategorie: Pointer

PEEK liest einen Wert direkt vom RAM.

- 'Datentyp' ist der Datentyp des Wertes, der gelesen werden soll; von 'Datentyp' hängt ab, wie viele Bytes gelesen werden. Auch STRINGs und UDTs werden akzeptiert. Die zu lesende Datenmenge richtet sich hier nach der Länge des STRINGs bzw. der Größe des UDTs. Wird 'Datentyp' ausgelassen, nimmt FreeBASIC automatisch UBYTE an.
- 'Adresse' ist ein Pointer auf die Speicherstelle im RAM, von der gelesen werden soll.
- Der Rückgabewert ist eine Referenz auf den Wert, der sich an der Speicherstelle befindet.

Mit PEEK und POKE wurden in älteren BASIC-Dialekten Pointer-Funktionen erstellt.

Beispiel: Wert dreier Variablen aus dem Speicher lesen:

```
DIM byteVal AS BYTE, shortVal AS SHORT, intVal AS INTEGER
DIM byteAddr AS BYTE PTR, shortAddr AS SHORT PTR, intAddr AS INTEGER PTR

byteAddr  = VARPTR(byteVal)
shortAddr = VARPTR(shortVal)
intAddr   = VARPTR(intVal)

byteVal   = 127
shortVal  = 32767
intVal    = 2 ^ 31 - 1

PRINT byteVal, shortVal, intVal
PRINT PEEK(BYTE, byteAddr), PEEK(SHORT, shortAddr), PEEK(INTEGER,
intAddr)
SLEEP
```

Diese Form funktioniert einwandfrei. FreeBASIC hat jedoch eigene Pointer-Funktionen, die ein komfortableres Programmieren ermöglichen. Mit den FB-eigenen Funktionen würde dieses Beispiel so aussehen:

```
DIM byteVal AS BYTE, shortVal AS SHORT, intVal AS INTEGER
DIM bytePtr AS BYTE PTR, shortPtr AS SHORT PTR, intPtr AS INTEGER PTR

bytePtr  = @byteVal
shortPtr = @shortVal
intPtr   = @intVal

byteVal   = 127
shortVal  = 32767
intVal    = 2 ^ 31 - 1

PRINT byteVal, shortVal, intVal
PRINT *bytePtr, *shortPtr, *intPtr
```

SLEEP

Die Syntax *Pointer gibt also den Wert zurück, der an der Speicherstelle steht, auf die der Pointer zeigt. Wie viele Bytes gelesen werden, hängt dabei vom Typ des Pointers ab (BYTE PTR, SINGLE PTR, ...)

Unterschiede zu QB:

- Unter QB ist es nicht möglich, den Datentyp des zu lesenden Werts festzulegen; es wird immer ein UBYTE zurückgegeben.
- Unter FreeBASIC gibt PEEK eine Referenz auf den Wert an der angegebenen Speicherstelle zurück.

Unterschiede zu früheren Versionen von FreeBASIC:

- Seit FreeBASIC v0.16 ist der Standard-Rückgabetyp UBYTE; davor war es BYTE.
- Seit FreeBASIC v0.13 ist es möglich, STRINGs und UDTs zu lesen.

Siehe auch:
POKE, VARPTR, SADD, Pointer, Datentypen, Speicher

Letzte Bearbeitung des Eintrags am 27.12.12 um 21:10:45
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

PIPE

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » P » **PIPE**
Siehe OPEN PIPE

Letzte Bearbeitung des Eintrags am 28.08.11 um 17:47:49
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

PMAP

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » P » **PMAP**
Syntax: PMAP (Koordinate, Funktion)
Typ: Funktion
Kategorie: Grafik

PMAP wandelt Sichtfensterkoordinaten in physische Koordinaten und umgekehrt.

- 'Koordinate' ist ein Ausdruck, der die umzuwandelnde Koordinate angibt.
- 'Funktion' gibt an, auf welche Weise die Koordinate umgewandelt werden soll.

PMAP verwandelt die Sichtfensterkoordinaten (abhängig von der letzten WINDOW-Anweisung) in physische Koordinaten (abhängig von der letzten VIEW-Anweisung). Je nach Wert von 'Funktion' liefert PMAP verschiedene Werte zurück.

Funktion	Ausgabe
0	behandelt den Ausdruck wie die X-Sichtfensterkoordinate und gibt die physische Koordinate zurück.
1	behandelt den Ausdruck wie die Y-Sichtfensterkoordinate und gibt die physische Koordinate zurück.
2	behandelt den Ausdruck wie die physische X-Koordinate und gibt die Sichtfensterkoordinate zurück.
3	behandelt den Ausdruck wie die physische Y-Koordinate und gibt die Sichtfensterkoordinate zurück.

Beispiel:

```
ScreenRes 640, 480
Window Screen (0, 0)-(100, 100)
Print "Logical x=50, Physical x="; PMap(50, 0)
Print "Logical y=50, Physical y="; PMap(50, 1)
Print "Physical x=160, Logical x="; PMap(160, 2)
Print "Physical y=60, Logical y="; PMap(60, 3)
Sleep
```

Siehe auch:
WINDOW, VIEW (Grafik), Grafik

Letzte Bearbeitung des Eintrags am 05.09.12 um 18:41:12
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

POINT

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » P » **POINT**
Syntax A: POINT (x, y [, Puffer])
Syntax B: POINT (Funktion)
Typ: Funktion
Kategorie: Grafik

POINT gibt Informationen über die Farbe eines Pixels oder über die aktuelle Position des Grafikcursors zurück.

- 'x' und 'y' sind die Koordinaten des Pixels, dessen Farbe zurückgegeben werden soll. Die Koordinaten sind von den letzten WINDOW- und VIEW-Anweisungen abhängig.
- 'Puffer' ist ein Puffer, wie er mit PUT verwendet wird. Wird 'Puffer' ausgelassen, so liest FreeBASIC direkt vom Bildschirm.
- 'Funktion' gibt an, welche Koordinate zurückgegeben werden soll.

Wenn die Koordinaten (Syntax A) außerhalb des aktuellen Clipping-Bereichs liegen, gibt POINT -1 zurück. Andernfalls ist das Ergebnis eine Zahl im Format, das auch bei COLOR verwendet wird.

Achtung: In einem TrueColor-Screenmodus kann auch ein weißer Pixel den Wert -1 zurückgeben!

Wenn POINT mit nur einem Parameter aufgerufen wird (Syntax B), wird die aktuelle Grafikcursorposition ausgegeben; der zurückgegebene Wert hängt von 'Funktion' ab:

0	aktuelle physische x-Koordinate
1	aktuelle physische y-Koordinate
2	aktuelle x-Bildschirmkoordinate. Wenn WINDOW nicht eingesetzt wurde, ist das Ergebnis dasselbe wie für POINT(0).
3	aktuelle y-Bildschirmkoordinate. Wenn WINDOW nicht eingesetzt wurde, ist das Ergebnis dasselbe wie für POINT(1).

Beispiel:

```
'Grafikfenster erstellen (500x200 Pixel mit 32bit Farbtiefe)
ScreenRes 500, 200, 32

'Hintergrundfarbe (rot) einstellen und einfärben
Dim As Integer farbe = &hFF0000
Color ,farbe
Cls

Print "Die Hintergrundfarbe wurde auf den Wert &h" & Hex(farbe) & "
gestellt."

Print "POINT gibt den Wert &h" & Hex(Point(10, 10)) & " zurueck."

Print "POINT mit Funktionswert '0' ergibt: " & Point(0)

Sleep
```

Wie man in einem Hi- oder Truecolor-Modus die von POINT zurückgegebene Farbe auswerten und interpretieren kann, wird in einem Codebeispiel zur RGB-Konvertierung behandelt.

Hinweis: POINT und seine Gegenfunktion PSET (Grafik) sind aufgrund der internen Berechnungen und Prüfungen sehr langsam. Wenn Sie stattdessen mithilfe von IMAGEINFO und SCREENINFO/SCREENPTR die Speicheradresse selbst bestimmen und direkten Pointer-Zugriff verwenden, können Sie eine viel bessere Performance erzielen. Mit ASM ist eine noch bessere Geschwindigkeitssteigerung möglich.

Unterschiede zu QB:

- 'Puffer' ist neu in FreeBASIC.
- In allen Bildschirm-Modi ab 15-Bit Farbtiefe gibt FreeBASIC einen 32-Bit RGB-Wert zurück.

Siehe auch:
PSET, PMAP, COLOR (Anweisung), Grafik

Letzte Bearbeitung des Eintrags am 05.09.12 um 19:02:39
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

POINTER

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » P » **POINTER**
Syntax: ... AS Datentyp POINTER ...
Typ: Schlüsselwort
Kategorie: Pointer

POINTER wird mit DIM, REDIM, COMMON, STATIC, EXTERN, TYPE und DECLARE verwendet, um eine Variable oder Funktion als Pointer zu definieren. Es kann also immer angegeben werden, wenn ein Datentyp gebraucht wird.

Das Schlüsselwort POINTER wird meist mit PTR abgekürzt, was dieselbe Bedeutung hat.

Beispiele:

```
DECLARE FUNCTION foobar (parameter AS INTEGER) AS INTEGER POINTER

TYPE mytype
   a AS INTEGER
   b AS INTEGER POINTER
END TYPE

DIM variable AS STRING POINTER
DIM typ AS mytype POINTER
```

Unterschiede zu QB: neu in FreeBASIC

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht POINTER nicht zur Verfügung und kann nur über **__POINTER** aufgerufen werden.

Siehe auch:
PTR, ALLOCATE, Grundlagen zu Pointern, Zusammenstellung von Pointer-Anweisungen

Letzte Bearbeitung des Eintrags am 27.12.12 um 21:11:37
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

POKE

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » P » **POKE**
Typ: Anweisung
Syntax: POKE [Datentyp,] Adresse, Wert
Kategorie: Pointer

POKE schreibt einen Wert direkt in den RAM.

- 'Datentyp' ist der Datentyp des Wertes, der geschrieben werden soll; von 'Datentyp' hängt ab, wie viele Bytes geschrieben werden. Auch STRINGs und UDTs werden akzeptiert. Die zu schreibende Datenmenge richtet sich hier nach der Länge des STRINGs bzw. der Größe des UDTs. Wird 'Datentyp' ausgelassen, nimmt FreeBASIC automatisch UBYTE an.
- 'Adresse' ist ein Pointer auf die Speicherstelle im RAM, auf die geschrieben werden soll.
- 'Wert' ist der Wert, der an die angegebene Adresse geschrieben werden soll.

Mit PEEK und POKE wurden in älteren BASIC-Dialekten Pointer-Funktionen erstellt.

Beispiel: Ändern einer Variablen direkt über ihre Adresse.

```
DIM x AS BYTE, addr AS BYTE PTR
addr = VARPTR(x)
x = 5
PRINT x
POKE BYTE, addr, 10
PRINT x
SLEEP
```

Diese Form funktioniert einwandfrei. FreeBASIC hat jedoch eigene Pointer-Funktionen, die ein komfortableres Programmieren ermöglichen. Mit den FB-eigenen Funktionen würde dieses Beispiel so aussehen:

```
DIM x AS BYTE, addr AS BYTE PTR
addr = @x
x = 5
PRINT x
*addr = 10
PRINT x
SLEEP
```

Die Syntax

```
*Pointer = Ausdruck
```

setzt also den Wert, der an der Speicherstelle steht, auf die der Pointer zeigt. Wie viele Bytes geschrieben werden, hängt dabei vom Typ des Pointers ab (BYTE PTR, SINGLE PTR, ...)

Unterschiede zu QB:
Unter QB ist es nicht möglich, den Datentyp des zu schreibende Werts festzulegen; es wird immer ein UBYTE geschrieben.

Unterschiede zu früheren Versionen von FreeBASIC:

- Seit FreeBASIC v0.16 ist der Standard-Datentyp UBYTE; davor war es BYTE.
- Seit FreeBASIC v0.13 ist es möglich, STRINGs und UDTs zu schreiben.

Siehe auch:
PEEK, VARPTR, SADD, Pointer, Datentypen, Speicher

Letzte Bearbeitung des Eintrags am 27.12.12 um 21:11:51
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

POS

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » P » **POS**
Syntax: POS [(Zahl)]
Typ: Funktion
Kategorie: Konsole

POS gibt die horizontale Position des Cursors zurück. Der Rückgabewert 1 bedeutet, dass sich der Cursor ganz links befindet.

Der optionale Parameter 'Zahl' hat keine Bedeutung; er besteht nur aus Kompatibilitätsgründen zu QB.

Beispiel:

```
PRINT "Position:"; POS
PRINT "hello world";
PRINT "Position:"; POS
SLEEP
```

Ausgabe:

```
Position: 10
hello worldPosition: 21
```

Unterschiede zu QB:
In FreeBASIC ist der Parameter 'Zahl' optional und hat keine Auswirkung.

Unterschiede zu früheren Versionen von FreeBASIC:
Seit FreeBASIC v0.15 ist die Angabe des optionalen Parameters zulässig.

Siehe auch:
CSRLIN, LOCATE (Anweisung), LOCATE (Funktion), TAB, Konsole

Letzte Bearbeitung des Eintrags am 01.02.12 um 23:38:25
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

PRAGMA (Meta)

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » P » **PRAGMA (Meta)**
Syntax A: #PRAGMA Option = Wert
Syntax B: #PRAGMA PUSH(Option)
Syntax C: #PRAGMA POP(Option)
Typ: Metabefehl
Kategorie: Metabefehle

#PRAGMA ermöglicht es, die Compiler-Optionen im Code zu verändern.

- Syntax A verändert die Compiler-Optionen.
- Syntax B speichert die aktuelle Einstellung.
- Syntax C stellt die zuletzt gespeicherte Einstellung wieder her.

Gültige Werte für 'Option' und 'Wert':

Option	Wert	Bedeutung
msbitfield	0	verwende Bitfelder, die mit gcc kompatibel sind (Standard)
msbitfield	(ungleich Null)	verwende Bitfelder, die mit den Microsoft-C-Compilern kompatibel sind
once	N/A	veranlasst die Quelldatei, in der das Pragma verwendet wird, sich so zu verhalten, als ob sie mit #INCLUDE ONCE eingebunden wurde

Beispiel:

```
' Die aktuellen Einstellungen zwischenspeichern
"hlzeichen">(msbitfields)

' umschalten zu MSVC-kompatiblen Bitfields
"hlzeichen">=1

' Code, der MS-kompatible Bitfields benötigt...

' Original-Einstellung wiederherstellen
"hlzeichen">(msbitfields)
```

Unterschiede zu QB: neu in FreeBASIC

Siehe auch:
INCLUDE (Meta), Präprozessoren, Präprozessor-Anweisungen

Letzte Bearbeitung des Eintrags am 27.12.12 um 21:12:27
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

PRESERVE

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » P » **PRESERVE**
Syntax: REDIM PRESERVE array(...) AS Datentyp
Typ: Schlüsselwort
Kategorie: Deklaration

PRESERVE wird zusammen mit REDIM benutzt, um ein Array zu redimensionieren, ohne seine Elemente zu löschen. Wenn das Array vergrößert wird, werden die neuen Elemente mit 0 (bzw. mit Nullstrings) initiiert; die alten Elemente bleiben erhalten.

REDIM PRESERVE funktioniert nur bei DYNAMIC-Arrays; STATIC-Arrays können nicht redimensioniert werden. Ab der Compilerversion 0.17 wird das Meta-Schlüsselwort '$DYNAMIC jedoch nicht mehr akzeptiert. Arrays sind nur dynamisch, wenn sie mit REDIM oder ohne Dimensionsangabe erstellt wurden.

Wichtig: Das Schlüsselwort kann nur bei eindimensionalen Arrays ohne Probleme verwendet werden, bei mehrdimensionalen Arrays bleibt wegen der Speicherverwaltung bei Arrays nur der erste Index erhalten!

Beispiel:

```
REDIM array(1 TO 3) AS INTEGER
array(1) = 10
array(2) = 5
array(3) = 8

REDIM PRESERVE array(2 TO 9) AS INTEGER
FOR i AS INTEGER = 2 TO 9
   PRINT "array(" & i & ") = " & array(i)
NEXT
SLEEP
```

Dadurch wird das Element 1 auf das 2. Element des neuen Arrays verschoben.
Die Ausgabe sieht dann so aus:

```
array(2) = 10
array(3) = 5
array(4) = 8
array(5) = 0
array(6) = 0
array(7) = 0
array(8) = 0
array(9) = 0
```

Unterschiede zu QB:
PRESERVE wird erst in PDS 7.1 unterstützt.

Siehe auch:
DIM, REDIM, STATIC (Metabefehl), DYNAMIC (Metabefehl), Arrays

Letzte Bearbeitung des Eintrags am 16.06.13 um 23:00:43
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

PRESET (Grafik)

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » P » **PRESET (Grafik)**
Typ: Anweisung
Syntax: PRESET [Puffer], [STEP] (x, y), [Farbe]
Kategorie: Grafik

PRESET zeichnet einen einzelnen Pixel auf den Bildschirm oder in einen Bildpuffer.

- 'Puffer' ist ein Speicherbereich wie ein mit IMAGECREATE erstellter Puffer oder ein Array. Beide können mit PUT angezeigt werden. Wird 'Puffer' ausgelassen, zeichnet FreeBASIC direkt auf den Bildschirm.
- 'x' und 'y' sind die Koordinaten des Punktes im Grafikfenster bzw. auf dem Bildschirm.
- 'STEP' gibt an, dass die angegebenen Koordinaten relativ zur aktuellen Position des Grafikcursors sind.
- 'Farbe' ist das Farbattribut. Es ist abhängig von der Farbtiefe, die mit der letzten SCREENRES-Anweisung gewählt wurde. Wenn 'Farbe' ausgelassen wird, verwendet PRESET standardmäßig die Hintergrundfarbe, die von der letzten COLOR-Anweisung eingestellt wurde.

PRESET zeichnet einen einzelnen Pixel auf den Bildschirm. Die Position des Pixels ist abhängig von den letzten VIEW- und WINDOW-Anweisungen. PRESET funktioniert wie PSET (Grafik). Der einzige Unterschied besteht darin, dass, wenn das Argument 'Farbe' ausgelassen wird, die aktuelle Hintergrundfarbe verwendet wird.

Beispiel:

```
Screenres 400, 300
Color , 12            ' hellrote Hintergrundfarbe setzen
PReset (100,100)      ' Pixel in der Hintergrundfarbe zeichnen
PReset Step (50, 20)  ' Pixel relativ zu den letzten Koordinaten
Sleep
```

Unterschiede zu QB:
In FreeBASIC ist es möglich, in einen Datenpuffer zu zeichnen.

Siehe auch:
PRESET (Methode), PSET (Grafik), COLOR (Anweisung), POINT, Grafik

Letzte Bearbeitung des Eintrags am 16.06.13 um 23:02:49
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

PRESET (Methode)

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » P » **PRESET (Methode)**
Syntax: { PUT | DRAW STRING } [Puffer,] [STEP] (x, y), [weitere Angaben ...], PRESET
Typ: Schlüsselwort
Kategorie: Grafik

PRESET ist ein Schlüsselwort, das im Zusammenhang mit PUT (Grafik) und DRAW STRING eingesetzt wird.

Bei der Wiedergabe von Pixeldaten aus Bildpuffern funktioniert die PRESET-Methode fast genau so, wie die PSET-Methode, nur mit dem Unterschied, dass die Farbnummer der gezeichneten Pixel das Ergebnis eines logischen NOT der gespeicherten Daten sind.
Ausgegeben wird also ein invertiertes Bild, d.h. ein Bild mit umgekehrten Farben.

Beispiel:

```
ScreenRes 320, 200, 32
Line (0, 0)-(319, 199), RGB(0, 128, 255), bf

' Bild mit transparenter Hintergrundfarbe erstellen
' (die Transparenz wird bei PRESET nicht genutzt!)
Dim img As Any Ptr = ImageCreate( 33, 33, RGB(255, 0, 255) )
Circle img, (16, 16), 15, RGB(255, 255, 0),     ,     , 1, f
Circle img, (10, 10), 3,  RGB(  0,   0, 0),     ,     , 2, f
Circle img, (23, 10), 3,  RGB(  0,   0, 0),     ,     , 2, f
Circle img, (16, 18), 10, RGB(  0,   0, 0), 3.14, 6.28

Dim As Integer x = 160 - 16, y = 100 - 16

' Grafik ausgeben
Put (x, y), img, PReset

' Bildspeicher freigeben und auf Tastendruck warten
ImageDestroy img
Sleep
```

Siehe auch:
PRESET (Grafik), PUT (Grafik), DRAW STRING, SCREENRES, AND (Methode), OR (Methode), XOR (Methode), PSET (Methode), ALPHA, ADD, TRANS, CUSTOM, Grafik

Letzte Bearbeitung des Eintrags am 27.12.12 um 21:13:19
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

PRINT (Anweisung)

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » P » **PRINT (Anweisung)**
Syntax: PRINT Printausdruck
Typ: Anweisung
Kategorie: Konsole

PRINT gibt einen Text auf dem Bildschirm (bzw. auf der Konsole) an der aktuellen Textcursorposition aus.

Der Befehl lässt sich auf vielerlei Weise einsetzen. Der Funktionsumfang reicht von der Ausgabe einfacher STRING-Konstanten (PRINT "Hello World") oder Nummern (PRINT 1337) über die Ausgabe von Variablen (PRINT a) bis hin zur Ausgabe komplexer Ausdrücke (PRINT a + (b * c)). Um mehrere Ausdrücke auszugeben, trennen Sie diese durch Semikola (PRINT a; b), um sie direkt aneinander zu hängen, oder durch Kommata (PRINT a, b), um sie durch einen Tabspace zu trennen.

Beispiel 1:
Die Verwendung eines Semikolons ermöglicht die Ausgabe zweier Elemente mit einem Befehl.

```
DIM Benutzername AS STRING
Benutzername = "Linus"
PRINT "Name: "; Benutzername
SLEEP
```

Ausgabe:

```
Name: Linus
```

Beispiel 2:
Nach der Ausgabe des Ausdrucks wird der Textcursor um eine Zeile nach unten und in die erste Spalte dieser Zeile versetzt. Ein Semikolon hinter dem Ausdruck unterdrückt diesen Zeilenumbruch.

```
PRINT "Hallo ";
PRINT "Welt";
PRINT "!"
SLEEP
```

Ausgabe:

```
Hallo Welt!
```

Beispiel 3:
Kommata erfüllen dieselbe Funktion wie Semikola, fügen aber ein Tabspace ein.

```
DIM x AS INTEGER
x = 30
PRINT "x ist ",
PRINT x
PRINT "Hallo", "Welt", "auf Wiedersehen!"
SLEEP
```

Ausgabe:

```
x ist          30
Hallo         Welt          auf Wiedersehen!
```

Beispiel 4:
Wenn der Ausdruck nicht mit einem Semikolon oder einem Komma endet, wird ein Zeilenumbruch eingeleitet.

```
PRINT "Willkommen"
PRINT
PRINT "... und auf Wiedersehen!"
SLEEP
```

Ausgabe:

```
Willkommen

... und auf Wiedersehen
```

PRINT kann auch durch ein '?' ersetzt werden; der Compiler behandelt dieses Zeichen wie ein ganz normales PRINT. Sämtliche oben genannten Funktionen stehen auch mit diesem Shortcut zur Verfügung:

```
? "Hallo Welt!"
? "a:", a
? "b:",
? b
```

Escape-Characters
Einige Sonderzeichen, die nicht über die Tastatur eingegeben werden können, stehen über die Funktion CHR zur Verfügung. Beispielsweise führt ein CRLF (CHR(13) & CHR(10) bzw. CHR(13, 10), die EDV-Version eines Zeilenumbruchs) im Ausgabestring dazu, dass FreeBASIC bei der Ausgabe des Strings eine neue Zeile beginnt. Möglich wären hier auch die Schreibweise !"\r\n". Das Ausrufezeichen markiert den Beginn eines Strings mit Escape-Characters; ausführlichere Informationen finden Sie im Referenzartikel zum Ausrufezeichen. Beispiel 4 lässt sich damit auch folgendermaßen umsetzen:

```
PRINT !"Willkommen\r\n\r\n... und auf Wiedersehen!"
SLEEP
```

Hinweis: Die Zeichenkombination für einen Zeilenumbruch ist abhängig vom verwendeten Betriebssystem; siehe dazu den Wikipedia-Artikel zum Thema Zeilenumbruch.

Wenn nur Stringkonstanten verwendet werden, ist es nicht zwingend nötig, diese durch Semikola (;) bzw. Kommata (,) zu trennen; sie können einfach durch ein Leerzeichen getrennt aufgeführt werden und werden dann wie ein einziger String behandelt. Nutzvoll kann dies besonders bei der Verwendung von Escape-Characters sein:

Beispiel 5:

```
PRINT !"String mit \34Escape Chars\34" " und mit Literalen \34"
SLEEP
```

Ausgabe:

```
String mit "Escape Chars" und mit Literalen \34
```

Weiteres:
Der Aufruf von PRINT setzt den Wert unter ERR zurück.

Siehe auch:
PRINT "reflinkicon" href="temp0320.html">PRINT USING, WRITE (Anweisung), WRITE #, LOCATE (Anweisung), WIDTH (Anweisung), INPUT (Anweisung), Konsole

Letzte Bearbeitung des Eintrags am 05.09.12 um 19:48:31
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

PRINT (Datei)

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » P » **PRINT (Datei)**
Syntax: PRINT #Dateinummer, Printausdruck
Typ: Anweisung
Kategorie: Dateien

PRINT # schreibt Daten in eine Datei, die für sequentiellen Schreibzugriff (im OUTPUT- oder APPEND-Modus) geöffnet ist. Es kann damit jedoch auch in eine im BINARY-Modus geöffnete Datei geschrieben werden.

- 'Dateinummer' ist die Dateinummer, die durch OPEN zugewiesen wurde.
- 'Printausdruck' ist ein Ausdruck, für den dieselben Regeln gelten wie für PRINT (Anweisung).

An den Stellen, an denen bei PRINT ein Zeilenumbruch erzeugt wird, schreibt PRINT # ein CRLF (ein CHR(13) & CHR(10)). Dies entspricht einem Zeilenumbruch.

Hinweis: Die Zeichenkombination für einen Zeilenumbruch ist abhängig vom verwendeten Betriebssystem; siehe dazu den Wikipedia-Artikel zum Thema Zeilenumbruch.

Siehe auch:
PRINT (Anweisung), PRINT (Meta), PRINT USING, WRITE #, OPEN, INPUT #, LINE INPUT #, Dateien (Files)

Letzte Bearbeitung des Eintrags am 27.12.12 um 21:13:33
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

PRINT (Meta)

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » P » **PRINT (Meta)**
Syntax: #PRINT Text
Typ: Metabefehl
Kategorie: Metabefehle

Durch #PRINT gibt der Compiler einen Text aus.
'Text' ist eine beliebige Zeichenfolge, die nicht in Anführungszeichen stehen muss; der Text wird genau so ausgegeben, wie er im Quellcode steht. #PRINT kann zur Fehlerfindung während des Compilier-Vorgangs verwendet werden.

Beispiel:

```
"reflinkicon" href="temp0317.html">PRINT (Anweisung), #ERROR,
Präprozessor-Anweisungen
```

Letzte Bearbeitung des Eintrags am 27.12.12 um 21:13:51
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

PRINT USING

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » P » **PRINT USING**
Syntax: PRINT USING Formatierungsstring; [Variablen]
Typ: Anweisung
Kategorie: Konsole

PRINT USING gibt einen Text formatiert aus.

FORMAT bietet bessere Möglichkeiten, einen String zu formatieren; PRINT USING ist dafür unter jeder Plattform voll verfügbar.

- 'Formatierungsstring' ist ein STRING, der bestimmte Formatierungszeichen enthält.
- 'Variablen' sind Ausdrücke, die durch Semikola getrennt werden.

Werden keine Variablen angegeben, so werden alle Zeichen des Ausdrucks, die keine Formatierungszeichen sind, bis zum ersten Formatierungszeichen ausgegeben. Obwohl nicht unbedingt eine Variable vorhanden sein muss, muss das Semikolon (';') dennoch angegeben werden.

Folgende Formatierungszeichen sind möglich:

ZIFFERNFORMATIERUNG	
#	Platzhalter für eine anzuzeigende Ziffer. Hat die darzustellende Zahl weniger Ziffern, wird die Zahl rechtsbündig dargestellt und die Platzhalter links mit Leerzeichen aufgefüllt. Sind nicht genug # für den Ganzzahl-Anteil angegeben, wird die Zahl ohne Beachtung der Formatierungszeichen ausgegeben und ein % davor gesetzt.
,	Setzt die Position des Tausendertrennzeichens.
.	Setzt die Position des Dezimaltrennzeichens.
+	Reserviert einen Platz für das Vorzeichen der Zahl. Das Vorzeichen wird ausgegeben, auch wenn es positiv ist.
-	Wird dieses Zeichen hinter einen Ausdruck geschrieben, wird das Vorzeichen einer negativen Zahl hinter statt vor der Zahl angezeigt.
^^^^	Bewirkt, dass eine Zahl in Zehnerpotenzschreibweise (wissenschaftliche Notation) ausgedrückt wird. Zuerst muss mit # und . der eigentliche Zahlenbereich festgelegt werden. Die Anzahl der Potenzzeichen ^ legt die Anzahl der Zeichen für den Exponenten fest. Beachten Sie, dass auch das e+ bzw. e- zwei Zeichen braucht.
$$	Vor einer Zahl plaziert, bewirkt es, dass ein Dollarzeichen vor der Zahl angezeigt wird (und zusätzlich ein Leerzeichen vor dem Dollarzeichen).
**	Am Anfang einer Zahl angegeben, werden alle vorangestellten Leerzeichen durch Asteriske ('*') ersetzt.
**$	Kombiniert ** und $
&	Gibt eine Zahl mit der exakten Anzahl von Zeichen aus.
_	Schreibt das nachfolgende Zeichen als Literal aus; _# wird also als # ausgegeben. Um einen Unterstrich auszugeben, muss im Formatierungsstring ein doppelter Unterstrich stehen.
STRINGFORMATIERUNG	
!	Gibt das erste Zeichen des Strings aus.
\ \	Gibt n+2 Zeichen des Strings aus. n ist die Anzahl der Leerzeichen zwischen

	den Schrägstrichen. Folgen die beiden Schrägstriche direkt hintereinander, so werden die ersten beiden Zeichen des Strings ausgegeben.
&	Gibt den String komplett aus.

Alle anderen Zeichen des Ausdrucks werden unverändert ausgegeben.

Beispiel 1:

```
FOR exponent AS INTEGER = 1 TO 5
   PRINT USING "10 ^ "; exponent; 10 ^ exponent
NEXT
PRINT USING "\ \";"1234"
PRINT USING "_";-15.12345
PRINT USING "+"; -1234.5678
PRINT USING "& Text 2"; "Text 1"
SLEEP
```

Ausgabe:

```
10 ^ 1 =    10
10 ^ 2 =   100
10 ^ 3 =  1000
10 ^ 4 = 10000
10 ^ 5 = %100000
123
"hlkw0">DIM AS INTEGER anz1 = 3, anz2 = 1
DIM AS STRING ware1 = "Kuchen", ware2 = "Zucker"
DIM AS SINGLE preis1 = 2.5
PRINT USING "Ich bekomme & Stueck & zu je$$" & CHR(13, 10); _
            anz1, ware1, preis1, anz2, ware2
SLEEP
```

Ausgabe

```
Ich bekomme 3 Stueck Kuchen zu je $2.50.
Ich bekomme 1 Stueck Zucker zu je
```

Hinweis:
Intern verwendet der Befehl einen 2048 Byte großen Buffer. Sollte diese Grenze erreicht werden, wird die Ausgabe des Befehls abgeschnitten.

Siehe auch:
COLOR (Anweisung), FORMAT

Unterschiede zu QB:
QB erlaubt kein '&' zur Ausgabe von Zahlen.

Unterschiede zu früheren Versionen von FreeBASIC:

- Der Formatierungsausdruck darf seit FreeBASIC v0.24 ein WSTRING sein.
- Das Formatierungszeichen '&' zur Ausgabe der kompletten Zahl existiert seit FreeBASIC v0.21.

Siehe auch:
PRINT (Anweisung), PRINT (Datei), FORMAT, USING (Namespace), Konsole

Letzte Bearbeitung des Eintrags am 15.08.13 um 12:46:18
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

PRIVATE (Klausel)

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » P » **PRIVATE (Klausel)**
Syntax: PRIVATE { SUB | FUNCTION } Name (Parameterliste)
Typ: Klausel
Kategorie: Klassen

Siehe SUB/FUNCTION zu weiteren Details zur Syntax.

Ein Programm kann aus mehreren Modulen, d. h. aus mehreren BAS-Dateien bestehen.
Welche Dateien zum Projekt gehören sollen, wird dem Compiler in der Kommandozeile mitgeteilt:

```
fbc Optionen Dateiname1 Dateiname2 ...
```

Siehe dazu Der Compiler.

Aus einem bestimmten Modul können dabei immer alle Prozeduren aufgerufen werden, die sich in diesem befinden. Prozeduren aus anderen Modulen können nur dann aufgerufen werden, wenn diese Prozeduren PUBLIC sind. PRIVATE Prozeduren hingegen können nur aus dem Modul heraus aufgerufen werden, in dem sie sich befinden.

Es dürfen in mehreren Modulen mehrere Prozeduren mit demselben Bezeichner vorhanden sein, sofern diese PRIVATE sind.

Beispiel 1:
Ein Projekt besteht aus den Modulen A und B.
Das Modul A besitzt die Prozeduren Priv1 (PRIVATE), Priv2 (PRIVATE) und Pub1 (PUBLIC).
Das Modul B besitzt die Prozeduren Priv1 (PRIVATE), Priv2 (PRIVATE) und Pub2 (PUBLIC).

Wird gerade Code aus Modul A ausgeführt, so können Priv1 aus A, Priv2 aus A, Pub1 aus A und Pub2 aus B aufgerufen werden. Priv1 aus B und Priv2 aus B können nicht aufgerufen werden.
Wird gerade Code aus Modul B ausgeführt, so können Priv1 aus B, Priv2 aus B, Pub1 aus A und Pub2 aus B aufgerufen werden. Priv1 aus A und Priv2 aus A können nicht aufgerufen werden.

PRIVATE legt explizit für eine SUB/FUNCTION fest, dass diese PRIVATE sein soll, unabhängig vom gesetzten Standard (siehe OPTION; beachten Sie dazu auch die FB-Dialektformen). Die Klausel muss vor der SUB- bzw. FUNCTION-Anweisung stehen, darf jedoch nicht in der DECLARE-Anweisung stehen.

Beispiel 2:
Compilieren Sie diese Dateien mit folgender Kommandozeile:

```
fbc PrivatePublicTest1.bas PrivatePublicTest2.bas
```

```
' PrivatePublicTest1.bas
DECLARE SUB PrivateSub ()
DECLARE SUB PublicSub  ()

PublicSub
PrivateSub
SLEEP

PRIVATE SUB PrivateSub ()
  PRINT "PrivateSub in PrivatePublicTest1.bas"
END SUB
```

```
'--------------------------------------'

' PrivatePublicTest2.bas
DECLARE SUB PrivateSub ()
DECLARE SUB PublicSub  ()

PRIVATE SUB PrivateSub ()
  PRINT "PrivateSub in PrivatePublicTest2.bas"
END SUB

PUBLIC SUB PublicSub ()
  PRINT "PublicSub in PivatePublicTest2.bas
  PrivateSub
END SUB
```

Unterschiede zu QB: neu in FreeBASIC

Siehe auch:
PRIVATE (Schlüsselwort), PUBLIC (Klausel), OPTION, DECLARE, SUB, FUNCTION, Prozeduren

Letzte Bearbeitung des Eintrags am 27.12.12 um 21:14:27
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

PRIVATE (Schlüsselwort)

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » P » **PRIVATE (Schlüsselwort)**
Syntax: OPTION PRIVATE
Typ: Schlüsselwort
Kategorie: Klassen

OPTION PRIVATE bestimmt, dass SUBs und FUNCTIONs standardmäßig PRIVATE deklariert werden. Die Option kann **nur bis FreeBASIC v0.16** eingesetzt werden, oder in entsprechend höheren Versionen, die mit der Kommandozeilenoption -lang deprecated compiliert wurden! Wird mit FreeBASIC v0.17 unter der Option -lang fb compiliert, so ist OPTION PRIVATE nicht mehr zulässig! Geben Sie in diesem Fall explizit an, wenn eine Prozedur PRIVATE sein soll.
Siehe dazu auch die FB-Dialektformen.

Ein Programm kann aus mehreren Modulen, d. h. aus mehreren BAS-Dateien bestehen. Welche Dateien zum Projekt gehören sollen, wird dem Compiler in der Kommandozeile mitgeteilt:

```
fbc Optionen Dateiname1 Dateiname2 ...
```

Siehe dazu Der Compiler.

Aus einem bestimmten Modul können dabei immer alle Prozeduren aufgerufen werden, die sich in diesem befinden. Prozeduren aus anderen Modulen können nur dann aufgerufen werden, wenn diese Prozeduren PUBLIC sind. PRIVATE Prozeduren hingegen können nur aus dem Modul heraus aufgerufen werden, in dem sie sich befinden.

Es dürfen in mehreren Modulen mehrere Prozeduren mit demselben Bezeichner vorhanden sein, sofern diese PRIVATE sind.

Beispiel:
Ein Projekt besteht aus den Modulen A und B.
Das Modul A besitzt die Prozeduren Priv1 (PRIVATE), Priv2 (PRIVATE) und Pub1 (PUBLIC).
Das Modul B besitzt die Prozeduren Priv1 (PRIVATE), Priv2 (PRIVATE) und Pub2 (PUBLIC).
Wird gerade Code aus Modul A ausgeführt, so können Priv1 aus A, Priv2 aus A, Pub1 aus A und Pub2 aus B aufgerufen werden. Priv1 aus B und Priv2 aus B können nicht aufgerufen werden.
Wird gerade Code aus Modul B ausgeführt, so können Priv1 aus B, Priv2 aus B, Pub1 aus A und Pub2 aus B aufgerufen werden. Priv1 aus A und Priv2 aus A können nicht aufgerufen werden.

OPTION PRIVATE gibt an, dass SUBs und FUNCTIONs standardmäßig PRIVATE sein sollen. Wenn Sie OPTION PRIVATE verwenden, müssen Sie explizit die Klausel PUBLIC verwenden, wenn eine Prozedur PUBLIC sein soll.

Beispiel:
Compilieren Sie diese Dateien mit folgender Programmzeile:

```
fbc PrivatePublicTest1.bas PrivatePublicTest2.bas
```

```
' PrivatePublicTest1.bas
"hlstring">"deprecated"
OPTION PRIVATE

DECLARE SUB PrivateSub ()
DECLARE SUB PublicSub  ()
```

```
PublicSub
PrivateSub
SLEEP

SUB PrivateSub ()
  PRINT "PrivateSub in PrivatePublicTest1.bas"
END SUB

'----------------------------------------'

' PrivatePublicTest2.bas
"hlstring">"deprecated"
OPTION PRIVATE
DECLARE SUB PrivateSub ()
DECLARE SUB PublicSub  ()

SUB PrivateSub ()
  PRINT "PrivateSub in PrivatePublicTest2.bas"
END SUB

PUBLIC SUB PublicSub ()
  PRINT "PublicSub in PivatePublicTest2.bas
  PrivateSub
END SUB
```

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC:
Die Option ist nur bis FreeBASIC v0.16 erlaubt. Seit FreeBASIC v0.17 ist diese Option nur noch zulässig, wenn mit der Kommandozeilenoption -lang deprecated compiliert wurde.

Siehe auch:
PRIVATE, PUBLIC, __FB_OPTION_PRIVATE__OPTION, DECLARE, SUB, FUNCTION, Verschiedenes

Letzte Bearbeitung des Eintrags am 19.01.13 um 16:50:00
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

PRIVATE (UDT)

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » P » **PRIVATE (UDT)**
Syntax:

```
TYPE TypeName
   PRIVATE:
   ' Deklaration privater Records

END TYPE
```

Typ: Schlüsselwort
Kategorie: Klassen

PRIVATE legt fest, dass die folgenden Deklarationen privat sein sollen, d. h. dass ein Zugriff nur von UDT-eigenen Prozeduren aus zulässig ist. Das Schlüsselwort ist in dieser Form nur dann zulässig, wenn mit der Kommandozeilenoption -lang fb compiliert wurde (Standard).

Mit PRIVATE ist es möglich, den Zugriff auf UDT-Records einzuschränken. Der Zweck liegt darin, dass der Zugriff auf die entsprechenden Records nur noch über benutzerdefinierte Prozeduren ermöglicht wird, die eventuelle Fehler schon beim Zugriffsversuch abfangen, den Wert in ein besser verwertbares Format umwandeln, etc.

Der UDT wird regulär definiert. Innerhalb der Typen-Deklaration wird die Zeile

```
PRIVATE:
```

eingefügt, sobald die folgenden Records privat sein sollen, d. h. wenn der Zugriff darauf eingeschränkt werden soll. PRIVATE gilt innerhalb des UDTs so lange, bis es durch das Schlüsselwort PUBLIC oder PROTECTED abgelöst wird.
Standardmäßig sind alle Records PUBLIC. Dies ist auch dann der Fall, wenn eine vorhergehende Typendeklaration mit PRIVATE oder PROTECTED abgeschlossen wurde.

Auf PUBLIC-Records kann aus jeder Programmsituation heraus zugegriffen werden; auf PRIVATE-Records dürfen nur UDT-eigene Prozeduren (SUBs, FUNCTIONs, PROPERTYs, OPERATORs, Klassen-Konstruktoren und Klassen-Destruktoren) zugreifen. PROTECTED-Records verhalten sich wie PRIVATE-Records, jedoch können auf sie auch Prozeduren zugreifen, die sich in UDTs befinden, welche vom Basis-UDT erben (siehe EXTENDS).
Beachten Sie, dass auch die Prozeduren eines UDTs PUBLIC, PROTECTED oder PRIVATE sein können.
Ein Zugriff auf einen privaten Record von außerhalb einer UDT-eigenen Prozedur führt zu der Compiler-Fehlermeldung:
Illegal member access

Es ist erlaubt, innerhalb eines UDTs beliebig viele PUBLIC-, PRIVATE- und PROTECTED-Blöcke einzurichten.

Beispiel:

```
Type SecureIntArray
  Private:
  _ubound As Integer
  _lbound As Integer
  _data   As Integer Ptr
```

```
  Public:
  Declare Property datas (index As Integer ) As Integer
  Declare Property datas (index As Integer, new_value As Integer)

  Declare Sub Redim (low As Integer, up As Integer)

  Declare Constructor ()
  Declare Destructor  ()
End Type

Constructor SecureIntArray()
  With This
    ._ubound = 0
    ._lbound = 0
    ._data   = Allocate(4)
  End With
End Constructor

Destructor SecureIntArray()
  DeAllocate This._data
End Destructor

Property SecureIntArray.datas (index As Integer) As Integer
  With This
    If (index < ._lbound) Or (index > ._ubound) Then Return 0

    Return ._data&"hlzahl">._lbound]
  End With
End Property

Property SecureIntArray.datas (index As Integer, new_value As Integer)
  With This
    If (index < ._lbound) Or (index > ._ubound) Then Return

    ._data&"hlzahl">._lbound] = new_value
  End With
End Property

Sub SecureIntArray.Redim (low As Integer, up As Integer)
  If low > up Then Return

  With This
    DeAllocate ._data

    ._ubound = up
    ._lbound = low
    ._data   = Allocate( (up - low + 1) * 4 )

  End With
End Sub

Dim As SecureIntArray mySecInt
Dim As Integer i
```

```
mySecInt.Redim 0, 20              ' Wir erstellen ein Array mit den
Indizes 0 bis 20...
For i = 0 To 24                   ' und versuchen, bis zum Index 24 Werte
einzufügen.
  mySecInt.datas(i) = 2 * i       ' Die Property wird diesen Versuch
abblocken.
Next

For i = 0 To 24                   ' Genauso, wie auch die Abfrage von
Werten außerhalb der Array-Dimensionen fehlschlägt.
  Print i, mySecInt.datas(i)
Next

Sleep

' Print mySecInt._data             ' Der Pointer ist nicht verfügbar, da
es sich um ein PRIVATE-Feld handelt.
```

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC:
PRIVATE bei UDTs existiert seit FreeBASIC v0.17.

Siehe auch:
PRIVATE, PUBLIC (UDT), PROTECTED, TYPE (UDT), THIS, EXTENDS, Objektorientierung

Letzte Bearbeitung des Eintrags am 16.02.13 um 12:30:11
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

PROCPTR

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » P » **PROCPTR**
Syntax: PROCPTR (Prozedurname)
Typ: Funktion
Kategorie: Pointer

PROCPTR gibt die Adresse einer Prozedur (SUB oder FUNCTION) im Speicher zurück. Diese Funktion lässt sich auch durch die Form

```
@Prozedurname
```

ersetzen.

Beispiel 1:

```
DECLARE SUB dummy

PRINT PROCPTR(dummy)
PRINT @dummy
SLEEP

SUB dummy
    ' ...
END SUB
```

Beispiel 2:

```
Declare Function Addieren(x As Integer, y As Integer) As Integer
Declare Function Subtrahieren(x As Integer, y As Integer) As Integer
Dim myFunction As Function(x As Integer, y As Integer) As Integer

'Verbindet den Funktionspointer mit der Funktion 'Addieren'
myFunction = @Addieren
Print myFunction(2, 3)

'Verbindet den Funktionspointer mit der Funktion 'Subtrahieren'
myFunction = ProcPtr(Subtrahieren)
Print myFunction(2, 3)

Sleep

Function Addieren(x As Integer, y As Integer) As Integer
    Return x + y
End Function

Function Subtrahieren(x As Integer, y As Integer) As Integer
    Return x - y
End Function
```

Unterschiede zu QB: neu in FreeBASIC

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht PROCPTR nicht zur Verfügung und kann nur über **__PROCPTR**

aufgerufen werden.

Siehe auch:
@, SADD, Grundlagen zu Pointern, Zusammenstellung von Pointer-Anweisungen

Letzte Bearbeitung des Eintrags am 17.06.13 um 00:02:04
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

PROPERTY

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » P » **PROPERTY**
Syntax A:

```
TYPE TypeName
    FeldName [(Indizes)] AS DatenTyp

    ' Wert abfragen
    DECLARE PROPERTY PropertyName [( [ BYREF | BYVAL ] Index AS
DatenTyp)] AS DatenTyp

    ' Wert setzen
    DECLARE PROPERTY PropertyName ([ [ BYREF | BYVAL ] Index AS DatenTyp,
] _
                                   [ BYREF | BYVAL ] NeuerWert AS
DatenTyp )
END TYPE
```

Syntax B:

```
PROPERTY TypeName.PropertyName [ (Parameterliste) ] [ AS DatenTyp ]
   ' Anweisungen
END PROPERTY
```

Typ: Anweisung
Kategorie: Klassen

PROPERTY erstellt eine Property einer Klasse. Dieses Schlüsselwort ist nur zulässig, wenn mit der Kommandozeilenoption -lang fb compilliert wird.

Eine Property kann mit einem UDT-Record gleichgesetzt werden; jedoch wird bei jedem Zugriff auf dieses Record ein benutzerdefiniertes Unterprogramm aufgerufen. Informieren Sie sich zuerst über Prozeduren in FreeBASIC, bevor Sie sich mit dem Thema Properties und Klassen befassen. Siehe dazu auch SUB, FUNCTION.
Ebenso nötig sind Kenntnisse über UDTs.

- 'TypeName' ist der Name eines UDTs, wie er unter TYPE (UDT) erklärt wird.
- 'Indizes' zeigt an, dass auch Arrays verwendet werden können; wie üblich ist die Anzahl der Dimensionen in diesem Fall aber fest.
- 'FeldName' ist der Name eines UTD-Records. Mindestens ein solcher Record muss existieren, um eine Property einzufügen. Es dürfen selbstverständlich auch mehrere Records verwendet werden.
- 'PropertyName' ist der Name der Property; dieser Bezeichner wird auch verwendet, um das Unterprogramm zu identifizieren, das beim Zugriff auf die Property aufgerufen wird. 'PropertyName' darf auch überladen werden, d. h. es dürfen mehrere Property-Prozeduren mit demselben Bezeichner existieren, die sich dann durch ihre Parameterliste unterscheiden müssen. Überladene Properties sind z. B. dann nötig wenn der Wert einer Property vom User gesetzt und abfragt werden soll.
- 'NeuerWert' ist ein Wert, der an die Speicherstelle der Property übergeben werden soll. Er wird an das Unterprogramm übergeben und dort abhängig von den 'Anweisungen' bearbeitet.
- 'Index' wird z. B. verwendet, wenn der über die Property geänderte Record ein Array ist. Über diesen Parameter kann der Index des Feldelements angegeben werden. Es ist auch möglich, hier andere Parameter zu übergeben, z. B. einen Pointer auf eine ganze Struktur von Parametern.
- 'Parameterliste' ist eine Parameterliste, die den Parametern in Syntax A entspricht.

- 'Anweisungen' ist ein Programmcode, der den Regeln einer FUNCTION folgt. Ein Rückgabewert kann mit PROPERTY, 'TypeName.PropertyName' und RETURN gesetzt werden.

PROPERTY-Felder werden dazu genutzt, um auf Werte innerhalb eines UDTs zuzugreifen und bei jedem Zugriff eine bestimmte Prozedur aufzurufen.
So, wie sie den Wert eines UDT-Records setzen und abfragen können, können auch Properties gesetzt und abgefragt werden. Die zugehörigen Prozeduren haben dabei denselben Bezeichner; sie unterscheiden sich lediglich durch die Anzahl der Parameter.

Innerhalb der Property kann auf die übrigen Records des zugeordneten UDTs über das Schlüsselwort THIS zugegriffen werden.

Bei der Bearbeitung der Records des UDTs mit mathematischen Operatoren (+, -, ...) kann das Prinzip "Operator Overloading" angewandt werden; siehe hierzu OPERATOR.

Beispiel: Abfragen und Setzen von Werten eines UDTs mittels Properties

```
Type Vector2D
  As Single x, y
  Declare Operator Cast() As String
  Declare Property Length() As Single
  Declare Property Length( ByVal new_length As Single )
End Type

Operator Vector2D.CAST () As String
  Return "(" & x & ", " & y & ")"
End Operator

Property Vector2D.Length() As Single
  Length = Sqr( x * x + y * y )
End Property

Property Vector2D.Length( ByVal new_length As Single )
  Dim m As Single = Length
  If m <> 0 Then
    '' neuer_Vektor = alter_Vektor / laenge * neue_laenge
    x *= new_length / m
    y *= new_length / m
  End If
End Property

Dim a As Vector2D = ( 3, 4 )

Print "a = "; a
Print "a.Length = "; a.Length
Print

a.Length = 10

Print "a = "; a
Print "aLength = "; a.Length

Sleep
```

Ausgabe:

```
a = (3, 4)
a.Length =  5

a = (6, 8)
a.Length =  10
```

Mittels der Schlüsselwörter PRIVATE und PUBLIC kann festgelegt werden, von welchen Programmpunkten aus auf bestimmte Records des UDTs zugegriffen werden darf. Damit kann erreicht werden, dass bestimmte Records nicht direkt, sondern nur über die bereitgestellten Properties gesetzt oder ausgelesen werden können. Die Property kann dann beispielsweise Zugriffsbeschränkungen wie Arraygrenzen überprüfen. Siehe dazu auch das Codebeispiel Zugriffskontrolle über Properties.

Unterschiede zu QB:
Properties und Methoden existieren nur in VisualBASIC, jedoch existiert dort keine Möglichkeit, solche selbst zu erstellen.

Unterschiede zu früheren Versionen von FreeBASIC: existiert seit FreeBASIC v0.17

Siehe auch:
TYPE (UDT), SUB, FUNCTION, OPERATOR, THIS, PUBLIC, PRIVATE, Objektorientierung

Letzte Bearbeitung des Eintrags am 19.01.13 um 16:50:39
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

PROTECTED

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » P » **PROTECTED**
Syntax:

```
TYPE TypeName
   PROTECTED:
   ' Deklaration geschützter Records

END TYPE
```

Typ: Schlüsselwort
Kategorie: Klassen

PROTECTED legt fest, dass die folgenden Deklarationen geschützt sein sollen, d. h. dass ein Zugriff nur von UDT-eigenen Prozeduren aus zulässig ist. Das Schlüsselwort ist in dieser Form nur dann zulässig, wenn mit der Kommandozeilenoption -lang fb compiliert wurde (Standard).

Mit PROTECTED ist es möglich, den Zugriff auf UDT-Records einzuschränken. Der Zweck liegt darin, dass der Zugriff auf die entsprechenden Records nur noch über benutzerdefinierte Prozeduren ermöglicht wird, die eventuelle Fehler schon beim Zugriffsversuch abfangen, den Wert in ein besser verwertbares Format umwandeln, etc.

Der UDT wird regulär definiert. Innerhalb der Typen-Deklaration wird die Zeile

```
PROTECTED:
```

eingefügt, sobald ein folgende Deklarationen geschützt sein soll, d. h. wenn der Zugriff darauf eingeschränkt werden soll. PROTECTED gilt innerhalb des UDTs so lange, bis es durch das Schlüsselwort PUBLIC oder PRIVATE abgelöst wird.
Standardmäßig sind alle Records PUBLIC. Dies ist auch dann der Fall, wenn eine vorhergehende Typendeklaration mit PRIVATE oder PROTECTED abgeschlossen wurde.

Auf PUBLIC-Records kann aus jeder Programmsituation heraus zugegriffen werden; auf PRIVATE-Records dürfen nur UDT-eigene Prozeduren (SUBs, FUNCTIONs, PROPERTYs, OPERATORs, Klassen-Konstruktoren und Klassen-Destruktoren) zugreifen. PROTECTED-Records verhalten sich wie PRIVATE-Records, jedoch können auf sie auch Prozeduren zugreifen, die sich in UDTs befinden, welche vom Basis-UDT erben (siehe EXTENDS).
Beachten Sie, dass auch die Prozeduren eines UDTs PUBLIC, PROTECTED oder PRIVATE sein können.
Ein Zugriff auf einen geschützten Record von außerhalb einer UDT-eigenen Prozedur führt zu der Compiler-Fehlermeldung
Illegal member access

Es ist erlaubt, innerhalb eines UDTs beliebig viele PUBLIC-, PRIVATE- und PROTECTED-Blöcke einzurichten.

Beispiel:

```
Type mutter
  Private:
  x As Integer = 1

  Protected:
  y As Integer = 2
```

```
  Public:
  z As Integer = 3
End Type

Type kind Extends mutter
  dummy As Integer
  Declare Sub ausgeben()
End Type

Sub kind.ausgeben
  ' Print x  ' funktioniert nicht, da x PRIVATE für die Mutter-Klasse ist
  Print y
  Print z
End Sub

Dim As kind einKind
einKind.ausgeben
Sleep
```

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC:

- Vererbung existiert seit FreeBASIC v0.24.0. Davor hatte PROTECTED keine eigene Bedeutung.
- In FreeBASIC v0.23 verhielt sich PROTECTED genauso wie PRIVATE.
- Vor FreeBASIC v0.23 hatte PROTECTED keine Funktion

Unterschiede unter den FB-Dialektformen: nur in der Dialektform -lang fb verfügbar

Siehe auch:
PRIVATE (UDT), PUBLIC (UDT), TYPE (UDT), THIS, EXTENDS, Objektorientierung

Letzte Bearbeitung des Eintrags am 16.02.13 um 12:26:59
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

PSET (Grafik)

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » P » **PSET (Grafik)**
Syntax: PSET [Puffer], [STEP] (x, y)[, Farbe]
Typ: Anweisung
Kategorie: Grafik

PSET zeichnet einen einzelnen Pixel auf den Bildschirm oder in einen Bildpuffer.

- 'Puffer' ist ein Speicherbereich wie ein mit IMAGECREATE erstellter Puffer oder ein Array. Beide können mit PUT angezeigt werden. Wird 'Puffer' ausgelassen, zeichnet FreeBASIC direkt auf den Bildschirm.
- 'x' und 'y' sind die Koordinaten des Punktes im Grafikfenster bzw. auf dem Bildschirm.
- 'STEP' gibt an, dass die angegebenen Koordinaten relativ zur aktuellen Position des Grafikcursors sind.
- 'Farbe' ist das Farbattribut. Es ist abhängig von der Farbtiefe, die mit der letzten SCREENRES-Anweisung gewählt wurde. Wenn 'Farbe' ausgelassen wird, verwendet PSET standardmäßig die Vordergrundfarbe, die von der letzten COLOR-Anweisung eingestellt wurde.

PSET zeichnet einen einzelnen Pixel auf den Bildschirm. Die Position des Pixels ist abhängig von den letzten VIEW- und WINDOW-Anweisungen.

Beispiel:

```
SCREENRES 640, 480

PSET (100, 100), 15                ' weißen Punkt an den Koordinaten (100,
100) zeichnen
SLEEP                             '  Warte auf Tastendruck

CLS                                ' Bildschirm löschen
WINDOW SCREEN (0, 0)-(63.9, 47.9) ' physischen Darstellungsbereich ändern
PSET (10, 10), 1                   ' blauen Punkt an den Koordinaten (10,
10) zeichnen
                                   ' Entspricht bei diesem
Darstellungsbereich den
                                   ' Koordinaten (100, 100)
SLEEP

CLS
WINDOW                             ' physischen Darstellungsbereich
zurücksetzen
VIEW (50, 50)-(590, 430)           ' Gültigkeitsbereich neu festlegen
PSET (50, 50), 2                   ' grünen Punkt an den Koordinaten (50,
50) zeichnen.
                                   ' Entspricht den Koordinaten (100, 100)
im Gesamtfenster
                                   ' (wenn der Gültigkeitsbereich
ignoriert wird.)
SLEEP
```

Hinweis: PSET und seine Gegenfunktion POINT sind aufgrund der internen Berechnungen und Prüfungen sehr langsam. Wenn Sie stattdessen mithilfe von IMAGEINFO und SCREENINFO/SCREENPTR die Speicheradresse selbst bestimmen und direkten Pointer-Zugriff verwenden, können Sie eine viel bessere

Performance erzielen. Mit ASM ist eine noch bessere Geschwindigkeitssteigerung möglich.

Unterschiede zu QB:

- In FreeBASIC ist es möglich, in einen Datenpuffer zu zeichnen.
- Im 16- und 32-Bit-Modus werden in FreeBASIC Farbwerte als 32-Bit Wert benötigt statt einem 8-Bit Paletten-Index.

Siehe auch:
PSET (Methode), PRESET (Grafik), COLOR (Anweisung), POINT, Grafik

Letzte Bearbeitung des Eintrags am 10.09.12 um 01:40:54
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

PSET (Methode)

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » P » **PSET (Methode)**
Syntax: { PUT | DRAW STRING } [Puffer,] [STEP] (x, y), [weitere Angaben ...], PSET
Typ: Schlüsselwort
Kategorie: Grafik

PSET ist ein Schlüsselwort, das im Zusammenhang mit PUT (Grafik) und DRAW STRING eingesetzt wird.

Bei der Darstellung von Pixeldaten aus einem Bildpuffer nach der PSET-Methode werden die Daten 1:1 auf dem Bildschirm wiedergegeben; die Darstellung erfolgt unabhängig von Pixeln in der Transparenzfarbe oder vom überzeichneten Hintergrund.

Beispiel:

```
ScreenRes 320, 200, 32
Line (0, 0)-(319, 199), RGB(0, 128, 255), bf

' Bild mit transparenter Hintergrundfarbe erstellen
' (die Transparenz wird bei PSET nicht genutzt!)
Dim img As Any Ptr = ImageCreate( 33, 33, RGB(255, 0, 255) )
Circle img, (16, 16), 15, RGB(255, 255, 0),     ,     , 1, f
Circle img, (10, 10), 3,  RGB(  0,   0, 0),     ,     , 2, f
Circle img, (23, 10), 3,  RGB(  0,   0, 0),     ,     , 2, f
Circle img, (16, 18), 10, RGB(  0,   0, 0), 3.14, 6.28

Dim As Integer x = 160 - 16, y = 100 - 16

' Grafik ausgeben
Put (x, y), img, Pset

' Bildspeicher freigeben und auf Tastendruck warten
ImageDestroy img
Sleep
```

Siehe auch:
PSET (Grafik), PUT (Grafik), DRAW STRING, SCREENRES, AND (Methode), OR (Methode), XOR (Methode), PRESET (Methode), ALPHA, ADD, TRANS, CUSTOM, Grafik

Letzte Bearbeitung des Eintrags am 10.09.12 um 01:43:43
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

PTR

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » P » **PTR**
Syntax: ... AS Datentyp PTR ...
Typ: Schlüsselwort
Kategorie: Pointer

PTR wird mit DIM, REDIM, COMMON, STATIC, EXTERN, TYPE und DECLARE verwendet, um eine Variable oder Funktion als Pointer zu definieren. Es kann also immer angegeben werden, wenn ein Datentyp gebraucht wird.

PTR ist eine Abkürzung des Schlüsselwortes POINTER und hat dieselbe Bedeutung.

Beispiele:

```
DECLARE FUNCTION foobar (Parameter AS INTEGER) AS INTEGER PTR

TYPE mytype
a AS INTEGER
b AS INTEGER PTR
END TYPE

DIM var AS STRING PTR
DIM typ AS mytype PTR
```

Unterschiede zu QB: neu in FreeBASIC

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht PTR nicht zur Verfügung und kann nur über **__PTR** aufgerufen werden.

Siehe auch:
ALLOCATE, Grundlagen zu Pointern, Zusammenstellung von Pointer-Anweisungen

Letzte Bearbeitung des Eintrags am 27.12.12 um 21:15:40
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

PUBLIC (Klausel)

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » P » **PUBLIC (Klausel)**
Syntax: PUBLIC { SUB | FUNCTION } Name (Parameterliste)
Typ: Klausel
Kategorie: Klassen

Siehe SUB/FUNCTION zu weiteren Details zur Syntax.

Ein Programm kann aus mehreren Modulen, d. h. aus mehreren BAS-Dateien bestehen.
Welche Dateien zum Projekt gehören sollen, wird dem Compiler in der Kommandozeile mitgeteilt:

```
fbc Optionen Dateiname1 Dateiname2 ...
```

Siehe dazu Der Compiler.

Aus einem bestimmten Modul können dabei immer alle Prozeduren aufgerufen werden, die sich in diesem befinden. Prozeduren aus anderen Modulen können nur dann aufgerufen werden, wenn diese Prozeduren PUBLIC sind. PRIVATE Prozeduren hingegen können nur aus dem Modul heraus aufgerufen werden, in dem sie sich befinden.

Es dürfen in mehreren Modulen mehrere Prozeduren mit demselben Bezeichner vorhanden sein, sofern diese PRIVATE sind.

Beispiel 1:
Ein Projekt besteht aus den Modulen A und B.
Das Modul A besitzt die Prozeduren Priv1 (PRIVATE), Priv2 (PRIVATE) und Pub1 (PUBLIC).
Das Modul B besitzt die Prozeduren Priv1 (PRIVATE), Priv2 (PRIVATE) und Pub2 (PUBLIC).

Wird gerade Code aus Modul A ausgeführt, so können Priv1 aus A, Priv2 aus A, Pub1 aus A und Pub2 aus B aufgerufen werden. Priv1 aus B und Priv2 aus B können nicht aufgerufen werden.
Wird gerade Code aus Modul B ausgeführt, so können Priv1 aus B, Priv2 aus B, Pub1 aus A und Pub2 aus B aufgerufen werden. Priv1 aus A und Priv2 aus A können nicht aufgerufen werden.

Standardmäßig sind alle Prozeduren PUBLIC, d.h. es muss nicht für jede SUB/FUNCTION explizit eine PUBLIC-Klausel benutzt werden. Jedoch kann mit OPTION PRIVATE festgelegt werden, dass Prozeduren standardmäßig PRIVATE sind. (Beachten Sie dazu auch die FB-Dialektformen.)

Das PUBLIC-Schlüsselwort muss nur im Prozedurheader stehen, nicht in der DECLARE-Anweisung.

Beispiel 2:
Compilieren Sie diese Dateien mit folgender Kommandozeile:

```
fbc PrivatePublicTest1.bas PrivatePublicTest2.bas
```

```
' PrivatePublicTest1.bas
DECLARE SUB PrivateSub ()
DECLARE SUB PublicSub  ()

PublicSub
PrivateSub
SLEEP

PRIVATE SUB PrivateSub ()
```

```
  PRINT "PrivateSub in PrivatePublicTest1.bas"
END SUB

'-------------------------------------------------------'

' PrivatePublicTest2.bas
DECLARE SUB PrivateSub ()
DECLARE SUB PublicSub  ()

PRIVATE SUB PrivateSub ()
  PRINT "PrivateSub in PrivatePublicTest2.bas"
END SUB

PUBLIC SUB PublicSub ()
  PRINT "PublicSub in PivatePublicTest2.bas
  PrivateSub
END SUB
```

Ausgabe:

```
PublicSub in PrivatePublicTest2.bas
PrivateSub in PrivatePublicTest2.bas
PrivateSub in PrivatePublicTest1.bas
```

Wie Sie sehen, dürfen zwei Prozeduren mit demselben Bezeichner existieren, wenn beide PRIVATE sind.

Unterschiede zu QB:
PUBLIC existiert nur in VisualBASIC und kann dort nur eingeschränkt mit TYPE verwendet werden.

Siehe auch:
PUBLIC, PRIVATE (Klausel), OPTION, DECLARE, SUB, FUNCTION, Prozeduren

Letzte Bearbeitung des Eintrags am 27.12.12 um 21:15:57
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

PUBLIC (UDT)

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » P » **PUBLIC (UDT)**
Syntax:

```
TYPE TypeName
   PUBLIC:
   ' Deklaration öffentlicher Records

END TYPE
```

Typ: Schlüsselwort
Kategorie: Klassen

PUBLIC legt fest, dass folgende UDT-Records öffentlich sein sollen, d. h. dass ein Zugriff von jeder Prozedur aus zulässig sein soll. Das Schlüsselwort ist in dieser Form nur dann zulässig, wenn mit der Kommandozeilenoption -lang fb wird.

Mit PRIVATE und PROTECTED ist es möglich, den Zugriff auf UDT-Records einzuschränken. Der Zweck liegt darin, dass der Zugriff auf die entsprechenden Records nur noch über benutzerdefinierte Prozeduren ermöglicht wird, die eventuelle Fehler schon beim Zugriffsversuch abfangen, den Wert in ein besser verwertbares Format umwandeln, etc. PUBLIC hebt diese Einschränkung auf.

Der UDT wird regulär definiert. Innerhalb der Typen-Deklaration wird die Zeile

```
PUBLIC:
```

eingefügt, sobald die folgenden Records öffentlich sein soll, d. h. wenn der Zugriff von jedem Punkt aus erlaubt sein soll. PUBLIC gilt innerhalb des UDTs so lange, bis es durch das Schlüsselwort PRIVATE oder PROTECTED abgelöst wird.
Standardmäßig sind alle Records PUBLIC. Dies ist auch dann der Fall, wenn eine vorhergehende Typendeklaration mit PRIVATE oder PROTECTED abgeschlossen wurde.

Auf PUBLIC-Records kann aus jeder Programmsituation heraus zugegriffen werden; auf PRIVATE-Records dürfen nur UDT-eigene Prozeduren (SUBs, FUNCTIONs, PROPERTYs, OPERATORs, Klassen-Konstruktoren und Klassen-Destruktoren) zugreifen. PROTECTED-Records verhalten sich wie PRIVATE-Records, jedoch können auf sie auch Prozeduren zugreifen, die sich in UDTs befinden, welche vom Basis-UDT erben (siehe EXTENDS).
Beachten Sie, dass auch die Prozeduren eines UDTs PUBLIC, PROTECTED oder PRIVATE sein können.
Ein Zugriff auf einen privaten Record von außerhalb einer UDT-eigenen Prozedur führt zu der Compiler-Fehlermeldung:
Illegal member access

Es ist erlaubt, innerhalb eines UDTs beliebig viele PUBLIC-, PRIVATE- und PROTECTED-Blöcke einzurichten.

Beispiel:

```
Type SecureIntArray
  Private:
  _ubound As Integer
  _lbound As Integer
  _data   As Integer Ptr
```

```
  Public:
  Declare Property datas (index As Integer                      ) As
Integer
  Declare Property datas (index As Integer, new_value As Integer)

  Declare Sub Redim (low As Integer, up As Integer)

  Declare Constructor ()
  Declare Destructor  ()
End Type

Constructor SecureIntArray()
  With This
    ._ubound = 0
    ._lbound = 0
    ._data   = Allocate(4)
  End With
End Constructor

Destructor SecureIntArray()
  DeAllocate This._data
End Destructor

Property SecureIntArray.datas (index As Integer) As Integer
  With This
    If (index < ._lbound) Or (index > ._ubound) Then Return 0

    Return ._data&"hlzahl">._lbound]
  End With
End Property

Property SecureIntArray.datas (index As Integer, new_value As Integer)
  With This
    If (index < ._lbound) Or (index > ._ubound) Then Return

    ._data&"hlzahl">._lbound] = new_value
  End With
End Property

Sub SecureIntArray.Redim (low As Integer, up As Integer)
  If low > up Then Return

  With This
    DeAllocate ._data

    ._ubound = up
    ._lbound = low
    ._data   = Allocate( (up - low + 1) * 4 )

  End With
End Sub

Dim As SecureIntArray mySecInt
Dim As Integer i
```

```
mySecInt.Redim 0, 20              ' Wir erstellen ein Array mit den
Indizes 0 bis 20...
For i = 0 To 24                   ' und versuchen, bis zum Index 24 werte
einzufügen.
  mySecInt.datas(i) = 2 * i       ' Die Property wird diesen Versuch
abblocken.
Next

For i = 0 To 24                   ' Genauso, wie auch die abfrage von
Werten außer der Array-Dimensionen fehlschlägt.
  Print i, mySecInt.datas(i)
Next

Sleep

'Print mySecInt._data             ' Der Pointer ist nicht verfügbar, da es
sich um ein PRIVATE-Feld handelt.
```

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC:
PUBLIC bei UDTs existiert seit FreeBASIC v0.17

Siehe auch:
PUBLIC, PRIVATE (UDT), PROTECTED, TYPE (UDT), THIS, EXTENDS, Objektorientierung

Letzte Bearbeitung des Eintrags am 19.01.13 um 16:54:53
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

PUT (Datei)

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » P » **PUT (Datei)**
Syntax: PUT #Dateinummer, [Position], Daten [, Menge]
Typ: Anweisung/Funktion
Kategorie: Dateien

PUT schreibt Daten in eine Datei, die im BINARY- oder RANDOM-Modus geöffnet wurde.

- '#Dateinummer' ist die Nummer, die beim Öffnen der Datei mit OPEN zugewiesen wurde.
- 'Position' ist die Position innerhalb der Datei, an die geschrieben werden soll. Für BINARY-Dateien ist es die Nummer des Bytes, für RANDOM-Dateien ist das die Nummer des Datensatzes. Wird dieser Parameter ausgelassen, schreibt FreeBASIC an die Position des Dateizeigers. Diese ist abhängig von den letzten Lese-/Schreibzugriffen auf die Datei und der SEEK-Anweisung. Zeigt die aktuelle Position innerhalb der Datei auf eine bereits beschriebene Position, so wird diese überschrieben. Direkt nach dem Öffnen zeigt der Dateizeiger immer auf die erste Stelle in der Datei.
- 'Daten' ist ein Wert, eine Variable oder ein Ausdruck. Auch Arrays und UDTs können geschrieben werden.
- 'Menge' wird eingesetzt, sobald die Daten aus einem ALLOCATE-Puffer geschrieben werden sollen. Dadurch wird PUT mitgeteilt, wie viele Speicherstellen aus dem Puffer in die Datei geschrieben werden sollen. PUT schreibt Menge * SIZEOF(Puffer_Datentyp) Bytes. Wird 'Menge' ausgelassen, nimmt FreeBASIC automatisch 1 an. PUT schreibt also eine einzelne Speicherstelle in die Datei (entspricht SIZEOF(DatenPufferTyp) Bytes, also z. B. 4 Bytes bei einem INTEGER PTR). Bei anderen Datentypen als ALLOCATE-Datenpuffern wird der Parameter 'Menge' ignoriert.
- Wird PUT als Funktion eingesetzt, ist der Rückgabewert eine der FreeBASIC-Fehlernummern. In diesem Fall müssen die Parameter in Klammern gesetzt werden; die Syntax lautet also: Variable = PUT(#Dateinummer, [Position], Daten [, Menge])

Der RANDOM-Modus wird nur noch selten verwendet; die Verwendung sequentieller Modi (INPUT, OUTPUT und APPEND) bzw. des BINARY-Modus hat sich als sinnvoller herausgestellt.
Es wird nicht garantiert, das FreeBASIC RANDOM-Daten im selben Umfang wie QB unterstützt.

Im BINARY-Modus richtet sich die Anzahl der zu schreibenden Bytes nach dem Datentyp der Variable, die in die Datei geschrieben wird. Sie lässt sich - außer für Strings variabler Länge - über SIZEOF ermitteln:

- BYTE und UBYTE - 1 Byte (8 bit)
- SHORT und USHORT - 2 Byte (16 bit)
- INTEGER, UINTEGER, LONG und ULONG - 4 Bytes (32 bit)
- LONGINT und ULONGINT - 8 Bytes (64 bit)
- STRINGs (fester und variabler Länge) - lässt sich mit LEN ermitteln
- ZSTRINGs (fester Länge) - gelesen werden SIZEOF(ZstringVariable) - 1 Bytes. Ein Byte wird als Terminierungs-Byte benötigt.
- ZSTRING PTR - werden wie ALLOCATE-Puffer behandelt (siehe unten).
- WSTRINGs (fester Länge) - geschrieben werden wie bei ZSTRINGs SIZEOF(WstringVariable) - 1 Bytes. Ein Byte wird als Terminierungs-Byte benötigt.
- ALLOCATE-Puffer - Der Parameter 'Menge' gibt an, wie viele Bytes gelesen werden. Achten Sie darauf, dass genügend Speicherplatz reserviert wurde!
- Arrays - Indexzahl * SIZEOF(Datentyp)
- UDTs - lässt sich mit SIZEOF ermitteln

Beispiel 1:

```
Dim buffer AS String
Dim f AS Integer = Freefile
```

```
Open "file.ext" For Binary As "hlzeichen">= "hello world within a file."
Put "hlzeichen">, , buffer
Close "reflinkicon" href="temp0423.html">UDTs zu lesen, geben Sie einfach
nur den Bezeichner ohne Index bzw. Verweis auf einen Record an.
```

Beispiel 2:

```
Type UDT
   a As Integer
   b As Single
   c As String * 5
END TYPE

Dim f AS Integer = Freefile
Open "file.ext" For Binary As "hlkw0">Dim outarray(5) As Integer
Dim outUDT As UDT

Put "hlzeichen">, , outarray()
Put "hlzeichen">, , outUDT

Close "reflinkicon" href="temp0533.html">Pointer angegeben ist, verwenden
Sie die Pointer-Dereferenzierung
*Pointer
```

oder

```
Pointer[index]
```

Außerdem müssen Sie angeben, wie viele Bytes geschrieben werden sollen. Dazu dient der Parameter 'Menge'.

Beispiel 3:

```
Dim f AS INTEGER = Freefile
Open "file.ext" For Binary As "hlkw0">Dim x As Byte Ptr
x = Allocate(8)

Put "hlzeichen">, 1, *x  , 4   ' die ersten vier Speicherstellen schreiben
Put "hlzeichen">, 5, x[4], 4   ' die nächsten vier Speicherstellen
schreiben

DeAllocate x
Close "hlkw0">Dim v1 As Byte, v2 As String * 2
Dim f As integer

v1 = 33
v2 = "33"
f = FreeFile

Open "file.ext" For Binary As "hlkw0">Put "hlzeichen">, , v1
   Put "hlzeichen">, , v2
Close "hlkw0">Open "file.ext" For Binary As "hlkw0">Get "hlzeichen">, ,
v2
   Get "hlzeichen">, , v1
Close "hlkw0">Print v1, v2
Sleep
```

Ausgabe:

```
51      !3
```

Wie Sie sehen, wird in die Datei beim ersten Zugriff zuerst ein BYTE-Wert und **anschließend** ein 2-Byte-STRING geschrieben. Beim zweiten Zugriff wird zuerst ein 2-Byte-STRING und anschließend ein BYTE-Wert eingelesen. Ergebnis ist, dass nicht - wie anzunehmen - wieder zweimal die Ausgabe '33' erfolgt, sondern nur Datenmüll auf dem Bildschirm erscheint. Obwohl die Informationen fehlerfrei gelesen wurden, kann mit den Informationen nicht gearbeitet werden, da sie nach dem Lesen auf die falsche Art und Weise behandelt werden.
Werden beim Lesezugriff die beiden GET-Zeilen vertauscht, so erfolgt die korrekte Ausgabe:

```
33      33
```

Die Ursache hierfür liegt darin, dass im BINARY- und RANDOM-Modus (mit denen GET und PUT ja arbeiten) die Daten nicht in einem für Menschen lesbaren Format abgelegt werden, sondern binär behandelt werden, d. h. so geschrieben werden, wie sie vom Prozessor behandelt werden. Hierbei existieren Unterschiede zwischen der Behandlungsweise von Zeichenketten (des STRINGs) und Zahlen (des BYTE-Werts).

Unterschiede zu QB:

- PUT kann in FreeBASIC auch als Funktion eingesetzt werden.
- In FreeBASIC können ganze Arrays und UDTs geschrieben werden. Auch die Verwendung von ALLLOCATE-Datenpuffern ist möglich.

Unterschiede zu früheren Versionen von FreeBASIC:

- PUT kann seit FreeBASIC v0.13 als Funktion eingesetzt werden.
- Seit FreeBASIC v0.10 können ganze Arrays gelesen werden.

Siehe auch:
PUT (Grafik), OPEN, BINARY, RANDOM, GET "reflinkicon" href="temp0596.html">Dateien (Files)

Letzte Bearbeitung des Eintrags am 27.12.12 um 21:16:13
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

PUT (Grafik)

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » P » **PUT (Grafik)**
Syntax: PUT [ZielPuffer,] [STEP] (x, y), QuellPuffer [(x1, y1)-[STEP](x2, y2)] [, Aktionswort [Parameterliste]]
Typ: Anweisung/Funktion
Kategorie: Grafik

PUT überträgt Daten (einen Bildausschnitt) aus einem Puffer auf den Bildschirm oder in einen anderen Puffer. Die Datenquelle ist vom selben Format, wie es bei GET (Grafik) verwendet wird.

- 'ZielPuffer' ist ein Bildpuffer, wie er mit IMAGECREATE verwendet wird. Wird 'ZielPuffer' ausgelassen, zeichnet FreeBASIC direkt auf den Bildschirm.
- 'STEP' gibt an, dass die darauf folgenden Koordinaten relativ zum aktuellen Grafikcursor angegeben sind.
- '(x, y)' sind die Koordinaten der linken oberen Ecke des Bereichs, der gezeichnet werden soll.
- 'QuellPuffer' ist der Name das Puffers, in dem der zu zeichnende Bildschirmausschnitt gespeichert ist. Ebenso wie 'ZielPuffer' handelt es sich hierbei um einen Bildpuffer, wie er unter oben angegebenen Link erklärt wird.
- '(x1, y1)' und '(x2, y2)' sind die gegenüberliegenden Eckpunkte eines Koordinatenbereichs innerhalb des Quellpuffers, der ausgegeben werden soll. Der Rest des gespeicherten Bildausschnitts wird nicht ausgegeben. Wird dieser Bereich ausgelassen, so gibt FreeBASIC den kompletten Pufferinhalt aus.
- 'Aktionswort' ist entweder PSET, PRESET, AND, OR, XOR, TRANS, ALPHA, ADD oder CUSTOM. Das Aktionswort legt fest, wie die zu zeichnende Grafik (die gespeicherten Daten aus dem Quellpuffer) mit den Pixeln auf dem Bildschirm interagieren sollen. Weiter unten werden die Details dazu aufgelistet. Wenn 'Aktionswort' ausgelassen wird, nimmt FreeBASIC XOR an.
- 'Parameterliste' sind Argumente, die im Zusammenhang mit 'Aktionswort' gefordert werden; siehe unten für Details.
- Wird PUT als Funktion eingesetzt, ist der Rückgabewert eine der FreeBASIC-Fehlernummern. Die Syntax bleibt die gleiche, allerdings müssen die Parameter in Klammern gesetzt werden, z. B.: Ergebnis = PUT((10, 10), myPic, TRANS)

Die Koordinaten sind abhängig von den letzten VIEW- und WINDOW-Anweisungen. Die Ausgabe auf dem Bildschirm hängt ab von den im Puffer gespeicherten Daten, dem Aktionswort und den bereits gesetzten Pixeln.
Abhängig vom Aktionswort wird die Farbnummer der zu zeichnenden Pixel auf unterschiedliche Art und Weise berechnet; für die Aktionswörter PSET, PRESET und TRANS werden zu dieser Berechnung nur die Daten aus dem Quellpuffer herangezogen. Für die anderen Aktionswörter wird zusätzlich die Farbe des Pixels zur Berechnung herangezogen, das überzeichnet werden soll. Beachten Sie, dass das Ergebnis hier auch von der Farbtiefe abhängig ist; während in Modi bis 8bpp die Farbe der ausgegebenen Pixel von der Palette abhängt, stellt bei allen höheren Modi dieselbe Farbnummer auch immer dieselbe Farbe dar.

Aktionswort	Interaktion der Pixel
PSET	Das gespeicherte Pixel überschreibt das Pixel auf dem Bildschirm. Mit diesem Aktionswort wird also tatsächlich der Bildschirmausschnitt ausgegeben, der im Puffer gespeichert wurde.
PRESET	Die gespeicherten Pixel werden invertiert und überschreiben die Pixel auf dem Bildschirm. Der neue Farbwert ist das Ergebnis aus NOT (alter Farbwert). Das Aktionswort PRESET entspricht also PSET, mit dem Unterschied, dass ein invertiertes Bild ausgegeben wird.
AND	Auf dem Bildschirm wird ein Pixel ausgegeben, dessen Farbnummer das Ergebnis eines logischen AND mit dem Pixel auf dem Bildschirm und dem im Puffer gespeicherten Pixel ist.
OR	

	Auf dem Bildschirm wird ein Pixel ausgegeben, dessen Farbnummer das Ergebnis eines logischen OR mit dem Pixel auf dem Bildschirm und dem im Puffer gespeicherten Pixel ist.
XOR	Auf dem Bildschirm wird ein Pixel ausgegeben, dessen Farbnummer das Ergebnis eines logischen XOR mit dem Pixel auf dem Bildschirm und dem im Puffer gespeicherten Pixel ist. XOR ist das Standard-Aktionswort.
TRANS	Die Pixel auf dem Bildschirm werden wie bei PSET durch die gespeicherten Pixel überschrieben. Wenn das gespeicherte Pixel die Masken-Farbe (siehe unten) besitzt, wird es ausgelassen, das Original-Pixel auf dem Bildschirm wird dann nicht überschrieben.
ALPHA [, AlphaWert]	Mit ALPHA lassen sich Transparenz-Effekte erzeugen. Die ALPHA-Methode wird verwendet, um Bildschirmausschnitte 'durchscheinend' auszugeben; das Ergebnis der Bildschirmausgabe ist eine 'Mischfarbe' aus dem Pixel, das überzeichnet wurde, und dem, das im Quellpuffer gespeichert war. Das Aktionswort ALPHA ist nur in hi/truecolor-Modi verfügbar, also in Modi ab 15bpp. 'AlphaWert' stellt den Transparenzgrad des zu zeichnenden Ausschnitts bzw. das Mischungsverhältnis der beiden Farben dar; 255 bedeutet dabei volle Überdeckung, 0 keine Überdeckung. 127 ist der exakte Mittelwert zwischen den beiden Farben. Soll ein Pixel in der Maskenfarbe gezeichnet werden (siehe unten), so arbeitet ALPHA wie TRANS, d. h. Flächen in der Maskenfarbe werden nicht gezeichnet. In 32bpp-Modi ist es auch zulässig, den Parameter 'AlphaWert' auszulassen; in diesem Fall benutzt FreeBASIC den Alphawert, der für jedes Pixel einzeln angegeben wurde. Dies ist nur in 32bpp-Modi möglich, da nur hier ein eingebetteter Alphawert für jedes Pixel möglich ist; siehe dazu auch Interne Pixelformate. Wird 'AlphaWert' ausgelassen, jedoch ein Modus mit einer Farbtiefe unter 32bpp verwendet, so geht FreeBASIC von AlphaWert = 255 aus; dies entspricht völliger Überdeckung bzw. dem Aktionswort TRANS.
ADD [, Faktor]	Die Farbnummer des gespeicherten Pixels wird mit 'Faktor' multipliziert und zur Sättigung des zu überzeichnenden Pixels addiert. 'Faktor' ist ein Wert zwischen 0 und 255. Ebenso wie TRANS und ALPHA werden Flächen in der Maskenfarbe nicht gezeichnet. Das Ergebnis der ADD-Methode sind ebenso wie bei ALPHA durchscheinende Bildschirmausschnitte. Der Transparenzgrad des Ausschnitts ist jedoch nicht nur vom angegebenen Faktor abhängig, sondern auch von der Helligkeit des darunter liegenden Pixels. Beim Überzeichnen schwarzer Pixel verhält sich ADD wie ALPHA; mit zunehmender Helligkeit des zu überzeichnenden Pixels allerdings verschiebt sich das Gleichgewicht der Farbmischung hin zur Transparenz des zu zeichnenden Pixels. Wird 'Faktor' ausgelassen, nimmt FreeBASIC automatisch Faktor = 255 an. Siehe auch Beispiel 2 zum Unterschied zwischen ADD und ALPHA.
CUSTOM, blender [, parameter]	'blender' ist ein Pointer auf eine Funktion der Form FUNCTION blender (BYVAL src AS UINTEGER, BYVAL dst AS UINTEGER, BYVAL parameter AS ANY PTR) AS UINTEGER 'src' ist die Farbe des gespeicherten Pixels, 'dst' ist die Farbe des Pixels, das überschrieben werden soll. 'parameter' ist ein Pointer auf eine beliebige Speicherstruktur, mit der Informationen an die Funktion übergeben werden können. Wird 'parameter' ausgelassen, so nimmt FreeBASIC automatisch 'parameter = 0' an. Das Ergebnis der FUNCTION ist die Farbe des neuen Pixels. Diese Funktion wird für jedes neu zu zeichnende Pixel aufgerufen; die Farbe wird jedes Mal neu berechnet. Achten Sie beim Einsatz dieses Aktionsworts darauf, dass die Ausführungsgeschwindigkeit unter aufwändigen Berechnungen leiden wird!

Die Maskenfarbe hängt von der aktuellen Farbtiefe ab; bei bis zu 8bpp ist sie 0, in den höheren Modi (15, 16, 24 und 32 bpp) ist sie &hFF00FF (helles Pink). Wollen Sie in Ihrem Programm einen Transparenzeffekt UND ein helles Pink verwenden, müssen Sie die Bildteile, die nicht transparent sein sollen, durch ein anderes Pink ersetzen. In der Regel ist der Farbunterschied zwischen &hFF00FF und &hFF01FF mit bloßem Auge nicht zu

erkennen.

Als weitere Alternative kann eine Blendermaske erzeugt werden. Hierbei erstellt man eine Schwarz-Weiß-Maske unter Berücksichtigung der Maskenfarbe und mischt diese mit dem zu überschreibenden Bildbereich. Anschließend kopiert man diese mit OR auf die zu überzeichnende Fläche. Zum Schluss kann das Bild, das man kopieren möchte, über diesen Bereich geschrieben werden.

Der Trick liegt darin, den Hintergrund teilweise über einen Zwischenschritt in das eigentliche Bild zu integrieren. Ein weiterer Vorteil ist hier auch, das eine beliebige Farbe als Transparenzfarbe genutzt werden kann. Natürlich muss die Farbe bekannt sein, wenn die Schwarz/Weiß-Maske erzeugt wird.

Die im Puffer gespeicherten Pixel müssen zur aktuellen Farbtiefe kompatibel sein; wenn Sie ein Bild mit GET einlesen und später die Farbtiefe via SCREENRES ändern, kann es sein, dass die gespeicherten Daten nicht mehr kompatibel sind, sodass PUT mit diesem Puffer nicht richtig eingesetzt werden kann. Eine Konversion zwischen verschiedenen Farbtiefen ist mit IMAGECONVERTROW möglich.

Beispiel 1: Überblick über die verschiedenen Methoden

```
Declare Function checkered_blend( BYVAL src   As UInteger, _
                                  ByVal dest  As UInteger, _
                                  ByVal param As Any Ptr ) AS UInteger

Screenres 320, 240, 16

Dim As Byte Ptr sprite                              ' Speicher für ein
32x32-Pixel-
sprite = ImageCreate( 32, 32 )                      ' Sprite reservieren

Line sprite, (  0,  0 )-( 31, 31 ), &hFF0000, BF    ' ein Sprite Zeichnen
Line sprite, (  0,  0 )-( 31, 31 ), &h00FF00, B
Line sprite, (  8,  8 )-( 23, 23 ), &hFF00FF, BF
Line sprite, (  1,  1 )-( 30, 30 ), &h0000FF
Line sprite, ( 30,  1 )-(  1, 30 ), &h0000FF

Dim As Integer i
For i = 0 To 63                                     ' Einen Hintergrund
zeichnen
  Line( i, 0 )-( i, 240 ), RGB( i * 4, i * 4, i * 4 )
Next i

' Alle Methoden demonstrieren
Put      (   8,  8 ), sprite, PSET                    : Locate  3,12 :
Print "PSET"
Put Step(  16, 24 ), sprite, PRESET                   : Locate  6,12 :
Print "PRESET"
Put Step( -16, 24 ), sprite, AND                      : Locate  9,12 :
Print "AND"
Put Step(  16, 24 ), sprite, OR                       : Locate 12,12 :
Print "OR"
Put Step( -16, 24 ), sprite, XOR                      : Locate 15,12 :
Print "XOR"
Put Step(  16, 24 ), sprite, TRANS                    : Locate 18,12 :
Print "TRANS"
Put Step( -16, 24 ), sprite, ALPHA, 96                : Locate 21,12 :
```

```
Print "ALPHA"
Put Step(  16, 24 ), sprite, ADD, 192                : Locate 24,12 :
Print "ADD"
Put Step( -16, 24 ), sprite, CUSTOM, @checkered_blend
Locate 27,12 : Print "CUSTOM"

ImageDestroy( sprite )                               ' Speicher wieder
freigeben
Sleep

' Benutzerdefinierter Blender: Karierte Ausgabe des Sprites
Function checkered_blend( ByVal src   As UInteger, _
                          ByVal dest  As UInteger, _
                          ByVal param As Any Ptr ) As UInteger
   Static cnt As UInteger
   Dim As UInteger pixel = IIF(((cnt And 4)ShR 2) XOR ((cnt And 128) ShR
7), src, dest)
   cnt += 1
   Return pixel
End Function
```

Beispiel 2: Unterschiede zwischen ALPHA und ADD

```
Screenres 400, 300, 16

Dim As Any Ptr sprite                                ' Speicher für ein
32x32-Pixel-
sprite = ImageCreate( 32, 32 )                       ' Sprite reservieren

Line sprite, (  0, 0 )-( 31, 31 ), &hFF0000, BF  ' ein Sprite Zeichnen
Line sprite, (  0, 0 )-( 31, 31 ), &h00FF00, B
Line sprite, (  8, 8 )-( 23, 23 ), &hFF00FF, BF
Line sprite, (  1, 1 )-( 30, 30 ), &h0000FF
Line sprite, ( 30, 1 )-(  1, 30 ), &h0000FF

Cls
Dim As Integer i : FOR i = 0 TO 63                 ' Einen Hintergrund
zeichnen
   Line( i      , 0 )-( i      , 300 ), RGB( i * 4, i * 4, i * 4 )
   Line( i + 128, 0 )-( i + 128, 300 ), RGB( i * 4, i * 4, i * 4 )
Next i

Put(   0, 0 ), sprite, ADD  , 0
Put(  32, 0 ), sprite, ADD  , 0
Put( 128, 0 ), sprite, ALPHA, 0
Put( 160, 0 ), sprite, ALPHA, 0
Draw String (240, 8), "0"
For i = 1 To 8
   Put (   0, I * 32), sprite, ADD  , i * 32 - 1
   Put (  32, I * 32), sprite, ADD  , i * 32 - 1
   Put ( 128, I * 32), sprite, ALPHA, i * 32 - 1
   Put ( 160, I * 32), sprite, ALPHA, i * 32 - 1
   Draw String (240, I * 32 + 8), Str(i * 32 - 1)
Next
```

```
Sleep
```

Unterschiede zu QB:

- In FreeBASIC ist es möglich, in einen Datenpuffer zu zeichnen.
- Die Methoden TRANS, ALPHA, ADD und CUSTOM sind neu in FreeBASIC.
- In QB können als Bildquelle nur Arrays verwendet werden.
- Das interne Speicherformat für Images in FreeBASIC unterscheidet sich von dem in QB. QB kann das Speicherformat von FreeBASIC nicht verarbeiten.
- FreeBASIC unterstützt Clipping (Einhaltung der Bildschirmgrenzen). QB erzeugt bei Koordinaten außerhalb der Bildschirmgrenzen eine Fehlermeldung.

Unterschiede zu früheren Versionen von FreeBASIC:

- Seit FreeBASIC v0.18.0 kann in der Angabe des darzustellenden Teilbereichs (x1, y1)-(x2, y2) das Schlüsselwort STEP verwendet werden.
- Der Parameter 'param' in der Funktion 'blender' für das Aktionswort CUSTOM existiert seit FreeBASIC v0.17.
- Das Aktionswort ADD existiert seit FreeBASIC v0.17.
- Die Möglichkeit, nur einen Teilbereich des gespeicherten Bildausschnitts auszugeben, besteht seit FreeBASIC v0.15.
- Die Möglichkeit, einen Zielpuffer anzugeben, existiert seit FreeBASIC v0.15.
- Die Aktionswörter ALPHA und CUSTOM existieren seit FreeBASIC v0.14.
- Seit FreeBASIC v0.14 bricht PUT mit einer Fehlermeldung ab, wenn versucht wird, ein Bild auszugeben, das für eine andere Farbtiefe als die aktuell gültige erstellt wurde.
- Die Möglichkeit, anstelle von Arrays auch Pointer als Puffer anzugeben, existiert seit FreeBASIC v0.12.

Siehe auch:
PUT (Datei), GET (Grafik), IMAGECREATE, SCREENRES, Interne Pixelformate, Grafik

Letzte Bearbeitung des Eintrags am 11.09.12 um 02:00:00
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

RANDOM

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » R » **RANDOM**
Syntax: OPEN Dateiname FOR RANDOM [...] , LEN = Recordlänge
Typ: Schlüsselwort
Kategorie: Dateien

Das Schlüsselwort RANDOM wird mit der OPEN-Anweisung verwendet und öffnet die Datei im RANDOM-Modus. Dieser Datei-Modus verwendet eine Puffervariable benutzerdefinierten Typs, um komplette Records einer Datei zu lesen oder zu schreiben. Die Puffervariable umfasst gewöhnlich mehrere Felder. Die Daten werden im Binär-Modus gespeichert, im selben internen Format, das FreeBASIC auch bei GET # und PUT # benutzt.

Es wird davon abgeraten, den RANDOM-Modus zu verwenden; stattdessen empfiehlt sich der BINARY-Modus oder die sequentiellen Modi. Es kann nicht garantiert werden, dass FreeBASIC den Dateimodus RANDOM in vollem Umfang unterstützt.

Unterschiede zu QB:

- In FreeBASIC können UDTs, die STRINGs variabler Länge enthalten, nicht mit RANDOM gespeichert werden.
- In FreeBASIC wird das Schlüsselwort FIELD mit TYPE benutzt, um das 'packing' des UDTs zu bestimmen.

Siehe auch:
OPEN (Anweisung), GET #, PUT #, Dateien (Files)

Weitere Informationen:
Pete's QB Site - QB Express Issue 6 - Random Access Files

Letzte Bearbeitung des Eintrags am 16.02.13 um 13:30:30
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

RANDOMIZE

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » R » **RANDOMIZE**
Syntax: RANDOMIZE [Startwert][, Algorithmus]
Typ: Anweisung
Kategorie: Mathematik

RANDOMIZE initialisiert den Zufallsgenerator.

- 'Startwert' ist vom Typ DOUBLE und gibt den Initialwert für den Zufallsgenerator an. Falls dieser Wert ausgelassen wird, wird ein Wert basierend auf TIMER verwendet.
- 'Algorithmus' ist ein INTEGER-Wert, der den zu verwendenden Algorithmus bestimmt. Falls dieser Wert ausgelassen wird, wird der dialektspezifische Algorithmus verwendet.

Der erstellte Zufallsinitialwert ermöglicht der Funktion RND, Zufallszahlen zu generieren, und bestimmt den Algorithmus, welcher verwendet werden soll. Zulässige Werte für 'Algorithmus' sind:

- **0** - Es soll der dialektspezifische Algorithmus verwendet werden. Bei -lang fb ist dieser 3, bei -lang qb 4 und bei -lang fblite 1.
- **1** - Verwendet die *rand()*-Funktion der *C runtime library*. Die Zufallswerte sind hierbei plattformspezifisch.
- **2** - Ein performanter Algorithmus wird verwendet. Diese Methode sollte plattformunabhängig sein und erzeugt 32-bit Doublewerte mit einem annehmbaren Maß an Zufälligkeit.
- **3** - Verwendet den Mersenne-Twister-Algorithmus. Diese Methode sollte plattformunabhängig sein und erzeugt 32-bit Doublewerte mit einem hohen Maß an Zufälligkeit.
- **4** - Eine Funktion wird verwendet, die die gleiche Nummernfolge wie QBASIC erzeugen soll. Diese Methode sollte plattformunabhängig sein und erzeugt 24-bit genaue Zahlen mit einem geringen Maß an Zufälligkeit.
- **5** - Verwendet wie im dritten Algorithmus den Mersenne-Twist-Algorithmus, nur wird beginnend mit dem ersten Aufruf von RND alle 256 Aufrufe ein zusätzlicher Wert auf den Startwert aufaddiert. Dieser Wert entspringt unter Windows der *CryptApi* und unter Linux */dev/urandom*. Dadurch soll ein noch besserer Zufallswert erreicht werden. Sollte diese Methode fehlschlagen, so wird der Standardwert des Mersenne-Twist-Algorithmus verwendet.

Für jeden Initialwert wird jeder Algorithmus eine bestimmte, reproduzierbare Reihenfolge von Zahlen generieren. Soll bei jedem Aufruf von RANDOMIZE eine andere Zahlenreihenfolge erstellt werden, sollte ein sich verändernder Initialwert übergeben werden - z. B. der Rückgabewert der Funktion TIMER. Wird 'Startwert' ausgelassen oder erhält er den Wert -1.0, dann wird ein auf TIMER basierender Startwert verwendet.

Beachten Sie: Wird TIMER als Initialwert öfter in einer Sekunde aufgerufen, werden immer dieselben Zahlen erstellt. Grundsätzlich ist es nicht nötig, RANDOMIZE öfter als einmal mit einem sich verändernden Übergabeparameter aufzurufen, da die Zahlenreihenfolge der späteren Aufrufe nicht *zufälliger* sind als die nach dem ersten Aufruf. In den meisten Fällen sollte der Mersenne-Twister-Algorithmus zu einer ausreichend zufälligen Reihenfolge von Zahlen führen, ohne dass der Initialwert zwischen RND-Aufrufen neu belegt werden müsste.

Beim Aufruf von RANDOMIZE mit dem QB-kompatiblen Algorithmus wird ein Teil der Zufallswerte aufbewahrt. Das bedeutet, dass bei mehrmaligen Aufrufen von RANDOMIZE mit demselben Initialwert nicht jedes Mal die gleiche Reihenfolge erzeugt wird. Um eine bestimmte Reihenfolge zu erhalten, muss RND mit einem negativen Übergabeparameter aufgerufen werden.

Beispiel:

```
'Die C-rand()-Funktion soll verwendet werden
Randomize , 1

'Gibt zehn Zufallszahlen aus
For i As Integer = 1 To 10
  Print Rnd
Next
Sleep
```

Unterschiede zu QB:
Der Parameter 'Algorithmus' ist neu in FreeBASIC. QB hat nur einen Algorithmus (nachgebaut in FB im Algorithmus 4 und der Standardalgorithmus im Dialekt -lang qb).

Plattformbedingte Unterschiede:
Algorithmus 5 verwendet unter Windows die *CryptApi*, unter Linux */dev/urandom*. Unter DOS ist der Algorithmus nicht vorhanden, hier wird bei Angabe des Wertes 5 der Standard-Algorithmus verwendet.

Unterschiede zu früheren Versionen von FreeBASIC:
Der Algorithmus 5 existiert seit FreeBASIC v0.24.

Unterschiede unter den FB-Dialektformen:
Der Standardalgorithmus hängt vom aktuellen Dialekt ab:

- -lang fb verwendet die 32-bit Mersenne-Twister-Funktion.
- -lang qb erzeugt dieselbe Ausgabe wie RND in QB.
- -lang deprecated und -lang fblite verwenden die *C-Funktion rand()*. Diese ergibt unter Win32 15-bit-Werte und unter Linux und DOS 32-bit-Werte.

Siehe auch:
RND, Mathematik

Letzte Bearbeitung des Eintrags am 27.12.12 um 21:17:11
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

READ

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » R » **READ**
Syntax: READ Variable1 [,Variable2 [,...]]
Typ: Anweisung
Kategorie: Speicher

READ liest Daten, die zuvor mit einer DATA-Anweisung gespeichert wurden. Diese Daten werden als einfache Liste betrachtet. Ist eine DATA-Anweisung abgearbeitet, wird mit der folgenden fortgefahren.

READ wird auch als Schlüsselwort im Zusammenhang mit ACCESS verwendet.

Beachte: READ kann nur DATA-Werte lesen, solange welche vorhanden sind. Soll READ darüber hinaus lesen, sind die Ergebnisse in allen FreeBASIC-Dialekten undefiniert und es kann zu einem Programmabsturz kommen.

Beispiel:

```
Dim As Integer h(4)
Dim As String hs
Dim As Integer readindex

' Daten in das Array einlesen
For readindex = 0 To 4
  Read h(readindex)       ' Integer einlesen
  Print "Nummer" ; readindex ; " = " ; h(readindex)
Next readindex
Print

Read hs                   ' String einlesen
Print  "String = " & hs
Sleep

' Datenblock.
Data 3, 234, 4354, 23433, 87643, "Bye!"
```

Siehe auch:
DATA, RESTORE, Verschiedenes

Letzte Bearbeitung des Eintrags am 27.12.12 um 21:17:38
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

REALLOCATE

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » R » **REALLOCATE**
Syntax: REALLOCATE (Pointer, Bytes)
Typ: Funktion
Kategorie: Speicher

REALLOCATE ändert die Größe eines reservierten Speicherbereichs.

- 'Pointer' ist ein Pointer auf einen bereits bestehenden Speicherbereich, dessen Größe geändert werden soll. Ist 'Pointer' 0, so verhält sich REALLOCATE exakt wie ALLOCATE.
- 'Bytes' ist die neue Größe des Speicherbereichs in Bytes.
- Die Werte, die im Speicherbereich 'Pointer' gespeichert sind, werden bei der Redimensionierung zwischengespeichert.
- Der Rückgabewert ist ein Pointer auf das erste Byte des redimensionierten Speicherbereichs. Er kann im selben Pointer gespeichert werden, der schon bei 'Pointer' angegeben wurde, darf (und sollte!) aber in einer anderen Pointer-Variablen gespeichert werden. Die Adresse des redimensionierten Speicherbereichs *kann* sich von der Adresse unterscheiden, auf die 'Pointer' zeigt.
- Wenn die Redimensionierung fehlschlägt, ist der Rückgabewert 0.

REALLOCATE ist kein Teil der FreeBASIC Runtime Library, sondern ein Alias von realloc der C-Lib.
Achtung: Es kann nicht garantiert werden, dass diese Funktion auf allen Plattformen Multithreading unterstützt, d.h. thread-safe ist. Unter Windows und Linux sind aktuell durch die verwendeten Implementationen der Betriebssysteme aber keine Probleme zu erwarten.

Beispiel:

```
DIM a AS INTEGER PTR, b AS INTEGER PTR, i AS INTEGER

a = ALLOCATE(5 * LEN(INTEGER))          ' Speicher für fünf
INTEGER-Stellen reservieren
FOR i = 0 TO 4
  a&"hlzeichen">] = (i + 1) * 2                  ' Werte in den
Speicher schreiben
NEXT i

b = REALLOCATE( a, 10 * LEN(INTEGER) )  ' Speicher für fünf zusätzliche
Werte reservieren
FOR i = 5 TO 9
  b&"hlzeichen">] = (i + 1) * 2                  ' Zusätzliche Werte in
den Puffer schreiben
NEXT i

FOR i = 0 TO 9                          ' Werte ausgeben
  PRINT b&"hlzeichen">];
NEXT i

DEALLOCATE b                            ' Speicher freigeben

SLEEP                                   ' Auf Tastendruck warten
```

Hinweis:
Wenn Speicher mit REALLOCATE vergrößert wird, ist beim Umgang mit STRINGs auf das gleiche Verhalten wie bei ALLOCATE zu achten. Der neue Speicherbereich wird nicht mit Nullen überschrieben,

sodass Probleme auftreten können. Um dem entgegenzuwirken kann der neu reservierte Speicherbereich etwa mit CLEAR auf null gesetzt werden.

Unterschiede zu QB: neu in FreeBASIC

Plattformbedingte Unterschiede:
Es kann nicht garantiert werden, dass die Prozedur auf allen Plattformen thread-safe ist.

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht REALLOCATE nicht zur Verfügung und kann nur über **__REALLOCATE** aufgerufen werden.

Siehe auch:
ALLOCATE, CALLOCATE, DEALLOCATE, Pointer, Speicher

Letzte Bearbeitung des Eintrags am 05.04.14 um 15:53:44
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

REDIM

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » R » **REDIM**
Syntax:

```
REDIM [PRESERVE] [SHARED] ArrayName([[Untergrenze TO] Obergrenze]  _
                                    [, [Untergrenze TO] Obergrenze] _
                                    [, ...]) [AS Datentyp] [, weitereArrays]

REDIM [PRESERVE] [SHARED] AS Datentyp ArrayName([[Untergrenze TO]
Obergrenze] _
                                    [, ...]) [, weitereArrays]
```

Typ: Anweisung
Kategorie: Deklaration

Anmerkung zur Syntax: Unterstriche (_) am Zeilenende werden von FreeBASIC so interpretiert, als wäre die Zeile nicht unterbrochen; dies dient nur der besseren Übersichtlichkeit und hat letzt–end–lich keine Auswirkungen auf die Programmausführung.

REDIM erstellt ein dynamisches Array oder ändert dessen Größe.

- Wird 'PRESERVE' bei einem vorhandenen Array benutzt, bleibt der Inhalt des Arrays bei einer Größenänderung erhalten.
  Beachten Sie, dass PRESERVE nicht mit mehrfach-dimensionierten Arrays funktioniert (siehe auch den Artikel zu PRESERVE).
- 'SHARED' legt den gemeinsamen (shared) Zugriff für ein Array für das Modul fest (file-scope).
- 'ArrayName' ist der Name des Arrays, das erstellt oder dessen Größe geändert werden soll.
- 'Untergrenze' und 'Obergrenze' legen die Größe des Arrays fest. Wird 'Untergrenze' ausgelassen, dann verwendet FreeBASIC standardmäßig den Wert 0.
- 'Datentyp' gibt den Datentyp des Arrays an. Wurde das Array bereits zuvor definiert, dann kann 'AS Datentyp' weggelassen werden, oder der Datentyp muss mit dem bereits zuvor verwendeten identisch sein.

REDIM kann benutzt werden, um neue, in der Länge variable Arrays zu erstellen oder vorhandene Arrays dieser Art zu vergrößern. REDIM erstellt IMMER variable Arrays, also können, anders als bei DIM, variable Arrays mit konstanten Indizes erstellt werden.

Wird ein neues variables Array erstellt, dann werden die Elemente vom Default-Konstruktor erstellt. Bei einfachen Datatypen wie INTEGER oder DOUBLE werden sie mit Null (0) initialisiert, bei UDTs vom Default-CONSTRUCTOR, der dann aufgerufen wird.

Wenn REDIM benutzt wird, um ein Array erstmalig zu dimensionieren, bewirkt es, dass es als dynamisches Array erstellt wird, während DIM ein statisches Feld erstellt. Der Metabefehl '$DYNAMIC bzw. die Einstellung OPTION DYNAMIC unterbindet die Erstellung statischer Felder.
Achtung: '$DYNAMIC bzw. OPTION DYNAMIC steht nur bis FreeBASIC v0.16 zur Verfügung!

Wird die Größe eines variablen Arrays ohne den PRESERVE-Parameter geändert, werden alle Elemente zerstört und neue Elemente werden vom Default-Konstruktor erstellt. Mit dem Parameter PRESERVE werden die vorhandenen Elemente nicht zerstört, außer wenn das Array verkleinert wird: dann gehen zwangsläufig die abgetrennten Elemente verloren. Wird ein variables Array vergrößert, dann werden die neuen Elemente vom Default-Konstruktor *am Ende* des Arrays erstellt.

Achtung: REDIM kann NICHT bei Arrays benutzt werden, die zu UDTs gehören, da momentan nur Arrays

mit fester Größe in UDTs unterstützt werden.

Beispiel 1:

```
DIM x() AS INTEGER

REDIM x(3)
PRINT UBOUND(x)

REDIM x(6)
PRINT UBOUND(x)
SLEEP
```

Beispiel 2:

```
' Array mit 5 Elementen anlegen
ReDim Array(1 To 5) As Integer

For index As Integer = LBound(Array) To UBound(Array)
  Array(index) = index
Next

' Auf 10 Elemente vergrößern; dabei wird auch die untere Grenze
verschoben
ReDim Preserve Array(9) As Integer

Print "Index", "Wert"
For index As Integer = LBound(Array) To UBound(Array)
  Print index, Array(index)
Next
Sleep
```

Ausgabe:

```
Index          Wert
 0              1
 1              2
 2              3
 3              4
 4              5
 5              0
 6              0
 7              0
 8              0
 9              0
```

Unterschiede zu QB:

- Das Schlüsselwort 'PRESERVE' existiert nur in Visual Basic
- Die alternative Syntax REDIM AS Typ Variable ist neu in FreeBASIC.
- Bei der Speicherung mehrdimensionaler Arrays folgen in FreeBASIC die Werte aufeinander, deren erster Index gleich ist. In QB folgen die Werte aufeinander, deren letzter Index gleich ist.

Unterschiede zu früheren Versionen von FreeBASIC:

- Seit FreeBASIC v0.17 muss der Typ explizit deklariert werden, außer das Programm wird mit der Option *-lang deprecated* compiliert.
- Seit FreeBASIC v0.16 dürfen Felder unbekannter Größe an jedem Programmpunkt, auch in Prozeduren, erstellt werden.

Unterschiede unter den FB-Dialektformen:

- In der Dialektform -lang qb und -lang fblite sind Variablen, die in einer Prozedur dimensioniert wurden, in der ganzen Prozedur sichtbar. Variablen, die mit 'SHARED' dimensioniert wurden, sind im ganzen Modul sichtbar.
- In der Dialektform -lang fb und -lang deprecated sind Variablen, die in einem Block dimensioniert wurden (FOR ... NEXT, WHILE ... WEND, DO ... LOOP, SCOPE ... END SCOPE) nur in diesem Block sichtbar.
- In der Dialektform -lang fb sind OPTION-Anweisungen (z. B. OPTION BASE, OPTION DYNAMIC) nicht erlaubt.

Siehe auch:
DIM, PRESERVE, SCOPE, DYNAMIC (Meta), STATIC (Meta), UBOUND, LBOUND, Arrays

Letzte Bearbeitung des Eintrags am 27.12.12 um 21:18:58
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

REM

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » R » **REM**
Syntax A: REM Kommentar
Syntax B: ' Kommentar
Syntax C:

```
/'
   Kommentare, mehrzeilig
'/
```

Typ: Anweisung
Kategorie: Kommentare

REM leitet einen Kommentar ein.
Ein Kommentar ist eine Zeile im Programm, die nicht ausgeführt wird; sie kann nützliche Informationen und Notizen enthalten.
Aufgrund besserer Übersichtlichkeit wird statt des Befehls REM bevorzugt das Kommentarsymbol ' benutzt. Es hat dieselbe Funktion.
Wenn Sie einen größeren Kommentarblock einfügen wollen, können Sie dies mit den Kommentarblock-Zeichen realisieren; ein Kommentarblock beginnt mit /' und endet mit '/.

Beispiel:

```
REM blabla etc etc

' Noch ein Kommentar.

/'
Diese Zeilen sind ebenfalls Kommentare.
Sie werden erst durch ein Kommentar-Ende-Zeichen beendet:
'/

"hlzahl">0
   Auf diese Weise wurde in früheren Versionen ein Kommentarblock
realisiert.
"reflinkicon" href="temp0203.html">INCLUDE und LANG eine Syntaxform gibt,
die das Kommentarsymbol nutzt. Es wird dabei geprüft, ob das zweite
Symbol ein Dollarzeichen ($) ist. Möchte man das verhindern, genügt es
beispielsweise schon, zwei Kommentarsymbole zu verwenden.
'' Ein Kommentar
```

Unterschiede zu QB:
Die Kommentarblock-Zeichen sind neu in FreeBASIC.

Unterschiede zu früheren Versionen von FreeBASIC:
Die Kommentarblock-Zeichen existieren seit FreeBASIC v0.16.

Letzte Bearbeitung des Eintrags am 27.12.12 um 21:19:31
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

RESET

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » R » **RESET**
Syntax: RESET [Argument]
Typ: Anweisung
Kategorie: Dateien

RESET schließt alle geöffneten Dateien oder setzt die Standardeingabe bzw. -ausgabe zurück.

- Wird 'Argument' ausgelassen, dann schließt der Befehl alle Dateien. So verwendet ist er identisch mit CLOSE ohne Argument.
- Wird für 'Argument' 0 oder 1 angegeben, dann werden umgeleitete Streams der Standardeingabe (*stdin*) bzw. Standardausgabe (*stdout*) geschlossen. 0 schließt Streams der Standardeingabe, 1 schließt Streams der Standardausgabe. Andere Werte als 0 und 1 werden ignoriert.

RESET setzt bei einem Fehler eine Fehlernummer, die mittels ERR abgefragt werden kann.

Beispiel:

```
Dim x As String

' Aus der Standardeingabe eines Datenstroms lesen
Open Cons For Input As "hlkw0">While EOF(1) = 0
  Input "hlzeichen">, x
  Print """; x; """
Wend
Close "hlkommentar">' Eingabe auf die Tastatur zurücksetzen
Reset(0)

Print "Gib einen Text ein:"
Input x

' Aus der Standardeingabe lesen (jetzt Tastatur)
Open Cons For Input As "hlkw0">While EOF(1) = 0
  Input "hlzeichen">, x
  Print """; x; """
Wend
Close "reflinkicon" href="temp0283.html">OPEN, OPEN CONS, OPEN PIPE,
CLOSE, Dateien (Files)
```

Weitere Informationen:
Wikipedia-Artikel zu den Standard-Datenströmen
Letzte Bearbeitung des Eintrags am 27.12.12 um 21:19:42
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

RESTORE

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » R » **RESTORE**
Syntax: RESTORE [Label]
Typ: Anweisung
Kategorie: Speicher

RESTORE gibt an, welche mit DATA gespeicherten Variablen von der nächsten READ-Anweisung gelesen werden sollen. Bei großen Mengen eingebetteter Daten kann RESTORE dazu verwendet werden, das Einlesen zu organisieren, indem ein Label angegeben wird, bei dem das Lesen begonnen werden soll.
Wird 'Label' ausgelassen, beginnt FreeBASIC das Einlesen an der ersten DATA-Anweisung.

Beispiel:

```
DIM AS INTEGER x, y

RESTORE bar
READ x
PRINT x

RESTORE
READ x,  y
PRINT x,  y

RESTORE foo
READ x, y
PRINT x, y
SLEEP

DATA 1, 2
foo:
DATA 3
bar:
DATA 4
```

Ausgabe:

```
 4
 1         2
 3         4
```

In diesem Beispiel wird nur eine kleine Datenmenge verwendet. Andere Anwendungen haben jedoch oft größere Datenmengen zu verwalten, sodass die Organisation per RESTORE sehr nützlich sein kann.

Siehe auch:
DATA, READ, Verschiedenes

Letzte Bearbeitung des Eintrags am 27.12.12 um 21:19:57
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

RESUME

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » R » **RESUME**
Syntax: RESUME [NEXT]
Typ: Anweisung
Kategorie: Fehlerbehandlung

RESUME wird im Zusammenhang mit ON ERROR verwendet und kehrt von einer Fehlerbehandlungsroutine zu der Anweisung, die den Fehler erzeugt hat, zurück.
RESUME setzt den Wert von ERR auf 0 zurück.

Das optionale 'NEXT' hinter RESUME bewirkt, dass nach Abarbeitung der Fehlerbehandlungsroutine nicht zu der Anweisung zurückgesprungen wird, die den Fehler hervorgerufen hat, sondern zu der darauf folgenden Anweisung.

Beispiel:

```
"hlstring">"deprecated"

ON ERROR GOTO FehlerHandler

DIM File AS STRING
INPUT "Welche Datei oeffnen? ", File

OPEN File FOR INPUT AS "hlkommentar">'Anweisungen
CLOSE "hlkw0">END

FehlerHandler:
PRINT "Datei existiert nicht"
INPUT "Bitte geben Sie eine andere Datei an: ", File
RESUME
```

Unterschiede unter den FB-Dialektformen:
RESUME steht nur in der Dialektform -lang deprecated und -lang qb zur Verfügung.

Siehe auch:
ERROR, ON ERROR, ERR, GOTO, Fehlerbehandlung, Debugging

Letzte Bearbeitung des Eintrags am 04.07.12 um 22:52:09
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

RETURN

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » R » **RETURN**
Syntax: RETURN [BYVAL] [{ Ausdruck | Label }]
Typ: Anweisung
Kategorie: Programmablauf

Mit RETURN wird eine FUNCTION oder SUB verlassen. In früheren Versionen von FreeBASIC wurde mit diesem Befehl von einem GOSUB-Aufruf zurückgekehrt.

- Beim Verlassen einer FUNCTION wird mit 'Ausdruck' der Rückgabewert festgelegt. RETURN dient damit als Kurzform für
  FUNCTION = [BYVAL] Ausdruck : EXIT FUNCTION
  Beim Verlassen einer SUB entfällt die Angabe von 'Ausdruck'. RETURN dient als Alternative zu EXIT SUB
- Wird RETURN zum Rücksprung von GOSUB verwendet, ist 'Label' ein optionaler Parameter, der angibt, an welcher Stelle das Programm fortgesetzt werden soll. Das Label muss zum Zeitpunkt des RETURN-Aufrufs bereits definiert sein, darf also nicht hinter der RETURN-Anweisung liegen.

Seit FreeBASIC v0.16 kann GOSUB/RETURN nur noch in den FB-Dialektformen -lang fblite oder -lang qb verwendet werden; siehe GOSUB für weitere Details.

Hinweis:
Die Angabe 'BYVAL' spielt nur bei Funktionen eine Rolle, die *by reference* mit BYREF arbeiten.

Beispiel:

```
DECLARE FUNCTION twice(x AS INTEGER) AS INTEGER

PRINT "Das doppelte von 4 ist " & twice(4)
SLEEP

FUNCTION twice(x AS INTEGER) AS INTEGER
  RETURN 2 * x
END FUNCTION
```

Unterschiede zu QB:
In der Dialektform -lang qb arbeitet RETURN genauso wie unter QB. Unterschiede bei anderen Dialektformen siehe unten.

Unterschiede zu früheren Versionen von FreeBASIC:

- Seit FreeBASIC v0.90 kann 'BYVAL' angegeben werden, um explizit nur den Wert, nicht aber die Referenz zu übergeben. Siehe: BYREF (Rückgaben)
- Seit FreeBASIC v0.16 kann mit RETURN eine SUB verlassen werden.
- Seit FreeBASIC v0.16 kann RETURN nicht mehr auf Prozedurebene eingesetzt werden.
- Seit FreeBASIC v0.13 kann RETURN zum Verlassen einer FUNCTION eingesetzt werden.

Unterschiede unter den FB-Dialektformen:

- In der Dialektform -lang qb kann RETURN *entweder* in Verbindung mit GOSUB *oder* mit Prozeduren verwendet werden. Standardmäßig wird GOSUB unterstützt. Mit OPTION NOGOSUB kann die Unterstützung der Prozeduren eingeschaltet werden

- In der Dialektform -lang fblite kann RETURN ebenfalls *entweder* in Verbindung mit GOSUB *oder* mit Prozeduren verwendet werden. Standardmäßig werden Prozeduren unterstützt. Mit OPTION GOSUB wird die Unterstützung von GOSUB eingeschaltet.
- In den Dialektformen -lang fb und -lang deprecated kann RETURN nur in Zusammenhang mit Prozeduren eingesetzt werden.

Siehe auch:
GOSUB, FUNCTION, SUB, OPTION GOSUB, OPTION NOGOSUB, Programmablauf, BYVAL (Rückgaben), Prozeduren

Letzte Bearbeitung des Eintrags am 04.07.13 um 22:48:10
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

RGB

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » R » **RGB**
Syntax: RGB (rot, grün, blau)
Typ: Funktion
Katerorie: Grafik

RGB errechnet die gültige 32-Farbnummer in hi-/truecolor-Bildschirmmodi bei angegebenem Rot-, Grün- und Blau-Wert der Farbe.

- 'rot', 'grün' und 'blau' müssen im Bereich von 0 bis 255 liegen.
- Der Rückgabewert ist ein UINTEGER und entspricht einer Farbnummer im Format &hAARRGGBB. RR, GG und BB sind dabei die Farbkomponenten im Hexadezimalformat. AA ist der implizite Alphawert und wird automatisch auf &hFF (=255) gesetzt.

RGB ist ein Makro, das folgendermaßen definiert ist:

```
#DEFINE RGB(r,g,b) ((CUINT(r) SHL 16) OR (CUINT(g) SHL 8) OR CUINT(b) OR
&hFF000000)
```

Beispiel:

```
ScreenRes 640,480,32                ' 32bit Farbtiefe
Line(0,0)-(319,479), RGB(255,0,0)   ' rote Strecke
Line(639,0)-(320,479), RGB(0,0,255) ' blaue Strecke

Sleep
```

Um aus einem RGB-Wert die einzelnen Farbkomponenten auszulesen, können die Befehle AND und SHR verwendet werden. Siehe dazu Beispiel 2 im Referenzeintrag RGBA.

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC:
Die Version __RGB in der Dialektform -lang qb existiert seit FreeBASIC v0.24.

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht RGB nicht zur Verfügung und kann nur über **__RGB** aufgerufen werden.

Siehe auch:
RGBA, COLOR (Anweisung), DEFINE (Meta), Grafik

Letzte Bearbeitung des Eintrags am 15.10.12 um 22:39:07
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

RGBA

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » R » **RGBA**
Syntax: RGBA (rot, grün, blau, alpha)
Typ: Funktion
Katerorie: Grafik

RGBA errechnet die gültige 32-Farbnummer in hi-/truecolor-Bildschirmmodi bei angegebenem Rot-, Grün-, Blau- und Alphawert der Farbe.

- 'rot', 'grün', 'blau' und 'alpha' müssen im Bereich von 0 bis 255 liegen.
- Der Rückgabewert ist ein UINTEGER und entspricht einer Farbnummer im Format &hAARRGGBB. RR, GG, BB und AA sind dabei die Farbkomponenten bzw. der Alphawert im Hexadezimalformat.

RGBA arbeitet genauso wie RGB, abgesehen von der Möglichkeit, einen Alphawert (Transparenzgrad) anzugeben. Es ist ein Makro, das folgendermaßen definiert ist:

```
#DEFINE RGBA(r,g,b,a) ((CUINT(r) SHL 16) OR (CUINT(g) SHL 8) OR CUINT(b)
OR (CUINT(a) SHL 24))
```

Beispiel 1: Zeichnen mit halbtransparenten Bildern

```
ScreenRes 320, 240, 32  ' Grafikfenster erstellen

Dim As Any Ptr img
Dim As Integer x, y

' Bild mit verschiedenen Transparenzstufen und Farben erstellen
img = ImageCreate(64, 64)
For x = 0 To 63
  For y = 0 To 63
    PSet img, (x, y), RGBA(x * 4, 0, y * 4, (x + y) * 2)
  Next y
Next x
Circle img, (31, 31), 25,      RGBA(0, 127, 192, 192), ,,, F '
halbtransparenter blauer Kreis
Line   img, (26, 20)-(38, 44), RGBA(255, 255, 255, 0),    BF '
transparentes weißes Rechteck

' Hintergrund zeichnen (diagonale weiße Strecken)
For x = -240 To 319 Step 10
  Line (x, 0)-Step(240, 240), RGB(255, 255, 255)
Next

Line (10,  10)-(310,  37), RGB(127, 0, 0), BF ' rotes Rechteck
Line (10, 146)-(310, 229), RGB(0, 127, 0), BF ' grünes Rechteck

' Bild zeichnen mit PSET
Draw String(64, 20), "PSet"
Put(48,  48), img, PSet
Put(48, 156), img, PSet

' Bild zeichnen mit ALPHA
Draw String (220, 20), "Alpha"
```

```
Put(208,  48), img, Alpha
Put(208, 156), img, Alpha

' Speicher freigeben und Programm beenden
ImageDestroy img
Sleep
```

Beispiel 2: Auslesen einzelner Farbkomponenten aus einem RGBA-Wert

```
"hlzeichen">(c) (CUInt(c) Shr 16 And 255)
"hlzeichen">(c) (CUInt(c) Shr  8 And 255)
"hlzeichen">(c) (CUInt(c)        And 255)
"hlzeichen">(c) (CUInt(c) Shr 24        )

Dim As UInteger r, g, b, a
Dim As UInteger col = RGBA(255, 192, 64, 128)

Print Using "Farbe: _&h\      \"; Hex(col, 8)

r = RGBA_R(col)
g = RGBA_G(col)
b = RGBA_B(col)
a = RGBA_A(col)

Print
Print Using "Rot:          _&h\\ = "; Hex(r, 2); r
Print Using "Gruen:        _&h\\ = "; Hex(g, 2); g
Print Using "Blau:         _&h\\ = "; Hex(b, 2); b
Print Using "Alpha:        _&h\\ = "; Hex(a, 2); a
```

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC:
Die Version __RGBA in der Dialektform -lang qb existiert erst seit FreeBASIC. v0.24.

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht RGBA nicht zur Verfügung und kann nur über **__RGBA** aufgerufen werden.

Siehe auch:
RGB, COLOR (Anweisung), DEFINE (Meta), Grafik

Letzte Bearbeitung des Eintrags am 15.10.12 um 22:47:03
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

RIGHT

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » R » **RIGHT**
Syntax: RIGHT[$](Text, Anzahl)
Typ: Funktion
Kategorie: Stringfunktionen

RIGHT gibt die letzten 'Anzahl' Zeichen von 'Text' zurück.

- 'Text' ist ein STRING, ZSTRING oder WSTRING, dessen Teilstring zurückgegeben werden soll.
- 'Anzahl' ist ein INTEGER mit der Anzahl der zurückgegebenen Zeichen. Ist 'Anzahl' kleiner als 0, dann wird ein Leerstring zurückgegeben. Ist 'Anzahl' größer als die Länge von 'Text', dann wird der gesamte 'Text' zurückgegeben.
- Der Rückgabewert ist ein STRING bzw. WSTRING, der die letzten 'Anzahl' Zeichen von 'Text' enthält.

Das Dollarzeichen ($) als Suffix ist optional.

Beispiel:

```
DIM AS STRING Text
Text = "hello world"
PRINT RIGHT(Text, 5) 'Ausgabe: "world"
```

Unterschiede zu QB:

- In QB ist das Suffix $ verbindlich.
- Da QB kein Unicode unterstützt, existiert dort auch nicht die Möglichkeit, WSTRING zu verwenden.

Plattformbedingte Unterschiede:
Unicode wird in der DOS-Portierung von FreeBASIC nicht unterstützt, deshalb kann dort auch kein WSTRING verwendet werden.

Unterschiede unter den FB-Dialektformen:

- In der Dialektform -lang qb ist das Suffix $ verbindlich.
- In den Dialektformen -lang fblite und -lang fb ist das Suffix optional.

Siehe auch:
LEFT, MID (Funktion), String-Funktionen

Letzte Bearbeitung des Eintrags am 27.12.12 um 21:20:47
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

RMDIR

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » R » **RMDIR**
Syntax: RMDIR (Ordnername)
Typ: Funktion
Kategorie: System

RMDIR löscht ein Verzeichnis aus dem Dateisystem. Das angegebene Verzeichnis darf keine Dateien oder Unterordner enthalten.

- 'Ordnername' ist ein String, der den Namen des Ordners enthält, der gelöscht werden soll. Er kann mit einer Pfadangabe versehen sein. Wird kein Arbeitspfad angegeben, so löscht RMDIR den Ordner im aktuellen Arbeitsverzeichnis; siehe CURDIR.
- Der Rückgabewert ist entweder 0, wenn der Ordner gelöscht werden konnte, oder -1, wenn ein Fehler aufgetreten ist. Dies kann z. B. bedeuten, dass kein Ordner mit angegebenen Namen existiert, dass sich noch Dateien im Ordner befinden oder dass auf das angegebene Gerät nicht zugegriffen werden kann (z. B. auf eine CD oder einen schreibgeschützten Ordner).

Beispiel:

```
' sicherstellen, dass ein Verzeichnis mit Namen foo existiert
MKDIR "foo"
IF RMDIR("foo") THEN
  PRINT "In "; CURDIR; "\foo\ befinden sich noch Dateien!"
END IF
SLEEP
```

Achtung: Der Ordner wird nicht in den Papierkorb verschoben, sondern unwiederbringlich gelöscht!

Unterschiede zu QB:

- FreeBASIC erlaubt auch lange Ordnernamen bis zu 255 Zeichen; die kurzen DOS-Namen sind nicht nötig.
- RMDIR kann in FreeBASIC auch als Funktion eingesetzt werden.

Plattformbedingte Unterschiede:

- Unter Linux muss der Ordnername 'case sensitive' erfolgen. Windows und DOS sind 'case insensitive'.
- Das Trennzeichen für den Dateipfad ist unter Linux der vorwärtsgerichtete Slash /. Windows verwendet den Backslash \, erlaubt aber auch den Slash. DOS verwendet den Backslash.

Siehe auch:
CHDIR, MKDIR, CURDIR, SHELL, KILL, Betriebssystem-Anweisungen

Letzte Bearbeitung des Eintrags am 11.01.14 um 23:56:05
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

RND

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » R » **RND**
Syntax: RND [(Wiederholung)]
Typ: Funktion
Kategorie: Mathematik

RND gibt eine zufällige DOUBLE-Zahl zwischen 0 und 1 (im Intervall [0; 1[) zurück, abhängig von der mit RANDOMIZE gesetzten Startzahl. Wird kein RANDOMIZE verwendet, dann liefert RND bei jedem Programmstart dieselbe Folge an Zufallszahlen.

'Wiederholung' ist ein optionaler Parameter, der entweder gleich oder ungleich null ist. Wenn er gleich null ist, wird immer dieselbe Zufallszahl ausgegeben; ist er ungleich null oder wird er ausgelassen, dann wird die nächste Zufallszahl der Liste ausgegeben.

Beispiel:

```
RANDOMIZE TIMER

PRINT "Ausgabe von 3 Zufallszahlen von 1 bis 6"
FOR i AS INTEGER = 1 TO 3
   PRINT INT(RND * 6) + 1
NEXT
SLEEP
```

Erklärung zum Beispiel:
Mit RND*6 wird eine zufällige Gleitkommazahl zwischen 0 und 6 erzeugt und mit INT abgerundet. Dadurch entsteht eine zufällige ganze Zahl von 0 bis 5. Durch Addition von 1 erhält man eine Zufallszahl von 1 bis 6.

Unterschiede zu QB:
In allen Dialektformen außer -lang qb unterscheidet das optionale Argument nur zwischen null und ungleich null; der in QB verfügbare RND-Reset RND(-1) ist damit unmöglich. Stattdessen muss der Zufallsgenerator mit RANDOMIZE zurückgesetzt werden.

Unterschiede unter den FB-Dialektformen:
Der Standard-Algorithmus zur Erzeugung der Zufallzahlen hängt von der verwendeten Dialektform ab. Siehe dazu RANDOMIZE.

Siehe auch:
RANDOMIZE

Weitere Informationen:
Tutorial: Zufallszahlen verwenden

Letzte Bearbeitung des Eintrags am 07.11.13 um 23:52:01
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

RSET

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » R » **RSET**
Syntax: RSET Ziel, Quelle
Typ: Anweisung
Kategorie: Stringfunktionen

RSET befüllt das 'Ziel' mit 'Quelle', behält aber die Länge von 'Ziel' bei. Es wird benutzt, um Strings rechtsbündig zu formatieren.

- 'Ziel' ist ein STRING, der befüllt werden soll.
- 'Quelle' ist ein String mit den Daten, mit denen 'Ziel' befüllt wird.

War 'Ziel' vor der Bearbeitung mit RSET z. B. ein fünf Zeichen langer String, so ist er auch nach der Bearbeitung noch fünf Zeichen lang, unabhängig von der Länge von 'Quelle'. Ist 'Quelle' länger als 'Ziel', so werden die vorderen Zeichen des Strings nach 'Ziel' kopiert, der Rest von 'Quelle' wird ignoriert. Ist 'Quelle' kürzer als 'Ziel', so wird 'Ziel' von vorn mit Leerzeichen aufgefüllt. In allen Fällen bleibt 'Quelle' unverändert.

Beispiel:

```
DIM Quelle AS STRING
DIM Ziel AS STRING

Ziel = "0123456789"
Quelle = "***"
PRINT Ziel, Quelle

RSET  Ziel, Quelle
PRINT Ziel, Quelle
SLEEP
```

Ausgabe:

```
0123456789   ***
       ***   ***
```

Unterschiede zu QB:
In QB lautet die Syntax RSET Ziel=Quelle.

Siehe auch:
LSET, SPACE, String-Funktionen

Letzte Bearbeitung des Eintrags am 27.12.12 um 21:21:16
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

RTRIM

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » R » **RTRIM**
Syntax: RTRIM[$] (Stringausdruck [, [ANY] zuEntfernen])
Typ: Funktion
Kategorie: Stringfunktionen

RTRIM (=right trim) entfernt bestimmte angehängte Zeichen aus einem String.

- 'Stringausdruck' ist der STRING, ZSTRING oder WSTRING, der gekürzt werden soll.
- 'zuEntfernen' ist ein Ausdruck, der angibt, welche Zeichen entfernt werden sollen. Wird dieser Parameter ausgelassen, entfernt FreeBASIC automatisch alle Leerzeichen am Ende des Strings. 'zuEntfernen' darf aus mehreren Zeichen bestehen.
- Wird die ANY-Klausel verwendet, entfernt FreeBASIC jedes Zeichen aus 'zuEntfernen' am Ende von 'Stringausdruck'.
- Der Rückgabewert ist der um die angegebenen Zeichen gekürzte String.

RTRIM arbeitet *case-sensitive*, d. h. die Groß-/Kleinschreibung ist ausschlaggebend.

Das Dollarzeichen ($) als Suffix ist optional.

Beispiel:

```
PRINT LEN( RTRIM("hello ") )
PRINT RTRIM("hello World", "ld")
PRINT RTRIM("hello World", "dl")
PRINT RTRIM("hello World", ANY "dl")
SLEEP
```

Ausgabe:

```
5
hello Wor
hello World
hello Wor
```

Unterschiede zu QB:

- Der Parameter 'zuEntfernen' und die ANY-Klausel sind neu in FreeBASIC.
- In QB ist das Suffix $ verbindlich.
- Da QB kein Unicode unterstützt, existiert dort auch nicht die Möglichkeit, WSTRING zu verwenden.

Plattformbedingte Unterschiede:
Unicode wird in der DOS-Portierung von FreeBASIC nicht unterstützt, deshalb kann dort auch kein WSTRING verwendet werden.

Unterschiede zu früheren Versionen von FreeBASIC:
Der Parameter 'zuEntfernen' und die ANY-Klausel existieren seit FreeBASIC v0.15

Unterschiede unter den FB-Dialektformen:

- In der Dialektform -lang qb ist das Suffix $ verbindlich.
- In den Dialektformen -lang fblite und -lang fb ist das Suffix optional.

Siehe auch:
LTRIM, TRIM, INSTR, String-Funktionen

Letzte Bearbeitung des Eintrags am 27.12.12 um 21:21:26
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

RUN

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » R » **RUN**
Syntax: RUN (Datei [, Argumente])
Typ: Funktion
Kategorie: System

RUN startet eine ausführbare Datei. Wenn die Datei ordnungsgemäß ausgeführt werden konnte, wird die Kontrolle anschließend an das Betriebssystem zurückgegeben.

- 'Datei' ist ein STRING, der den vollen Dateinamen (inklusive Erweiterung) der auszuführenden Datei enthält.
- 'Argumente' ist ein STRING, der die Argumente enthält, die an das Programm übergeben werden sollen.
- Wenn ein Fehler aufgetreten ist, z. B. wenn die aufzurufende Datei nicht existiert, wird -1 zurückgegeben.

RUN transferiert die Kontrolle auf das aufgerufene Programm. Nachdem das aufgerufene Programm beendet wurde, erhält das aufrufende Programm die Kontrolle nicht mehr zurück. Sie wird ans Betriebssystem zurückgegeben. Das bedeutet, dass als Rückgabewert von RUN immer -1 aufgefangen wird, da bei erfolgreichem Aufruf kein Rückgabewert mehr ausgewertet werden kann.

Beispiel:
Wenn die Datei "file.exe" im aktuellen Arbeitsverzeichnis existiert, wird sie durch diesen Code aufgerufen:

```
IF RUN "file.exe" = -1 THEN
  PRINT """file.exe"" existiert scheinbar doch nicht."
ELSE
    PRINT "Es ist egal was hier passiert, es wird nie ausgefuehrt."
END IF

SLEEP 'Wird nur ausgefuehrt, wenn der RUN-Aufruf -1 zurueckgibt.
```

Da das Programm automatisch beendet wird, wenn "file.exe" erfolgreich ausgeführt werden konnte, muss nicht geprüft werden, ob der Rückgabewert -1 war.

Unterschiede zu QB:

- In FreeBASIC muss der volle Dateiname inklusive Erweiterung (.exe unter Win32 und DOS) angegeben werden.
- RUN kann in FreeBASIC auch als Funktion eingesetzt werden.

Plattformbedingte Unterschiede:

- Unter Linux muss der Dateiname 'case sensitive' erfolgen. Windows und DOS sind 'case insensitive'.
- Das Trennzeichen für den Dateipfad ist unter Linux der vorwärtsgerichtete Slash /. Windows verwendet den Backslash \, erlaubt aber auch den Slash. DOS verwendet den Backslash.

Siehe auch:
CHAIN, EXEC, SHELL, END, COMMAND, Betriebssystem-Anweisungen

Letzte Bearbeitung des Eintrags am 27.12.12 um 21:21:48
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

SADD

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » S » **SADD**
Syntax: SADD (Stringvariable)
Typ: Funktion
Kategorie: Pointer

SADD gibt einen Pointer auf eine String-Variable zurück. Es ist ein Alias zu STRPTR und verweist auf den Anfang des tatsächlichen String-Inhalts, während @ auf den Anfang der Datenstruktur zeigt. Im Artikel zum Datentyp STRING finden Sie nähere Informationen zur Handhabung des Speichers.

Beispiel:

```
DIM s AS STRING
PRINT SADD(s)

s = "hello"
PRINT SADD(s)

s = "abcdefg, 1234567, 54321"
PRINT SADD(s), @s

POKE UBYTE, SADD(s), 65
PRINT s
```

Ausgabebeispiel (x86):

```
0
160527952
160527952     3215135288
Abcdefg, 1234567, 54321
```

Unterschiede zu QB:

- In FreeBASIC können auch die Adressen von STRINGs fester Länge ausgegeben werden.
- FreeBASIC gibt eine 32bit- bzw. 64bit-Adresse aus; in QB wird hier ein 16bit-Offset ausgegeben, das Segment muss zusätzlich ermittelt werden.

Siehe auch:
STRING (Datentyp), @, STRPTR, PEEK, POKE, Grundlagen zu Pointern, Zusammenstellung von Pointer-Anweisungen

Letzte Bearbeitung des Eintrags am 11.01.14 um 23:59:55
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

SCOPE

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » S » **SCOPE**
Syntax:

```
SCOPE
  ' Programmcode
END SCOPE
```

Typ: Anweisung
Kategorie: Deklaration

Ein SCOPE-Block ermöglicht es, Variablen temporär zu (re)dimensionieren und zu benutzen.
Wenn eine Variable innerhalb einer SCOPE-Struktur dimensioniert wird, kann sie nur bis zum Ende des SCOPE-Blocks verwendet werden. Besteht die Variable bereits außerhalb des Blocks und wird im Block neu dimensioniert, dann wird die alte Variable bis zum Ende des Blocks ignoriert - auf ihren Wert kann innerhalb des Blocks nicht zugegriffen werden. Am Ende des Blocks wird der Wert der alten Variablen wiederhergestellt.

Alle Anweisungen innerhalb eines SCOPE-Blocks vor der Redimensionierung einer Variable werden auf die Variable zugreifen, die außerhalb des Blocks definiert wurde.

SCOPEs dürfen bis zu einem beliebigen Grad ineinander verschachtelt sein. Sie können sowohl auf Modulebene als auch auf Prozedurebene verwendet werden; siehe Prozeduren.

Hinweis: Alle Schleifen (DO ... LOOP, WHILE ... WEND, FOR ... NEXT), Bedingungen (IF ... THEN, SELECT CASE) sowie TYPE, UNION und ENUM erzeugen intern einen eigenen SCOPE-Block.

Beispiel 1:

```
DIM AS INTEGER x = 1, y = 2
PRINT "Initialisierung:"
PRINT "x ="; x; ", "; "y ="; y
PRINT

SCOPE
   x += 1
   y += 1
   PRINT "x und y werden im ersten SCOPE um eins erhoeht."
   PRINT "x ="; x; ", "; "y ="; y
   PRINT

   DIM x AS INTEGER = 11
   PRINT "Erstes SCOPE, Redimensionierung von x"
   PRINT "x ="; x; ", "; "y ="; y
   PRINT

   SCOPE
      DIM y AS INTEGER = 12
      PRINT "Zweites SCOPE, Redimensionierung von y"
      PRINT "x ="; x; ", "; "y ="; y
      PRINT
   END SCOPE
```

```
END SCOPE

PRINT "nach den SCOPEs"
PRINT "x ="; x; ", "; "y ="; y
SLEEP
```

Ausgabe:

```
Initialisierung:
x = 1, y = 2

x und y werden im ersten SCOPE um eins erhoeht.
x = 2, y = 3

Erstes SCOPE, Redimensionierung von x
x = 11, y = 3

Zweites SCOPE, Redimensionierung von y
x = 11, y = 12

nach den SCOPEs
x = 2, y = 3
```

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC:

- Seit FreeBASIC v0.17 erzeugen TYPE, UNION und ENUM implizit einen eigenen SCOPE-Block.
- Seit FreeBASIC v0.16 erzeugen alle Schleifen und Bedingungen implizit einen eigenen SCOPE-Block.
- SCOPE existiert seit FreeBASIC v0.15.

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb und -lang fblite steht SCOPE nicht zur Verfügung.

Siehe auch:
DIM, REDIM, Datentypen und Deklarationen, Gültigkeitsbereich von Variablen

Letzte Bearbeitung des Eintrags am 27.12.12 um 21:31:12
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

SCREEN (Anweisung)

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » S » **SCREEN (Anweisung)**
Syntax: SCREEN Modus[,[Farbtiefe][,[Seitenzahl][, [Flags][, Bildwiederholrate]]]]
Typ: Anweisung
Kategorie: Grafik

SCREEN setzt den aktuellen Bildschirm-Grafikmodus. Sofern möglich, sollte der flexiblere SCREENRES-Befehl statt SCREEN verwendet werden, um das Grafikfenster einzustellen. Die Möglichkeit, Grafikbildschirme mit einer im Modus-Parameter fest kodierten Größe festlegen zu können, existiert lediglich aus Kompatibilitätsgründen zu QB. Mit dem SCREENRES-Befehl kann die Fenstergröße explizit festgelegt werden.

- 'Modus' ist die Nummer des Bildschirmmodus (siehe unten).
- Die anderen Parameter entsprechen denen von SCREENRES.

'Farbtiefe' hat nur Auswirkungen, wenn 'Modus' größer ist als 13. Wird sie ausgelassen, dann wird bei einem Bildschirmmodus über 13 die Farbtiefe von 8 bpp (256 Farben) verwendet.

Bedeutung der Modi (Auszug aus dem englischen Handbuch):

Modus	Auflösung	Text	Farben
0	Textmodus	80x25 oder 80x50 Text Spalten/Zeilen 8x14 Zeichensatz oder 8x8 Zeichensatz	Jeweils 16 Vorder- und Hintergrundfarben aus der Standardpalette
1	320x200 in CGA Emulation	40x25 Text Spalten/Zeilen 8x8 Zeichensatz	16 Hintergrundfarben und einen von 4 Vordergrundsets für die COLOR-Anweisung
2	640x200 in CGA Emulation	80x25 Text Spalten/Zeilen 8x8 Zeichensatz	Gleichzeitig je 2 Farben aus der Standardpalette
7	320x200 in EGA Emulation	40x25 Text Spalten/Zeilen 8x8 Zeichensatz	Gleichzeitig je 16 Farben aus der Standardpalette
8	640x200 in EGA Emulation	80x25 Text Spalten/Zeilen 8x8 Zeichensatz	Gleichzeitig je 16 Farben aus der Standardpalette
9	640x350 in EGA Emulation	80x25 oder 80x43 Text Spalten/Zeilen 8x14 oder 8x8 Zeichensatz	Gleichzeitig je 16 Farben aus der Standardpalette
10	640x350 in EGA Emulation	80x25 oder 80x43 Text Spalten/Zeilen 8x14 oder 8x8 Zeichensatz	Gleichzeitig je 2 Farben aus 16 Mio. Farben (24 Bit)
11	640x480 in VGA Emulation	80x30 oder 80x60 Text Spalten/Zeilen 8x16 oder 8x8 Zeichensatz	Gleichzeitig je 2 Farben aus 16 Mio. Farben (24 Bit)
12	640x480 in VGA Emulation	80x30 oder 80x60 Text Spalten/Zeilen 8x16 oder 8x8 Zeichensatz	Gleichzeitig je 16 Farben aus 16 Mio. Farben (24 Bit)
13	320x200 in MCGA Emulation	40x25 Text Spalten/Zeilen 8x8 Zeichensatz	Gleichzeitig je 256 Farben aus 16 Mio. Farben (24 Bit)
14	320x240	40x30 Text Spalten/Zeilen 8x8 Zeichensatz	Gleichzeitig je 256 Farben aus 16 Mio. Farben (24 Bit) oder Truecolor

15	400x300	50x37 Text Spalten/Zeilen 8x8 Zeichensatz	Gleichzeitig je 256 Farben aus 16 Mio. Farben (24 Bit) oder Truecolor
16	512x384	64x24 oder 64x48 Text Spalten/Zeilen 8x16 oder 8x8 Zeichensatz	Gleichzeitig je 256 Farben aus 16 Mio. Farben (24 Bit) oder Truecolor
17	640x400	80x25 oder 80x50 Text Spalten/Zeilen 8x16 oder 8x8 Zeichensatz	Gleichzeitig je 256 Farben aus 16 Mio. Farben (24 Bit) oder Truecolor
18	640x480	80x30 oder 80x60 Text Spalten/Zeilen 8x16 oder 8x8 Zeichensatz	Gleichzeitig je 256 Farben aus 16 Mio. Farben (24 Bit) oder Truecolor
19	800x600	100x37 oder 100x75 Text Spalten/Zeilen 8x16 oder 8x8 Zeichensatz	Gleichzeitig je 256 Farben aus 16 Mio. Farben (24 Bit) oder Truecolor
20	1024x768	128x48 oder 128x96 Text Spalten/Zeilen 8x16 oder 8x8 Zeichensatz	Gleichzeitig je 256 Farben aus 16 Mio. Farben (24 Bit) oder Truecolor
21	1280x1024	160x64 oder 160x128 Text Spalten/Zeilen 8x16 oder 8x8 Zeichensatz	Gleichzeitig je 256 Farben aus 16 Mio. Farben (24 Bit) oder Truecolor

Die Anzahl der verwendeten Spalten und Zeilen kann mit WIDTH geändert werden.

Beispiel: Bildschirmmodus 640x480x32bpp mit 4 Bildschirmseiten im Vollbildmodus einstellen:

```
SCREEN 18, 32, 4, 1
IF SCREENPTR = 0 THEN
  PRINT "Bildschirmmodus nicht initialisiert!"
  END
END IF
'...
SLEEP
```

Unterschiede zu QB:

- QB erlaubt keine SCREEN-Flags.
- Die Bedeutung der Parameter ist in FreeBASIC nicht mehr dieselbe wie in QB. Siehe dazu die alte QB-Syntax unter 'Unterschiede unter den FB-Dialektformen'.
- In FreeBASIC gibt es mehr verfügbare Bildschirmmodi.

Plattformbedingte Unterschiede:

- Unter DOS werden keine Fenster- und OpenGL-Flags unterstützt.
- Das SCREEN-Flag 'GFX_HIGH_PRIORITY' existiert nur unter Win32.

Unterschiede zu früheren Versionen von FreeBASIC:

- Seit FreeBASIC v0.18.4 existiert das SCREEN-Flag 'GFX_HIGH_PRIORITY'.
- Seit FreeBASIC v0.17 existieren die SCREEN-Flags 'GFX_NO_FRAME', 'GFX_SHAPED_WINDOW', 'GFX_ALPHA_PRIMITIVES', 'GFX_ALWAYS_ON_TOP' und 'GFX_GFX_MULTISAMPLE'.
- Die SCREEN-Flags 'GFX_STENCIL_BUFFER' und GFX_ACCUMULATION_BUFFER' haben seit v0.17 eine neue Belegung.

- Seit FreeBASIC v0.17 führt ein Aufruf von SCREEN 0 zu einem 80x25-Textmodus mit geleertem Bildschirm.
- Der Grafikmodus ohne visuelles Feedback existiert seit v0.15 (Screen-Flag 'GFX_NULL')
- Das Kein-Moduswechsel-Flag exisitert seit v0.15 (Screen-Flag 'GFX_NO_SWITCH')
- Die Möglichkeit, eine Bildwiederholrate anzugeben, existiert seit v0.13

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb muss die Syntax von QBASIC verwendet werden:

```
SCREEN &"reflinkicon" href="temp0355.html">SCREEN (Funktion), SCREENRES,
SCREENINFO, SCREENLIST, Grafik
```

Letzte Bearbeitung des Eintrags am 20.10.12 um 21:59:27
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

SCREEN (Funktion)

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » S » **SCREEN (Funktion)**
Syntax: SCREEN (Zeile, Spalte [, Typ])
Typ: Funktion
Kategorie: Konsole

SCREEN gibt Informationen über den Text im Konsole- oder Grafik-Fenster zurück.

- 'Zeile' ist die Zeile (der Hochwert), in der das Zeichen steht, über das Informationen ausgegeben werden sollen.
- 'Spalte' ist die Spalte (der Rechtswert), in der das Zeichen steht, über das Informationen ausgegeben werden sollen.
- 'Typ' ist die Art der Information, die ausgegeben werden soll. Wird hier 0 angegeben oder 'Typ' ausgelassen, ist der Rückgabewert der ASCII-Code des Zeichens an der angegebenen Position. Wird 1 angegeben, ist der Rückgabewert die Farbe des Zeichens an der angegebenen Position in der Form Vordergrund OR (Hintergrund SHL 8)

Beispiel:

```
DIM i AS INTEGER
WIDTH 80, 25

FOR i = 1 TO 16
  COLOR i, (i + 8) MOD 16
  PRINT "1234567890"
NEXT

COLOR 15, 0
LOCATE 1, 20
PRINT "ASCII-Code in Zeile 5, Spalte 7:"
LOCATE 2, 20
PRINT SCREEN(5, 7, 0)

LOCATE 4, 20
PRINT "Vordergrundfarbe in Zeile 2, Spalte 10:"
LOCATE 5, 20
PRINT (SCREEN(2, 10, 1)) MOD 16

LOCATE 7, 20
PRINT "Hintergrundfarbe in Zeile 7, Spalte 5:"
LOCATE 8, 20
PRINT (SCREEN(7, 5, 1)) \ 16

SLEEP
```

Unterschiede zu QB:
In QB erzeugt SCREEN einen Fehler, wenn Koordinaten außerhalb des Fensters verwendet werden.

Plattformbedingte Unterschiede:
Unter Linux kann der zurückgegebene Wert vom in der Konsole angezeigten Wert abweichen. Zum Beispiel können nicht anzeigbare Kontrollcodes wie das LF-Zeichen (ASCII-Zeichen 10), welches das Ende des geschriebenen Textes markiert, übernommen werden.

Siehe auch:
SCREEN (Anweisung), LOCATE, COLOR (Anweisung), WIDTH (Anweisung), PRINT (Anweisung), Konsole

Letzte Bearbeitung des Eintrags am 20.10.12 um 22:05:39
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

SCREENCONTROL

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » S » **SCREENCONTROL**
Syntax A: SCREENCONTROL Option, [param1 [, param2 [, param3 [, param4]]]]
Syntax B: SCREENCONTROL Option, caption
Typ: Anweisung
Kategorie: Grafik

SCREENCONTROL ermittelt oder bearbeitet Einstellungen der gfxlib.

- 'Option' ist ein INTEGER-Wert, der festlegt, welche Einstellungen bearbeitet oder ermittelt werden sollen.
- 'param1', 'param2', 'param3' und 'param4' sind Variablen, die entweder die neuen Einstellungen enthalten, wenn solche geändert werden sollen, oder die die abgefragten Einstellungen aufnehmen. Alle param-Variablen sind vom Typ INTEGER.
- 'caption' ist eine STRING-Variable, die ebenso wie die param-Variablen die neue Einstellung enthält oder die aktuelle Einstellung aufnimmt.

Im folgenden werden die für 'Option' einsetzbaren Werte aufgelistet. Bitte beachten Sie, dass die angegebenen Konstanten aus der fbgfx.bi Elemente des NAMESPACEs 'FB' sind. Alle param-Variablen, die hier nicht erwähnt sind, werden mit null ausgefüllt.

In der Regel muss ein Grafik- oder OpenGL-Fenster initialisiert worden sein (siehe SCREENRES), bevor der Befehl sinnvoll eingesetzt werden kann.

Leseoptionen

Wert	Konstante in der fbgfx.bi	Bedeutung
0	GET_WINDOW_POS	Gibt die aktuelle Position des Grafikfensters als Desktopkoordinaten an. 'param1' enthält die X-Koordinate, 'param2' die Y-Koordinate. Wird diese Funktion aufgerufen, bevor ein Grafikfenster mit SCREENRES oder SCREEN initialisiert wurde, so wird in 'param1' und 'param2' der Wert 0 zurückgegeben.
1	GET_WINDOW_TITLE	Gibt den Titel des Programmfensters in 'caption' zurück. Dies funktioniert auch ohne initialisiertem Grafikscreen, wenn der Titel zuvor mit SCREENCONTROL oder WINDOWTITLE gesetzt wurde.
2	GET_WINDOW_HANDLE	Gibt den Handle des aktiven Programmfensters in 'param1' zurück. Der Handle ist eine Zahl, die das Fenster im System identifiziert. Unter Windows ist dieser Wert ein HWND. Unter X11 stellt es eine "Window"-ID dar. Wird diese Funktion aufgerufen, bevor ein Grafikfenster mit SCREENRES oder SCREEN initialisiert wurde, so wird in 'param1' der Wert 0 zurückgegeben.
3	GET_DESKTOP_SIZE	Gibt die aktuelle Größe des Desktops in Pixeln an. 'param1' enthält die Breite, 'param2' die Höhe des Desktops. Die Aktion funktioniert auch ohne initialisiertem Grafikfenster.
4	GET_SCREEN_SIZE	Gibt die aktuelle Größe des Grafikfensters in Pixeln an. 'param1' enthält die Breite, 'param2' die Höhe des Grafikfensters. Wird diese Funktion aufgerufen, bevor ein Grafikfenster mit SCREENRES oder SCREEN initialisiert wurde, so wird in 'param1'

		und 'param2' der Wert 0 zurückgegeben.
5	GET_SCREEN_DEPTH	Gibt in 'param1' die aktuelle Farbtiefe des Grafikfensters in Bits pro Pixel zurück. Wird diese Funktion aufgerufen, bevor ein Grafikfenster mit SCREENRES oder SCREEN initialisiert wurde, so wird in 'param1' der Wert 0 zurückgegeben.
6	GET_SCREEN_BPP	Gibt in 'param1' die aktuelle Farbtiefe des Grafikfensters in Byte pro Pixel zurück. Wird diese Funktion aufgerufen, bevor ein Grafikfenster mit SCREENRES oder SCREEN initialisiert wurde, so wird in 'param1' der Wert 0 zurückgegeben.
7	GET_SCREEN_PITCH	Gibt in 'param1' die Breite einer Bildschirmzeile in Bytes aus; dieser Wert ist das Produkt aus der Breite in Pixeln und der Farbtiefe in Byte pro Pixel. Wird diese Funktion aufgerufen, bevor ein Grafikfenster mit SCREENRES oder SCREEN initialisiert wurde, so wird in 'param1' der Wert 0 zurückgegeben.
8	GET_SCREEN_REFRESH	Gibt in 'param1' die Bildwiederholrate in Hertz zurück. Wird diese Funktion aufgerufen, bevor ein Grafikfenster mit SCREENRES oder SCREEN initialisiert wurde, so wird in 'param1' der Wert 0 zurückgegeben.
9	GET_DRIVER_NAME	Gibt den Namen des aktiven Grafiktreibers in 'caption' zurück. Wird diese Funktion aufgerufen, bevor ein Grafikfenster mit SCREENRES oder SCREEN initialisiert wurde, so wird in 'caption' ein Leerstring zurückgegeben.
10	GET_TRANSPARENT_COLOR	Gibt in 'param1' die aktuelle Maskenfarbe zurück, die von der aktuellen Farbtiefe abhängig ist. Wird diese Funktion aufgerufen, bevor ein Grafikfenster mit SCREENRES oder SCREEN initialisiert wurde, so wird in 'param1' der Wert 0 zurückgegeben.
11	GET_VIEWPORT	Gibt die Koordinaten des aktuellen Viewports (der Clipping-Regionen, die durch VIEW (Grafik) gesetzt wurden) zurück. Dabei nehmen 'param1' und 'param2' die X- und Y-Koordinate der linken oberen Ecke und 'param3' und 'param4' die X- und Y-Koordinate der rechten unteren Ecke des Viewports auf. Wird diese Funktion aufgerufen, bevor ein Grafikfenster mit SCREENRES oder SCREEN initialisiert wurde, so wird in den Parametern der Wert 0 zurückgegeben.
12	GET_PEN_POS	Gibt in 'param1' und 'param2' die X- und Y-Koordinate des Grafikcursors in Grafikfensterkoordinaten aus. Diese Koordinaten werden von Grafik-Anweisungen benutzt, die über das STEP-Schlüsselwort relative Koordinaten unterstützen. Wird diese Funktion aufgerufen, bevor ein Grafikfenster mit SCREENRES oder SCREEN initialisiert wurde, so wird in 'param1' und 'param2' der Wert 0 zurückgegeben.
13	GET_COLOR	Gibt die von COLOR (Anweisung) festgelegten Farben zurück; 'param1' nimmt die Vordergrundfarbe, 'param2' die Hintergrundfarbe auf. Wird diese Funktion aufgerufen, bevor ein Grafikfenster mit SCREENRES oder SCREEN initialisiert wurde, so wird in 'param1' und 'param2' der Wert 0 zurückgegeben.
14	GET_ALPHA_PRIMITIVES	

		Gibt in 'param1' den Wert TRUE (-1) zurück, wenn GFX_ALPHA_PRIMITIVES gesetzt wurde (siehe SCREENRES). Ansonsten wird FALSE (0) zurückgegeben.
15	GET_GL_EXTENSIONS	Gibt in 'caption' einen String zurück, der alle unterstützten GL-Erweiterungen enthält. Wird kein OpenGL-Modus verwendet, dann wird ein leerer String zurückgegeben.
16	GET_HIGH_PRIORITY	Gibt in 'param1' den Wert TRUE (-1) zurück, wenn GFX_HIGH_PRIORITY gesetzt wurde (siehe SCREENRES). Ansonsten wird FALSE (0) zurückgegeben.

Schreiboptionen

Wert	Konstante in der fbgfx.bi	Bedeutung
100	SET_WINDOW_POS	Legt die Position des Grafikfensters in Desktopkoordinaten fest; 'param1' und 'param2' sind die neue X- und Y-Koordinate des Grafikfensters relativ zum linken oberen Bildschirmrand.
101	SET_WINDOW_TITLE	Ändert den Text in der Titelleiste des aktiven Fensters. 'caption' enthält den neuen Titel des Fensters. Diese Aktion hat dieselbe Auswirkung wie WINDOWTITLE caption (siehe WINDOWTITLE). Sie kann auch ausgeführt werden, bevor ein Grafikfenster initialisiert wurde.
102	SET_PEN_POS	Bestimmt die Position des Grafikcursors. 'param1' und 'param2' enthalten die neue X- und Y-Koordinate des Grafikcursors. Diese Koordinaten werden von Grafik-Anweisungen benutzt, die über das Schlüsselwort STEP relative Koordinaten unterstützen.
103	SET_DRIVER_NAME	Setzt mit 'caption' den aktiven Grafiktreiber. Die Aktion wirkt sich erst auf das nächste initialisierte Grafikfenster aus.
104	SET_ALPHA_PRIMITIVES	Bestimmt mit 'param1', ob GFX_ALPHA_PRIMITIVES gesetzt sein soll (-1) oder nicht (0) (siehe SCREENRES).
105	SET_GL_COLOR_BITS	Setzt mit 'param1' die Anzahl der notwendigen Bits für den OpenGL color buffer. Die Aktion kann auch ausgeführt werden, bevor ein Grafikfenster initialisiert wurde.
106	SET_GL_COLOR_RED_BITS	Setzt mit 'param1' die Anzahl der Bits für die Rot-Komponente des OpenGL color buffer. Die Aktion kann auch ausgeführt werden, bevor ein Grafikfenster initialisiert wurde.
107	SET_GL_COLOR_GREEN_BITS	Setzt die Anzahl der Bits für die Grün-Komponente des OpenGL color buffer. Die Aktion kann auch ausgeführt werden, bevor ein Grafikfenster initialisiert wurde.
108	SET_GL_COLOR_BLUE_BITS	Setzt mit 'param1' die Anzahl der Bits für die Blau-Komponente des OpenGL color buffer. Die Aktion kann auch ausgeführt werden, bevor ein Grafikfenster initialisiert wurde.
109	SET_GL_COLOR_ALPHA_BITS	Setzt mit 'param1' die Anzahl der Bits für die Alpha-Komponente des OpenGL color buffer. Die Aktion kann auch ausgeführt werden, bevor ein Grafikfenster initialisiert wurde.
110	SET_GL_DEPTH_BITS	Setzt mit 'param1' die Anzahl der notwendigen Bits für den OpenGL depth buffer. Die Aktion kann auch ausgeführt werden, bevor ein Grafikfenster initialisiert wurde.
111	SET_GL_STENCIL_BITS	Setzt mit 'param1' die Anzahl der notwendigen Bits für den OpenGL stencil buffer. Die Aktion kann auch ausgeführt werden, bevor ein Grafikfenster initialisiert wurde.
112	SET_GL_ACCUM_BITS	

		Setzt mit 'param1' die Anzahl der notwendigen Bits für den OpenGL accumulation buffer. Die Aktion kann auch ausgeführt werden, bevor ein Grafikfenster initialisiert wurde.
113	SET_GL_ACCUM_RED_BITS	Setzt mit 'param1' die Anzahl der Bits für die Rot-Komponente des OpenGL accumulation buffer. Die Aktion kann auch ausgeführt werden, bevor ein Grafikfenster initialisiert wurde.
114	SET_GL_ACCUM_GREEN_BITS	Setzt mit 'param1' die Anzahl der Bits für die Grün-Komponente des OpenGL accumulation buffer. Die Aktion kann auch ausgeführt werden, bevor ein Grafikfenster initialisiert wurde.
115	SET_GL_ACCUM_BLUE_BITS	Setzt mit 'param1' die Anzahl der Bits für die Blau-Komponente des OpenGL accumulation buffer. Die Aktion kann auch ausgeführt werden, bevor ein Grafikfenster initialisiert wurde.
116	SET_GL_ACCUM_ALPHA_BITS	Setzt mit 'param1' die Anzahl der Bits für die Alpha-Komponente des OpenGL accumulation buffer. Die Aktion kann auch ausgeführt werden, bevor ein Grafikfenster initialisiert wurde.
117	SET_GL_NUM_SAMPLES	Setzt mit 'param1' die Anzahl der Samples, die von OpenGL Multisampling genutzt werden sollen. Die Aktion kann auch ausgeführt werden, bevor ein Grafikfenster initialisiert wurde.

Weitere Optionen

Wert	Konstante in der fbgfx.bi	Bedeutung
200	POLL_EVENTS	Alle Ereignisse (Events) wie Tasten- oder Mausbetätigung werden abgefragt. Nützlich in OpenGL-Programmen, wenn FLIP nicht verwendet wird, da die FLIP-Anweisung diese Abfragen veranlasst.

Beispiel:
Es wird ein transparentes Fenster mit einem Schriftzug erzeugt. Bei einem Mausklick auf den Schriftzug kann das Fenster (samt Inhalt) verschoben werden.

```
"hlstring">"fbgfx.bi"
USING FB

DIM AS EVENT e
DIM AS INTEGER x, y, pressed, col
DIM AS ANY PTR img

' Splashscreen (durchsichtiges Fenster) erzeugen
SCREENRES 384, 64, 32,, GFX_SHAPED_WINDOW

img = IMAGECREATE(48, 8)
' graphischen Schriftzug erzeugen
DRAW STRING img, (0, 0), "GfxLib"
FOR y = 0 TO 63
  FOR x = 0 TO 383
    col = POINT(x \ 8, y \ 8, img)
    IF (col <> RGB(255, 0, 255)) THEN
      col = RGB((x + y) AND &hFF, (x + y) AND &hFF, _
          (x + y) AND &hFF)
    END IF
    PSET (x, y), col
  NEXT x
NEXT y
```

```
pressed = 0
DO
  IF SCREENEVENT(@e) THEN
    SELECT CASE e.type
    CASE EVENT_MOUSE_BUTTON_PRESS
      ' Druck der Maustaste merken
      pressed = -1
    CASE EVENT_MOUSE_BUTTON_RELEASE
      ' Loslassen der Maustaste merken
      pressed = 0
    CASE EVENT_MOUSE_MOVE
      IF pressed THEN
        ' Fenster verschieben
        SCREENCONTROL GET_WINDOW_POS, x, y
        SCREENCONTROL SET_WINDOW_POS, x + e.dx, y + e.dy
      END IF
    END SELECT
  END IF
  SLEEP 5
LOOP WHILE NOT MULTIKEY(1) ' Ende bei Druck der ESC-Taste
```

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC:

- Die Option GET_HIGH_PRIORITY existiert seit FreeBASIC v0.18.4
- SCREENCONTROL existiert seit FreeBASIC v0.17

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht SCREENCONTROL nicht zur Verfügung und kann nur über **__SCREENCONTROL** aufgerufen werden.

Siehe auch:
SCREENRES, SCREENINFO, SCREENEVENT, WINDOWTITLE, VIEW (Grafik), Grafik

Letzte Bearbeitung des Eintrags am 20.10.12 um 23:09:46
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

SCREENCOPY

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » S » **SCREENCOPY**
Syntax: SCREENCOPY [Quellseite][, Zielseite]
Typ: Anweisung
Kategorie: Grafik

SCREENCOPY kopiert den Inhalt einer Bildschirmseite in eine andere.

- 'Quellseite' ist die Seite, die kopiert werden soll. Wird dieser Parameter ausgelassen, nimmt FreeBASIC automatisch die Nummer der aktiven Bildschirmseite an.
- 'Zielseite' ist die Seite, auf welche die Daten kopiert werden sollen. Die Daten auf dieser Seite werden dabei überschrieben. Wird dieser Parameter ausgelassen, nimmt FreeBASIC automatisch die Nummer der sichtbaren Bildschirmseite an.
- Beide Werte dürfen im Bereich von 0 bis Zahl_der_Seiten - 1 liegen. Zahl_der_Seiten ist die Zahl der Seiten, die durch den letzen SCREENRES-Aufruf festgelegt wurde.

Wenn durch eine vorhergehende VIEW-Anweisung ein Darstellungsfeld definiert wurde, wird nur dieses kopiert; ansonsten kopiert SCREENCOPY die gesamte Bildschirmseite.

Mit SCREENCOPY können Sie in FreeBASIC double buffering realisieren; diese Technik wird zur Erzeugung flimmerfreier Animationen benutzt.

SCREENCOPY funktioniert mit jedem Bildschirmmodus, der mit mehreren Seiten initiiert wurde. Bei einseitigen Modi hat SCREENCOPY keinen Effekt.

Im Grafikmodus bewirken die Befehle FLIP und PCOPY dasselbe wie SCREENCOPY.

Beispiel:

```
ScreenRes 320, 240, 32, 2  ' Fenster mit 320x240 Pixeln und 32bit
Farbtiefe und zwei Bildseiten

For n As Integer = 50 To 270
  ScreenSet 1, 0           ' eine Seite anzeigen, während die andere
bearbeitet wird
  Cls
  Circle (n, 50), 50 , RGB(255, 255, 0) ' gelben Kreis auf die aktive
Seite zeichnen
  ScreenSet 0, 0           ' die aktive Seite auf die sichtbare Seite
einstellen
  ScreenSync               ' auf die Bildschirmaktualisierung warten
  ScreenCopy 1, 0          ' Kreis von der vorher aktiven Seite auf die
sichtbare Seite kopieren

  Sleep 25
Next

Print "Taste druecken um zu beenden."
Sleep
```

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC:

Die Möglichkeit, mit SCREENCOPY nur den durch VIEW (Grafik) definierten Darstellungsbereich zu kopieren, besteht seit FreeBASIC v0.13.

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht SCREENCOPY nicht zur Verfügung und kann nur über **__SCREENCOPY** aufgerufen werden.

Siehe auch:
SCREENRES, SCREENSET, SCREENSYNC, FLIP, PCOPY, Grafik

Letzte Bearbeitung des Eintrags am 14.05.14 um 19:36:48
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

SCREENEVENT

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » S » **SCREENEVENT**
Syntax: SCREENEVENT [(Ereignis)]
Typ: Funktion
Kategorie: Benutzereingabe

SCREENEVENT gibt Informationen zu einem Systemereignis zurück, welches das Grafikfenster betrifft. Ein Systemereignis ist eine Maus- oder Tastaturaktivität. Das Ereignis wird nur ausgewertet, wenn das Programmfenster den Fokus hat.

- 'Ereignis' ist ein Pointer (ANY PTR) auf einen Datenpuffer, in dem die Informationen gespeichert werden sollen. Ausgegeben werden die Daten in Form des UDTs 'EVENT'. Wird dieser Parameter ausgelassen, so wird nichts kopiert.
- Der Rückgabewert ist entweder -1, wenn Ereignisse aufgetreten sind, oder 0, wenn dies nicht der Fall ist.

Der Typ 'EVENT' ist folgendermaßen definiert:

```
TYPE EVENT FIELD = 1
  type         AS INTEGER
  UNION
    TYPE
      scancode AS INTEGER
      ascii    AS INTEGER
    END TYPE
    TYPE
      x        AS INTEGER
      y        AS INTEGER
      dx       AS INTEGER
      dy       AS INTEGER
    END TYPE
    button     AS INTEGER
    z          AS INTEGER
    w          AS INTEGER
  END UNION
END TYPE
```

Diese Definition kann direkt aus der fbgfx.bi übernommen werden; durch einfaches Einbinden der Datei mittels "reflinkicon" href="temp0269.html">NAMESPACEs 'FB') verfügbar.

Sofern verfügbar werden die Informationen über das Systemereignis an die Adresse kopiert, auf die der Pointer 'Ereignis' zeigt. Dieser Pointer muss auf einen Speicherbereich zeigen, der mindestens 20 Bytes lang ist, da sonst Programmdaten überschrieben werden können, was zu unerwünschten Ergebnissen wie Programmabsturz führen kann.

In die ersten vier Bytes des Puffers wird im INTEGER-Format die ID des Ereignisses geschrieben, d. h. eine Zahl, die angibt, welches Ereignis aufgetreten ist. Welche Zahl welches Ereignis beschreibt, kann aus der Tabelle unten eingesehen werden.
Die restlichen 16 Bytes (vier INTEGER-Stellen) werden mit Informationen befüllt, die vom Typ des Ereignisses abhängig sind.
Generell ist es sinnvoll, auf den Speicherbereich wie auf einen EVENT PTR zuzugreifen, da so aussagekräftige Bezeichner für die Speicherstellen zur Verfügung stehen.

Wenn der Parameter 'Ereignis' ausgelassen wird, können keine Informationen kopiert werden; daher werden auch keine Ereignisse aus der Gfxlib-Ereignistabelle entfernt. Ein Aufruf von SCREENEVENT ohne Parameter ist daher eine gute Möglichkeit, um zu prüfen, ob ein Ereignis aufgetreten ist, ohne diese tatsächlich abzufragen und auszuwerten, da der Rückgabewert angibt, ob ein relevantes Ereignis aufgetreten ist.

Die Bedeutung des Records '.type' wird im Folgenden aufgelistet. Bitte beachten Sie, dass die angegebenen Konstanten in der fbgfx.bi ebenfalls Element des NAMESPACEs 'FB' sind.

Ereignisnummer	Konstante in der fbgfx.bi	Bedeutung
1	EVENT_KEY_PRESS	Dieses Ereignis wird zurückgegeben, wenn eine Taste auf der Tastatur gedrückt wurde. In diesem Fall enthält der Record '.scancode' den plattformunabhängigen Scancode der Taste. Ist der Taste ein ASCII-Code zugeordnet, so kann dieser aus dem Record '.ascii' gelesen werden, andernfalls hat '.ascii' den Wert 0.
2	EVENT_KEY_RELEASE	Eine gedrückte Taste wurde wieder losgelassen. Die Records '.scancode' und '.ascii' werden in gleicher Weise ausgefüllt wie bei EVENT_KEY_PRESS.
3	EVENT_KEY_REPEAT	Eine Taste wird so lange gedrückt gehalten, bis sie als wiederholter Tastenanschlag behandelt wird. Die Records '.scancode' und '.ascii' werden in gleicher Weise ausgefüllt wie bei EVENT_KEY_PRESS.
4	EVENT_MOUSE_MOVE	Die Maus wurde bewegt, während sie sich auf dem Programmfenster befand. Die Records '.x' und '.y' enthalten die neuen Koordinaten des Mauszeigers. Die Records '.dx' und '.dy' enthalten die Differenz der alten Koordinaten zu den Neuen, geben also an, wie weit die Maus bewegt wurde.
5	EVENT_MOUSE_BUTTON_PRESS	Einer der Mausbuttons wurde gedrückt. Der Record '.button' gibt an, welcher das war; 1 symbolisiert dabei den linken, 2 den rechten und 3 den mittleren Mausbutton. Sie können auch die in der fbgfx.bi vordefinierten Konstanten 'BUTTON_LEFT', 'BUTTON_RIGHT' und 'BUTTON_MIDDLE' verwenden. Bitte beachten Sie, dass auch diese Konstanten Elemente des NAMESPACEs 'FB' sind.
6	EVENT_MOUSE_BUTTON_RELEASE	Einer der Mausbuttons wurde wieder losgelassen. Der Record '.button' wird in gleicher Weise ausgefüllt wie bei EVENT_MOUSE_BUTTON_PRESS.
7	EVENT_MOUSE_DOUBLE_CLICK	Einer der Mausbuttons wurde doppelt angeklickt. Der Record '.button' wird in gleicher Weise ausgefüllt wie bei EVENT_MOUSE_BUTTON_PRESS.
8	EVENT_MOUSE_WHEEL	

		Das Mausrad wurde benutzt. Die neue Position des Mausrads wird im Record '.z' eingetragen.
9	EVENT_MOUSE_ENTER	Die Maus wurde in das Programmfenster bewegt.
10	EVENT_MOUSE_EXIT	Die Maus wurde aus dem Programmfenster bewegt.
11	EVENT_WINDOW_GOT_FOCUS	Das Programmfenster hat den Fokus bekommen (es wurde also zum aktiven Fenster).
12	EVENT_WINDOW_LOST_FOCUS	Das Programmfenster hat den Fokus verloren (es ist also in den Hintergrund getreten, da ein anderes Fenster jetzt den Fokus hat).
13	EVENT_WINDOW_CLOSE	Der User hat versucht das Fenster zu schließen (z. B. über den Schließen-Button in der Titelleiste oder über das Kontextmenü des Fensters in der Taskleiste).
14	EVENT_MOUSE_HWHEEL	Das *horizontale* Mausrad wurde benutzt. Die neue Position des Mausrads wird im Record '.w' eingetragen.

Beispiel:

```
"hlstring">"fbgfx.bi"

Dim evt As FB.EVENT

Function x_Button(x As Integer) As Integer 'welcher Button
  If     x = FB.BUTTON_LEFT  Then
    Print "Die Linke";
  ElseIf x = FB.BUTTON_RIGHT Then
    Print "Die rechte";
  ElseIf x = FB.BUTTON_MIDDLE Then
    Print "Die mittlere";
  ElseIf x = FB.BUTTON_X1 Then
    Print "Die X1";
  ElseIf x = FB.BUTTON_X2 Then
    Print "Die X2";
  End If
  Return 0
End Function

ScreenRes 640, 480
Width 640\8, 480\16

Do
  If ScreenEvent(@evt) Then
    Select Case evt.type
      Case FB.EVENT_KEY_PRESS
        If evt.scancode = FB.SC_ESCAPE Then End
        If evt.ascii > 0 Then
          Print "'" & evt.ascii & "'";
        Else
          Print "eine unbekannte Taste";
        End If
        Print " wurde gedrueckt (Scancode " & evt.scancode & ")"
```

```
      Case FB.EVENT_KEY_RELEASE
        If evt.ascii > 0 Then
          Print "'" & evt.ascii & "'";
        Else
          Print "eine unbekannte Taste";
        End If
        Print " wurde losgelassen (Scancode " & evt.scancode & ")"

      Case FB.EVENT_KEY_REPEAT
        If evt.ascii > 0 Then
          Print "'" & evt.ascii & "'";
        Else
          Print "eine unbekannte Taste";
        End If
        Print " wird gehalten (Scancode " & evt.scancode & ")"

      Case FB.EVENT_MOUSE_MOVE
        Print "Maus wurde bewegt nach " & evt.x & ", " & evt.y & _
        " (delta " & evt.dx & ", " & evt.dy & ")"

      Case FB.EVENT_MOUSE_DOUBLE_CLICK
        Locate CsrLin -2,1
        x_Button(evt.button)
        Print " Maustaste wurde doppelt gedrueckt"

      Case FB.EVENT_MOUSE_BUTTON_PRESS
        x_Button(evt.button)
        Print " Maustaste wurde gedrueckt"

      Case FB.EVENT_MOUSE_BUTTON_RELEASE
        x_Button(evt.button)
        Print " Maustaste wurde losgelassen"

      Case FB.EVENT_MOUSE_WHEEL
        Print "Das Mausrad wurde verstellt auf " & evt.z

      Case FB.EVENT_MOUSE_ENTER
        Print "Die Maus wurde in das Programmfenster bewegt"

      Case FB.EVENT_MOUSE_EXIT
        Print "Die Maus wurde aus dem Programmfenster bewegt"

      Case FB.EVENT_WINDOW_GOT_FOCUS
        Print "Das Programmfenster hat den Fokus erhalten"

      Case FB.EVENT_WINDOW_LOST_FOCUS
        Print "Das Programmfenster hat den Fokus verloren"

      Case FB.EVENT_WINDOW_CLOSE
        End
    End Select
  End If
```

```
  Sleep 10
Loop
```

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC: existiert seit FreeBASIC v0.17

Siehe auch:
SCREENRES, MULTIKEY, INKEY, GETMOUSE, NAMESPACE, USING (NAMESPACE), Grafik

Letzte Bearbeitung des Eintrags am 29.08.13 um 00:20:59
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

SCREENGLPROC

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » S » **SCREENGLPROC**
Syntax: SCREENGLPROC (OpenGL_Prozedurname)
Typ: Funktion
Kategorie: Grafik

SCREENGLPROC ermittelt die Adresse einer OpenGL-Prozedur. Sie wird eingesetzt, um Zeiger zu neuen Funktionen bei OpenGL-Erweiterungen abzufragen.

- 'OpenGL_Prozedurname' ist der Name der Prozedur, deren Adresse abgefragt wird.
- Der Rückgabewert ist ein PROCPTR mit der Adresse der OpenGL-Prozedur. Wenn die Prozedur nicht gefunden wurde, wird NULL (0) zurückgegeben.

Beispiel:

```
"hlstring">"fbgfx.bi"     ' für einige nützliche Definitionen

Dim SwapInterval As Function(ByVal interval As Integer) As Integer
Dim extensions As String

' OpenGL initialisieren und unterstützte Erweiterungen ermitteln
ScreenRes 640, 480, 32,, FB.GFX_OPENGL
ScreenControl FB.GET_GL_EXTENSIONS, extensions

If InStr(extensions, "WGL_EXT_swap_control") <> 0 Then
  ' Erweiterung unterstützt; Adresse der Prozedur ermitteln
  SwapInterval = ScreenGLProc("wglSwapIntervalEXT")
  If SwapInterval <> 0 Then
    ' Adresse ermittelt; mit OpenGL auf vertikale Synchronisation warten
    SwapInterval(1)
  End If
End If
Sleep
```

Unterschiede zu QB: neu in FreeBASIC

Plattformbedingte Unterschiede: wird unter DOS nicht unterstützt

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht SCREENGLPROC nicht zur Verfügung und kann nur über **__SCREENGLPROC** aufgerufen werden.

Siehe auch:
SCREENRES, SCREENSET, Grafik

Letzte Bearbeitung des Eintrags am 21.12.12 um 20:33:34
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

SCREENINFO

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » S » **SCREENINFO**
Syntax: SCREENINFO [Breite][,[Höhe][,[Farbtiefe][,[bpp][,[Pitch][,[Rate][,Treiber]]]]]]
Typ: Anweisung
Kategorie: Grafik

SCREENINFO gibt Informationen über den aktuellen Videomodus zurück. Verfügbar sind unter anderem der Name des Grafiktreibers, die Farbtiefe oder die Bildschirmgröße.

- 'Breite' und 'Höhe' sind INTEGER-Variablen, in welche die Breite bzw. Höhe des Bildschirms in Pixeln zurückgegeben werden.
- 'Farbtiefe' ist ein INTEGER mit dem aktuellen Pixelformat in Bit pro Pixel. Es kann 1, 2, 4, 8, 16 oder 32 sein. Siehe Interne Pixelformate.
- Das INTEGER 'bpp' gibt die Byte pro Pixel zurück.
- Das INTEGER 'Pitch' enthält die Größe einer Framebuffer-Zeile in Byte.
- Das INTEGER 'Rate' gibt die Bildschirmaktualisierungsrate an.
- 'Treiber' ist ein STRING mit dem Namen des aktiven Grafiktreibers, wie "DirectX" oder "X11"

Wenn in dem Zeitpunkt, in dem Sie SCREENINFO aufrufen, kein Grafikfenster aktiv ist, werden die Daten des Desktops angegeben. Als Treiber wird ein Leerstring zurückgegeben.
Wenn SCREENINFO eine Information nicht nachfragen kann, wird in der entsprechenden Variable der Wert 0 gespeichert.
Wenn Sie den Bildschirmmodus via SCREENRES oder SCREEN ändern, sind die gespeicherten Informationen nicht mehr gültig; Sie müssen SCREENINFO erneut aufrufen, um die Daten zu aktualisieren.

Beispiel:

```
DIM AS INTEGER w, h, depth, refresh
DIM driver AS STRING

SCREENRES 400, 300, 32

' Informationen über aktuellen Modus nachfragen:
SCREENINFO w, h, depth, , , refresh, driver

PRINT w & "x" & h & "x" & depth;
IF (refresh > 0) THEN
   PRINT " @ " & refresh & " Hz";
END IF
PRINT " unter Verwendung des Treibers " & driver
SLEEP 2500

' Grafikfenster schließen und Informationen über Desktop nachfragen
SCREEN 0
SCREENINFO w, h, depth
PRINT "Desktopauflösung: " & w & "x" & h & "x" & depth
SLEEP
```

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC: existiert seit FreeBASIC v0.20.0

Unterschiede unter den FB-Dialektformen:

In der Dialektform -lang qb steht SCREENINFO nicht zur Verfügung und kann nur über **__SCREENINFO** aufgerufen werden.

Siehe auch:
SCREENRES, SCREEN (Anweisung), SCREENLIST, Interne Pixelformate, Grafik

Letzte Bearbeitung des Eintrags am 21.12.12 um 20:35:10
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

SCREENLIST

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » S » **SCREENLIST**
Syntax: SCREENLIST ([Farbtiefe])
Typ: Funktion
Kategorie: Grafik

SCREENLIST gibt eine Liste aller unterstützten Bildschirmauflösungen zurück.

- 'Farbtiefe' gibt die Farbtiefe in Bit an, für welche die unterstützten Auflösungen abgefragt werden sollen. Möglich ist die Farbtiefe 8, 15, 16, 24 oder 32. Wird 'Farbtiefe' ausgelassen, dann wird die nächste verfügbare Auflösung der zuletzt abgefragten Farbtiefe zurückgegeben (s. u.).
- Der Rückgabewert ist ein INTEGER, welches die Bildschirmbreite und -höhe der nächsten verfügbaren Auflösung enthält. Steht keine weitere Auflösung zur Verfügung, dann wird 0 zurückgegeben.

SCREENLIST kann verwendet werden, um zur Laufzeit herauszufinden, welche Vollbildmodi auf dem Computer verfügbar sind. Die Funktion arbeitet ähnlich wie DIR: Sie müssen SCREENLIST zuerst mit dem Parameter 'Farbtiefe' aufrufen, und angeben, für welche Farbtiefe die Prüfung durchgeführt werden soll. Ausgegeben wird die erste unterstützte Auflösung für diese Farbtiefe, codiert in einer INTEGER-Zahl. Der nächste Aufruf von SCREENLIST *ohne Parameter* gibt die nächste unterstützte Auflösung im selben Format zurück. Wenn keine weiteren Auflösungen unterstützt werden, ist das Ergebnis 0.

Das obere Word des Ergebnisses (obere 16 Bit, durch HIWORD zurückgegeben) enthält die Breite, und das untere Word (untere 16 Bit, durch LOWORD zurückgegeben) die Höhe.

Die Auflösungen werden von der niedrigsten zur höchsten sortiert ausgegeben. Die Funktion kann zu jedem Zeitpunkt aufgerufen werden, auch wenn noch kein Grafikmodus initiiert wurde.

Beispiel:

```
DIM AS INTEGER modus, breite, hoehe
' Welche 8bit-Auflösungen werden unterstützt?
modus = SCREENLIST(8)
WHILE modus
   breite = HIWORD(modus)
   hoehe = LOWORD(modus)
   PRINT breite & "x" & hoehe
   modus = SCREENLIST
WEND
SLEEP
```

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC: existiert seit FreeBASIC v0.14

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht SCREENLIST nicht zur Verfügung und kann nur über **__SCREENLIST** aufgerufen werden.

Siehe auch:
SCREENRES, Grafik

Letzte Bearbeitung des Eintrags am 21.12.12 um 20:37:03
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

SCREENLOCK

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » S » **SCREENLOCK**
Syntax: SCREENLOCK
Typ: Anweisung
Kategorie: Grafik

SCREENLOCK sperrt den Zugriff auf eine Bildschirmseite.

Die Anweisung SCREENLOCK sperrt den direkten Zugriff auf die aktive Bildschirmseite. Sobald eine Bildschirmseite gesperrt wurde, werden die Daten vom Video-RAM nicht mehr automatisch auf den Bildschirm übertragen, und die Änderungen durch jegliche Grafikbefehle (z.B. Drawing Primitives wie PSET, LINE, CIRCLE, etc.) sind nicht mehr sichtbar, solange die Seite nicht mit SCREENUNLOCK entsperrt wurde.
Während die aktive Seite gesperrt ist, können Sie ihren Speicherbereich frei lesen und beschreiben; Sie müssen die Seite mit SCREENUNLOCK entsperren, um die Änderungen zu aktualisieren.
SCREENLOCK und SCREENUNLOCK müssen immer paarweise verwendet werden. Bei jeder Verwendung von SCREENLOCK wird ein interner Zähler hochgezählt und bei SCREENUNLOCK wieder reduziert. Wenn dieser Zähler auf 0 steht, dann ist die Bildschirmseite entsperrt.

ACHTUNG: Während der Bildschirm gesperrt ist, sollten nur Zeichenbefehle aufgerufen werden. Input/Output und Wartebefehle müssen vermieden werden. Unter Win32 und Linux wird der Bildschirm gesperrt, indem der Thread gestoppt wird, der auch für die Events des Betriebssystems zuständig ist. Wenn der Bildschirm für längere Zeit gesperrt bleibt, kann das System instabil werden. Aus diesem Grund sollten Sie eine Seite so kurz wie möglich gesperrt halten.

Beispiel:

```
' SCREENLOCK/-UNLOCK macht nur mit Grafikscreens Sinn
ScreenRes 300, 100, 32

' ohne SCREENLOCK/-UNLOCK
Do
  Cls
  Locate 2, 2 : Print "Ausgabe"
  Locate 4, 2 : Print "Ein Text"
  Locate 6, 2 : Print "Dieser Text flackert"
  Sleep 1 'Auslastung senken
Loop Until InKey = Chr(32) ' auf Druck der Leertaste warten

'mit SCREENLOCK/-UNLOCK
Do
  ScreenLock
  Cls
  Locate 2, 2 : Print "Ausgabe"
  Locate 4, 2 : Print "Ein Text"
  Locate 6, 2 : Print "Dieser Text flackert nicht"
  ScreenUnLock
  Sleep 1
Loop Until InKey = Chr(32) ' auf Druck der Leertaste warten
```

Unterschiede zu QB: neu in FreeBASIC

Plattformbedingte Unterschiede:
In DOS reagiert der Mauszeiger nicht auf Mausbewegungen, solange der Bildschirm gesperrt ist.

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht SCREENLOCK nicht zur Verfügung und kann nur über **__SCREENLOCK** aufgerufen werden.

Siehe auch:
SCREENRES, SCREENUNLOCK, SCREENPTR, Grafik

Letzte Bearbeitung des Eintrags am 21.12.12 um 21:06:31
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

SCREENPTR

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » S » **SCREENPTR**
Syntax: SCREENPTR
Typ: Funktion
Kategorie: Grafik

SCREENPTR gibt einen Pointer zurück, der auf den Datenbereich (Video-RAM oder Framebuffer, siehe Hinweis unten) der aktiven Bildschirmseite zeigt, oder null, wenn kein Grafikmodus aktiv ist.
SCREENPTR kann dazu verwendet werden, um zu testen, ob ein SCREENRES-Aufruf erfolgreich war, oder um den Videospeicher direkt zu lesen oder zu beschreiben - vorausgesetzt, die Seite wurde zuvor mit SCREENLOCK gesperrt und wird nach dem Speicherzugriff mit SCREENUNLOCK entsperrt.

Um auf die Pixel im Bildschirmpuffer zuzugreifen, müssen die Bildschirmbreite, -höhe, Bytes pro Pixel und Bytes pro Zeile (Pitch) bekannt sein. Diese Informationen können mithilfe von SCREENINFO ermittelt werden.

Der von SCREENPTR zurückgelieferte Pointer ist nur nach einem erfolgreichen SCREENRES-Aufruf gültig und verliert seine Gültigkeit nach jedem weiteren SCREENRES-Aufruf.

Beispiel:

```
' Bildschirmmodus 320x200x8bpp setzen
' Keine Angaben zu Bildschirmseiten => nur eine aktive Seite = sichtbare
Seite
SCREENRES 320, 200

' In 8bpp Modi brauchen Sie einen BYTE PTR für den Speicherzugriff.
DIM framebuffer AS BYTE PTR
framebuffer = SCREENPTR

' Die aktive Seite muss zuerst gesperrt werden, damit ein Zugriff möglich
wird.
SCREENLOCK

' Zeichne einen weißen Pixel an den Koordinaten 160, 100
POKE BYTE, framebuffer + (100 * 320) + 160, 15

' Seite entsperren, damit die Änderungen sichtbar werden
SCREENUNLOCK
SLEEP
```

Hinweis:
Da FreeBASICs Gfxlib intern immer zwei Seiten vorhält (Double Buffering), zeigt SCREENPTR immer auf die zweite Seite, die im Hintergrund zur Bearbeitung bereitliegt. Die vordere Seite dient nur der Anzeige. Dies hat bei der Verwendung allerdings keine Bedeutung, da FreeBASIC automatisch zwischen diesen zwei Seiten wechselt.

Unterschiede zu QB: neu in FreeBASIC

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht SCREENPTR nicht zur Verfügung und kann nur über **__SCREENPTR** aufgerufen werden.

Siehe auch:
SCREENRES, SCREENLOCK, Grafik

Letzte Bearbeitung des Eintrags am 21.12.12 um 22:00:26
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

SCREENRES

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » S » **SCREENRES**
Syntax: SCREENRES (Breite, Höhe[, [Farbtiefe][, [Seitenzahl][, [Flags][, Bildwiederholrate]]]])
Typ: Funktion
Kategorie: Grafik

SCREENRES initiiert ein Grafikfenster. Der Befehl entspricht der SCREEN-Anweisung aus Version 0.10. Die Syntax ist ähnlich dem jetzigen SCREEN-Befehl, jedoch ist es möglich, eine benutzerdefinierte Bildschirmauflösung anzugeben.

- 'Breite' und 'Höhe' sind die Breite und die Höhe des zu initiierenden Grafikfensters.
- 'Farbtiefe' ist die Farbtiefe in Bits pro Pixel (bpp). Wenn Sie diesen Parameter auslassen, wird die Standard-Farbtiefe von 8 bpp (256 Farben) verwendet.
  Achtung: Die Farbtiefe kann im Fenstermodus nicht über der Farbtiefe der Desktopoberfläche liegen; wird dennoch eine höhere Farbtiefe angegeben, setzt FreeBASIC diese automatisch auf die Desktop-Farbtiefe zurück.
- 'Seitenzahl' gibt an, wie viele Bildschirmseiten reserviert werden sollen. Eine Bildschirmseite ist entweder ein Speicherbereich, in dem die Daten gespeichert sind, die auf dem Bildschirm ausgegeben werden sollen (sichtbare Seite), oder ein Datenpuffer derselben Größe wie die der sichtbaren Seite (ein solcher Puffer wird aktive Seite oder Arbeitsseite genannt). Sie können auf einer Seite arbeiten, während die andere angezeigt wird; siehe dazu die SCREENSET-Anweisung. Sie können beliebig viele Bildschirmseiten reservieren (die einzige Grenze stellt der verfügbare Speicher dar). Wenn Sie 'Seitenzahl' nicht angeben, wird nur die aktive Seite (mit der Nummer 0) verfügbar sein.
- 'Flags' gibt verschiedene Modusinformationen an (siehe unten). Wenn dieser Parameter ausgelassen wird, nimmt FreeBASIC automatisch 0 an.
- 'Bildwiederholrate' gibt die Bildwiederholrate in Hz an. Wenn dieser Parameter ausgelassen wird, nimmt FreeBASIC 0 an. Dadurch wird automatisch die günstigste Rate ermittelt. Lassen Sie diesen Parameter am besten einfach aus, außer Sie sind sich völlig im Klaren darüber, was Sie tun - es gibt eigentlich nicht viele Gründe, die vom Benutzer oder System festgelegten Wiederholraten außer Kraft zu setzen. Ganz im Gegenteil, eine Ihrerseits fest gewählte Bildwiederholrate könnte auf einem anderen Bildschirm eventuell gar nicht dargestellt werden!
- Der Rückgabewert ist 0, wenn das Grafikfenster initialisiert werden konnte, und 1, wenn ein Fehler auftrat. Der Rückgabewert kann verworfen werden; SCREENRES wird dann wie eine Anweisung eingesetzt.

VORSICHT: Geben Sie keine Auflösung an, welche die aktuelle Bildschirmgröße übersteigt!
Benutzen Sie im Zweifelsfall vorher SCREENINFO, um die aktuelle Auflösung zu ermitteln.

Hinweis: Falls Sie **Alphatransparenz** nutzen möchten, muss als 'Farbtiefe' mindestens 24-Bit angegeben werden. Des Weiteren muss als 'Flags' der Wert &h40 gesetzt werden oder mit eingebundener fbgfx.bi die Konstante **GFX_ALPHA_PRIMITIVES**, damit *Drawing Primitives* wie PSET, LINE etc. die Alphatransparenz verarbeiten (siehe dazu auch die folgende Beschreibung zu 'Flags').

Bedeutung des Parameters 'Flags'
'Flags' ist einer von folgenden Werten, die durch ein logisches OR miteinander verknüpft werden können. Binden Sie per #INCLUDE die Datei fbgfx.bi ein, um die Bezeichner in der Spalte 'Symbol' verwenden zu können! Bitte beachten Sie, dass die genannten Konstanten Elemente des NAMESPACEs 'FB' sind.

Wert	Symbol	Wirkung
&h00	GFX_WINDOWED	Normaler Fenstermodus Standard-Option
&h01	GFX_FULLSCREEN	Vollbildmodus

&h02	GFX_OPENGL	OpenGL-Modus
&h04	GFX_NO_SWITCH	kein Moduswechsel
&h08	GFX_NO_FRAME	kein Rahmen
&h10	GFX_SHAPED_WINDOW	Splashscreen-Modus
&h20	GFX_ALWAYS_ON_TOP	Fenster, das immer auf oberster Ebene bleibt
&h40	GFX_ALPHA_PRIMITIVES	Bearbeite auch ALPHA-Werte bei Drawing Primitives wie PSET, LINE, etc.
&h80	GFX_HIGH_PRIORITY	Höhere Priorität für Grafikprozesse, nur unter Win32
&h10000	GFX_STENCIL_BUFFER	Stencil Buffer (Schablonenpuffer) verwenden (nur im OpenGL-Modus)
&h20000	GFX_ACCUMULATION_BUFFER	Accumulation Buffer (nur im OpenGL-Modus)
&h40000	GFX_MULTISAMPLE	Bewirkt im Vollbildmodus Antialiasing durch die ARB_multisample-Erweiterung
-1	GFX_NULL	Grafikmodus ohne visuelles Feedback

- Wenn das **Vollbild-Flag** gesetzt ist, wird SCREENRES versuchen, den angegebenen Bildschirmmodus zu initialisieren. Wenn es gesetzt ist und das System den Bildschirm nicht im Vollbildmodus initialisieren kann, wird SCREENRES versuchen, den Bildschirm im Fenstermodus zu initialisieren. Wenn das Vollbild-Flag nicht gesetzt ist und das System den Bildschirm nicht im Fenstermodus initialisieren kann, wird SCREENRES versuchen, den Bildschirm im Vollbildmodus zu initialisieren. Wenn beide Versuche fehlschlagen, hat SCREENRES keine Auswirkungen.
- Wenn das **OpenGL-Flag** gesetzt ist, initiiert die Gfxlib den OpenGL-Modus. Alle Drawing Primitives haben keine Auswirkungen, und der Bildschirm wird nicht automatisch aktualisiert. Im OpenGL-Modus stehen Ihnen nur ein funktionierendes OpenGL-Fenster und die Befehle zum direkten Speicherzugriff auf den VideoRAM zur Verfügung. Wenn Sie eine Bildschirmaktualisierung durchführen wollen, benutzen Sie die Anweisung FLIP.
- Wenn das **Moduswechsel-Flag** gesetzt ist, kann der User nicht mehr mit [ALT]+[ENTER] zwischen Vollbild- und Fenstermodus hin- und herschalten. Der Maximieren-Button am Fenster erscheint als inaktiv.
- Wenn das **Stencil-Buffer-Flag** gesetzt ist, wird ein Stencil Buffer (Schablonenpuffer) erstellt. Dieser Buffer wird verwendet, um Pixel zu maskieren, d. h. die Farbe bestimmter Pixel durch eine andere Farbe zu ersetzen oder diese erst gar nicht zu zeichnen. Man kann dieses Flag nur setzen, wenn man OpenGL verwendet.
- Das **Accumulation-Flag** hat auch nur im OpenGL-Modus eine Wirkung. Dadurch wird ein Accumulation Buffer erstellt. Dieser kann eine höhere Auflösung haben als der Bildschirmmodus. Somit kann man eine Szene z.B. mit einer Auflösung von 1024x768 in den Puffer zeichnen, aber mit einer Auflösung von 800x600 auf den Bildschirm rendern. Dies bewirkt eine Glättung der gesamten Szene (Anti-Aliasing).
- Das **Rahmen-Flag** bewirkt, dass auf dem Bildschirm ein Grafikbereich ohne Titelleiste und Fensterrahmen erscheint. Der User kann dieses Fenster mit der Maus nicht verschieben, und es hat keinen Schließen-, Maximieren- und Minimieren-Button. Allerdings reagiert das Fenster auf die üblichen Tastaturkommandos, wechselt also mit [ALT]+[ENTER] in den Vollbildmodus und wird mit [WINDOWSTASTE]+[D] minimiert. Es erscheint in der Taskleiste, bekommt dort aber kein Kontextmenü. Sinnvollerweise wird dieses Flag nicht mit dem Vollbild-Flag kombiniert.
- Das **Splashscreen-Flag** bewirkt dasselbe wie das Rahmen-Flag. Zusätzlich ist die transparente Farbe (modusabhängig) in diesem Fall tatsächlich transparent, d. h. der Desktop bzw. die Fenster, die sich hinter dem FreeBASIC-Grafikfenster befinden, sind 'durch das Grafikfenster hindurch' sichtbar. Siehe PUT (Grafik) für die Transparenzfarben. Sinnvollerweise wird dieses Flag nicht mit dem Vollbild-Flag kombiniert. Ein Klick auf transparente Flächen wird von SCREENEVENT nicht erkannt; nur 'solide Flächen' werden für die Ereigniserkennung beachtet. Es ist auch möglich, 'durch das Fenster hindurch' Elemente anzuklicken, die sich hinter transparenten Flächen befinden.

- Der **Grafikmodus ohne visuelles Feedback** aktiviert alle Grafikbefehle und reserviert einen entsprechenden Speicherbereich. Auf dem Bildschirm werden diese Änderungen jedoch nicht sichtbar. Mit GET (Grafik) können von dieser virtuellen Bildschirmseite Ausschnitte eingelesen und später in einem sichtbaren Bildschirmmodus mit PUT (Grafik) ausgegeben werden. Nützlich, um Grafiken für betriebssystem-abhängige Systeme (GDI, Gtk etc.) einzubinden. Wird dieses Flag benutzt, können keine weiteren Flags verwendet werden.

Wenn ein Bildschirmmodus einmal eingestellt wurde, können Sie ihn jederzeit durch Drücken von [ALT]+[ENTER] zwischen Vollbild- und Fenstermodus umschalten, vorausgesetzt, das System unterstützt dies und das Umschalten wurde nicht durch Verwendung des Moduswechsel-Flags unterbunden.

Anders als im Konsole-Modus wird ein Programm beim Klick auf den Schließen-Button nicht beendet; stattdessen wird ein CHR(255, 107) in den Tastaturpuffer geschrieben, der mit INKEY abgerufen werden kann. Diese Kombination entspricht [ALT]+[F4]. Ein Klick auf den Maximieren-Button hingegen wird automatisch in den Vollbildmodus wechseln, wenn dieser verfügbar ist.

FreeBASIC wird versuchen, den Bildschirmmodus so gut wie möglich zu initialisieren; wenn alle Versuche fehlschlagen, hat SCREENRES keine Auswirkungen. Die Ausführung des Programms wird mit der Zeile nach der SCREENRES-Anweisung fortgesetzt. Sie sollten daher nachprüfen, ob der Bildschirmmodus erfolgreich initialisiert wurde; dies können Sie mit der SCREENPTR-Funktion tun.

Beispiel 1: ein 320x240 Pixel großes Grafikfenster öffnen und prüfen, ob der Aufruf erfolgreich war

```
SCREENRES 320, 240, 16
IF SCREENPTR = 0 THEN
  PRINT "Bildschirmmodus nicht initialisiert!"
  SLEEP
  END
END IF
'...
SLEEP
```

Beispiel 2: ein 200x200 Pixel großes Grafikfenster im Splashscreen-Modus, immer im Vordergrund
Ob das Fenster korrekt initialisiert werden konnte, wird mithilfe des Rückgabewertes geprüft.

```
"hlstring">"fbgfx.bi"
IF SCREENRES(200, 200, 32,, FB.GFX_SHAPED_WINDOW or FB.GFX_ALWAYS_ON_TOP)
THEN
  PRINT "Fehler: Grafikfenster konnte nicht initialisiert werden!"
  SLEEP
  END
END IF
CIRCLE (100, 100), 50, RGB(255, 0, 255),,,, F

DO
  SLEEP 1
LOOP UNTIL LEN(INKEY)
```

Weitere Auswirkungen von SCREENRES:
Ein erfolgreicher SCREENRES-Aufruf setzt ...

- die Nummer der sichtbaren als auch die der aktiven Bildschirmseite auf 0
- in den indizierten Farbmodi die Palette zurück (siehe Standard-Paletten)
- die Clipping-Bereiche (bestimmt durch die vorhergehenden Anweisungen WINDOW und VIEW

(Grafik)) des Bildschirms zurück
- den Text-Cursor in die obere linke Ecke des Bildschirms
- die Schriftgröße auf 8x8 (dies kann geändert werden, indem mit WIDTH die Anzahl der Zeilen und Spalten festgelegt wird)
- den Grafik-Cursor in die Mitte des Bildschirms
- die Vordergrundfarbe auf Weiß (Farbnummer 15 in einer Farbtiefe bis zu 8bpp bzw. &hFFFFFF in allen höheren Farbtiefen)
- die Hintergrundfarbe auf Schwarz (Farbnummer 0)
- den Mauszeiger auf 'sichtbar' (Sie können den Mauszeiger innerhalb des Fensters mit SETMOUSE verbergen)

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC:
Seit FreeBASIC v0.90.0 lässt sich SCREENRES als Funktion mit Rückgabewert aufrufen.
Einige Bildschirmflags existierten in früheren Versionen nicht oder hatten eine andere Bedeutung:

- GFX_HIGH_PRIORITY (&h80) existiert seit FreeBASIC v0.18
- Folgende Flags existieren erst seit FreeBASIC v0.17: GFX_NO_FRAME (&h08), GFX_ALPHA_PRIMITIVES (&h80) und GFX_MULTISAMPLE (&h40000)
- GFX_STENCIL_BUFFER hatte vor FreeBASIC v0.17 den Wert &h10.
- GFX_ACCUMULATION_BUFFER hatte vor FreeBASIC v0.17 den Wert &h20.

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht SCREENRES nicht zur Verfügung und kann nur über **__SCREENRES** aufgerufen werden.

Siehe auch:
SCREEN (Anweisung), SCREENINFO, SCREENLIST, Grafik

Letzte Bearbeitung des Eintrags am 25.06.13 um 22:07:59
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

SCREENSET

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » S » **SCREENSET**
Syntax: SCREENSET [aktive_Seite][, sichtbare_Seite]
Typ: Anweisung
Kategorie: Grafik

SCREENSET setzt die aktive und die sichtbare Bildschirmseite.

- 'aktive_Seite' ist die neue Nummer der aktiven Seite. Die aktive Seite ist diejenige, auf die gezeichnet wird. Wird dieser Parameter ausgelassen, belässt FreeBASIC den zuvor eingestellten Wert.
- 'sichtbare_Seite' ist die neue Nummer der sichtbaren Seite. Die sichtbare Seite ist diejenige, die angezeigt wird. Wird dieser Parameter ausgelassen, belässt FreeBASIC den zuvor eingestellten Wert.
- Beide Werte dürfen im Bereich von 0 bis Zahl_der_Seiten - 1 liegen. Zahl_der_Seiten ist die Zahl der Seiten, die durch den letzen SCREENRES-Aufruf festgelegt wurde.
- Wenn Sie eines der beiden Argumente auslassen, dann bleibt die dazu gehörige Einstellung unverändert. Werden beide Argumente ausgelassen, dann werden sowohl die aktive als auch die sichtbare Seite auf 0 zurückgesetzt.

SCREENSET ermöglicht es Ihnen, die Nummer der aktuellen aktiven und sichtbaren Bildschirmseite zu bestimmen.
Sie können SCREENSET benutzen, um "page-flipping" oder "double-buffering" zu ermöglichen.

Beispiel:

```
' Bildschirmmodus 320x200x8bpp mit 2 Seiten setzen
' Hintergrundfarbe auf weiß setzen
SCREENRES 320, 200, ,2
COLOR ,15

DIM x AS INTEGER
x = -40

' eine Seite anzeigen, während die andere bearbeitet wird
SCREENSET 1, 0

DO
   CLS
   LINE (x, 80)-(x + 39, 119), 4, BF
   x += 1
   IF (x > 319) THEN x = -40
   ' warte auf Bildschirmaktualisierung
   SCREENSYNC
   ' aktive Seite auf sichtbare kopieren
   SCREENCOPY
LOOP WHILE INKEY = ""
```

Unterschiede zu QB: neu in FreeBASIC

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht SCREENSET nicht zur Verfügung und kann nur über **__SCREENSET** aufgerufen werden.

Siehe auch:

SCREENRES, SCREENCOPY, SCREENSYNC, Grafik

Letzte Bearbeitung des Eintrags am 21.12.12 um 22:30:56
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

SCREENSYNC

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » S » **SCREENSYNC**
Syntax: SCREENSYNC
Typ: Anweisung
Kategorie: Grafik

SCREENSYNC wartet mit der Programmausführung auf eine Bildschirmaktualisierung. Damit wird ein Flimmern des Bildschirms vermieden.
Diese Anweisung soll WAIT ersetzen.

Beispiel: siehe SCREENSET

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC: existiert seit FreeBASIC v0.14

Siehe auch:
SCREENRES, SCREENSET, SCREENCOPY, WAIT, Grafik

Letzte Bearbeitung des Eintrags am 21.12.12 um 22:31:50
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

SCREENUNLOCK

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » S » **SCREENUNLOCK**
Syntax: SCREENUNLOCK [Startzeile][, Endzeile]
Typ: Anweisung
Kategorie: Grafik

SCREENUNLOCK hebt eine Sperrung durch SCREENLOCK auf.

- 'Startzeile' ist ein optionaler Parameter, der die erste Bildschirmzeile (Y-Koordinate) angibt, ab welcher der Bildschirm aktualisiert werden soll. Wenn Sie diesen Parameter auslassen, wird die oberste Bildschirmzeile angenommen.
- 'Endzeile' ist ein optionaler Parameter, der die letzte Bildschirmzeile angibt, die aktualisiert werden soll. Wenn Sie 'Endzeile' auslassen, wird die unterste Bildschirmzeile angenommen.
- Werden beide Parameter ausgelassen, entsperrt FreeBASIC den gesamten Bildschirm.

Diese Anweisung lässt das System den Bildschirm wieder aktualisieren und aktualisiert ihn bereits beim Aufruf selbst. Diese erste Aktualisierung kann auf einen Bildschirmausschnitt begrenzt werden, indem eine Start- und Endzeile angegeben wird. Z. B. kann angegeben werden, dass nur der Bereich aktualisiert werden soll, der tatsächlich verändert wurde.

Sobald eine Sperre aufgelöst wurde, können Sie wieder die Drawing Primitives (einfachste Grafikfunktionen, wie PSET, LINE, CIRCLE etc.) einsetzen; der direkte Zugriff via SCREENPTR ist jedoch nicht mehr möglich.

SCREENLOCK und SCREENUNLOCK müssen immer paarweise verwendet werden. Bei jeder Verwendung von SCREENLOCK wird ein interner Zähler hochgezählt und bei SCREENUNLOCK wieder reduziert. Wenn dieser Zähler auf 0 steht, dann ist die Bildschirmseite entsperrt.

ACHTUNG: Während der Bildschirm gesperrt ist, sollten nur Zeichenbefehle aufgerufen werden. Input/Output und Wartebefehle müssen vermieden werden. Unter Win32 und Linux wird der Bildschirm gesperrt, indem der Thread gestoppt wird, der auch für die Events des Betriebssystems zuständig ist. Wenn der Bildschirm für längere Zeit gesperrt bleibt, kann das System instabil werden.

Beispiel:

```
' SCREENLOCK/-UNLOCK macht nur mit Grafikscreens Sinn
ScreenRes 300, 100, 32

' ohne SCREENLOCK/-UNLOCK
Do
  Cls
  Locate 2, 2 : Print "Ausgabe"
  Locate 4, 2 : Print "Ein Text"
  Locate 6, 2 : Print "Dieser Text flackert"
  Sleep 1 'Auslastung senken
Loop Until InKey = Chr(32) ' auf Druck der Leertaste warten

'mit SCREENLOCK/-UNLOCK
Do
  ScreenLock
  Cls
  Locate 2, 2 : Print "Ausgabe"
```

```
  Locate 4, 2 : Print "Ein Text"
  Locate 6, 2 : Print "Dieser Text flackert nicht"
  ScreenUnLock
  Sleep 1
Loop Until InKey = Chr(32) ' auf Druck der Leertaste warten
```

Unterschiede zu QB: neu in FreeBASIC

Plattformbedingte Unterschiede:
In DOS reagiert der Mauszeiger nicht auf Mausbewegungen, solange der Bildschirm gesperrt ist.

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht SCREENUNLOCK nicht zur Verfügung und kann nur über **__SCREENUNLOCK** aufgerufen werden.

Siehe auch:
SCREENRES, SCREENLOCK, SCREENPTR, Grafik

Letzte Bearbeitung des Eintrags am 21.12.12 um 22:33:55
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

SCRN

Siehe OPEN SCRN

Letzte Bearbeitung des Eintrags am 28.08.11 um 17:46:38
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

SECOND

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » S » **SECOND**
Syntax: SECOND (Serial)
Typ: Funktion
Kategorie: Datum und Zeit

Anmerkung:
Um diese Funktion nutzen zu können, muss die Datei datetime.bi in Ihren Quellcode eingebunden werden, z. B. mit INCLUDE. Alternativ können Sie auch die Datei vbcompat.bi einbinden, da dadurch auch automatisch datetime.bi eingebunden wird.

SECOND extrahiert die Sekunde einer Serial Number.

- 'Serial' ist ein DOUBLE-Wert, der als Serial Number behandelt wird.
- Der Rückgabewert ist die Sekunde, die in der Serial Number gespeichert ist.

Beispiel:
Die aktuelle Sekunde aus NOW extrahieren:

```
"hlstring">"vbcompat.bi"
DIM sek AS INTEGER
sek = SECOND(NOW)
```

Unterschiede zu QB: existiert nur in QBX PDS und in VBDOS.

Unterschiede zu früheren Versionen von FreeBASIC: existiert seit FreeBASIC v0.15

Siehe auch:
NOW, TIMESERIAL, TIMEVALUE, HOUR, MINUTE, FORMAT, Serial Numbers, Datum und Zeit

Letzte Bearbeitung des Eintrags am 27.12.12 um 00:59:05
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

SEEK (Anweisung)

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » S » **SEEK (Anweisung)**
Syntax: SEEK (Dateinummer, Position)
Typ: Anweisung
Kategorie: Dateien

SEEK setzt die Position des Zeigers innerhalb einer Datei.

- 'Dateinummer' ist die Nummer der Datei, die durch eine OPEN-Anweisung zugewiesen wurde.
- 'Position' ist die neue Position des Cursors. Sie ist entweder die Position in Bytes, wenn die Datei im BINARY-Modus bzw. in einem der sequentiellen Modi (INPUT, OUTPUT und APPEND) geöffnet wurde, oder die Nummer des Datensatzes für den RANDOM-Modus. Der Positionswert 1 setzt den Zeiger auf den Anfang der Datei.

Beispiel:
Cursor auf das 100ste Byte in der Datei setzen:

```
DIM AS INTEGER f = FREEFILE
OPEN "file.ext" FOR BINARY AS "hlkw0">SEEK f, 100
CLOSE "reflinkicon" href="temp0371.html">SEEK (Funktion), OPEN, Dateien
(Files)
```

Letzte Bearbeitung des Eintrags am 27.12.12 um 21:33:39
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

SEEK (Funktion)

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » S » **SEEK (Funktion)**
Syntax: Position = SEEK (Dateinummer)
Typ: Funktion
Kategorie: Dateien

SEEK gibt die Position des Zeigers innerhalb einer Datei zurück.

- 'Dateinummer' ist die Nummer der Datei, die durch eine OPEN-Anweisung zugewiesen wurde.
- Der Rückgabewert ist entweder die Position in Bytes, wenn die Datei im BINARY-Modus bzw. in einem der sequentiellen Modi (INPUT, OUTPUT und APPEND) geöffnet wurde, oder die Nummer des Datensatzes für den RANDOM-Modus. Zu Beginn steht der Dateizeiger auf 1, nach dem Lesen eines Bytes (bzw. eines Datensatzes) auf Position 1 usw. (im Gegensatz dazu startet LOC mit der Position 0)

Beispiel:

```
DIM AS INTEGER f = FREEFILE, position
OPEN "file.ext" FOR BINARY AS "hlkommentar">'... Programmcode,
Lese/Schreibe-Anweisungen, etc...
position = SEEK(f)
'...weitere Anweisungen
CLOSE "reflinkicon" href="temp0370.html">SEEK (Anweisung), OPEN, LOC,
Dateien (Files)
```

Letzte Bearbeitung des Eintrags am 17.06.13 um 00:40:10
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

SELECT CASE

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » S » **SELECT CASE**
Syntax:

```
SELECT CASE [AS CONST] Ausdruck
CASE (Ausdrucksliste)
  ' Code
[CASE (Ausdrucksliste2)
  ' Code ]
[ ... ]
[CASE ELSE
  ' Code ]
END SELECT
```

Typ: Anweisung
Kategorie: Programmablauf

SELECT CASE führt, abhängig vom Wert eines Ausdrucks, bestimmte Codeteile aus.

- 'Ausdruck' ist eine beliebige, aber sinnvolle Aneinanderreihung von Konstanten, Variablen, Operatoren und Funktionen, die einen Zahlenwert oder einen String ergeben. Je nach Wert dieses Ausdrucks wird der Code auf verschiedene Art und Weise fortgesetzt.
- 'Ausdrucksliste', 'Ausdrucksliste2' usw. enthalten ebenso wie 'Ausdruck' Konstanten, Variablen, Operatoren und Funktionen, die einen Wert ergeben. Dabei können pro CASE-Zeile mehrere Ausdrücke und auch Vergleichsoperatoren angegeben werden. Die dazu verwendete Syntax wird weiter unten erläutert. Der Datentyp von 'Ausdrucksliste' muss derselbe sein wie in 'Ausdruck'.
- 'Code' stellt normalen FreeBASIC-Programmcode dar, der unter gegebenen Voraussetzungen (dem Wert von 'Ausdruck' und dessen Verhältnis zu 'Ausdrucksliste') ausgeführt wird.
- Entspricht 'Ausdruck' der ersten 'Ausdrucksliste', so führt FreeBASIC den Code zwischen CASE Ausdrucksliste und CASE Ausdrucksliste2 aus. Ist dies nicht der Fall, prüft FreeBASIC das Verhältnis zwischen 'Ausdruck' und der zweiten, dritten, ... 'Ausdrucksliste', und verfährt dort genauso. Wird keine der angegebenen Bedingungen erfüllt, führt FreeBASIC den Code hinter CASE ELSE aus, sofern diese Zeile existiert. Ansonsten wird gar kein Code ausgeführt und die Programmausführung wird hinter dem SELECT-Block normal fortgesetzt.
- Auch wenn mehrere Bedingungen erfüllt sind, wird immer nur der Code hinter der ersten zutreffenden CASE-Zeile ausgeführt; nachfolgende Bedingungen werden ignoriert.
- Die Klausel 'AS CONST' bewirkt, dass die Abfrage wesentlich schneller durchgeführt wird, schränkt aber die Wahl der Parameter in den Ausdruckslisten ein. Details hierzu sind weiter unten aufgeführt.

SELECT CASE stellt eine etwas übersichtlichere Form dar von:

```
IF Ausdruck = Ausdrucksliste1 THEN
   ' Code
[ ELSEIF Ausdruck = Ausdrucksliste2 THEN
   ' Code ]
[ ... ]
[ ELSE
   ' Code ]
END IF
```

Da nicht in jeder Zeile erneut 'Ausdruck =' geschrieben werden muss, erspart sich der Programmierer Tipparbeit. Außerdem ist schneller ersichtlich, welche Bedingung geprüft wird.

Der in 'Ausdruck' bzw. 'Ausdrucksliste' verwendete Datentyp kann eine Ganzzahl (z. B. INTEGER), Gleitkommazahl (z. B. DOUBLE) oder ein STRING, ZSTRING oder WSTRING sein. Nicht zulässig sind hingegen UDTs. UDT-Records hingegen dürfen selbstverständlich benutzt werden, sofern sie von einem einfachen Datentyp sind. Siehe dazu auch die FB-Datentypen.

Als Block-Anweisung initialisiert SELECT CASE einen SCOPE-Block, der mit der Zeile END SELECT endet. Variablen, die nach einem 'CASE Ausdrucksliste' deklariert werden, sind nur bis zum Ende des SELECT-Blocks gültig.

SELECT-CASE-Blocks dürfen bis zu einem beliebigen Level ineinander verschachtelt werden.

Syntax einer Ausdrucksliste (hinter CASE):

```
{ Ausdruck [TO Ausdruck] | IS (Bedingungsoperator) Ausdruck} [, ...]
```

- 'Ausdruck' ist ebenso wie in SELECT CASE eine sinnvolle Aneinanderreihung von Konstanten, Variablen, Operatoren und Funktionen, die einen Wert ergeben müssen, dessen Datentyp derselbe ist, wie vom Ausdruck in der SELECT-CASE-Zeile vorgegeben.
- Wird nur ein einziger Ausdruck angegeben, so prüft FreeBASIC auf Gleichheit der Werte.
- Sind mehrere Ausdrücke, getrennt durch Kommata angegeben, so prüft FreeBASIC jeden einzelnen davon auf Gleichheit; die Bedingung ist erfüllt, wenn mindestens ein Ausdruck in der Liste gleich ist.
- Mit TO wird ein Wertebereich angegeben; links vom TO muss dabei der kleinere, rechts der größere Wert stehen. Die Bedingung ist dann erfüllt, wenn der Ausdruck in der SELECT-CASE-Zeile zwischen den in der CASE-Zeile angegebenen Ausdrücken liegt. Die Werte dieser Ausdrücke sind dabei einzuschließen. CASE 1 TO 3 ist also erfüllt, wenn Ausdruck den Wert 1, 2 oder 3 annimmt. Bei STRINGs wird dabei zeichenweise der ASCII-Wert der Zeichen verglichen (siehe dazu ASC).
- Wird das Schlüsselwort 'IS' eingesetzt, so kann ein eigener Vergleichsoperator angegeben werden. Die Bedingung ist dann nicht mehr erfüllt, wenn die Werte in der SELECT-CASE-Zeile und der CASE-Zeile gleich sind, sondern z. B. ungleich, größer als, usw.

Beispiele für Ausdrucksliste

```
SELECT CASE a
' ist a = 5?
  CASE 5
' ist a ein Wert von 5 bis 10?
' Die kleinere Zahl muss zuerst angegeben werden
  CASE 5 TO 10
' ist a größer als 5?
  CASE IS > 5
' ist a gleich 1 oder ein Wert von 3 bis 10?
  CASE 1, 3 TO 10
' ist a gleich 1, 3, 5, 7 oder b + 8?
  CASE 1, 3, 5, 7, b + 8
END SELECT
```

Beispiel für die Benutzung von SELECT CASE:

```
DIM AS INTEGER wahl
INPUT "Geben Sie eine Zahl von 1 bis 10 ein: ", wahl
SELECT CASE wahl
  CASE 1
    PRINT "Sie haben 1 angegeben"
  CASE 2
```

```
    PRINT "Die Zahl ist 2"
  CASE 3, 4
    PRINT "Die Zahl ist 3 oder 4"
  CASE 5 TO 10
    PRINT "Die Zahl liegt im Bereich von 5 bis 10"
  CASE IS > 10
    PRINT "Die Zahl ist größer als 10"
  CASE ELSE
    PRINT "Die Zahl ist kleiner als 1"
END SELECT
SLEEP
```

Die Zeileneinrückungen sind optional; sie dienen lediglich der besseren Übersichtlichkeit. In Programmiererkreisen sind sie in jedem Falle gern gesehen.

Wenn eine Bedingung doppelt vorkommt, wird nur der erste Codeblock ausgeführt.

Beispiel:

```
SELECT CASE a
  CASE 1
    PRINT "a ist 1"
  CASE 1, 2
    PRINT "a ist entweder 1 oder 2"
END SELECT
```

Die Zeile "a ist entweder 1 oder 2" wird nur angezeigt, wenn a = 2 ist. Im anderen Fall a = 1 wird nur die Zeile "a ist 1" ausgegeben.

Beachten Sie bitte, dass die Ausdrücke in der Reihenfolge abgearbeitet werden, in der sie angegeben werden. Im obigen Beispiel wird also zuerst geprüft, ob a = 1 ist, danach ob a = 2 ist, usw. Wenn sehr häufig solche Prüfungen stattfinden, sollten Sie daher das wahrscheinlichste Ergebnis an die Spitze der Liste stellen, um nicht zu viel Zeit zu verlieren.

In frühen Formen der Sprachfamilie BASIC wurde das Konstrukt ON ... GOTO bzw. ON ... GOSUB eingesetzt, um solche Prüfstrukturen umzusetzen. Dies gilt inzwischen als veraltete und umständliche Technik.

SELECT CASE AS CONST
Bei dieser Version wird der Code effizienter umgesetzt, so dass das Programm die Entscheidungen schneller treffen kann. Allerdings gelten hier bestimmte Beschränkungen für 'Ausdruck' und 'Ausdrucksliste':

- 'Ausdruck' muss vom Typ INTEGER oder UINTEGER sein.
- 'Ausdrucksliste' darf nur Konstanten oder einfache Ausdrücke enthalten, also Ausdrücke, die keine Variablen enthalten.
- Funktionen sind in 'Ausdrucksliste' nur dann erlaubt, sofern es sich um FreeBASICs eigene mathematische Funktionen handelt (wie etwa FIX, ABS, SGN, etc.), der Rückgabetyp INTEGER oder UINTEGER ist und der Parameter eine Konstante oder ein einfacher Ausdruck wie oben beschrieben ist.
- Der Operator IS steht nicht zur Verfügung.
- In 'Ausdrucksliste' kann mit 4097 aufeinanderfolgenden (U)INTEGER-Werten verglichen werden, also z. B. im Bereich von 0 bis 4096 oder von 4 bis 4100.

CASE ELSE ist mit SELECT CASE AS CONST zulässig.

Durch diese Einschränkungen ist die Abfrage um einiges schneller als das normale SELECT CASE.
Man spricht von 'spürbaren' Unterschieden bei mehr als vier CASE-Zeilen.

Unterschiede zu QB:

- Die Variante SELECT CASE AS CONST ist neu in FreeBASIC.
- In einer Bereichsangabe 'Ausdruck TO Ausdruck' wertet QB immer beide Ausdrücke aus, auch wenn der erste Ausdruck bereits größer ist als der zu prüfende Wert.

Unterschiede zu früheren Versionen von FreeBASIC:

- Seit FreeBASIC v0.16 können CASE-Anweisungsblöcke manuell über EXIT SELECT verlassen werden.
- Seit FreeBASIC v0.16 initialisiert SELECT CASE einen SCOPE-Block.
- Bitfelder können in CASE-Zeilen seit FreeBASIC v0.16 eingesetzt werden.
- Die Variante SELECT CASE AS CONST existiert seit FreeBASIC v0.12

Siehe auch:
IF ... THEN, IIF, ON ... GOTO, ON ... GOSUB, SCOPE, EXIT, CONST, CONST (Klausel), Ausdrücke, Bedingungsstrukturen

Letzte Bearbeitung des Eintrags am 09.08.13 um 19:14:58
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

SETDATE

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » S » **SETDATE**
Syntax: SETDATE NeuesDatum
Typ: Anweisung
Kategorie: Datum und Zeit

SETDATE ändert das Systemdatum.

- 'NeuesDatum' ist ein STRING, der eines der folgenden Formate besitzen muss:
  mm/dd/yyyy
  mm-dd-yyyy
  mm/dd/yy
  mm-dd-yy
- 'mm' ist dabei der Monat,
- 'dd' der Tag
- 'yy' bzw. 'yyyy' ist das Jahr.

Hinweis: Abhängig vom System bleibt SETDATE unter Umständen ohne Auswirkungen.

Beispiel:

```
DIM AS STRING month, day, year
month = "03"        ' März
day = "13"          ' der 13
year = "1994"       ' die guten alten Zeiten...
SETDATE month & "/" & day & "/" & year
```

Unterschiede zu QB:
In QB kann das Systemdatum (nur) mit DATE$ = NeuesDatum zugewiesen werden.

Siehe auch:
DATE, TIME, TIMER, SETTIME, Datum und Zeit

Letzte Bearbeitung des Eintrags am 21.12.12 um 23:15:18
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

SETENVIRON

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » S » **SETENVIRON**
Syntax: SETENVIRON "Variable=Wert"
Typ: Anweisung
Kategorie: System

SETENVIRON verändert die Systemumgebungsvariablen.

- 'Variable' ist die Systemumgebungsvariable, die mit dem neuen Wert 'Wert' belegt werden soll.
- Der Rückgabewert ist 0, wenn die Änderung erfolgreich durchgeführt werden konnte, oder ungleich 0, wenn ein Fehler auftrat.

Beispiel Die Systemvariable "PATH" mit "C:" belegen:

```
SUB zeigePATH
  ' Den Wert von "PATH" anzeigen
  "hlkw0">__FB_PCOS__
    SHELL "SET PATH"
  "hlkw0">__FB_UNIX__
    SHELL "echo $PATH"
  "hlkw0">END SUB

zeigePATH
SETENVIRON "PATH=C:"  ' diese Zuweisung macht unter Linux wenig Sinn ...
zeigePATH
SLEEP
```

Hinweis: Die Systemvariable hätte auch mit ENVIRON abgefragt werden können.

Unterschiede zu QB:
Dieser neue Befehl ersetzt den in QB verwendeten Befehl ENVIRON.

Plattformbedingte Unterschiede:
Unter Linux muss die Systemvariable 'case sensitive' erfolgen. Windows und DOS sind 'case insensitive'.

Siehe auch:
ENVIRON, SHELL, Betriebssystem-Anweisungen

Letzte Bearbeitung des Eintrags am 27.12.12 um 21:34:37
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

SETMOUSE

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » S » **SETMOUSE**
Syntax: SETMOUSE [x][,[y][, [cursor][, clip]]]
Typ: Anweisung
Kategorie: Benutzereingabe

SETMOUSE setzt die Koordinaten des Mauscursors und bestimmt, ob die Maus sichtbar oder unsichtbar ist.

- 'x' und 'y' sind die neuen Koordinaten des Mauscursors.
- 'cursor' ist 1, wenn die Maus angezeigt werden soll, oder 0, wenn sie unsichtbar sein soll.
- 'clip' ist 1, wenn die Mausbewegung auf das Fenster begrenzt ist, oder 0, wenn der Mauscursor das Fenster verlassen kann.
- Wird einer der Parameter ausgelassen, behält FreeBASIC den alten Status bei.

Beispiel:

```
Dim As Integer x, y, buttons, clip
'erstellt eine Fenster 800x600, 32-Bit Farbe, 1 Seite
Screenres 800, 600, 32, 1

Do
  ' Mausposition ermitteln und ausgeben.
  GetMouse x, y , , buttons, clip
  Locate 1, 1
  PRINT x, y, Bin(buttons, 3), clip

  If buttons = 1 Then     'linke Maustaste
    SetMouse 400, 300, 1, 0 'Mauspfeil sichtbar und zentrieren
  End If
  If buttons = 2 Then     'rechte Maustaste
    SetMouse 400, 300,1,1 'Mauspfeil auf das Fenster begrenzt
  End If
  If buttons = 4 Then     'mittlere Maustaste
    SetMouse 400, 300, 0, 0 'Mauspfeil unsichtbar
  End If
Loop Until MultiKey(1)
```

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC:

- Maus-Clipping wird seit FreeBASIC v0.18 unterstützt.
- Seit FreeBASIC 0.14 wird SETMOUSE im Konsolenfenster unterstützt.

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht SETMOUSE nicht zur Verfügung und kann nur über **__SETMOUSE** aufgerufen werden.

Siehe auch:
SCREENRES, GETMOUSE, Benutzereingaben

Letzte Bearbeitung des Eintrags am 22.12.12 um 00:22:29
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

SETTIME

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » S » **SETTIME**
Syntax: SETTIME Uhrzeit
Typ: Anweisung
Kategorie: Datum und Zeit

SETTIME setzt die neue Systemzeit.

- 'Uhrzeit' ist ein STRING, der die neue Zeit angibt. Er muss dazu das Format hh:mm:ss oder hh:mm oder hh besitzen.
- 'hh' sind dabei die Stunden,
- 'mm' die Minuten
- 'ss' die Sekunden der einzustellenden Zeit.

Hinweis: Abhängig vom System bleibt SETTIME unter Umständen ohne Auswirkungen.

Beispiel:

```
"hlstring">"vbcompat.bi"
DIM AS DOUBLE alteZeit = NOW, neueZeit

' Uhrzeit drei Srunden vorstellen
neueZeit = DATEADD("h", 3, alteZeit)
SETTIME FORMAT(neueZeit, "hh:mm:ss")
PRINT "Die Zeit wurde auf " & FORMAT(neueZeit, "hh:mm:ss") & "
vorgestellt."

SLEEP 5000, 1

' wieder zurueckstellen
neueZeit = DATEADD("s", 5, alteZeit)
SETTIME FORMAT(neueZeit, "hh:mm:ss")
PRINT "Die Zeit wurde auf " & FORMAT(neueZeit, "hh:mm:ss") & "
zurueckgestellt."

SLEEP
```

Unterschiede zu QB:
In QB kann die Systemzeit (nur) mit TIME$ = Uhrzeit zugewiesen werden.

Unterschiede zu früheren Versionen von FreeBASIC:

- existiert seit FreeBASIC v0.10 für Windows
- existiert seit FreeBASIC v0.13 für Linux

Siehe auch:
TIME, DATE, TIMER, SETDATE, Datum und Zeit

Letzte Bearbeitung des Eintrags am 27.12.12 um 21:35:04
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

SGN

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » S » **SGN**
Syntax: SGN (Zahl)
Typ: Funktion
Kategorie: Mathematik

SGN gibt einen Wert aus, der das Vorzeichen eines Ausdrucks identifiziert.

- 'Zahl' ist eine beliebiger numerischer Ausdruck. Variablen, Konstanten, Operatoren und Funktionen sind erlaubt. Der Ausdruck darf von jedem Datentyp außer STRING, ZSTRING oder WSTRING sein.
- Der Rückgabewert ist ein INTEGER. Er hat den Wert 1, wenn 'Zahl' größer als null ist, den Wert 0, wenn 'Zahl' gleich null ist, und den Wert -1, wenn 'Zahl' kleiner als null ist.

SGN kann mithilfe von OPERATOR überladen werden.

Beispiel:

```
DIM x AS DOUBLE
INPUT "Geben Sie bitte eine Zahl ein: ", x
IF SGN(x) = -1 THEN
   PRINT "Ihre Zahl war negativ"
ELSEIF SGN(x) = 1 THEN
   PRINT "Ihre Zahl war positiv"
ELSE
   PRINT "Ihre Zahl war null"
END IF
```

Ausgabebeispiel:

```
Geben Sie bitte eine Zahl ein: 8.41
Ihre Zahl war positiv
```

Siehe auch:
ABS, mathematische Funktionen

Letzte Bearbeitung des Eintrags am 27.12.12 um 21:35:26
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

SHARED

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » S » **SHARED**
Syntax A: { DIM | REDIM | COMMON | STATIC } SHARED Variable AS Typ [, Variable AS Typ ...]
Syntax B: { DIM | REDIM | COMMON | STATIC } SHARED AS Typ Variablenliste
Typ: Schlüsselwort
Kategorie: Deklaration

SHARED wird zusammen mit DIM / REDIM, COMMON und STATIC eingesetzt und bewirkt, dass Variablen sowohl auf Modulebene als auch innerhalb von Prozeduren (SUBs und FUNCTIONs) verfügbar sind.

DIM SHARED und STATIC SHARED bewirken dabei nur Zugriffsrechte für Prozeduren innerhalb desselben Moduls, während auf die Variablen, die mit COMMON SHARED deklariert werden, von jeder Prozedur und von jedem Modul aus zugegriffen werden kann.
DIM und STATIC sind in Verbindung mit SHARED gleichbedeutend, was auf die Eigenschaften von STATIC zurückzuführen ist. STATIC SHARED ist faktisch nur aus Kompatibilitätsgründen zu QuickBASIC vorhanden.

Beispiel:

```
DECLARE SUB MySub ()

DIM SHARED x AS INTEGER
DIM y AS INTEGER

x = 10
y = 5
PRINT "x ist:";x
PRINT "y ist:";y
PRINT

MySub
SLEEP

SUB MySub
   PRINT "x ist: "; x
   PRINT "y ist: "; y  ' fehlerhafter Aufruf
END SUB
```

Dieser Code erzeugt beim Compilieren mit der Option -lang fb (Standard) einen Fehler, weil 'y' in der SUB MySub nicht deklariert wurde.

In der Sub MySub kann durchaus mit einer Variable mit dem Namen 'y' gearbeitet werden:

```
DECLARE SUB MySub ()

DIM SHARED x AS INTEGER
DIM y AS INTEGER

x = 10
y = 5
PRINT "x ist:"; x
PRINT "y ist:"; y
```

```
PRINT

MySub
PRINT

PRINT "x ist:"; x
PRINT "y ist:"; y
SLEEP

SUB MySub
  DIM y AS INTEGER = 15
  PRINT "Innerhalb der Prozedur:"
  PRINT "x ist:"; x
  PRINT "y ist:"; y
  x = 3
END SUB
```

Ausgabe:

```
x ist: 10
y ist: 5

Innerhalb der Prozedur:
x ist: 10
y ist: 15

x ist: 3
y ist: 5
```

Durch SHARED wird eine Variable von einer lokalen zu einer globalen Variable. Das bedeutet, dass sie von jedem beliebigen Punkt im Programm, also von jedem Unterprogramm aus, gelesen und auch geändert werden kann.
Eine lokale Variable hingegen ist immer nur in der aktuellen Prozedur verfügbar. Die Variable mit der Bezeichnung 'y' in MySub ist von der Variable mit der Bezeichnung 'y' im Hauptmodul vollkommen unabhängig.
Um eine lokale Variable einer Prozedur zugänglich zu machen, muss sie in der Parameterliste der Prozedur erwähnt werden; siehe dazu Prozeduren (Parameterübergabe), SUBs, FUNCTIONs.

Hinweis zur expliziten Variableninitialisierung:
Da SHARED-Variablen zur Compilezeit umgesetzt werden, können direkt bei der Deklaration nur Konstanten zur expliziten Initialisierung zugewiesen werden.

```
Dim Shared As Integer variable = 20
```

Da Integer-Literale (Zahlen) immer zu Konstanten umgesetzt werden, ist die oben stehende Deklaration mit expliziter Initialisierung möglich, wohingegen Folgendes nicht compiliert (Compilezeit-Fehler):

```
Dim As Integer x = 20
Dim Shared As Integer variable = x
```

Stattdessen müssen Deklaration und Wertzuweisung der Integer-Variablen variable aufgeteilt werden, sodass variable zunächst in der zweiten Zeile implizit mit 0 (Konstante) initialisiert wird:

```
Dim As Integer x = 20
```

```
Dim Shared As Integer variable
variable = 20 + x
```

String-Variablen, die SHARED deklariert wurden, können nie initialisiert werden, auch nicht mit konstanten Strings. Dies ist auf die interne Struktur eines variablen Strings in FreeBASIC zurückzuführen.

Hinweis zu Constructoren von Objekten:
Wird eine Variable eines UDTs erzeugt und mit SHARED global zugänglich gemacht, wird das Objekt noch vor Programmbeginn initiiert. Dies hat gerade dann Bedeutung, wenn der CONSTRUCTOR des Objekts bestimmte Arbeiten übernehmen soll. Die exakte Reihenfolge kann anhand des folgenden Beispiels gesehen werden:

```
Type foo
  bar As Integer
  Declare Constructor ()
End Type
Constructor foo ()
  Print "Ctor"
End Constructor

Sub Start1 Constructor 101
  Print "Start1"
End Sub
Sub Start2 Constructor
  Print "Start2"
End Sub

Print "Hauptprogramm startet"
Print "Deklariere Variable"

Dim Shared As foo bar

Print "Hauptprogramm beendet"
Sleep
```

Ausgabe:

```
Start1
Ctor
Start2
Hauptprogramm startet
Deklariere Variable
Hauptprogramm beendet
```

Soll die Initialisierung dagegen nicht zu Programmbeginn, sondern an einer bestimmten (späteren) Stelle im Programmablauf erfolgen, so kann dies durch die Verwendung von Pointern erzwungen werden. Der Grund hierfür ist, dass Pointervariablen erst initialisiert werden, wenn mit NEW ein Objekt erstellt wird.

Unterschiede zu QB:

- SHARED ist unter FreeBASIC nur noch ein Schlüsselwort, kein eigener Befehl; es funktioniert nur im Zusammenhang mit DIM, REDIM oder COMMON.
- Die alternative Syntax DIM SHARED AS Typ Variablenliste (Syntax B) ist neu in FreeBASIC.

Siehe auch:
DIM, REDIM, COMMON, Datentypen und Deklarationen

Letzte Bearbeitung des Eintrags am 20.12.13 um 17:09:10
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

SHELL

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » S » **SHELL**
Syntax: SHELL [Kommando]
Typ: Funktion
Kategorie: System

SHELL führt ein Systemkommando aus und gibt die Kontrolle an das aufrufende Programm zurück, sobald das aufgerufene Kommando abgearbeitet wurde.

- 'Kommando' ist ein STRING, der als Befehlszeile an die Konsole übergeben wird. Dies kann der Name eines Programms inklusive Parametern oder ein Systemkommando sein.
- Der Rückgabewert ist 0, wenn der Aufruf erfolgreich war, oder -1, wenn ein Fehler aufgetreten ist. Ein Fehler kann z. B. sein, dass die angegebene Datei oder der Befehl nicht existiert.

Beispiel (Windows):

```
SHELL "DIR C:\*.*"
```

Beispiel (Linux):

```
SHELL "ls -alh"
```

Beispiel zum Aufruf als Funktion:

```
DIM AS INTEGER Result
Result = SHELL ("NOTEPAD.EXE")
Print "Rueckgabe war " & Result & "."
Sleep
```

Soll das aufgerufene Programm parallel zum FB-Programm laufen, kann unter Win32 "Start" gestartet werden:

```
SHELL "START notepad"
```

Unterschiede zu QB:

- SHELL kann in FreeBASIC auch als Funktion eingesetzt werden.
- In FreeBASIC wird ohne Parameterangabe keine CMD-Sitzung gestartet.

Plattformbedingte Unterschiede:

- Unter Linux muss das Kommando 'case sensitive' erfolgen. Windows und DOS sind 'case insensitive'.
- Das Trennzeichen für den Dateipfad ist unter Linux der vorwärtsgerichtete Slash /. Windows verwendet den Backslash \, erlaubt aber auch den Slash. DOS verwendet den Backslash.

Siehe auch:
RUN, EXEC, CHAIN, END, COMMAND, Betriebssystem-Anweisungen

Letzte Bearbeitung des Eintrags am 22.12.12 um 14:08:29
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

SHL

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » S » **SHL**
Syntax A: Ergebnis = Zahl1 SHL Zahl2
Syntax B: Zahl1 SHL= Zahl2
Typ: Operator
Kategorie: Mathematik

SHL verschiebt alle Bits in der Variablen um eine bestimmte Stellenzahl nach links.

- 'Zahl1' und 'Zahl2' sind INTEGER-Werte. Wird SHL mit Gleitkommazahlen (SINGLE oder DOUBLE) verwendet, so werden diese zuerst mithilfe von CINT mathematisch gerundet.
- Beim kombinierten SHL (Syntax B) wird der Wert von 'Zahl1' um 'Zahl2' Stellen nach links verschoben; es ist die Kurzform für Zahl1 = Zahl1 SHL Zahl2
- Bits, die über die letzte Stelle hinaus verschoben werden, gehen verloren.

SHL ist für einen INTEGER gleichwertig mit der Multiplikation einer Zweierpotenz; der Exponent dabei ist 'Zahl2'.

```
Ergebnis = Zahl1 SHL Zahl2
```

entspricht also

```
Ergebnis = Zahl1 * (2 ^ Zahl2)
```

SHL wird dabei aber schneller durchgeführt als der oben genannte Ausdruck, selbst wenn (2 ^ Zahl2) durch seinen Wert als Konstante ersetzt wird. SHL wird daher zur Optimierung des Codes verwendet.

SHL ist die Gegenfunktion zu SHR. Sie kann mithilfe von OPERATOR überladen werden.

Beispiel:

```
DIM AS UBYTE  a, b, c
DIM AS SINGLE d, e, f

a = 3
b = 2
c = a SHL b

PRINT BIN(a, 8); " wird um "; b; " Stellen nach links";
PRINT " verschoben. Das Ergebnis ist "; BIN(c, 8); "."
c SHL= 5
PRINT "Um fuenf weitere Stellen nach links verschoben";
PRINT " ergibt sich "; BIN(c, 8); "."

d = 2.9
e = 2.4
f = d SHL e

PRINT BIN(d, 8); " wird um "; e; " Stellen nach links";
PRINT " verschoben. Das Ergebnis ist "; BIN(f, 8); "."
f SHL= 5
PRINT "Um fuenf weitere Stelle nach links verschoben";
PRINT " ergibt sich "; BIN(f); "."
```

```
SLEEP
```

Ausgabe:

```
00000011 wird um 2 Stellen nach links verschoben. Das Ergebnis ist
00001100.
Um fuenf weitere Stelle nach links verschoben ergibt sich 10000000.
00000011 wird um  2.4 Stellen nach links verschoben. Das Ergebnis ist
00001100.
Um fuenf weitere Stelle nach links verschoben ergibt sich 110000000.
```

Ein UBYTE hat nur acht Bits. Deswegen geht bei der Verschiebung von &h00001100 um fünf Stellen nach links das erste Bit verloren, wodurch das Ergebnis &h10000000 zustande kommt.
Die Variable 'd' wird hier zu 3 auf-, die Variable 'e' zu 2 abgerundet. Deshalb ergeben sich auch für die SINGLE-Zeilen dieselben Werte wie bei den UBYTE-Zeilen. Da für SINGLE-Variablen aber 32 Bit anstelle von 8 reserviert werden, geht das erste Bit nicht verloren. So kommt das Ergebnis &h110000000 anstelle von &h10000000 zustande.
Dieser Code bietet sich zum Experimentieren an; ändern Sie einfach die Zahlenwerte für a, b, c, d, e und f.

Unterschiede zu QB: neu in FreeBASIC

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht SHL nicht zur Verfügung und kann nur über **__SHL** aufgerufen werden.

Siehe auch:
SHR, MOD, mathematische Funktionen

Letzte Bearbeitung des Eintrags am 22.12.12 um 14:51:40
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

SHORT

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » S » **SHORT**
Typ: Datentyp

SHORT-Zahlen sind vorzeichenbehaftete 16-bit-Ganzzahlen. Sie liegen im Bereich von -(2^15) bis (2^15)-1, bzw. von -32768 bis 32767.

Unterschiede zu QB:
Der Datentyp SHORT ist neu in FreeBASIC; er hat dieselben Eigenschaften wie der Datentyp INTEGER in QB.

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht SHORT nicht zur Verfügung und kann nur über **__SHORT** aufgerufen werden.

Siehe auch:
DIM, CAST, CSHORT, Datentypen, Datentypen und Deklarationen

Letzte Bearbeitung des Eintrags am 28.12.12 um 00:43:03
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

SHR

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » S » **SHR**
Syntax A: Ergebnis = Zahl1 SHR Zahl2
Syntax B: Zahl1 SHR= Zahl2
Typ: Operator
Kategorie: Mathematik

SHR verschiebt alle Bits in der Variablen um eine bestimmte Stellenzahl nach rechts.

- 'Zahl1' und 'Zahl2' sind INTEGER-Werte. Wird SHR mit Gleitkommazahlen (SINGLE oder DOUBLE) verwendet, so werden diese zuerst mithilfe von CINT mathematisch gerundet.
- Beim kombinierten SHR (Syntax B) wird der Wert von 'Zahl1' um 'Zahl2' Stellen nach rechts verschoben; es ist die Kurzform für Zahl1 = Zahl1 SHR Zahl2
- Bits, die über die letzte Stelle hinaus verschoben werden, gehen verloren.

SHR ist für einen INTEGER gleichwertig mit der Division durch eine Zweierpotenz; der Exponent dabei ist 'Zahl2'.

```
Ergebnis = Zahl1 SHR Zahl2
```

entspricht also

```
Ergebnis = Zahl1 \ (2 ^ Zahl2)
```

SHR wird dabei aber schneller durchgeführt als der oben genannte Ausdruck, selbst wenn (2^Zahl2) durch seinen Wert als Konstante ersetzt wird. SHR wird daher zur Optimierung des Codes verwendet.

SHR ist die Gegenfunktion zu SHL. Sie kann mithilfe von OPERATOR überladen werden.

Beispiel:

```
DIM AS INTEGER a, b, c
DIM AS SINGLE  d, e, f

a = 12
b = 2
c = a SHR b

PRINT BIN(a, 8); " wird um "; b; " Stellen nach rechts verschoben. Das
Ergebnis ist "; BIN(c, 8); "."
c SHR= 1
PRINT "Um eine weitere Stelle nach rechts verschoben ergibt sich ";
BIN(c, 8); "."

d = 11.5
e = 2.4
f = d SHR e

PRINT BIN(d, 8); " wird um "; e; " Stellen nach rechts verschoben. Das
Ergebnis ist "; BIN(f, 8); "."
f SHR= 1
PRINT "Um eine weitere Stelle nach rechts verschoben ergibt sich ";
```

```
BIN(f, 8); "."
SLEEP
```

Ausgabe:

```
00001100 wird um 2 Stellen nach rechts verschoben. Das Ergebnis ist
00000011.
Um eine weitere Stelle nach rechts verschoben ergibt sich 00000001.
00001100 wird um 2.4 Stellen nach rechts verschoben. Das Ergebnis ist
00000011.
Um eine weitere Stelle nach rechts verschoben ergibt sich 00000001.
```

Bei der Verschiebung von &h00000011 um eine Stelle nach rechts geht das letzte Bit verloren; deswegen ist das Ergebnis &h00000001. Die Variable 'd' wird hier zu 12 auf-, die Variable 'e' zu 2 abgerundet. Deshalb ergeben sich auch für die SINGLE-Zeilen dieselben Werte.
Dieser Code bietet sich zum Experimentieren an; ändern Sie einfach die Zahlenwerte für a, b, c, d, e und f.

Unterschiede zu QB: neu in FreeBASIC

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht SHR nicht zur Verfügung und kann nur über **__SHR** aufgerufen werden.

Siehe auch:
SHL, MOD, mathematische Funktionen

Letzte Bearbeitung des Eintrags am 22.12.12 um 14:51:04
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

SIN

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » S » **SIN**
Syntax: SIN (winkel)
Typ: Funktion
Kategorie: Mathematik

SIN gibt den Sinus eines Winkels 'winkel' im Bogenmaß zurück. Der Sinus zu einem Winkel ist die relative Höhe innerhalb des Normkreises zu seinem Mittelpunkt (Wikipedia-Artikel zum Thema).

- 'winkel' ist ein beliebiger numerischer Ausdruck. Variablen, Konstanten, Operatoren und Funktionen sind erlaubt. Der Ausdruck darf von jedem Datentyp außer STRING, ZSTRING oder WSTRING sein.
- Der Rückgabewert ist ein DOUBLE-Wert, der den Sinus-Wert zum Winkel enthält.

Die Umkehrfunktion zu SIN lautet ASIN.

SIN kann mithilfe von OPERATOR überladen werden.

Beispiel:

```
CONST pi AS DOUBLE = ACOS(0)*2
DIM a AS DOUBLE
DIM r AS DOUBLE

INPUT "Bitte geben Sie einen Winkel in Grad (DEG) ein: ", a
r = a * PI / 180   ' Grad in Bogenmaß umrechnen
PRINT ""
PRINT "Der Sinus eines" ; a; "°-Winkels ist ";
PRINT USING ""; SIN(r)
SLEEP
```

Ausgabe:

```
Bitte geben Sie einen Winkel in Grad (DEG) ein: 30

Der Sinus eines 30°-Winkels ist 0.50
```

Unterschiede zu früheren Versionen von FreeBASIC:
Die Überladung von SIN für benutzerdefinierte Datentypen ist seit FreeBASIC v0.22 möglich.

Siehe auch:
ASIN, COS, ACOS, TAN, ATN, ATAN2, mathematische Funktionen

Letzte Bearbeitung des Eintrags am 24.12.12 um 18:46:24
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

SINGLE

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » S » **SINGLE**
Typ: Datentyp

SINGLE bedeutet einfache Genauigkeit. Variablen vom Typ SINGLE sind vorzeichenbehaftete 32-bit-Gleitkommazahlen.
Damit lassen sich Zahlen mit circa 6 Stellen darstellen, der Wertebereich für SINGLE-Zahlen liegt bei positiven Zahlen zwischen 1.401298e-45 und 3.402823e+38 und bei negativen zwischen -1.401298e-45 und -3.402823e+38.

SINGLE werden benutzt, um Dezimalzahlen zu speichern, die nicht sehr genau sein müssen. Sie verhalten sich ähnlich wie DOUBLE-Zahlen, sind jedoch nicht so genau.

Beispiel:

```
DIM a AS SINGLE
a = 0.123456
PRINT a
SLEEP
```

Siehe auch:
DIM, CAST, CSNG, Datentypen, Datentypen und Deklarationen

Letzte Bearbeitung des Eintrags am 28.12.12 um 00:44:42
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

SIZEOF

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » S » **SIZEOF**
Syntax: SIZEOF (Variable)
Typ: Funktion
Kategorie: Datentypen

SIZEOF gibt die Größe einer Struktur im Speicher in Bytes aus.

- Diese Struktur kann eine Variable (a, b(), ...) sein, ein Datentyp (z. B. INTEGER) oder ein UDT-Bezeichner (siehe TYPE).
- Wird SIZEOF an einem String variabler Länge angewandt, wird die Größe des Typs ausgegeben. Dieser beträgt bei Strings 12 oder 24, abhängig davon, ob der 32bit oder 64bit Compiler verwendet wird.
- Wird SIZEOF an einem String fester Länge oder an einem ZSTRING bzw. WSTRING angewandt, wird die Größe des reservierten Speicherbereichs ausgegeben.

Das Ergebnis von SIZEOF ist *oft* gleich dem von LEN, jedoch **nicht immer!**

Wird SIZEOF auf Arrays angewendet, gibt es immer nur die Größe eines Elements zurück, nicht des ganzen Arrays. Um Missverständnisse zu vermeiden, sollte SIZEOF deswegen immer nur auf einzelne Elemente statt auf das ganze Array angewandt werden.

Beispiel:

```
DIM a AS INTEGER, b AS STRING, c AS STRING * 5, d(5) AS INTEGER

PRINT "1) LEN"
PRINT LEN(a)       ' Ausgabe: 4 oder 8 => 32bit bzw. 64bit-Integer, je
nach Compiler
PRINT LEN(b)       ' Ausgabe: 0 => Leerstring
PRINT LEN(c)       ' Ausgabe: 5 => 5-Zeichen-String
PRINT LEN(d(0))    ' Ausgabe: 4 oder 8 => 32bit bzw. 64bit-Integer, je
nach Compiler
PRINT LEN(d)       ' Ausgabe: 4 oder 8 => Pointer auf Array, also
32bit-Integer bzw. 64bit-Integer, je nach Compiler
PRINT
PRINT "2) SIZEOF"
PRINT SIZEOF(a)    ' Ausgabe: 4 oder 8 => 32bit bzw. 64bit-Integer, je
nach Compiler
PRINT SIZEOF(b)    ' Ausgabe: 12 oder 24 => Siehe STRING (Datentyp) für
Erklärung
PRINT SIZEOF(c)    ' Ausgabe: 6  => 5 Zeichen + 1 Nullzeichen CHR(0)
PRINT SIZEOF(d(0)) ' Ausgabe: 4 oder 8 => 32bit bzw. 64bit-Integer, je
nach Compiler
PRINT SIZEOF(d)    ' Ausgabe: 4 oder 8 => Pointer auf Array, also
32bit-Integer bzw. 64bit-Integer, je nach Compiler
SLEEP
```

Unterschiede zu QB: neu in FreeBASIC

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht SIZEOF nicht zur Verfügung und kann nur über **__SIZEOF** aufgerufen werden.

Siehe auch:
LEN, OFFSETOF, STRING (Datentyp), Verschiedenes

Letzte Bearbeitung des Eintrags am 11.01.14 um 23:10:58
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

SLEEP

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » S » **SLEEP**
Syntax: SLEEP [Zeit] [, Flag]
Typ: Anweisung
Kategorie: Programmablauf

SLEEP wartet eine bestimmte Zeit oder bis eine Taste gedrückt wird.

- 'Zeit' ist die Zahl der Millisekunden, die vergehen sollen, bis das Programm fortgesetzt wird.
- Wird 'Zeit ausgelassen, wartet FreeBASIC mit der Programmfortsetzung bis zum nächsten Tastendruck.
- Auch wenn eine Wartezeit angegeben wurde, setzt FreeBASIC bei einem Tastendruck die Programmausführung fort, selbst wenn die Wartezeit noch nicht verstrichen ist.
- Wenn für 'Flag' 1 angegeben wird, reagiert SLEEP nicht auf Tastendrücke; es wird wirklich gewartet, bis die Zeit verstrichen ist. Für Zeiteinheiten unter 100 Millisekunden wird dieses Flag allerdings ignoriert.
- Wird SLEEP ohne 'Zeit', aber mit 'Flag' aufgerufen, wird ein Fehler erzeugt.

Achtung: Der Tastaturpuffer wird durch SLEEP nicht geleert! Er muss nach einem Aufruf von SLEEP erst neu geleert werden! Alternativ kann, wenn einfach nur auf eine Taste gewartet werden soll, GETKEY verwendet werden.

Hinweis: Angegebene Wartezeiten und tatsächliche Wartezeiten können insbesondere bei sehr kurzen Zeitangaben stark voneinander abweichen! Die Genauigkeit beträgt:

- 15 ms unter Windows NT / 2000 / XP
- 50 ms unter Windows 9x / Me
- 10 ms unter Linux
- 55 ms unter DOS

SLEEP **sollte in Schleifen regelmäßig** (am besten in jedem Durchgang) bis zu 25 ms warten (hierfür bietet sich ein simples "SLEEP 1" an), um anderen Prozessen die Gelegenheit zu geben, Berechnungen durchzuführen. Dies wirkt sich positiv auf die Rechnerperformance und auch auf die Performance des eigenen Programms aus.

Beispiel:

```
PRINT "Drücken Sie bitte eine Taste"
SLEEP
PRINT "Sie haben "; INKEY ; " gedrückt."
PRINT "Warte 5 Sekunden"
SLEEP 5000
```

Unterschiede zu QB:

- Die zu wartende Zeit wird in Millisekunden angegeben (außer in -lang qb; siehe unten). In QB wird die angegebene Zeit in Sekunden gewartet.
- Das optionale Flag zum Ignorieren der Tastendrücke ist neu in FreeBASIC.

Unterschiede zu früheren Versionen von FreeBASIC:

- Seit FreeBASIC v0.18.3 wartet SLEEP in der Dialektform -lang qb die angegebene Zeit in Sekunden (siehe unten).
- Das optionale Flag zum Ignorieren der Tastendrücke existiert seit FreeBASIC v0.15

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb wartet SLEEP wie unter QB die angegebene Zeit in Sekunden. Wird 'Flag' angegeben oder **__SLEEP** verwendet, dann wird die angegebene Zeit in Millisekunden gewartet.

Siehe auch:
GETKEY, TIMER, INKEY, Datum und Zeit

Letzte Bearbeitung des Eintrags am 27.12.12 um 21:38:18
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

SPACE

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » S » **SPACE**
Syntax: SPACE[$](n)
Typ: Funktion
Kategorie: Stringfunktionen

SPACE gibt einen String zurück, der aus *'n'* Leerzeichen besteht.

Das Dollarzeichen ($) als Suffix ist optional.

Beispiel:

```
DIM s AS STRING
s = SPACE(6)
PRINT "nicht eingerueckter Text"
PRINT s & "eingerueckter Text"
SLEEP
```

Unterschiede unter den FB-Dialektformen:

- In der Dialektform -lang qb ist das Suffix $ verbindlich.
- In den Dialektformen -lang fblite und -lang fb ist das Suffix optional.

Siehe auch:
STRING (Funktion), TAB, SPC, WSPACE, String-Funktionen

Letzte Bearbeitung des Eintrags am 27.12.12 um 21:38:44
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

SPC

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » S » **SPC**
Syntax: SPC (n)
Typ: Anweisung
Kategorie: Konsole

SPC wird im Zusammenhang mit PRINT oder PRINT # verwendet, um Texteinrückungen zu erzeugen. Der Cursor wird um 'n' Zeichen weiter gerückt. Der dazwischen liegende Text wird dabei nicht überschrieben, sondern nur der Cursor neu positioniert.

Befindet sich die Anweisung SPC am Ende der PRINT-Anweisung, so wird anschließend kein Zeilenumbruch durchgeführt.

Beispiel:

```
Dim As String A1, B1, A2, B2

A1 = "Jana"
B1 = "Bauer"
A2 = "Robert"
B2 = "Meier"

Print String(35, ".");
Locate , 1
Print "VORNAME"; Spc(35 - 10); "NACHNAME"
Print "-------"; Spc(35 - 10); "--------"
Print A1; Spc(35 - Len(A1)); B1
Print A2; Spc(35 - Len(A2)); B2

Sleep
```

Ausgabe:

```
VORNAME.....................NACHNAME
-------                     --------
Jana                           Bauer
Robert                         Meier
```

Unterschiede zu QB:
In QB wird der Zwischenraum mit Leerzeichen überschrieben. In FreeBASIC kann dieses Verhalten mit SPACE erreicht werden.

Unterschiede zu früheren Versionen von FreeBASIC:
Seit FreeBASIC v0.21 wird SPC auch von PRINT USING unterstützt.

Siehe auch:
PRINT (Anweisung), SPACE, TAB, LOCATE (Anweisung), Konsole

Letzte Bearbeitung des Eintrags am 25.02.12 um 01:32:50
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

SQR

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » S » **SQR**
Syntax: SQR (Wert)
Typ: Funktion
Kategorie: Mathematik

SQR gibt die Quadratwurzel (square root) von 'Wert' aus. SQR hat dieselbe Funktion wie Wert^(1/2).

'Wert' ist ein beliebiger numerischer Ausdruck. Variablen, Konstanten, Operatoren und Funktionen sind erlaubt. Der Ausdruck darf von jedem Zahlendatentyp (INTEGER, DOUBLE usw.) sein.

Beispiel:

```
PRINT SQR(9)       ' Ausgabe 3
PRINT 9 ^ (1/2)    ' Ausgabe 3
PRINT SQR(20 ^ 2)  ' Ausgabe 20
SLEEP
```

Siehe auch:
mathematische Funktionen

Letzte Bearbeitung des Eintrags am 22.12.12 um 19:26:30
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

STATIC (Anweisung)

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » S » **STATIC (Anweisung)**
Syntax A: STATIC Variable AS Typ [= einfacherAusdruck] [, Variable AS Typ [= einfacherAusdruck] [, ...]]
Syntax B: STATIC AS Typ Variable [= einfacherAusdruck] [, Variable [= einfacherAusdruck] [, ...]]
Syntax C: STATIC Array([Start TO] Ende [, [Start TO] Ende] [, ...]) AS Typ [= {Ausdruck [, ...] }]
Typ: Anweisung
Kategorie: Deklaration

STATIC deklariert Variablen oder Arrays einer lokalen Prozedur und speichert sie, wenn die Prozedur beendet wird, sodass ihr Wert beim nächsten Aufruf der Prozedur wieder verfügbar ist. Die Anweisung arbeitet damit ähnlich wie DIM, nur dass der Wert für den nächsten Prozeduraufruf erhalten bleibt.

- 'Variable' bzw. 'Array' ist ein Bezeichner für eine Variable oder ein Array, die/das in der lokalen Prozedur dimensioniert und gespeichert werden soll.
- 'Typ' ist der Datentyp der Variable / des Arrays. Siehe Datentypen.
- Ebenso wie bei DIM ist es möglich, Variablen-Initiatoren zu verwenden.

Beispiel 1:

```
DECLARE SUB zaehl

zaehl
zaehl
SLEEP

SUB zaehl
  STATIC i AS INTEGER
  i += 1
  PRINT "Dies ist der " & i & ". Aufruf."
END SUB
```

Ausgabe:

```
Dies ist der 1. Aufruf.
Dies ist der 2. Aufruf.
```

STATIC kann auch den Wert einer Variable initialisieren. In diesem Fall wird die Wertzuweisung des Initiators nur beim ersten Aufruf der Prozedur vorgenommen. Bei späteren Aufrufen wird mit dem gespeicherten Wert weitergearbeitet.

Beispiel 2:

```
FUNCTION produkt (param AS INTEGER) AS INTEGER
  STATIC wert AS INTEGER = 1

  wert *= param
  RETURN wert
END FUNCTION

PRINT produkt(3)
PRINT produkt(4)
PRINT produkt(5)
```

```
SLEEP
```

Ausgabe:

```
 3
 12
 60
```

STATIC kann auch in UDTs verwendet werden, wodurch sich alle Instanzen der Klasse den Wert teilen. Die Variable verhält sich also, als ob sie SHARED wäre, allerdings nur für die Objekte der Klasse selbst.
Eine statische Variable in einer Klasse muss auch einmalig außerhalb der Klasse deklariert werden. Siehe dazu folgendes Beispiel:

Beispiel 3:

```
Type Teil
  Private:
  Static As Integer anzahl
  Dim As String teileName

  Public:
  Declare Constructor (bezeichnung As String)
  Declare Function wieviele As Integer
End Type

Dim As Integer Teil.anzahl = 0

Constructor Teil (bezeichnung As String)
  This.anzahl += 1
  This.teileName = bezeichnung
End Constructor

Function Teil.wieviele As Integer
  Return This.anzahl
End Function

Dim As Teil teil1 = Teil("Schraube")
Print "Teileanzahl: " & teil1.wieviele

Dim As Teil teil2 = Teil("Mutter")
Print "Teileanzahl: " & teil2.wieviele

Sleep
```

Auf eine statische Variable innerhalb einer Klasse kann auch ohne eine direkte Instanz zugegriffen werden, indem man sich einfach auf die Klasse bezieht. Im oberen Beispiel kann also auch außerhalb der Klasse stets auf *'Teil.anzahl'* zugegriffen werden.

Da sich eine statische Variable wie SHARED verhält, ist dies zurzeit auch die einzige Möglichkeit, ein dynamisches Array innerhalb eines UDTs zu erstellen:

```
Type Udt
    'Dim As Integer dynamicArrayDim() <- Dynamische Arrays in UDTs sind
```

```
nicht erlaubt
    Static As Integer dynamicArrayStatic() '<- Durch Static aber möglich

    Dim As Integer dummy
End Type

Dim As Integer Udt.dynamicArrayStatic()

Print LBound(Udt.dynamicArrayStatic), UBound(Udt.dynamicArrayStatic)

ReDim Udt.dynamicArrayStatic(0 To 5)
Print LBound(Udt.dynamicArrayStatic), UBound(Udt.dynamicArrayStatic)

Sleep
```

Diese Methode eignet sich aber nur bedingt für den Produktiveinsatz, da sich alle Instanzen der Klasse die deklarierten statischen dynamischen Arrays teilen müssen.

Unterschiede zu QB:
In QB kann STATIC nur in Prozeduren und innerhalb DEF FN verwendet werden.

Unterschiede zu früheren Versionen von FreeBASIC:

- Seit FreeBASIC v0.90 kann STATIC auch bei Variablen in UDTs verwendet werden.
- Seit FreeBASIC v0.16 kann STATIC auch auf Modulebene (außerhalb von SUBs/FUNCTIONs) angewandt werden. So verwendet arbeitet es genauso wie DIM.

Siehe auch:
STATIC (Klausel), STATIC (Meta), STATIC (Schlüsselwort), STATIC (UDT), DIM, SHARED, Datentypen und Deklarationen

Letzte Bearbeitung des Eintrags am 14.12.13 um 19:50:25
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

STATIC (Klausel)

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » S » **STATIC (Klausel)**
Syntax: { SUB | FUNCTION } Bezeichner (Parameterliste) [AS Typ] STATIC
Typ: Klausel
Kategorie: Programmablauf

Als Klausel bei Prozeduren bewirkt STATIC, dass die Werte aller in der Prozedur lokal dimensionierten Variablen zwischengespeichert werden; das bedeutet, dass nach einem Aufruf die Belegung der Variablen innerhalb der Prozedur erhalten bleibt.

Siehe SUB bzw. FUNCTION zu Details zu den weiteren Syntax-Bestandteilen.

Beispiel:

```
DECLARE SUB Std_Sub
DECLARE SUB Sta_Sub

PRINT "Standard-Sub:"
Std_Sub
Std_Sub
PRINT

PRINT "STATIC-Sub:"
Sta_Sub
Sta_Sub

SLEEP

SUB Std_Sub
   DIM a AS INTEGER
   a += 1
   PRINT a
END SUB

SUB Sta_Sub STATIC
   DIM a AS INTEGER
   a += 1
   PRINT a
END SUB
```

Ausgabe:

```
Standard-Sub:
 1
 1

STATIC-Sub:
 1
 2
```

Unterschiede zu früheren Versionen von FreeBASIC:
Bis FreeBASIC v0.17 war es auch zulässig, das STATIC-Schlüsselwort links vom Prozedur-Header zu platzieren (STATIC SUB Name); seit v0.17 ist diese Form allerdings ungültig.

Siehe auch:
STATIC (Anweisung), STATIC (Schlüsselwort), STATIC (UDT), STATIC (Meta), DECLARE, SUB, FUNCTION, Prozeduren

Letzte Bearbeitung des Eintrags am 27.12.12 um 21:39:53
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

STATIC (Meta)

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » S » **STATIC (Meta)**
Syntax: '$STATIC
Typ: Metabefehl
Kategorie: Metabefehle

'$STATIC bewirkt, dass alle Arrays zum Compilierzeitpunkt eine feste Speichergröße zugewiesen bekommen. Das bedeutet, sie können nicht neu festgelegt (mit REDIM) oder gelöscht werden (mit ERASE). '$STATIC hat dieselbe Wirkung wie OPTION STATIC. Der Metabefehl DYNAMIC und OPTION DYNAMIC setzen diesen Standard außer Kraft. Wird keiner von beiden Standards explizit gesetzt, geht FreeBASIC von OPTION STATIC aus.

'$STATIC kann **nur bis FreeBASIC v0.16** eingesetzt werden, oder in entsprechend höheren Versionen, die mit der Kommandozeilenoption -lang deprecated compiliert wurden! Wird mit FreeBASIC v0.17 unter der Option -lang fb compiliert, so sind alle Arrays statisch, außer sie werden mit REDIM oder ohne Dimensionsangabe erstellt.

```
REDIM Dynamisch1 (200) AS INTEGER
DIM Dynamisch2 () AS INTEGER
DIM Statisch (200) AS INTEGER
```

Folgendes Beispiel ist also nur bis v0.16 bzw. bei genannter Kommandozeilenoption gültig:
Beispiel:

```
'$STATIC
DIM StaticArray(5) AS INTEGER

'$DYNAMIC
DIM DynamicArray(5) AS INTEGER
REDIM DynamicArray(6) AS INTEGER
```

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC:
Der Metabefehl ist nur bis FreeBASIC v0.16 erlaubt. Seit FreeBASIC v0.17 ist der Befehl nur noch zulässig, wenn mit der Kommandozeilenoption -lang deprecated compiliert wurde.

Siehe auch:
STATIC (Schlüsselwort), STATIC (Anweisung), STATIC (Klausel), STATIC (UDT), DYNAMIC (Meta), DIM, REDIM, Präprozessor-Anweisungen, Verschiedenes

Letzte Bearbeitung des Eintrags am 19.01.13 um 17:27:47
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

STATIC (Schlüsselwort)

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » S » **STATIC (Schlüsselwort)**
Syntax: OPTION STATIC
Typ: Schlüsselwort
Kategorie: Programmoptionen

OPTION STATIC legt fest, dass Arrays standardmäßig STATIC verwaltet werden sollen. Der Metabefehl STATIC hat dieselbe Wirkung wie OPTION STATIC. Der Metabefehl DYNAMIC und OPTION DYNAMIC setzen diesen Standard außer Kraft. Wird keiner von beiden Standards explizit gesetzt, geht FreeBASIC von OPTION STATIC aus.

OPTION STATIC kann **nur bis FreeBASIC v0.16** eingesetzt werden, oder in entsprechend höheren Versionen, die mit der Kommandozeilenoption -lang deprecated compiliert wurden! Wird mit FreeBASIC v0.17 unter der Option -lang fb compiliert, so sind alle Arrays statisch, außer sie werden mit REDIM oder ohne Dimensionsangabe erstellt.

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC:
Die Option ist nur bis FreeBASIC v0.16 erlaubt. Seit FreeBASIC v0.17 ist diese Option nur noch zulässig, wenn mit der Kommandozeilenoption -lang deprecated compiliert wurde.

Siehe auch:
STATIC (Meta), DYNAMIC (Schlüsselwort), __FB_OPTION_DYNAMIC__, OPTION, DIM, REDIM, Verschiedenes

Letzte Bearbeitung des Eintrags am 19.01.13 um 17:28:20
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

STATIC (UDT)

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » S » **STATIC (UDT)**
Syntax:

```
Type typename
  Declare Static Function functionname ...
  Declare Static Sub subname ...
  Declare Static Operator New&"hlzeichen">] ...
  Declare Static Operator Delete&"hlzeichen">] ...
End Type
```

Typ: Klausel
Kategorie: Klassen

STATIC wird verwendet, um Methoden zu deklarieren, denen beim Aufruf keine implizite Instanz des UDTs mitgegeben werden soll. Dadurch kann diese statische Methode auch aufgerufen werden, ohne dass eine Instanz des UDTs existieren muss.

- 'typename' ist der Name des UDTs
- 'functionname' und 'subname' sind die Namen der Prozeduren

Beim Aufruf einer Methode (also einer innerhalb des UDTs deklarierten SUBs oder FUNCTION) wird für gewöhnlich implizit eine Instanz der zugehörigen UDT-Variablen mitgeben. Diese kann dann innerhalb der Methode mit THIS angesprochen werden. Dadurch ist ein Zugriff auf die Records des UDTs möglich. Bei der Deklaration einer *'static'*, also einer statischen Prozedur wird dagegen intern keine implizite Instanz des Objekts an die Prozedur übergeben, sodass nur Zugriff auf übergebene Parameter besteht.

Statische Memberprozeduren können nur auf andere statische Member (Attribute und Methoden) des UDTs zugreifen, nicht aber auf normale Member. Um dies doch zu tun, muss ein Objekt-Pointer als zusätzlicher Parameter übergeben werden. Außerhalb des UDTs kann eine statische Memberprozedur wie eine nicht statische verwendet werden. Die Adresse einer statischen Memberfunktion kann wie gewohnt mit Hilfe von PROCPTR oder dem @-Operator einem Function-Pointer zugewiesen werden.

STATIC funktioniert nur mit SUB, FUNCTION und den Operatoren NEW&"reflinkicon" href="temp0121.html">DELETE[].

Beispiel:

```
'Ein Beispiel, wie man während der Laufzeit einen Pointer auf eine
Prozedur innerhalb eines UDTs setzt
Type myType

  Enum handlertype
    ht_default
    ht_A
    ht_B
  End Enum

  Declare Constructor (ByVal ht As handlertype = ht_default)

  Declare Sub handler

Private:
```

```
  Declare Static Sub handler_default (ByRef obj As myType)
  Declare Static Sub handler_A (ByRef obj As myType)
  Declare Static Sub handler_B (ByRef obj As myType)
  handler_func As Sub (ByRef obj As myType)

End Type

Constructor myType (ByVal ht As handlertype)
  Select Case ht
  Case ht_A
    handler_func = @myType.handler_A
  Case ht_B
    handler_func = @myType.handler_B
  Case Else
    handler_func = @myType.handler_default
  End Select
End Constructor

Sub myType.handler
  handler_func(This)
End Sub

Sub myType.handler_default (ByRef obj As myType)
  Print "Standardhandler"
End Sub

Sub myType.handler_A(ByRef obj As myType)
  Print "Handler der Methode A"
End Sub

Sub myType.handler_B (ByRef obj As myType)
  Print "Handler der Methode B"
End Sub

Dim objects(1 To 4) As myType => { myType.handlertype.ht_B,
myType.handlertype.ht_default, myType.handlertype.ht_A }
  'Das vierte Array-Element wird die Standardhandlermethode sein

For i As Integer = 1 To 4
  Print i,
  objects(i).handler()
Next

Sleep
```

Beispiel 2 - Verwendung einer statischen Memberprozedur wie eine nicht statische:

```
Type myType
  Declare Static Function add (x As Integer, y As Integer) As Integer

  dummy As Integer
End Type

Function myType.add (x As Integer, y As Integer) As Integer
```

```
  Return x + y
End Function

Print myType.add (5, 3)

Sleep
```

Unterschiede zu QB: neu in FreeBASIC

Siehe auch:
STATIC (Anweisung), STATIC (Klausel), STATIC (Meta), STATIC (Schlüsselwort), DECLARE, TYPE, CLASS, Objektorientierung

Letzte Bearbeitung des Eintrags am 27.12.12 um 22:16:51
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

STDCALL

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » S » **STDCALL**
Syntax: [DECLARE] {SUB | FUNCTION} prozedurname STDCALL [OVERLOAD] [ALIAS "externerName"] [(parameterliste) ...] {AS rueckgabetyp} {CONSTRUCTOR [priorität]} [STATIC] [EXPORT]
Typ: Schlüsselwort
Kategorie: Bibliotheken

Die STDCALL-Aufrufkonvention ist die Standard-Aufrufkonvention für FreeBASIC und die Microsoft Win32-API.
Wird in der Deklaration keine Angabe zur Aufrufkonvention gemacht, benutzt FreeBASIC 'STDCALL'.

Angegebene Parameter werden von rechts nach links (<-) übergeben, der letzte angegebene Parameter wird also als erster auf den Stack gelegt. Die aufgerufene Funktion muss den Stack vor dem Rücksprung mit RET XX (Return, Anzahl Byte) ausgleichen.

STDCALL kann nicht mit variabler Argumenten-Anzahl (siehe ...) verwendet werden.

STDCALL ist die Aufrufkonvention, die standardmäßig unter Windows verwendet wird, sofern keine andere Konvention explizit festgelegt oder durch einen EXTERN-Block impliziert wird. Es ist außerdem die gebräuchlichste Konvention in BASIC-Dialekten und in der Windows-API.

Beispiel 1:

```
"hlstring">"fbgfx.bi"

'SCREEN 18,32
Asm
  push 0  '5. Parameter optional
  push 0  '4. Parameter optional
  push 0  '3. Parameter optional
  push 32 '2. Parameter Farbbits
  push 18 '1. Parameter Screenmodus
  call _fb_GfxScreen@20
End Asm
```

Die aufgerufene Funktion (im Beispiel '_fb_GfxScreen@20') nimmt die Werte vom Stack und bereinigt den Stack.
Der x86-Befehl RET (Return) erlaubt als Parameter einen optionalen Byte-Wert, der die Größe des abzubauenden Stacks angibt (RET 20).

Beispiel 2: Vergleich der verschiedenen Aufrufmöglichkeiten

```
Sub s StdCall (s1 As Integer, s2 As Integer)
  Print "StdCall ", s1, s2
End Sub
Sub c Cdecl (c1 As Integer, c2 As Integer)
  Print "Cdecl ", c1, c2
End Sub
Sub p Pascal (p1 As Integer, p2 As Integer)
  Print "Pascal ", p1, p2
End Sub
```

```
Asm
  push 2 '2. Parameter - s2
  push 1 '1. Parameter - s1
  Call s 'rechts nach links

  push 2 '2. Parameter - c2
  push 1 '1. Parameter - c1
  Call c 'rechts nach links
  Add esp, 8 'der Stack muss durch die aufrufende Funktion aufgeräumt
werden

  push 1 '1. Parameter - p1
  push 2 '2. Parameter - p2
  Call p 'links nach rechts
End Asm

Sleep
```

Unterschiede zu QB: neu in FreeBASIC

Siehe auch:
DECLARE, CDECL, PASCAL, Module (Library / DLL)

Letzte Bearbeitung des Eintrags am 24.12.12 um 00:31:57
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

STEP

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » S » **STEP**
STEP ist ein Schlüsselwort, das in FOR-Schleifen und in Grafikbefehlen verwendet wird.

FOR-Schleife:
In einer FOR-Schleife gibt STEP die Schrittweite an, um welche die Zählvariable in jedem Durchlauf verändert werden soll.
siehe FOR ... NEXT

FOR-Schleife für UDTs:
Als OPERATOR ist STEP auch für eigene Typen überladbar.
siehe Überladen von Prozeduren und Operatoren (Overload)

Grafikbefehle:
In Grafikbefehlen wie z. B. PSET, LINE oder PUT / GET legt STEP fest, dass die darauf folgenden Koordinaten relativ zum aktuellen Grafikcursor angegeben sind.
siehe Grafik

Letzte Bearbeitung des Eintrags am 24.12.12 um 00:34:12
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

STICK

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » S » **STICK**
Syntax: STICK (n)
Typ: Funktion
Kategorie: Benutzereingabe

STICK wird in QB verwendet, um die x-y-Koordinaten zweier Joysticks zu ermitteln. Wenn ein Joystick erkannt wurde, wird als Ergebnis ein Wert von 1 bis 200 ausgegeben, sonst wird 0 zurückgegeben.

STICK funktioniert **nur in der Dialektform -lang qb** und dient der Abwärtskompatibilität zu QB. Verwenden Sie stattdessen GETJOYSTICK zur umfassenden Joystick-Abfrage.

'n' gibt an, welcher Joystick bzw. welche Koordinate abgefragt werden soll:

- 0: x-Position von Joystick A
- 1: y-Position von Joystick A (nach STICK(0)-Aufruf)
- 2: x-Position von Joystick B (nach STICK(0)-Aufruf)
- 3: y-Position von Joystick B (nach STICK(0)-Aufruf)

Jede Abfrage des Joysticks muss mit STICK(0) beginnen, da sonst für STICK(1) bis STICK(3) immer die gleichen Werte ausgegeben werden.

Beispiel:

```
"hlstring">"qb"
Screen 12
Do
    Locate 1, 1
    Print "Joystick A, X-Position : "; Stick(0); "    "
    Print "Joystick A, Y-Position : "; Stick(1); "    "
    Print "Joystick B, X-Position : "; Stick(2); "    "
    Print "Joystick B, Y-Position : "; Stick(3); "    "
    Print
    Print "Druecke ESC zum Beenden"

    If Inkey$ = Chr$(27) Then
        Exit Do
    End If
    Sleep 1
Loop
```

Unterschiede zu früheren Versionen von FreeBASIC: existiert seit FreeBASIC v0.18.3

Unterschiede unter den FB-Dialektformen: nur in der Dialektform -lang qb verfügbar

Siehe auch:
STRIG, GETJOYSTICK, Benutzereingaben

Letzte Bearbeitung des Eintrags am 24.12.12 um 02:00:45
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

STOP

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » S » **STOP**
Syntax: STOP [Errorlevel]
Typ: Anweisung
Kategorie: System

STOP beendet die Ausführung des Programms und gibt einen Errorlevel an das System zurück. Wird 'Errorlevel' ausgelassen, nimmt FreeBASIC automatisch 0 an.

STOP hat dieselbe Funktion wie END oder SYSTEM und besteht nur aus Kompatibilitätsgründen zu älteren BASIC-Dialekten. Es wird empfohlen, stattdessen END zu verwenden.

Wenn das Programm mit STOP beendet wird, dann werden Variablen und Speicher nicht automatisch zerstört. Die Destruktoren von Objekten werden nicht aufgerufen. Benötigte Objekt-Destruktoren müssen daher vor der STOP-Anweisung explizit aufgerufen werden.

Beispiel:

```
Print "Dieser Text wird angezeigt."
Sleep
Stop
Print "Dieser Text wird niemals angezeigt."
```

Unterschiede zu QB:
In QB hält STOP den Programmablauf im Interpreter an. Das Programm kann an dieser Stelle wieder fortgeführt werden.

Siehe auch:
END, SYSTEM, Betriebssystem-Anweisungen

Letzte Bearbeitung des Eintrags am 18.06.13 um 00:29:25
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

STR

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » S » **STR**
Syntax A: STR[$](numerischer Ausdruck)
Syntax B: STR[$](Unicode-String)
Typ: Funktion
Kategorie: Typumwandlung

STR verwandelt einen numerischen Ausdruck in einen STRING. Es ist die Gegenfunktion zu VAL.

STR kann auch verwendet werden, um einen Unicode-String in einen ASCII-String zu verwandeln. So verwendet ist es die Gegenfunktion von WSTR.

Das Dollarzeichen ($) als Suffix ist optional.

Beispiel:

```
DIM a AS INTEGER
DIM b AS STRING
a = 8421
b = STR(a)
PRINT a, b, STR(&HaB1)
SLEEP
```

Die Funktion von STR kann vom Operator & übernommen werden, wenn die auszugebende Zahl nicht am Stringanfang stehen soll:

```
PRINT "Test=" & (17+4)*2
```

Beginnt der nachfolgende Befehl mit einem H, B oder O, so muss er durch ein Leerzeichen vom & getrennt sein, da es sonst als Einleitung für eine Hexadezimal-, Binär- bzw. Oktalzahl interpretiert wird.

Unterschiede zu QB:

- In QB ist das Suffix $ verbindlich.
- Da QB kein Unicode unterstützt, existiert dort auch nicht die Möglichkeit, WSTRING zu verwenden.

Plattformbedingte Unterschiede:
Unicode wird in der DOS-Portierung von FreeBASIC nicht unterstützt, deshalb kann dort auch kein WSTRING verwendet werden.

Unterschiede unter den FB-Dialektformen:

- In der Dialektform -lang qb wird bei positiven Zahlen ein Leerzeichen vorangestellt.
- In der Dialektform -lang qb ist das Suffix $ verbindlich.
- In den Dialektformen -lang fblite und -lang fb ist das Suffix optional.

Siehe auch:
VAL, WSTR, ASC, CHR, STRING (Funktion), STRING (Datentyp)

Letzte Bearbeitung des Eintrags am 27.12.12 um 22:18:02
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

STRIG

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » S » **STRIG**
Syntax: STRIG (button)
Typ: Funktion
Kategorie: Benutzereingabe

STRIG wird in QB verwendet, um den Zustand der Tasten an zwei Joysticks zu ermitteln. Wenn die Taste als gedrückt erkannt wurde, wird -1 ausgegeben, sonst wird 0 zurückgegeben.

STRIG funktioniert **nur in der Dialektform -lang qb** und dient der Abwärtskompatibilität zu QB. Verwenden Sie stattdessen GETJOYSTICK zur umfassenden Joystick-Abfrage.

'button' gibt an, welcher Joystick bzw. welcher Button geprüft werden soll:

- 0: Erste Taste von Joystick A wurde seit dem letzten STICK(0)-Aufruf gedrückt.
- 1: Erste Taste von Joystick A wird gedrückt.
- 2: Erste Taste von Joystick B wurde seit dem letzten STICK(0)-Aufruf gedrückt.
- 3: Erste Taste von Joystick B wird gedrückt.
- 4: Zweite Taste von Joystick A wurde seit dem letzten STICK(0)-Aufruf gedrückt.
- 5: Zweite Taste von Joystick A wird gedrückt.
- 6: Zweite Taste von Joystick B wurde seit dem letzten STICK(0)-Aufruf gedrückt.
- 7: Zweite Taste von Joystick B wird gedrückt.

Jede Abfrage mit STICK(0) setzt die Werte von STRIG(0; 2; 4; 6) auf 0 zurück.

Beispiel:

```
"hlstring">"qb"
Screen 12
Do
    Locate 1, 1
    Stick(0)
    Print "Knopf A1 wurde gedrueckt : "; STRIG(0); "  "
    Print "Knopf A1 wird gedrueckt  : "; STRIG(1); "  "
    Print "Knopf B1 wurde gedrueckt : "; STRIG(2); "  "
    Print "Knopf B1 wird gedrueckt  : "; STRIG(3); "  "
    Print "Knopf A2 wurde gedrueckt : "; STRIG(4); "  "
    Print "Knopf A2 wird gedrueckt  : "; STRIG(5); "  "
    Print "Knopf B2 wurde gedrueckt : "; STRIG(6); "  "
    Print "Knopf B2 wird gedrueckt  : "; STRIG(7); "  "
    Print
    Print "Druecke ESC zum Beenden"

    If Inkey$ = Chr$(27) Then
        Exit Do
    End If
    Sleep 1
Loop
```

Unterschiede zu früheren Versionen von FreeBASIC: existiert seit FreeBASIC v0.18.3

Unterschiede unter den FB-Dialektformen: nur in der Dialektform -lang qb verfügbar

Siehe auch:
STICK, GETJOYSTICK, Benutzereingaben

Letzte Bearbeitung des Eintrags am 24.12.12 um 02:01:23
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

STRING (Datentyp)

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » S » **STRING (Datentyp)**
Typ: Datentyp

Ein STRING ist eine Zeichenkette, die einen zusammenhängenden Text, einzelne Wörter, Buchstaben oder sonstige Zeichen enthalten kann; auch Leerstrings (also Strings ohne jeglichen Inhalt) sind möglich.
"Hallo!" ist ein Beispiel für einen String.
In FreeBASIC werden Strings immer von "Anführungszeichen" eingerahmt. Sollen in Ihrer Zeichenkette Anführungszeichen vorkommen, so können Sie dies auf verschiedene Weise erreichen:

- Durch Stringverkettung und CHR: Der ASCII-Code eines Anführungszeichens ist 34. Sie können also einfach CHR(34) an einen String anhängen, wie in diesem Beispiel:
  "Und er sagte: " & CHR(34) & "Möge die Macht mit dir sein" & CHR(34)
- Zwei Anführungszeichen in Folge werden von FreeBASIC genauso implementiert wie ein CHR(34). Der genannte Beispielcode lässt sich also auch bequem eingeben als:
  "Und er sagte: ""Möge die Macht mit dir sein"""
  Beachten Sie dabei aber, dass das abschließende Anführungszeichen, welches das Stringende markiert, nicht fehlen darf!
- Seit v0.17 kann auch die folgende Schreibweise benutzt werden:
  !"Und er sagte: \34Möge die Macht mit dir sein\34"
  (man beachte das Ausrufezeichen vor dem ersten Anführungszeichen)

Man unterscheidet zwei Arten von Strings:

- var-length-Strings (Strings variabler Länge)
- fixed-length-Strings (Strings fester Länge)

Var-length-Strings werden dynamisch verwaltet; ihre Größe im Speicher hängt von der Länge des Strings ab. Sie werden durch diese Zeile erstellt:

```
DIM variable AS STRING
```

Strings variabler Länge werden intern über einen Descriptor (dt: Bezeichner) gehandhabt. Dieser Bezeichner enthält einen Pointer zum eigentlichen String und der Länge des Strings.
VARPTR gibt einen Pointer auf den Bezeichner zurück, während STRPTR einen Pointer auf den tatsächlichen String zurückgibt.
Am Ende eines Strings wird automatisch ein CHR(0) angehängt. Dadurch kann bei der Übergabe an externe Funktionen auf zeitaufwändiges Kopieren verzichtet werden.
Ein String variabler Länge wird also durch drei Dinge beschrieben: Die Adresse seines Bezeichners, die Adresse der Zeichenkette und ihre Länge. Aus diesem Grund gibt SIZEOF(STRING) 12 bzw. 24 bei einem 64bit Compiler aus. Dieser Wert ergibt sich durch zwei Pointer Adressen zu 4 bzw. 8 Bytes und einem INTEGER, ebenfalls 4 bzw. 8 Bytes groß.

Achtung: Bei der Nutzung von variablen Strings sollte auch der Artikel zu BYVAL (Schlüsselwort) gelesen werden, da die Nutzung von BYVAL in Parameter-Deklarationen von Funktionen oder SUB-Routinen zu Problemen führen kann.

Ein fixed-length-String verhält sich wie ein QB-String fester Länge. Er wird folgendermaßen erstellt:

```
DIM variable AS STRING * Länge
```

Länge ist dabei eine Konstante oder ein einfacher Ausdruck; Länge darf KEINE Variablen enthalten.
Strings fester Länge haben keinen descriptor und werden nicht redimensioniert, wenn sich die Länge der

enthaltenen Zeichenkette ändert. Wenn einem String fester Länge eine Zeichenkette zugewiesen werden soll, die größer ist als die zuvor festgelegte Länge des Strings, dann wird sie - wie in QB - am rechten Ende abgeschnitten, so dass sie in die String-Speicherstelle passt.
Soll also

```
1234567890
```

in einem String mit fester Länge 5 gespeichert werden, enthält der String anschließend die Zeichenkette

```
12345
```

Ein String - egal ob fixed oder var length - kann bei einem 32bit Compiler bis zu 2.147.483.648 Zeichen lang sein, bei 64bit bis zu 9.223.372.036.854.775.808; dies entspricht einer Datenmenge von 2 GB bzw. 8.388.607 TB.

Alle Strings können indiziert werden. Das bedeutet, man kann den ASCII-Code jedes beliebigen Zeichens innerhalb eines Strings über seinen Index ermitteln. Dazu verwendet man die Syntax

```
ASCII = StringVariable&"hlzeichen">]
```

wobei 'n' die Nummer des Zeichens im String minus eins ist. Das erste Zeichen eines Strings wird also über StringVariable[0] indiziert. Das letzte Zeichen wäre 'LEN(StringVariable) - 1'. Werden Bereiche außerhalb dieser Werte verwendet, kann das zu schweren Fehlern führen.

Beispiel:

```
DIM a AS STRING, b AS STRING * 11  ' Strings erstellen
a = "hello World"   ' und befüllen
b = a
PRINT a   ' auf dem Bildschirm ausgeben
PRINT b
PRINT

DIM aa AS STRING PTR, ab AS byte PTR  ' Zwei Pointer zur Analyse...
aa = @a                               ' die Adresse des Bezeichners
ab = cptr(byte ptr, @b)               ' die Adresse des ersten Zeichens
PRINT aa                              ' Adressen ausgeben
PRINT ab
PRINT
PRINT *ab                             ' 104, ASCII-Code von h, dem
                                      ' ersten Zeichen im String
PRINT CHR(*ab)                        ' h, das erste Zeichen im String
PRINT
PRINT PEEK(UINTEGER, aa)              ' beide Male die Adresse der
PRINT STRPTR(a)                       ' eigentlichen Zeichenkette
PRINT
PRINT PEEK(UBYTE, PEEK(UINTEGER, aa)) ' Beide Male das erste Zeichen des
PRINT a&"hlzahl">0]                           ' var-length-Strings als
ASCII...
PRINT
PRINT CHR(PEEK(UBYTE, PEEK(UINTEGER, aa)))  ' und als Zeichen.
PRINT CHR(a&"hlzahl">0])
SLEEP
```

Ausgabe:

```
hello World
hello World

3216641896
3216641884

 104
h

166291024
166291024

104
104

h
h
```

Unterschiede zu QB:

- Ein QB-String kann maximal 32.767 Zeichen lang sein; ein FB-String darf bis zu 2.147.483.648 bzw. 9.223.372.036.854.775.808 Zeichen lang sein.
- In FreeBASIC wird einem String intern ein CHR(0) angehängt.

Siehe auch:
SIZEOF, VARPTR, STRPTR, CAST, CHR, ASC, STR, VAL, BIN, HEX, OCT, SPACE, INPUT (Funktion), DIM, ZSTRING, WSTRING (Datentyp), Datentypen, Datentypen und Deklarationen, String-Funktionen

Letzte Bearbeitung des Eintrags am 15.01.14 um 20:07:25
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

STRING (Funktion)

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » S » **STRING (Funktion)**
Syntax A: STRING[$] (Anzahl, Code)
Syntax B: STRING[$] (Anzahl, Charakter)
Typ: Funktion
Kategorie: Stringfunktionen

STRING gibt eine Zeichenkette aus 'Anzahl' Zeichen aus; jedes Zeichen dieser Kette ist 'Charakter' bzw. CHR('Code').
Wenn 'Charakter' aus mehreren Zeichen besteht, wird eine Kette ausgegeben, die nur aus dem ersten Zeichen von 'Charakter' besteht.

Das Dollarzeichen ($) als Suffix ist optional.

Beispiel: Text unterstreichen

```
Dim msg As String
dim l as integer
msg = "FreeBASIC ist ein BASIC-Compiler."
l = LEN(msg)
PRINT msg
PRINT STRING(l, "-")
SLEEP
```

Unterschiede zu QB:
In QB ist das Suffix $ verbindlich.

Unterschiede unter den FB-Dialektformen:

- In der Dialektform -lang qb ist das Suffix $ verbindlich.
- In den Dialektformen -lang fblite und -lang fb ist das Suffix optional.

Siehe auch:
WSTRING (Funktion), SPACE, ASCII-Codes, String-Funktionen

Letzte Bearbeitung des Eintrags am 27.12.12 um 22:18:59
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

STRPTR

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » S » **STRPTR**
Syntax: STRPTR(StringVariable)
Typ: Funktion
Kategorie: Pointer

STRPTR gibt einen Pointer zu einer String-Variablen zurück. Es ist ein Alias zu SADD und verweist auf den Anfang des tatsächlichen String-Inhalts, während @ auf den Anfang der Datenstruktur zeigt. Im Artikel zum Datentyp STRING finden Sie nähere Informationen zur Handhabung des Speichers.

Beispiel:

```
DIM s AS STRING
PRINT STRPTR(s)

s = "hello"
PRINT STRPTR(s)
s = "abcdefg, 1234567, 54321"
PRINT STRPTR(s), @s

POKE UBYTE, STRPTR(s), 65
PRINT s
```

Ausgabebeispiel:

```
0
160527952
160527952     3215135288
Abcdefg, 1234567, 54321
```

Unterschiede zu QB: neu in FreeBASIC

Siehe auch:
STRING (Datentyp), @, SADD, PEEK, POKE, Pointer, thematische Übersicht zu Pointer

Letzte Bearbeitung des Eintrags am 27.12.12 um 22:19:23
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

SUB

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » S » **SUB**
Syntax:

```
[{PRIVATE | PUBLIC }] SUB Subname [Aufrufkonvention] [ALIAS "AliasName"]
_
  [OVERLOAD] ([Parameterliste]) [{CONSTRUCTOR | DESTRUCTOR}] [STATIC]
[EXPORT]
   ' Anweisungen
END SUB
```

Typ: Prozedur
Kategorie: Programmablauf

Anmerkung zur Syntax: Unterstriche (_) am Zeilenende werden von FreeBASIC so interpretiert, als wäre die Zeile nicht unterbrochen; dies dient nur der besseren Übersichtlichkeit und hat letztendlich keine Auswirkungen auf die Programmausführung.

SUB definiert ein Unterprogramm. Ein Unterprogramm besteht aus normalen Anweisungen und folgt denselben Regeln wie der Code auf Modulebene (dem Code außerhalb eines Unterprogramms).

- Die Klausel 'PRIVATE' bewirkt, dass das Unterprogramm nur aus dem Modul heraus aufgerufen werden kann, in dem es sich befindet. Siehe dazu die Referenzeinträge PRIVATE und PUBLIC.
- 'Subname' ist der Bezeichner, unter dem die SUB aufgerufen wird.
- 'Aufrufkonvention' ist ein Schlüsselwort, das angibt, in welcher Reihenfolge die Parameter übergeben werden sollen. Möglich sind: STDCALL, CDECL und PASCAL.
- 'ALIAS' gibt einer Prozedur in einer Library einen neuen Namen, mit dem man auf sie verweisen kann.
- 'OVERLOAD' erlaubt, Prozeduren mit unterschiedlicher Parameterliste, aber gleichem Subnamen zu deklarieren. Siehe dazu den Referenzeintrag OVERLOAD.
- 'Parameterliste' gibt die Parameter an, die an das Unterprogramm übergeben werden. Die Parameterübergabe wird weiter unten erläutert.
- Die Klausel 'CONSTRUCTOR und DESTRUCTOR bewirken, dass eine SUB aufgerufen wird, bevor das Programm gestartet wird bzw. nachdem es beendet wurde. Siehe dazu CONSTRUCTOR (Module) und DESTRUCTOR (Module). CONSTRUCTOR/DESTRUCTOR-SUBs dürfen keine Parameter besitzen.
- Die Klausel 'STATIC' bewirkt, dass alle Variablen innerhalb des Unterprogramms zwischengespeichert werden. Das bedeutet, dass die Werte von Variablen, die in der Sub verwendet werden, nach einem Aufruf nicht verloren gehen, sondern beim nächsten Aufruf der Sub wieder verfügbar sind. Siehe auch STATIC (Klausel).
- Die Klausel 'EXPORT' bewirkt, dass das Unterprogramm zur PUBLIC-EXPORT-Tabelle hinzugefügt wird, sodass sie von externen Programmen aus mit DYLIBSYMBOL verlinkt werden kann. Siehe dazu den Referenzeintrag EXPORT.

Um ein Unterprogramm aufzurufen, bevor es im Quellcode definiert wurde, muss es zuvor deklariert werden:

```
DECLARE SUB Subname &"hlzeichen">] [OVERLOAD] [[LIB
"DLLName"&"hlzeichen">_
      ALIAS "AliasName"&"hlzeichen">[ ( Parameterliste ) ] [{CONSTRUCTOR
| DESTRUCTOR}&"reflinkicon" href="temp0105.html">DECLARE

Der SUB-Code kann aus jeder Programmsituation heraus aufgerufen werden,
d. h. beim Aufruf der SUB wird ihr Code ausgeführt. Dadurch ist es
```

möglich, ein Programm übersichtlich zu gestalten, da immer wiederkehrende Aufgaben in eine SUB 'ausgelagert' und mittels nur einer einzigen Programmzeile ausgeführt werden können.

Parameterübergabe
Der vom Unterprogramm ausgeführte Code kann von übergebenen Parametern abhängig sein. Die Parameterliste besitzt die Form

```
[ {BYVAL | BYREF } &"hlzeichen">[Parameter1] AS &"hlzeichen">] Typ [=
Wert]   [, _
   [ {BYVAL | BYREF } &"hlzeichen">[Parameter2] AS &"hlzeichen">] Typ [=
Wert] ] [, ...]
```

- 'Parameter1', 'Parameter2', ... sind die Parameter (auch Argumente genannt). Es sind Variablen (bzw. ihre Werte), die im Unterprogramm verwendet werden können.
- Wird 'CONST' angegeben, kann der Wert innerhalb des Unterprogramms zwar gelesen, aber nicht mehr beschrieben werden.
- 'Typ' ist der Datentyp des übergebenen Parameters. Auch UDTs und Pointer können verwendet werden.
- Durch '= Wert' wird ein Parameter als optional deklariert; er kann beim Aufruf der Prozedur ausgelassen werden. In diesem Fall wird an die Funktion für diesen Parameter 'Wert' übergeben. Das VisualBASIC-Äquivalent heißt OPTIONAL.
- BYVAL und BYREF regeln die Art der Parameterübergabe an das Unterprogramm.

Der Datentyp muss bei den Parametern immer angegeben werden.
Siehe auch Parameterübergabe.

Aufgerufen wird ein Unterprogramm über seinen Bezeichner. Fügen Sie in Ihrem Code einfach eine Zeile mit dem Bezeichner der SUB ein, so als wäre dieser Bezeichner eine Anweisung. Ein solcher Aufruf darf an jeder Stelle im Programm erfolgen, auch wenn sich das Programm bereits auf Prozedurebene (also innerhalb einer SUB, FUNCTION, PROPERTY oder einem OPERATOR) befindet.

Hinweis: Früher wurde der Befehl CALL verwendet, um eine SUB aufzurufen. Dieser Befehl ist aber in der Dialektform -lang fb (Standard) nicht mehr zulässig.

Die im Hauptprogramm verwendeten Variablen sind in Prozeduren nicht verfügbar. Um auf sie dennoch zuzugreifen, müssen sie zuerst in der Parameterliste übergeben werden. Bei einer SUB (oder Prozeduren allgemein) handelt es sich um SCOPE-Blöcke. Das bedeutet, dass Variablen, die auf SUB-Ebene dimensioniert werden, auf Modulebene nicht verfügbar sind. Siehe auch Gültigkeitsbereich von Variablen.

Beispiel: farbigen Text mittels einer SUB ausgeben:

```
DECLARE SUB colortext (txt AS STRING, clr AS INTEGER)
DIM i AS INTEGER

colortext "blue" , 1
colortext "green", 2
colortext "red"  , 4
PRINT

FOR i = 1 TO 15
  colortext "COLOR " & i, i
```

```
NEXT
SLEEP

'-----------------------------------------------'

SUB colortext (txt AS STRING, clr AS INTEGER)
  COLOR clr
  PRINT txt
END SUB
```

Unterschiede zu QB:

- Parameter können in FreeBASIC optional sein.
- In FreeBASIC kann eine SUB auch mit RETURN verlassen werden.
- SUBs können in FreeBASIC überladen werden.

Unterschiede zu früheren Versionen von FreeBASIC:

- Bis FreeBASIC v0.17 war es auch zulässig, das STATIC-Schlüsselwort links vom Prozedur-Header zu platzieren; seit v0.17 allerdings ist die Form STATIC SUB Name ungültig.
- Seit FreeBASIC v0.17 ist es zulässig, auf Prozedurebene (z. B. innerhalb einer SUB) TYPEs, UNIONs und ENUMs zu erstellen.
- Seit FreeBASIC v0.16 ist es zulässig, Arrays undefinierter Größe auch innerhalb von SUBs zu definieren. (z. B. DIM AS INTEGER foo().)
- Seit FreeBASIC v0.16 wird RETURN in SUBs als Shortcut für EXIT SUB eingesetzt. Damit verbunden ist, dass GOSUB und ON ... GOSUB nicht mehr auf Prozedurebene eingesetzt werden können.
- Das Überladen einer SUB ist seit FreeBASIC v0.14 möglich; siehe OVERLOAD.
- Variable Argumente sind seit FreeBASIC v0.13 möglich; siehe VA_ARG.
- Seit FreeBASIC v0.13 können optionale Parameter auch bei Strings eingesetzt werden.
- Seit FreeBASIC v0.12 müssen in den DECLARE-Zeilen keine Parameter-Namen mehr genannt werden; es ist jedoch immer noch zulässig.

Unterschiede unter den FB-Dialektformen:

- In der Dialektform -lang fb werden Parameter standardmäßig BYVAL übergeben. In den Dialektformen -lang qb und -lang fblite ist die Standardübergabe BYREF. Siehe die zugehörigen Referenzeinträge für genauere Informationen.
- In den Dialektformen -lang qb und -lang fblite kann RETURN nur eingesetzt werden, wenn OPTION GOSUB ausgeschalten ist. In der Dialektform -lang qb muss dies explizit durch OPTION NOGOSUB festgelegt werden.

Siehe auch:
FUNCTION, PROPERTY, DECLARE, OVERLOAD, STATIC (Klausel), EXIT, END, BYVAL, BYREF, SHARED, PRIVATE (Klausel), PUBLIC (Klausel), OPTION, CONSTRUCTOR, DESTRUCTOR (Module), Prozeduren, Parameterübergabe

Letzte Bearbeitung des Eintrags am 27.12.12 um 22:19:41

FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

SWAP

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » S » **SWAP**
Syntax: SWAP Variable1, Variable2
Typ: Anweisung
Kategorie: Speicher

SWAP tauscht die Werte zweier Variablen vom selben Typ.

Beispiel:

```
' Benutzung von Swap zum Ordnen zweier Zahlen:
Dim As Integer a, b

Input "Eingabe einer Zahl: "; a
Input "Eingabe einer weiteren Zahl: "; b
If a > b Then Swap a, b
Print "Die Zahlen, in aufsteigender Reihenfolge, sind:"
Print a, b
Sleep
```

Siehe auch:
LET, Verschiedenes

Letzte Bearbeitung des Eintrags am 27.12.12 um 22:20:00
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

SYSTEM

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » S » **SYSTEM**
Syntax: SYSTEM [Errorlevel]
Typ: Anweisung
Kategorie: System

SYSTEM beendet die Ausführung des Programms und gibt einen Errorlevel an das System zurück. Wird 'Errorlevel' ausgelassen, nimmt FreeBASIC automatisch 0 an.

SYSTEM hat dieselbe Funktion wie END oder STOP und besteht nur aus Kompatibilitätsgründen zu älteren BASIC-Dialekten. Es wird empfohlen, stattdessen END zu verwenden.

Wenn das Programm mit SYSTEM beendet wird, dann werden Variablen und Speicher nicht automatisch zerstört. Die Destruktoren von Objekten werden nicht aufgerufen. Benötigte Objekt-Destruktoren müssen daher vor der SYSTEM-Anweisung explizit aufgerufen werden.

Beispiel:

```
Print "Dieser Text wird angezeigt."
Sleep
System
Print "Dieser Text wird niemals angezeigt."
```

Siehe auch:
END, STOP, Betriebssystem-Anweisungen

Letzte Bearbeitung des Eintrags am 18.06.13 um 00:28:47
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

TAB

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » T » **TAB**
Syntax: TAB (Spalte)
Typ: Anweisung
Kategorie: Konsole

TAB dient zur Einrückung eines Textes (Tabulator). Die Anweisung wird zusammen mit PRINT und PRINT # benutzt, um die Ausgabe in der angegebenen Spalte fortzusetzen. TAB funktioniert ähnlich wie LOCATE (Anweisung), wird jedoch innerhalb einer PRINT-Anweisung eingesetzt und dient nur zum Setzen der Spalte.

- Ist 'Spalte' größer als die maximale Zeichenzahl in einer Zeile (siehe WIDTH), so wird zuerst Spalte MOD Zeichenzahl gerechnet.
- Ist 'Spalte' bzw. der mit Spalte MOD Zeichenzahl errechnete Wert kleiner als die aktuelle Cursorposition, so wird der Cursor außerdem eine Zeile nach unten gesetzt. Ansonsten bleibt der Cursor in der aktuellen Zeile.

TAB wird vorwiegend dazu genutzt, Text in Tabellenform auszugeben.

Beispiel:

```
PRINT "foo"; TAB(20); "bar"
PRINT "Hallo"; TAB(20); "Welt"
PRINT
PRINT "Weiteres"; TAB(5); "Beispiel"
SLEEP
```

Ausgabe:

```
foo                 bar
Hallo               Welt

Weiteres
    Beispiel
```

Unterschiede zu QB:
In QB wird der Zwischenraum mit Leerzeichen überschrieben. In FreeBASIC kann dieses Verhalten mit SPACE erreicht werden.

Unterschiede zu früheren Versionen von FreeBASIC:
Seit FreeBASIC v0.21 wird TAB auch von PRINT USING unterstützt.

Siehe auch:
PRINT (Anweisung), SPC, LOCATE (Anweisung), SPACE, POS, Konsole

Letzte Bearbeitung des Eintrags am 25.02.12 um 01:29:40
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

TAN

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » T » **TAN**
Syntax: TAN (winkel)
Typ: Funktion
Kategorie: Mathematik

TAN gibt den Tangens des Winkels 'winkel' im Bogenmaß zurück. Der Tangens zu einem Winkel ist die Steigung einer Geraden, die mit der x-Achse den Winkel einschließt (Wikipedia-Artikel zum Thema).

- 'winkel' ist ein beliebiger numerischer Ausdruck. Variablen, Konstanten, Operatoren und Funktionen sind erlaubt. Der Ausdruck darf von jedem Datentyp außer STRING, ZSTRING oder WSTRING sein.
- Der Rückgabewert ist ein DOUBLE-Wert, der die Steigung einer Geraden darstellt.

Die Umkehrfunktionen zu TAN lauten ATN und ATAN2.

TAN kann mithilfe von OPERATOR überladen werden.

Hinweis: Die TAN-Funktion lässt sich auch durch SIN und COS ausdrücken:

```
TAN(w) = SIN(w) / COS(w)
```

Hieran erkennt man auch, dass TAN für die Werte 90°, 270°=3*90°, 450°=5*90° usw. (bzw. w=pi/2, w=pi*3/2, w=pi*5/2 usw.) nicht definiert ist.

Beispiel:

```
CONST pi AS DOUBLE  = ACOS(0)*2
DIM a AS DOUBLE
DIM r AS DOUBLE
INPUT "Bitte geben Sie einen Winkel in Grad (DEG) ein: ", a
r = a * PI / 180   ' Grad in Bogenmaß umrechnen
PRINT ""
PRINT "Der Tangens eines" ; a; "°-Winkels ist"; TAN(r)
SLEEP
```

Ausgabebeispiel:

```
Bitte geben Sie einen Winkel in Grad (DEG) ein: 75
Der Tangens eines 75°-Winkels ist 3.732050807568878
```

Siehe auch:
SIN, ASIN, COS, ACOS, ATN, ATAN2, mathematische Funktionen

Letzte Bearbeitung des Eintrags am 27.12.12 um 22:57:55
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

THEN

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » T » **THEN**
Siehe IF ... THEN

Letzte Bearbeitung des Eintrags am 28.08.11 um 18:09:05
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

THIS

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » T » **THIS**
Syntax: THIS
Typ: Schlüsselwort
Kategorie: Klassen

THIS wird in Prozeduren verwendet, die einem UDT (user defined type) fest zugeordnet sind, um auf die Records des UDTs zuzugreifen. Dazu zählen auch Properties, nicht-statische Operatoren sowie CONSTRUCTOR- und DESTRUCTOR-Prozeduren.

Auf THIS kann mit WITH zugegriffen werden.

Beispiel:

```
Type myType
  Declare Sub MyCall ()
  myVar As Integer
End Type

Dim As myType test

' Setze den Record 'myVar' auf 0
test.myVar = 0
Print "'myVar' = "; test.myVar

' Rufe die zugeordnete SUB auf
test.MyCall()

' Jetzt ist der Wert gleich 10
Print test.myVar
Sleep

Sub myType.MyCall
  this.myVar = 10
  ' Alternativ kann WITH verwendet werden:
  ' WITH THIS
  '   .myVar = 10
  ' END WITH
End Sub
```

THIS kann auch ausgelassen werden; so funktioniert das Beispiel auch, wenn die Sub folgendermaßen aussieht:

```
Sub myType.MyCall
  myVar = 10
End Sub
```

Lokalen Variablen, die denselben Namen besitzen wie ein UDT-Record, haben Vorrang. Wird eine solche lokale Variable verwendet, kann auf den Record nur mit THIS zugegriffen werden. Wenn eine globale Variable denselben Namen besitzt wie ein UDT-Record, besitzt der Record Vorrang. Man kann dennoch auf den Wert der globalen Variablen zugreifen, indem vor ihrem Namen zwei Punkte gesetzt werden.

```
Type myType
  Declare Sub changeVar (newValue As Integer)
  myVar As Integer = 5 ' Initialwert für den Record
End Type

' Damit die globale Variable innerhalb des Unterprogramms der Klasse
bekannt ist,
' muss sie vorher dimensioniert werden
Dim Shared As Integer myVar = 1

Sub myType.changeVar (newValue As Integer)
  ' verändere den Record - erkennbar an 'THIS'
  this.myVar = newValue
  Print this.myVar

  ' verändere nochmals den Record - 'THIS' kann ausgelassen werden
  myVar = newValue
  Print myVar

  ' Verwendung einer lokalen Variablen
  Dim As Integer myVar = 999
  Print "lokal:  "; myVar
  Print "Record: "; THIS.myVar  ' jetzt ist 'THIS' notwendig
  ' Zugriff auf die globale Variable
  ' die zwei Punkte sorgen dafür, dass der "Scope" der Klasse verlassen
wird
  Print "global: "; ..myVar
End Sub

Dim As myType test
test.changeVar(10)

Sleep
```

Es sei dazu aber gesagt, dass man solche Situationen vermeiden sollte, da sich sonst schnell Fehler einschleichen und Variablen nicht die erwarteten Werte enthalten, weil statt auf die globale nur auf die lokale Variable zugegriffen wurde oder umgekehrt.

Eine andere Art der Verwendung von THIS ist die mit Pointern auf Typen. In diesen Fällen darf nicht wie gewohnt mit einem Punkt auf die Variablen zugegriffen werden,

```
Print this.myVar
```

sondern mit einem Pfeil.

Beispiel:

```
Type myType
  Declare Sub changeVar (newValue As Integer)
  myVar As Integer = 5 ' Initialwert für die lokale Variable
End Type

Sub myType.changeVar (newValue As Integer)
  this.myVar = newValue
```

```
  Print this.myVar
End Sub

Dim As myType test
Dim As myType Ptr testPtr = @test

testPtr->changeVar(10)

Sleep
```

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC: existiert seit FreeBASIC v0.17

Unterschiede unter den FB-Dialektformen: nur in der Dialektform -lang fb verfügbar

Siehe auch:
TYPE (UDT), CONSTRUCTOR, DESTRUCTOR, BASE, PROPERTY, Objektorientierung

Letzte Bearbeitung des Eintrags am 27.12.12 um 22:58:13
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

THREADCALL

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » T » **THREADCALL**
Syntax: THREADCALL Prozedurname [(Parameterliste)]
Typ: Funktion
Kategorie: Multithreading

THREADCALL startet eine Prozedur Ihres Programms als eigenständigen Thread.

- 'Prozedurname' ist der Name einer SUB, die unabhängig vom Hauptprogramm arbeiten soll.
- 'Parameterliste' enthält die Parameter, die an die SUB übergeben werden sollen. Sie ist identisch mit der Parameterliste, die bei einem normalen Aufruf der SUB übergeben werden muss. Die Klammern um die Parameterliste sind erforderlich, außer die Parameterliste ist leer.
- Der Rückgabewert ist ein ANY PTR und gibt den Handle an, der den Thread identifiziert. Später kann der Thread über diesen Handle angesprochen werden. Ist der Rückgabewert gleich null, so trat ein Fehler auf; kein Thread wurde gestartet.

THREADCALL arbeitet ähnlich wie THREADCREATE. Wird eine Prozedur mit THREADCALL aufgerufen, so wird sie in einem neuen Thread gestartet und gleichzeitig mit dem Code ausgeführt, von dem sie aufgerufen wurde. Das Betriebssystem erreicht diese Gleichzeitigkeit, indem es die Prozedur einem anderen Prozessor zuweist, sofern einer vorhanden ist, oder indem es die Wartezeiten des Hauptprogramms ausnutzt.

Bevor Sie Ihr Programm beenden, müssen Sie mit THREADWAIT auf die Beendigung aller Threads warten, die von ihm gestartet wurden.

Um mehrere Threads zu synchronisieren, existieren verschiedene Möglichkeiten; sie können entweder über Mutexe aufeinander abgestimmt werden oder auf ein COND-Signal warten. Siehe dazu MUTEXCREATE sowie CONDCREATE.

THREADCALL ist eine einfachere Methode, Threads zu erstellen, und erlaubt die Übergabe von Daten, ohne dass dabei auf globale Variablen oder Pointer zurückgegriffen werden muss. Trotzdem ist die Funktion THREADCREATE effizienter und sollte in Programmen verwendet werden, die eine große Zahl an Threads erstellen.
Unter Linux wird die Bibliothek libffi (bzw. das zugehörige devel-Paket) benötigt, um die Funktion verwenden zu können, unter Windows wird die statische Version der Bibliothek mitgeliefert. Da es sich bei *libffi* um eine externe Bibliothek handelt, sollte die Lizenz beachtet werden.

Obwohl die meisten Prozeduren unterstützt werden, können folgende Prozeduren nicht mit THREADCALL gestartet werden:

- Prozeduren mit einer variablen Argumentenzahl
- Prozeduren mit UNIONs, die BYVAL übergeben werden
- Prozeduren mit UDTs, die UNIONs, Arrays, Strings oder Bitfelder enthalten und BYVAL übergeben werden

Beispiel:

```
Sub thread(id As String, tlock As Any Ptr, anzahl as Integer = 5)
  For i As Integer = 1 To anzahl
    MutexLock tlock
    Print "Schleife "; id; " befindet sich im Durchlauf Nr. "; i
    MutexUnlock tlock
    Sleep 1
```

```
  Next
End Sub

Dim tlock As Any Ptr = MutexCreate()
Dim a As Any Ptr = ThreadCall thread("A", tlock)
Dim b As Any Ptr = ThreadCall thread("B", tlock, 7)
ThreadWait a
ThreadWait b
MutexDestroy tlock
Print "Fertig (und das ohne die Verwendung von DIM SHARED!)"
Sleep
```

Mögliche Ausgabe:

```
Schleife B befindet sich im Durchlauf Nr.  1
Schleife A befindet sich im Durchlauf Nr.  1
Schleife B befindet sich im Durchlauf Nr.  2
Schleife A befindet sich im Durchlauf Nr.  2
Schleife B befindet sich im Durchlauf Nr.  3
Schleife A befindet sich im Durchlauf Nr.  3
Schleife B befindet sich im Durchlauf Nr.  4
Schleife A befindet sich im Durchlauf Nr.  4
Schleife B befindet sich im Durchlauf Nr.  5
Schleife A befindet sich im Durchlauf Nr.  5
Schleife B befindet sich im Durchlauf Nr.  6
Schleife B befindet sich im Durchlauf Nr.  7
Fertig (und das ohne die Verwendung von DIM SHARED!)
```

Unterschiede zu QB: neu in FreeBASIC

Plattformbedingte Unterschiede:

- In der DOS-Version von FreeBASIC steht THREADCREATE nicht zur Verfügung, da Threads nicht unterstützt werden.
- Unter Linux starten Threads in der Reihenfolge, in der sie erstellt wurden. Dies kann unter Windows systembedingt nicht garantiert werden.
- Unter Linux werden die Aufrufkonventionen STDCALL und PASCAL nicht unterstützt.
- Unter Windows wird die Aufrufkonvention PASCAL nicht unterstützt.

Unterschiede zu früheren Versionen von FreeBASIC: existiert seit FreeBASIC v0.24

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht THREADCALL nicht zur Verfügung.

Siehe auch:
THREADCREATE, THREADWAIT, Multithreading

Letzte Bearbeitung des Eintrags am 28.02.14 um 15:18:59
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

THREADCREATE

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » T » **THREADCREATE**
Syntax: THREADCREATE (MySub [, Argument] [, Stackgroesse])
Typ: Funktion
Kategorie: Multithreading

THREADCREATE startet eine Prozedur Ihres Programmes als eigenständigen Thread.

- 'MySub' ist ein Pointer zu der SUB, die unabhängig vom Hauptprogramm arbeiten soll. Die Parameterliste dieser SUB muss aus einem einzelnen ANY PTR bestehen, siehe Beispiel unten.
- 'Argument' ist der Integer-Parameter von MySub. Wenn mehr als ein Parameter benötigt wird, kann hier ein Pointer auf eine Array- oder UDT-Struktur übergeben werden. Da 'Argument' BYVAL übergeben wird, kann der Parameter nicht verwendet werden, um ein Ergebnis zurückzuliefern. 'Argument' ist optional; erwartet das Programm, das als Thread gestartet wird, einen Parameter, so wird null übergeben, wenn dieser Parameter ausgelassen wird.
- 'Stackgroesse' ist optional und gibt an, wie viele Byte für den Stack des Thread reserviert werden sollen.
- Der Rückgabewert ist ein ANY PTR und gibt den Handle an, der den Thread identifiziert. Später kann der Thread über diesen Handle angesprochen werden. Ist der Rückgabewert gleich null, so trat ein Fehler auf; kein Thread wurde gestartet.

Die Prozedur, die als eigener Thread gestartet wurde, läuft parallel zum Hauptprogramm. Das Betriebssystem erreicht dies, indem es die Prozedur einem anderen Prozessor zuweist, sofern einer vorhanden ist, oder indem es die Wartezeiten des Hauptprogramms ausnutzt.

Bevor Sie Ihr Programm beenden, müssen Sie mit THREADWAIT auf die Beendigung aller Threads warten, die von ihm aufgerufen wurden.

Um mehrere Threads zu synchronisieren, existieren verschiedene Möglichkeiten; sie können entweder über Mutexe aufeinander abgestimmt werden oder auf ein COND-Signal warten. Siehe dazu MUTEXCREATE sowie CONDCREATE.

Auf manchen Systemen steigt die Stackgröße automatisch über die angegebene Größe, falls nötig, auf anderen stellt der Wert die absoulte Grenze dar. Wenn auf Systemen, auf denen der Stack nicht automatisch wachsen kann, ein größerer Stack verwendet wird, als reserviert wurde, kann es das Programmverhalten negativ beeinflussen.

Beispiel:

```
Dim As Any Ptr i
Dim Shared As Any Ptr mutex

Dim As Integer DotCount
Dim Shared As Integer terminate

Sub MyThread (ByVal Parameter as Any Ptr)
  Do 'alle 100ms einen "*" ausgeben
    Print "*";
    Sleep 100, 1
    MutexLock(mutex)
    If terminate = 1 Then 'abbrechen wenn 1
        MutexUnLock(mutex)
        Exit Do
```

```
    EndIf
    MutexUnLock(mutex)
  Loop
End Sub

Print "Hauptprogramm gibt Punkte aus"
Print "Thread gibt Sterne aus"
Print
terminate = 0

mutex = MutexCreate() 'Mutex, um die terminate-Variable zu schützen

i = ThreadCreate(@MyThread)

Do  'alle 50ms einen "." ausgeben
  Print ".";
  Sleep 50, 1
  DotCount += 1
Loop Until DotCount = 100

MutexLock(mutex)
terminate = 1 'MyThread abbrechen
MutexUnLock(mutex)

Print
Print "Beendevorgang begonnen"
ThreadWait(i) 'warten bis MyThread endet
Print "Thread beendet"

MutexDestroy(mutex)

Sleep
```

Ausgabebeispiel:

```
Hauptprogramm gibt Punkte aus
Thread gibt Sterne aus

.*..*..*..*.*..*..*..*..*..*..*...*.*..*..*...*.*..*..*...*.*...*.*...*.*..*
.*...*.*..*..*..*.*..*...*.*..*..*...*.*..*..*...*.*..*..*...*.*..*
Beendevorgang begonnen
Thread beendet
```

Wie Sie sehen, werden die Punkte und Sterne nicht ganz gleichmäßig ausgegeben, obwohl das Zeitverhältnis für Hauptprogramm und Thread konstant bei 2:1 liegt (erreicht durch die beiden SLEEP-Anweisungen). Dies liegt an diversen Prozessen, die das System im Hintergrund beanspruchen und so das Programm beeinflussen. Wenn eine exakte Abstimmung zweier Threads aufeinander nötig ist, muss dies über Mutexe oder Conds erreicht werden.

Unterschiede zu QB: neu in FreeBASIC

Plattformbedingte Unterschiede:

- In der DOS-Version von FreeBASIC steht THREADCREATE nicht zur Verfügung, da Threads nicht

unterstützt werden.
- Unter Linux starten Threads in der Reihenfolge, in der sie erstellt wurden. Dies kann unter Windows systembedingt nicht garantiert werden.

Unterschiede zu früheren Versionen von FreeBASIC:

- Seit FreeBASIC v0.17 gibt THREADCREATE einen ANY PTR zurück. Davor war es ein INTEGER.
- Seit FreeBASIC v0.17 ist der Parameter 'Argument', der an die Sub übergeben wird, optional.
- THREADCREATE existiert seit FreeBASIC v0.13.

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht THREADCREATE nicht zur Verfügung.

Siehe auch:
THREADCALL, THREADWAIT, Multithreading

Letzte Bearbeitung des Eintrags am 27.12.12 um 22:58:45
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

THREADWAIT

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » T » **THREADWAIT**
Syntax: THREADWAIT handle
Typ: Anweisung
Kategorie: Multithreading

THREADWAIT wartet mit der Fortsetzung des Hauptprogramms, bis eine Prozedur, die mit THREADCREATE oder THREADCALL als eigener Thread gestartet wurde, beendet ist.

'handle' ist ein Wert vom Typ ANY PTR, der von THREADCREATE bzw. THREADCALL zurückgegeben wurde.

THREADWAIT erzwingt nicht die Beendigung des Threads. Wenn ein Thread ein Signal braucht, um sich selbst zu beenden, muss ein Mechanismus wie eine globale Variable verwendet werden.

Beispiel: siehe THREADCREATE und THREADCALL

Unterschiede zu QB: neu in FreeBASIC

Plattformbedingte Unterschiede:

- In der DOS-Version von FreeBASIC steht THREADWAIT nicht zur Verfügung, da Threads nicht unterstützt werden.
- Unter Linux starten Threads in der Reihenfolge, in der sie erstellt wurden. Dies kann unter Windows systembedingt nicht garantiert werden.

Unterschiede zu früheren Versionen von FreeBASIC:

- Seit FreeBASIC v0.17 verlangt THREADWAIT einen ANY PTR als Parameter. Davor war es ein INTEGER.
- THREADWAIT existiert seit FreeBASIC v0.13

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht THREADWAIT nicht zur Verfügung.

Siehe auch:
Multithreading

Letzte Bearbeitung des Eintrags am 27.12.12 um 22:59:00
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

TIME

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » T » **TIME**
Syntax: TIME[$]
Typ: Funktion
Kategorie: Datum und Zeit

TIME gibt die aktuelle Uhrzeit im Format hh:mm:ss aus.

Das Dollarzeichen ($) als Suffix ist optional.

Beispiel:

```
Dim t As String = TIME
PRINT "Es ist jetzt "; t; " Uhr."
SLEEP
```

Ausgabebeispiel:

```
Es ist jetzt 00:13:37 Uhr.
```

Unterschiede zu QB:
Um in FreeBASIC eine neue Uhrzeit einzustellen, muss SETTIME verwendet werden.

Unterschiede unter den FB-Dialektformen:

- In der Dialektform -lang qb ist das Suffix $ verbindlich.
- In den Dialektformen -lang fblite und -lang fb ist das Suffix optional.

Siehe auch:
DATE, SETDATE, TIMER, SETTIME, Datum und Zeit

Letzte Bearbeitung des Eintrags am 27.12.12 um 22:59:43
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

TIMER

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » T » **TIMER**
Syntax: TIMER
Typ: Funktion
Kategorie: Datum und Zeit

TIMER gibt die Zahl der Sekunden zurück, die seit dem Systemstart (unter DOS/Windows) bzw. seit der Unix-Epoche (unter Unix/Linux) vergangen sind.
Der Rückgabetyp ist ein DOUBLE.

Wenn Ihr Prozessor einen Präzisions-Timer besitzt (wie ihn beispielsweise die Performance Counter Pentium Prozessoren von Intel besitzen), wird die Zeitauflösung mit der Prozessoruhr verknüpft; eine Genauigkeit von Mikrosekunden kann angenommen werden. Ältere Prozessoren (wie 486er) haben eine Zeitauflösung von 1/18-Sekunden, was der Genauigkeit von QB entspricht.

Beachten Sie: Wenn Ihr Programm für 0.1 oder mehr Sekunden unterbrochen werden muss, sollten Sie SLEEP verwenden, da dieser Befehl anderen Programmen erlaubt, während der Wartezeit weiterzuarbeiten.

Beispiel:

```
Dim zeit As Double
PRINT "Sekunden, die seit Systemstart vergangen sind: "; TIMER

PRINT "Warte 3 Sekunden"

zeit = TIMER
DO
  SLEEP 1  ' Prozessorlast senken
LOOP UNTIL TIMER > zeit + 3
' Alternativ hätte statt der Schleife auch  SLEEP 3000,1  verwendet
werden können.
PRINT "Abgeschlossen."
```

Unterschiede zu QB:

- In QB wird die Zeit seit Mitternacht zurückgegeben.
- Die Zeitauflösung ist in FreeBASIC vom Prozessor abhängig; höhere Auflösungen als 1/18 sek sind möglich.

Plattformbedingte Unterschiede:

- Unter DOS und Windows gibt TIMER die Zeit seit Systemstart zurück.
- Unter unixartigen Betriebssystemen wie Linux wird die Zeit seit der Unix-Epoche (01.01.1970) zurückgegeben.

Siehe auch:
TIME, SETTIME, DATE, SETDATE, SLEEP, Datum und Zeit

Letzte Bearbeitung des Eintrags am 27.12.12 um 23:00:09
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

TIMESERIAL

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » T » **TIMESERIAL**
Syntax: TIMESERIAL (Stunde, Minute, Sekunde)
Typ: Funktion
Kategorie: Datum und Zeit

Anmerkung:
Um diese Funktion nutzen zu können, muss die Datei datetime.bi in Ihren Quellcode eingebunden werden, z. B. mit INCLUDE. Alternativ können Sie auch die Datei vbcompat.bi einbinden, da dadurch auch automatisch datetime.bi eingebunden wird.

Diese Funktion wandelt eine angegebene Uhrzeit in eine Serial Number um. Der Rückgabetyp ist DOUBLE. Durch die interne Darstellung von DOUBLE-Werten kann es zu Ungenauigkeiten bei großen Werten kommen.
Diese Funktion wird vom Compiler solange ignoriert, bis die Datei datetime.bi (oder vbcompat.bi) in den Quellcode eingebunden wurde.

- 'Stunde', 'Minute' und 'Sekunde' geben die Uhrzeit an.
- Wenn Sie "unsinnige" Zeiten angeben, wie z.B. 25:60:60, werden diese Fehler automatisch umgerechnet; das Ergebnis wäre hier 2:01:00.

Beispiel:

```
"hlstring">"vbcompat.bi"
Dim a As Double
a = DATESERIAL(2005, 11, 28) + TIMESERIAL(19, 3, 26)
PRINT FORMAT(a, "dd.mm.yyyy hh:mm:ss")
SLEEP
```

Ausgabe:

```
28.11.2005 19:03:26
```

Unterschiede zu QB: existiert nur in QBX PDS und in VBDOS

Unterschiede zu früheren Versionen von FreeBASIC: existiert seit FreeBASIC v0.15

Siehe auch:
NOW, DATESERIAL, TIMEVALUE, HOUR, MINUTE, SECOND, FORMAT, Serial Numbers, Datum und Zeit

Letzte Bearbeitung des Eintrags am 27.12.12 um 23:01:05
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

TIMEVALUE

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » T » **TIMEVALUE**
Syntax: TIMEVALUE (Zeit)
Typ: Funktion
Kategorie: Datum und Zeit

Anmerkung:
Um diese Funktion nutzen zu können, muss die Datei datetime.bi in Ihren Quellcode eingebunden werden, z. B. mit INCLUDE. Alternativ können Sie auch die Datei vbcompat.bi einbinden, da dadurch auch automatisch datetime.bi eingebunden wird.

TIMEVALUE verwandelt einen STRING mit einer Zeitangabe in eine Serial Number.

- 'Zeit' ist der STRING, der in eine Serial Number verwandelt werden soll. Er muss in einem dieser Formate übergeben werden:
  hh:mm:ss oder h:m:s
- Der Rückgabewert ist ein DOUBLE, der die Serial Number enthält

Beispiel: Die aktuelle Zeit mit TIME ermitteln und in eine Serial Number verwandeln:

```
"hlstring">"vbcompat.bi"
DIM ts AS DOUBLE
ts = TIMEVALUE(TIME)
```

Unterschiede zu QB: existiert nur in QBX PDS und in VBDOS

Unterschiede zu früheren Versionen von FreeBASIC: existiert seit FreeBASIC v0.15

Siehe auch:
NOW, TIMESERIAL, DATEVALUE, HOUR, MINUTE, SECOND, FORMAT, Serial Numbers, Datum und Zeit

Letzte Bearbeitung des Eintrags am 26.03.13 um 17:32:55
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

TO

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » T » **TO**
Typ: Schlüsselwort
Kategorie: Programmablauf

Das Schlüsselwort TO gibt Bereiche für andere Befehle an.

TO wird verwendet um...

- den Zählbereich einer FOR...NEXT-Schleife festzulegen.
- die Größe eines Arrays mit DIM festzulegen.
- einen Bereich zu akzeptierender Werte für SELECT CASE anzugeben.
- einen Bereich einer Datei zu sperren (siehe dazu LOCK).

Beispiel:

```
Dim As Integer array(1 To 5)

For i As Integer = 1 To 5
  array(i) = i

  Select Case array(i)
    Case 1 To 3
      Print "1 bis einschliesslich 3"
    Case 4 To 5
      Print "4 bis einschliesslich 5"
  End Select
Next

Sleep
```

Siehe auch:
DIM, FOR, SELECT CASE, LOCK, Verschiedenes

Letzte Bearbeitung des Eintrags am 28.12.12 um 00:37:18
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

TRANS

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » T » **TRANS**
Syntax: { PUT | DRAW STRING } [Puffer,] [STEP] (x, y), [weitere Angaben ...], TRANS
Typ: Schlüsselwort
Kategorie: Grafik

TRANS ist ein Schlüsselwort, das im Zusammenhang mit PUT (Grafik) und DRAW STRING eingesetzt wird. Ähnlich wie bei der Methode PSET werden die Pixeldaten wiedergegeben, ohne dabei die überzeichneten Pixel zu beachten. Allerdings werden bei der Grafikausgabe die Teile in der Maskenfarbe ignoriert, d. h. nicht gezeichnet.
Siehe dazu auch Interne Pixelformate.

Beispiel:

```
ScreenRes 320, 200, 32
Line (0, 0)-(319, 199), RGB(0, 128, 255), bf

' Bild mit transparenter Hintergrundfarbe erstellen
Dim img As Any Ptr = ImageCreate( 33, 33, RGB(255, 0, 255) )
Circle img, (16, 16), 15, RGB(255, 255, 0),     ,     , 1, f
Circle img, (10, 10), 3,  RGB(  0,   0, 0),     ,     , 2, f
Circle img, (23, 10), 3,  RGB(  0,   0, 0),     ,     , 2, f
Circle img, (16, 18), 10, RGB(  0,   0, 0), 3.14, 6.28

Dim As Integer x = 160 - 16, y = 100 - 16

' Grafik ausgeben
Put (x, y), img, Trans

' Bildspeicher freigeben und auf Tastendruck warten
ImageDestroy img
Sleep
```

Unterschiede zu QB: neu in FreeBASIC

Siehe auch:
PUT (Grafik), DRAW STRING, SCREENRES, AND (Methode), OR (Methode), XOR (Methode), PSET (Methode), PRESET (Methode), ALPHA, ADD, CUSTOM, Grafik

Letzte Bearbeitung des Eintrags am 24.12.12 um 21:07:10
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

TRIM

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » T » **TRIM**
Syntax: TRIM[$] (Stringausdruck [, [ANY] zuEntfernen])
Typ: Funktion
Kategorie: Stringfunktionen

TRIM gibt einen String aus, aus dem bestimmte führende oder angehängte Zeichen entfernt werden sollen.

- 'Stringausdruck' ist der STRING, ZSTRING oder WSTRING, der gekürzt werden soll.
- 'zuEntfernen' ist ein Ausdruck, der angibt, welche Zeichen entfernt werden sollen. Wird dieser Parameter ausgelassen, entfernt FreeBASIC automatisch alle Leerzeichen am Anfang und Ende des Strings. 'zuEntfernen' darf aus mehreren Zeichen bestehen.
- Wird die Klausel ANY verwendet, entfernt FreeBASIC jedes Zeichen aus 'zuEntfernen' am Rand von 'Stringausdruck'.
- Der Rückgabewert ist der um die angegebenen Zeichen gekürzte String.

TRIM arbeitet *case-sensitive*, d. h. die Groß-/Kleinschreibung ist ausschlaggebend.

Das Dollarzeichen ($) als Suffix ist optional.

Beispiel:

```
DIM AS STRING foobar, dadada
foobar = "   foo bar   "
dadada = "da da da"

PRINT TRIM(foobar)
PRINT TRIM(dadada, "da")
PRINT TRIM(dadada, ANY "bd")
SLEEP
```

Ausgabe:

```
foo bar
 da
a da da
```

Unterschiede zu QB: neu in FreeBASIC

Plattformbedingte Unterschiede:
Unicode wird in der DOS-Portierung von FreeBASIC nicht unterstützt, deshalb kann dort auch kein WSTRING verwendet werden.

Unterschiede zu früheren Versionen von FreeBASIC:
Der Parameter 'zuEntfernen' und die Klausel ANY existieren seit FreeBASIC v0.15.

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht TRIM nicht zur Verfügung und kann nur über **__TRIM** aufgerufen werden.

Siehe auch:
LTRIM, RTRIM, INSTR, String-Funktionen

Letzte Bearbeitung des Eintrags am 28.12.12 um 00:37:48
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

TYPE (Forward Referencing)

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » T » **TYPE (Forward Referencing)**
Syntax A: TYPE ForwardName AS TypeName
Syntax B: TYPE AS TypeName ForwardName
Typ: Anweisung
Kategorie: Klassen

In manchen Situationen ist es notwendig, einen Pointer auf ein UDT zu legen, der erst später im Programm definiert wird. Um dies zu ermöglichen, kann ein *Forward-Type* erstellt werden.

Beispiel:

```
TYPE FwdUDT2 AS UDT2

TYPE UDT1
  a AS INTEGER
  b AS FwdUDT2 PTR
END TYPE

TYPE UDT2
  c AS INTEGER
  d AS INTEGER
END TYPE
```

Wer stattdessen den folgenden Code versucht, erhält die Fehlermeldung "Typ nicht definiert":

```
' Fehlerhafte Einbindung eines noch nicht existenten UDTs
TYPE UDT1
  a AS INTEGER
  b AS UDT2 PTR
END TYPE

TYPE UDT2
  c AS INTEGER
  d AS INTEGER
END TYPE
```

Beim *Forward Referencing* ist darauf zu achten, dass nur UDTs zur Deklaration verwendet werden können, die *vollständig* sind. Ein UDT ist dann vollständig, wenn die Größe aller Elemente im UDT bekannt sind. Anders gesagt muss vom gesamten UDT die Größe in Byte bekannt sein. Folgender Code wird den Compilerfehler *'Incomplete type'* ergeben:

```
' Fehlerhafte Einbindung eines unvollständigen UDTs
Type einTyp As forwardTyp

Dim As einTyp beispiel

Type forwardTyp
  dummy As Integer
End Type
```

Um das Problem zu lösen, muss, bevor eine Instanz des UDTs erstellt wird, die Größe bekannt sein. Für das kleine Beispiel würde es folgendermaßen aussehen:

```
Type einTyp As forwardTyp

Type forwardTyp
  dummy As Integer
End Type

Dim As einTyp beispiel
```

Benötigt man die Instanz aber aus irgendwelchen Gründen vor dem Forward-Type, kann das durch einen Pointer gelöst werden. Die Größe eines Pointers ist immer bekannt, folglich auch kein Problem:

```
Type einTyp As forwardTyp

Dim As einTyp Ptr beispiel

Type forwardTyp
  dummy As Integer
End Type
```

Beispiel einer Fenster-Klasse mit Button:

```
Type tempButton As button            ' Forward-Type für 'button'

Type fenster                         ' Fenster-Klasse
  beendenknopf As tempButton Ptr     ' Beenden-Button der noch unbekannten
Klasse
  x As Integer
  y As Integer
End Type

Type button                          ' Button-Klasse
  parent As fenster Ptr              ' Eltern-Type der Button-Klasse
  x As Integer
  y As Integer
End Type

' Objekte erstellen
Dim As fenster Ptr meinProgramm

meinProgramm = New fenster
meinProgramm->beendenknopf = New button

' Daten zuweisen
meinProgramm->x = 500
meinProgramm->y = 200

meinProgramm->beendenknopf->parent = meinProgramm
meinProgramm->beendenknopf->x = 150
meinProgramm->beendenknopf->y = 25

' Fenster erstellen
ScreenRes meinProgramm->x, meinProgramm->y
Line (0, 0)-(meinProgramm->beendenknopf->x,
meinProgramm->beendenknopf->y), 15, BF
```

```
Draw String (5, 5), meinProgramm & " = " &
meinProgramm->beendenknopf->parent, 0

' Objekte zerstören
Delete meinProgramm->beendenknopf
Delete meinProgramm

Sleep
```

Unterschiede zu QB: neu in FreeBASIC

Siehe auch:
TYPE (UDT), TYPE (Funktion), Datentypen und Deklarationen, Objektorientierung

Letzte Bearbeitung des Eintrags am 23.11.13 um 00:37:04
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

TYPE (Funktion)

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » T » **TYPE (Funktion)**
Syntax: TYPE [<typename>](a [, b [, ...]])
Typ: Funktion
Kategorie: Klassen

TYPE erzeugt einen temporären Typen und stellt damit eine noch kürzere Alternative als WITH dar, wenn alle Records geändert werden.

Beispiel 1:

```
TYPE Example
  AS INTEGER field1
  AS INTEGER field2
END TYPE

DIM ex AS Example

' Standard-Methode zur Befüllung der Records
ex.field1 = 1
ex.field2 = 2

' WITH-Methode
WITH ex
  .field1 = 1
  .field2 = 2
END WITH

' TYPE-Methode
ex = TYPE(1, 2)
```

Um den Code besser lesbar zu gestalten, kann auch diese Syntax verwendet werden:

```
ex = TYPE<Example>(1, 2)
```

Wie bereits angesprochen, kann TYPE() nur verwendet werden, wenn alle Records eines UDTs neu festgelegt werden. Es gibt aber auch noch andere Verwendungsmöglichkeiten.

Beispiel 2: Funktionsaufruf mit Übergabe eines temporären UDTs

```
Type S
  As Single x, y
End Type

Sub test (v As S)
  Print "S", v.x, v.y
End Sub

test Type(1, 2)
test TYPE<S>(1, 2)
Sleep
```

Eine ähnliche Funktionalität bietet die Nutzung eines Constructors:

```
Type S
  Declare Constructor (a As Single, b As Single)
  As Single x, y
End Type

Constructor S (a As Single, b As Single)
  x = a
  y = b
End Constructor

Sub test (v As S)
  Print "S", v.x, v.y
End Sub

test S(1, 2)
Sleep
```

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC:
TYPE in der Verwendung als Funktion existiert seit FreeBASIC v0.16.

Siehe auch:
TYPE (UDT), TYPE (Forward Referencing), WITH, Datentypen und Deklarationen, Objektorientierung

Letzte Bearbeitung des Eintrags am 28.12.12 um 00:38:36
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

TYPE (UDT)

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » T » **TYPE (UDT)**
Syntax A:

```
TYPE TypenName [EXTENDS Mutterklasse] [FIELD = {1|2|4}]
  [PRIVATE:|PUBLIC:|PROTECTED:]
  variable[(Arraygrenzen)] AS Datentyp [= Wert]
  variable : bits AS Datentyp [= Wert]
  AS Datentyp variable [(Arraygrenzen)] [= Wert], ...
  AS Datentyp variable : bits [= Wert]

  DECLARE {SUB|FUNCTION|CONSTRUCTOR|DESTRUCTOR|PROPERTY|OPERATOR} ...
  ...
END TYPE
```

Syntax B:

```
TYPE TypenName AS Datentyp
```

Typ: Anweisung
Kategorie: Klassen

TYPE wird verwendet, um UDTs (*user defined types*, also benutzerdefinierte Datentypen) zu erstellen, die mehrere Variablen/Arrays enthalten können.

- 'variable' ist der Bezeichner einer Variablen, die im UDT verwendet wird. Sie wird als Record bezeichnet. Die Syntax zur Definition von Records ist ähnlich wie bei DIM.
- 'Datentyp' ist der Typ, den der Record besitzen soll. Jeder Record kann seinen eigenen Datentyp besitzen. Auch Arrays, Strings fester und variabler Länge, Pointer (Zeiger) und andere UDTs sind erlaubt.
- 'bits' wird zusammen mit Bitfeldern eingesetzt. Die Definition von Bitfeldern wird weiter unten erläutert.
- UDTs können Methoden enthalten. Diese werden innerhalb der Typendefinition deklariert. Es handelt sich hierbei um Prozeduren, die an den UDT gebunden sind. Siehe auch SUB und FUNCTION.
- Für die weiteren Deklarationsmöglichkeiten siehe CONSTRUCTOR / DESTRUCTOR, PROPERTY und OPERATOR.
- EXTENDS kann verwendet werden, um den eigenen Datentypen die Records und Methoden eines anderen UDTs erben zu lassen.
- FIELD bestimmt, auf welche Größe das Feld ausgedehnt werden soll ("Padding"). Siehe FIELD für weitere Informationen.

Ein mit TYPE erstellter Datentyp kann einer Variable mit DIM, REDIM, COMMON oder STATIC (Anweisung) zugewiesen werden. Auf die einzelnen Records wird dann über die Syntax Variable.Record zugegriffen (siehe auch WITH, TYPE (Funktion)).

Beispiel 1:

```
TYPE clr
  AS UBYTE red, green, blue
END TYPE

TYPE AnotherUDT
  IntVal   AS INTEGER
```

```
  fStrVal  AS STRING * 7
  vStrVal  AS STRING
  Array(5) AS clr
  UdtPtr   AS clr PTR
END TYPE

DIM a AS clr
DIM b AS AnotherUDT

a.red = 255
a.green = 128
a.blue = 64
PRINT a.red, a.green, a.blue
PRINT

b.IntVal  = 10023
b.fStrVal = "abcdefg"
b.vStrVal = "abcdefghijklmnopqrstuvwxyz"
b.Array(0).green = 255
b.UdtPtr  = @a

PRINT b.IntVal
PRINT b.fStrVal
PRINT b.vStrVal
PRINT b.Array(0).green
PRINT b.UdtPtr->blue
SLEEP
```

Bitfelder

Wollen Sie eine Reihe von Ganzzahlen in einem UDT speichern, die in einem relativ kleinen Wertebereich liegen, können Sie das über ein Bitfeld erreichen. Siehe dazu Bitfelder.

```
TYPE BitFeldName &"hlkw0">TYPE CheckBoxenType
  CB1 : 1 AS INTEGER
  CB2 : 1 AS INTEGER
  CB3 : 1 AS INTEGER
END TYPE
DIM CheckBoxen AS CheckBoxenType

' Status setzen:
CheckBoxen.CB1 = 1 ' aktiv
CheckBoxen.CB2 = 0 ' nicht aktiv
CheckBoxen.CB3 = 1 ' aktiv

' ... Programmcode ...

' Status abfragen:
IF CheckBoxen.CB1 THEN '...
IF CheckBoxen.CB2 THEN '...
IF CheckBoxen.CB3 THEN '...
```

TYPE zur Festlegung eines Alias
Syntax B erlaubt es, eine Variable direkt wie einen eingebauten Datentypen zu verwenden:

```
TYPE meinDatentyp AS andererDatentyp
```

Der Vorteil dieser Methode ist, dass man bei einer späteren Änderung den Datentypen nur an einer Stelle ändern muss, statt an jeder Stelle, an welcher der Datentyp verwendet wird.

FreeBASIC erlaubt für diese Syntaxform neben den allgemeinen Datentypen auch Pointer auf Prozeduren (Callbacks). Dies hat die Vorteile, dass einerseits die Deklaration verkürzt wird, was Schreibaufwand ersparen kann und andererseits, im Gegensatz zur normalen Deklaration, auch Pointer auf Callbackfunktionen ermöglicht werden:

```
' Beispielfunktionen
Function testAufruf() As Integer
  Return 1
End Function
Function testAufrufPointer() As Integer Ptr
  Return Cast(Integer Ptr, 2)
End Function

' Callbackfunktionen
Dim func As Function() As Integer
Dim pointerFunc As Function() As Integer Ptr

' Pointer für Callbackfunktionen
Type pointerFunc_Type As Function() As Integer
Dim pointerPointerFunc As pointerFunc_Type Ptr

' Funktionszuweisungen
func = @testAufruf 'Callback auf die erste Funktion
pointerFunc = @testAufrufPointer 'Callback auf die zweite Funktion
pointerPointerFunc = @func 'Pointer auf das Callback der ersten Funktion

Print "Funktionsadressen:", func, pointerFunc, pointerPointerFunc
Print "Funktionsaufrufe:", func(), pointerFunc(), *pointerPointerFunc()
Sleep
```

Objektorientierte Programmierung
FreeBASIC bietet objektorientierte Ansätze, die beispielsweise Methoden, Konstruktoren und Destruktoren in UDTs erlauben. Eine ausführliche Anleitung dazu findet sich im UDT-Tutorial.

Beispiel 3 für objektorientierte Programmierung:

```
' Vektorklasse
Type Vector
  W as Integer
  H as Integer
  Declare Constructor (nW as Integer, nH as Integer)
End Type
```

```
Constructor Vector (nW as Integer, nH as Integer)
  W = nW
  H = nH
End Constructor

' Klasse zur Erstellung eines Objekts
Type AObject
  Private:
    X as Integer
    Y as Integer
    Movement as Vector Pointer
  Public:
    ' öffentliche Methoden inklusive eines Konstruktors und eines
Destruktors
    Declare Constructor (nX as Integer, nY as Integer)
    Declare Destructor ()
    Declare Sub SetMotion (Motion as Vector Pointer)
    Declare Sub Move ()
    Declare Property GetX as Integer
End Type

' Initialwerte setzen
Constructor AObject (nX as Integer, nY as Integer)
  X = nX
  Y = nY
End Constructor

' allozierten Speicher freigeben
Destructor AObject ()
  Delete Movement
End Destructor

' Bewegungsvektor setzen
Sub AObject.SetMotion (Motion as Vector Pointer)
  Movement = Motion
End Sub

' das Objekt anhand seines Bewegungsvektors bewegen
Sub AObject.Move ()
  X += Movement->W
  Y += Movement->H
End Sub

' Rückgabe von X, welches sonst nicht zugänglich wäre
Property AObject.GetX as Integer
  Return X
End Property

' Hauptprogramm

' eine neue Instanz von 'AObject' an den Koordinaten 100, 100 erstellen
Dim Player as AObject = Type<AObject>(100, 100)

' ein neues Vektorobjekt dynamisch allozieren
```

```
' und dessen Position um 10 nach links und 5 nach unten verschieben
Player.SetMotion(new Vector (-10, 5))

' die Position von 'Player' aktualisieren
Player.Move()

' den neuen Wert von X (90) anzeigen
Print Player.GetX

' Weil 'Player' eine lokale Variable ist, wird sein Destruktor
' am Ende des Scopes automatisch aufgerufen

' vor Programmende auf Tastendruck warten
Sleep
```

Alle UDTs besitzen Standard-Konstruktoren und -Destruktoren, falls keine anderen definiert wurden, die diese überschreiben. Sie können jederzeit explizit aufgerufen werden. Dies gilt allerdings auch für selbsterstellte Konstruktoren und Destruktoren.

```
Type myType
  x As Integer
End Type

Dim As myType Ptr myVar = New myType

myVar->Constructor
myVar->Destructor

Delete myVar
Sleep
```

Unterschiede zu QB:

- In FreeBASIC besitzen Strings fester Länge im Gegensatz zu QB ein zusätzliches Zeichen am Ende, wodurch UDTs mit diesen Strings nicht kompatibel zu QB sind, wenn sie bei Dateioperationen verwendet werden.
- FreeBASIC unterstützt auch Bitfelder.

Plattformbedingte Unterschiede:

- Das Standardpadding unter Linux und DOS ist 4 Byte.
- Das Standardpadding unter Windows ist 8 Byte.

Unterschiede zu früheren Versionen von FreeBASIC:

- Bitfelder existieren seit FreeBASIC v0.13.
- Methoden in UDTs existieren seit v0.17.

Unterschiede unter den FB-Dialektformen:

- Methoden in UDTs sind nur in der Dialektform -lang fb verfügbar.
- Das Standardpadding in -lang fb und -lang fblite hängt von der Plattform ab, während in -lang qb kein Padding durchgeführt wird.

Siehe auch:
TYPE (Funktion), TYPE (Forward Referencing), DIM, OFFSETOF, FIELD, WITH, OPERATOR, CONSTRUCTOR, DESTRUCTOR, PROPERTY, OBJECT, EXTENDS, Bitfelder, Datentypen und Deklarationen, Objektorientierung

Letzte Bearbeitung des Eintrags am 30.12.12 um 01:11:07
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

TYPEOF

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » T » **TYPEOF**
Syntax: TYPEOF (Variable | Datentyp | Funktionspointertyp)
Typ: Anweisung
Kategorie: Metabefehle

TYPEOF ist eine compiler-interne Variable, die **nur während des Compilier-Vorgangs** zur Verfügung steht. Der Parameter kann sein:

- 'Variable': der Name einer Variablen oder Funktion, oder
- 'Datentyp', z. B. UBYTE, INTEGER, STRING ..., oder
- 'Funktionspointertyp': die 'Deklaration' einer FUNCTION oder SUB, oder
- ein Literal, z. B. 123 für INTEGER, 55.5 für DOUBLE oder "" für ZSTRING

In der Form DIM AS TYPEOF(INTEGER) foo oder DIM AS TYPEOF(12345) foo wird die Variable foo als INTEGER definiert.
Wenn der Name einer Funktion übergeben wird, liefert TYPEOF den Datentyp des Rückgabewerts der Funktion zurück.

Beispiel (bis 0.24 und ab 0.91):

```
Dim As TypeOf("Text") foo

"hlkw0">TypeOf(foo)

"hlkw0">TypeOf(foo) = String
  Print "String"
"hlkw0">TypeOf(foo) = ZString
  Print "ZString"
"hlkw0">Print foo

Sleep
```

Beispiel (0.90.0 und 0.90.1):

```
Function testfunktion As Single
  Return 0
End Function

Dim As TypeOf("Text") foo

' mit Variablennamen
"hlkw0">TypeOf(testfunktion)
"hlkw0">TypeOf(foo)

"hlkw0">TypeOf(foo) = "STRING"
  Print "String"
"hlkw0">TypeOf(foo) = "ZSTRING * 5"
  Print "ZString"
"hlkw0">Print foo

' mit Funktionspointertyp
"hlkw0">TypeOf(Function () As Integer)
```

```
"hlkw0">TypeOf(Sub ())

Sleep
```

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht TYPEOF nicht zur Verfügung und kann nur über **__TYPEOF** aufgerufen werden.

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC:

- In FreeBASIC v0.90.0 und v0.90.1 muss "STRING" bei Vergleichen in Anführungszeichen gesetzt werden. Die in Anführungszeichen gesetzten Datentypen sollten großgeschrieben werden, da TYPEOF diese auch so zurückliefert. Dies betrifft nur diese beiden Versionen
- Seit FreeBASIC v0.90 wird bei ZSTRING die exakte Größe mit angegeben (siehe neues Beispiel).
- Seit FreeBASIC v.90 können auch Funktion-Pointer-Typen mit TYPEOF überprüft werden (siehe neues Beispiel).

Siehe auch:
Datentypen, TYPE (UDT), Verschiedenes

Letzte Bearbeitung des Eintrags am 17.01.14 um 22:44:03
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

UBOUND

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » U » **UBOUND**
Syntax: UBOUND (Array[, Dimension])
Typ: Funktion
Kategorie: Speicher

UBOUND gibt den größten Index des angegebenen Arrays zurück.

- 'Array' ist der Name des Arrays, dessen oberster Index zurückgegeben werden soll.
- 'Dimension' ist die Nummer der Dimension, deren oberster Index zurückgegeben werden soll. Wird dieser Wert ausgelassen, so nimmt FreeBASIC automatisch 1 an. Ist 'Dimension' gleich 0, so wird die Anzahl an Dimensionen im Array zurückgegeben. Ist 'Dimension' kleiner als 0, so werden ungültige Werte erzeugt. Ist 'Dimension' größer als die Anzahl der Dimensionen, so wird 0 ausgegeben.

Ist das Array bisher nur deklariert, hat aber noch keine Dimensionen, so gibt UBOUND den Wert -1 aus. Ob das Array nun den höchsten Index -1 besitzt oder einfach keine Dimensionen, kann man über die Adresse des ersten Elements erfahren, welche 0 ist, sofern es keine Dimensionen gibt. Alternativ kann zusätzlich LBOUND abgefragt werden, was in einem solchen Fall 0 zurück gibt. Eine weitere Möglichkeit besteht darin, LBOUND und UBOUND mit Dimension 0 abzufragen, wobei bei ersterem der Wert 1 und bei letzterem der Wert 0 (keine Dimensionen) zurückgegeben wird.

Beispiel:

```
DIM Array(-10 TO 10, 5 TO 15, 1 TO 2) AS INTEGER
DIM unArray() AS INTEGER  ' Array ohne Dimensionen

PRINT UBOUND(Array, 1)
PRINT UBOUND(Array, 2)
PRINT UBOUND(Array, 3)
PRINT UBOUND(Array, 4)
PRINT

' Überprüfung, ob das Array dimensioniert wurde
PRINT UBOUND(unArray)
PRINT @unArray(0)
IF UBOUND(unArray) < LBOUND(unArray) THEN
  PRINT "Das Array wurde noch nicht dimensioniert."
END IF
SLEEP
```

Ausgabe:

```
 10
 15
 2
-1

-1
0
Das Array wurde noch nicht dimensioniert.
```

Unterschiede zu früheren Versionen von FreeBASIC:

- Bis einschließlich FreeBASIC v0.24 führten sowohl der Wert 0 als auch 1 bei 'Dimension' dazu, dass die Grenzen der ersten Dimension abgefragt werden.
- In den FreeBASIC-Versionen von v0.16 bis v0.24 ergaben leere Dimensionen statt -1 den Wert 0.

Siehe auch:
LBOUND, DIM, REDIM, Arrays

Letzte Bearbeitung des Eintrags am 04.07.13 um 19:06:45
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

UBYTE

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » U » **UBYTE**
Typ: Datentyp

Daten vom Typ UBYTE sind vorzeichenlose 8-bit-Ganzzahlen.
Der Wertebereich für Variablen vom Typ UBYTE erstreckt sich von 0 bis 255.

Unterschiede zu QB: neu in FreeBASIC

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht UBYTE nicht zur Verfügung und kann nur über **__UBYTE** aufgerufen werden.

Siehe auch:
DIM, CAST, CUBYTE, Datentypen, Datentypen und Deklarationen

Letzte Bearbeitung des Eintrags am 28.12.12 um 00:42:47
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

UCASE

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » U » **UCASE**
Syntax: UCASE[$] (Stringausdruck [, Modus])
Typ: Funktion
Kategorie: Stringfunktionen

UCASE wandelt einen Stringausdruck in Großbuchstaben.

- 'Stringausdruck' ist ein STRING, ZSTRING oder WSTRING, der in Großbuchstaben zurückgegeben werden soll.
- 'Modus' gibt den ASCII-Modus an, wobei der Wert 1 den Modus aktiviert. Wird 0 angegeben oder 'Modus' ausgelassen, so wird der ASCII-Modus deaktiviert. Ist der ASCII-Modus aktiv, so werden nur ASCII-Zeichen unterstützt statt der durch die Konsole eingestellte Lokalisierung.
- Der Rückgabewert ist der in Großbuchstaben gewandelte STRING bzw. WSTRING.

Achtung: UCASE wandelt die ASCII-Zeichen "a" - "z" um. Welche weiteren Zeichen umgewandelt werden, hängt vom Parameter 'Modus' sowie von der systeminternen Lokalisierung ab. Wenn Sie Unicode-Dateien verwenden, ist es außerdem nötig, das Byte Order Mark (BOM) zu setzen, damit Sonderzeichen wie z. B. Umlaute korrekt behandelt werden.

Das Dollarzeichen ($) als Suffix ist optional.

Beispiel:

```
PRINT UCASE("Hello World abCDefGHäÄ", 1)
SLEEP
```

Ausgabe:

```
"HELLO WORLD ABCDEFGHäÄ"
```

Hinweis: Im Grafikfenster wird eine andere Codepage verwendet als in der Konsole, weshalb Umlaute andere Codenummern besitzen und nicht mit UCASE bearbeitet werden können. Sie müssen bei Bedarf gesondert umgewandelt werden.

Unterschiede zu QB:

- In QB ist das Suffix $ verbindlich.
- Da QB kein Unicode unterstützt, existiert dort auch nicht die Möglichkeit, WSTRING zu verwenden.

Unterschiede zu früheren Versionen von FreeBASIC:
Der Modus-Parameter existiert seit FreeBASIC v0.90.

Plattformbedingte Unterschiede:
Unicode wird in der DOS-Portierung von FreeBASIC nicht unterstützt, deshalb kann dort auch kein WSTRING verwendet werden.

Unterschiede unter den FB-Dialektformen:

- In der Dialektform -lang qb ist das Suffix $ verbindlich.
- In den Dialektformen -lang fblite und -lang fb ist das Suffix optional.

Siehe auch:
LCASE, String-Funktionen

Letzte Bearbeitung des Eintrags am 03.07.13 um 22:26:12
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

UINTEGER

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » U » **UINTEGER**
Syntax:

```
DIM AS UINTEGER variable
DIM AS UINTEGER<Bits> variable
```

Typ: Datentyp

Bei der x86-Version des Compilers ist ein UINTEGER eine vorzeichenlose 32-bit-Ganzzahl (vgl. ULONG). Sie liegt im Bereich von 0 bis (2^32)-1, bzw. von 0 bis 4'294'967'295.
Bei der x64-Version des Compilers ist ein UINTEGER eine vorzeichenlose 64-bit-Ganzzahl (vgl. ULONGINT). Sie liegt im Bereich von 0 bis (2^64)-1, bzw. von 0 bis 18'446'744'073'709'551'615.

Wird der Parameter 'Bits' angegeben, so kann die exakte Größe des Datentyps definiert werden. Erlaubte Werte sind dabei 8, 16, 32 und 64.

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC:
Der 'Bits'-Parameter existiert seit FreeBASIC v0.90.

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht UINTEGER nicht zur Verfügung und kann nur über **__UINTEGER** aufgerufen werden.

Siehe auch:
DIM, CAST, CUINT, Datentypen, Datentypen und Deklarationen

Letzte Bearbeitung des Eintrags am 04.07.13 um 18:00:43
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

ULONG

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » U » **ULONG**
Typ: Datentyp

ULONG-Zahlen sind vorzeichenlose 32-bit-Ganzzahlen. Sie liegen im Bereich von 0 bis (2^32)-1, bzw. von 0 bis 4'294'967'295.

Unterschiede zu QB: neu in FreeBASIC

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht ULONG nicht zur Verfügung und kann nur über **__ULONG** aufgerufen werden.

Siehe auch:
SIZEOF, ANY, PTR, DIM, CAST, CULNG, Datentypen, Datentypen und Deklarationen

Letzte Bearbeitung des Eintrags am 30.06.13 um 14:16:16
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

ULONGINT

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » U » **ULONGINT**
Typ: Datentyp

ULONGINT-Zahlen sind vorzeichenlose 64-bit-Ganzzahlen. Sie liegen im Bereich von 0 bis (2^64)-1, bzw. von 0 bis 18'446'744'073'709'551'615.

Unterschiede zu QB: neu in FreeBASIC

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht ULONGINT nicht zur Verfügung und kann nur über **__ULONGINT** aufgerufen werden.

Siehe auch:
DIM, CAST, CULNGINT, Datentypen, Datentypen und Deklarationen

Letzte Bearbeitung des Eintrags am 14.07.10 um 00:23:12
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

UNDEF (Metabefehl)

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » U » **UNDEF (Metabefehl)**
Syntax: #UNDEF Symbol
Typ: Metabefehl
Kategorie: Metabefehle

#UNDEF löscht eine Definition, die mit #DEFINE erstellt wurde. Dies kann nützlich sein, wenn ein Makro nur in einem bestimmten Codeteil gültig sein soll, ein Symbol später als Variable verwendet werden soll oder der Wert einer Konstanten geändert werden muss. Der Befehle kann außerdem dazu verwendet werden, um von FreeBASIC reservierte Schlüsselwörter freizugeben.

#UNDEF darf nicht benutzt werden, um Variablen- oder Prozedurnamen im aktuellen SCOPE-Block zu entfernen. Diese Namen werden intern vom Compiler benötigt; das Löschen dieser Bezeichner kann zu unerwarteten Ergebnissen führen.

Beispiel:

```
"hlzeichen">(a_, b_) ((a_) + (b_))
Print Add2(1, 2)

' Makro wird nicht mehr benötigt; es wird entfernt.
"hlkommentar">' Jetzt kann das Symbol wieder verwendet werden:
Dim Add2 As Integer
```

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC:
Seit FreeBASIC v0.17 ersetzt "reflinkicon" href="temp0273.html">NOKEYWORD.

Siehe auch:
DEFINE (Meta), MACRO (Meta), IF (Meta), IFDEF (Meta), IFNDEF (Meta), DEFINED, Präprozessoren, Präprozessor-Anweisungen

Letzte Bearbeitung des Eintrags am 28.12.12 um 00:46:24
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

UNION

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » U » **UNION**
Syntax:

```
UNION &"hlzeichen">] [EXTENDS type_union&"hlzeichen">[FIELD =
{1|2|4}&"hlzeichen">[AS Typ&"hlzeichen">[Element2 [AS Typ&"hlzeichen">]
  [...]
END UNION
```

Typ: Anweisung
Kategorie: Klassen

UNION definiert einen UDT, dessen Elemente sich eine Speicherstelle teilen. Da die Elemente denselben Speicherplatz belegen, wird durch die Änderung eines Elements auch jedes andere Element geändert.

Eine UNION kann auch Teil eines anderen UDTs sein.

- 'UnionName' ist der Bezeichner der UNION, für den dieselben Regeln wie für den Bezeichner eines TYPE gelten. Dieser Bezeichner muss immer dann angegeben werden, wenn die UNION selbst den UDT darstellt. Ist die UNION Teil eines anderen UDTs, so darf sie keinen eigenen Bezeichner besitzen
- EXTENDS wird in Verbindung mit Vererbung verwendet. Das Union kann so von anderen UNIONs und TYPEs erben.
- FIELD bestimmt, auf welche Größe das Feld ausgedehnt werden soll ("Padding").
- 'Element1', 'Element2', ... sind die Namen der Records, über die auf die UNION-Felder zugegriffen wird. Dies können auch Konstruktoren, Destruktoren, SUBs, FUNCTIONs und andere sein, die auch in UDTs erlaubt sind.
- 'Typ' legt den Datentyp des Elements fest. Er darf ein beliebiger FreeBASIC-Datentyp oder ein UDT sein.

Wie bereits erwähnt kann eine UNION auch Teil eines anderen UDTs sein; die einzelnen Elemente werden einfach ineinander geschachtelt:

```
TYPE TypeName
  Record1 AS Typ
  UNION
    UnionElement1 AS Typ
    UnionElement2 AS Typ
  END UNION
END TYPE
```

Solche verschachtelten UNIONs wie im obigen Beispiel dürfen keinen eigenen Namen besitzen. Auf 'UnionElement1' und 'UnionElement2' wird über 'TypeName' zugegriffen; sie zählen formal auch als Records von 'TypeName', teilen sich allerdings dieselbe Speicherstelle.

Beispiel:

```
' Eine UNION definieren
UNION AUnion
  a As UBYTE
  b As INTEGER
END UNION
```

```
' Ein normaler UDT, der später in einer UNION
' verwendet wird.
TYPE Words
  LoWord AS SHORT
  HiWord AS SHORT
END TYPE

' Einen verschachtelten UDT definieren
TYPE CompType
  s AS STRING * 20
  ' Flag zur Anzeige, was in der Union
  ' benutzt werden soll
  uType AS BYTE
  UNION
    au AS UBYTE
    bu AS INTEGER
    cu AS Words
  END UNION
END TYPE


' Flags zur Anzeige, was in der Union benutzt
' werden soll. Es kann nur ein Element einer
' Union benutzt werden
CONST IsInteger = 1
CONST IsUByte   = 2
CONST IsUDT     = 3

DIM AS AUnion   MyUnion
DIM AS CompType MyComposite

' Wird ein Wert einer Union geändert...
MyUnion.a = 128
' ... ändern sich auch alle anderen Records.
PRINT MyUnion.b
PRINT

MyComposite.s = "Type + Union"
' Anhand dieses Flags könnte das Programm später
' erkennen, dass es den UNION-Wert wie einen
' INTEGER behandeln soll.
MyComposite.uType = IsInteger
MyComposite.bu = 1500 ' Der Wert der UNION selbst.

PRINT "Zusammengesetzter Typ: ";
SELECT CASE MyComposite.uType
  CASE IsInteger
    PRINT MyComposite.bu
  CASE IsUByte
    PRINT MyComposite.au
  CASE IsUDT
    PRINT MyComposite.cu.LoWord, _
              MyComposite.cu.HiWord
  CASE ELSE
```

```
    PRINT "Unbekannter Typ ..."
END SELECT
PRINT

PRINT "Wenn bei der Typenpruefung gemurkst wird:"
PRINT "MyComposite:"
PRINT ".au        (UBYTE)   "; MyComposite.au
PRINT ".bu        (INTEGER)"; MyComposite.bu
PRINT ".cu.LoWord (SHORT)   "; MyComposite.cu.LoWord
PRINT ".cu.HiWord (SHORT)   "; MyComposite.cu.HiWord

SLEEP
```

Ausgabe:

```
128

Zusammengesetzter Typ:  1500

Wenn bei der Typenpruefung gemurkst wird:
MyComposite:
.au        (UBYTE)   220
.bu        (INTEGER) 1500
.cu.LoWord (SHORT)   1500
.cu.HiWord (SHORT)   0
```

Beachten Sie, dass Strings unbestimmter Länge schwierig zu verwalten sind. Sie sollten stattdessen nach Möglichkeit in Ihren UNIONs Strings fester Länge verwenden.

UNIONs können verwendet werden, wenn eine Information oft in Form verschiedener Datentypen benötigt wird. Z. B. kann die Information '128', die in 'MyUnion' gespeichert ist, bequem als INTEGER und als UBYTE behandelt werden, ohne dazu die CAST-Funktion verwenden zu müssen. Der vermeintlich zusätzliche Aufwand, der mit dem Flag 'uType' in 'MyComposite' verbunden ist, muss tatsächlich sowieso aufgebracht werden, da auf die eine oder andere Weise erkannt werden muss, wie die Information zu handhaben ist. Praktisch ist auch die Möglichkeit, auf einzelne Bestandteile zuzugreifen, wie es in diesem Beispiel mit dem UDT 'Words' geschieht.

Unterschiede zu QB: neu in FreeBASIC

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht UNION nicht zur Verfügung und kann nur über **__UNION** aufgerufen werden.

Siehe auch:
TYPE, url=ref:634&"footer"> Letzte Bearbeitung des Eintrags am 14.09.13 um 14:48:37
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

UNLOCK

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » U » **UNLOCK**
Syntax: UNLOCK Dateinummer, { Satznr. | Start TO Ende }
Typ: Anweisung
Kategorie: Dateien

UNLOCK entsperrt eine zuvor mit LOCK gesperrte Datei bzw. einen Teil einer Datei.
Achtung:
Offensichtlich funktioniert das UNLOCK-Kommando nicht wie vorgesehen.

Siehe LOCK für Details zu den Parametern.

Letzte Bearbeitung des Eintrags am 28.12.12 um 00:47:43
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

UNSIGNED

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » U » **UNSIGNED**
Syntax: AS UNSIGNED { BYTE | SHORT | INTEGER | LONG | LONGINT }
Typ: Schlüsselwort
Kategorie: Datentypen

UNSIGNED erzeugt einen ganzzahligen Datentyp, der vorzeichenlos ist. Eine Variable dieses Typs kann keine negativen Zahlen enthalten, dafür ist der Maximalwert doppelt so groß.

Das Schlüsselwort wurde aus Kompatibilitätsgründen beibehalten. Verwenden Sie stattdessen die gleichbedeutenden Typen UBYTE, USHORT, UINTEGER, ULONG und ULONGINT.

Beispiel:

```
DIM x AS UNSIGNED INTEGER, y AS UINTEGER

x = -1
y = -1
PRINT x, y
SLEEP
```

Ausgabe:

```
4294967295    4294967295
```

Unterschiede zu QB: neu in FreeBASIC

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht UNSIGNED nicht zur Verfügung und kann nur über **__UNSIGNED** aufgerufen werden.

Siehe auch:
DIM, Datentypen, Datentypen und Deklarationen

Letzte Bearbeitung des Eintrags am 28.12.12 um 00:48:10
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

UNTIL

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » U » **UNTIL**
Syntax A:

```
Do Until Bedingung
  ' Anweisungen
Loop
```

Syntax B:

```
Do
  ' Anweisungen
Loop Until Bedingung
```

Typ: Schlüsselwort
Kategorie: Programmablauf

UNTIL wird mit DO...LOOP verwendet.

Beispiel:

```
Dim As Integer a = 1

Do
  Print "Hallo"
  a = a + 1
Loop Until a > 10

Sleep
```

Die Schleife wird so lange ausgeführt, bis a größer ist als 10. Wenn a den Wert 10 überschreitet, dann wird die Schleife verlassen.

Siehe auch:
DO ... LOOP, WHILE, Schleifen

Letzte Bearbeitung des Eintrags am 28.12.12 um 00:52:21
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

USHORT

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » U » **USHORT**
Typ: Datentyp

Daten vom Typ USHORT sind vorzeichenlose 16-bit-Ganzzahlen. Sie liegen im Bereich von 0 bis (2^16)-1, bzw. von 0 bis 65535.

Unterschiede zu QB: neu in FreeBASIC

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht USHORT nicht zur Verfügung und kann nur über **__USHORT** aufgerufen werden.

Siehe auch:
DIM, CAST, CUSHORT, Datentypen, Datentypen und Deklarationen

Letzte Bearbeitung des Eintrags am 14.07.10 um 00:19:46
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

USING (Namespace)

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » U » **USING (Namespace)**
Syntax: USING Bereichsname
Typ: Anweisung
Kategorie: Deklaration

USING bindet die Symbole eines Namespaces in den globalen Namespace ein, sofern bisher kein gleichnamiger Bezeichner vorhanden ist.
Vor der USING-Zeile wird der Code so behandelt, als würden die Symbole des Namespaces noch nicht existieren; ein Zugriff auf sie kann bis dahin nur über die allgemeine Syntax Bereichsname.Bezeichner geschehen. Nach der USING-Zeile werden die eingebundenen Bezeichner behandelt, als wären sie außerhalb eines Namespaces definiert worden.

Beispiel:

```
NAMESPACE Bsp
  TYPE T
    x AS INTEGER
  END TYPE

  TYPE nichtEingebunden
    x AS INTEGER
  END TYPE
END NAMESPACE

TYPE nichtEingebunden
  x AS SINGLE
END TYPE

' Ohne das Präfix 'Bsp.' kann kein Typ 'T' gefunden werden.
DIM a AS Bsp.T

' Jetzt wird der komplette Namespace 'Bsp' im globalen Space eingebunden.
Ausnahme
' ist der Typ 'nichtEingebunden', da ein gleichnamiger Bezeichner bereits
existiert.
USING Bsp

' Nun ist dies eine zulässige Anweisung, da 'Bsp' eingebunden wurde.
DIM b As T
```

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC: existiert seit FreeBASIC v0.16

Siehe auch:
NAMESPACE, PRINT USING, Datentypen und Deklarationen

Letzte Bearbeitung des Eintrags am 28.12.12 um 00:52:51
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

VAL

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » V » **VAL**
Syntax: VAL (Stringvariable)
Typ: Funktion
Kategorie: Typumwandlung

VAL konvertiert einen STRING zu einer Zahl.

- 'Stringvariable' ist ein STRING, ZSTRING oder WSTRING, welcher den umzuwandelnden Zahlenwert enthält.
- Der Rückgabewert ist ein DOUBLE.

Die Zahl muss am Anfang des STRINGs stehen. Führende Leerzeichen werden jedoch entfernt. Sobald VAL auf ein Zeichen stößt, das nicht zu einer Zahl gehören kann, wird an dieser Stelle abgebrochen und nur der davor stehende Wert umgewandelt. Die wissenschaftliche Notation (z. B. "1.0e+23") wird unterstützt. Wenn der String ungültig ist (keine Zahl enthält oder nicht mit einer Zahl beginnt), wird als Ergebnis 0 ausgegeben.

Wenn Sie einen STRING in eine Ganzzahl umwandeln wollen, nutzen Sie besser VALINT o. ä.; siehe dazu die Zusammenstellung Datentypen umwandeln

VAL verwandelt auch Zahlen anderer Zahlensysteme ins Dezimalsystem zurück. Die STRINGs müssen dazu mit dem entsprechenden Präfix ("Vorsatz") versehen sein:

- **&h** - Hexadezimal
- **&o** oder **&** - Oktal
- **&b** - Binär

Beispiel:

```
PRINT VAL("10")        ' Ausgabe 10
PRINT VAL("-271.3")    ' Ausgabe -271.3
PRINT VAL("&h10")      ' Ausgabe 16
PRINT VAL("&o10")      ' Ausgabe 8
PRINT VAL("&10")       ' Ausgabe 8
PRINT VAL("&b10")      ' Ausgabe 2
PRINT VAL("123a45")    ' Ausgabe 123
PRINT VAL("  10")      ' Ausgabe 10
PRINT VAL(".234")      ' Ausgabe 0.234
PRINT VAL("a321")      ' Ausgabe 0
PRINT VAL("+123")      ' Ausgabe 123
PRINT VAL("2.1E+30")   ' Ausgabe 2.1e+30
SLEEP
```

"Umkehrfunktionen" zu VAL, die den Wert einer Variablen numerischen Datentyps in eine bestimmte Notation umschreiben und einen STRING zurück liefern, sind u. a.:

- HEX: wandelt eine als Argument übergebene Zahl (z. B. vom Typ INTEGER) in ihre Hexadezimaldarstellung um.
- BIN: liefert die Darstellung einer übergebenen Zahl im Dualsystem zurück.
- FORMAT: formatiert eine übergebene Zahl in Bezug auf Nachkommastellen, führende Nullen usw.
- STR: wandelt den Wert einer Variablen numerischen Typs in eine Zeichenkette um.
- und weitere; siehe unten

Unterschiede zu früheren Versionen von FreeBASIC:
Seit FreeBASIC v0.21.0 akzeptiert VAL ein führendes "&" für Oktalwerte (z. B. "&123")

Siehe auch:
VALINT, VALUINT, VAL64, VALLNG, VALULNG, BIN, HEX, OCT, STR, WBIN, WHEX, WOCT, WSTR, Datentypen umwandeln

Letzte Bearbeitung des Eintrags am 28.12.12 um 00:53:22
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

VAL64

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » V » **VAL64**
Syntax: VAL64 (Variable)
Typ: Funktion
Kategorie: Typumwandlung

VAL64 wandelt einen STRING, ZSTRING oder WSTRING in einen LONGINT um.

Der Befehl **existierte nur bis FreeBASIC v0.14**. In FreeBASIC v0.15 wurde diese Funktion durch VALLNG ersetzt.

Siehe auch:
VAL, VALINT, VALUINT, VALLNG, VALULNG, Datentypen umwandeln

Letzte Bearbeitung des Eintrags am 28.12.12 um 00:53:33
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

VALINT

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » V » **VALINT**
Syntax: VALINT (Variable)
Typ: Funktion
Kategorie: Typumwandlung

VALINT wandelt einen STRING, ZSTRING oder WSTRING in einen INTEGER um. Die wissenschaftliche Notation wird nicht unterstützt. Ansonsten arbeitet der Befehl wie VAL, nur dass VAL in eine DOUBLE-Gleitkommazahl umwandelt.

Beispiel:

```
PRINT VALINT(".12345")      ' Ausgabe 0
PRINT VALINT("&h1ABC")      ' Ausgabe 6844
PRINT VALINT("   -42")      ' Ausgabe -42
PRINT VALINT("12.987")      ' Ausgabe 12
PRINT VALINT("133e7")       ' Ausgabe 133

PRINT 'Leerzeile

PRINT "&hFFFFFFFF als INTEGER:  " & VALINT("&hFFFFFFFF")  ' Ausgabe -1
' Vergleiche dazu:
PRINT "&hFFFFFFFF als UINTEGER: " & VALUINT("&hFFFFFFFF") ' Ausgabe
4294967295

SLEEP
```

Hinweis: VALUINT funktioniert analog zu VALINT, liefert jedoch einen vorzeichenlosen UINTEGER zurück.

Achtung:
VALINT schneidet Nachkommastellen ab, es wird nicht gerundet!

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC:
Seit FreeBASIC v0.21.0 akzeptiert VALINT ein führendes "&" für Oktalwerte (z. B. "&123")

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht VALINT nicht zur Verfügung und kann nur über **__VALINT** aufgerufen werden.

Siehe auch:
VAL, VALLNG, VALUINT, VALULNG, Datentypen umwandeln

Letzte Bearbeitung des Eintrags am 28.12.12 um 00:53:44
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

VALLNG

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » V » **VALLNG**
Syntax: VALLNG (Variable)
Typ: Funktion
Kategorie: Typumwandlung

VALLNG wandelt einen STRING, ZSTRING oder WSTRING in einen LONGINT um. Die wissenschaftliche Notation wird nicht unterstützt. Ansonsten arbeitet der Befehl wie VAL, nur dass VAL in eine DOUBLE-Gleitkommazahl umwandelt.

Beispiel:

```
Print ValLng(".12345")      ' Ausgabe 0
Print ValLng("&H1ABC")      ' Ausgabe 6844
Print ValLng("   -42")      ' Ausgabe -42
Print ValLng("12.987")      ' Ausgabe 12
PRINT VALlng("133e7")       ' Ausgabe 133

Print 'Leerzeile

Print "&hFFFFFFFFFFFFFFFF als LONGINT:  ";
Print ValLng("&hFFFFFFFFFFFFFFFF")  ' Ausgabe -1
' Vergleiche dazu:
Print "&hFFFFFFFFFFFFFFFF als ULONGINT: ";
Print ValULng("&hFFFFFFFFFFFFFFFF") ' Ausgabe 18446744073709551615

Sleep
```

Hinweis: VALULNG funktioniert analog zu VALLNG, liefert jedoch einen vorzeichenlosen ULONGINT zurück.

Achtung:
VALLNG schneidet Nachkommastellen ab, es wird nicht gerundet!

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC:

- VALLNG existiert seit FreeBASIC v0.15
- Seit FreeBASIC v0.21.0 akzeptiert VALLNG ein führendes "&" für Oktalwerte (z. B. "&123")

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht VALLNG nicht zur Verfügung und kann nur über **__VALLNG** aufgerufen werden.

Siehe auch:
VAL, VALINT, VALUINT, VALULNG, Datentypen umwandeln

Letzte Bearbeitung des Eintrags am 28.12.12 um 00:54:50
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

VALUINT

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » V » **VALUINT**
Syntax: VALUINT (Variable)
Typ: Funktion
Kategorie: Typumwandlung

VALUINT wandelt einen STRING, ZSTRING oder WSTRING in einen UINTEGER um. Die wissenschaftliche Notation wird nicht unterstützt. Ansonsten arbeitet der Befehl wie VAL, nur dass VAL in eine DOUBLE-Gleitkommazahl umwandelt.

Beispiel:

```
PRINT VALUINT(".12345")       ' Ausgabe 0
PRINT VALUINT("&h1ABC")       ' Ausgabe 6844
PRINT VALUINT("   -42")       ' Ausgabe 4294967254
PRINT VALUINT("12.987")       ' Ausgabe 12
PRINT VALUINT("133e7")        ' Ausgabe 133
PRINT VALUINT("    -1")       ' Ausgabe 4294967295
PRINT VALUINT("&hFFFFFFFF")   ' Ausgabe 4294967295
SLEEP
```

Hinweis: VALINT funktioniert wie VALUINT, liefert jedoch einen vorzeichenbehafteten INTEGER zurück.

Achtung:
VALUINT schneidet Nachkommastellen ab, es wird nicht gerundet!

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC:

- VALUINT existiert seit FreeBASIC v0.15
- Seit FreeBASIC v0.21.0 akzeptiert VALUINT ein führendes "&" für Oktalwerte (z. B. "&123")

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht VALUINT nicht zur Verfügung und kann nur über **__VALUINT** aufgerufen werden.

Siehe auch:
VAL, VALINT, VALLNG, VALULNG, Datentypen umwandeln

Letzte Bearbeitung des Eintrags am 28.12.12 um 00:55:38
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

VALULNG

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » V » **VALULNG**
Syntax: VALULNG (Variable)
Typ: Funktion
Kategorie: Typumwandlung

VALULNG wandelt einen STRING, ZSTRING oder WSTRING in einen ULONGINT um. Die wissenschaftliche Notation wird nicht unterstützt. Ansonsten arbeitet der Befehl wie VAL, nur dass VAL in eine DOUBLE-Gleitkommazahl umwandelt.

Beispiel:

```
Print ValULng(".12345")              ' Ausgabe 0
Print ValULng("&h1ABC")              ' Ausgabe 6844
Print ValULng("   -42")              ' Ausgabe 18446744073709551574
Print ValULng("12.987")              ' Ausgabe 12
PRINT ValULng("133e7")               ' Ausgabe 133
Print ValULng("    -1")              ' Ausgabe 18446744073709551615
Print ValULng("&hFFFFFFFFFFFFFFFF")  ' Ausgabe 18446744073709551615
Sleep
```

Hinweis: VALLNG funktioniert wie VALULNG, liefert jedoch einen vorzeichenbehafteten LONGINT zurück.

Achtung:
VALULNG schneidet Nachkommastellen ab, es wird nicht gerundet!

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC:

- VALULNG existiert seit FreeBASIC v0.15
- Seit FreeBASIC v0.21.0 akzeptiert VALULNG ein führendes "&" für Oktalwerte (z. B. "&123")

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht VALULNG nicht zur Verfügung und kann nur über **__VALULNG** aufgerufen werden.

Siehe auch:
VAL, VALINT, VALUINT, VALLNG, Datentypen umwandeln

Letzte Bearbeitung des Eintrags am 28.12.12 um 00:55:49
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

VAR

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » V » **VAR**
Syntax: VAR [SHARED] SymbolName = Ausdruck[, SymbolName = Ausdruck [, ...]]
Typ: Anweisung
Kategorie: Deklaration

VAR deklariert eine Variable, deren Typ aus dem initialisierenden Ausdruck abgeleitet wird.

Bei einer Deklaration mit VAR ist es nicht erlaubt, explizit einen Variablentyp anzugeben. Stattdessen wird der Typ aus dem initialisierenden Ausdruck ermittelt. Dieser Ausdruck kann eine Konstante oder eine Variable eines beliebigen Typs sein. Einzige Ausnahme bildet der WSTRING, der von VAR nicht unterstützt wird, da es keinen WSTRING-Typ mit variabler Länge gibt. Das wird sich höchstwahrscheinlich auch nicht ändern, wegen der Komplexität, die durch die Behandlung von Unicode hervorgerufen wird.

Da der Variablentyp daraus abgeleitet wird, was der Variablen zugewiesen wird, ist es hilfreich, zu wissen, wie die Zeichenfolgen bewertet werden. Alle Ziffernfolgen **ohne** Dezimal-Punkt sind standardmäßig INTEGER. Alle Ziffernfolgen **mit** Dezimal-Punkt sind standardmäßig DOUBLE.

Alle alphanumerischen Strings sind standardmäßig STRING. Wird eine Variable mit WSTRING und einer Zeichenfolge initialisiert, wird sie "umgewandelt" zu einem String mit variabler Länge.

Bestimmte Suffixe können bei Variablenbezeichnern benutzt werden, um ihren Typ festzulegen. Siehe Datentypen zu weiteren Informationen über Suffixe, die nach einer Zeichenfolge erlaubt sind.

Beachte: Suffixe müssen hinter dem Initialisierer erscheinen, nicht hinter der Variablen. Der Versuch, Variablen zu verwenden, die ein Suffix besitzen, ruft einen Compiler-Fehler hervor.

Beispiele:

```
Var a  = Cast(Byte, 0)
Var b  = Cast(Short, 0)
Var c  = Cast(Integer, 0)
Var d  = Cast(LongInt, 0)
Var d2 = 0ll     ' LONGINT, festgelegt über das Suffix
Var au = Cast(UByte, 0)
Var bu = Cast(UShort, 0)
Var cu = Cast(UInteger, 0)
Var du = Cast(ULongInt, 0)
Var e  = Cast(Single, 0.0)
Var f  = Cast(Double, 0.0)
Var g  = @c      ' Integer Pointer
Var h  = @a      ' Byte Pointer
Var s2 = "hello" ' String mit Variabler Länge

Var ii = 6728   ' impliziter Integer
Var id = 6728.0 ' impliziter Double

Print "Byte: ";len(a)
Print "Short: ";len(b)
Print "Integer: ";len(c)
Print "Longint: ";len(d)
Print "noch ein Longint: ";len(d2)
Print "UByte: ";len(au)
```

```
Print "UShort: ";len(bu)
Print "UInteger: ";len(cu)
Print "ULongint: ";len(du)
Print "Single: ";len(e)
Print "Double: ";len(f)
Print "Integer Pointer: ";len(g)
Print "Byte Pointer: ";len(h)
Print "Variabler String: ";len(s2)
Print
Print "Integer: ";len(ii)
Print "Double: ";len(id)

Sleep
```

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC: existiert seit FreeBASIC v0.17

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb ist die Verwendung von VAR nicht möglich.

Siehe auch:
DIM, REDIM, COMMON, STATIC (Anweisung), SHARED, CONST, Datentypen und Deklarationen

Letzte Bearbeitung des Eintrags am 28.12.12 um 00:56:18
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

VARPTR

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » V » **VARPTR**
Syntax: VARPTR (Variable)
Typ: Funktion
Kategorie: Speicher

VARPTR gibt die Adresse einer Variablen im Speicher zurück. Damit wurden in älteren BASIC-Dialekten Pointer-Funktionen erstellt. In FreeBASIC wird dazu üblicherweise das Zeichen @ verwendet; dort finden Sie weitere Informationen zum Gebrauch.

Beispiel:

```
DIM AS INTEGER a
DIM AS INTEGER PTR addr
a = 10
addr = VARPTR(a)   ' identisch mit addr = @a
POKE INTEGER, addr, -1000
PRINT a, HEX(a)
POKE BYTE, addr, 1
PRINT a, HEX(a)
SLEEP
```

Ausgabe:

```
-1000          FFFFFC18
-1023          FFFFFC01
```

Wie Sie sehen, wird das letzte Byte (&h18) durch den Wert 1 (&h01) ersetzt. Ob das erste oder letzte signifikante Byte ersetzt wird, hängt von der Byte-Reihenfolge der CPU ab.

Siehe auch:
@, OFFSETOF, SADD, PEEK, POKE, Grundlagen zu Pointern, Zusammenstellung von Pointer-Anweisungen

Letzte Bearbeitung des Eintrags am 26.12.12 um 01:03:48
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

VA_ARG

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » V » **VA_ARG**
Syntax: VA_ARG (Startpointer, Datentyp)
Typ: Funktion
Kategorie: Speicher

VA_ARG gibt den Wert eines Parameters in der Parameterliste einer Prozedur zurück.

- 'Startpointer' ist ein Pointer, der mit VA_FIRST ermittelt bzw. mit VA_NEXT aktualisiert wurde.
- 'Datentyp' ist der Datentyp des Parameters, dessen Wert zurückgegeben werden soll. Siehe auch Datentypen.

VA_ARG wird intern folgendermaßen behandelt:

```
#DEFINE VA_ARG(a,t) PEEK(t, a)
```

Beispiel: siehe VA_FIRST

Unterschiede zu QB: neu in FreeBASIC

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht VA_ARG nicht zur Verfügung und kann nur über **__VA_ARG** aufgerufen werden.

Siehe auch:
VA_FIRST, VA_NEXT, DECLARE, SUB, FUNCTION, CDECL, Datentypen, ... (Auslassung[Ellipsis]), Prozeduren

Letzte Bearbeitung des Eintrags am 28.12.12 um 00:56:52
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

VA_FIRST

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » V » **VA_FIRST**
Syntax: VA_FIRST
Typ: Funktion
Kategorie: Speicher

VA_FIRST gibt einen Pointer auf den ersten Parameter einer Prozedur zurück.

Unter FreeBASIC ist es möglich, Prozeduren zu erstellen, die keine festgelegte Parameterliste besitzen. Diesen Prozeduren darf eine beliebig lange Parameterliste übergeben werden, und die Parameter dürfen von jedem Datentyp sein. Eine solche Liste wird variable Parameterliste genannt. Um dies zu ermöglichen, verwendet man bei DECLARE und im Prozedurheader folgende Syntax:

```
DECLARE {SUB|FUNCTION} (<'normaleParameter'>, ...)
```

'normaleParameter' ist dabei eine normale Parameterliste, wie bei DECLARE erklärt. Das '...' leitet die variable Parameterliste ein.

Bei dieser Form der Übergabe geht aber die Information verloren, um welche Datentypen es sich in der variablen Liste handelt bzw. wie viele Parameter übergeben wurden. Daher werden oft sogenannte Format-Strings in der normalen Parameterliste übergeben, in denen beschrieben wird, welche Parameter übergeben wurden. Dieser Format-String darf jedes beliebige Format besitzen, da der Programmierer die Parameterübergabe komplett selbst übernimmt. Der String muss ihm lediglich die Informationen liefern, die er braucht, um die Werte aus der variablen Parameterliste herauszulesen.

VA_ARG wird dazu benutzt, um den Wert eines Parameters zu ermitteln. VA_NEXT aktualisiert den von VA_FIRST zurückgegebenen Pointer, sodass er auf das nächste Argument der variablen Parameterliste zeigt.

Beispiel (aus dem examples-Ordner des FreeBASIC-Verzeichnisses):

```
' Beispiel für variable Parameter
Declare Sub myPrintf cdecl (byRef FormatierString As String, ...)

Dim s as String = "bar"

myPrintf "Integer=%i, LongInt=%l, Single=%f, Double=%d, String=%s,
String=%s", _
         1, 1LL shl 32, 2.2, 3.3, "foo", s
Sleep

Sub myPrintf cdecl (byRef FormatierString As String, ...)
  Dim As Integer i, c
  Dim As String s
  Dim As Any Ptr Arg = va_first  ' Pointer zum ersten variablen Argument
holen
  Dim As ZString Ptr p

  p = StrPtr(FormatierString)
  i = Len(FormatierString)
  Do While i>0  ' Für jedes Zeichen in FormatierString..
    c = *p
    p += 1
    i -= 1
```

```
    ' Ist es ein Formatierzeichen?
    If Chr(c) = "%" Then  ' Typ holen
      c = *p
      p += 1
      i -= 1
      ' variables Argument ausgeben, abhängig vom Typ
      Select Case Chr(c)
        Case "i"  ' Integer?
          s = s & va_arg(Arg, Integer)
          Arg = va_next(Arg, Integer)
          ' Anders als in C muss va_next() benutzt werden, da
          ' va_arg() die Pointer nicht aktualisiert
        Case "l"  ' LongInteger? (64-bit)
          s = s & va_arg(Arg, LongInt)
          Arg = va_next(Arg, LongInt)
        Case "f", "d"
          ' Single oder Double? (Beachte: wegen der C-ABI werden alle
Singles, die
          ' übergeben wurden, in Doubles konvertiert)
          s = s & va_arg(Arg, Double)
          Arg = va_next(Arg, Double)
        Case "s"  ' String?
          ' Strings werden byVal übergeben, also ist ihre Länge unbekannt
          s = s & *va_arg(Arg, ZString Ptr)
          Arg = va_next(Arg, ZString Ptr)
      End Select
    Else  ' normales Zeichen, einfach ausgeben..
      s = s & Chr( c )
    End If
  Loop
  Print s;
End Sub
```

Beispiel 2 (verkürzte Variante):

```
Sub myPrint Cdecl (FormatierString As String, ...)
  Dim As Any Ptr argument = va_first
  Dim As Integer i
  Dim As ZString Ptr p

  For i = 1 To Len(FormatierString)

    Select Case Mid(FormatierString, i, 1)
      Case "i"
        Print "Integer:  " & va_arg(argument, Integer)
        argument = va_next(argument, Integer)
      Case "l"
        Print "LongInt:  " & va_arg(argument, LongInt)
        argument = va_next(argument, LongInt)
      Case "f", "d"
        Print "Double:   " & va_arg(argument, Double)
        argument = va_next(argument, Double)
      Case "s"
        Print "String:   " & *va_arg(argument, ZString Ptr)
```

```
        argument = va_next(argument, ZString Ptr)
    End Select
  Next
End Sub

'"hlkw0">Dim s As String = "String2"

myPrint( "ilfdss", 1, 4294967296, 1.2, 3.4, "String1", s )
Sleep
```

Hinweis:
Eine SUB/FUNCTION, die eine variable Parameterliste verwendet, muss zwingend mit CDECL deklariert werden. Ansonsten meldet der Compiler einen Fehler.

Unterschiede zu QB: neu in FreeBASIC

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht VA_FIRST nicht zur Verfügung und kann nur über **__VA_FIRST** aufgerufen werden.

Siehe auch:
VA_ARG, VA_NEXT, DECLARE, SUB, FUNCTION, CDECL, Datentypen, ... (Auslassung&"reflinkicon" href="temp0580.html">Prozeduren

Letzte Bearbeitung des Eintrags am 28.12.12 um 00:57:05
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

VA_NEXT

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » V » **VA_NEXT**
Syntax: VA_NEXT (Startpointer, Datentyp)
Typ: Funktion
Kategorie: Speicher

VA_NEXT dereferenziert den Pointer auf ein Argument einer variablen Parameterliste, sodass er auf das nächste Argument zeigt.

- 'Startpointer' ist ein Pointer, der mit VA_FIRST ermittelt bzw. mit VA_NEXT aktualisiert wurde.
- 'Datentyp' ist der Datentyp des Parameters, auf den 'Startpointer' bisher gezeigt hat. Siehe auch Datentypen.

VA_ARG wird intern folgendermaßen behandelt:

```
#DEFINE VA_NEXT(a,t) (a + LEN(t))
```

Beispiel: siehe VA_FIRST

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC:
Die Version __VA_NEXT in der Dialektform -lang qb existiert seit FreeBASIC v0.24.

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht VA_NEXT nicht zur Verfügung und kann nur über **__VA_NEXT** aufgerufen werden.

Siehe auch:
VA_FIRST, VA_ARG, DECLARE, SUB, FUNCTION, CDECL, Datentypen, ... (Auslassung[Ellipsis]), Prozeduren

Letzte Bearbeitung des Eintrags am 28.12.12 um 00:57:18
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

VIEW (Grafik)

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » V » **VIEW (Grafik)**
Syntax: VIEW [[SCREEN] (x1, y1)-(x2, y2) [,[Farbe][, Rand]]]
Typ: Anweisung
Kategorie: Grafik

VIEW definiert neue Gültigkeitsbereiche für physische Bildschirmkoordinaten, setzt also neue Clipping-Grenzen.

- Wenn das Argument SCREEN ausgelassen wird, sind alle Koordinaten zukünftiger Grafikanweisungen relativ zur linken oberen Ecke des Darstellungsfeldes anstelle zur linken oberen Ecke des Grafikfensters.
- 'x1', 'y1', 'x2' und 'y2' sind die Koordinaten zweier gegenüberliegender Eckpunkte des neuen Darstellungsfeldes. Sie sind relativ zur linken oberen Ecke des Grafikfensters.
- 'Farbe' ist eine Farbnummer, mit welcher der angegebene Bereich ausgefüllt werden soll. Wird dieser Parameter ausgelassen, bleibt die alte Grafik in diesem Bereich erhalten.
- 'Rand' ist eine Farbnummer, in der ein Rahmen um den angegebenen Bereich gezeichnet werden soll. Wird dieser Parameter ausgelassen, wird kein Rahmen gezeichnet.

VIEW wird benutzt, um neue Clipping-Grenzen zu setzen und damit ein neues Darstellungsfeld festzulegen. Ein neu definiertes Darstellungsfeld ersetzt ein eventuell bereits bestehendes. VIEW hat nur im Grafikfenster eine Auswirkung.

Alle Drawing Primitives sind von der Größe des Darstellungsfeldes abhängig; Grafikanweisungen, die auf Koordinaten außerhalb dieser Grenzen verweisen, haben keine Auswirkungen.

Farben müssen im selben Format angegeben werden wie bei COLOR (Anweisung). Sie sind abhängig von der Farbtiefe, die mit SCREENRES eingestellt wurde.

Wenn alle Argumente ausgelassen werden, setzt VIEW alle Grenzen auf den Standard zurück.

Beispiel:

```
Screenres 640, 480
Dim bild As Any Ptr
Dim As Integer i, j, k

' einfaches Sprite erstellen
bild = ImageCreate(64, 64)
For i = 0 To 63 : For j = 0 To 63
  PSet bild, (i,j), (i\4) Xor (j\4)
Next j, i

' blaues Kreuz zur Demonstration:
' Der Bereich außerhalb der Clipping-Grenzen ist später nicht mehr
betroffen
line (0, 0)-(639, 479), 1
line (639, 0)-(0, 479), 1
' Clipping-Grenzen setzen, mit blauem Rand
View (220,140)-(420,340),, 1

' Sprite bewegen
k = 0
```

```
Do
  i = 100* Sin(k*.02)+50
  j = 100* Sin(k*.027)+50
  ScreenSync
  ScreenLock
  Cls 1  ' löscht nur das Darstellungsfeld
  Put (i, j), bild, PSet
  ScreenUnlock
  k += 1
Loop Until Len(Inkey)
ImageDestroy bild
```

Unterschiede zu früheren Versionen von FreeBASIC:
In Versionen vor v0.16 hatte der Parameter 'Rand' keine Auswirkungen.

Unterschiede zu QB:

- QB behält die Koordinatentransformationen mittels WINDOW auch bei einer Größenänderung des Clipping-Bereichs in gleicher Form bei.
- FreeBASIC behält derzeit die Eckkoordinaten bei, wodurch sich die Koordinatentransformation ändert. Dieses Verhalten wird möglicherweise in einer zukünftigen Compilerversion geändert. Derzeit muss jedoch erneut WINDOW mit entsprechenden Parametern aufgerufen werden, um z. B. nach einem VIEW die vorherige Koordinatentransformation in gleicher Form beizubehalten. Andernfalls wird der Koordinatenbereich eventuell verschoben oder skaliert.

Siehe auch:
VIEW (Text), SCREENRES, WINDOW, PMAP, Grafik

Letzte Bearbeitung des Eintrags am 26.12.12 um 01:24:31
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

VIEW (Text)

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » V » **VIEW (Text)**
Syntax A: VIEW PRINT [Startzeile TO Endzeile]
Syntax B: VIEW PRINT()
Typ: Anweisung (A) bzw. Funktion (B)
Kategorie: Konsole

VIEW PRINT setzt die Grenzen des Textanzeigebereichs oder gibt diese zurück.

- 'Startzeile' und 'Endzeile' sind Zeilenangaben, wie sie auch bei LOCATE (Anweisung) verwendet werden. Der Textcursor wird auf den Beginn der Startzeile gesetzt.
- Werden die Parameter in Syntax A ausgelassen, dann wird das gesamte Fenster als Darstellungsbereich verwendet.
- Wird VIEW PRINT als Funktion eingesetzt (Syntax B), dann ist der Rückgabewert ein INTEGER, der die oberste und unterste Zeile des Textanzeigebereichs enthält. Die obere Zeile ist das untere Word (LOWORD), die untere Zeile ist das obere Word (HIWORD).

Der Textanzeigebereich hat Auswirkungen auf Textanweisungen in der Konsole und im Grafikfenster. Die auf VIEW PRINT folgenden Textanweisungen sind nur innerhalb der angegebenen Grenzen wirksam.

Beispiel 1:

```
SCREENRES 640, 400
DIM AS INTEGER i
WIDTH 80, 25
FOR i = 1 TO 24
  PRINT i
  SLEEP 200
NEXT

' Textanzeigebereich einschränken
VIEW PRINT 10 TO 15
SLEEP 200
COLOR 15, 1
CLS
FOR i = 1 TO 10
  PRINT i
  SLEEP 200
NEXT

' Grenzen des Anzeigebereichs ausgeben
DIM AS INTEGER bereich = VIEW PRINT()
PRINT "Angezeigt wurden die Zeilen " & LOWORD(bereich) & " - " &
HIWORD(bereich)
SLEEP
```

Beispiel 2:
Wird versucht, den Cursor mit LOCATE auf eine Zeile außerhalb des Anzeigebereichs zu setzen, dann hat LOCATE keine Auswirkung.

```
VIEW PRINT 10 TO 15
LOCATE 1, 5
PRINT "Zeile 1, Spalte 5"
```

```
LOCATE 13, 5
PRINT "Zeile 13, Spalte 5"
SLEEP
```

Unterschiede zu QB:
In FreeBASIC kann VIEW PRINT() als Funktion eingesetzt werden.

Unterschiede zu früheren Versionen von FreeBASIC:

- Seit FreeBASIC v0.15 kann VIEW PRINT() als Funktion eingesetzt werden.
- Seit FreeBASIC v0.15 kann VIEW PRINT auch in einem Grafikfenster eingesetzt werden.

Siehe auch:
VIEW (Grafik), CLS, Grafik

Letzte Bearbeitung des Eintrags am 26.12.12 um 01:28:45
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

VIRTUAL

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » V » **VIRTUAL**
Syntax:

```
TYPE Typname EXTENDS Elterntyp
  DECLARE VIRTUAL {SUB|FUNCTION|PROPERTY|OPERATOR} ...
END TYPE
```

Typ: Klausel
Kategorie: Klassen

Mit dem Schlüsselwort VIRTUAL werden virtuelle Methoden in Klassen deklariert. Diese können von erbenden Klassen überschrieben werden und erlauben so Polymorphismus. Im Gegensatz zu abstrakten Methoden müssen virtuelle einen implementierten Methodenkörper (body) besitzen, der verwendet wird, wenn die Methode nicht überschrieben wird.

Erbende Klassen können eine virtuelle Methode überschreiben, indem sie eine gleichnamige Methode mit denselben Parametern deklarieren. Die Eigenschaft, dass eine Methode virtuell ist, wird beim Vererben nicht implizit weitergegeben, sodass VIRTUAL angegeben werden muss, falls auch die Kindklasse die Methode als virtuell deklarieren soll.

Damit das Programm zur Laufzeit entscheiden kann, welcher Typ aktuell gebraucht wird, enthält die Klasse intern einen zusätzlichen Eintrag, eine sog. *vtable*, in der die nötigen Informationen enthalten sind. Damit dieser Eintrag hinzugefügt wird, **muss** eine Klasse mit virtuellen Methoden von OBJECT erben.

Konstruktoren können nicht virtuell sein, da ein Objekt, in diesem Fall vor allem dessen *vtable*, initialisiert werden muss. Der Compiler fügt darüber hinaus jedem Konstruktor an dessen Beginn eine Initialisierung der *vtable* hinzu. Wird nun in so einem Konstruktor eine virtuelle Methode verwendet, wird nur die Basisimplementation genutzt, da zum Zeitpunkt des Aufrufs die Initialisierung der Kindklassen nicht stattgefunden hat. Aus diesem Grund dürfen abstrakte Methoden nicht in einem Konstruktor verwendet werden.

Destruktoren können virtuell sein, da die *vtable* zum Zeitpunkt des Aufrufs initialisiert ist. Werden allerdings im Destruktor virtuelle Methoden aufgerufen, so darf es sich dabei nicht um Methoden von Kindklassen handeln, da alle Kindklassen bis zu diesem Zeitpunkt bereits zerstört wurden.

VIRTUAL kann für bessere Lesbarkeit des Codes auch beim Methodenkörper angegeben werden. Dies ist allerdings nicht zwingend erforderlich, wichtig ist die Angabe in der Deklaration.

Beachte:
In einer mehrstufigen Vererbungshierarchie kann eine überschriebene Methode auf jeder Ebene als abstrakt, virtuell oder normal deklariert werden. Werden die Varianten gemischt, gilt folgende Reihenfolge von oben nach unten in der Hierarchie: Abstrakt -> Virtuell -> Normal.

Beispiel:

```
Type Hello Extends Object
  Declare Virtual Sub hi
End Type

Type HelloEnglish Extends Hello
  Declare Sub hi
End Type
```

```
Type HelloFrench Extends Hello
  Declare Sub hi
End Type

Type HelloGerman Extends Hello
  Declare Sub hi
End Type


Sub Hello.hi
  Print "Hi!"
End Sub

Sub HelloEnglish.hi
  Print "Hello!"
End Sub

Sub HelloFrench.hi
  Print "Salut!"
End Sub

Sub HelloGerman.hi
  Print "Hallo!"
End Sub


Randomize

Dim As Hello Ptr h

For i As Integer = 0 To 9
  Select Case Int(Rnd*4) + 1
    Case 1
      h = New HelloEnglish
    Case 2
      h = New HelloFrench
    Case 3
      h = New HelloGerman
    Case Else
      h = New Hello
  End Select

  h->hi
  Delete h
Next

Sleep
```

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC: existiert seit FreeBASIC v0.90.0

Unterschiede unter den FB-Dialektformen: nur in der Dialektform -lang fb verfügbar

Siehe auch:
ABSTRACT, TYPE, EXTENDS, OBJECT, Objektorientierung

Letzte Bearbeitung des Eintrags am 25.06.13 um 23:40:22
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

WAIT

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » W » **WAIT**
Syntax: WAIT Port, AND_Ausdruck[, XOR_Ausdruck]
Typ: Anweisung
Kategorie: Hardware

WAIT liest regelmäßig ein Byte von einem Port und wartet mit der Programmausführung, bis dieses Byte bestimmte Bedingungen erfüllt.

- 'Portnummer' ist die Nummer des Ports, von dem gelesen werden soll. Zugriffe auf den VGA-Port werden emuliert.
- Der gelesene Wert wird über ein logisches AND mit dem 'AND_Ausdruck' verknüpft. Die Programmausführung wird fortgesetzt, sobald das Ergebnis dieses bitweisen Vergleichs ungleich 0 ist.
- Wird ein 'XOR_Ausdruck' angegeben, wird der eingelesene Wert über ein logisches XOR mit ihm verknüpft, bevor er an den AND_Ausdruck übergeben wird. Wenn dieses Argument ausgelassen wird, nimmt FreeBASIC 0 an.

Wenn der Zugriff auf den Port fehlschlägt, wird ein Laufzeit-Fehler erzeugt.

Unterschiede zu früheren Versionen von FreeBASIC:
Vor FreeBASIC v0.14 emulierte WAIT das Warten auf die vertikale Bildschirmsynchronisation (VSync). Es arbeitete nur mit dem Port &h3DA (Warten auf vertical sync) und sollte nur so eingesetzt werden: WAIT &h3DA, 8. Jede andere Kombination von Port, AND_Ausdruck und XOR_Ausdruck hatte keine Auswirkung. Seit FreeBASIC v0.14 sollte die vertikale Bildschirmsynchronisation mit der Anweisung SCREENSYNC durchgeführt werden!

Siehe auch:
INP, OUT, SCREENSYNC,Hardware-Zugriffe

Letzte Bearbeitung des Eintrags am 26.12.12 um 01:39:49
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

WBIN

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » W » **WBIN**
Syntax: WBIN[$] (Ausdruck [, Stellen])
Typ: Funktion
Kategorie: Unicode

WBIN gibt den binären Wert eines beliebigen Ausdrucks als WSTRING zurück. Binärzahlen haben die Basis 2; sie bestehen aus den Zeichen 0 und 1.

- 'Ausdruck' ist eine Ganzzahl (eine Zahl ohne Kommastellen), die ins Binärformat übersetzt werden soll.
- 'Stellen' ist die Anzahl der Stellen, die dafür aufgewandt werden soll. Ist 'Stellen' größer als die benötigte Stellenzahl, wird der Rückgabewert mit führenden Nullen aufgefüllt; der zurückgegebene Wert ist jedoch nie länger, als maximal für den Datentyp von 'Ausdruck' benötigt wird.
  Ist 'Stellen' kleiner als die benötigte Stellenzahl, werden nur die hinteren Zeichen des Rückgabewerts ausgegeben.
  Wird 'Stellen' ausgelassen, besteht der Rückgabewert aus so vielen Zeichen, wie benötigt werden, um die Zahl korrekt darzustellen.
- Der Rückgabwert ist ein WSTRING, der den Wert von 'Ausdruck' im Binärformat enthält.

Das Dollarzeichen ($) als Suffix ist optional.

WBIN ist das Pendant zu BIN, gibt aber einen WSTRING zurück.

Beispiel:

```
PRINT WBIN(54321)   ' Ausgabe: 1101010000110001
PRINT WBIN(3, 3)    ' Ausgabe: 011
PRINT WBIN(255, 4) ' Ausgabe: 1111
```

Um eine Binärzahl in ihre dezimale Form zurückzuverwandeln, wird VALINT verwendet:

```
DIM binaer AS WSTRING * 7

binaer = "1001"
'Kennung &b zeigt an, dass der folgende String eine Binärzahl ist.
binaer = "&b" & binaer

PRINT VALINT(binaer)
SLEEP
```

gibt 9 aus.

Unterschiede zu QB: neu in FreeBASIC

Plattformbedingte Unterschiede:
Unicode wird in der DOS-Portierung von FreeBASIC nicht unterstützt.

Unterschiede zu früheren Versionen von FreeBASIC: existiert seit FreeBASIC v0.15

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht WBIN nicht zur Verfügung und kann nur über **__WBIN** aufgerufen werden.

Siehe auch:
WHEX, WOCT, BIN, VAL, WSTRING (Datentyp), BIT, BITSET, BITRESET, Datentypen umwandeln

Letzte Bearbeitung des Eintrags am 28.12.12 um 01:00:21
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

WCHR

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » W » **WCHR**
Syntax: WCHR[$](Unicode_Wert1 [, Unicode_Wert2 [, ...]])
Typ: Funktion
Kategorie: Unicode

WCHR verwandelt einen Unicode-Wert in seinen Character. Das Dollarzeichen ($) als Suffix ist optional.

WCHR wird z. B verwendet, um Zeichen darzustellen, die nicht auf der Tastatur sind. Es ist das Pendant zu CHR, gibt aber einen WSTRING zurück.

Beispiel:

```
PRINT "127 ist Unicode-Zeichen: "; WCHR(127)
PRINT "Codes 65, 66 und 67:     "; WCHR(65, 66, 67)
```

Ausgabe:

```
127 ist Unicode-Zeichen: Y
Codes 65, 66 und 67:     ABC
```

Unterschiede zu QB: neu in FreeBASIC

Plattformbedingte Unterschiede:
Unicode wird in der DOS-Portierung von FreeBASIC nicht unterstützt.

Unterschiede zu früheren Versionen von FreeBASIC: existiert seit FreeBASIC v0.15

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht WCHR nicht zur Verfügung und kann nur über **__WCHR** aufgerufen werden.

Siehe auch:
CHR, ASC, WSTRING (Funktion), Datentypen umwandeln

Weitere Informationen:
ASCII-Tabelle anzeigen lassen

Letzte Bearbeitung des Eintrags am 28.12.12 um 01:00:36
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

WEEKDAY

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » W » **WEEKDAY**
Syntax: WEEKDAY (Serial[, System])
Typ: Funktion
Kategorie: Datum und Zeit

Anmerkung:
Um diese Funktion nutzen zu können, muss die Datei datetime.bi in Ihren Quellcode eingebunden werden, z. B. mit INCLUDE. Alternativ können Sie auch die Datei vbcompat.bi einbinden, da dadurch auch automatisch datetime.bi eingebunden wird.

WEEKDAY extrahiert den Wochentag einer Serial Number.

- 'Serial' ist ein DOUBLE-Wert, der als Serial Number behandelt wird.
- Der Rückgabewert ist eine Zahl zwischen 1 und 7, die den Wochentag symbolisiert. 1 steht dabei für Sonntag, 7 für Samstag.
- 'System' wird verwendet, um festzulegen, welcher Tag der Woche der erste sein soll; dies beeinflusst das Ergebnis von WEEKDAY. Wird dieser Parameter ausgelassen, benutzt FreeBASIC die oben genannte Nummerierung.

Um bequem festzulegen, welcher Tag der Woche der erste sein soll, können die in der datetime.bi definierten Konstanten verwendet werden:

```
"hlzahl">1
"hlzahl">2
"hlzahl">3
"hlzahl">4
"hlzahl">5
"hlzahl">6
"hlzahl">7
```

Beispiel:
Die Nummer des aktuellen Wochentags aus NOW extrahieren:

```
"hlstring">"vbcompat.bi"
DIM Tag AS INTEGER
Tag = WEEKDAY(NOW)
```

Unterschiede zu QB: existiert nur in QBX PDS und in VBDOS

Unterschiede zu früheren Versionen von FreeBASIC: existiert erst seit FreeBASIC v0.15

Siehe auch:
NOW, DATESERIAL, DATEVALUE, YEAR, MONTH, DAY, HOUR, MINUTE, SECOND, WEEKDAYNAME, DATEDIFF, DATEPART, DATEADD, FORMAT, ISDATE, Serial Numbers, Datum und Zeit

Letzte Bearbeitung des Eintrags am 27.12.12 um 01:00:30
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

WEEKDAYNAME

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » W » **WEEKDAYNAME**
Syntax: WEEKDAYNAME (Wochentag [, [Kurzform] [, System]])
Typ: Funktion
Kategorie: Datum und Zeit

Anmerkung:
Um diese Funktion nutzen zu können, muss die Datei datetime.bi in Ihren Quellcode eingebunden werden, z. B. mit INCLUDE. Alternativ können Sie auch die Datei vbcompat.bi einbinden, da dadurch auch automatisch datetime.bi eingebunden wird.

WEEKDAYNAME gibt den Namen eines Wochentags aus.

- 'Wochentag' ist die Nummer des Wochentags, also 1 für Sonntag, 2 für Montag und so weiter.
- Der Rückgabewert ist ein String, der den Namen des Wochentags enthält.
- 'Kurzform' ist ein Wert, der entweder gleich oder ungleich null ist. Wenn er ungleich null ist, wird ein abgekürzter Wochentagsname ausgegeben, der (abhängig von der Systemeinstellung) aus zwei oder drei Zeichen besteht. Ansonsten wird der volle Name ausgegeben.
- 'System' wird verwendet, um festzulegen, welcher Tag der Woche der erste sein soll; dies beeinflusst das Ergebnis von WEEKDAYNAME. Wird dieser Parameter ausgelassen, benutzt FreeBASIC die oben genannte Nummerierung.

Um bequem festzulegen, welcher Tag der Woche der erste sein soll, können die in der datetime.bi definierten Konstanten verwendet werden:

```
"hlzahl">1
"hlzahl">2
"hlzahl">3
"hlzahl">4
"hlzahl">5
"hlzahl">6
"hlzahl">7
```

Beispiele:
Den Namen des aktuellen Wochentags mit NOW ermitteln:

```
"hlstring">"vbcompat.bi"
DIM Wochentag AS STRING
Wochentag = WEEKDAYNAME(WEEKDAY(NOW))
```

Alle Wochentage ausgeben, beginnend mit Montag:

```
"hlstring">"vbcompat.bi"
DIM i AS INTEGER
FOR i = 1 TO 7
   PRINT WEEKDAYNAME(i, , fbMonday)
NEXT
```

Unterschiede zu QB: existiert nur in QBX PDS und in VBDOS.

Unterschiede zu früheren Versionen von FreeBASIC: existiert seit FreeBASIC v0.15

Siehe auch:

NOW, DATESERIAL, DATEVALUE, YEAR, MONTH, DAY, WEEKDAY, MONTHNAME, FORMAT, ISDATE, Serial Numbers, Datum und Zeit

Letzte Bearbeitung des Eintrags am 28.12.12 um 01:01:46
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

WEND

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » W » **WEND**
Siehe WHILE ... WEND

Letzte Bearbeitung des Eintrags am 28.08.11 um 18:08:26
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

WHEX

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » W » **WHEX**
Syntax: WHEX[$] (Ausdruck [, Stellen])
Typ: Funktion
Kategorie: Unicode

WHEX gibt den hexadezimalen Wert eines beliebigen numerischen Ausdrucks als WSTRING zurück. Hexadezimale Zahlen enthalten Ziffern aus dem Bereich 0-F (0123456789ABCDEF).

- 'Ausdruck' ist eine Ganzzahl (eine Zahl ohne Kommastellen), die ins Hexadezimalformat übersetzt werden soll
- 'Stellen' ist die Anzahl der Stellen, die dafür aufgewandt werden soll. Ist 'Stellen' größer als die benötigte Stellenzahl, wird der Rückgabewert mit führenden Nullen aufgefüllt; der zurückgegebene Wert ist jedoch nie länger, als maximal für den Datentyp von 'Ausdruck' benötigt wird.
  Ist 'Stellen' kleiner als die benötigte Stellenzahl, werden nur die hinteren Zeichen des Rückgabewerts ausgegeben.
  Wird 'Stellen' ausgelassen, besteht der Rückgabewert aus so vielen Zeichen, wie benötigt werden, um die Zahl korrekt darzustellen.
- Der Rückgabwert ist ein String, der den Wert von 'Ausdruck' im Hexadezimalformat enthält.

Das Dollarzeichen ($) als Suffix ist optional.

WHEX ist das Pendant zu HEX, gibt aber einen WSTRING zurück.

Beispiel:

```
PRINT HEX(54321)     ' Ausgabe: D431
PRINT HEX(255, 4)    ' Ausgabe: 00FF
PRINT HEX(70000, 3) ' Ausgabe: 170
```

Um einen hexadezimalen Wert in einen dezimalen zurückzuverwandeln, benutzen Sie VALINT. Damit VALINT den nachfolgenden WSTRING als Hexadezimalwert behandelt, muss ihm ein "&h" vorausgehen:

```
PRINT VALINT("&hB0")
```

gibt 192 aus.

Unterschiede zu QB: neu in FreeBASIC

Plattformbedingte Unterschiede:
Unicode wird in der DOS-Portierung von FreeBASIC nicht unterstützt.

Unterschiede zu früheren Versionen von FreeBASIC: existiert seit FreeBASIC v0.15

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht WHEX nicht zur Verfügung und kann nur über **__WHEX** aufgerufen werden.

Siehe auch:
WBIN, WOCT, HEX, VAL, WSTRING (Datentyp), Datentypen umwandeln

Letzte Bearbeitung des Eintrags am 28.12.12 um 01:02:37
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

WHILE

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » W » **WHILE**
Syntax A:

```
DO WHILE Bedingung
  ' Anweisungen
LOOP
```

Syntax B:

```
DO
  ' Anweisungen
LOOP WHILE Bedingung
```

Typ: Schlüsselwort
Kategorie: Programmablauf

WHILE wird mit DO ... LOOP verwendet.

Beispiel:

```
Dim As Integer a = 1

Do While a < 10
  Print "Hallo"
  a = a + 1
Loop

Sleep
```

Die Schleife wird so lange ausgeführt, wie a kleiner ist als 10. Wenn a den Wert 10 erreicht oder überschreitet, dann wird die Schleife verlassen. Der Code bewirkt genau dasselbe, wenn stattdessen WHILE ... WEND eingesetzt wird:

```
Dim As Integer a = 1

While a < 10
  Print "Hallo"
  a = a + 1
Wend

Sleep
```

Siehe auch:
DO ... LOOP, WHILE ... WEND, UNTIL, Schleifen

Letzte Bearbeitung des Eintrags am 28.12.12 um 01:02:58
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

WHILE ... WEND

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » W » **WHILE ... WEND**
Syntax:

```
WHILE Bedingung
  ' [...Programmcode...]
WEND
```

Typ: Anweisung
Kategorie: Programmablauf

Die WHILE-Schleife wiederholt einen Anweisungsblock, solange eine Bedingung erfüllt ist. 'Bedingung' ist ein numerischer Ausdruck, der entweder wahr/erfüllt (ungleich null) oder falsch/nicht erfüllt (gleich null) sein kann.

Der Code innerhalb der Schleife wird nur dann ausgeführt, wenn schon vor Beginn der Schleife die Bedingung erfüllt wurde; andernfalls wird der WHILE...WEND-Block übersprungen. Die Schleife arbeitet damit exakt genauso wie

```
DO WHILE Bedingung
  ' &"hlkw0">LOOP
```

In oben stehendem Code dient WHILE als Schlüsselwort für die DO-Schleifen, um den Typ der Bedingung festzulegen. Bringen Sie die zusammengehörigen Schlüsselworter DO...LOOP bzw. WHILE...WEND nicht durcheinander!

Wie in allen Schleifen können auch die Kontrollanweisungen CONTINUE und EXIT verwendet werden. CONTINUE WHILE bewirkt, dass der Rest des Schleifen-Codes nicht weiter ausgeführt wird; FreeBASIC springt bei der Ausführung einer solchen Zeile zurück zur WHILE-Anweisung, prüft, ob die Bedingung noch immer erfüllt ist, und führt, wenn dies der Fall ist, die Schleife weiter normal aus.
Mit EXIT WHILE kann die Schleife 'außerplanmäßig' verlassen werden; FreeBASIC fährt dann mit der Codeausführung nach der WEND-Zeile fort.

WHILE...WEND-Blöcke können beliebig oft ineinander und mit anderen Schleifen verschachtelt werden.

Als Block-Anweisung initialisiert WHILE...WEND einen SCOPE-Block, der mit der WEND-Zeile endet. Variablen, die innerhalb einer solchen Schleife deklariert werden, existieren außerhalb nicht mehr.
Achtung: Eine innerhalb der WHILE-Schleife deklarierte Variable kann auch nicht in 'Bedingung' zur Überprüfung des Schleifenendes verwendet werden, da sie an dieser Stelle bereits nicht mehr existiert!

Beispiel:
In diesem Beispiel wird eine WHILE-Schleife eingesetzt, um einen String umzukehren. Benutzt wird dazu die String-Indizierung. Die Schleife stoppt, wenn der Index kleiner als 0 ist, da 0 der Index des ersten Zeichens eines Strings ist.

```
Dim As String satz, ztas
Dim As Integer index

satz = "The quick brown fox jumps over the lazy dog."
index = Len(satz) - 1

While index >= 0
  ztas &= Chr(satz&"hlzeichen">])
```

```
  index -= 1
Wend

Print "original:  """ ; satz ; """"
Print "umgedreht: """ ; ztas ; """"
SLEEP
```

Unterschiede zu QB:
In QB dürfen innerhalb einer Schleife keine Variablen definiert werden.

Unterschiede zu früheren Versionen von FreeBASIC:
Seit FreeBASIC v0.16 wirkt eine WHILE-Schleife wie ein SCOPE-Block.

Unterschiede unter den FB-Dialektformen:
In den Dialektformen -lang qb und -lang fblite erzeugt WHILE...WEND keinen eigenen Scope-Block, sondern arbeitet wie in QB.

Siehe auch:
DO ... LOOP, FOR ... NEXT, EXIT, CONTINUE, SCOPE, Schleifen, Bedingungsstrukturen

Letzte Bearbeitung des Eintrags am 28.12.12 um 01:02:21
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

WIDTH (Anweisung)

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » W » **WIDTH (Anweisung)**
Syntax A: WIDTH [Spalten], [Zeilen]
Syntax B: WIDTH LPRINT Spalten
Syntax C: WIDTH { #Kanal | Gerätename }, Spalten
Typ: Anweisung
Kategorie: Konsole

WIDTH legt die Anzahl der Zeilen sowie der Zeichen pro Zeile für die Textausgabe in Konsolen- und Grafikfenstern (Syntax A), bei der Druckerausgabe (Syntax B) bzw. bei der Ausgabe in eine Datei (Syntax C) fest.

- 'Spalten' ist in einem Konsole-Fenster die Anzahl der Spalten, die für das Fenster bereitgestellt werden sollen, also die Breite des Fensters in Zeichen.
- 'Zeilen' ist in einem Konsole-Fenster die Anzahl der Zeilen, die für das Fenster bereitgestellt werden sollen, also die Höhe des Fensters in Zeichen.
- Für Windows-Konsolen-Fenster darf hier jeder Wert angegeben werden, der größer ist als 0. Für Vollbild-Modi sind die Dimensionen von der Hardware abhängig. Unter Linux ist diese Syntax unzulässig; die Größe des Konsolenfensters kann nicht auf diesem Weg festgelegt werden.
- In einem Grafikfenster (siehe SCREENRES) wird die Anzahl der Spalten und Zeilen und damit indirekt die Höhe des Standard-Schriftsatzes festgelegt. Es werden die Schriftgrößen 8x8 (Standard), 8x14 und 8x16 unterstützt. Bei der Verwendung des Befehls SCREEN sind möglicherweise nicht alle diese Schriftgrößen verfügbar; siehe auch den dazu gehörenden Referenzartikel.
- 'LPRINT' (Syntax B) gibt an, dass die Breite der Druckerausgabe in Zeichen festgelegt werden soll. Wird eine Zeile ausgegeben, die diese Breite überschreitet, wird automatisch ein Zeilenumbruch eingefügt; dabei kann auch das Wort unterbrochen werden. Siehe auch LPRINT (Anweisung). Die Standard-Ausgabebreite des Druckers ist 80.
- '#Kanal' bzw. 'Gerätename' (Syntax C) gibt an, dass die Breite der Dateiausgabe in Zeichen festgelegt werden soll. Wird eine Zeile ausgegeben, die diese Breite überschreitet, wird automatisch ein Zeilenumbruch eingefügt; dabei kann auch das Wort unterbrochen werden. Siehe auch OPEN (Anweisung).
- Wird ein Argument ausgelassen, behält FreeBASIC die vorige Einstellung bei.

Anmerkung: Die Syntax-Varianten B und C konnten noch nicht erfolgreich getestet werden.

Beim Aufruf von WIDTH in Grafikmodi wird ein CLS-Aufruf erzwungen, der Bildschirm wird also gelöscht. Wenn die Kombination Spalten x Zeilen unzulässig ist (z.B. da der Bildschirm nicht so viele Zeichen anzeigen kann), wird keine Änderung vorgenommen.

Beispiele: WIDTH innerhalb der Konsole
Diese Beispiele funktionieren nur unter Windows und DOS.

```
WIDTH 40, 25
PRINT "Hallo Welt"
SLEEP
```

Wenn das Konsole-Fenster bereits beschrieben ist, bevor seine Größe mit
WIDTH geändert wird, bleibt dieser Text erhalten:

```
WIDTH 40, 25
PRINT "ein kleines Fenster"
SLEEP
WIDTH 80, 60
```

```
PRINT "ein grosses Fenster"
SLEEP
```

Im Vollbildmodus unterstützt FreeBASIC nur die Auflösungen im Bereich von 80x25 bis 80x50. Sie können durch das Drücken von &"hlkw0">WIDTH 91, 25
DIM Zeile AS STRING
FOR i as integer = 32 TO 122
 ' Zeile aus 90 Zeichen erzeugen, die alle ASCII-Zeichen
 ' vom '!' bis zum 'z' enthält.
 Zeile &= CHR(i)
NEXT
PRINT Zeile
PRINT "Im Fenstermodus kann diese ganze Zeile angezeigt werden."
PRINT "Wechseln Sie nun bitte per &"
PRINT "Druecken Sie dann bitte eine beliebige Taste, um fortzusetzen."
SLEEP
PRINT "Im Vollbildmodus koennen nur 80x25 Zeichen angezeigt werden."
PRINT "Daher werden von der ersten Zeile nur noch die ersten 80 Zeichen"
PRINT "angezeigt."
PRINT "Wechseln Sie nun bitte wieder per &"
PRINT "Druecken Sie dann bitte eine beliebige Taste, um fortzusetzen."
SLEEP
PRINT "Die ganze Zeile wird wieder angezeigt."
PRINT "Druecken Sie bitte eine beliebige Taste, um fortzusetzen."
SLEEP
CLS
WIDTH 40, 26
PRINT "Das Fenster ist jetzt 40x26 Zeichen gross."
PRINT "Wechseln Sie nun bitte per &"
PRINT "druecken Sie dann eine beliebige Taste, um fortzusetzen."
SLEEP

LOCATE 40, 41
PRINT "Wie Sie sehen, bietet der Vollbild-Modus immer noch genug Platz,"
PRINT "um 80x25 Zeichen darzustellen."
PRINT "Allerdings wird jede Zeile schon nach 40 Zeichen umgebrochen."
PRINT "Wechseln Sie nun bitte per &"
PRINT "Druecken Sie dann eine beliebige Taste, um zu beenden."
SLEEP

Beispiel: WIDTH im Grafikmodus
Im Grafikmodus wird die Breite und Höhe des Fensters durch SCREENRES festgelegt. Abhängig von dieser Größe sind nur noch bestimmte Angaben zur Spalten- und Zeilenzahl möglich. Um etwa in einem 600x400 Pixel großen Fenster eine 8x16 Pixel große Schrift zu wählen, müssen 25 Zeilen zu je 75 Spalten eingestellt werden. Die Anzahl berechnet sich aus 600 \ 8 = 75 und 400 \ 16 = 25.
Ist die Fensterauflösung zu dem Zeitpunkt, zu dem die Einstellung der Zeichenhöhe vorgenommen werden soll, nicht bekannt, kann sie mit SCREENINFO abgefragt werden. Die Angabe von 'Spalten' und 'Zeilen' richtet sich dann nach den ermittelten Werten.

```
Dim As Integer breit, hoch
ScreenInfo breit, hoch
Width breit\8, hoch\16 ' für eine Schriftgröße von 8x16
' Für eine Schriftgröße von 8x14 muss hoch\14 gesetzt
' werden, für eine Schriftgröße von 8x8 entsprechend hoch\8
```

Unterschiede zu QB:
In QB ist die Spaltenzahl auf die Werte 40 und 80 beschränkt, während für die Zeilenzahl die Werte 25, 30, 43, 50 und 60 annehmen kann, abhängig von der verwendeten Grafikhardware und dem eingestellten Screen-Modus.

Plattformbedingte Unterschiede:

- In einem Windows-Konsolenfenster sind als Spalten- und Zeilenzahl alle Werte größer als 0 zulässig.
- In einer Vollbild-Konsole unter DOS oder Windows hängen die zulässigen Werte von der Hardware ab.
- Linux erlaubt Anwendungen nicht, die Größe des Konsolenfensters zu ändern.

Unterschiede zu früheren Versionen von FreeBASIC:
Seit FreeBASIC v0.15 kann WIDTH bei Dateien, Druckern oder Geräten eingesetzt werden.

Siehe auch:
WIDTH (Funktion), LOCATE (Anweisung), Konsole

Letzte Bearbeitung des Eintrags am 26.12.12 um 09:15:40
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

WIDTH (Funktion)

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » W » **WIDTH (Funktion)**
Syntax: WIDTH
Typ: Funktion
Kategorie: Konsole

Die Funktion WIDTH gibt Informationen über die aktuelle Größe des Konsolenfensters bzw. die eingestellte Schriftgröße im Grafikfenster zurück.

Der zurückgegebene Wert wird folgendermaßen berechnet:

```
(Spalten OR (Zeilen SHL 16))
```

Das untere Word ist also die Anzahl der Spalten, während das obere Word die Anzahl der Zeilen angibt.

Beispiel:
Hinweis: Da unter Linux die Größe des Konsolenfensters nicht vom Programm geändert werden kann, findet bei folgendem Beispiel auch keine Änderung der Ausgabe statt.

```
DIM AS INTEGER Zeilen, Spalten
Zeilen = HIWORD(WIDTH)
Spalten = LOWORD(WIDTH)

PRINT "Zeilen & Spalten:"
PRINT "Zeilen: "; Zeilen
PRINT "Spalten:  "; Spalten

SLEEP
PRINT

WIDTH 80, 30
Zeilen = HIWORD(WIDTH)
Spalten = LOWORD(WIDTH)

PRINT "Zeilen & Spalten:"
PRINT "Zeilen: "; Zeilen
PRINT "Spalten:  "; Spalten

SLEEP
```

Unterschiede zu QB:
Die Verwendung von WIDTH als Funktion ist neu in FreeBASIC.

Unterschiede zu früheren Versionen von FreeBASIC:
Bis FreeBASIC v0.14 wurde die Größe des Puffers ausgegeben, der für das Konsolenfenster reserviert wurde. Seit FreeBASIC v0.15 wird die tatsächliche Fenstergröße zurückgegeben.

Siehe auch:
WIDTH (Anweisung), LOCATE (Funktion), Konsole

Letzte Bearbeitung des Eintrags am 26.12.12 um 09:21:22
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

WINDOW

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » W » **WINDOW**
Syntax: WINDOW [[SCREEN] (x1, y1)-(x2, y2)]
Typ: Anweisung
Kategorie: Grafik

WINDOW bestimmt den neuen physischen Darstellungsbereich, also den Bereich, in dem Koordinaten gültig sind.

- 'SCREEN' ist ein optionaler Parameter, der bewirkt, dass die y-Koordinaten von oben nach unten gezählt werden.
- 'x1', 'y1', 'x2' und 'y2' sind die neuen Koordinaten der Eckpunkte des physischen Darstellungsbereichs. Alle Koordinaten können Gleitkommazahlen sein.

WINDOW wird benutzt, um ein neues Koordinatensystem zu definieren. '(x1, y1)' und '(x2, y2) sind die einander gegenüberliegenden Koordinaten der Eckpunkte des grafischen Darstellungsfensters. Alle Koordinaten, die nach einem WINDOW-Aufruf an 'drawing primitives' übergeben werden, sind abhängig von diesen Koordinaten. Wenn SCREEN ausgelassen wird, sind die neuen Koordinaten kartesisch, die y-Koordinaten steigen also von unten nach oben.

Ein Aufruf von WINDOW ohne Parameter stellt den Standard wieder her, der bei der Initialisierung des Grafikfensters (siehe SCREENRES) gegolten hat.

FreeBASIC handhabt die angegeben Koordinaten als Eckkoordinaten des aktuellen Grafikbereichs. Wenn dieser z. B. durch Clipping mittels VIEW (Grafik) eingeschränkt wurde, beziehen sich die angegebenen Eckkoordinaten auf den Clipping-Bereich. Die Koordinatentransformation kann sich also ändern, wenn z. B. der Clipping-Bereich mittels VIEW verändert wird.
Die angegebene Koordinatentransformation gilt ebenso für Zeichenoperationen auf Offscreen-Puffer.

Anmerkung: Die Verwendung von WINDOW wird sich negativ auf die Ausführungsgeschwindigkeit auswirken, da alle Koordinaten zuerst umgerechnet werden müssen, bevor eine Ausgabe erfolgen kann.

Beispiel:

```
Screenres 320, 200
View (10, 10) - (310, 150), 1, 15    ' Clipping definieren
Window (-1, -1) - (1, 1)             ' Fensterkoordinaten setzen

' x-Achse zeichnen
Line (-1,0)-(1,0), 7
Draw String (0.8, -0.1), "X"
' y-Achse zeichnen
Line (0,-1)-(0,1), 7
Draw String (0.1, 0.8), "Y"

Dim As Single x, y, s

' Schrittweite berechnen
s = 2 / PMap (1, 0)

' Funktion plotten
For x = -1 To 1 Step s
  y = x ^ 3
```

```
  PSet (x, y), 14
Next x

Window                                  ' Standard-Koordinatensystem
View Screen                             ' Clipping deaktivieren
Draw String (120, 160), "Y = X ^ 3"     ' Titel schreiben
Sleep
```

Unterschiede zu QB:

- QB behält die Koordinatentransformation auch bei einer Größenänderung der Zeichenfläche (z. B. mittels VIEW) in gleicher Form bei.
- FreeBASIC behält derzeit die Eckkoordinaten bei, wodurch sich die Koordinatentransformation ändert. Dieses Verhalten wird möglicherweise in einer zukünftigen Compilerversion geändert. Derzeit muss jedoch erneut WINDOW mit entsprechenden Parametern aufgerufen werden, um z. B. nach einem VIEW die vorherige Koordinatentransformation in gleicher Form beizubehalten. Andernfalls wird der Koordinatenbereich eventuell verschoben oder skaliert.

Siehe auch:
SCREEN (Anweisung), VIEW (Grafik), PMAP, Grafik

Letzte Bearbeitung des Eintrags am 26.12.12 um 09:49:47
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

WINDOWTITLE

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » W » **WINDOWTITLE**
Syntax: WINDOWTITLE Titel
Typ: Anweisung
Kategorie: Grafik

WINDOWTITLE ändert die Beschriftung eines Grafikfensters.

'Titel' ist ein String, der die neue Beschriftung für das Grafikfenster enthält.

Der neue Titel wird sofort gültig, wenn bereits ein Grafikfenster aktiv ist, oder sobald ein SCREEN- oder SCREENRES-Aufruf erfolgt, der ein Grafikfenster öffnet. Wenn Sie WINDOWTITLE nicht aufrufen, erhält das Grafikfenster den Dateinamen ohne Erweiterung als Beschriftung.

Beispiel:

```
WINDOWTITLE "FreeBASIC Beispielprogramm"
SCREENRES 400, 300
SLEEP
```

WINDOWTITLE hat keine Auswirkung auf Konsolenfenster. Um unter Windows den Titel des Konsolenfensters zu ändern, kann folgender Code genutzt werden:

```
"hlstring">"windows.bi"
Dim Titel As Zstring *MAX_PATH
'existiert eine Konsole?
If GetConsoleTitle (@Titel, MAX_PATH) Then
  SetConsoleTitle "Neuer Konsolentitel"
End If
Sleep
```

Unterschiede zu QB: neu in FreeBASIC

Plattformbedingte Unterschiede: wird unter DOS nicht unterstützt

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht WINDOWTITLE nicht zur Verfügung und kann nur über **__WINDOWTITLE** aufgerufen werden.

Siehe auch:
SCREENRES, SCREENCONTROL, Betriebssystem-Anweisungen

Letzte Bearbeitung des Eintrags am 26.12.12 um 09:55:16
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

WINPUT

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » W » **WINPUT**
Syntax: WINPUT (Anzahl [, Dateinummer])
Typ: Funktion
Kategorie: Unicode

WINPUT liest 'Anzahl' Zeichen von der Tastatur oder aus einer Datei. Die Funktion arbeitet wie INPUT (Funktion), gibt jedoch einen WSTRING zurück.

- 'Anzahl' ist die Zahl der Zeichen, die eingelesen werden sollen.
- 'Dateinummer' ist die Nummer einer im INPUT-Modus geöffneten Datei.
- Der Rückgabewert ist ein WSTRING, der die eingegebenen Zeichen enthält.

Wird eine Dateinummer angegeben, dann liest der Befehl 'Anzahl' Zeichen aus einer Datei und aktualisiert den Dateicursor. Ansonsten wartet er auf die Eingabe von 'Anzahl' Zeichen von der Tastatur und gibt diese als WSTRING zurück. Erweiterte Zeichen (wie z. B. die Pfeiltasten) werden nicht eingelesen. Die Zeichen werden nicht auf dem Bildschirm ausgegeben.

Hinweis: Zur Zeit unterstützt FreeBASIC keine Unicode-Eingabe über die Tastatur! Der Befehl kann nur zum Lesen aus einer Datei verwendet werden.

Beispiel:

```
Dim char As WString * 2

Dim filename As String, enc As String
Dim f As Integer

Line Input "Bitte den Dateinamen eingeben: ", filename
Line Input "Encoding eingeben (optional):  ", enc
If enc = "" Then enc = "ascii"

f = FreeFile
If Open(filename For Input Encoding enc As "hlzeichen">) = 0 Then
  Print "Druecken Sie die Leertaste, um ein Zeichen aus der Datei
auszulesen, oder ESC zum Beenden."
  Do
    Select Case Input(1)
    Case " "   ' Space
      If EOF(f) Then
        Print "Das Dateiende wurde erreicht."
        Exit Do
      End If
      char = WInput(1, f)
      Print char & " (Zeichen Nr. " & Asc(char) & ")"
    Case Chr(27)   ' Escape
      Exit Do
    End Select
  Loop
  Close "hlkw0">Else
  Print "Die Datei konnte nicht geöffnet werden."
End If
Sleep
```

Achtung: ENCODING kann zusammen mit UTF-codierten Dateien nur dann erfolgreich eingesetzt werden, wenn das Byte Order Mark (BOM) gesetzt ist und mit der angegebenen Codierung übereinstimmt. Ansonsten wird der Laufzeitfehler 2 (File not found) zurückgegeben.

Unterschiede zu QB: neu in FreeBASIC

Plattformbedingte Unterschiede:
Unicode wird in der DOS-Portierung von FreeBASIC nicht unterstützt.

Unterschiede zu früheren Versionen von FreeBASIC: existiert seit FreeBASIC v0.15

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht WINPUT nicht zur Verfügung und kann nur über **__WINPUT** aufgerufen werden.

Siehe auch:
INPUT (Funktion), WSTRING (Datentyp), OPEN, Dateien (Files), Benutzereingaben

Letzte Bearbeitung des Eintrags am 28.12.12 um 01:05:11
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

WITH

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » W » **WITH**
Syntax:

```
WITH UDT_Name
  &"hlzeichen">]
  [Ausdruck_mit_Record2]
  [...]
END WITH
```

Typ: Anweisung
Kategorie: Klassen

WITH erlaubt es, auf die Records und Methoden eines UDTs (TYPE oder UNION) zuzugreifen, ohne den Namen des UDTs mit angeben zu müssen. Den Elementen muss stattdessen nur ein Punkt vorangestellt werden.

'UDT_Name' ist der Name des UDTs, auf das zugegriffen werden soll. Es kann sich auch um einen dereferenzierten Pointer handeln.

Beispiel 1:

```
TYPE AdrType
  Empf AS STRING
  City AS STRING
  PLZ  AS INTEGER
END TYPE
DIM Adressen(10) AS AdrType

WITH Adressen(0)
  .PLZ  = 98765
  .Empf = "Karl Mustermann"
END WITH
```

Innerhalb eines WITH-Blocks können beliebige Anweisungen stehen. Der Block dient lediglich dazu, die Schreibarbeit beim Zugriff auf die UDT-Member zu vereinfachen. Auch der Zugriff auf andere UDTs als den im WITH-Block festgelegten ist möglich.

Beispiel 2:

```
TYPE MyType
  MyStr AS STRING
  MyInt AS INTEGER
  MyPtr AS INTEGER PTR
END TYPE

TYPE OtherType
  OthStr AS STRING
  OthArr(20) AS INTEGER
END TYPE

DIM Anything(10) AS MyType
DIM AnythingElse AS OtherType
```

```
WITH Anything(5)
  FOR i = 0 TO 10
    .MyStr = STR(AnythingElse.OthArr(i))
    PRINT .MyStr
    PRINT @.MyPtr
  NEXT
END WITH
```

WITH-Blöcke dürfen auch verschachtelt sein. Elemente, die in der abgekürzten Form verwendet werden, beziehen sich immer auf den UDT, der im innersten WITH-Block angegeben wird.

Beispiel 3:

```
TYPE MyType
  MyStr AS STRING
  MyInt AS INTEGER
  MyPtr AS INTEGER PTR
END TYPE

TYPE OtherType
  OthStr AS STRING
  OthArr(20) AS INTEGER
  OthUdt As MyType
END TYPE

DIM Anything(10) AS MyType
DIM AnythingElse AS OtherType

WITH AnythingElse
  .OthStr = "hello"
  .OthArr(5) = 7
  WITH .OthUdt
    .MyStr = "world"
    .MyInt = 8
    .MyPtr = @.MyInt
  END WITH
END WITH
```

Beispiel 4: WITH-Block mit einem dereferenzierten Pointer als Argument

```
TYPE SubType
  d  AS INTEGER
  e  AS INTEGER
  f  AS INTEGER
End Type

TYPE MainType
  a  AS INTEGER
  b  AS INTEGER
  c  AS SubType
End Type
```

```
DIM x AS MainType PTR
x = CALLOCATE(LEN(MainType))

WITH x->c
  .d = 5
  .e = 6
  .f = 7

  PRINT .d
  PRINT .e
  PRINT .f
END WITH

SLEEP
```

Hinweise:

- WITH funktioniert (noch) nicht mit NAMESPACE.
- Aufgrund der inneren Struktur sollte nicht mit GOTO in einen WITH-Block gesprungen werden.

Unterschiede zu QB: neu in FreeBASIC

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht WITH nicht zur Verfügung und kann nur über **__WITH** aufgerufen werden.

Siehe auch:
TYPE, UNION, DIM, Objektorientierung

Letzte Bearbeitung des Eintrags am 26.12.12 um 21:09:52
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

WOCT

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » W » **WOCT**
Syntax: WOCT[$] (Ausdruck [, Stellen])
Typ: Funktion
Kategorie: Unicode

WOCT gibt den oktalen Wert eines beliebigen Ausdrucks als WSTRING zurück. Oktalzahlen haben die Basis 8; die Ziffern reichen von 0 bis 7.

- 'Ausdruck' ist eine Ganzzahl (eine Zahl ohne Kommastellen), die ins Oktalformat übersetzt werden soll.
- 'Stellen' ist die Anzahl der Stellen, die dafür aufgewandt werden soll. Ist 'Stellen' größer als die benötigte Stellenzahl, wird der Rückgabewert mit führenden Nullen aufgefüllt; der zurückgegebene Wert ist jedoch nie länger, als maximal für den Datentyp von 'Ausdruck' benötigt wird.
  Ist 'Stellen' kleiner als die benötigte Stellenzahl, werden nur die hinteren Zeichen des Rückgabewerts ausgegeben.
  Wird 'Stellen' ausgelassen, besteht der Rückgabewert aus so vielen Zeichen, wie benötigt werden, um die Zahl korrekt darzustellen.
- Der Rückgabwert ist ein WSTRING, der den Wert von 'Ausdruck' im Oktalformat enthält.

Das Dollarzeichen ($) als Suffix ist optional.

WOCT ist das Pendant zu OCT, gibt aber einen WSTRING zurück.

Beispiel:

```
PRINT WOCT(8)       ' Ausgabe: 10
PRINT WOCT(20, 4)   ' Ausgabe: 0024
PRINT WOCT(100, 2) ' Ausgabe: 44
```

Um eine Oktalzahl in ihre dezimale Form zurückzuverwandeln, wird VALINT verwendet:

```
DIM oktal AS WSTRING * 6

oktal = "100"

' Kennung &o zeigt an, dass der folgende String eine Oktalzahl ist.
oktal = "&o" & oktal

PRINT VALINT(oktal)
SLEEP
```

gibt 64 aus.

Unterschiede zu QB: neu in FreeBASIC

Plattformbedingte Unterschiede:
Unicode wird in der DOS-Portierung von FreeBASIC nicht unterstützt.

Unterschiede zu früheren Versionen von FreeBASIC: existiert seit FreeBASIC v0.15

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht WOCT nicht zur Verfügung und kann nur über **__WOCT** aufgerufen

werden.

Siehe auch:
WHEX, WBIN, OCT, VAL, WSTRING (Datentyp), Datentypen umwandeln

Letzte Bearbeitung des Eintrags am 28.12.12 um 01:05:44
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

WRITE (Anweisung)

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » W » **WRITE (Anweisung)**
Syntax: WRITE [Printausdruck]
Typ: Anweisung
Kategorie: Konsole

WRITE erfüllt dieselbe Funktion wie PRINT (Anweisung), formatiert den Text jedoch anders. STRINGs, egal ob als Variable oder Konstante, werden in "Anführungszeichen" ausgegeben. Zahlen zwischen 0 und 1 erhalten eine führende 0 vor dem Dezimalpunkt. Kommata werden als Literale ausgegeben und führen nicht mehr zu Tabspaces. Die Verwendung von Semikola ist mit WRITE unzulässig.
Wird 'Printausdruck' ausgelassen, gibt WRITE einen Zeilenumbruch aus.

Beispiel:

```
DIM AS INTEGER a = 10, b = 3
WRITE "String A", "String B", .1, 2
WRITE
WRITE a * b
WRITE a & b
WRITE "a = " & a
```

Ausgabe:

```
"String A","String B",0.1,2

30
"103"
"a = 10"
```

Siehe auch:
WRITE (Datei), PRINT (Anweisung), LOCATE (Anweisung), Konsole

Letzte Bearbeitung des Eintrags am 30.05.12 um 21:36:21
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

WRITE (Datei)

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » W » **WRITE (Datei)**
Syntax: WRITE #f, Printausdruck
Typ: Anweisung
Kategorie: Dateien

WRITE # funktioniert wie WRITE (Anweisung), leitet aber die Ausgabe in eine zuvor mit OPEN geöffnete Datei um; die Datei muss in einem sequentiellen Ausgabemodus (OUTPUT, APPEND) oder im Modus BINARY geöffnet sein. Der Vorteil gegenüber PRINT # ist, dass durch die spezielle Formatierung mehrere Datensätze pro Zeile geschrieben werden können; einen Datensatz kann man per INPUT # wieder auslesen. Der einsetzbare Zeichensatz ist bei WRITE # jedoch eingeschränkt, zumindest, wenn die Datei im Anschluss wieder mit INPUT # gelesen werden soll; "Anführungszeichen" dürfen nicht verwendet werden.

Beispiel 1:

```
DIM Tmp AS STRING, nr AS INTEGER
nr = FREEFILE

OPEN "Adressen.txt" FOR APPEND AS "hlkw0">WRITE "hlzeichen">, "Karl
Mustermann", "12345", "Oberhausen", "Rottstraße 2"
WRITE "hlzeichen">, "Eva Mustermann", "12345", "Oberhausen", "Rottstraße
2"
WRITE "hlzeichen">, "Andrea Müller", "12346", "Unterhausen", "Wendeweg
17"
CLOSE "hlkw0">OPEN "Adressen.txt" FOR INPUT AS "hlkw0">DO
  FOR Satz As Integer = 1 TO 4
    INPUT "hlzeichen">, tmp
    PRINT tmp; " ";
  NEXT
  PRINT
LOOP UNTIL EOF(nr)
SLEEP
```

So wie sich die Datensätze in einer Zeile schreiben lassen, können sie auch mit einer Zeile wieder eingelesen werden. Dazu dient die Anweisung LINE INPUT "hlkw0">Dim AS STRING tmp, dateiname="Adressen.txt"
Dim As Integer nr = FREEFILE
OPEN dateiname FOR Output AS "hlkw0">WRITE "hlzeichen">, "Karl Mustermann", "12345",
"Oberhausen", "Rottstraße 2"
WRITE "hlzeichen">, "Eva Mustermann", "12345", "Oberhausen", "Rottstraße 2"
WRITE "hlzeichen">, "Andrea Müller", "12346", "Unterhausen", "Wendeweg 17"
CLOSE "hlkw0">OPEN dateiname FOR INPUT AS "hlkw0">DO
 Line Input "hlzeichen">, tmp
 PRINT tmp
LOOP UNTIL EOF(nr)
Sleep

Bei dieser Variante müssen die einzelnen Strings bei Bedarf anhand der Trennzeichen wieder aufgeteilt werden.

Siehe auch:
OPEN (Anweisung), PRINT "reflinkicon" href="temp0207.html">INPUT #, LINE INPUT #, ACCESS, Dateien (Files)

Letzte Bearbeitung des Eintrags am 28.12.12 um 01:06:03
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

WSPACE

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » W » **WSPACE**
Syntax: WSPACE[$](n)
Typ: Funktion
Kategorie: Unicode

WSPACE gibt einen WSTRING zurück, der aus 'n' Leerzeichen besteht. WSPACE ist das Pendant zu SPACE, gibt aber einen WSTRING zurück.

Das Dollarzeichen ($) als Suffix ist optional.

Beispiel:

```
DIM w AS WSTRING * 7
w = WSPACE(6)
PRINT WSTR("nicht eingerückter Text")
PRINT w; WSTR("eingerückter Text")
SLEEP
```

Achtung: Unicode-Zeichen im Quelltext können vom Compiler nur korrekt umgesetzt werden, wenn das Byte Order Mark (BOM) gesetzt ist.

Unterschiede zu QB: neu in FreeBASIC

Plattformbedingte Unterschiede:
Unicode wird in der DOS-Portierung von FreeBASIC nicht unterstützt.

Unterschiede zu früheren Versionen von FreeBASIC: existiert seit FreeBASIC v0.15

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht WSPACE nicht zur Verfügung und kann nur über **__WSPACE** aufgerufen werden.

Siehe auch:
SPACE, STRING (Funktion), TAB, SPC, WSTRING (Datentyp), String-Funktionen

Letzte Bearbeitung des Eintrags am 28.12.12 um 01:06:21
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

WSTR

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » W » **WSTR**
Syntax A: WSTR[$](numerischer Ausdruck)
Syntax B: WSTR[$](ASCII-String)
Typ: Funktion
Kategorie: Unicode

WSTR verwandelt einen übergebenen numerischen Ausdruck in einen WSTRING. Es ist die Gegenfunktion zu VAL.

WSTR kann auch verwendet werden, um einen ASCII-String in einen WSTRING zu verwandeln. So verwendet ist es die Gegenfunktion von STR.

Das Dollarzeichen ($) als Suffix ist optional.

Beispiel:

```
DIM a AS INTEGER
DIM b AS WSTRING * 5
a = 8421
b = WSTR(a)
PRINT a, b, WSTR(&haB1)
PRINT WSTR("Bitte eine Taste drücken")
SLEEP
```

Ausgabe:

```
8421          8421           2737
Bitte eine Taste drücken
```

Achtung: Unicode-Zeichen im Quelltext können vom Compiler nur korrekt umgesetzt werden, wenn das Byte Order Mark (BOM) gesetzt ist.

Unterschiede zu QB: neu in FreeBASIC

Plattformbedingte Unterschiede:
Unicode wird in der DOS-Portierung von FreeBASIC nicht unterstützt.

Unterschiede zu früheren Versionen von FreeBASIC: existiert seit FreeBASIC v0.15

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht WCHR nicht zur Verfügung und kann nur über **__WCHR** aufgerufen werden.

Siehe auch:
VAL, STR, WSTRING (Datentyp), Datentypen umwandeln

Letzte Bearbeitung des Eintrags am 28.12.12 um 01:06:38
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

WSTRING (Datentyp)

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » W » **WSTRING (Datentyp)**
Typ: Datentyp

Ein STRING ist eine Zeichenkette, die einen zusammenhängenden Text, einzelne Wörter, Buchstaben oder sonstige Zeichen enthalten kann; auch Leerstrings (also Strings ohne jeglichen Inhalt) sind möglich.

Ein WSTRING im Speziellen ist ein Array (siehe DIM) fester Größe aus wide-chars (je nach Betriebssystem 16bit- bzw. 32bit-Zeichen), das mit einem Nullzeichen (WCHR(0)) endet; er ist also wie der ZSTRING ein null-terminierter String. Deshalb darf WCHR(0) niemals innerhalb eines WSTRINGs verwendet werden; andernfalls wird die Zeichenkette unterbrochen.

Im Allgemeinen verhält sich der WSTRING wie ein ZSTRING. Der einzige Unterschied besteht darin, dass pro Zeichen zwei (unter Windows) bzw. vier (unter Linux) Bytes reserviert werden.

Dieser Typ wurde bereitgestellt, um nicht-latein-basierende Alphabete zu unterstützen. Alle STRING-Operatoren und STRING-Funktionen (wie LEFT) funktionieren bereits mit WSTRINGs. In einigen Fällen wurden spezielle WSTRING-Routinen wie WCHR geschrieben.

Achtung: Unicode-Zeichen im Quelltext können vom Compiler nur korrekt umgesetzt werden, wenn das Byte Order Mark (BOM) gesetzt ist.

Der FreeBASIC-Compiler selbst unterstützt neben ASCII-Dateien mit UNICODE-Escape-Sequenzen (\u; siehe dazu erklärend auch OPTION ESCAPE) auch in UTF-8, UTF-16LE, UTF-16BE, UTF-32LE und in UTF-32BE codierte Quellcodes.

Beispiel:

```
' Bereite einen normalen STRING, einen
' WSTRING fester Länge (statisches Array)
' und einen WSTRING PTR (dynamisches Array) vor
DIM AS STRING nst
DIM AS WSTRING * 11 wst     ' später sollen 10 Zeichen gespeichert
                            ' werden. Zusätzlich muss Platz für ein
                            ' Null-Zeichen bereitgehalten werden.
DIM AS WSTRING PTR wstp

' befülle jeden Stringtyp mit der Zeichenkette "hello"
nst  = "hello"
wst  = "hello"
wstp = CALLOCATE(12)   ' Da dieser ein Pointer ist, muss zuerst der
                       ' Speicherbereich für fünf Zeichen + ein
*wstp = "hello"        ' Nullzeichen reserviert werden. Für jedes
                       ' Zeichen, auch für das Nullzeichen, werden
                       ' je zwei Bytes reserviert.

' die drei Stringtypen ausgeben
PRINT "Inhalte:"
PRINT "nst  :", nst
PRINT "wst  :", wst
PRINT "wstp :", wstp
PRINT "*wstp:", *wstp
PRINT
```

```
' die Länge der Stringtypen ausgeben
PRINT "LEN(x)"
PRINT "nst  :", LEN(nst  )
PRINT "wst  :", LEN(wst  )
PRINT "wstp :", LEN(wstp )
PRINT "*wstp:", LEN(*wstp)
PRINT

' die Größe der Speicherstruktur ausgeben
PRINT "SIZEOF(x)"
PRINT "nst  :", SIZEOF(nst  )
PRINT "wst  :", SIZEOF(wst  )
PRINT "wstp :", SIZEOF(wstp )
PRINT "*wstp:", SIZEOF(*wstp)
PRINT

' die Zeichenkette " Welt" an die Stringtypen anhängen
nst &= " Welt"    ' FreeBASIC verwaltet den Speicher für
                  ' normale Strings automatisch.
wst &= " Welt"    ' Da dieser WTRING fixed length ist, muss
                  ' kein zusätzlicher Speicher reserviert
                  ' werden. Da nur 12 * 2 Bytes reserviert
                  ' wurden, ist 'wst' jetzt voll, eine
                  ' längere Zeichenkette kann nicht ge-
                  ' speichert werden.
wstp = REALLOCATE(wstp, 24)   ' Zusätzlichen Speicher re-
                              ' servieren, um die Zeichen-
                              ' kette anhängen zu können,
                              ' und dabei das Nullzeichen
                              ' beachten.
*wstp &= " Welt"  ' Jetzt kann auch hier ganz normal ver-
                  ' fahren werden.

' ein einzelnes Zeichen austauschen:
nst &"hlzahl">1] = 97     ' Benutze String-Indizierung: tausche
                 ' das zweite Zeichen durch
wst [1] = 97     ' CHR(97) aus - ein kleines a.
wstp[1] = 97     ' Funktioniert problemlos bei allen
                 ' Typen.

' die drei Stringtypen ausgeben
PRINT "Inhalte:"
PRINT "nst  :", nst
PRINT "wst  :", wst
PRINT "wstp :", wstp
PRINT "*wstp:", *wstp
PRINT

' nicht vergessen: den Speicher wieder freigeben:
DEALLOCATE wstp

SLEEP
```

Ausgabe:

```
Inhalte:
nst  :        hello
wst  :        hello
wstp :        3291344
*wstp:        hello

LEN(x)
nst  :         5
wst  :         5
wstp :         4
*wstp:         5

SIZEOF(x)
nst  :         12
wst  :         22
wstp :         4
*wstp:         0

Inhalte:
nst  :        hallo Welt
wst  :        hallo Welt
wstp :        3291440
*wstp:        hallo Welt
```

Unterschiede zu QB: neu in FreeBASIC

Plattformbedingte Unterschiede:
Die Unterstützung von WSTRINGs hängt von der C runtime library ab, die unter den verschiedenen Plattformen variieren kann.

- Unicode wird in der DOS-Portierung von FreeBASIC nicht unterstützt. WSTRINGs verhalten sich wie ASCII-ZSTRINGs
- Unter Windows werden WSTRINGs in UCS-2 (UTF-16 LE) codiert. Ein Zeichen belegt 2 Bytes.
- Unter Linux werden WSTRINGs in UCS-4 codiert. Ein Zeichen belegt 4 Bytes.

Unterschiede zu früheren Versionen von FreeBASIC: existiert seit FreeBASIC v0.15

Siehe auch:
WSTRING (Funktion), WCHR, WSTR, WBIN, WHEX, WOCT, WSPACE, WINPUT, STRING (Datentyp), ZSTRING, Datentypen, Datentypen und Deklarationen, String-Funktionen

Letzte Bearbeitung des Eintrags am 30.08.14 um 18:53:53
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

WSTRING (Funktion)

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » W » **WSTRING (Funktion)**
Syntax A: WSTRING[$](n, code)
Syntax B: WSTRING[$](n, char)
Typ: Funktion
Kategorie: Unicode

Die Funktion gibt einen WSTRING aus 'n' Zeichen aus; jedes Zeichen dieser Kette ist 'char' bzw. WCHR('code').

- 'n' gibt die Länge des WSTRING an, der zurückgegeben wird.
- 'code' ist der Unicode-Wert des Zeichens, aus dem die Rückgabe zusammengesetzt sein soll.
- 'char' ist ein WSTRING mit dem Zeichen, aus dem die Rückgabe zusammengesetzt sein soll. Wenn 'char' aus mehreren Zeichen besteht, wird eine Kette ausgegeben, die nur aus dem ersten Zeichen von 'char' besteht.

Die Funktion WSTRING ist das Pendant zur Funktion STRING (Funktion), gibt aber einen WSTRING zurück. Das Dollarzeichen ($) als Suffix ist optional.

Beispiel: Text unterstreichen:

```
DIM msg AS STRING
DIM l AS INTEGER
msg = "FreeBASIC ist ein BASIC-Compiler"
l = LEN(msg)
PRINT msg
PRINT WSTRING(l, WChr(42))
```

Ausgabe:

```
FreeBASIC ist ein BASIC-Compiler
********************************
```

Unterschiede zu QB: neu in FreeBASIC

Plattformbedingte Unterschiede: Unicode wird unter DOS nicht unterstützt

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht WSTRING nicht zur Verfügung und kann nur über **__WSTRING** aufgerufen werden.

Siehe auch:
WSTRING (Datentyp), STRING (Funktion), WSPACE, String-Funktionen

Letzte Bearbeitung des Eintrags am 28.12.12 um 01:07:02
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

XOR (Methode)

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » X » **XOR (Methode)**
Syntax: { PUT | DRAW STRING } [Puffer,] [STEP] (x, y), [weitere Angaben ...], XOR
Typ: Schlüsselwort
Kategorie: Grafik

XOR ist ein Schlüsselwort, das im Zusammenhang mit PUT und DRAW STRING eingesetzt wird.

Die Farbe des gezeichneten Pixels ist das Ergebnis eines logischen XOR des zu zeichnenden Pixels mit dem zu überschreibenden Pixel.

Beispiel:
An der Position (100, 100) befindet sich ein Pixel mit dem Farbattribut 172. Dieses soll nach der XOR-Methode mit einem Pixel des Farbattributs 47 überzeichnet werden. Das Ergebnis ist ein Pixel des Farbattributs 131. Dies ergibt sich folgendermaßen:

```
Dezimal     Binär
 172        10101100
  47        00101111
-XOR---------XOR---
 131        10000011
```

Die XOR-Methode kann mit allen Farbtiefen angewandt werden, also sowohl in palettenindizierten Modi als auch in High-/Truecolor-Modi. Beachten Sie, dass in palettenindizierten Modi das sichtbare Ergebnis nicht nur von den Farbattributen, sondern auch von den zugeordneten Paletten-Einträgen abhängig ist. Siehe dazu PALETTE und Standardpaletten.

Wenn Sie bei PUT kein Aktionswort angeben, wird automatisch XOR verwendet.

Beispiel:

```
' erstellt ein Grafikfenster
ScreenRes 320, 200, 16

' erstellt einen Bildbuffer mit Kreis
Dim As Integer r = 32
Dim c As Any Ptr = ImageCreate(r * 2 + 1, r * 2 + 1, 0)
Circle c, (r, r), r, RGBA(255, 255, 255, 0), , , 1, f

' legt den Bildbuffer dreimal so übereinander, dass sich die Kreise in
der Mitte überlappen
Put (146 - r, 108 - r), c, Xor
Put (174 - r, 108 - r), c, Xor
Put (160 - r,  84 - r), c, Xor

' gibt den Bildbuffer wieder frei (sollte man immer tun)
ImageDestroy c

'Wartet vor dem Beenden des Programms auf einen Tastendruck
Sleep
```

Siehe auch:
XOR (Operator), PUT (Grafik), DRAW STRING, SCREENRES, OR (Methode), AND (Methode), PSET

(Methode), PRESET (Methode), ALPHA, ADD, TRANS, CUSTOM, Grafik

Letzte Bearbeitung des Eintrags am 26.12.12 um 22:56:47
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

XOR (Operator)

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » X » **XOR (Operator)**
Syntax A: Ergebnis = Ausdruck1 XOR Ausdruck2
Syntax A: Ausdruck1 XOR= Ausdruck2
Typ: Operator
Kategorie: Operatoren

XOR kann als einfacher (Syntax A) und kombinierter (Syntax B) Operator eingesetzt werden. Syntax B ist eine Kurzform von

```
Ausdruck1 = Ausdruck1 XOR Ausdruck2
```

XOR steht für exklusives OR, bzw. "entweder das eine oder das andere". Ähnlich wie OR setzt es im Ergebnis das Bit, wenn eines der beiden Bits in Ausdruck1 und Ausdruck2 gesetzt ist; wenn jedoch beide gesetzt sind, wird das Bit im Ergebnis nicht gesetzt. Ebenso wird es nicht gesetzt, wenn beide Bits in den Ausdrücken nicht gesetzt sind.

XOR wird in Bedingungen verwendet, wenn eine Aussage erfüllt sein muss, jedoch nicht beide erfüllt sein dürfen.

XOR kann mithilfe von OPERATOR überladen werden.

Beispiel 1: XOR in einer IF-THEN-Bedingung:

```
IF (a = 1) XOR (b = 7) THEN
  PRINT "Entweder a = 1 oder b = 7, aber nicht beides zugleich."
ELSE
  PRINT "a <> 1 und b <> 7 oder a = 1 und b = 7."
END IF
```

Beispiel 2: Kontravalenz zweier Zahlen mit XOR:

```
DIM AS INTEGER z1, z2

z1 = 6
z2 = 10

PRINT z1, BIN(z1, 4)
PRINT z2, BIN(z2, 4)
PRINT "---", "----"
PRINT z1 XOR z2, BIN(z1 XOR z2, 4)
SLEEP
```

Ausgabe:

```
  6           0110
 10           1010
---           ----
 12           1100
```

Beispiel 3: XOR als kombinierter Operator

```
DIM AS INTEGER Checkboxen

' simuliere Anklicken von Checkbox Nr 5:

Checkboxen XOR= (1 SHL 5)
```

Wie Sie sehen, kann XOR benutzt werden, um ein einzelnes Bit oder mehrere Bits bequem zu invertieren. Der Ausdruck (1 SHL 5) entspricht &b100000. Ein XOR mit 0 bewirkt, dass der Status des Bits erhalten bleibt; ein XOR mit 1 löscht ein gesetztes Bit und setzt eines, das zuvor nicht gesetzt war.

Unterschiede zu QB:
Kombinierte Operatoren sind neu in FreeBASIC.

Siehe auch:
XOR (Methode), AND (Operator), NOT, OR (Operator), IMP, EQV, Bit Operatoren / Manipulationen

Letzte Bearbeitung des Eintrags am 28.12.12 um 01:07:40
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

YEAR

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » Y » **YEAR**
Syntax: YEAR (Serial)
Typ: Funktion
Kategorie: Datum und Zeit

Anmerkung:
Um diese Funktion nutzen zu können, muss die Datei datetime.bi in Ihren Quellcode eingebunden werden, z. B. mit INCLUDE. Alternativ können Sie auch die Datei vbcompat.bi einbinden, da dadurch auch automatisch datetime.bi eingebunden wird.

YEAR extrahiert das Jahr einer Serial Number.

- 'Serial' ist ein DOUBLE-Wert, der als Serial Number behandelt wird.
- Der Rückgabewert ist das Jahr, das in der Serial Number gespeichert ist.

Beispiel: Das aktuelle Jahr aus NOW extrahieren:

```
"hlstring">"vbcompat.bi"
DIM jahr AS INTEGER
jahr = YEAR(NOW)
```

Unterschiede zu QB: existiert nur in QBX PDS und in VBDOS

Unterschiede zu früheren Versionen von FreeBASIC: existiert seit FreeBASIC 0.15

Siehe auch:
NOW, DATESERIAL, DATEVALUE, MONTH, DAY, WEEKDAY, HOUR, MINUTE, SECOND, MONTHNAME, WEEKDAYNAME, DATEDIFF, DATEPART, DATEADD, FORMAT, ISDATE, Serial Numbers, Datum und Zeit

Letzte Bearbeitung des Eintrags am 26.12.12 um 23:02:00
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

ZSTRING

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » Z » **ZSTRING**
Typ: Datentyp

Ein ZSTRING ist eine Zeichenkette, die einen zusammenhängenden Text, einzelne Wörter, Buchstaben oder sonstige Zeichen enthalten kann; auch Leerstrings (also Strings ohne jeglichen Inhalt) sind möglich.

Was unterscheidet STRING und ZSTRING?
STRING- und ZSTRING-Variablen eignen sich beide zur Speicherung von Zeichenketten, also Abfolgen von Buchstaben, Zahlen, Satz- und Sonderzeichen. Arithmetische Operationen (z. B. Substraktion) sind auf sie nicht anwendbar.
Der Unterschied zwischen den beiden Datentypen liegt darin, wie die Länge der gespeicherten Zeichenkette festgelegt wird. Anders als elementare Datentypen wie INTEGER, DOUBLE etc., die jeweils eine genau bestimmte und immer gleiche Anzahl an Bytes im Speicher belegen, können (Z-)Strings praktisch nahezu beliebig lang sein.
Ein FreeBASIC-Programm könnte zum Beispiel sowohl die Zeichenkette *"Fest gemauert"* als auch das gesamte Lied von der Glocke, das mit diesen Worten beginnt, speichern. Woher weiß das Programm nun aber, wo im Speicher die Zeichenkette aufhört und etwas anderes (zum Beispiel eine andere Zeichenkette) beginnt? Hinter der vom Benutzer angegebenen String-Variable verbirgt sich nämlich intern im Hintergrund zunächst nur die Information, wo die Zeichenkette im Speicher anfängt, ein Zeiger auf ihren Beginn also.
Um nun eindeutig bestimmen zu können, wie lang die Zeichenkette ist, gibt es in FreeBASIC zwei Ansätze:

- Man speichert - für den Nutzer unsichtbar - vor dem Anfang jedes Strings eine Zahl mit fester Speicherlänge, die Aufschluss darüber gibt, wie viele Stringbytes auf sie folgen:
  - (006) WETTER
  - (013) WETTERBERICHT

  Dieses Verfahren findet bei den "normalen" STRING-Variablen in FreeBASIC Anwendung.
- Alternativ ist es möglich, am Ende der Zeichenkette eine Stopp-Markierung anzuhängen, statt die Länge als Zahl zu speichern. Das Programm liest also, von der Anfangsadresse des Strings an, so lange weiter, bis diese Stopp-Markierung gefunden wurde:
  - WETTERBERICHT<STOP>

  Dieses Prinzip liegt den ZSTRINGs zugrunde. Die Stopp-Markierung besteht bei ihnen aus dem ASCII-Zeichen mit dem Code 0 ("**z**ero"), was die Bezeichnung **Z**STRING erklärt. ZSTRINGs werden daher auch als **null-terminierte** oder **zero-terminated** Strings bezeichnet.

Der Nachteil der Längenkennzeichnungsmethode der ZSTRINGs besteht darin, dass CHR(0), also die Stopp-Markierung, **niemals** in den Zeichenkettendaten auftauchen darf. Taucht die Stopp-Markierung CHR(0) dennoch in den Daten auf (z.B. HALLO <CHR(0)> WELT), so wird die Zeichenkette genau an dieser Stelle abgeschnitten. Im soeben eingeführten Beispiel bliebe also nur "HALLO" übrig. Daten, die Nullbytes enthalten, können mit ZSTRINGs daher nicht gespeichert werden.

Wird einer ZSTRING-Variablen ein Wert zugewiesen, wird automatisch das abschließende Nullzeichen angehängt.
Dies muss beachtet werden, wenn Speicherplatz für einen ZSTRING reserviert wird, da auch das abschließende Nullzeichen ein Byte belegt.

Ein ZSTRING kann auch als ein UBYTE-Array (siehe DIM) betrachtet werden. Dabei ist es möglich, dieses Array statisch zu behandeln; der ZSTRING ist dann **fixed length** (siehe zu fixed length auch STRING (Datentyp)). Alternativ kann er auch wie ein dynamisches Array gehandhabt werden; die Speicherverwaltung liegt dann jedoch ganz in den Händen des Benutzers. Im Gegensatz zu 'klassischen' Arrays geschieht eine Redimensionierung nämlich nicht über REDIM, sondern über REALLOCATE.

Werden fixed-length-ZSTRINGs verwendet, so verhindert FreeBASIC automatisch einen Overflow, indem es

Zeichenketten automatisch abschneidet, wenn sie länger sind als der zur Verfügung stehende Speicherbereich. Bei einem ZSTRING POINTER muss allerdings selbstständig darauf geachtet werden.

ZSTRINGs werden sehr häufig bei der Übergabe von Strings an oder von externen SUBs/FUNCTIONs benutzt. Dieser Typ wurde bereitgestellt, um einen einfachen Umgang mit C-Bibliotheken zu gewährleisten und um die fixed-length-Strings zu ersetzen, die nicht als Pointer verwendet werden können.

Beispiel:

```
' Bereite einen normalen STRING, einen ZSTRING fester Länge (statisches
Array)
' und einen ZSTRING PTR (dynamisches Array) vor
DIM AS STRING nstr
DIM AS ZSTRING * 11 zstr
' später sollen 10 Zeichen gespeichert werden. Zusätzlich
' muss Platz für ein null-zeichen bereitgehalten werden.

DIM AS ZSTRING PTR zstrp

' Befülle jeden Stringtyp mit der Zeichenkette "hello".
nstr  = "hello"
zstr  = "hello"
zstrp = CALLOCATE(6)    ' Da dieser ein Pointer ist, muss
                        ' zuerst der Speicherbereich
*zstrp = "hello"        ' für fünf Zeichen + ein Nullzeichen
                        ' reserviert werden.

' Die drei Stringtypen ausgeben
PRINT "Inhalte:"
PRINT "nstr  :", nstr
PRINT "zstr  :", zstr
PRINT "zstrp :", zstrp
PRINT "*zstrp:", *zstrp
PRINT

' Die Länge der Stringtypen ausgeben
PRINT "LEN(x)"
PRINT "nstr  :", LEN(nstr  )
PRINT "zstr  :", LEN(zstr  )
PRINT "zstrp :", LEN(zstrp )
PRINT "*zstrp:", LEN(*zstrp)
PRINT

' Die Größe der Speicherstruktur ausgeben
PRINT "SIZEOF(x)"
PRINT "nstr  :", SIZEOF(nstr  )
PRINT "zstr  :", SIZEOF(zstr  )
PRINT "zstrp :", SIZEOF(zstrp )
PRINT "*zstrp:", SIZEOF(*zstrp)
PRINT

' die Zeichenkette " Welt" an die Stringtypen anhängen
nstr &= " Welt"   ' FreeBASIC verwaltet den Speicher für
                  ' normale Strings automatisch
```

```
zstr &= " Welt"    ' da dieser ZString fixed length ist,
                   ' muss kein zusätzlicher Speicher re-
                   ' serviert werden. Da nur 11 Bytes re-
                   ' serviert wurden, ist 'zstr' jetzt
                   ' voll, eine längere Zeichenkette kann
                   ' nicht gespeichert werden.
zstrp = REALLOCATE(zstrp, 12) ' zusätzlichen Speicher re-
                              ' servieren, um die Zeichen-
                              ' kette anhängen zu können,
                              ' und dabei das Nullzeichen
                              ' beachten.
*zstrp &= " Welt" ' jetzt kann auch hier ganz normal ver-
                  ' fahren werden.

' ein einzelnes Zeichen austauschen:
nstr &"hlzahl">1] = 97     ' benutze String-Indizierung: Tausche
                  ' das zweite Zeichen durch
zstr [1] = 97     ' CHR(97) aus - ein kleines a.
zstrp[1] = 97     ' funktioniert problemlos bei allen
                  ' Typen.

' Die drei Stringtypen ausgeben
PRINT "Inhalte:"
PRINT "nstr  :", nstr
PRINT "zstr  :", zstr
PRINT "zstrp :", zstrp
PRINT "*zstrp:", *zstrp
PRINT

' nicht vergessen: Den Speicher wieder freigeben:
DEALLOCATE zstrp

SLEEP
```

Ausgabe:

```
Inhalte:
nstr  :       hello
zstr  :       hello
zstrp :       3291152
*zstrp:       hello

LEN(x)
nstr  :        5
zstr  :        5
zstrp :        4
*zstrp:        5

SIZEOF(x)
nstr  :        12
zstr  :        11
zstrp :        4
*zstrp:        0
```

```
Inhalte:
nstr  :       hallo Welt
zstr  :       hallo Welt
zstrp :       3291464
*zstrp:       hallo Welt
```

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC: existiert seit FreeBASIC 0.13

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht ZSTRING nicht zur Verfügung und kann nur über **__ZSTRING** aufgerufen werden.

Siehe auch:
SIZEOF, VARPTR, STRPTR, CAST, CHR, ASC, STR, VAL, BIN, HEX, OCT, SPACE, INPUT (Funktion), DIM, STRING (Datentyp), WSTRING (Datentyp), Datentypen, Datentypen und Deklarationen, String-Funktionen

Letzte Bearbeitung des Eintrags am 26.12.12 um 23:11:58
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

__DATE_ISO__

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » Metabefehle » **__DATE_ISO__**
Typ: Vordefiniertes Symbol

__DATE_ISO__ wird in der ausführbaren Datei zu einem String umgesetzt. Dieser enthält das Compilier-Datum im Format yyyy-mm-dd. Damit entspricht es der ISO 8601 für Datumsangaben und kann für Datumsvergleiche verwendet werden.

Beispiel:

```
Print "Compiliert am: " & __DATE_ISO__

If __DATE_ISO__ < "2011-12-25" Then
    Print "Vor Weihnachten 2011 compiliert."
Else
    Print "Nach Weihnachten 2011 compiliert."
End If
```

Ausgabe:

```
Compiliert am: 2011-09-29
Vor Weihnachten 2011 compiliert.
```

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC: existiert seit FreeBASIC v0.24

Siehe auch:
__DATE__, __TIME__, DEFINE (Meta), Präprozessoren

Letzte Bearbeitung des Eintrags am 19.01.13 um 17:29:49
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

__DATE__

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » Metabefehle » **__DATE__**
Typ: Vordefiniertes Symbol

__DATE__ wird in der ausführbaren Datei zu einem String umgesetzt. Dieser enthält das Compilier-Datum im Format mm-dd-yyyy.

Beispiel:

```
PRINT "Dieses Programm wurde erstellt am "; __DATE__; " um "; __TIME__;
"."
```

Ausgabe:

```
Dieses Programm wurde erstellt am 01-16-2007 um 15:16:22.
```

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC: existiert seit FreeBASIC v0.13

Siehe auch:
__DATE_ISO__, __TIME__, DEFINE (Meta), Präprozessoren

Letzte Bearbeitung des Eintrags am 19.01.13 um 16:01:19
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

__FB_64BIT__

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » Metabefehle » **__FB_64BIT__**
Typ: Vordefiniertes Symbol

__FB_64BIT__ ist ein Symbol ohne Wert. Dieses Symbol wird definiert, wenn der Code von der x64-Version des Compilers umgesetzt wird; durch #IFDEF ist es möglich, bestimmte Codeteile nur zu compilieren, wenn der Code für x64-Programme optimiert werden soll.

Beispiel:

```
"hlkw0">__FB_64BIT__
  "hlstring">"Dies ist ein x64-Programm"
"hlkw0">"hlstring">"Dies ist ein x86-Programm"
"reflinkicon" href="temp0108.html">DEFINE (Meta), Präprozessoren,
__FB_PCOS__, __FB_UNIX__
```

Letzte Bearbeitung des Eintrags am 02.07.13 um 00:32:33
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

__FB_ARGC__

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » Metabefehle » **__FB_ARGC__**
Typ: Vordefiniertes Symbol

__FB_ARGC__ wird in der ausführbaren Datei zu einem Integer umgesetzt. Dieser gibt die Anzahl der Argumente an, die in der Kommandozeile verwendet wurden. Gemeint ist die Kommandozeile, die an das Programm selbst, nicht an den Compiler übergeben wurde; siehe dazu COMMAND.

Beispiel:

```
Dim As Integer i
For i = 0 To __FB_ARGC__ - 1
Print "arg "; i; " = '"; Command(i); "'"
Next
```

Hinweis:
__FB_ARGC__ ist nur auf Hauptmodulebene bekannt und funktioniert nicht in Unterprogrammen.

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC: existiert seit FreeBASIC v0.17

Siehe auch:
__FB_ARGV__, COMMAND, DEFINE (Meta), Präprozessoren

Letzte Bearbeitung des Eintrags am 11.06.11 um 21:00:45
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

__FB_ARGV__

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » Metabefehle » **__FB_ARGV__**
Typ: Vordefiniertes Symbol

__FB_ARGV__ wird in der ausführbaren Datei zu einem Pointer umgesetzt. Dieser zeigt auf einen Speicherbereich, in dem sich weitere ZSTRING PTRs befinden, die auf die einzelnen Kommandozeilenparameter zeigen.
Gemeint ist die Kommandozeile, die an das Programm selbst, nicht an den Compiler übergeben wurde; siehe dazu COMMAND.

Beispiel:

```
Declare Function main ( ByVal argc As Integer, ByVal argv As ZString Ptr
Ptr ) As Integer

End main( __FB_ARGC__, __FB_ARGV__ )

Function main ( ByVal argc As Integer, ByVal argv As ZString Ptr Ptr ) As
Integer
For i As Integer = 0 To argc - 1
Print "arg "; i; " = '"; *argv&"hlstring">"'"
Next

Sleep
Return 0
End Function
```

Beispiel 2:

```
Declare Sub mySub(argc As Integer = 0, argv As ZString Ptr Ptr = 0)

Print "Hauptprogramm:"
Print "ARGC: " & __FB_ARGC__
Print "ARGV: " & *__FB_ARGV__&"hlzahl">0]
Print

mySub(__FB_ARGC__, __FB_ARGV__)    'mit Übergabeparametern aufrufen
Print

mySub                                                    'ohne
Übergabeparameter aufrufen
Print

Sleep

'"hlkw0">Sub mySub(argc As Integer = 0, argv As ZString Ptr Ptr = 0)
   If argc <> 0 And argv <> 0 Then
      Print WStr("Unterprogramm mit Übergabe:")
      Print "ARGC: " & argc
      Print "ARGV: " & *argv&"hlzahl">0]
   Else
      Print WStr("Unterprogramm ohne Übergabe:")
      'Print "ARGC: " & __FB_ARGC__          '__FB_ARGC__ und __FB_ARGV__
```

```
sind
      'Print "ARGV: " & *__FB_ARGV__&"hlkw0">End If
End Sub
```

Hinweis:
__FB_ARGV__ ist nur auf Hauptmodulebene bekannt und funktioniert nicht in Unterprogrammen.

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC: existiert seit FreeBASIC v0.17

Siehe auch:
__FB_ARGC__, COMMAND, DEFINE (Meta), Präprozessoren

Letzte Bearbeitung des Eintrags am 17.05.12 um 21:49:24
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

__FB_BACKEND__

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » Metabefehle » **__FB_BACKEND__**
Typ: Vordefiniertes Symbol

__FB_BACKEND__ gibt an, ob Maschinencode (*'gas'*) oder C-Emitter-Code (*'gcc'*) erzeugt wurde. Dies lässt sich mit dem Compilerflag *'-gen'* ausgewählt.

Beispiel:

```
If __FB_BACKEND__ = "gcc" Then
    Print "Der Code wurde mit Hilfe des C-Emitters umgesetzt."
ElseIf __FB_BACKEND__ = "gas"
    Print "Der Code wurde direkt zu Maschinencode umgesetzt."
End If
```

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC: existiert seit FreeBASIC v0.22.0

Siehe auch:
DEFINE (Meta), Präprozessoren

Letzte Bearbeitung des Eintrags am 17.05.12 um 21:51:48
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

__FB_BIGENDIAN__

Typ: Vordefiniertes Symbol

Dieses Symbol ohne Wert wird immer dann definiert, wenn für ein System compiliert werden soll, das die big-endian-Regeln anwendet. Es kann eingesetzt werden, um Teile des Programms nur für big-endian-Systeme auszuführen.

Beispiel:

```
"hlkw0">__FB_BIGENDIAN__
   '... Anweisungen nur für big-endian-Systeme
"hlkommentar">'... Anweisungen nur für little-endian-Systeme
"reflinkicon" href="temp0108.html">DEFINE (Meta), Präprozessoren
```

Letzte Bearbeitung des Eintrags am 02.07.10 um 20:32:44
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

__FB_BUILD_DATE__

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » Metabefehle » **__FB_BUILD_DATE__**
Typ: Vordefiniertes Symbol

__FB_BUILD_DATE__ gibt das Datum (mm-dd-yyyy) aus, an dem der FB-Compiler erstellt wurde.

Beispiel:

```
Print "Der FB-Compiler wurde erstellt am:" & __FB_BUILD_DATE__
```

Unterschiede zu QB: neu in FreeBASIC

Siehe auch:
__FB_SIGNATURE__, __DATE__, Präprozessoren

Letzte Bearbeitung des Eintrags am 17.05.12 um 22:13:28
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

__FB_CYGWIN__

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » Metabefehle » **__FB_CYGWIN__**
Typ: Vordefiniertes Symbol

__FB_CYGWIN__ ist ein Symbol ohne einen Wert. Dieses Symbol wird definiert, wenn der Code in der Cygwin-Umgebung umgesetzt werden soll.

Beispiel:

```
"hlkw0">__FB_CYGWIN__
... Anweisungen nur für Cygwin ...
"hlkw0">ELSE
... Anweisungen nicht für Cygwin ...
"hlkw0">ENDIF
```

Unterschiede zu QB: neu in FreeBASIC

Siehe auch:
DEFINE (Meta), Präprozessoren, __FB_DOS__, __FB_WIN32__, __FB_LINUX__, __FB_XBOX__, __FB_FREEBSD__, __FB_DARWIN__, __FB_OPENBSD__, __FB_NETBSD__, __FB_PCOS__, __FB_UNIX__

Letzte Bearbeitung des Eintrags am 17.05.12 um 22:18:12
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

__FB_DARWIN__

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » Metabefehle » **__FB_DARWIN__**
Typ: Vordefiniertes Symbol

__FB_DARWIN__ ist ein Symbol ohne einen Wert. Dieses Symbol wird definiert, wenn der Code in der Version für den Darwin-Compiler umgesetzt werden soll.

Beispiel:

```
"hlkw0">__FB_DARWIN__
... Anweisungen nur für Darwin ...
"hlkw0">ELSE
... Anweisungen nicht für Darwin ...
"hlkw0">ENDIF
```

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC: existiert seit FreeBASIC v0.21

Siehe auch:
DEFINE (Meta), Präprozessoren, __FB_DOS__, __FB_WIN32__, __FB_LINUX__, __FB_XBOX__, __FB_CYGWIN__, __FB_FREEBSD__, __FB_OPENBSD__, __FB_NETBSD__, __FB_PCOS__, __FB_UNIX__

Letzte Bearbeitung des Eintrags am 17.05.12 um 22:20:38
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

__FB_DEBUG__

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » Metabefehle » **__FB_DEBUG__**
Typ: Vordefiniertes Symbol

__FB_DEBUG__ ist ein Symbol, das beim Compilieren in eine Zahl umgesetzt wird. Wurde beim Compilieren die Kommandozeilenoption '-g' angewandt, so hat es den Wert -1, andernfalls hat es den Wert 0.

Beispiel:

```
"hlkw0">__FB_DEBUG__
"hlkw0">Else
"hlkw0">EndIf
```

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC: existiert seit FreeBASIC v0.17

Siehe auch:
__FB_ERR__, __FB_MT__, DEFINE (Meta), Präprozessoren

Letzte Bearbeitung des Eintrags am 19.01.13 um 15:51:01
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

__FB_DOS__

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » Metabefehle » **__FB_DOS__**
Typ: Vordefiniertes Symbol

__FB_DOS__ ist ein Symbol ohne Wert. Dieses Symbol wird definiert, wenn der Code von der DOS-Version des Compilers umgesetzt wird; durch #IFDEF ist es möglich, bestimmte Codeteile nur zu compilieren, wenn der Code für DOS umgesetzt werden soll.

Beispiel:

```
"hlkw0">__FB_DOS__
  "hlstring">"MyProjectDOSVersion.bi"
"hlkw0">"hlkw0">"reflinkicon" href="temp0108.html">DEFINE (Meta),
Präprozessoren, __FB_WIN32__, __FB_LINUX__, __FB_XBOX__, __FB_CYGWIN__,
__FB_FREEBSD__, __FB_DARWIN__, __FB_OPENBSD__, __FB_NETBSD__, ,
__FB_PCOS__, __FB_UNIX__
```

Letzte Bearbeitung des Eintrags am 19.01.13 um 15:53:36
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

__FB_ERR__

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » Metabefehle » **__FB_ERR__**
Typ: Vordefiniertes Symbol

__FB_ERR__ ist ein Symbol, das abhängig von den Kommandozeilenoptionen zu verschiedenen Zahlen umgesetzt wird.

Option	Wert
'-e', '-ex' oder '-exx' nicht angegeben. (Keine Fehlerprüfung)	0
'-e' - einfache Fehlerbehandlungsroutinen aktiviert.	1
'-ex' - QB-Fehlerbehandlungsroutinen aktiv	3
'-exx' - Alle Fehlerbehandlungsroutinen aktiv	7

Beispiel:

```
"hlkw0">__FB_ERR__ = 0
   "hlstring">"Fehlerüberprüfung wurde deaktiviert!"
"hlkw0">__FB_ERR__ = 1
   "hlstring">"Die Überprüfung einfacher Fehler wurde aktiviert!"
"hlkw0">__FB_ERR__ = 3
   "hlstring">"Eine QBasic-artige Fehlerüberprüfung wurde aktiviert!"
"hlkw0">__FB_ERR__ = 7
   "hlstring">"Alle Fehlerbehandlungsroutinen wurden aktiviert!"
"hlkw0">"hlstring">"Es wurde ein unbekanntes Fehler-Level gesetzt!"
"reflinkicon" href="temp0499.html">__FB_MT__, __FB_DEBUG__, DEFINE,
Präprozessoren
```

Letzte Bearbeitung des Eintrags am 02.07.10 um 20:42:07
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

__FB_FPMODE__

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » Metabefehle » **__FB_FPMODE__**
Typ: Vordefiniertes Symbol

__FB_FPMODE__ ist ein Symbol mit vordefiniertem Wert "fast". Dieses Symbol wird definiert, wenn der Compiler 'SSE floating point arithmetics' compiliert. Dazu ist die Compiler-Option -fpu SSE nötig.

Beispiel:

```
"hlkw0">__FB_FPMODE__ = "fast"
  '... Anweisungen für "fast" floating point instructions
"hlkommentar">'... Anweisungen für "normal" floating point instructions
"reflinkicon" href="temp0573.html">-lang qb steht __FB_FPMODE__ nicht zur
Verfügung.
```

Siehe auch:
Compileroption -fmode [FAST | PRECISE], __FB_SSE__, __FB_FPU__, DEFINE (Meta), Präprozessoren
Letzte Bearbeitung des Eintrags am 19.01.13 um 17:30:36
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

__FB_FPU__

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » Metabefehle » **__FB_FPU__**
Typ: Vordefiniertes Symbol

__FB_FPU__ ist ein Symbol, das als "sse" definiert ist, wenn mit 'SSE floating point arithmetics' compiliert wurde. Ansonsten hat es den Wert "x87".

Beispiel:

```
"hlkw0">__FB_FPU__ = "sse"
  ' ... Anweisungen für SSE
"hlkommentar">' ... Anweisungen für andere
"reflinkicon" href="temp0572.html">Compileroption -fpu [FPU|SSE],
__FB_SSE__, __FB_FPMODE__, DEFINE (Meta), Präprozessoren
```

Letzte Bearbeitung des Eintrags am 19.01.13 um 17:31:04
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

__FB_FREEBSD__

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » Metabefehle » **__FB_FREEBSD__**
Typ: Vordefiniertes Symbol

__FB_FREEBSD__ ist ein Symbol ohne einen Wert. Dieses Symbol wird definiert, wenn der Code für FreeBSD umgesetzt werden soll.

Beispiel:

```
"hlzeichen">__FB_FREEBSD__
... Anweisungen nur für FreeBSD ...
#ELSE
... Anweisungen nicht für FreeBSD ...
"hlkw0">ENDIF
```

Unterschiede zu QB: neu in FreeBASIC

Siehe auch:
DEFINE (Meta), Präprozessoren, __FB_DOS__, __FB_WIN32__, __FB_LINUX__, __FB_XBOX__, __FB_CYGWIN__, __FB_DARWIN__, __FB_OPENBSD__, __FB_NETBSD__, __FB_PCOS__, __FB_UNIX__

Letzte Bearbeitung des Eintrags am 04.07.10 um 00:46:44
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

__FB_GCC__

Typ: Vordefiniertes Symbol

__FB_GCC__ ist ein Symbol, dessen Wert -1 ist, wenn der Code mit *-gen gcc* compiliert wird, bzw. 0, wenn *-gen gas* verwendet wird.

Beispiel:

```
"hlkw0">"hlstring">"Dieses Programm wurde mit -gen gcc compiliert"
"hlkw0">"hlstring">"Dieses Programm wurde mit -gen gas compiliert"
"reflinkicon" href="temp0108.html">DEFINE (Meta), Präprozessoren,
__FB_64BIT__, __FB_BACKEND__
```

Letzte Bearbeitung des Eintrags am 15.01.14 um 18:18:40
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

__FB_LANG__

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » Metabefehle » **__FB_LANG__**
Typ: Vordefiniertes Symbol

__FB_LANG__ wird beim Compilieren in eine Zeichenkette umgesetzt, die angibt, nach welchen FB-Dialektregeln compiliert wird. Standardmäßig wird __FB_LANG__ auf "fb" gesetzt. Der Wert kann durch folgende Methoden geändert werden:

- die Compileroption -lang
- die Compileroption -forcelang
- die Anweisung #lang
- der Metabefehl '$lang

Der Rückgabewert ist einen String mit einem der folgenden Werte:

Wert	Beschreibung
fb	kompatibel zu FreeBASIC ab Version 0.17
qb	kompatibel zu QBASIC
fblite	kompatibel zu FreeBASIC, mit erhöhter QBASIC-Kompatibilität
deprecated	kompatibel zu FreeBASIC bis Version 0.16

Beispiel:

```
"hlkw0">__FB_LANG__
"hlkw0">__FB_LANG__ <> "fb"
Option Explicit
"hlkw0">EndIf
"hlkw0">Else
Option Explicit
"hlkw0">Endif
```

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC: existiert seit FreeBASIC v0.17

Siehe auch:
LANG (Meta), __FB_VERSION__, DEFINE (Meta), Präprozessoren

Letzte Bearbeitung des Eintrags am 19.01.13 um 15:54:15
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

__FB_LINUX__

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » Metabefehle » **__FB_LINUX__**
Typ: Vordefiniertes Symbol

__FB_LINUX__ ist ein Symbol ohne Wert. Dieses Symbol wird definiert, wenn der Code von der LINUX-Version des Compilers umgesetzt wird; durch #IFDEF ist es möglich, bestimmte Codeteile nur zu compilieren, wenn der Code für Linux umgesetzt werden soll.

Beispiel:

```
"hlkw0">__FB_LINUX__
  "hlstring">"MyProjectLinuxVersion.bi"
"hlkw0">"hlkw0">"reflinkicon" href="temp0108.html">DEFINE (Meta),
Präprozessoren, __FB_DOS__, __FB_WIN32__, __FB_XBOX__, __FB_CYGWIN__,
__FB_FREEBSD__, __FB_DARWIN__, __FB_OPENBSD__, __FB_NETBSD__,
__FB_PCOS__, __FB_UNIX__
```

Letzte Bearbeitung des Eintrags am 19.01.13 um 17:31:31
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

__FB_MAIN__

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » Metabefehle » **__FB_MAIN__**
Typ: Vordefiniertes Symbol

__FB_MAIN__ ist ein Symbol ohne Wert. Dieses Symbol wird definiert, sobald die Symbole und Makros des Hauptmoduls übersetzt werden. Es ist eine vom Compiler verwaltete Variable.

__FB_MAIN__ ist im Hauptmodul definiert und in anderen Modulen NICHT definiert.

Das Hauptmodul wird vom Compiler bestimmt. Es ist entweder die erste in der Befehlszeile aufgeführte Quelldatei oder ausdrücklich per Compileroption '-m' in der Befehlszeile angegeben.

Beispiel:

```
"hlkw0">__FB_MAIN__
  "hlkw0">"hlkw0">"hlkw0">"reflinkicon" href="temp0108.html">DEFINE
(Meta), Präprozessoren
```

Letzte Bearbeitung des Eintrags am 19.01.13 um 17:32:07
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

__FB_MIN_VERSION__

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » Metabefehle » **__FB_MIN_VERSION__**
Syntax: __FB_MIN_VERSION__(major, minor, patch)
Typ: Vordefiniertes Symbol

Dieses Makro vergleicht die Version des verwendeten Compilers mit den angegebenen Daten. Es gibt -1 aus, wenn die Version des Compilers größer oder gleich den Spezifikationen ist, bzw. 0, wenn die Version kleiner ist. Damit kann sichergestellt werden, dass der Code mit der Compilerversion kompatibel ist.

__FB_MIN_VERSION__ ist folgendermaßen definiert:

```
#DEFINE __FB_MIN_VERSION__(major, minor, patch_level) _
   ((__FB_VER_MAJOR__ > major) OR _
   ((__FB_VER_MAJOR__ = major) AND ((__FB_VER_MINOR__ > minor) OR _
   ( __FB_VER_MINOR__ = minor  AND __FB_VER_PATCH__ >= patch_level))))
```

Beispiel:

```
"hlkw0">NOT __FB_MIN_VERSION__(0, 15, 0)
  "hlkw0">"hlkw0">__FB_MIN_VERSION__(0, 17, 0)
  "hlzahl">16.
"reflinkicon" href="temp0573.html">-lang qb seit FreeBASIC v0.24
```

Siehe auch:
__FB_VERSION__, __FB_VER_MAJOR__, __FB_VER_MINOR__, __FB_VER_PATCH__, DEFINE (Meta), Präprozessoren

Letzte Bearbeitung des Eintrags am 17.05.12 um 22:44:29
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

__FB_MT__

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » Metabefehle » **__FB_MT__**
Typ: Vordefiniertes Symbol

__FB_MT__ wird beim Compilieren in eine Zahl umgesetzt, die angibt, ob der Code mit der FB-Multithread-Lib umgesetzt wird oder mit der FB-Standard-Lib. Wird mit der Multithread-Lib umgesetzt, so hat dieses Symbol den Wert -1, andernfalls wird es mit 0 umgesetzt.

Beispiel:

```
"hlkw0">__FB_MT__
   "hlkw0">"hlkw0">"hlkw2">single-threaded library verwendet
"reflinkicon" href="temp0488.html">__FB_DEBUG__, __FB_ERR__, DEFINE
(Meta), Präprozessoren
```

Letzte Bearbeitung des Eintrags am 19.01.13 um 15:50:38
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

__FB_NETBSD__

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » Metabefehle » **__FB_NETBSD__**
Typ: Vordefiniertes Symbol

__FB_NETBSD__ ist ein Symbol ohne einen Wert. Dieses Symbol wird definiert, wenn der Code in der Version für den NetBSD-Compiler umgesetzt werden soll.

Beispiel:

```
"hlkw0">__FB_NETBSD__
... Anweisungen nur für NetBSD ...
"hlkw0">ELSE
... Anweisungen nicht für NetBSD ...
"hlkw0">ENDIF
```

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC: existiert seit FreeBASIC v0.21

Siehe auch:
DEFINE (Meta), Präprozessoren, __FB_DOS__, __FB_WIN32__, __FB_LINUX__, __FB_XBOX__, __FB_CYGWIN__, __FB_FREEBSD__, __FB_DARWIN__, __FB_OPENBSD__, __FB_PCOS__, __FB_UNIX__

Letzte Bearbeitung des Eintrags am 04.07.10 um 00:45:57
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

__FB_OPENBSD__

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » Metabefehle » **__FB_OPENBSD__**
Typ: Vordefiniertes Symbol

__FB_OPENBSD__ ist ein Symbol ohne einen Wert. Dieses Symbol wird definiert, wenn der Code in der Version für den OpenBSD-Compiler umgesetzt werden soll.

Beispiel:

```
"hlkw0">__FB_OPENBSD__
... Anweisungen nur für OpenBSD ...
"hlkw0">ELSE
... Anweisungen nicht für OpenBSD ...
"hlkw0">ENDIF
```

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC: existiert seit FreeBASIC v0.21

Siehe auch:
DEFINE (Meta), Präprozessoren, __FB_DOS__, __FB_WIN32__, __FB_LINUX__, __FB_XBOX__, __FB_CYGWIN__, __FB_FREEBSD__, __FB_DARWIN__, __FB_NETBSD__, __FB_PCOS__, __FB_UNIX__

Letzte Bearbeitung des Eintrags am 04.07.10 um 00:46:10
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

__FB_OPTION_BYVAL__

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » Metabefehle » **__FB_OPTION_BYVAL__**
Typ: Vordefiniertes Symbol

__FB_OPTION_BYVAL__ ist ein Symbol, dessen Wert -1 ist, wenn die Variablenübergabe standardmäßig BYVAL geschieht, bzw. 0, wenn dies nicht der Fall ist. Der Wert hängt von der Dialektform ab, mit der das Programm compiliert wurde.

Beispiel:

```
"hlzeichen">( __FB_OPTION_BYVAL__  <> 0 )
  "hlkw0">OPTION BYVAL darf mit diesem Quelltext nicht verwendet werden!
"reflinkicon" href="temp0041.html">BYVAL (Schlüsselwort), DEFINE (Meta),
Präprozessoren
```

Letzte Bearbeitung des Eintrags am 02.07.10 um 21:10:19
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

__FB_OPTION_DYNAMIC__

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » Metabefehle » **__FB_OPTION_DYNAMIC__**
Typ: Vordefiniertes Symbol

__FB_OPTION_DYNAMIC__ ist ein Symbol, dessen Wert -1 ist, wenn OPTION DYNAMIC verwendet wird, bzw. 0, wenn dies nicht der Fall ist.

Beispiel:

```
"hlkw0">__FB_OPTION_DYNAMIC__ <> 0
  "hlkw0">OPTION DYNAMIC nicht verwenden!
"reflinkicon" href="temp0133.html">DYNAMIC (Meta), DYNAMIC
(Schlüsselwort), DEFINE (Meta), Präprozessoren
```

Letzte Bearbeitung des Eintrags am 02.07.10 um 21:14:14
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

__FB_OPTION_ESCAPE__

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » Metabefehle » **__FB_OPTION_ESCAPE__**
Typ: Vordefiniertes Symbol

__FB_OPTION_ESCAPE__ ist ein Symbol, dessen Wert -1 ist, wenn OPTION ESCAPE verwendet wird, bzw. 0, wenn dies nicht der Fall ist. Der Wert hängt von der Dialektform ab, mit der das Programm compiliert wurde.

Beispiel:

```
"hlzeichen">( __FB_OPTION_ESCAPE__  <> 0 )
  "hlkw0">OPTION ESCAPE darf mit diesem Quellcode nicht verwendet werden!
"reflinkicon" href="temp0153.html">ESCAPE, DEFINE (Meta), Präprozessoren
```

Letzte Bearbeitung des Eintrags am 02.07.10 um 21:10:06
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

__FB_OPTION_EXPLICIT__

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » Metabefehle » **__FB_OPTION_EXPLICIT__**
Typ: Vordefiniertes Symbol

__FB_OPTION_EXPLICIT__ ist ein Symbol, dessen Wert -1 ist, wenn OPTION EXPLICIT verwendet wird, bzw. 0, wenn dies nicht der Fall ist.

Beispiel:

```
"hlzeichen">( __FB_OPTION_EXPLICIT__  = 0 )
  "hlkw0">OPTION EXPLICIT verwendet werden!
"reflinkicon" href="temp0158.html">EXPLICIT, DIM, DEFINE (Meta),
Präprozessoren
```

Letzte Bearbeitung des Eintrags am 02.07.10 um 21:03:22
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

__FB_OPTION_GOSUB__

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » Metabefehle » **__FB_OPTION_GOSUB__**
Typ: Vordefiniertes Symbol

__FB_OPTION_GOSUB__ ist ein Symbol, das angibt, ob OPTION GOSUB verwendet wird oder nicht. Ist der Wert -1, dann kann GOSUB verwendet werden, und RETURN wird ausschließlich als "return-from-gosub" verwendet. Ist der Wert 0, dann kann GOSUB nicht verwendet werden, und RETURN wird ausschließlich als "return-from-procedure" verwendet.

Beispiel:

```
"hlzeichen">( __FB_OPTION_GOSUB__ <> 0 )
    ' Unterstützung von GOSUB ausschalten
    Option nogosub
"reflinkicon" href="temp0573.html">-lang qb ist der Standardwert -1, in
allen anderen Dialektformen 0.
```

Siehe auch:
GOSUB (Schlüsselwort), NOGOSUB (Schlüsselwort), GOSUB, RETURN, DEFINE (Meta), Präprozessoren
Letzte Bearbeitung des Eintrags am 02.07.10 um 21:43:19
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

__FB_OPTION_PRIVATE__

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » Metabefehle » **__FB_OPTION_PRIVATE__**
Typ: Vordefiniertes Symbol

__FB_OPTION_PRIVATE__ ist ein Symbol, dessen Wert -1 ist, wenn SUBs und FUNCTIONs standardmäßig nur innerhalb des Moduls gültig sind. Es hat den Wert 0, wenn SUBs und FUNCTIONs standardmäßig global gültig sind. Der Wert hängt von der Dialektform ab, mit der das Programm compiliert wurde, bzw. von der Verwendung von OPTION PRIVATE.

Beispiel:

```
"hlzeichen">( __FB_OPTION_PRIVATE__  <> 0 )
  "hlkw0">OPTION PRIVATE darf mit diesem Modul nicht verwendet werden!
"reflinkicon" href="temp0322.html">PRIVATE (Schlüsselwort), DEFINE
(Meta), Präprozessoren
```

Letzte Bearbeitung des Eintrags am 02.07.10 um 21:29:59
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

__FB_OUT_DLL__

Typ: Vordefiniertes Symbol

__FB_OUT_DLL__ ist ein Symbol, dessen Wert -1 ist, wenn der Code zu einer dynamischen Bibliothek (*.dll bzw. *.so) compiliert wird, bzw. 0, wenn zu einem anderen Typ compiliert wird. Nur eines der Symbole __FB_OUT_DLL__, __FB_OUT_EXE__, __FB_OUT_LIB__ und __FB_OUT_OBJ__ besitzt den Wert -1, alle anderen besitzen den Wert 0.

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC: existiert seit FreeBASIC v0.16

Siehe auch:
DEFINE (Meta), Präprozessoren

Letzte Bearbeitung des Eintrags am 13.06.13 um 20:45:12
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

__FB_OUT_EXE__

Typ: Vordefiniertes Symbol

__FB_OUT_EXE__ ist ein Symbol, dessen Wert -1 ist, wenn der Code zu einer ausführbaren Datei (*.exe) compiliert wird, bzw. 0, wenn zu einem anderen Typ compiliert wird. Nur eines der Symbole __FB_OUT_DLL__, __FB_OUT_EXE__, __FB_OUT_LIB__ und __FB_OUT_OBJ__ besitzt den Wert -1, alle anderen besitzen den Wert 0.

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC: existiert seit FreeBASIC v0.16

Siehe auch:
DEFINE (Meta), Präprozessoren, Der Compiler

Letzte Bearbeitung des Eintrags am 19.01.13 um 17:46:43
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

__FB_OUT_LIB__

Typ: Vordefiniertes Symbol

__FB_OUT_LIB__ ist ein Symbol, dessen Wert -1 ist, wenn der Code zur statischen Bibliothek (*.lib) compiliert wird, bzw. 0, wenn zu einem anderen Typ compiliert wird. Nur eines der Symbole __FB_OUT_DLL__, __FB_OUT_EXE__, __FB_OUT_LIB__ und __FB_OUT_OBJ__ besitzt den Wert -1, alle anderen besitzen den Wert 0.

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC: existiert seit FreeBASIC v0.16

Siehe auch:
DEFINE (Meta), Präprozessoren, Der Compiler

Letzte Bearbeitung des Eintrags am 19.01.13 um 17:47:26
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

__FB_OUT_OBJ__

Typ: Vordefiniertes Symbol

__FB_OUT_OBJ__ ist ein Symbol, dessen Wert -1 ist, wenn der Code zu einer Objekt-Datei (*.obj) compiliert wird, bzw. 0, wenn zu einem anderen Typ compiliert wird. Nur eines der Symbole __FB_OUT_DLL__, __FB_OUT_EXE__, __FB_OUT_LIB__ und __FB_OUT_OBJ__ besitzt den Wert -1, alle anderen besitzen den Wert 0.

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC: existiert seit FreeBASIC v0.16

Siehe auch:
DEFINE (Meta), Präprozessoren, Der Compiler

Letzte Bearbeitung des Eintrags am 19.01.13 um 17:48:21
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

__FB_PCOS__

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » Metabefehle » **__FB_PCOS__**
Typ: Vordefiniertes Symbol

__FB_PCOS__ ist ein Symbol ohne Wert. Dieses Symbol wird definiert, wenn der Code in einem Betriebssystem umgesetzt wird, dessen Dateisystem PC-artig aufgebaut ist (Laufwerksbuchstaben, Backslashes usw.). Dies ist z. B. bei DOS, Windows, OS/2 und Symbian OS der Fall. Durch #IFDEF ist es möglich, bestimmte Codeteile nur zu compilieren, wenn der Code für PC-artige Betriebssysteme umgesetzt werden soll.

Beispiel:

```
"hlkw0">__FB_PCOS__
  '... Anweisungen für PC-artige Betriebssysteme ...
"hlkommentar">'... Anweisungen für andere Betriebssysteme ...
"reflinkicon" href="temp0108.html">DEFINE (Meta), Präprozessoren,
__FB_UNIX__, __FB_DOS__, __FB_WIN32__, __FB_LINUX__, __FB_XBOX__,
__FB_CYGWIN__, __FB_FREEBSD__, __FB_DARWIN__, __FB_OPENBSD__,
__FB_NETBSD__
```

Letzte Bearbeitung des Eintrags am 19.01.13 um 15:54:28
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

__FB_SIGNATURE__

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » Metabefehle » **__FB_SIGNATURE__**
Typ: Vordefiniertes Symbol

__FB_SIGNATURE__ gibt einen String zurück, der die Signatur des Compilers enthält, z. B.:

```
FreeBASIC 0.21.1
```

Beispiel:

```
PRINT "Dieses Programm wurde compiliert mit: "; __FB_SIGNATURE__
SLEEP
END
```

Das Beispiel erzeugt eine Bildschirmausgabe wie

```
Dieses Programm wurde compiliert mit: FreeBASIC 0.24.0
```

und beendet sich anschließend auf einen Tastendruck hin.

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC: existiert seit FreeBASIC v0.16

Siehe auch:
DEFINE (Meta), __FB_VERSION__, Präprozessoren

Letzte Bearbeitung des Eintrags am 14.12.13 um 20:08:28
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

__FB_SSE__

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » Metabefehle » **__FB_SSE__**
Typ: Vordefiniertes Symbol

__FB_SSE__ ist ein Symbol ohne einen Wert. Dieses Symbol wird definiert, wenn der Compiler 'SSE floating point arithmetics' compiliert. Dazu ist die Compiler-Option -fpu SSE nötig.

Beispiel:

```
"hlkw0">__FB_SSE__
 ' ... Anweisungen aus den SSE / SSE2 floating point instructions
"hlkommentar">' ... Anweisungen für die FPU 387
"reflinkicon" href="temp0572.html">Compileroption -fpu [FPU|SSE],
__FB_FPU__, __FB_VECTORIZE__, __FB_FPMODE__, DEFINE (Meta),
Präprozessoren
```

Letzte Bearbeitung des Eintrags am 19.01.13 um 17:48:48
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

__FB_UNIX__

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » Metabefehle » **__FB_UNIX__**
Typ: Vordefiniertes Symbol

__FB_UNIX__ ist ein Symbol ohne Wert. Dieses Symbol wird definiert, wenn der Code in einem UNIX-artigen Betriebssystem compiliert wurde; durch #IFDEF ist es möglich, bestimmte Codeteile nur zu compilieren, wenn der Code für UNIX-artige Betriebssysteme umgesetzt werden soll.

Beispiel:

```
"hlkw0">__FB_UNIX__
  '... Anweisungen für UNIX-artige Betriebssysteme ...
"hlkommentar">'... Anweisungen für andere Betriebssysteme ...
"reflinkicon" href="temp0108.html">DEFINE (Meta), Präprozessoren,
__FB_PCOS__, __FB_WIN32__, __FB_DOS__, __FB_LINUX__, __FB_XBOX__,
__FB_CYGWIN__, __FB_FREEBSD__, __FB_DARWIN__, __FB_OPENBSD__,
__FB_NETBSD__
```

Letzte Bearbeitung des Eintrags am 19.01.13 um 15:54:45
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

__FB_VECTORIZE__

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » Metabefehle » **__FB_VECTORIZE__**
Typ: Vordefiniertes Symbol

__FB_VECTORIZE__ ist ein Symbol, welches das durch die Compiler-Option -vec eingestellte Level (0, 1 oder 2) enthält. Dazu ist die Compiler-Option -fpu SSE nötig.

Beispiel:

```
"hlkw0">__FB_VECTORIZE__ = 2
  ' ... Anweisungen für Vektorialisierungslevel 2.
"hlkommentar">' ...
"reflinkicon" href="temp0572.html">Compileroption -vec [0|1|2],
__FB_SSE__, __FB_FPU__, DEFINE (Meta), Präprozessoren
```

Letzte Bearbeitung des Eintrags am 17.05.12 um 23:23:36
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

__FB_VERSION__

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » Metabefehle » **__FB_VERSION__**
Typ: Vordefiniertes Symbol

Ein String, der die Version des Compilers enthält, z. B.:

```
0.17
```

Beispiel:

```
"hlkw0">__FB_VERSION__ < "0.18"
"hlzahl">18 oder höher.
"reflinkicon" href="temp0518.html">__FB_VER_MAJOR__, __FB_VER_MINOR__,
__FB_VER_PATCH__, __FB_MIN_VERSION__, __FB_SIGNATURE__, DEFINE (Meta),
Präprozessoren
```

Letzte Bearbeitung des Eintrags am 13.06.13 um 21:39:52
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

__FB_VER_MAJOR__

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » Metabefehle » **__FB_VER_MAJOR__**
Typ: Vordefiniertes Symbol

Ein Integer, der die Versionshauptnummer des Compilers enthält, z. B.:

```
0
```

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC: existiert seit FreeBASIC v0.16

Siehe auch:
__FB_VERSION__, __FB_VER_MINOR__, __FB_VER_PATCH__, __FB_MIN_VERSION__, DEFINE (Meta), Präprozessoren

Letzte Bearbeitung des Eintrags am 13.06.13 um 21:57:36
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

__FB_VER_MINOR__

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » Metabefehle » **__FB_VER_MINOR__**
Typ: Vordefiniertes Symbol

Ein Integer, der die Versionsunternummer des Compilers enthält, z. B.:

```
20
```

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC: existiert seit FreeBASIC v0.13

Siehe auch:
__FB_VERSION__, __FB_VER_MAJOR__, __FB_MIN_VERSION__, __FB_VER_PATCH__, DEFINE (Meta), Präprozessoren

Letzte Bearbeitung des Eintrags am 13.06.13 um 21:58:13
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

__FB_VER_PATCH__

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » Metabefehle » **__FB_VER_PATCH__**
Typ: Vordefiniertes Symbol

Ein Integer, der die Patchnummer des Compilers enthält, z. B.:

```
0
```

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC: existiert seit FreeBASIC v0.16

Siehe auch:
__FB_VERSION__, __FB_VER_MAJOR__, __FB_VER_MINOR__, __FB_MIN_VERSION__, DEFINE (Meta), Präprozessoren

Letzte Bearbeitung des Eintrags am 13.06.13 um 21:58:54
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

__FB_WIN32__

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » Metabefehle » **__FB_WIN32__**
Typ: Vordefiniertes Symbol

__FB_WIN32__ ist ein Symbol ohne Wert. Dieses Symbol wird definiert, wenn der Code von der Win32-Version des Compilers umgesetzt wird; durch #IFDEF ist es möglich, bestimmte Codeteile nur zu compilieren, wenn der Code für Win32 umgesetzt werden soll.

Beispiel:

```
"hlkw0">__FB_WIN32__
  "hlstring">"windows.bi"
"hlkw0">"hlkw0">"reflinkicon" href="temp0108.html">DEFINE (Meta),
Präprozessoren, __FB_DOS__, __FB_LINUX__, __FB_XBOX__, __FB_CYGWIN__,
__FB_FREEBSD__, __FB_DARWIN__, __FB_OPENBSD__, __FB_NETBSD__,
__FB_PCOS__, __FB_UNIX__
```

Letzte Bearbeitung des Eintrags am 19.01.13 um 15:53:54
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

__FB_XBOX__

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » Metabefehle » **__FB_XBOX__**
Typ: Vordefiniertes Symbol

__FB_XBOX__ ist ein Symbol ohne einen Wert. Dieses Symbol wird definiert, wenn der Code für die Xbox umgesetzt werden soll.

Beispiel:

```
"hlkw0">__FB_XBOX__
... Anweisungen nur für Xbox ...
"hlkw0">ELSE
... Anweisungen nicht für Xbox ...
"hlkw0">ENDIF
```

Unterschiede zu QB: neu in FreeBASIC

Siehe auch:
DEFINE (Meta), Präprozessoren, __FB_DOS__, __FB_WIN32__, __FB_LINUX__, __FB_CYGWIN__, __FB_FREEBSD__, __FB_DARWIN__, __FB_OPENBSD__, __FB_NETBSD__, __FB_PCOS__, __FB_UNIX__

Letzte Bearbeitung des Eintrags am 04.07.10 um 00:47:23
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

__FILE_NQ__

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » Metabefehle » **__FILE_NQ__**
Typ: Vordefiniertes Symbol

__FILE_NQ__ wird in der compilierten Datei zu einer Zeichenkette umgesetzt. Diese enthält den Dateinamen der Quellcode-Datei, die gerade umgesetzt wird. NQ steht dabei für 'non-quoted', also 'nicht in Anführungszeichen'; die Zeichenkette wird also nicht durch Anführungszeichen eingeschlossen.

Beispiel:

```
"hlzeichen">: __FILE__
"hlzeichen">: __FILE_NQ__
```

Ausgabe:

```
quoted: "test.bas"
unquoted: test.bas
```

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC: existiert seit FreeBASIC v0.17

Siehe auch:
__FILE__, __PATH__, __FUNCTION__, __FUNCTION_NQ__, __LINE__, LINE (Meta), DEFINE (Meta), Präprozessoren

Letzte Bearbeitung des Eintrags am 13.06.13 um 22:28:44
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

__FILE__

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » Metabefehle » **__FILE__**
Typ: Vordefiniertes Symbol

__FILE__ wird in der compilierten Datei zu einem String umgesetzt. Dieser enthält den Dateinamen der Quellcode-Datei, die gerade umgesetzt wird. Dies kann zur Fehlersuche eingesetzt werden.

Beispiel:

```
IF a > 0 THEN PRINT "Fehler: a = "; a;" im Modul "; __FILE__
```

Ausgabe:

```
Fehler: a = 32767 im Modul test.bas
```

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC: existiert seit FreeBASIC v0.13

Siehe auch:
__FILE_NQ__, __PATH__, __FUNCTION__, __FUNCTION_NQ__, __LINE__, LINE (Meta), DEFINE (Meta), Präprozessoren

Letzte Bearbeitung des Eintrags am 19.01.13 um 17:49:44
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

__FUNCTION_NQ__

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » Metabefehle » **__FUNCTION_NQ__**
Typ: Vordefiniertes Symbol

__FUNCTION_NQ__ wird in der compilierten Datei zu einer Zeichenkette umgesetzt. Diese enthält das Symbol der Prozedur (FUNCTION oder SUB), die gerade umgesetzt wird. NQ steht dabei für 'non-quoted', also 'nicht in Anführungszeichen'.

Beispiel:

```
Declare Sub MySub ()

Sub MySub ()
   Print "Die Addresse von " & __FUNCTION__ & " ist 0x";
   Print Hex( @__FUNCTION_NQ__, 6 )
   GetKey
End Sub

MySub
```

Ausgabebeispiel:

```
Die Addresse of MYSUB ist 0x4012D0
```

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC: existiert seit FreeBASIC v0.17

Siehe auch:
__FUNCTION__, __FILE__, __FILE_NQ__, __PATH__, __LINE__, LINE (Meta), DEFINE (Meta), Präprozessoren

Letzte Bearbeitung des Eintrags am 19.01.13 um 17:50:33
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

__FUNCTION__

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » Metabefehle » **__FUNCTION__**
Typ: Vordefniertes Symbol

__FUNCTION__ wird in der compilierten Datei zu einem String umgesetzt. Dieser enthält den Namen der Prozedur (FUNCTION oder SUB), die gerade umgesetzt wird. Dies kann zur Fehlersuche eingesetzt werden.

Wird gerade Code auf der Modulebene umgesetzt, so wird das Symbol zu "__FB_MAINPROC__" umgesetzt.

Beispiel:

```
IF a > 0 THEN PRINT "Fehler: a= "; a;" in "; __FUNCTION__
```

Ausgabe:

```
Fehler: a = 32767 in __FB_MAINPROC__
```

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC:

- __FUNCTION__ existiert seit FreeBASIC v0.13.
- Seit FreeBASIC v0.17 wird dieses Symbol zu "__FB_MAINPROC__" umgesetzt, wenn __FUNCTION__ auf Modullebene abgefragt wird; davor war der Wert "(main)"

Siehe auch:
__FUNCTION_NQ__, __FILE__, __FILE_NQ__, __PATH__, __LINE__, LINE (Meta), DEFINE (Meta), Präprozessoren

Letzte Bearbeitung des Eintrags am 19.01.13 um 17:50:57
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

__LINE__

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » Metabefehle » **__LINE__**
Typ: Vordefiniertes Symbol

__LINE__ wird in der compilierten Datei zu einem Integer umgesetzt. Dieser enthält die Zeile, die gerade umgesetzt wird. Dies kann zur Fehlersuche eingesetzt werden.

Beispiel:

```
IF a > 0 THEN PRINT "Fehler: a = " & a & " in Zeile " & __LINE__
```

Ausgabe:

```
Fehler: a = 32767 in Zeile 42
```

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC: existiert seit FreeBASIC v0.13

Siehe auch:
LINE (Meta), __FILE__, __FILE_NQ__, __PATH__, __FUNCTION__, __FUNCTION_NQ__, DEFINE (Meta), Präprozessoren

Letzte Bearbeitung des Eintrags am 19.01.13 um 17:51:24
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

__PATH__

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » Metabefehle » **__PATH__**
Typ: Vordefiniertes Symbol

__PATH__ wird in der compilierten Datei zu einem String umgesetzt. Dieser enthält den Namen des Verzeichnisses, in dem sich die Quellcode-Datei befindet, die gerade umgesetzt wird.

Beispiel: Teile dem Compiler mit, wo sich die INCLUDE-Dateien befinden:

```
"hlkw0">__PATH__
```

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC: existiert seit FreeBASIC v0.17

Siehe auch:
LIBPATH (Meta), __FILE__, __FILE_NQ__, __FUNCTION__, __FUNCTION_NQ__, __LINE__, DEFINE (Meta), Präprozessoren

Letzte Bearbeitung des Eintrags am 19.01.13 um 17:51:47
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

__TIME__

FreeBASIC-Referenz » Alphabetische Befehlsreferenz » Metabefehle » **__TIME__**
Typ: Vordefiniertes Symbol

__TIME__ wird in der compilierten Datei zu einem String umgesetzt. Dieser enthält die Compilier-Uhrzeit im Format hh:mm:ss.

Beispiel:

```
PRINT "Dieses Programm wurde erstellt am "; __DATE__; " um "; __TIME__;
"."
```

Ausgabe:

```
Dieses Programm wurde erstellt am 01-16-2007 um 15:16:22.
```

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC: existiert seit FreeBASIC v0.13

Siehe auch:
__DATE__, __DATE_ISO__, DEFINE (Meta), Präprozessoren

Letzte Bearbeitung des Eintrags am 19.01.13 um 16:01:48
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

Datentypen

FreeBASIC-Referenz » BASIC-Grundlagen » **Datentypen**
Übersicht über die in FreeBASIC verfügbaren Datentypen:

Name	Bits	Format	kleinster Wert	größter Wert	Suffix*	C
BYTE	8	vorzeichenbehaftete Ganzzahl	-128	+127		0
UBYTE	8	vorzeichenlose Ganzzahl	±0	+255		0
SHORT	16	vorzeichenbehaftete Ganzzahl	-32.768	+32.767	s	0
USHORT	16	vorzeichenlose Ganzzahl	±0	65.535		0
INTEGER	32/64***	vorzeichenbehaftete Ganzzahl	-2.147.483.648	+2.147.483.647	%	0
UINTEGER	32/64***	vorzeichenlose Ganzzahl	±0	4.294.967.295		0
LONG	32	vorzeichenbehaftete Ganzzahl	-2.147.483.648	+2.147.483.647	&, l	0
ULONG	32	vorzeichenlose Ganzzahl	±0	4.294.967.295	ul	0
LONGINT	64	vorzeichenbehaftete Ganzzahl	-9.223.372.036.854.775.808	+9.223.372.036.854.775.807	ll	0
ULONGINT	64	vorzeichenlose Ganzzahl	±0	+18.446.744.073.709.551.615	ull	0
SINGLE	32	Gleitkommazahl	+/-1.401 298 E-45****	+/-3.402 823 E+38****	! oder f	6
DOUBLE	64	Gleitkommazahl	+/-4.940 656 458 412 465 E-324****	+/-1.797 693 134 862 316 E+308****	# oder d	1
STRING	variable/feste Länge	Zeichenkette*****	0	2GB	$	
ZSTRING	feste Länge	nullterminierte Zeichenkette	0	2GB		
WSTRING	feste Länge	nullterminierte Zeichenkette	0	2GB		

Fußnoten:

- * Suffixe sind standardmäßig, also im Modus -lang fb, nicht mehr zulässig (siehe FB-Dialektformen). Eine Ausnahme sind hierbei konstante Zahlen, z. B. "DIM f AS SINGLE = 0.22332f". So kann der Datentyp einer Zahl bestimmt werden.
- ** Angegeben ist die ungefähre Anzahl der Stellen bei Gleitkommazahlen.
- *** Bei Verwendung der x86-Version des Compilers ist die Größe 32 Bit, in der x64-Version 64 Bit.
- **** Die angegebenen Werte bei den Gleitkommazahlen sind die Werte, die Null bzw. positiver und negativer Unendlichkeit am Nächsten kommen.
- ***** STRING-Variablen enthalten zwar aus Kompatibilitätsgründen zu externen Bibliotheken ein implizites Nullbyte am Ende, werden von diesem jedoch nicht terminiert, d. h. innerhalb des Strings dürfen auch Nullbytes vorkommen, ohne dass der String an der Stelle des ersten Vorkommens von CHR(0) abgeschnitten werden würde. Die Unterschiede zwischen STRING und ZSTRING werden am Anfang des Artikels zu ZSTRING erklärt.

Unterschiede zu QB:

- INTEGERs sind in FreeBASIC 32-Bit-Variablen, die von den heutigen 32-Bit-Prozessoren besser verarbeitet werden können. Im Gegensatz dazu sind INTEGER in QB 16 Bit lang. Wer explizit 16 Bit lange Ganzzahlen benötigt, muss in FreeBASIC auf SHORT- bzw. USHORT-Variablen zurückgreifen.
- LONGs (32-Bit-Ganzzahltyp in QB) werden in FreeBASIC nicht mehr benötigt, wurden aber aus Kompatibilitätsgründen beibehalten.

Siehe auch: Thematische Übersicht: Datentypen und Deklarationen

Letzte Bearbeitung des Eintrags am 30.06.13 um 14:19:26
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

Ausdrücke und Operatoren

FreeBASIC-Referenz » BASIC-Grundlagen » **Ausdrücke und Operatoren**
Ausdrücke setzen sich aus Werten und Operatoren zusammen.

Ein Wert kann eine Zahl (z. B. 4), ein STRING (z. B. "hallo") oder eine Variable (z. B. x) sein. Ein Ausdruck kann aus einem einzelnen Wert oder aus mehreren, durch Operatoren verknüpften Werten bestehen (z. B. 4*x).

Ein Operator führt mathematische oder logische Operationen an den Werten aus (z. B. 4*x; 4 und x sind Werte, * ist der Operator für die Multiplikation). Operatoren werden für die meisten mathematischen Operationen, Vergleiche (z. B. x > 4), logische Tests (z. B. IF x > 4 AND x < 10 THEN, wobei AND ein logischer Operator ist) und Stringmanipulationen verwendet. Diese werden vor allem in Bedingungsstrukturen eingesetzt.

Die Operatoren lassen sich einteilen in:

- mathematische Operatoren
- Vergleichsoperatoren
- logische Operatoren
- Stringoperatoren
- Kurzformen (kombinierte Operatoren und Variablen-Initiatoren)

Mathematische Operatoren
Zu den mathematischen Operatoren zählen:

- Negierung (-)
- Potenzierung (^)
- Division und Integerdivision (/, \)
- Multiplikation (*)
- Bitverschiebung nach links/rechts (SHL, SHR)
- Addition und Subtraktion (+, -)

Vergleichsoperatoren
Vergleichsoperatoren geben abhängig von der gegebenen Bedingung true (-1) oder false (0) aus. Sie vergleichen zwei Ausdrücke. Geprüft kann werden, ob ein Ausdruck größer, kleiner, ungleich, gleich, größer/gleich oder kleiner/gleich einem anderen ist. Zu den Vergleichsoperatoren zählen:

- > größer als
- < kleiner als
- = gleich
- >= größer/gleich
- <= kleiner/gleich
- <> ungleich (größer oder kleiner)

Beispiel:

```
DIM  money AS  INTEGER
INPUT "Wieviel Geld haben Sie? ", money
IF money < 100 THEN
  PRINT "Sie haben weniger als $100, lassen Sie mich"
  PRINT "Ihnen weitere $100 geben."
  money = money + 100
  PRINT "Jetzt haben Sie $" & money
```

```
ELSE
  PRINT "Sie haben genau oder mehr als $100. Das"
  PRINT "reicht vorerst."
END IF
SLEEP
```

Logische Operatoren

Logische Operatoren helfen, Entscheidungen unter Berücksichtigung mehrerer Faktoren zu treffen. Beispiel: Ich möchte Schuhe, die rot sind AND (und) weniger kosten als $100.
Die meisten Entscheidungen können mit AND (beide Bedingungen müssen wahr sein), OR (mindestens eine Bedingung muss erfüllt sein), NOT (die Bedingung darf nicht erfüllt sein) oder XOR (eine Bedingung muss erfüllt sein, die andere darf es nicht sein) getroffen werden. Es gibt aber auch noch weitere logische Operatoren:

- AND - Sind beide Bedingungen erfüllt?
- OR - Ist mindestens eine Bedingung erfüllt?
- NOT - Ist die Bedingung nicht erfüllt?
- XOR - Ist eine Bedingung erfüllt, aber die andere nicht?
- EQV - Sind beide Bedingunen gleich?
- IMP - Folgt die zweite Bedingung auf die ersten? a IMP b entspricht (NOT a) OR b
- ANDALSO - ähnlich wie AND, jedoch wird die zweite Bedingung nur ausgewertet, wenn nötig
- ORELSE - ähnlich wie OR, jedoch wird die zweite Bedingung nur ausgewertet, wenn nötig

Siehe dazu auch Bit-Operatoren.

Hierarchie der Operatoren

Mehrere Operatoren in einer Anweisung werden nach einer vordefinierten Reihenfolge abgearbeitet. Ein Operator mit höherer Hierarchie wird vor einem Operator abgearbeitet, der eine niedrigere Hierarchie besitzt. Haben zwei Operatoren die gleiche Hierarchie, werden sie in der Reihenfolge abgearbeitet, in der sie auftreten. Ist eine andere Reihenfolge bei der Abarbeitung gewünscht, kann diese durch eine Klammerung der Ausdrücke erreicht werden.

Die Abarbeitung von Operatoren der gleichen Hierarchie kann von *links nach rechts* oder aber von *rechts nach links* erfolgen. Z. B. ist a+b+c gleichbedeutend mit (a+b)+c, während **a gleichbedeutend mit *(*a) ist. Ein *'N/A'* gibt an, dass aufgrund der Eigenschaften des Operators keine Richtung existiert.

Die folgende Tabelle enthält alle Operatoren absteigend nach ihrer Hierarchie:

Operator	Klassifizierung	Kardinalität	Bedeutung	Assoziativität
19				
CAST	Funktion	unär	Typumwandlung	N/A
PROCPTR	Funktion	unär	Adressoperator	N/A
STRPTR	Funktion	unär	Adressoperator	N/A
VARPTR	Funktion	unär	Adressoperator	N/A
18				
&"reflinkicon" href="temp0548.html">()	Indizierung	/	Arrayindex	von links
()	Auswertungsoperator	/	Funktionsaufruf	von links
.	Zugriffsoperator	/	Strukturzugriff	von links

->	Zugriffsoperator	/	Indirektzugriff	von links	
17					
@	Zugriffsoperator	unär	Adressoperator	von rechts	
*	Zugriffsoperator	unär	Dereferenzierung	von rechts	
NEW	Datenoperator	unär	Speicher alloziieren	von rechts	
DELETE	Datenoperator	unär	Speicher dealloziieren	von rechts	
16					
^	Exponent	unär	Exponent	von links	
15					
-	Arithmetischer Operator	unär	Negativ	von rechts	
14					
*	Arithmetischer Operator	binär	Multiplizieren	von links	
/	Arithmetischer Operator	binär	Dividieren	von links	
13					
\	Arithmetischer Operator	binär	Integerdivision	von links	
12					
MOD	Arithmetischer Operator	binär	Modulo Division	von links	
11					
SHL	Bitoperator	binär	Bitverschiebung nach links	von links	
SHR	Bitoperator	binär	Bitverschiebung nach rechts	von links	
10					
+	Arithmetischer Operator	binär	Addieren	von links	
-	Arithmetischer Operator	binär	Subtrahieren	von links	
9					
&	Verknüpfungsoperator	binär	Stringverkettung	von links	
8					
=	Vergleichsoperator	binär	Gleich	von links	
<>	Vergleichsoperator	binär	Ungleich	von links	
<	Vergleichsoperator	binär	Kleiner als	von links	
<=	Vergleichsoperator	binär	Kleiner oder gleich	von links	
>=	Vergleichsoperator	binär	Größer oder gleich	von links	
>	Vergleichsoperator	binär	Größer als	von links	
7					
NOT	Bitweiser Operator	unär	Verneinung	von rechts	
6					
AND	Bitweiser Operator	binär	Und	von links	
5					

OR		Bitweiser Operator	binär	Oder	von links
	4				
EQV		Bitweiser Operator	binär	Äquivalenz	von links
IMP		Bitweiser Operator	binär	Implikat	von links
XOR		Bitweiser Operator	binär	Exklusives Oder	von links
	3				
ANDALSO		Logischer Operator	binär	Verkürztes und	von links
ORELSE		Logischer Operator	binär	Verkürztes oder	von links
	2				
=		Zuweisungsoperator	binär	Zuweisung	N/A
&=		Zuweisungsoperator	binär	Verkettung + Zuweisung	N/A
+=		Zuweisungsoperator	binär	Addition + Zuweisung	N/A
-=		Zuweisungsoperator	binär	Subtraktion + Zuweisung	N/A
*=		Zuweisungsoperator	binär	Multiplikation + Zuweisung	N/A
/=		Zuweisungsoperator	binär	Division + Zuweisung	N/A
\=		Zuweisungsoperator	binär	Integerdivision + Zuweisung	N/A
^=		Zuweisungsoperator	binär	Exponent + Zuweisung	N/A
MOD=		Zuweisungsoperator	binär	Modulo + Zuweisung	N/A
AND=		Zuweisungsoperator	binär	Und + Zuweisung	N/A
EQV=		Zuweisungsoperator	binär	Äquivalenz + Zuweisung	N/A
IMP=		Zuweisungsoperator	binär	Implikat + Zuweisung	N/A
OR=		Zuweisungsoperator	binär	Oder + Zuweisung	N/A
XOR=		Zuweisungsoperator	binär	Exklusives oder + Zuweisung	N/A
SHL=		Zuweisungsoperator	binär	Bitverschiebung nach links + Zuweisung	N/A
SHR=		Zuweisungsoperator	binär	Bitverschiebung nach rechts + Zuweisung	N/A
LET		Zuweisungsoperator	binär	Zuweisung	N/A
	1				
LET ()		Zuweisungsoperator	binär	Zuweisung	N/A

Stringoperatoren

Die folgenden Operatoren können bei STRINGs angewandt werden:

- Strings verketten (String1 & String2 oder auch String1 + String2)
- Symbole zu Strings umwandeln und verketten (Symbol1 & Symbol2)
- Prüfen, ob ein String alphabetisch vor einem anderen steht (String1 < String2)
- Prüfen, ob ein String alphabetisch hinter einem anderen steht (String1 > String2)
- Prüfen, ob zwei Strings gleich oder ungleich sind (String1 = String2, String1 <> String2)

Anmerkung: Die Beispiele in den Klammern sind unvollständig; um sie in einem Programm zu benutzen,

müssen sie in eine Bedingungsabfrage eingebaut werden wie z. B.

```
IF String1 > String2 THEN ...
```

oder als Ausdruck behandelt werden, der einer anderen Variable zugewiesen wird, z. B.

```
a = (String1 = String2)
```

Bei Größenvergleichen werden zeichenweise die ASCII-Werte verglichen. Daher ist der String "B" (ASCII-Code 66) kleiner als der String "a" (ASCII-Code 97).

Kurzformen

Bestimmte Ausdrücke müssen relativ häufig eingegeben werden. Die dabei anfallende Tipparbeit ist relativ lästig, außerdem wird vom Compiler längst nicht immer die schnellste Methode gewählt. Um dem entgegenzuwirken, haben die Entwickler von FreeBASIC zwei Techniken eingeführt: zum einen die sogenannten kombinierten Operatoren, zum anderen die Variablen-Initiatoren.

Kombinierte Operatoren

Kombinierte Operatoren sind Kurzformen für Ausdrücke der Form

```
Variable = Variable Operator Ausdruck
```

Sie können verkürzt werden auf:

```
Variable Operator= Ausdruck
```

Beispiel:

```
DIM a AS INTEGER

' Beide Zeilen bewirken dasselbe:
a = a + 1
a += 1
```

Dies funktioniert mit allen Operatoren (+, -, *, /, \, ^, AND, OR, XOR, IMP, EQV, SHL, SHR) bei jedem Datentyp. Auch Arrays und UDTs (User defined Type, siehe TYPE) werden unterstützt (siehe dazu auch OPERATOR).

Beispiel:

```
TYPE myUDT
  Int1 AS INTEGER
  Int2 AS INTEGER
END TYPE

DIM AS INTEGER a, b, c(4)
DIM d AS myUDT

a = 5
b = 7
c(0) = 2
c(4) = -5
d.Int2 = 400
```

```
a += 5
b -= a * c(4)
c(0) *= a + b
d.Int2 /= c(0)

a OR= d.Int2 + b
```

Mit STRINGs funktioniert nur die Additions-Kurzform. Diese arbeitet übrigens schneller als die Form

```
String1 = String1 & String2
```

da kein temporärer String erstellt werden muss. Der Compiler übersetzt solche Stringverkettungen der Form

```
String1 = String2 & String3 & String4 & ...
```

allerdings automatisch in dieses Format:

```
String1 = String2 : String1 &= String3 : String1 &= String4 : String1 &=
...
```

Achtung: Wenn Sie Pointer mit der Addition bzw. Subtraktion benutzen, wird der hinzugezählte/abgezogene Wert mit der Länge des Pointer-Datentyps multipliziert, bevor die Berechnung durchgeführt wird. Aus

```
DIM a AS INTEGER PTR
a += 1
```

wird also beim Compilieren zu

```
DIM a AS INTEGER PTR
a += 1 * LEN(INTEGER)
```

Dies ist in den meisten Fällen von Vorteil, da die Berechnung nicht manuell eingegeben werden muss und dem Programmierer Tipparbeit erspart bleibt. Wenn tatsächlich eine Verschiebung um ein Byte nötig ist, kann dies mit CAST geschehen:

```
DIM a AS INTEGER PTR
DIM b AS BYTE PTR

a = ALLOCATE(10)
b = CAST(BYTE PTR, a)
b += 1
a = CAST(INTEGER PTR, b)
```

Variablen-Initiatoren

Variablen-Initiatoren ermöglichen es, einer Variablen bei ihrer Dimensionierung einen Wert zuzuweisen. Dies funktioniert sowohl für einfache Variablen als auch für Arrays. Die Syntax für einfache Variablen ist:

```
DIM Variable AS Typ = einfacherAusdruck
```

bzw.

```
DIM AS Typ Variable1 = Ausdruck &"hlkw2">CONST drei = 3
DIM a AS INTEGER = 5
```

```
DIM b AS SINGLE = SIN(drei ^ 0.5) * a
```

Für Arrays gelten dieselben Regeln, jedoch eine andere Syntax:

```
DIM Array(Elemente) AS Typ => { Element1, Element2, Element3, ... }
```

(man beachte, dass {geschweifte} Klammern eingesetzt werden!)
Statt => könnte auch hier = verwendet werden.

Beispiel:

```
CONST drei = 3
DIM Array(2) AS INTEGER => {1, drei * 7}
```

Werden - wie hier - weniger Initiatoren als Feldelemente angegeben, befüllt FreeBASIC das Feld von vorne nach hinten (hier also von Index 0 nach Index 1) mit den vorgegebenen Werten, und weist den "überschüssigen" Elementen den Wert 0 zu (bzw. einen Nullstring, falls es sich um ein String-Array handelt). Werden mehr Initiatoren als Elemente angegeben, wird ein Fehler erzeugt und die Compilierung wird abgebrochen.

Bei mehrdimensionalen Arrays müssen innerhalb der Klammern noch einmal geschweifte Klammern gesetzt werden. Dabei zählt die Nummer der Klammer in der Liste wie der Index der ersten Dimension und die Nummer des Eintrags innerhalb der Klammer wie der Index der zweiten Dimension.

Beispiel:

```
DIM AS INTEGER a(1, 1) => {{1, 2}, {3, 4}}

PRINT a(0, 0) ' 1
PRINT a(0, 1) ' 2
PRINT a(1, 0) ' 3
PRINT a(1, 1) ' 4
SLEEP
```

Variablen-Initiatoren funktionieren auch mit UDT-Arrays. Siehe dazu die Datei *var_initializers.bas* im examples-Verzeichnis.

Letzte Bearbeitung des Eintrags am 21.08.13 um 01:01:03
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

Bedingungsstrukturen

FreeBASIC-Referenz » BASIC-Grundlagen » **Bedingungsstrukturen**
Bedingungen sind Ausdrücke, die entweder wahr (ungleich 0) oder falsch (gleich 0) sein können. Häufig wird zur Kennzeichnung wahrer Ausdrücke der Wert -1 oder 1 verwendet, allerdings interpretiert FreeBASIC alle Werte ungleich 0 als wahr. Für Bedingungen gelten dieselben Regeln wie für Ausdrücke und Operatoren.

- Schon eine einzelne Konstante kann eine gesamte Bedingung sein.
- Sinnvoll sind Bedingungen jedoch erst mit Variablen.
- Eine Bedingung darf nahezu unendliche Komplexität erreichen.
- Sie kann auch Funktionen und Operatoren enthalten.
- Bedingungsstrukturen werden hauptsächlich mit IF...THEN und SELECT CASE oder als Abbruchbedingung in Schleifen verwendet, können aber auch in Wertzuweisungen bei Variablen auftauchen; siehe dazu BIT und natürlich auch IIF.

Beispiele für einfache Bedingungsstrukturen:

```
' Abfrage eines Variablenwerts
IF alter < 18 THEN
  PRINT "Du bist noch nicht volljährig!"
END IF

' Schleife, die von einer Bedingung abhängt
DO UNTIL EOF(dateinummer)  ' bis Dateiende erreicht
  LINE INPUT "hlzeichen">, variable
LOOP

' Zuweisung abhängig von einer Bedingung
minimum = IIF(wert1 < wert2, wert1, wert2)
```

Mehrere Bedingungen können über Operatoren wie AND und OR verknüpft werden. Dabei ist zu beachten, dass hierbei keine Wahrheitswerte verknüpft werden, sondern eine bitweise Verknüpfung der Ausdrücke stattfindet. Ist das Ergebnis gleich 0, so wird es als falsch interpretiert, sonst als wahr. Das bedeutet z. B., dass die Werte 2 und 4 für sich genommen wahr sind, 2 AND 4 dagegen falsch.

ANDALSO und ORELSE verknüpfen nicht bitweise, sondern interpretieren die Angaben als Wahrheitswerte; 2 ANDALSO 4 ergibt damit -1 (wahr). Außerdem brechen beide Operatoren sofort ab, wenn das Ergebnis feststeht. Ist der Ausdruck links vom ANDALSO 0 (falsch), wird der rechte Ausdruck nicht mehr ausgewertet, da das Ergebnis nicht mehr wahr werden kann. Analog dazu gibt ORELSE -1 (wahr) zurück, wenn der linke Ausdruck ungleich 0 ist; der rechte Ausdruck wird nicht mehr ausgewertet. Dieses Verhalten kann hilfreich sein, wenn der rechte Ausdruck nur unter bestimmten Bedingungen ausgewertet werden darf.

Beispiele für kombinierte Bedingungen:

```
' Kombination mit AND und OR
IF (alter > 13 AND begleitung = "Eltern") OR alter >= 18 THEN
  Zutritt
END IF

' Vorab-Prüfung mit ANDALSO
IF meinPointer > 0 ANDALSO *meinPointer = 1337 THEN
  ' wäre meinPointer = 0, würde die Dereferenzieung einen Fehler erzeugen
  tueIrgendwas
END IF
```

Durch die Klammerung im ersten Beispiel kann die gewünschte Reihenfolge bei der Abarbeitung erzwungen werden. Hier hätte auch ORELSE statt OR verwendet werden können.

Siehe auch:
Ausdrücke und Operatoren, Bit-Operatoren, Programmablauf

Weitere Informationen:
Wikipedia-Artikel zur boolschen Algebra

Letzte Bearbeitung des Eintrags am 30.12.12 um 00:28:21
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

Pointer

FreeBASIC-Referenz » BASIC-Grundlagen » **Pointer**
Ein **Pointer** ist eine positive Ganzzahl. Diese Zahl ist aber nicht nur ein simpler Wert, sondern gibt eine Adresse (Nummer des Bytes) im Speicher an. Pointer heißen auf deutsch auch "Zeiger": der Pointer "zeigt" auf die Speicherstelle, welche die Pointer-Adresse besitzt. Auf diese Speicherstelle kann über besondere Syntax-Regeln zugegriffen werden. Mit Hilfe von Pointern kann man direkt auf den Speicher zugreifen und ihn so einfacher, flexibler verwalten, als es mit PEEK und POKE möglich ist. PEEK und POKE sind nur aus Kompatibilitätsgründen zu QB vorhanden und sollten nicht verwendet werden.

Im Zusammenhang mit Pointern treten folgende Sprachkonstrukte auf:

PTR oder POINTER
Die beiden Schlüsselwörter PTR und POINTER sind gleichbedeutend. Sie werden im Zusammenhang mit Datentypen verwendet und geben an, dass eine Variable ein Pointer, also ein Zeiger auf eine Variable des angegebenen Typs ist.

```
DIM a AS BYTE PTR
```

Im Programm kann auf die als Pointer deklarierte Variable ganz normal zugegriffen werden; sie verhält sich immer wie eine Variable des Typs UINTEGER.

```
DIM a AS BYTE PTR
DIM z AS BYTE
z = 120
a = VARPTR(z)
PRINT z, a
' Ausgabe: 120 4210692
' Also ist a mindestens 32bit

a = -1
PRINT a
' Ausgabe: 4294967295
' Also ist jeder Pointer ein UInteger-Typ. Ist auch logisch,
' negative Adressen gibt es nicht.

SLEEP
```

Da eine direkte Zuweisung konstanter Werte an einen Pointer, abgesehen vom Nullsetzen (myPointer = 0), unüblich und möglicherweise vom Programmierer nicht beabsichtigt ist, gibt der Compiler in solchen Fällen die Warnung aus:

```
Suspicious pointer assignment
```

Es ist zu beachten, dass hierbei die Adresse verändert wird, die der Pointer repräsentiert, nicht aber der Wert, der sich an dieser Stelle im Speicher befindet.
Es existieren auch einige Pointeroperatoren, die nur mit Pointern funktionieren, darunter auch welche, womit gezielt der Wert im Speicher geändert wird. Diese werden im folgenden aufgeführt.

@Variable
Der Operator @ gibt die Adresse einer normalen Variablen zurück, also einen Pointer, der auf die Variable zeigt. Es ist eine Kurzform für VARPTR. Dies kann auch an Nicht-Pointer-Variablen angewandt werden; zurückgegeben wird dann einfach die Adresse als UINTEGER-Wert.

Beispiel:

```
DIM myValue AS INTEGER
DIM myPointer AS INTEGER PTR

myPointer = @myValue

' Adresse von myValue und die Adresse des Pointers ausgeben
PRINT myPointer, @myPointer

SLEEP
```

***Pointer**
Der Operator Stern gibt den Wert zurück, der an der Speicherstelle steht, auf die der Pointer zeigt (Dereferenzierung). Auch komplexe Pointerausdrücke können dereferenziert werden, d. h. in der Klammer kann ein ganzer Ausdruck stehen, der dann als Pointer behandelt wird.
Vorsicht: Wenn Sie auf Speicherstellen zugreifen, die nicht von Variablen Ihres Programms verwendet werden, riskieren Sie einen Absturz Ihres Programms oder sogar des Betriebssystems!

Beispiel:

```
DIM myValue AS INTEGER
DIM myPointer AS INTEGER PTR

myValue = 5
myPointer = @myValue

PRINT *myPointer
' gibt den Wert von myValue aus, da myPointer
' auf die Adresse von myValue zeigt.

PRINT *(myPointer + 1)
' gibt einen anderen Wert aus, der i. d. R. nicht
' genau bestimmt werden kann, da auf der
' Speicherstelle, die ein Byte hinter myValue liegt,
' bereits ein anderer Wert gespeichert ist.

SLEEP
```

Pointer&"reflinkicon" href="temp0550.html">Pointerindizierung wird zum Pointer index hinzugezählt, bevor weitere Operationen mit ihm durchgeführt werden. Der Pointer wird also dereferenziert.

```
Pointer[index]
```

hat dieselbe Funktion wie

```
*(Pointer + index)
```

Diese Art der Dereferenzierung wird oft mit ALLOCATE verwendet.

Beispiel:

```
DIM a AS BYTE PTR
a = ALLOCATE(10)
a&"hlzahl">3] = 20
PRINT a&"hlzahl">3]
PRINT a&"hlzahl">4]
PRINT a

DEALLOCATE a
SLEEP
```

Wenn a&"reflinkicon" href="temp0043.html">CALLOCATE erzeugt dagegen zehn leere Speicherstellen.

UdtPtr->Record

Der Pfeiloperator dereferenziert einen Pointer, der auf einen UDT zeigt, sodass er auf den entsprechenden Record verweist. Die Pointer-Variable zeigt also auf eine Variable im Speicher vom angegebenen UDT. So kommt man leicht an die im UDT enthaltenen Daten.

Beispiel:

```
TYPE myType
  a AS INTEGER
  b AS SHORT
  c AS INTEGER PTR
END TYPE

DIM udtPtr AS myType PTR = NEW myType

PRINT udtPtr->b

DELETE udtPtr
SLEEP
```

Im Vergleich dazu die Nicht-Pointer-Version:

```
TYPE myType
  a AS INTEGER
  b AS SHORT
  c AS INTEGER PTR
END TYPE

DIM udt AS myType

PRINT udt.b

SLEEP
```

Letzte Bearbeitung des Eintrags am 30.12.12 um 17:12:27
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

Inline-Assembler

FreeBASIC-Referenz » BASIC-Grundlagen » **Inline-Assembler**
Unter Inline-Assembler versteht man die direkte Einbindung von Maschinenspracheprogrammen in den Quellcode.

Während das unter QBASIC nur umständlich über Opcodes möglich war, ist es in FreeBASIC dank des Befehls ASM kein Problem, Assembler-Codes direkt einzubinden:

```
Asm mov  eax, ebx
Asm imul eax, ecx
```

oder

```
ASM
mov eax, ebx
imul eax, ecx
End Asm
```

Auch der Zugriff auf Variablen ist "ohne Tricks" möglich; ihre Bezeichner sind auch innerhalb eines ASM-Blocks verfügbar.
Wird ein ASM-Block innerhalb einer FUNCTION verwendet, kann das Symbol FUNCTION als Pointer auf die Adresse verwendet werden, an der das Ergebnis der Funktion gespeichert werden soll.

```
Function Mal(ByVal x As Integer, ByVal y As Integer) As Integer
  Asm
    mov  eax, &"hlzeichen">] 'hole x nach eax
    imul eax, [y] 'Multipliziere mit y
    mov [Function], eax 'Ergebnis als Rückgabewert
  End Asm
End Function

Dim As Integer a = 45, b = 54
Print Mal(a, b)
```

Syntax
FreeBASIC verwendet den GCC Assembler **AS**, der im bin-Verzeichnis als AS.EXE enthalten ist.
Der Assembler verwendet die einfache Intel-Syntax wie die bekannteren x86-Assembler MASM, TASM, NASM, YASM und FASM.
Wie in BASIC-Quelltexten wird das Hochkomma als Zeichen für einen nachfolgenden Kommentar gesetzt, nicht das ";" Zeichen wie sonst in Assemblerquelltexten üblich.
Der Assembler ist in Bezug auf Labelnamen 'case sensitive', d. h. Sie müssen die Groß- und Kleinschreibung beachten.

Registerbezeichnungen

- 4 Byte Integer Register: eax, ebx, ecx, edx, ebp, esp, edi, esi
- 2 Byte Integer Register: ax, bx, cx, dx, bp, sp, di, si (Low Word der 4 Byte e__-Register)
- 1 Byte Integer Register: al, ah, bl, bh, cl, ch, dl, dh (Low(l) oder High(h) Byte der 2 Byte _x-Register)
- Floatingpoint Register: st(0), st(1), st(2), st(3), st(4), st(5), st(6), st(7)
- MMX Register (8-Byte) : mm0, mm1, mm2, mm3, mm4, mm5, mm6, mm7
- SSE Register (16-Byte) : xmm0, xmm1, xmm2, xmm3, xmm4, xmm5, xmm6, xmm7

Kritische Assembleranweisungen
Je nach Betriebssystem oder Prozessortyp sind einige Assembleranweisungen nicht anwendbar. Im günstigsten Fall werden sie ignoriert oder mit einem Systemerror angezeigt, aber auch ein Systemcrash ist möglich.
Diese kritischen Assembleranweisungen für jedes Betriebssystem oder unterschiedliche Prozessortypen kann nicht für jeden Einzelfall aufgelistet werden.
Wer in Assembler arbeiten möchte, sollte die Einschränkungen der Hardware und des Betriebssystems, für das er Assembleranweisungen schreiben möchte, kennen.

Letzte Bearbeitung des Eintrags am 30.12.12 um 02:06:05
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

Schleifen

FreeBASIC-Referenz » BASIC-Grundlagen » **Schleifen**
Schleifen sind Programmteile, die (vielfach) wiederholt werden; die Anzahl der Wiederholungen ist in der Regel an Bedingungen geknüpft (z. B. "Wiederhole, bis a den Wert 10 hat"), sogenannte Endlosschleifen sind aber ebenso möglich.

Verschiedene Schleifentypen:

Einfache Bedingungsschleife: WHILE ... WEND
Der Codeblock zwischen WHILE und WEND wird so lange wiederholt, wie die Bedingung hinter WHILE erfüllt ist. Wenn die Bedingung beim Start eines neuen Schleifendurchlaufs nicht mehr erfüllt ist, wird die Schleife verlassen.

```
WHILE bedingung
  anweisungen
WEND
```

Die Bedingung wird vor dem Schleifendurchlauf geprüft. Ist die Bedingung schon vor dem ersten Schleifendurchlauf nicht erfüllt, dann wird die Schleife gar nicht erst betreten.

Erweiterte Bedingungsschleife: DO ... LOOP
Ähnlich wie bei WHILE...WEND wird bei jedem Schleifendurchlauf überprüft, ob die Bedingung erfüllt ist und ob die Schleife ein weiteres Mal durchlaufen werden soll. Die Bedingung kann am Beginn oder am Ende der Schleife überprüft werden:

```
DO {WHILE | UNTIL} bedingung
  anweisungen
LOOP
```

oder

```
DO
  anweisungen
LOOP {WHILE | UNTIL} bedingung
```

Im ersten Fall ist es möglich, dass die Schleife gar nicht betreten wird, im zweiten Fall wird sie auf jeden Fall mindestens einmal durchlaufen.
Das Schlüsselwort WHILE (zu deutsch "während") gibt an, dass die Schleife so lange durchlaufen wird, wie die Bedingung erfüllt ist. Damit funktioniert DO WHILE bedingung ... LOOP exakt genauso wie WHILE bedingung ... WEND.
Wird stattdessen UNTIL (zu deutsch "bis") verwendet, dann wird die Schleife so lange ausgeführt, *bis* die Bedingung erfüllt ist. Da es sich bei beiden Varianten genau um die Gegenereignisse handelt, können sie problemlos gegeneinander ausgetauscht werden. Welche Variante bevorzugt wird, hängt weitgehend vom persönlichen Geschmack ab und davon, welche Variante in der aktuellen Situation besser lesbar erscheint.
Ebenso ist es möglich, überhaupt keine Bedingung zu prüfen, und somit eine Endlosschleife zu erzeugen:

```
DO
  staendig_wiederholter_Programmcode
LOOP
```

Endlosschleifen sollten unbedingt eine andere Möglichkeit zum Verlassen besitzen (s. u.:

"Kontrollanweisungen"), da der Benutzer das Programm sonst nicht auf normalem Weg beenden kann.

Zählschleife: FOR ... NEXT
FOR...NEXT wiederholt den Codeblock zwischen FOR und NEXT und erhöht einen Zähler bei jedem Durchlauf des Codeblocks. Die Schleife wird verlassen, sobald der Zähler einen Zielwert erreicht oder überschritten hat.

```
FOR zaehler &"hlkw0">AS datentyp&"hlzeichen">= startwert TO endwert
&"hlkw0">STEP schrittweite&"hlkw0">NEXT
```

Zu Beginn der Schleife wird der Zähler auf den Startwert gesetzt. Sobald der Befehl NEXT erreicht wird, wird der Zähler um die angegebene Schrittweite erhöht - wurde keine Schrittweite angegeben, dann erhöht sich der Zähler um 1. Anschließend wird der Zähler mit dem Endwert verglichen und die Schleife gegebenenfalls verlassen.

Kontrollanweisungen

Für den Fall, dass eine Schleife nicht regulär von Anfang bis Ende durchlaufen werden soll, stehen zwei Kontrollanweisungen zur Verfügung:

In der Schleife fortfahren: CONTINUE
CONTINUE {WHILE | DO | FOR} überspringt den Rest des Schleifeninhalts und springt direkt an das Ende der Schleife. Dort wird wie gewohnt die Überprüfung der Schleifenbedingung vorgenommen. In einer FOR-Schleife wird nach einem CONTINUE FOR auch der Zähler erhöht.

Die Schleife verlassen: EXIT
EXIT {WHILE | DO | FOR} verlässt die Schleife und fährt mit dem nächsten Befehl nach dem Schleifenende fort.

Gültigkeit von Variablen

Schleifen erzeugen eigene SCOPE-Blöcke, d. h. es ist möglich, innerhalb einer Schleife Variablen zu definieren, die dann nur innerhalb dieser Schleife gültig sind. Beachten Sie, dass diese Variablen auch nicht als Schleifenbedingung abgefragt werden können, da sie an dieser Stelle bereits nicht mehr definiert sind (eine Ausnahme bildet der Zähler der FOR-Schleife).

Näheres zu SCOPE-Blöcken und zu den Gültigkeitsbereichen von Variablen finden Sie im zugehörigen Referenzartikel.

Letzte Bearbeitung des Eintrags am 01.01.13 um 00:19:26
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

Bitfelder

FreeBASIC-Referenz » BASIC-Grundlagen » **Bitfelder**
Beim Programmieren stößt man gelegentlich auf Aufgaben, bei denen man nur den Status mehrerer Objekte speichern muss, also z. B. aktiv oder nicht aktiv. Für jedes einzelne Objekt eine eigene Integer-Variable zu verwenden wäre Speicherverschwendung, da nur eines der 32 Bit genutzt wird. In älteren BASIC-Dialekten wurde dieses Problem gelöst, indem man eine INTEGER-Variable benutzt und die einzelnen Bits direkt mit den logischen Operatoren manipuliert.

Beispiel: Der Status von drei Checkboxen soll in einer INTEGER-Variablen gespeichert werden:

```
CheckBox1 = 1 ' aktiv
CheckBox2 = 0 ' nicht aktiv
CheckBox3 = 1 ' aktiv

GesamtStatus = 0 ' Variable vorbereiten: Alle nicht aktiv

IF CheckBox1 THEN GesamtStatus = GesamtStatus OR 1
IF CheckBox2 THEN GesamtStatus = GesamtStatus OR 2
IF CheckBox3 THEN GesamtStatus = GesamtStatus OR 4

' ...Programmcode...

' Status abfragen:
If GesamtStatus AND 1 THEN CheckBox1 = 1
If GesamtStatus AND 2 THEN CheckBox2 = 1
If GesamtStatus AND 4 THEN CheckBox3 = 1
```

Das Problem bei dieser Methode ist, dass nicht immer klar ist, welches Bit in 'GesamtStatus' für welche Checkbox steht. Der Programmierer müsste also entweder die Belegung auswendig lernen oder ständig in seinen Notizen nachsehen, wofür welches Bit steht. Beides ist relativ mühselig.

Unter FreeBASIC wurde dieses Problem durch die sogenannten Bitfelder (die Programmierern in C/C++ bereits bekannt sein sollten) gelöst. Ein Bitfeld ist einem UDT ähnlich, unterscheidet sich von einem normalen UDT jedoch dadurch, dass jedem Element der Struktur nur Werte innerhalb eines bestimmten Bereiches zugewiesen werden können. Die Syntax für ein Bitfeld ist die folgende:

```
TYPE BitFeldName
   Bit1Name : Bitanzahl AS INTEGER
   Bit2Name : Bitanzahl AS INTEGER
   Bit3Name : Bitanzahl AS INTEGER
   ' ...
END TYPE
DIM Variable AS BitFeldName
```

In Bitfelder aufteilen lassen sich nur die Datentypen (U)BYTE, (U)SHORT, (U)INTEGER bzw. (U)LONG. Alle anderen Datentypen erzeugen eine Fehlermeldung des Compilers. Die Namen der Strukturelemente können dabei frei gewählt werden. Es dürfen beliebig viele Records verwendet werden. 'Bitanzahl' ist die Anzahl der Bits, die für diesen Record verwendet werden sollen. Diese Zahl kann nicht größer sein als die Anzahl der Bits, die im angegebenen Datentyp gespeichert werden kann.

Beispiel: Checkbox-Verwaltung mit Bitfeldern:

```
TYPE CheckBoxenType
   CB1 : 1 AS INTEGER
   CB2 : 1 AS INTEGER
   CB3 : 1 AS INTEGER
END TYPE
DIM CheckBoxen AS CheckBoxenType

' Status setzen:
CheckBoxen.CB1 = 1 ' aktiv
CheckBoxen.CB2 = 0 ' nicht aktiv
CheckBoxen.CB3 = 1 ' aktiv

' ... Programmcode ...

' Status abfragen:
IF CheckBoxen.CB1 THEN PRINT "Bit 1 ist gesetzt."
IF CheckBoxen.CB2 THEN PRINT "Bit 2 ist gesetzt."
IF CheckBoxen.CB3 THEN PRINT "Bit 3 ist gesetzt."
SLEEP
```

Man erspart sich hierbei die umständliche Bitmanipulation mit AND und OR, außerdem ist sofort klar, welcher Wert gerade bearbeitet wird. Ein weiterer Vorteil ist, dass die Bitzahl nicht 1 sein muss; so können auch Checkboxen mit drei Zuständen realisiert werden (aktiviert/nicht aktiviert/nicht festgelegt).

Beispiel:

```
TYPE a
   a1 : 2 AS INTEGER
   a2 : 2 AS INTEGER
END TYPE
DIM b AS a

b.a1 = 5
PRINT b.a1
SLEEP
```

Ausgabe:

```
1
```

5 ist in Binär &b101. Da für den Record a1 nur zwei Bits verwendet werden, ignoriert FreeBASIC das dritte Bit. Statt &b101 wird in a1 also nur &b01 gespeichert, was 1 entspricht.

Letzte Bearbeitung des Eintrags am 30.12.12 um 01:23:44
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

Präprozessoren

FreeBASIC-Referenz » BASIC-Grundlagen » **Präprozessoren**
Präprozessor-Anweisungen sind Symbole, die vor der Compilation durch einen bestimmten Ausdruck ersetzt werden; siehe #DEFINE.

Themen:

- Vordefinierte Symbole
- Symbole ohne Wert
- Makros

Vordefinierte Symbole
Beachten Sie, dass zu jedem Symbol ein eigener Eintrag in der Referenz existiert.

Symbolbezeichner	Übersetzung/Bedeutung
__FB_WIN32__	Ohne Wert; wird definiert, wenn der Code von der Win32-Version des Compilers umgesetzt wird. Durch #IFDEF ist es möglich, bestimmte Codeteile nur zu compilieren, wenn der Code für Win32 umgesetzt werden soll
__FB_DOS__	Ohne Wert; wird definiert, wenn der Code von der DOS-Version des Compilers umgesetzt wird. Durch #IFDEF ist es möglich, bestimmte Codeteile nur zu compilieren, wenn der Code für DOS umgesetzt werden soll
__FB_LINUX__	Ohne Wert; wird definiert, wenn der Code von der LINUX-Version des Compilers umgesetzt wird. Durch #IFDEF ist es möglich, bestimmte Codeteile nur zu compilieren, wenn der Code für Linux umgesetzt werden soll
__FB_XBOX__ , __FB_CYGWIN__ , __FB_DARWIN__ , __FB_FREEBSD__ , __FB_OPENBSD__ , __FB_NETBSD__	Ohne Wert; diese Symbole werden definiert, wenn der Code in der jeweiligen Version des Compilers umgesetzt wird. Durch #IFDEF ist es möglich, bestimmte Codeteile nur zu compilieren, wenn der Code für diese bestimmte Version umgesetzt werden soll.
__FB_PCOS__	Ohne Wert; wird definiert, wenn unter einem PC-artigen Betriebssystem compiliert wurde (z. B. DOS, Windows, OS/2, Symbian OS)
__FB_UNIX__	Ohne Wert; wird definiert, wenn unter einem UNIX-artigen Betriebssystem compiliert wurde
__FB_MAIN__	Ohne Wert; wird definiert, sobald die DEFINE-Symbole und Makros des Hauptmoduls übersetzt werden.
__FB_SIGNATURE__	ein String, der die Signatur des Compilers enthält, z. B.: "FreeBASIC 0.24.0"
__FB_VERSION__	ein String, der die Version des Compilers enthält, z. B.: "0.24.0"
__FB_VER_MAJOR__	ein Integer, der die Versionshauptnummer des Compilers enthält, z. B.: 0
__FB_VER_MINOR__	ein Integer, der die Versionsunternummer des Compilers enthält, z. B.: 24
__FB_VER_PATCH__	

	ein Integer, der die Patchnummer des Compilers enthält, z. B.: 0
__FB_MIN_VERSION__(major, minor, patch)	Dieses Makro vergleicht die Version des verwendeten Compilers mit den angegebenen Daten. Es gibt -1 aus, wenn die Version des Compilers größer oder gleich den Spezifikationen ist, bzw. 0, wenn die Version kleiner ist.
__DATE__	ein String, der das Compilier-Datum im Format mm-dd-yyyy enthält, z. B.: "11-15-2012"
__DATE_ISO__	ein String, der das Compilier-Datum im Format yyyy-mm-dd enthält, z. B.: "2012-11-15"
__TIME__	ein String, der die Compilier-Uhrzeit im Format hh:mm:ss enthält, z. B.: "16:05:09"
__FB_BUILD_DATE__	ein String, der das Datum, an dem der FB-Compiler erstellt wurde, im Format mm-dd-yyyy enthält
__FILE__	ein String, der den Dateinamen der Quellcode-Datei, die gerade umgesetzt wird, enthält; dies kann zur Fehlersuche eingesetzt werden
__FILE_NQ__	eine Zeichenkette, die den Namen des Moduls enthält, das gerade umgesetzt wird, jedoch nicht in Anführungszeichen wie bei __FILE__
__FUNCTION__	ein String, der den Namen der Prozedur, die gerade umgesetzt wird, enthält; dies kann zur Fehlersuche eingesetzt werden
__FUNCTION_NQ__	eine Zeichenkette, die das Symbol der Prozedur enthält, die gerade umgesetzt wird, jedoch nicht in Anführungszeichen wie bei __FUNCTION__
__LINE__	ein Integer, der die Zeile angibt, die gerade umgesetzt wird; dies kann zur Fehlersuche eingesetzt werden
__PATH__	ein String, der den Namen des Verzeichnisses enthält, in dem sich die gerade umgesetzte Quellcode-Datei befindet
__FB_OPTION_EXPLICIT__	ein Integer, dessen Wert -1 ist, wenn OPTION EXPLICIT verwendet wird, bzw. 0, wenn dies nicht der Fall ist
__FB_OPTION_ESCAPE__	ein Integer, dessen Wert -1 ist, wenn OPTION ESCAPE verwendet wird, bzw. 0, wenn dies nicht der Fall ist
__FB_OPTION_DYNAMIC__	ein Integer, dessen Wert -1 ist, wenn OPTION DYNAMIC verwendet wird, bzw. 0, wenn dies nicht der Fall ist
__FB_OPTION_PRIVATE__	ein Integer, dessen Wert -1 ist, wenn OPTION PRIVATE verwendet wird, bzw. 0, wenn dies nicht der Fall ist
__FB_OPTION_BYVAL__	ein Integer, dessen Wert -1 ist, wenn OPTION BYVAL verwendet wird, bzw. 0, wenn dies nicht der Fall ist
__FB_OPTION_GOSUB__	ein Integer, dessen Wert -1 ist, wenn OPTION GOSUB verwendet wird, bzw. 0, wenn dies nicht der Fall ist
__FB_OUT_OBJ__	ein Integer, dessen Wert -1 ist, wenn der Code zu einer Ressource (*.obj) compiliert wird, bzw. 0, wenn zu einem anderen Typ compiliert wird
__FB_OUT_LIB__	ein Integer, dessen Wert -1 ist, wenn der Code zur statischen Bibliothek (lib*.a oder lib*.a.dll für Importbibliotheken) compiliert wird, bzw. 0, wenn zu einem anderen Typ compiliert wird
__FB_OUT_DLL__	ein Integer, dessen Wert -1 ist, wenn der Code zur dynamischen Bibliothek (*.dll) compiliert wird, bzw. 0, wenn

	zu einem anderen Typ compiliert wird
__FB_OUT_EXE__	ein Integer, dessen Wert -1 ist, wenn der Code zur Executable (*.exe) compiliert wird, bzw. 0, wenn zu einem anderen Typ compiliert wird
__FB_ARGC__	eine Zahl, die die Anzahl der Kommandozeilenparameter an das Programm enthält
__FB_ARGV__	ein Pointer auf einen Speicherbereich, in dem sich ZSTRING PTR befinden; diese zeigen auf die einzelnen Kommandozeilenparameter
__FB_BIGENDIAN__	hat den Wert -1, wenn für Big-Endian-Systeme compiliert wird, oder 0, wenn dies nicht der Fall ist
__FB_DEBUG__	hat den Wert -1, wenn mit der Kommandozeilenoption '-g' compiliert wird, oder 0, wenn dies nicht der Fall ist
__FB_ERR__	hat den Wert 0, 1, 3 oder 7, je nach der in der Kommandozeile aktivierten Fehlerbehandlungsroutine (keine, '-e', '-ex' oder '-exx')
__FB_MT__	hat den Wert -1, wenn mit der FB-Multithread-Lib compiliert wird, oder 0, wenn dies nicht der Fall ist
__FB_LANG__	wird zu einem String umgesetzt, der das Argument für die Kommandozeilenoption '-lang' enthält
__FB_SSE__	ohne einen Wert; wird definiert, wenn der Compiler 'SSE floating point arithmetics' compiliert (-fpu SSE)
__FB_FPU__	ein String mit dem Wert "sse", wenn mit 'SSE floating point arithmetics' compiliert wurde, ansonsten "x87"
__FB_FPMODE__	ein String mit dem mittels -fpmode eingestellte Wert ("fast" oder "precise")
__FB_VECTORIZE__	ein Integer mit dem mittels -vec eingestellte Level (0, 1 oder 2)
__FB_GCC__	ein Integer, dessen Wert -1 ist, wenn der Code mit -gen gcc compiliert wird, bzw. 0, wenn -gen gas verwendet wird.
__FB_BACKEND__	ein String mit dem mittels -gen eingestellte Backend ("gas" oder "gcc")
__FB_64BIT__	Ohne Wert; wird definiert, wenn der Code von der 64bit-Version des Compilers umgesetzt wird

Symbole ohne Wert

Ein Symbol muss nicht zwingend einen Wert haben, um nutzbringend eingesetzt werden zu können. Mit DEFINED kann geprüft werden, ob ein Symbol bereits definiert wurde. Auf diese Weise lässt sich z. B. ermitteln, ob eine Datei bereits eingebunden wurde oder auf welcher Plattform das Programm gerade läuft.

Beispiel:

```
' Code der Datei Definitions.bi
"hlkw0">TYPE xyz
  x AS INTEGER
  y AS INTEGER
  z AS INTEGER
END TYPE

'---------'
```

```
' Code der Datei Test.bas

"hlstring">"Definitions.bi"

"hlkw0">"hlstring">"Definitions.bi" wurde nicht eingebunden
"hlkw0">"hlkw0">__FB_WIN32__
  "hlkw0">"hlkw0">DEFINED(__FB_DOS__)
  "hlkw0">"hlkw0">DEFINED(__FB_LINUX__)
  "hlkw0">"reflinkicon" href="temp0562.html">FUNCTIONs Ausdrücke, die von
einem Parameter abhängig sind. Im Gegensatz zu solchen werden die Symbole
aber vor der Compilation in ihre Bedeutung umgesetzt, so dass kein Sprung
im Programm ausgeführt werden muss. Im Gegensatz zu FUNCTIONs allerdings
öffnen Makros keinen SCOPE-Block.
```

Beispiel:

```
"hlzahl">3.141592654
"hlzeichen">(deg) ( (deg / 180) * pi )
"hlzeichen">(rad) ( (rad / pi) * 180 )
PRINT "Pi = " & pi                          ' Ausgabe: 3.141592654
PRINT "180 im Bogenmass = " &deg2rad(180)   ' Ausgabe: 3.141592654
PRINT "Pi in Grad = " &rad2deg(Pi)          ' Ausgabe: 180
```

Eine besondere Bedeutung kommt Makros dadurch zu, dass sie im Code Zeichenketten zu Symbolen verketten können; dies geschieht über die Zeichen "hlkw0">"hlzeichen">(

```
n) t##n
DIM AS INTEGER TmpVar(1) = 5, TmpVar(2) = 7
PRINT "Macros: "; TmpVar(1) * TmpVar(2)
PRINT "Variables: "; t1 * t2
```

Bei jedem Zugriff auf TmpVar wird also tatsächlich auf ein Symbol zugegriffen, das mit 't' beginnt, und mit dem Wert von 'n' endet.

Auch mehrzeilige Makros können erstellt werden. Dazu bedient man sich des Metabefehls "footer"> Letzte Bearbeitung des Eintrags am 15.01.14 um 18:21:56
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

Gültigkeitsbereich von Variablen

FreeBASIC-Referenz » BASIC-Grundlagen » **Gültigkeitsbereich von Variablen**
Eine Variable hat immer nur in dem Umfeld Gültigkeit, in dem sie definiert wird. Solche Gültigkeitsbereiche werden als Scope-Blöcke bezeichnet. Variablen, die im Hauptprogramm deklariert werden, sind in der Regel im gesamten Hauptprogramm gültig oder "sichtbar", nicht jedoch in den Prozeduren (SUBs und FUNCTIONs). Eine Varable, die in einer Prozedur deklariert wurde, ist dagegen nur in dieser Prozedur sichtbar. Die genaueren Regeln, welche Variablen wo sichtbar sind, werden im Folgenden aufgeführt.

In FreeBASIC gibt es vier Kategorien von Scope-Blöcken: LOCAL, SHARED, COMMON und COMMON SHARED. Jeder dieser Blöcke hat seine eigenen Sichtbarkeitsregeln. Außerdem existiert die Anweisung EXTERN, die eine ähnliche Bedeutung wie COMMON SHARED besitzt.

LOCAL

Variablen, die in einem lokalen Scope deklariert wurden, sind auch nur in diesem sichtbar.

- SUB, FUNCTION, das Hauptprogramm und jeder Befehlsblock (SCOPE...END SCOPE, FOR...NEXT, DO...LOOP, IF...END IF usw.) bildet einen eigenen Scope-Block.
- Explizit mit DIM oder REDIM definierte Variablen gelten im Scope-Block, in dem sie deklariert wurden.
- Wenn in einem Scope-Block eine Variable verwendet wird, die in diesem Block nicht explizit deklariert wurde, dann wird die Variable des umgebenden Scope-Blocks verwendet.

Beispiele:

```
DIM AS INTEGER a = 0, b = 0
DO
  a += 1                      ' Zugriff auf die Variable des Hauptmodus
  DIM AS INTEGER b = a^2      ' Dieses b ist nur in der Schleife bekannt
  PRINT "In der Schleife: ";
  PRINT "a =";a, "   b =";b
LOOP UNTIL a = 10
PRINT "Ausserhalb: ";
PRINT "a =";a; "   b =";b
SLEEP

SUB fehlerhafterAufruf
  DIM AS INTEGER c
  PRINT a                     ' erzeugt einen Fehler
END SUB
```

In der DO-Schleife wird die Variable *a* aus dem Hauptmodul verwendet. Deswegen sind Änderungen innerhalb der Schleife anschließend auch im Hauptmodul bekannt. Die Änderungen der Variablen *b* betreffen jedoch nur die Schleife und haben anschließend keine Auswirkung mehr.
Die in der SUB verwendete Variable ist dort nicht bekannt, weshalb es zu einem Compiler-Fehler kommt. Dasselbe würde passieren, wenn man versuchen würde, die in der SUB deklarierte Variable *c* im Hauptprogramm aufzurufen.

In der FOR-Schleife kann die Zählvariable direkt als lokale Variable des FOR-Blocks deklariert werden. Sie ist dann außerhalb der Schleife nicht mehr gültig. Ermöglicht wird dies durch die Syntax FOR variable AS datentyp = start TO ende

```
DIM AS STRING s = "Hallo Welt!"
FOR s AS INTEGER = 1 to 3
  PRINT s
NEXT
PRINT s
```

Achtung: Während im oben stehenden Beispiel die Variable im Schleifenkopf deklariert wurde und daher auch im Schleifenfuß abgefragt werden kann, ist eine im Schleifenrumpf deklarierte Variable im Kopf und Fuß der Schleife nicht bekannt und kann dort nicht als Abbruchbedingung genutzt werden. Es ist in diesem Fall nur möglich, die Schleife über EXIT zu verlassen.

```
' funktioniert nicht:
' DO
'   DIM AS STRING taste = INKEY ' Deklaration im Rumpf
'   SLEEP 1
' LOOP UNTIL LEN(taste)          ' taste ist hier nicht bekannt


' Alternative
DO
  DIM AS STRING taste = INKEY
  SLEEP 1
  IF LEN(taste) THEN EXIT DO
LOOP
```

SHARED

Globale Variablen, die mit dem Befehl DIM SHARED deklariert wurden, sind im ganzen Modul einschließlich seiner SUBs und FUNCTIONs sichtbar. Das Modul teilt also die Variable mit seinen Prozeduren.
DIM SHARED darf nur auf Modulebene verwendet werden, nicht jedoch in einer SUB, einer FUNCTION oder einem Befehlsblock (wie DO...LOOP). Eine mit DIM SHARED deklarierte globale Variable kann jedoch wieder lokal "überschrieben" werden.

Beispiel:

```
DECLARE SUB test
DIM SHARED AS INTEGER variable = 1
PRINT "vor der SUB:   "; variable    ' globale Variable
test
PRINT "nach der SUB:  "; variable    ' globale Variable, geändert
SLEEP

SUB test
  variable += 1                      ' globale Variable ändern
  PRINT "in der SUB (1):"; variable  ' noch die globale Variable
  DIM AS STRING variable = "neu"
  PRINT "in der SUB (2):"; variable  ' jetzt eine lokale Variable
END SUB
```

COMMON

COMMON kann zum Einsatz kommen, wenn ein Programm aus mehreren Modulen zusammengesetzt ist, d. h. wenn es aus mehreren einzelnen .bas-Dateien besteht, die zusammen compiliert werden. Wird eine

Variable mit der Anweisung COMMON deklariert, dann steht sie nicht nur in diesem Modul zur Verfügung, sondern auch in allen anderen Modulen, welche die Variable ebenfalls mit COMMON deklarieren.

Für folgendes Beispiel müssen Sie beide Code-Teile unter dem jeweils angegebenen Namen speichern. Compilieren Sie anschließend mit

```
fbc modul1.bas modul2.bas
```

modul1.bas

```
COMMON m1 AS INTEGER
COMMON m2 AS INTEGER

' Dies wird nach allen anderen Modulen ausgeführt
m1 = 1

PRINT "Modul 1"
PRINT "m1 = "; m1      ' m1 = 1, wie in diesem Modul gesetzt
PRINT "m2 = "; m2      ' m2 = 2, wie im Modul 2 gesetzt
SLEEP
```

module2.bas

```
COMMON m1 AS INTEGER
COMMON m2 AS INTEGER

m2 = 2

' Dies wird zuerst ausgeführt
PRINT "Modul 2"
PRINT "m1 = "; m1      ' m1 = 0 (Standardbelegung)
PRINT "m2 = "; m2      ' m2 = 2

SUB unterprogramm
  ' m1 und m2 sind hier nicht sichtbar. Ein Befehl wie
  ' PRINT "m1 = "; m1
  ' würde zu einem Compilier-Fehler führen.
END SUB
```

Ausgabe:

```
Modul 2
m1 = 0
m2 = 2
Modul 1
m1 = 1
m2 = 2
```

COMMON SHARED

Wie unter COMMON steht die mit COMMON SHARED deklarierte Variable in allen Modulen zur Verfügung. Innerhalb eines Moduls sorgt das Schlüsselwort SHARED außerdem dafür, dass die Variable in allen SUBs und FUNCTIONs des betreffenden Moduls sichtbar ist.

Für folgendes Beispiel müssen Sie beide Code-Teile unter dem jeweils angegebenen Namen speichern. Compilieren Sie anschließend mit

```
fbc modul3.bas modul4.bas
```

modul3.bas

```
DECLARE SUB unterprogramm
COMMON m1 AS INTEGER
COMMON m2 AS INTEGER

' Dies wird nach allen anderen Modulen ausgeführt
m1 = 1

PRINT "Modul 3"
PRINT "m1 = "; m1      ' m1 = 1, wie in diesem Modul gesetzt
PRINT "m2 = "; m2      ' m2 = 2, wie im Modul 4 gesetzt

unterprogramm
SLEEP
```

module4.bas

```
COMMON SHARED m1 AS INTEGER
COMMON SHARED m2 AS INTEGER

m2 = 2

' Dies wird zuerst ausgeführt
PRINT "Modul 4"
PRINT "m1 = "; m1      ' m1 = 0 (Standardbelegung)
PRINT "m2 = "; m2      ' m2 = 2

SUB unterprogramm
  PRINT "Modul4.unterprogramm"
  PRINT "m1 = "; m1      ' m1 = 1
  PRINT "m2 = "; m2      ' m2 = 2
END SUB
```

Ausgabe:

```
Modul 4
m1 = 0
m2 = 2
Modul 3
m1 = 1
m2 = 2
Modul4.unterprogramm
m1 = 1
m2 = 2
```

EXTERN

Der Vollständigkeit halber sei die Möglichkeit erwähnt, EXTERN zu verwenden. Die Funktionsweise entspricht in etwa der von COMMON SHARED. Die Variable wird also in einem Modul deklariert und in einem anderem verwenden. Dabei ist die Variable im Modul automatisch global, auch ohne SHARED.

Doch während COMMON SHARED in jedem Modul eigenen Speicher für die Variable reserviert, zeigt EXTERN immer nur auf den Speicherbereich des Moduls, in dem die Variable auch normal deklariert wurde.

Für folgendes Beispiel müssen Sie beide Code-Teile unter dem jeweils angegebenen Namen speichern. Compilieren Sie anschließend mit

```
fbc modul5.bas modul6.bas
```

modul5.bas

```
Declare Sub unterprogramm
Extern m1 As Integer
Extern m2 As Integer
Dim m1 As Integer
Dim m2 As Integer

' Dies wird nach allen anderen Modulen ausgeführt
m1 = 1

Print "Modul 5"
Print "m1 = "; m1      ' m1 = 1, wie in diesem Modul gesetzt
Print "m2 = "; m2      ' m2 = 2, wie im Modul 6 gesetzt

unterprogramm
Sleep
```

module6.bas

```
Extern m1 As Integer
Extern m2 As Integer

m2 = 2

' Dies wird zuerst ausgeführt
Print "Modul 6"
Print "m1 = "; m1      ' m1 = 0 (Standardbelegung)
Print "m2 = "; m2      ' m2 = 2

Sub unterprogramm
  Print "Modul6.unterprogramm"
  Print "m1 = "; m1      ' m1 = 1
  Print "m2 = "; m2      ' m2 = 2
End Sub
```

Ausgabe:

```
Modul 6
m1 = 0
```

```
m2 = 2
Modul 5
m1 = 1
m2 = 2
Modul6.unterprogramm
m1 = 1
m2 = 2
```

Letzte Bearbeitung des Eintrags am 30.12.12 um 01:59:23
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

Plus

FreeBASIC-Referenz » Operatoren in FreeBASIC » **Plus**
Syntax A: Wert = Ausdruck1 + Ausdruck2
Syntax B: Ausdruck1 += Ausdruck2
Typ: Operator
Kategorie: Operatoren

Das Plus-Zeichen kann in FreeBASIC zur einfachen (Syntax A) und kombinierten (Syntax B) Addition eingesetzt werden. Syntax B ist eine Kurzform von

```
Ausdruck1 = Ausdruck1 + Ausdruck2
```

Je nach Datentyp der Ausdrücke bewirkt die Addition unterschiedliches. Zahlen werden wie gewohnt addiert, Strings werden aneinander gehängt. Außerdem gibt es auch die Möglichkeit, zu einem Pointer zu addieren.

Das Plus-Zeichen kann mithilfe von OPERATOR überladen werden.

Addition zweier Zahlen
In dieser Form eingesetzt bewirkt das Plus die Addition zweier Argumente; der Rückgabewert ist die Summe der beiden Argumente. Die Argumente dürfen Zahlen, numerische Konstanten, numerische Variablen, und numerische Rückgabewerte von Funktionen sein. Die Addition ist die Gegenfunktion zur Subtraktion.

Beispiel:

```
DIM AS SINGLE n
n = 4.54 + 5.46
n += 7.5
PRINT n
SLEEP
```

Ausgabe:

```
17.5
```

Stringverkettung
Auf Strings angewendet verkettet das Plus-Zeichen zwei Strings zu einem, in dem auf den Inhalt von String1 der von String2 folgt. Dieses Konstrukt funktioniert mit Strings, ZStrings und WStrings. Zur Stringverkettung kann auch das Zeichen & verwendet werden.

Beispiel:

```
DIM AS STRING a, c
a = "HELLO,"
a += " WORLD"
c = a + "!"
PRINT c
SLEEP
```

Ausgabe:

```
HELLO, WORLD!
```

Addition an Pointern
Addition an Pointern ermöglicht, die Adresse zu ändern, auf die ein Pointer zeigt. Der addierte Wert wird jedoch zuvor mit SIZEOF(Pointertyp) multipliziert. Dies vereinfacht dem Programmierer, sicherzustellen, dass seine Pointer auf sinnvolle Adressen zeigen.

Beispiel:

```
DIM a(10) AS INTEGER
DIM pa AS INTEGER PTR
DIM i AS INTEGER

' Das Array mit Werten von 0 bis 10 befüllen
FOR i = 0 TO 10
  a(i) = i
NEXT

' Einen Pointer auf den Beginn des Arrays setzen
pa = @a(0)

FOR i = 0 TO 10
  ' Den Wert ausgeben, auf den der Pointer gerade zeigt:
  PRINT *pa;
  ' den Pointer um eine INTEGER-Stelle verschieben:
  pa += 1
NEXT
SLEEP
```

Ausgabe:

```
0 1 2 3 4 5 6 7 8 9 10
```

Erläuterung: Mit jeder Zeile

```
pa += 1
```

wird zum Wert von 'pa' tatsächlich 4 addiert, denn 4=1*SIZEOF(INTEGER). Würden wir hier einen SHORT-Pointer (und dazu sinnvollerweise ein SHORT-Array) verwenden, würde sich der Wert um 2 verschieben.

Unterschiede zu QB:

- Kombinierte Operatoren sind neu in FreeBASIC.
- Addition an Pointern ist neu in FreeBASIC.

Unterschiede zu früheren Versionen von FreeBASIC:
Seit FreeBASIC v0.14 wird bei der Subtraktion an Pointern automatisch die Multiplikation mit der Größe des Pointertyps durchgeführt. Davor musste der Programmierer selbst sicherstellen, dass Pointer weit genug verschoben wurden.

Siehe auch:

Und (et-Ligatur), Pointer, mathematische Funktionen

Letzte Bearbeitung des Eintrags am 05.01.13 um 22:30:41
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

At

FreeBASIC-Referenz » Operatoren in FreeBASIC » **At**
Syntax: @ { Variable | Stringkonstante | Prozedur }
Typ: Operator
Kategorie: Operatoren

Der @-Operator gibt die Adresse seines Operanden zurück. @ kann anstelle jeder beliebigen FB-eigenen xxxPTR-Funktion benutzt werden (abgesehen von STRPTR bzw. SADD).

Wenn der @-Operator mit einem String variabler Länge benutzt wird, gibt er einen Pointer auf den internen STRING-Bezeichner zurück. Benutzen Sie STRPTR oder SADD, um die Adresse der eigentlichen Zeichen zu erhalten. Siehe auch STRING (Datentyp) für Informationen zum STRING-Bezeichner bzw. dem internen Management von Zeichenketten.

Wenn dieser Operator mit einem Array verwendet wird, gibt er einen Pointer auf das per Index angegebene Element des Arrays zurück. Beispielsweise gibt

```
@myArray(7)
```

einen Pointer auf myArray(7) zurück. Um einen Pointer auf den Beginn eines Arrays zu erhalten, geben Sie den niedrigsten Index an (siehe LBOUND).

@ kann mithilfe von OPERATOR überladen werden.

Beispiel 1:

```
DIM AS INTEGER a, b
DIM AS INTEGER PTR addr

a = 5
b = 10

PRINT "Der Wert von a ist "; a;
PRINT "; er ist gespeichert an Adresse "; @a
PRINT "Der Wert von b ist "; b;
PRINT "; er ist gespeichert an Adresse "; @b

' Die Adresse von a wird jetzt in dem
' INTEGER PTR gespeichert:
addr = @a

PRINT "Der Pointer addr zeigt jetzt auf a."
PRINT "Der Wert von a kann mit dem Pointer ";
PRINT "ermittelt werden: "; *addr
SLEEP
```

Während der Operator bei Strings variabler Länge nicht auf den Inhalt der Zeichenkette zeigt, kann er bei Stringkonstanten angewandt werden.

Beispiel 2:

```
DIM  AS  ZSTRING  PTR  YesNoMaybe(2) => { _
  @"ja", @"nein", @"vielleicht" }
```

```
DIM AS INTEGER i

PRINT "Standard-Liebesbrief v7.23"
PRINT
PRINT "Willst du mit mir gehen?"
PRINT
FOR i = 0 TO 2
  PRINT i; ")", *YesNoMaybe(i)
NEXT
DO
  i = VAL(INPUT(1))
LOOP UNTIL i < 3
PRINT
PRINT "Du hast "; *YesNoMaybe(i); " gesagt."
SLEEP
```

Möglich ist auch die Verwendung dieses Operators mit Prozeduren:

```
DECLARE SUB Morgen
DECLARE SUB Mittag
DECLARE SUB Nacht

RANDOMIZE

SCREENRES 320, 200

DIM Tageszeit(2) AS SUB ()
DIM AS INTEGER i

Tageszeit(0) = @Morgen()
Tageszeit(1) = @Mittag()
Tageszeit(2) = @Nacht()

i = INT(RND*3)
Tageszeit(i)()
SLEEP

SUB Morgen
  PAINT  (  0,   0), 176
  CIRCLE (100, 120), 50, 42, , , , F
  CIRCLE ( 50, 199), 200, 192, , , .5, F
  CIRCLE (319, 199), 100, 195, , , .3, F
END SUB

SUB Mittag
  PAINT  (  0,   0), 53
  CIRCLE (170,  60), 50, 44, , , , F
  CIRCLE ( 50, 199), 200, 120, , , .5, F
  CIRCLE (319, 199), 100, 123, , , .3, F
END SUB

SUB Nacht
  PAINT (0, 0), 0
  DIM AS INTEGER i
```

```
  FOR i = 0 TO 200
    PSET (RND * 320, RND * 200), 31
  NEXT

  CIRCLE (170,  40), 30, 27, , , , F
  FOR i = 0 TO 10
    CIRCLE (COS( 6.28 * i / 5) * RND*23 + 170, _
        SIN ( 6.28 * i / 5 ) * RND*23 + 40 ), _
        RND*2 + 3, 23, , , , F
  NEXT

  CIRCLE ( 50, 199), 200, 240, , , .5, F
  CIRCLE (319, 199), 100, 243, , , .3, F
END SUB
```

Die Verwendung mit Prozeduren stellt auch die Grundlage für Callbacks dar.

Unterschiede zu QB: neu in FreeBASIC

Siehe auch:
* (Wert von), &"reflinkicon" href="temp0542.html">Pfeil, VARPTR, PROCPTR, STRPTR, SADD, Pointer

Letzte Bearbeitung des Eintrags am 02.01.13 um 23:15:28
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

Punkt

FreeBASIC-Referenz » Operatoren in FreeBASIC » **Punkt**
Syntax: UDTBezeichner.Feldname
Typ: Operator
Kategorie: Operatoren

Der Punkt (Feld-Operator) wählt für die angegebene Operation ein Feld aus einer Variable eines benutzerdefinierten Typs (siehe: TYPE (UDT)).

Wenn mit der Compileroption -lang qb compiliert wurde, sind zur Kompatibilität mit QB außerdem 'gefälschte' Punkt-Operatoren in den Namen einfacher Variablen zulässig.

Beispiel:

```
"hlstring">"qb"
TYPE  rect
  x AS INTEGER
  y AS INTEGER
END TYPE

DIM r AS rect
DIM fake.dot AS INTEGER
r.x = 4
r.y = 2
fake.dot = 1

PRINT "x= "; r.x, "y="; r.y
PRINT fake.dot

SLEEP
```

Siehe auch:
TYPE (UDT), TYPE (Funktion), WITH, -> (Pointer-Dereferenzierung), @ (Adresse von), * (Wert von)

Letzte Bearbeitung des Eintrags am 19.01.13 um 17:24:43
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

Pfeil

FreeBASIC-Referenz » Operatoren in FreeBASIC » **Pfeil**
Syntax: UDT_Pointer->Feldname
Typ: Operator
Kategorie: Operatoren

Der Pfeil-Operator greift auf ein Element eines Feldes zu, zu dem ein Pointer angegeben wurde.
Der Operator kann mithilfe von OPERATOR überladen werden.

Der Pointer auf das gewünschte Element kann auch mithilfe von OFFSETOF berechnet und anschließend dereferenziert werden. Aufgrund der Pointerarithmetik ist es dabei jedoch notwendig, auf die richtigen Typen zu casten, was diese Methode sehr umständlich macht. Im folgenden Beispiel wird sie zur Demonstration aufgeführt.

Beispiel:

```
TYPE rect
  x AS SHORT
  y AS INTEGER
END TYPE

DIM r  AS rect
DIM rp AS rect PTR = @r

rp->x = 4
rp->y = 2

' Direkter Zugriff ohne Pointer
PRINT "x = " & r.x   & ", y = " & r.y
' Zugriff über den Pfeil-Operator
PRINT "x = " & rp->x & ", y = " & rp->y
' Pointer-Zugriff über OFFSETOF
PRINT "x = " & *Cast(Short   Ptr, (Cast(UInteger, rp) + OFFSETOF(rect,
x))) & _
    ", y = " & *Cast(Integer Ptr, (Cast(UInteger, rp) + OFFSETOF(rect,
y)))
SLEEP
```

Wie man sieht, wird tatsächlich die Variable 'r' durch Zugriffe über diesen Operator verändert.

Beispiel zur Operator-Überladung:

```
Type T Extends Object
    Public:
    Declare Function memberFunction() As Integer
End Type
Function T.memberFunction() As Integer
    Return 15 'irgendeine beispielhafte Rückgabelogik
End Function

Type D
    As T Ptr m_ptr
End Type
```

```
Operator -> (ByRef d_ As D) ByRef As T
    Return *d_.m_ptr
End Operator

Dim As D d_
Print d_->memberFunction()
Sleep
```

In dem Beispiel wird der Pfeil für die Klasse D so überladen, dass als Rückgabe der innere T Ptr dereferenziert wird. Dabei ist die BYREF Rückgabe entscheidend.
Somit kann direkt auf memberFunction von m_ptr zugegriffen werden.

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC:

- existiert seit FreeBASIC v0.13
- die gezeigte Form der Überladung ist erst mit BYREF in FreeBASIC v0.90 möglich geworden

Unterschiede unter den FB-Dialektformen:
In der Dialektform -lang qb steht dieser Operator nicht zur Verfügung.

Siehe auch:
TYPE (UDT), TYPE (Funktion), WITH, OFFSETOF, . (Feldzugriff), @ (Adresse von), * (Wert von)

Letzte Bearbeitung des Eintrags am 25.01.14 um 10:42:37
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

Minus

FreeBASIC-Referenz » Operatoren in FreeBASIC » **Minus**
Syntax A: Wert = Ausdruck1 - Ausdruck2
Syntax B: Ausdruck1 -= Ausdruck2
Syntax C: Wert = -Ausdruck2
Typ: Operator
Kategorie: Operatoren

Das Minus-Zeichen kann in FreeBASIC zur Subtraktion (Syntax A und B) und zur Negation (Syntax C) eingesetzt werden. Bei Syntax B handelt es sich um eine kombinierte Subtraktion; dies ist die Kurzform für

```
Ausdruck1 = Ausdruck1 - Ausdruck2
```

Sowohl die Subtraktion als auch die Negation kann mithilfe von OPERATOR überladen werden.

Subtraktion zweier Zahlen
Wert = Ausdruck1 - Ausdruck2
Ausdruck1 -= Ausdruck2
In dieser Form eingesetzt bewirkt das Minus die Subtraktion zweier Argumente; der Rückgabewert ist die Differenz der beiden Argumente. Die Argumente dürfen Zahlen, numerische Konstanten, numerische Variablen, und numerische Rückgabewerte von Funktionen sein. Die Subtraktion ist die Gegenfunktion zur Addition.

Beispiel:

```
DIM AS BYTE n
n = 120 - 119
n -= 10
PRINT n
SLEEP
```

Ausgabe:

```
-9
```

Subtraktion an Pointern
neuerPointer = alterPointer - Wert
Pointer -= Wert
Subtraktion an Pointern ermöglicht, die Adresse zu ändern, auf die ein Pointer zeigt. Der subtrahierte Wert wird jedoch zuvor mit SIZEOF(Pointertyp) multipliziert. Dies vereinfacht dem Programmierer, sicherzustellen, dass seine Pointer auf sinnvolle Adressen zeigen.

Beispiel:

```
DIM a(10) AS INTEGER
DIM pa AS INTEGER PTR
DIM i AS INTEGER

FOR i = 0 TO 10
  ' Das Array mit Werten von 0 bis 10 befüllen
```

```
  a(i) = i
NEXT

' Einen Pointer auf das Ende des Arrays setzen
pa = @a(10)

FOR i = 0 TO 10
  ' Den Wert ausgeben, auf den der Pointer gerade zeigt:
  PRINT *pa;
  ' den Pointer um eine INTEGER-Stelle verschieben:
  pa -= 1
NEXT
```

Ausgabe:

```
10 9 8 7 6 5 4 3 2 1 0
```

Erläuterung: Mit jeder Zeile

```
pa -= 1
```

wird vom Wert von 'pa' tatsächlich 4 subtrahiert, denn 4=1*SIZEOF(INTEGER). Würden wir hier einen SHORT-Pointer (und dazu sinnvollerweise ein SHORT-Array) verwenden, würde sich der Wert um 2 verschieben.

Negation
Wert = -Ausdruck
Dieser Operator gibt den negativen Wert des Arguments zurück, ändert also das Vorzeichen. 'Ausdruck' kann eine Zahl, eine numerische Konstante, eine numerische Variable oder ein numerischer Rückgabewert einer Funktionen sein.

Beispiel:

```
DIM n AS LONGINT
PRINT -5
n = 65432568459
n = -n
PRINT n
SLEEP
```

Ausgabe:

```
-5
-65432568459
```

Unterschiede zu QB:

- Kombinierte Operatoren sind neu in FreeBASIC.
- Subtraktion an Pointern ist neu in FreeBASIC.

Unterschiede zu früheren Versionen von FreeBASIC:
Seit FreeBASIC v0.14 wird bei der Subtraktion an Pointern automatisch die Multiplikation mit der Größe des

Pointertyps durchgeführt. Davor musste der Programmierer selbst sicherstellen, dass Pointer weit genug verschoben wurden.

Siehe auch:
Pointer, mathematische Funktionen

Letzte Bearbeitung des Eintrags am 05.01.13 um 22:31:20
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

Stern

FreeBASIC-Referenz » Operatoren in FreeBASIC » **Stern**
Syntax A: Wert = Ausdruck1 * Ausdruck2
Syntax B: Ausdruck1 *= Ausdruck2
Syntax C: *{Pointervariable | Pointerfunktion} [= Ausdruck]
Typ: Operator
Kategorie: Operatoren

Das Stern-Zeichen kann in FreeBASIC zur Multiplikation (Syntax A und B) und zur Pointer-Dereferenzierung (Syntax C) eingesetzt werden. Bei Syntax B handelt es sich um eine kombinierte Multiplikation; dies ist die Kurzform für

```
Ausdruck1 = Ausdruck1 * Ausdruck2
```

Multiplikation
Wert = Ausdruck1 * Ausdruck2
Ausdruck1 *= Ausdruck2
In dieser Form eingesetzt bewirkt der Stern die Multiplikation zweier Argumente; der Rückgabewert ist das Produkt der beiden Argumente. Die Argumente dürfen Zahlen, numerische Konstanten, numerische Variablen, und numerische Rückgabewerte von Funktionen sein. Die Multiplikation ist die Gegenfunktion der Division.

Die Multiplikation kann mithilfe von OPERATOR überladen werden.

Beispiel 1:

```
DIM n AS DOUBLE
n = 4 * 5
PRINT n
SLEEP
```

Ausgabe:

```
20
```

Beispiel 2:

```
DIM n AS DOUBLE
n = 6
n *= n
PRINT n
SLEEP
```

Ausgabe:

```
36
```

Pointer-Dereferenzierung
*{Pointervariable | Pointerfunction} &"reflinkicon" href="temp0533.html">Pointer zeigt. Der Zugriff kann

sowohl zum Lesen als auch zum Schreiben erfolgen. Der 'Indirection-Level' kann unendlich groß sein. (Pointer auf einen Pointer auf ...) Um sinnvoll eingesetzt werden zu können, müssen die Pointer, mit denen * verwendet wird, auf eine sinnvolle Adresse zeigen. Zahlenwerte sind dafür nicht vorhersagbar, die Adressen müssen auf anderem Wege ermittelt werden, siehe dazu Pointer.

Beispiel:

```
' Zugriff auf einen Pointer mittels *

DIM a AS INTEGER
DIM pa AS INTEGER PTR

' @ wird benutzt, um pa auf a zeigen zu lassen.
' 'a' ist in diesem Fall eine normale Integer-Variable:
pa = @a

a = 9     ' 'a' erhält den Wert 9.

' Den Wert mittels seiner Adresse anzeigen:
PRINT "Der Wert von 'a' ist"; *pa

' An die Adresse des Pointers einen neuen Wert schreiben:
*pa = 1
' Den neuen Wert von 'a' anzeigen:
PRINT "Der neue Wert von 'a' ist"; a
```

Ausgabe:

```
Der Wert von 'a' ist 9
Der neue Wert von 'a' ist 1
```

Unterschiede zu QB:

- Kombinierte Operatoren sind neu in FreeBASIC.
- Pointer-Dereferenzierung ist neu in FreeBASIC.

Siehe auch:
At (@), Pointer, mathematische Funktionen

Letzte Bearbeitung des Eintrags am 05.01.13 um 22:05:27
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

Slash

FreeBASIC-Referenz » Operatoren in FreeBASIC » **Slash**
Syntax A: Wert = Ausdruck1 / Ausdruck2
Syntax B: Ausdruck1 /= Ausdruck2
Typ: Operator
Kategorie: Operatoren

Das Slash-Zeichen kann in FreeBASIC zur einfachen (Syntax A) oder kombinierten (Syntax B) Division eingesetzt werden. Syntax B ist eine Kurzform von

```
Ausdruck1 = Ausdruck1 / Ausdruck2
```

Der Slash bewirkt die Division zweier Argumente; der Rückgabewert ist der Quotient aus 'Ausdruck1' und 'Ausdruck2'. Die Argumente dürfen Zahlen, numerische Konstanten, numerische Variablen, und numerische Rückgabewerte von Funktionen sein. Der Rückgabewert einer Division ist immer DOUBLE. Die Division ist die Gegenfunktion zur Multiplikation.

Die Division kann mithilfe von OPERATOR überladen werden.

Beispiel 1:

```
DIM AS SINGLE n, m
n = 24 / 6
m = n / -8
PRINT n, m
SLEEP
```

Ausgabe:

```
4             -0.5
```

Beispiel 2:

```
DIM n AS DOUBLE
n = 25
n /= 2
PRINT n
SLEEP
```

Ausgabe:

```
12.5
```

Unterschiede zu QB:
Kombinierte Operatoren sind neu in FreeBASIC.

Siehe auch:
Backslash, mathematische Funktionen

Letzte Bearbeitung des Eintrags am 05.01.13 um 21:27:51
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

Backslash

FreeBASIC-Referenz » Operatoren in FreeBASIC » **Backslash**
Syntax A: Wert = Ausdruck1 \ Ausdruck2
Syntax B: Ausdruck1 \= Ausdruck2
Typ: Operator
Kategorie: Operatoren

Das Backslash-Zeichen kann in FreeBASIC zur einfachen (Syntax A) oder kombinierten (Syntax B) Integerdivision eingesetzt werden. Syntax B ist eine Kurzform von

```
Ausdruck1 = Ausdruck1 \ Ausdruck2
```

Der Backslash bewirkt die Integerdivision zweier Argumente; der Rückgabewert ist der ganzzahlige Quotient aus 'Ausdruck1' und 'Ausdruck2'. Die Argumente dürfen Zahlen, numerische Konstanten, numerische Variablen und numerische Rückgabewerte von Funktionen sein. Der Rückgabewert der Division ist immer INTEGER, selbst wenn das Ergebnis der Division Nachkommastellen hätte. Die Nachkommastellen werden dabei abgeschnitten (vgl. FIX). Bei Berechnungen mit Variablen wird die Integerdivision allerdings wesentlich schneller berechnet als die normale Division; sie sollte daher immer benutzt werden, wenn ohnehin nur mit Integers gerechnet wird.

Wird die Integerdivision mit Fließkommawerten als Argumenten benutzt, so werden diese zuerst mithilfe von CINT mathematisch gerundet.

Die Integerdivision kann mithilfe von OPERATOR überladen werden.

Beispiel:

```
DIM n AS DOUBLE
n = 7 \ 2.6  '' => 7 \ 3  => 2.33333  => 2
PRINT n
n = 7
n \= 2.4  '' => 7 \ 2 => 3.5 => 3
PRINT n
SLEEP
```

Ausgabe:

```
2
3
```

Unterschiede zu QB:
Kombinierte Operatoren sind neu in FreeBASIC.

Siehe auch:
Slash, MOD, INT, mathematische Funktionen

Letzte Bearbeitung des Eintrags am 06.08.14 um 11:36:30
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

Exp

FreeBASIC-Referenz » Operatoren in FreeBASIC » **Exp**
Syntax A: Wert = Ausdruck1 ^ Ausdruck2
Syntax B: Ausdruck1 ^= Ausdruck2
Typ: Operator
Kategorie: Operatoren

Das Exponent-Zeichen kann in FreeBASIC zur einfachen (Syntax A) und kombinierten (Syntax B) Potenzrechnung eingesetzt werden. Syntax B ist eine Kurzform von

```
Ausdruck1 = Ausdruck1 ^ Ausdruck2
```

Dieser Operator gibt die Potenz mit Basis 'Ausdruck1' und Exponent 'Ausdruck2' zurück. Potenzierung bedeutet, dass 'Ausdruck1' so oft mit sich selbst multipliziert wird, wie in 'Ausdruck2' angegeben ist.

```
2 ^ 3
```

entspricht also

```
2 * 2 * 2
```

Wenn der Exponent eine Gleitkommazahl ist, wird eine Wurzel gezogen.

```
2 ^ (1 / 3)
```

entspricht also der dritten Wurzel ("Kubikwurzel") aus 2. Die Argumente dürfen Zahlen, numerische Konstanten, numerische Variablen und numerische Rückgabewerte von Funktionen sein.
Beachten Sie: Der Operator ^ ist ein sehr rechenintensiver Operator!

Das Exponent-Zeichen kann mithilfe von OPERATOR überladen werden.

Beispiel:

```
DIM n AS DOUBLE
INPUT "Bitte geben Sie eine positive Zahl ein: ", n
PRINT
PRINT n;" zum Quadrat ist gleich"; n ^ 2
PRINT "Die fünfte Wurzel von"; n;" ist"; n ^ 0.2
SLEEP
```

Ausgabebeispiel:

```
Bitte geben Sie eine positive Zahl ein: 3.4

3.4 zum Quadrat ist gleich 11.56
Die fünfte Wurzel von 3.4 ist 1.27730844458754
```

Hinweis: Es kann nicht garantiert werden, dass die erzielten Ergebnisse in allen Stellen exakt sind. In den letzten signifikanten Bits der Ergebniszahl kann es architekturbedingt zu Ungenauigkeiten kommen.

Siehe auch:
LOG, EXP, mathematische Funktionen

Letzte Bearbeitung des Eintrags am 02.01.13 um 23:16:35
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

Runde Klammern

FreeBASIC-Referenz » Operatoren in FreeBASIC » **Runde Klammern**
Die runden Klammern werden in FreeBASIC für zwei Sprachkonstrukte verwendet:

- Reihenfolge der Abarbeitung der Glieder eines Ausdrucks
- Indizierung von Arrays

Reihenfolge der Abarbeitung der Glieder eines Ausdrucks
Die runden Klammern werden in FreeBASIC wie in jeder mathematischen Gleichung eingesetzt und beeinflussen die Abarbeitung eines Ausdrucks. Dabei gilt, dass zuerst der Wert der Klammer berechnet wird und anschließend alle anderen Operationen in der Reihenfolge ihrer Priorität abgehandelt werden.

Dabei gilt die folgende Hierarchie der Operatoren:

- Klammern ()
- Potenzierung (^)
- Negierung (-)
- Division (/)
- Multiplikation (*)
- Integerdivision (\)
- Addition, Subtraktion (+, -)
- Vergleiche (=, <>, <, >, <=, >=)
- logische Operatoren (NOT, AND, OR, XOR, EQV, IMP)
- ANDALSO, ORELSE
- Wertzuweisung (=)

Eine vollständige Liste zur Hierarchie findet sich im Referenzartikel zu den Operatoren.

Dementsprechend kommen folgende Werte zustande:

```
PRINT (3 + 4) * 5
PRINT -(1 + 1) ^ 2
PRINT (3 * (5 + 1)) + (-2 - (1 * 1))
```

Ausgabe:

```
35
-4
 15
```

Auch in Bedingungsstrukturen müssen oft Klammern eingesetzt werden, um bei der Abarbeitung die gewünschte Reihenfolge zu erreichen.

Beispiel:

```
DIM AS INTEGER a = 4
IF (a < 10 OR a > 20) AND a <> 4 THEN
  PRINT "a liegt im richtigen Bereich."
END IF

' Überprüfen, ob a eine ungerade Zahl ist
```

```
IF (a AND 1) = 1 THEN PRINT "a ist ungerade."

SLEEP
```

Beide Bedingungen sind nicht erfüllt. Würden die Klammern weggelassen, dann wäre die Abarbeitungsreihenfolge eine andere, weshalb beide Bedingungen erfüllt wären.

Indizierung von Arrays

Wie unter DIM beschrieben, kann man sich ein Array als eine Liste bzw. Tabelle vorstellen, in der Werte gespeichert werden. Der Zugriff auf diese Werte erfolgt mittels Indizes, die in Klammern hinter dem Bezeichner des Arrays genannt werden.

Beispiel 1: Dimensionierung (Erstellung) eines Arrays

```
DIM AS INTEGER myArray(1 TO 10)
'     ^           ^       ^
'     |           |       +-- Anzahl der Elemente
'     |           +---------- Bezeichner des Arrays
'     +---------------------- Datentyp des Arrays
```

Diese Zeile legt das Verhalten des Arrays im Wesentlichen fest.
Auf die einzelnen Elemente des Arrays wird zugegriffen, indem zuerst der Bezeichner genannt wird, gefolgt vom Index des Elements in Klammern. Dabei gelten dieselben Regeln für den Lese- wie für den Schreibzugriff.

Beispiel 2: Zugriff auf Elemente eines Arrays

```
DIM AS INTEGER myArray(10)

myArray(5) = 17

PRINT myArray(5)
SLEEP
```

Bei mehrdimensionalen Arrays werden sowohl bei der Dimensionierung als auch beim Zugriff die Indizes durch Kommata getrennt.

Beispiel 3: Mehrdimensionale Arrays

```
RANDOMIZE TIMER

SCREENRES 320, 200

DIM AS INTEGER myTable(32, 20)
DIM AS INTEGER x, y

FOR x = 0 TO 32
  FOR y = 0 TO 20
    myTable(x, y) = RND * 255
  NEXT
NEXT

FOR x = 0 TO 32
```

```
  FOR y = 0 TO 20
    LINE (x * 10, y * 10) - (x * 10 + 10, _
       y * 10 + 10), myTable(x, y), BF
  NEXT
NEXT

SLEEP
```

Es ist möglich, ganze Arrays an eine Prozedur zu übergeben. Dazu wird als Parameter der Bezeichner des Arrays übergeben, gefolgt von den runden Klammern, jedoch ohne Angabe von Indizes, aber immer mit Typ.

Beispiel 4: Übergabe eines Arrays an eine Prozedur

```
RANDOMIZE

DECLARE SUB ShowTable(Table() AS INTEGER)

DIM AS INTEGER myTable(4, 4)
DIM AS INTEGER x, y

FOR x = 0 TO 4
  FOR y = 0 TO 4
    myTable(x, y) = RND * 9
  NEXT
NEXT

ShowTable myTable()
SLEEP

SUB ShowTable(Table() AS INTEGER)
  DIM AS INTEGER x, y, MaxX, MaxY
  MaxX = UBOUND(Table, 1)
  MaxY = UBOUND(Table, 2)

  FOR x = 0 TO MaxX
    FOR y = 0 TO MaxY
      LOCATE y * 2 + 1, x * 3 + 1
      PRINT Table(x, y); "|"
      LOCATE y * 2 + 2, x * 3 + 1
      PRINT "--+"
    NEXT
  NEXT
END SUB
```

In DECLARE-Zeilen können die Namen der Parameter ausgelassen werden (anonyme Parameter). Wird dabei der Name eines Array-Parameters ausgelassen, müssen dennoch die runden Klammern verwendet werden, um dem Compiler zu signalisieren, dass es sich bei dem anonymen Parameter um ein Array handelt.

Beispiel 5: Anonyme Array-Parameter in DECLARE-Zeilen

```
Declare Sub foo(() As Integer)

Dim As Integer bar(0 To ...) = {0, 1, 2}
foo(bar())
```

```
Sleep

Sub foo(array() As Integer)
  For i As Integer = LBound(array) To UBound(array)
    Print array(i)
  Next
End Sub
```

Unterschiede zu früheren Versionen von FreeBASIC:
Seit FreeBASIC v0.90 können in DECLARE-Zeilen die Namen von Array-Parametern ausgelassen werden.

Letzte Bearbeitung des Eintrags am 02.07.13 um 21:49:55
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

Und (et-Ligatur)

FreeBASIC-Referenz » Operatoren in FreeBASIC » **Und (et-Ligatur)**
Syntax A: Ergebnis = Ausdruck1 & Ausdruck2
Syntax B: Ausdruck1 &= Ausdruck2
Typ: Operator
Kategorie: Operatoren

& kann zur Stringverkettung (Syntax A) und kombinierter Stringverkettung (Syntax B) eingesetzt werden. Syntax B ist eine Kurzform von

```
Ausdruck1 = Ausdruck1 & Ausdruck2
```

Der Rückgabewert ist ein STRING, der aus den beiden aneinandergehängten Werten besteht. Wenn einer der beiden Werte kein String ist, wird automatisch STR aufgerufen, um den Wert in einen String zu verwandeln. Für 'Wert1' und 'Wert2' sind alle STRINGs, ZSTRINGs und WSTRINGs erlaubt, sowie alle Datentypen, die sich über STR in einen String umwandeln lassen. Dies betrifft insbesondere alle Zahlendatentypen. UDTs können nur verwendet werden, wenn sie eine CAST-Methode für Strings bereitstellen (siehe OPERATOR).

Beispiel 1:

```
DIM AS STRING a,c
DIM AS SINGLE b
a = "Das Ergebnis ist: "
b = 124.3
c = a & b
PRINT c
SLEEP
```

Ausgabe:

```
Das Ergebnis ist: 124.3
```

Beispiel 2:

```
DIM s AS STRING
s = "HELLO"
s &= " WORLD "
s &= 15
PRINT s
SLEEP
```

Ausgabe:

```
HELLO WORLD 15
```

Anmerkung: Zwischen dem Variablennamen und dem Zeichen '&' bzw. '&=' muss ein Leerzeichen stehen, damit der Compiler das '&' nicht irrtümlicherweise als Suffix für den Variablennamen hält und es bei der Übersetzung bemängelt. Ähnliches gilt auch zwischen dem '&' und dem Variablennamen. Ein '&' vor einer Variablen wird jedoch nur als Präfix interpretiert, wenn der folgende Buchstabe ein 'h', 'o' oder 'b' ist.

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC: existiert seit FreeBASIC v0.15

Siehe auch:
Plus

Letzte Bearbeitung des Eintrags am 05.01.13 um 22:33:25
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

Eckige Klammern

FreeBASIC-Referenz » Operatoren in FreeBASIC » **Eckige Klammern**
Die eckigen Klammern können in FreeBASIC für zwei Sprachkonstrukte eingesetzt werden:

- String-Indizierung
- Pointer-Indizierung
- Operator-Überladung

String-Indizierung
Syntax: Stringvariable[Index]
Der Stringindizierungs-Operator [ ] ermöglicht es dem Programmmierer, einzelne Zeichen eines Strings über einen Index anzusprechen; dabei ist Lese- und Schreibzugriff möglich. Für STRINGs, ZSTRINGs fester Länge und WSTRINGs fester Länge ist der Rückgabewert der ASCII-Code des indizierten Zeichens. Der Programmierer muss sicherstellen, dass die Indizes von 0 bis LEN(Stringvariable)-1 reichen; andernfalls können die Ergebnisse nicht vorhergesagt werden, da ein Zugriff auf Speicherbereiche auftritt, der nicht mehr zur STRING-Variable gehört, evtl. sogar nicht mehr zum Speicherbereich des laufenden Programms.

Beispiel:

```
DIM txt AS STRING = "Hallo, world!"
DIM i AS INTEGER
txt&"hlzahl">1] = 101  ' im zweiten Buchstaben a durch e tauschen
FOR i = 0 TO LEN(txt) - 1
  PRINT CHR(txt&"hlzeichen">]);
NEXT
SLEEP
```

Ausgabe:

```
Hello, World!
```

Pointerindizierung
Syntax: Pointer&"reflinkicon" href="temp0533.html">Pointer zeigt. Die Adresse im Speicher, auf die tatsächlich zugegriffen wird, errechnet sich nach:

```
Adresse = Pointer + index * SIZEOF(*Pointer)
```

Der Pointer wird dabei nicht verändert. Der Programmierer muss selbst sicherstellen, dass der Pointer auf eine sinnvolle Adresse zeigt.

Beispiel:

```
DIM a(10) AS INTEGER
DIM pa AS INTEGER PTR

FOR i AS INtEGER = 0 To 10
  ' Das Array mit den Werten von 0 bis 10 befüllen
  a(i) = i
NEXT i
```

```
pa = @a(0)

' Den Inhalt des Arrays ausgeben:
FOR i AS INTEGER = 0 TO 10
  PRINT pa&"hlzeichen">]
NEXT i
SLEEP
```

Ausgabe:

```
 0
 1
 2
 3
 4
 5
 6
 7
 8
 9
 10
```

Operator-Überladung
Seit FreeBASIC 0.91 lässt sich &"reflinkicon" href="temp0291.html">Operator innerhalb von eigenen Types überladen.

Beispiel:

```
Type StringWrapper
    As String s

    Declare Operator &"hlzeichen">] (ByVal index As Integer) ByRef As
UByte
End Type

Operator StringWrapper.&"hlzeichen">] (ByVal index As Integer) ByRef As
UByte
    Return s&"hlzeichen">]
End Operator

Dim As String foo = "foo"

Dim As StringWrapper sw
sw.s = "foo"

Print foo&"hlzahl">0], foo[1], foo[2]
Print sw&"hlzahl">0],  sw[1],  sw[2]
Sleep
```

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC: Der Indexoperator &"reflinkicon" href="temp0539.html">Addition und Subtraktion bei Pointern

Letzte Bearbeitung des Eintrags am 21.01.14 um 19:14:32
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

Geschweifte Klammern

FreeBASIC-Referenz » Operatoren in FreeBASIC » **Geschweifte Klammern**
Geschweifte Klammern werden in FreeBASIC zur Array-Initialisierung verwendet.

Initialisierung von Arrays
Arrays (auch Felder oder Matrizen) werden mit der DIM-Anweisung aufgeteilt. So bezeichnet

```
Dim As Integer feld(3, 3)
```

ein 4x4 Elemente großes Array (FreeBASIC benutzt automatisch 0 als Untergrenze).
Die Ober- und Untergrenze kann dabei mit dem Schlüsselwort TO explizit festgelegt werden:

```
Dim As Integer feldA(1 To 3, 1 To 3), feldB(3 TO 7)
```

'feldA' ist hier ein Array mit 3x3 Elementen, 'feldB' ein Array mit 5 Elementen.
Die einzelnen Feldelemente können mit einer direkten Zuweisung oder mittels DATA-Anweisung(en) initialisiert werden, was aber auf dasselbe Prinzip hinausläuft.

Arrays können aber auch direkt bei der Initialisierung mit Werten belegt werden. Im Beispiel handelt es sich u. a. um ein Array von 3 Zeilen und 3 Spalten. Um den einzelnen Elementen Werte zuzuweisen, dienen die geschweiften Klammern (curly brackets) { }. Jede Zeile wird dabei von geschweiften Klammern begrenzt. In einem mehrdimensionalen Array wird die ganze Matrix ebenfalls von geschweiften Klammern umschlossen. Die einzelnen Einträge sowie die Zeilen werden durch Kommata getrennt.
Soll das Array 'feldA' die Elemente

```
1 2 3
4 5 6
7 8 8
```

enthalten und 'feldB' die Zahlen von 1 bis 5, dann sieht die DIM-Anweisung so aus:

```
Dim As Integer feldA(1 To 3, 1 To 3) = { _
    {1, 2, 3}, _
    {4, 5, 6}, _
    {7, 8, 8} _
  }
Dim As Integer feldB(3 To 7) = {1, 2, 3, 4, 5}
```

Die Aufteilung auf mehrere Zeilen mithilfe des Zeilenfortsetzungszeichens dient hier zwar nur zur Übersicht, ist aber auch allgemein zu empfehlen.

Letzte Bearbeitung des Eintrags am 02.01.13 um 00:05:07
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

Zeilenfortsetzungszeichen _

FreeBASIC-Referenz » Operatoren in FreeBASIC » **Zeilenfortsetzungszeichen _**
In vielen Codes (auch in der Befehlsreferenz) befindet sich der Unterstrich _ am Zeilenende. Damit wird dem Compiler mitgeteilt, dass die folgende Zeile Bestandteil der vorangegangenen ist. Der Vorteil ist, dass lange Zeilen (z. B. bei API-Aufrufen) in übersichtliche Pakete aufgeteilt werden können. So kann ein Funktionsaufruf z. B. ausführlich dokumentiert werden, da nach dem _ immer noch das Kommentarzeichen ' angefügt werden kann (gefolgt von einem Kommentar):

```
Function InputLn( _
   byVal s As String, _            ' Meldungs-String
   byVal Sys As String="$", _      ' Mit diesem Zeichen wird der
Eingabe-Typ festgelegt
   byVal Upper As Integer=1=0, _   ' Flag zur automatischen Umwandlung in
Großbuchstaben
   byVal pw As String="", _        ' hier kann ein Zeichen zu
Passwortmaskierung stehen
   byVal AddLf As Integer=1=1, _   ' Flag, um nach der Eingabe ein LF
(neue Zeile) einzufügen
   byVal Comma As String=",", _    ' falls gewünscht, alternatives Zeichen
für '.' übergeben
   byVal Edit As String="" _       ' hier kann ein zu editierender String
übergeben werden
) As String                        ' (C) 2007 by Autor ohne jede Garantie
...
```

Abgesehen davon, dass die Zeile an einem Stück geschrieben ziemlich lang wäre, gäbe es keine Möglichkeit, die einzelnen Parameter mittels ' zu kommentieren, da der Compiler alle Zeichen nach dem ' (dieses einschließlich) ignorieren würde und die Zeile damit nicht syntaxgerecht abgeschlossen würde.

Letzte Bearbeitung des Eintrags am 05.01.13 um 23:11:19
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

Das '?'-Zeichen

FreeBASIC-Referenz » Operatoren in FreeBASIC » **Das '?'-Zeichen**
In den frühen Basic-Versionen gab es zur Eingabevereinfachung die Möglichkeit, den PRINT-Befehl mit dem Zeichen '?' abzukürzen. In einer Interpreter-Sprache und in einer Zeit des stark begrenzten Speicherplatzes war die Abkürzung durchaus sinnvoll. Heute wird diese Methode nicht mehr benötigt. Aus Gründen der Kompatibilität wird sie jedoch auch von FreeBASIC unterstützt. Das '?' kann sowohl als normale PRINT-Anweisung als auch in ? #DateiNr,.. benutzt werden.

Beispiel:

```
Print "Hallo"
? "Wie geht es dir?"
Sleep   ' auf Tastendruck vor dem Beenden warten
```

Unterschiede zu früheren Versionen von FreeBASIC: existiert seit FreeBASIC v0.15

Siehe auch:
PRINT (Anweisung), PRINT (Datei), PRINT USING

Letzte Bearbeitung des Eintrags am 01.01.13 um 20:40:14
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

Das '#'-Zeichen

FreeBASIC-Referenz » Operatoren in FreeBASIC » **Das '#'-Zeichen**
Befehle, die mit einem '#' beginnen, werden als Präprozessor-Anweisungen oder auch Meta-Befehle bezeichnet. Präprozessor-Anweisungen werden an den Compiler geschickt, um zu bestimmen, *was* compiliert werden soll und *wie*. Sie können benutzt werden, um einen bestimmten Anweisungsblock nur unter bestimmten Bedingungen auszuführen, damit das Programm plattformunabhängig compiliert werden kann, inkl. der Header- oder anderen Quell-Dateien. Es können kleine ein- oder mehrzeilige Makros definiert werden, oder es lässt sich ändern, wie der Compiler Variablen behandelt.

Außerdem wird das Zeichen '#' eingesetzt, um festzulegen, dass ein Schreib- bzw. Lesezugriff auf eine Datei stattfinden soll. In diesen Anweisungen steht das '#' jedoch nie am Anfang der Zeile.

Beispiel

```
' Verwendung der Logfile aktivieren - um sie zu deaktivieren,
' muss nur die folgende Zeile entfernt oder auskommentiert werden
"hlkommentar">' ... Programmablauf ...

' Statusmeldung in die Logfile schreiben
"hlkw0">SCOPE
    DIM AS INTEGER file = FREEFILE
    OPEN "status.log" FOR APPEND AS "hlkw0">PRINT "hlzeichen">, DATE; ",
"; TIME; " Uhr: Status X erreicht"
    CLOSE "hlkw0">END SCOPE
"hlkommentar">' ... weiterer Programmablauf
```

Siehe auch:
Präprozessor-Anweisungen, Präprozessoren, PRINT #, INPUT #, OPEN

Letzte Bearbeitung des Eintrags am 19.01.13 um 16:15:32
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

... (Auslassung[Ellipsis])

FreeBASIC-Referenz » Operatoren in FreeBASIC » **... (Auslassung[Ellipsis])**
Die drei Auslassungspunkte werden anstelle von Prozedur-Parametern benutzt, um eine variable Anzahl von Argumenten zu übergeben.

Prozeduren
Die Auslassung (drei Punkte "...") wird in Prozedur-Deklarationen und Definitionen benutzt, um eine variable Argumentenliste anzugeben. Das erste Argument **muss immer** angegeben werden, und die Prozedur muss mit der 'C calling convention' CDECL aufgerufen werden.

Beispiel:

```
Declare Function FOO cdecl (x As Integer, ...) As Integer
```

Der Zugriff auf die Parameter wird über VA_FIRST, VA_ARG und VA_NEXT ermöglicht.

Arrays
Bei der Deklaration von Arrays kann die Auslassung verwendet werden, um die Größe des Arrays durch die Initialisierung festzulegen.

Beispiel:

```
dim a(1 to ...) as integer = {1, 2, 3, 4}
print ubound(a)     ' Ausgabe: 4
sleep
```

Macros
In Makros wird die Auslassung verwendet, um Makros mit variabler Parameteranzahl zu erstellen.

Beispiel:

```
"hlstring">"crt.bi"
"hlzeichen">(Format, args...) fprintf(stderr, Format, args)
eprintf(!"Hello from printf: %i %s %i\n", 5, "test", 123)

"hlzeichen">(a, b...) a
"hlzeichen">(a, b...) b
PRINT "car (1, 2, 3, 4) = "; car(1, 2, 3, 4)
PRINT "cdr (1, 2, 3, 4) = "; cdr(1, 2, 3, 4)
SLEEP
```

Unterschiede zu QB: neu in FreeBASIC

Unterschiede zu früheren Versionen von FreeBASIC:

- Seit FreeBASIC v0.22 können Auslassungen in Makros verwendet werden
- Seit FreeBASIC v0.21 können Auslassungen bei der Deklaration von Arrays verwendet werden.

Siehe auch:
CDECL, VA_FIRST, VA_ARG, VA_NEXT, Arrays, DEFINE (Meta), MACRO (Meta)

Letzte Bearbeitung des Eintrags am 01.01.13 um 00:45:50
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

Doppelpunkt

FreeBASIC-Referenz » Operatoren in FreeBASIC » **Doppelpunkt**
Der Doppelpunkt wird in verschiedenen Sprachkonstrukten verwendet:

- Ein Label ist durch einen abschließenden Doppelpunkt gekennzeichnet; siehe GOTO, RESTORE.
- In UDTs wird nach einem Doppelpunkt die Anzahl benutzter Bits der Variable angegeben; siehe Bitfelder.
- Der Doppelpunkt wird verwendet, um bei der Formatierung von Zeitwerten die Stunden, Minuten und Sekunden voneinander zu trennen; siehe FORMAT, SETTIME.
- Ein Doppelpunkt trennt mehrere BASIC-Anweisungen in einer Zeile; siehe unten.

Beispiel für mehrere BASIC-Anweisungen in einer Zeile:

```
Print "Hi, "; : Print "ich bin ein Text" : Sleep
```

Diese Art der Verwendung kann den Code aber auch sehr unübersichtlich machen. Deswegen sollte man möglichst darauf verzichten und für jede Anweisung eine neue Zeile beginnen.

Letzte Bearbeitung des Eintrags am 01.01.13 um 20:59:24
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

Ausrufezeichen

FreeBASIC-Referenz » Operatoren in FreeBASIC » **Ausrufezeichen**
Das Ausrufezeichen ! aktiviert die Verwendung von Escape-Zeichen in Strings. Dadurch werden in FreeBASIC Backslashes ("\", ASCII-Code 92) als Escape-Zeichen verwendet. Der Ausdruck hinter dem Escape-Zeichen wird nicht 1:1 ausgegeben, sondern zuerst interpretiert. Folgt dem Zeichen eine Zahl, so funktioniert das Zeichen wie ein CHR; anstelle der Zahl wird das zugehörige ASCII-Zeichen ausgegeben. Die Zahl kann in jedem beliebigen Zählsystem angegeben werden, wenn das Präfix &? verwendet wird (z. B. &h für hexadezimal).

Außerdem werden folgende Strings von FreeBASIC gesondert behandelt:

- \r wie ein CHR(13) (carriage-return)
- \n und \l wie ein CHR(10) (line-feed). \r\n ergibt also ein CRLF, unter Windows die EDV-Version eines Zeilenumbruchs.
- \a wie ein CHR(7) (Bell)
- \b wie ein CHR(8) (Backspace)
- \t wie ein CHR(9) (Tab)
- \v wie ein CHR(11) (vtab)
- \f wie ein CHR(12) (formfeed)
- \" wie ein CHR(34) (Doppeltes Anführungszeichen ")
- \' wie ein CHR(39) (Einfaches Anführungszeichen ')
- Alle anderen Zeichen hinter dem Escape-Zeichen werden so interpretiert, wie sie im Ausdruck stehen. \\ wird also zu \.

Beispiel:

```
"hlkw0">__FB_PCOS__
  "hlstring">"\r\n"   ' Windows-Zeilenumbruch
"hlkw0">DEFINED(__FB_UNIX__)
  "hlstring">"\n"     ' UNIX/Linux-Zeilenumbruch
"hlkw0">"hlstring">"Fehler: Plattform nicht unterstützt"
"hlkw0">DIM AS STRING meldung

meldung = !"\45 Das ist die \"erste\" Zeile\r\nDas "
meldung &= !"ist die \"zweite\" Zeile \45"
PRINT meldung

PRINT NEWLINE & "^-- NEWLINE --v" & NEWLINE;
SLEEP
```

Ausgabe:

```
- Das ist die "erste" Zeile
Das ist die "zweite" Zeile -

^-- CRLF --v
```

Das Ausrufezeichen arbeitet als Präprozessor. Es kann nur vor in Anführungszeichen gesetzten Zeichenketten eingesetzt werden, nicht jedoch vor Variablen o. ä. Folgendes Beispiel wird daher einen Compilier-Fehler erzeugen:

```
DIM AS STRING text = "Text mit einem\r\nZeilenumbruch"
PRINT !text     ' nicht erlaubter Aufruf!
```

Bis FreeBASIC v0.16 war es möglich, den Compiler dazu anzuweisen, generell in allen Strings die Escape-Sequenzen zu interpretieren. Dazu wurde die Anweisung OPTION ESCAPE eingesetzt. Um anschließend Strings zu kennzeichnen, deren Inhalt *nicht* interpretiert werden sollte, wurde vor diese Strings ein Dollarzeichen $ gesetzt. Seit FreeBASIC v0.17 ist OPTION ESCAPE jedoch nur noch in der Dialektform -lang deprecated erlaubt. Das Dollarzeichen kann zwar weiter verwendet werden, hat aber ohne OPTION ESCAPE keine praktische Auswirkung mehr.

Unterschiede zu QB: neu in FreeBASIC

Letzte Bearbeitung des Eintrags am 01.01.13 um 15:59:03
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

Dollarzeichen

FreeBASIC-Referenz » Operatoren in FreeBASIC » **Dollarzeichen**
Das Dollarzeichen $ deaktiviert die Verwendung von Escape-Zeichen in Strings.

Üblicherweise werden Escape-Sequenzen in Strings nicht interpretiert. Um sie verwenden zu können, muss dem String, der die Escape-Zeichen enthält, ein Ausrufezeichen vorangestellt werden. Näheres zu den Escape-Sequenzen finden Sie im Referenzeintrag Ausrufezeichen.

Bis FreeBASIC v0.16 war es möglich, den Compiler mit OPTION ESCAPE dazu anzuweisen, generell in allen Strings die Escape-Sequenzen zu interpretieren. Um anschließend Strings zu kennzeichnen, deren Inhalt nicht interpretiert werden sollte, wurde vor diese Strings ein Dollarzeichen $ gesetzt. Seit FreeBASIC v0.17 ist OPTION ESCAPE jedoch nur noch in der Dialektform -lang deprecated erlaubt. Das Dollarzeichen kann zwar weiter verwendet werden, hat aber ohne OPTION ESCAPE keine praktische Auswirkung mehr.

Beispiel:

```
"hlstring">"deprecated"

OPTION ESCAPE  ' Escape-Sequenzen generell interpretieren
PRINT "Der Pfad lautet \34" $"C:\FREEBASIC\33a" "\34"
SLEEP
```

Ausgabe:

```
Der Pfad lautet "C:\FREEBASIC\33a"
```

In der Pfadangabe wurden die Escape-Sequenzen nicht ausgewertet; ansonsten hätten anstelle der beiden Backslashes im Pfad jeweils doppelte Backslashes stehen müssen, um die gewünschte Ausgabe zu erhalten.

Das Dollarzeichen arbeitet wie das Ausrufezeichen als Präprozessor. Es kann nur vor in Anführungszeichen gesetzten Zeichenketten eingesetzt werden, nicht jedoch vor Variablen o. ä. Folgendes Beispiel wird daher einen Compilier-Fehler erzeugen:

```
DIM AS STRING text = !"Text mit einem\r\nZeilenumbruch"
PRINT $text    ' nicht erlaubter Aufruf!
```

Unterschiede zu QB: neu in FreeBASIC

Letzte Bearbeitung des Eintrags am 01.01.13 um 20:50:36
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

Zuweisung

FreeBASIC-Referenz » Operatoren in FreeBASIC » **Zuweisung**
Syntax A: Variable = Wert
Syntax B: Variable => Wert
Typ: Operator
Kategorie: Operatoren

Die Zuweisung von Werten wird in FreeBASIC üblicherweise mit '=' bewerkstelligt, es kann allerdings auch '=>' verwendet werden.

Beispiel 1:

```
Dim As Integer foo1 = 5
Dim As Integer foo2 => 7

Dim As Integer bar1(0 To ...) = {0, 1, 2, 3}
Dim As Integer bar2(0 To ...) => {4, 5, 6, 7}
```

In den meisten Fällen sind beide Zuweisungsoperatoren gleichbedeutend. Da '=' allerdings auch als Vergleichsoperator verwendet wird, gilt '=>' als expliziter Zuweisungsoperator.
Dies ist vor allem bei Funktionen mit einer Rückgabe *'by reference'* wichtig, da die Zuweisung an eine solche Funktion durch den Compiler auch als Vergleich interpretiert werden könnte.

Beispiel 2:

```
Function f(ByRef s As String) ByRef As String
  Return s
End Function

Dim As String s = "foo"
Print f(s)

' f(s) = "bar"         ' <--- Achtung, Fehler!
Print f(s)

f(s) => "foobar"
Print f(s)

Sleep
```

Die auskommentierte Stelle wird in diesem Fall als 'f((s) = "bar")' interpretiert. Dabei handelt es sich also um einen Vergleich, der die Variable *'s'* und den String *"bar"* miteinander vergleicht und einen numerischen Wahrheitswert zurückliefert, der an die Funktion *'f()'* übergeben wird. Da die Funktion allerdings einen String als Übergabeparameter erwartet, führt der Aufruf zwangsläufig zu einem Fehler.
Für diesen speziellen Fall muss '=>' verwendet werden, um dem Compiler zu signalisieren, dass eine explizite Zuweisung erwünscht ist.

Letzte Bearbeitung des Eintrags am 02.07.13 um 00:41:19
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

Einleitung

FreeBASIC-Referenz » Prozeduren in FreeBASIC » **Einleitung**
Unter dem Begriff **"Prozedur"** versteht man einen Programmteil, der vom Hauptprogramm oder einer anderen Prozedur aus aufgerufen werden kann. Oft werden Prozeduren auch als "Unterprogramme" bezeichnet. Eine Prozedur kann von jedem Programmpunkt aus aufgerufen werden. Dadurch ist es möglich, Programmteile, die häufig ausgeführt werden müssen, durch nur einen (CALL-)Befehl auszuführen, anstatt den Codeteil kopieren und einfügen zu müssen. Dies wirkt sich positiv auf die Programmgröße aus. Es ist außerdem möglich, Prozeduren abhängig von einer oder mehreren Variablen arbeiten zu lassen. Siehe dazu weiter unten.

Es gibt einige gute Gründe, in Ihrem Programm Prozeduren einzusetzen:

- Reduzierung von redundanten Programmzeilen
- Möglichkeit der Wiederverwendung von Programmzeilen in anderen Projekten
- Leichtere Les- und Wartbarkeit des Programms
- Leichtere Erweiterbarkeit Ihres Programms

Man unterscheidet SUBs, FUNCTIONs und GOTO/GOSUB-Routinen. Letztere sollten aber nach Möglichkeit vermieden werden, da der Code durch sie leicht unübersichtlich werden kann ("Spaghetti-Code").

Diese Kapitel stehen zur Verfügung:

Letzte Bearbeitung des Eintrags am 12.01.13 um 23:21:42
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

SUBs

FreeBASIC-Referenz » Prozeduren in FreeBASIC » **SUBs**
SUBs sind die einfachere Version einer Prozedur. Sie bestehen aus dem Prozedurheader, dem Programmcode und der abschließenden Zeile END SUB. Der Prozedurheader enthält die Liste der übergebenen Parameter (siehe Parameterübergabe). Die im Header übergebenen Variablen werden wie bereits dimensionierte Variablen behandelt; sie können innerhalb der SUB nicht redimensioniert werden. Eine Ausnahme bilden dynamische Arrays. Sie können zwar redimensioniert werden, ihr Datentyp (INTEGER, STRING, ...) bleibt aber gleich. Siehe SUB für weitere Details zum Prozedurheader.

Der Programmcode folgt denselben Regeln wie im Hauptmodul. Er darf fast alle Anweisungen enthalten und kann auch andere Prozeduren oder sogar sich selbst aufrufen (rekursiver Aufruf). Nicht verwendet werden dürfen die sogenannten 'nicht ausführbaren Anweisungen' wie DECLARE, COMMON usw. Auch Präprozessoren sind auf Prozedurebene erlaubt. Variablen, die auf Modulebene deklariert wurden, stehen in der Regel innerhalb der Prozedur nicht zur Verfügung; eine Ausnahme bilden die über SHARED, COMMON SHARED oder EXTERN deklarierten Variablen. Beachten Sie dazu den Gültigkeitsbereich von Variablen.

Die Zeile END SUB markiert das Ende der Prozedur. Code, der hinter der Zeile END SUB steht, wird wie Modulebenen-Code behandelt.

Bevor eine Prozedur verwendet werden kann, muss sie im Programm bekannt sein. Soll eine Prozedur verwendet werden, bevor sie im Quellcode definiert wurde, dann muss sie dem Programm vor der Verwendung mit DECLARE bekannt gemacht werden. CONSTRUCTOR- und DESTRUCTOR-SUBs die tatsächlich nur zu Beginn des Programms bzw. bei dessen Beendigung automatisch aufgerufen werden sollen, müssen nicht deklariert werden.

Eine SUB wird direkt über ihren Bezeichner aufgerufen. Die Parameterliste kann beim Aufruf in Klammern stehen oder ohne Klammern angegeben werden.

Beispiel:

```
DECLARE SUB colortext (txt AS STRING, clr AS INTEGER)

colortext "blue", 1   ' oder mit Klammern: colortext("blue", 1)

'-----------------'

SUB colortext (txt AS STRING, clr AS INTEGER)
  "hlzahl">10
  COLOR clr
  PRINT txt
END SUB

'-----------------'

FOR i AS INTEGER = 2 TO 15
  colortext "COLOR " & STR(i), i
NEXT
PRINT a    ' Fehler: a ist auf Modulebene nicht definiert
SLEEP
```

Siehe auch: FUNCTIONs

Letzte Bearbeitung des Eintrags am 13.01.13 um 00:59:02
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

FUNCTIONs

FreeBASIC-Referenz » Prozeduren in FreeBASIC » **FUNCTIONs**
FUNCTIONs sind ebenso wie SUBs Prozeduren. Die dort aufgeführten Informationen gelten weitgehend auch für FUNCTIONs. Im Gegensatz zu einer SUB gibt eine FUNCTION jedoch einen Wert zurück.

FUNCTIONs bestehen aus dem Prozedurheader, dem Programmcode und der Zeile END FUNCTION. Der Prozedurheader und der Programmcode folgen denselben Regeln wie bei SUBs. Siehe FUNCTION für weitere Details zum Prozedurheader.

Im Programmcode können andere FUNCTIONs und SUBs aufgerufen werden. Es ist auch möglich, die FUNCTION selbst aufzurufen. Solche sogenannten rekursiven FUNCTIONs können manche Probleme auf sehr elegante Weise lösen, sollten jedoch mit Bedacht eingesetzt werden, da sie oft schwerer zu warten sind.

FUNCTIONs werden im Programmcode wie Variablen behandelt, die abhängig von ihren Parametern verschiedene Werte annehmen. Sie werden durch ihren Bezeichner aufgerufen, die Parameterliste muss in Klammern übergeben werden. Der Wert einer FUNCTION kann auf dreierlei Weise gesetzt werden:

- über ihren Bezeichner: Funktionsname = Ausdruck
- über das Symbol "FUNCTION": FUNCTION = Ausdruck
- über RETURN: RETURN Ausdruck
  Achtung: RETURN funktioniert wie FUNCTION=Ausdruck: EXIT FUNCTION. Die Funktion wird also mit diesem Befehl verlassen.

Beispiel:

```
DECLARE FUNCTION twice (x AS UINTEGER) AS INTEGER

FUNCTION twice (x AS INTEGER) AS INTEGER
  twice = x * 2
END FUNCTION

' gibt 20 aus, das Doppelte von 10:
PRINT twice(10)
SLEEP
```

Eine rekursive FUNCTION:

```
FUNCTION Fakultaet (x AS UINTEGER) AS DOUBLE
  IF x = 0 THEN
    RETURN 1
  ELSE
    RETURN x * Fakultaet(x-1)
  END IF
END FUNCTION

PRINT Fakultaet(6)
SLEEP
```

Als Rückgabewert wurde ein DOUBLE gewählt, weil damit wesentlich größere Fakultäten berechnet werden können als mit Ganzzahl-Typen (allerdings wird das Ergebnis ab 19! gerundet). Rekursive Funktionen haben den Nachteil, dass der benötigte Speicherplatz von der Tiefe der Verschachtelung abhängig ist. Im obigen Beispiel liegen die Ergebnisse aber schon lange außerhalb des Speicherbereichs eines DOUBLEs, bevor der Programmspeicherplatz knapp wird.

Letzte Bearbeitung des Eintrags am 12.01.13 um 23:26:06
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

Parameterübergabe

FreeBASIC-Referenz » Prozeduren in FreeBASIC » **Parameterübergabe**
Prozeduren können i. d. R. nicht auf den Speicher des Hauptprogramms zugreifen. Wird im Hauptprogramm eine Variable mit dem Bezeichner 'a' verwendet, ist ihr Wert im Unterprogramm nicht verfügbar, selbst wenn in diesem auch eine Variable mit dem Bezeichner 'a' verwendet wird. Um einer Prozedur Variablen zu übergeben, gibt es zwei Möglichkeiten: die direkte Übergabe in der Parameterliste der Prozedur oder die Globalisierung der Variable.

Direkte Übergabe in der Parameterliste der Prozedur
Jedes Unterprogramm muss zunächst deklariert werden; dazu verwendet man DECLARE. Die DECLARE-Anweisung enthält auch eine Parameterliste; dies ist eine Auflistung der Variablen, die an das Unterprogramm übergeben werden sollen. Dabei ist der verwendete Bezeichner nebensächlich; er kann in der Prozedur selbst unter einem ganz anderen Namen übergeben werden. Wichtig ist aber der Datentyp!

Beispiel:

```
DECLARE SUB foo (a AS BYTE, b AS STRING, c AS INTEGER)
DECLARE FUNCTION bar (a AS BYTE, b AS STRING, c AS INTEGER) AS INTEGER
```

Hier werden zwei Prozeduren definiert. Bei beiden ist der erste Parameter vom Typ BYTE, der zweite vom Typ STRING und der dritte vom Typ INTEGER. Der Rückgabetyp der FUNCTION ist Integer.
Beim Aufruf der Prozeduren können dennoch andere Bezeichner für die Variablen verwendet werden:

```
DECLARE SUB foo (a AS BYTE, b AS STRING, c AS INTEGER)
DECLARE FUNCTION bar (a AS BYTE, b AS STRING, c AS INTEGER) AS INTEGER

DIM a AS STRING, b AS INTEGER
DIM c AS BYTE, d AS INTEGER

foo c, a, b
d = bar(c, a, b)
```

Jede Prozedur hat einen "Header". Dieser enthält ähnlich wie die DECLARE-Anweisung die Parameterliste. Im Gegensatz zur DECLARE-Zeile sind die Bezeichner der Parameter hier aber von Bedeutung:

```
DECLARE SUB foo (a AS BYTE, b AS STRING, c AS INTEGER)
DECLARE FUNCTION bar (a AS BYTE, b AS STRING, c AS INTEGER) AS INTEGER

DIM a AS STRING, b AS INTEGER
DIM c AS BYTE, d AS INTEGER

foo c, a, b
d = bar(c, a, b)
SLEEP

'---------------'

SUB foo (d AS BYTE, e AS STRING, f AS INTEGER)
  PRINT e, f + d
END SUB

FUNCTION bar (e AS BYTE, z AS STRING, a AS INTEGER) AS INTEGER
  PRINT z
```

```
  bar = e * a
END FUNCTION
```

Die Werte von a, b und c werden in der Reihenfolge übergeben, in der sie beim Aufruf aufgeführt sind. Innerhalb der Prozedur erhalten sie dann einen neuen Namen. In der SUB foo wird c als d, a als e und b als f bezeichnet. In der FUNCTION bar wird c als e, a als z und b als a bezeichnet.

Bei dieser Art der Übergabe unterscheidet man weiter in BYVAL- und BYREF-Übergabe. Werden die Variablen BYREF übergeben, können die Prozeduren den Wert der Variablen so verändern, dass er auch im aufrufenden Programmteil geändert wird. Variablen, die BYVAL übergeben werden, erhalten eine eigene Speicherstelle in der Prozedur. Auch wenn der Wert der übergebenen Variablen innerhalb der Prozedur verändert wird, bleibt er in der aufrufenden Prozedur unverändert. Auf welche Weise die Variablen übergeben werden sollen, steht im Prozedurheader und in der DECLARE-Zeile; vor dem jeweiligen Parameter wird ein BYREF bzw. BYVAL angegeben.

Anmerkung:
Bis einschließlich FreeBASIC v0.16 wurden Parameter an Prozeduren standardmäßig BYREF übergeben; seit FreeBASIC v0.17 ist dies nur noch der Fall, wenn mit der Kommandozeilenoption -lang deprecated oder -lang qb compiliert wird. Andernfalls geht FreeBASIC ab Version v0.17 vom Standard BYVAL aus.

Beispiel:

```
DECLARE SUB foobar (BYREF a, BYVAL b)

a = 10
b = 5

PRINT "Vor dem Prozeduraufruf:"
PRINT a, b
PRINT

foobar a, b
PRINT

PRINT "Nach dem Prozeduraufruf:"
PRINT a, b
SLEEP

SUB foobar (BYREF a, BYVAL b)
  PRINT "Beginn der Prozedur:"
  PRINT a, b
  PRINT

  a = 20
  b = 40

  PRINT "Ende der Prozedur:"
  PRINT a, b
END SUB
```

Ausgabe:

```
Vor dem Prozeduraufruf:
 10    5
```

```
Beginn der Prozedur:
 10    5

Ende der Prozedur:
 20    40

Nach dem Prozeduraufruf:
 20    5
```

Beim Aufruf von Prozeduren können Parameter ausgelassen werden, wenn in der DECLARE-Anweisung für diesen Parameter eine Konstante angegeben wird. Die DECLARE-Anweisung enthält dann, neben der Parameterliste, die Zuweisung der Konstanten, die nur dann benutzt werden, wenn im Aufruf an dieser Stelle kein Wert übergeben wird. Dabei ist wichtig, dass die verwendeten Bezeichner in der DECLARE-Anweisung und im Prozedur-Header gleich sind. Die Zuweisung der Konstanten muss im Prozedur-Header nicht wiederholt werden. Diese optionalen Parameter können von jedem Datentyp sein; auch STRINGs sind erlaubt. Lediglich UDTs und Arrays unterstützen diese Technik bislang nicht.

Beispiel:

```
DECLARE SUB plus (BYREF a AS INTEGER,  BYVAL i AS INTEGER = 1)
DECLARE FUNCTION minus (BYREF a AS INTEGER, BYVAL i AS INTEGER = 1) AS
INTEGER

DIM x AS INTEGER

x = 6
' Konstante benutzen
plus x
PRINT x, minus(x)

' angegebenen Werte benutzen
plus x, 15
PRINT x, minus(x, 8)
SLEEP

SUB plus (BYREF a AS INTEGER, BYVAL i  AS INTEGER)
  a += i
END SUB

FUNCTION minus (BYREF a AS INTEGER, BYVAL i AS INTEGER) AS INTEGER
  minus = a - i
END FUNCTION
```

Ausgabe:

```
 7             6
 22            14
```

FreeBASIC selbst macht von dieser Möglichkeit Gebrauch, z. B. bei

```
MID(Text AS STRING, Start &"cnf">GET #Dateinummer, [Position], Variable
hier können Sie z. B.
GET #1, , x
```

schreiben und erhalten dann, von der aktuellen Position der geöffneten Datei, einen Wert in x eingelesen.

Arrays
Function und Sub können neben Variablen auch Arrays als Parameter verwenden:

```
' SUB mit einem Integer-Array als Parameter
Declare Sub unsicher (array() As Integer)
Declare Sub sicher (array() As Integer)

' zweidimenionales Array mit 8 * 5 Feldern (beginnend ab 0!)
Dim As Integer meinArray(7, 4)

' Beispielwerte
meinArray(2, 2) = 2
meinArray(4, 4) = 4

' Aufruf der Sub - beim Array sind die Klammern () nötig um Arrayzugriff
zu signalisieren
unsicher(meinArray())

Print "----"

' Aufruf der sicheren Sub, bei der auf den Zugriff geachtet wird
sicher(meinArray())

Sleep ' auf Tastendruck warten

Sub unsicher (array() As Integer)
  ' Mit UBOUND lässt sich die Obergrenze ermitteln
  ' Problem dabei: es wird nur die Obergrenze der ersten Dimension
abgefragt
  For i As Integer = 0 To UBound(array)
    Print array(i,i) '*
  Next

  ' * Hier ist Vorsicht geboten, da ein Arrayzugriff außerhalb des
  '   gültigen Bereichs unvorhersagbare Werte ergeben und sogar bis
  '   zum Programmansturz führen kann!
End Sub

Sub sicher (array() As Integer)
  ' Um Probleme mit der Obergrenze zu umgehen, wird auch die zweite
Dimension abgefragt
  For i As Integer = 0 To UBound(array)
    If i <= UBound(array, 2) Then Print array(i,i) '*
  Next
End Sub
```

TYPES (UDTs)
Auch die Übergabe von UDTs ist möglich:

```
Type myType
  x As Integer
  y As Integer
End Type

Declare Sub changeMyVar(anyVar As myType)

Dim As myType myVar

myVar.x = 5
myVar.y = 7

Print "Vorher:"
Print myVar.x
Print myVar.y

changeMyVar(myVar)

Print
Print "Nachher:"
Print myVar.x
Print myVar.y
Sleep

' UDTs werden automatisch BYREF übergeben
' Durch explizite Angabe von BYVAL kann dies aber auch verhindert werden
Sub changeMyVar(anyVar As myType)
  anyVar.x += 1
  anyVar.y -= 1
End Sub
```

Einen Parameter, der ein Array ist und vom Typ eines eigenen UDTs ist natürlich auch möglich.

Sub/Function Pointer
Neben den bisher beschriebenen, können auch Pointer auf Subs und Functions übergeben werden. Folgendes Beispiel zeigt die Verwendung auf:

```
Sub eineSub(parameter As Integer)
    Print "eineSub", parameter
End Sub

Function eineFunction(parameter As Integer) As Integer
    Print "eineFunction", parameter
    Return 0
End Function

Sub foo(parameter As Sub (parameter As Integer))
    parameter(1)
End Sub
Sub bar(parameter As Function (parameter As Integer) As Integer)
    Print parameter(2)
End Sub

foo(@eineSub)
```

```
bar(@eineFunction)

Sleep
```

Ausgabe:

```
eineSub         1
eineFunction    2
 0
```

Globalisierung
Manche Variablen werden in (fast) jedem Unterprogramm benötigt. Es wäre sehr umständlich, diese Variablen bei jedem Aufruf der Prozedur in die Parameterliste einzugeben. Stattdessen ist es möglich, einzelne Variablen allen Prozeduren zugänglich zu machen; sie werden dann so behandelt, als wären sie jedem Unterprogramm BYREF übergeben worden. Möglich ist das mit dem Schlüsselwort SHARED. Bei ihrer ersten Definition mit DIM oder COMMON muss nur SHARED angegeben werden, um diesen Effekt zu erzielen. Zu weiteren Details siehe SHARED und Gültigkeitsbereich von Variablen.

Letzte Bearbeitung des Eintrags am 17.01.14 um 22:53:02
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

GOTO und GOSUB

FreeBASIC-Referenz » Prozeduren in FreeBASIC » **GOTO und GOSUB**
Mit den beiden Befehlen GOTO und GOSUB wurden in älteren BASIC-Dialekten Prozeduren erstellt. Es ist theoretisch möglich, einer GOTO/GOSUB-Prozedur Parameter zu übergeben. Dazu müssten (zusätzliche) Variablen deklariert werden, die dann nur in der Prozedur benutzt werden. Auch durch diesen erhöhten Programmieraufwand werden die Unterprogramme leicht unübersichtlich. GOTO und GOSUB sind daher unter Programmierern nicht sonderlich gerne gesehen; der Vollständigkeit halber seien sie jedoch dennoch hier aufgeführt.

GOTO
GOTO "springt" an einen bestimmten Punkt innerhalb des Programms, der sowohl vor als auch hinter der GOTO-Anweisung liegen kann. Die Ausführung des Programms wird dann von dieser Stelle ab fortgesetzt. Der Einsprungpunkt wird durch ein sogenanntes Label markiert. Ein Label ist ein beliebiger Bezeichner, gefolgt von einem Doppelpunkt. In den Dialektformen -lang qb, -lang deprecated und -lang fblite kann als Label auch eine Zahl verwendet werden.

```
"hlstring">"qb"
10 PRINT "hello World"
MyLabel:
PRINT "eine zweite Anweisung"
```

Das Programm merkt sich nicht, von welcher Stelle aus der Sprung durchgeführt wurde. Soll später wieder zurückgesprungen werden, dann muss dieser Rücksprung "manuell" durchgeführt werden, d. h. eine weitere GOTO-Zeile muss das Programm wieder dorthin zurückspringen lassen, von wo es aufgerufen wurde.

Beispiel:

```
z1: GOTO z4
z2: PRINT "Ciao"
z3: GOTO ende
z4: PRINT "Welcome!"
z5: GOTO z2

ende:
END
```

Das obige Beispiel ist nicht besonders sinnvoll, sondern soll lediglich das Verhalten des Befehls GOTO demonstrieren. Eingesetzt werden kann GOTO z. B. dann, wenn bestimmte Programmteile unter gewissen Voraussetzungen übersprungen werden sollen oder wenn umgekehrt ein bereits ausgeführter Programmteil nochmal durchlaufen werden soll. Mit GOTO ist es jedoch sehr leicht, sogenannten Spaghetticode zu erzeugen, der sich nur noch schwer warten lässt. Daher sollte man in der Regel versuchen, den Befehl zu vermeiden.

GOTO kann fast immer durch andere Programmstrukturen ersetzt werden. Mit Bedingungsstrukturen wie IF ... THEN und SELECT CASE können auch große Programmteile unter bestimmten Bedingungen übersprungen werden, während Schleifen die mehrfache Ausführung eines Programmabschnitts ermöglichen.

GOSUB
GOSUB springt ähnlich wie GOTO zu einem Label im Programm, merkt sich jedoch die Stelle, von der aus das GOSUB durchgeführt wurde. Sobald das Programm auf die Anweisung RETURN stößt, führt es einen Rücksprung aus. Der Befehl arbeitet damit sehr ähnlich wie eine Prozedur, nur dass eventuell benötigte Parameter "manuell" übergeben werden müssen. Da GOSUB fast vollständig durch Prozeduren ersetzt

werden kann, steht es in der Dialektform -lang fb nicht mehr zur Verfügung.

Beispiel:

```
"hlstring">"qb"
GOSUB message
END

message:
PRINT "Welcome!"
RETURN
```

Das Programm benötigt für jeden GOSUB-Aufruf Speicherplatz, um sich die Rücksprungadresse zu merken. Dieser Speicherplatz wird nach dem RETURN wieder freigegeben. Beenden Sie daher einen mit GOSUB ausgeführten Sprung immer mit RETURN, um einen Speicherüberlauf zu vermeiden.

Letzte Bearbeitung des Eintrags am 12.01.13 um 23:32:22
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

Geschwindigkeit und Größe

Geschwindigkeit: Die interne Gfxlib benutzt den normalen RAM als Seitenspeicher. Alle Lese-/Schreibzugriffe auf den Seitenpuffer (inklusive Direktzugriffe via SCREENPTR) werden im RAM ausgeführt. Bei Win32 und Linux übernimmt ein eigener Thread die Screen-Updates, und verwendet dabei eine separat wählbare Bildschirmwiederholrate (wenn keine Rate angegeben wird, ermittelt FreeBASIC eine Standardrate; siehe dazu SCREENRES). Die Farbkonversion wird, falls nötig, ebenfalls übernommen. Obwohl dies überflüssig für Vollbild-Modi scheint, ist es nötig für Fenstertechniken. Um die Bibliothek klein und einfach zu halten, wurde dieselbe Methode auch für das Vollbild benutzt, auf Kosten einer kleinen 'Zeitstrafe' wegen der Speicherkopie. MMX wird jedoch, wo immer möglich, in der gesamten Gfxlib benutzt, womit immer noch eine hohe Geschwindigkeit der Bibliothek erreicht wird.

Größe: Eines der Hauptziele der Gfxlib war es, unabhängig von externen Bilbiotheken zu sein. Die Bilbiothek hat ihre eigenen Treiber (jeder 10-15 KB) und ist komplett unabhängig. Natürlich müssen in Ihre Programme auch andere Codes eingebunden werden, zumindest zur Grafikinitialisierung, Schriftart-Dateien, Unterstützung für alle Farbtiefen, Standardpaletten und Ähnliches. Das bedeutet, dass die Benutzung der Gfxlib Ihre Programme um 40 bis 90 KB vergrößern wird, abhängig von den Funktionen, die Sie verwenden (fbc wird nur das linken, was Sie wirklich benutzen). Dafür sind Ihre Programme Standalones, also von Runtime-DLLs unabhängig. Natürlich bedeutet Standalone nicht, dass überhaupt keine Abhängigkeiten bestehen; es handelt sich allerdings nur Abhängigkeiten von den Systembibliotheken, von denen Sie annehmen können, dass sie immer vorhanden sind.

Folgende Daten werden zusätzlich geladen, wenn Sie die Gfxlib benutzen:

Win32	Linux
user32.dll (ddraw.dll und dinput.dll nur zur Laufzeit verwendet)	libX11, libXext, libXxf86vm, libpthread

Letzte Bearbeitung des Eintrags am 13.01.13 um 16:09:23
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

Standard-Paletten

Beim Öffnen eines Ausgabefensters (eines Konsolenfensters oder eines Grafikfensters) sowie bei der Reinitialisierung eines Bildschirmmodus via SCREEN (Anweisung) werden die folgenden Farben ihren Indizes zugeordnet. In Grafikmodi können diese später mittels PALETTE geändert werden; im Textmodus ist dies nicht möglich.

Modus	Index	Farbe
1	0	Schwarz
	1	Zyan
	2	Magenta
	3	Weiß
2, 10 und 11	0	Schwarz
	1	Weiß
7, 8, 9, 12 und Konsole	0	Schwarz
	1	Blau
	2	Grün
	3	Zyan
	4	Rot
	5	Magenta
	6	Braun
	7	Hellgrau
	8	Dunkelgrau
	9	Hellblau
	10	Hellgrün
	11	Hellzyan
	12	Hellrot
	13	Pink
	14	Gelb
	15	Weiß

Hinweis: Im SCREEN 1 ist die Farbzuordnung anders als bei den anderen Bildschirmmodi. Vergleiche dazu COLOR (Anweisung).

Die Modi 13 und aufwärts mit der standardmäßigen Farbtiefe von 8bpp haben einen besonderen Aufbau nach "Farbbändern". Ein Farbband ist ein Bereich von Indizes, innerhalb dessen nur Farben mit bestimmten Eigenschaften vorkommen.

Farbband	Indizes	Inhalt
Konsolenband	0 - 15	Dieselbe Farbzuordnung wie bei den Modi 7, 8, 9, 12 und im Konsolenmodus
graues Band	16 - 31	Verschiedene Grautöne, beginnend bei schwarz und endend bei weiß
Band hoher Sättigung und hoher Helligkeit	32	blau
	36	magenta
	40	rot
	44	gelb
	48	grün
	52	cyan
Band hoher Sättigung und mittlerer Helligkeit	56	blau
	60	magenta
	64	rot

	68	gelb
	72	grün
	76	cyan
	80	blau
	84	magenta
Band hoher Sättigung und niedriger Helligkeit	88	rot
	92	gelb
	96	grün
	100	cyan
	104	blau
	108	magenta
Band mittlerer Sättigung und hoher Helligkeit	112	rot
	116	gelb
	120	grün
	124	cyan
	128	blau
	132	magenta
Band mittlerer Sättigung und mittlerer Helligkeit	136	rot
	140	gelb
	144	grün
	148	cyan
	152	blau
	156	magenta
Band mittlerer Sättigung und niedriger Helligkeit	160	rot
	164	gelb
	168	grün
	172	cyan
	176	blau
	180	magenta
Band niedriger Sättigung und hoher Helligkeit	184	rot
	188	gelb
	192	grün
	196	cyan
	200	blau
	204	magenta
Band niedriger Sättigung und mittlerer Helligkeit	208	rot
	212	gelb
	216	grün
	220	cyan
	224	blau
	228	magenta
Band niedriger Sättigung und niedriger Helligkeit	232	rot
	236	gelb
	240	grün
	244	cyan
schwarzes Band	248 - 255	schwarz (Meist für Animationen genutzt - Siehe PALETTE)

Abgesehen vom ersten und zweiten Band gilt für diese Modi also:

- Innerhalb eines Farbbands sind Helligkeit und Sättigung der Farbe jeweils dieselbe.
- Jedes Farbband enthält 24 verschiedene Farbtöne, die bei blau beginnen, das Spektrum durchwandern und schließlich wieder bei blau enden.

Die Farbreihenfolge ist:

- Blau, Übergang über vier Schritte nach
- Magenta, Übergang über vier Schritte nach
- Rot, Übergang über vier Schritte nach
- Gelb, Übergang über vier Schritte nach
- Grün, Übergang über vier Schritte nach
- Cyan Übergang über vier Schritte nach Blau des nächsten Bandes.

Dieses Programm erstellt eine gute Ansicht der verfügbaren Farben:

```
SCREENRES 820, 240
DIM AS INTEGER i

DRAW STRING (490,4),"Konsolenband"
FOR i = 0 TO 15
  LINE (i*20,0)-((i+1)*20-1,19),i,BF
NEXT

DRAW STRING (490,24),"Graues Band"
FOR i =16 TO 31
  LINE ((i-16)*20,20)-((i-15)*20-1,39),i,BF
NEXT

DRAW STRING (490,44),"hohe Helligkeit,hohe Saettigung"
FOR i = 32 TO 55
  LINE ((i-32)*20,40)-((i-31)*20-1,59),i,BF
NEXT

DRAW STRING (490,64),"hohe Helligkeit,mittlere Saettigung"
FOR i = 56 TO 79
  LINE ((i-56)*20,60)-((i-55)*20-1,79),i,BF
NEXT

DRAW STRING (490,84),"hohe Helligkeit,niedrige Saettigung"
FOR i = 80 TO 103
  LINE ((i-80)*20,80)-((i-79)*20-1,99),i,BF
NEXT

DRAW STRING (490,104),"mittlere Helligkeit,hohe Saettigung"
FOR i = 104 TO 127
  LINE ((i-104)*20,100)-((i-103)*20-1,119),i,BF
NEXT

DRAW STRING (490,124),"mittlere Helligkeit,mittlere Saettigung"
FOR i = 128 TO 151
  LINE ((i-128)*20,120)-((i-127)*20-1,139),i,BF
NEXT

DRAW STRING (490,144),"mittlere Helligkeit,niedrige Saettigung"
FOR i = 152 TO 175
  LINE ((i-152)*20,140)-((i-151)*20-1,159),i,BF
NEXT
```

```
DRAW STRING (490,164),"niedrige Helligkeit,hohe Saettigung"
FOR i = 176 TO 199
  LINE ((i-176)*20,160)-((i-175)*20-1,179),i,BF
NEXT

DRAW STRING (490,184),"niedrige Helligkeit,mittlere Saettigung"
FOR i = 200 TO 223
  LINE ((i-200)*20,180)-((i-199)*20-1,199),i,BF
NEXT

DRAW STRING (490,204),"niedrige Helligkeit,niedrige Saettigung"
FOR i = 224 TO 247
  LINE ((i-224)*20,200)-((i-223)*20-1,219),i,BF
NEXT

DRAW STRING (490,224),"schwarzes Band"
FOR i = 248 TO 255
  LINE ((i-248)*20,220)-((i-247)*20-1,239),i,BF
NEXT

SLEEP
```

Letzte Bearbeitung des Eintrags am 19.01.13 um 02:20:51
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

Keyboard Scancodes

FreeBASIC-Referenz » Die Gfxlib » **Keyboard Scancodes**
Die nachfolgende Liste enthält die Scancodes, die bei MULTIKEY verwendet werden. Sie entsprechen den DOS-Scancodes und funktionieren garantiert auf jeder Plattform. Sie finden diese Liste ebenfalls in der Datei fbgfx.bi, die sich in Ihrem inc-Verzeichnis befinden sollte.

```
"hlzahl">&h01
"hlzahl">1              &h02
"hlzahl">2              &h03
"hlzahl">3              &h04
"hlzahl">4              &h05
"hlzahl">5              &h06
"hlzahl">6              &h07
"hlzahl">7              &h08
"hlzahl">8              &h09
"hlzahl">9              &h0A
"hlzahl">0              &h0B
"hlzahl">&h0C
"hlzahl">&h0D
"hlzahl">&h0E
"hlzahl">&h0F
"hlzahl">&h10
"hlzahl">&h11
"hlzahl">&h12
"hlzahl">&h13
"hlzahl">&h14
"hlzahl">&h15
"hlzahl">&h16
"hlzahl">&h17
"hlzahl">&h18
"hlzahl">&h19
"hlzahl">&h1A
"hlzahl">&h1B
"hlzahl">&h1C
"hlzahl">&h1D
"hlzahl">&h1E
"hlzahl">&h1F
"hlzahl">&h20
"hlzahl">&h21
"hlzahl">&h22
"hlzahl">&h23
"hlzahl">&h24
"hlzahl">&h25
"hlzahl">&h26
"hlzahl">&h27
"hlzahl">&h28
"hlzahl">&h29
"hlzahl">&h2A
"hlzahl">&h2B
"hlzahl">&h2C
"hlzahl">&h2D
"hlzahl">&h2E
```

```
"hlzahl">&h2F
"hlzahl">&h30
"hlzahl">&h31
"hlzahl">&h32
"hlzahl">&h33
"hlzahl">&h34
"hlzahl">&h35
"hlzahl">&h36
"hlzahl">&h37
"hlzahl">&h38
"hlzahl">&h39
"hlzahl">&h3A
"hlzahl">&h3B
"hlzahl">&h3C
"hlzahl">&h3D
"hlzahl">&h3E
"hlzahl">&h3F
"hlzahl">&h40
"hlzahl">&h41
"hlzahl">&h42
"hlzahl">&h43
"hlzahl">&h44
"hlzahl">&h45
"hlzahl">&h46
"hlzahl">&h47
"hlzahl">&h48
"hlzahl">&h49
"hlzahl">&h4B
"hlzahl">&h4D
"hlzahl">&h4E
"hlzahl">&h4F
"hlzahl">&h50
"hlzahl">&h51
"hlzahl">&h52
"hlzahl">&h53
"hlzahl">&h57
"hlzahl">&h58

' erweiterte Scancodes, unter DOS ab FreeBASIC v0.15 verfügbar:

"hlzahl">&h5B
"hlzahl">&h5C
"hlzahl">&h5D
```

Letzte Bearbeitung des Eintrags am 19.01.13 um 02:14:30
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

Interne Treiber

Wie bereits angesprochen benutzt die Gfxlib ihre eigenen Treiber, um mit verschiedenen Systemen arbeiten zu können.

Windows:

- **DirectX**®: Dieser Treiber basiert auf DirectDraw® und DirectInput®; er wird auf jedem System funktionieren, auf dem zumindest DirectX 3.0 installiert wurde. Windows 98 ist dabei die Untergrenze. Wenn Sie Windows 95 benutzen, wird nicht garantiert, dass er funktioniert. Die nötigen DirectX-DLLs werden bei der Ausführung geladen, so dass Ihre EXE davon unabhängig bleibt. Ihre Programme bleiben auch auf Systemen lauffähig, die kein DX installiert haben (in diesem Fall scheitert die Initiierung des Grafikmodus mit DX). Der Treiber unterstützt Fenster- und Vollbildmodi.
- **GDI**: Der einfache Windows GDI-Treiber ist langsamer als DX, aber garantiert immer lauffähig. Wenn die Initiierung eines Modus mit DX scheiterte, wird automatisch GDI verwendet. Der Treiber unterstützt nur Fenstermodi.

Linux:

- **X11**: Der auf Raw Xlib basierende Treiber funktioniert immer. Wenn Ihr Programm läuft, wird die XSHM-Erweiterung verwendet, um schnellere Framebufferzugriffe via shared memory zu ermöglichen. Um Vollbildmodi zu benutzen, wird die Xvidmode-Erweiterung verwendet; Ihr Programm muss lokal ausgeführt werden und die entsprechenden modelines sollten von Ihrem X11-Server unterstützt werden. Ansonsten ist kein Vollbildmodus verfügbar.

Die Gfxlib verwendet immer den am besten geeigneten Treiber auf Ihrem System, wenn ein neuer Videomodus per SCREENRES initiiert werden soll. Um diese Wahl zu überbrücken, können Sie vor der Initialisierung des Grafikfensters den Befehl SCREENCONTROL einsetzen:

```
"hlstring">"fbgfx.bi"
SCREENCONTROL FB.SET_DRIVER_NAME, "GDI"
SCREENRES 400, 300
' ...
```

Sie können stattdessen auch die Umgebungsvariable 'FBGFX' setzen, um den Treiber zu benennen, den Sie bevorzugt verwenden wollen. Unter Windows kann das z. B. mit der folgenden Anweisung in der Konsole erledigt werden:

```
SET FBGFX=gdi
```

oder direkt im Programmcode durch

```
SetEnviron("fbgfx=GDI")
```

Dadurch wird die Gfxlib versuchen, einen Bildschirmmodus zuerst mit dem Treiber "GDI" zu initialisieren; wenn der angegebene Treiber nicht gefunden wird oder die Aufgabe nicht bewältigen kann, fährt die Gfxlib in der automatischen Reihenfolge fort.

Letzte Bearbeitung des Eintrags am 19.01.13 um 02:12:31
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

Interne Pixelformate

FreeBASIC-Referenz » Die Gfxlib » **Interne Pixelformate**
Die Darstellung von Grafiken im Speicher mit FreeBASIC lässt sich in mehrere Kapitel untergliedern:

- Darstellung eines einzelnen Pixels
- Transparenz - Maskenfarben und Alpha-Wert
- Struktur eines Bildpuffers für die Drawing Primitives

Darstellung eines einzelnen Pixels

Die Gfxlib verwendet immer eines von drei Pixelformaten: indizierte 1-Byte-Pixel, Direct Color 2-Bytes-Pixel (Hicolor) und Direct Color 4-Bytes-Pixel (Truecolor). Diese Formate werden in den entsprechenden Modi verwendet:

Farbtiefe (bpp)	Pixelelformat
1	1 Byte pro Pixel, indiziert, Index von 0 bis 1.
2	1 Byte pro Pixel, indiziert, Index von 0 bis 3.
4	1 Byte pro Pixel, indiziert, Index von 0 bis 15.
8	1 Byte pro Pixel, indiziert, Index von 0 bis 255.
15, 16	2 Bytes pro Pixel, Direct Color, Format &bRRRRRGGGGGGBBBBB.
24, 32	4 Bytes pro Pixel, Direct Color, Format &bAAAAAAAARRRRRRRRGGGGGGGGBBBBBBBB.

Indiziert bedeutet hierbei, dass jeder Zahl eine Farbe zugeordnet ist; beispielsweise kann ein Pixel mit dem Farbattribut 4 rot sein. Welche Farbe welcher Zahl zugeordnet ist, hängt vom Bildschirmmodus ab; siehe dazu die Standard-Paletten. Diese können mit PALETTE noch im Programmverlauf bearbeitet werden. Bei Direct-Color-Modi stellt die Zahl bereits die Farbe dar. Jeweils ein bestimmter Teil der Zahl stellt den Rot-Grün- und Blau-Anteil der Farbe dar. Bei der Angabe des Formats in obiger Tabelle steht jeweils ein Zeichen für ein Bit der Farbinformation. R symbolisiert jeweils den Rot-, G den Grün- und B den Blauanteil. Die Stellen A, die bei 24bpp und 32bpp aufgeführt sind, können tatsächlich nur in 32bpp-Modi genutzt werden. Sie stellen die Alpha- oder Transparenz-Komponente dar. In 24bpp-Modi sind diese Stellen jeweils mit null besetzt, sie werden lediglich zur leichteren Verwaltbarkeit beibehalten. Wie Sie sehen, verwenden der 15bpp- und der 16bpp-Modus dasselbe 16bpp-Format. Ähnliches gilt den 24bpp- und den 32bpp-Modus sagen: beide benutzen das 32bpp-Format. Das interne Format ist plattformunabhängig. Der Bildschirmspeicher wird in einem der drei Formate gehalten. Denken Sie daran, wenn Sie per SCREENPTR darauf zugreifen; dasselbe gilt für Daten, die in GET- und PUT-Feldern gespeichert sind. Mehr dazu unter Struktur eines Bildpuffers für die Drawing Primitives. In Modi mit mehr als einem Byte pro Pixel werden die Bytes in umgekehrter Reihenfolge im Speicher abgelegt; das obere Byte des Gesamtwertes ist also das zweite im Speicher, das untere Byte wird zuerst abgelegt. Befindet sich an der angenommenen Adresse 0 also eine 32bit-Pixelinformation, so werden ihre Komponenten folgendermaßen zerlegt:

Byte	Komponente
0	Blau
1	Grün
2	Rot
3	Alpha

Transparenz - Maskenfarben und Alpha-Wert

Maskenfarbe

Die Maskenfarbe hängt von der aktuellen Farbtiefe ab; sie ist einer von diesen Werten:

Farbtiefe	Maskenfarbnummer
1, 2, 4, 8	0
15, 16	&hF81F
24, 32	&hFF00FF

Indizierte Modi benutzen immer den Index 0 als Transparenzmaske, während Direct Color-Modi immer Pink verwenden.

Alphawert
FreeBASIC unterstützt Transparenzeffekte nach der ALPHA-Methode. Dabei wird neben dem Farb-Tripplet (Rot, Grün, Blau) auch ein Transparenzgrad, der Alphawert angegeben. Ebenso wie die Werte des Farbtripplets liegt auch der Alphawert im Bereich von 0 bis 255; dabei entspricht Alpha=0 völliger Transparenz und Alpha=255 völliger Überdeckung. Für alle Werte dazwischen werden 'durchscheinende' Bilder erzeugt, d. h. ein Farbwert wird berechnet, der 'zwischen' überzeichnender und überzeichneter Farbe liegt. Dabei gibt der Alphawert an, welcher der beiden Werte den Farbeindruck dominiert. Berechnet wird die neue Farbe nach dieser Funktion:

```
Function custom_alpha(Byval src As Uinteger, Byval dst _
   As Uinteger, Byval param As Any Ptr ) As Uinteger
  Dim As Ubyte ptr color_src, color_dst
  Dim As Ubyte     r, g, b, a

  color_src = Cast(Ubyte ptr, @src)
  color_dst = Cast(Ubyte ptr, @dst)

  If param <> 0 Then
    a = *Cast(Ubyte ptr, param)
    ' ein Alphawert für alle Pixel
  Else
    a = color_src&"hlzahl">3]
    ' aus jedem Pixel den Alphawert lesen
  End If

  r=(color_src&"hlzahl">2]*a+color_dst[2]*(255-a))\255
  g=(color_src[1]*a+color_dst[1]*(255-a))\255
  b=(color_src[0]*a+color_dst[0]*(255-a))\255
  Return RGBA(r, g, b, 255)
End Function
```

Diese Funktion ist so gehalten, dass sie mit der Methode CUSTOM benutzt werden kann (siehe PUT (Grafik), DRAW STRING).

- 'src' entspricht der Farbinformation des überzeichnenden (neuen) Pixels.
- 'dst' entspricht der Farbinformation des zu überzeichnenden (bereits auf der Zeichenfläche befindlichen) Pixels.
- 'param' ist ein Pointer auf einen UBYTE-Wert zwischen 0 und 255, der den Alpha-Wert darstellt.

Struktur eines Bildpuffers für die Drawing Primitives
Ein Bildpuffer ist ein Speicherbereich, in dem größere Blocks von Pixeldaten - kurz Bildausschnitte - abgelegt werden können. Dies kann ein Array sein (siehe DIM, REDIM) oder ein mit ALLOCATE, CALLOCATE oder (besonders zu empfehlen) IMAGECREATE reservierter Bereich. In FreeBASIC werden zwei verschiedene Formate für Bildpuffer verwendet; abhängig von der Version des Compilers und den

Kommandozeilenoptionen verwendet FreeBASIC Version 1 oder Version 2 der verfügbaren Formate. Beide Formate sind mit allen Drawing Primitives (z. B. PSET, LINE (Grafik)) kompatibel. Siehe auch PUT (Grafik), GET (Grafik).

Version 2 (aktuell)
Version 2 des Bildpuffers ist seit v0.17 verfügbar; das Programm kann ohne Kommandozeilenoption oder mit -lang fb compiliert werden. Ebenso wie in der weiter unten aufgeführten Version 1 lässt sich dieses Bildpuffer-Format in einen Header und die eigentlichen Pixeldaten zerlegen. Der Bildpuffer ist dabei so angelegt:

```
TYPE Image FIELD = 1
  UNION
    old             AS _OLD_HEADER
    type            AS UINTEGER
  END UNION
  bpp                AS INTEGER
  width              AS UINTEGER
  height             AS UINTEGER
  pitch              AS UINTEGER
  _reserved(1 to 12) AS UBYTE
END TYPE
```

Wie Sie sehen, enthält der Header noch die Informationen des alten Formates. Dadurch ist eine schnelle und einfache Prüfung der Pufferversion möglich, wodurch die Abwärtskompatibilität gewahrt wird. Die einzelnen Elemente bedeuten in der Deklaration bedeuten Folgendes:

- 'old' ist eine Speicherstruktur, die dem Header der Version 1 entspricht.
- 'type' ist ein INTEGER-Wert, der die Version des Headers angibt. Da es sich in einem UNION mit 'old' befindet, wird es diesen Record überschreiben. Soll die neue Header-Struktur benutzt werden, so hat 'type' den Wert 7. Diesen Wert kann der alte Header nie erreichen, wodurch sicher gestellt ist, dass die Version eindeutig erkennbar ist.
- 'bpp' gibt die Farbtiefe in Bytes pro Pixel an. Dieser Wert ist entweder gleich 1 (für 1-8bit), 2 (für 15bit und 16bit), oder gleich 4 (für 24bit und 32bit)
- 'width' und 'height' geben die Breite und Höhe des Bildpuffers in Pixeln an.
- 'pitch' gibt die Anzahl der Bytes pro Zeile an. Der Wert berechnet sich auch für alle Farbtiefen (bpp = 1, 2 oder 4): pitch=(width*bpp+15) AND -16. Jede Bildzeile wird also so aufbereitet, dass die Anzahl der Byte pro Zeile ein Vielfaches von 16 ist. Dies wirkt sich darin aus, dass in fast jedem Bildpuffer einige ungenutzte Bytes vorhanden sind; in Anbetracht der Größe des zur Verfügung stehenden Speichers fallen diese aber gar nicht ins Gewicht. Der Vorteil dieser Methode liegt darin, dass sich der Umgang mit den Bildpuffern optimieren lässt; die heutigen 32bit-Prozessoren bieten 128-Bit-Register (= 16 Byte) die damit optimal angewandt werden können. Einem minimal größeren Speicheraufwand steht also ein spürbar kleinerer Zeitaufwand bei Berechnungen gegenüber.
- '_reserved' ist eine 12 Byte große Speicherstelle, die später möglicherweise für verschiedene OpenGL-Operationen verwendet werden kann; die Verwendung dieser Stellen steht im Moment noch nicht fest. Der Programmierer kann frei entscheiden, ob er diese Speicherstellen nutzen möchte, sollte dabei aber bedenken, dass sein Programm damit möglicherweise später nicht mehr kompatibel ist.

Wie bei 'pitch' bereits angesprochen, sind im neuen Datenformat die Pixel nicht mehr direkt aneinander gereiht, sondern zeilenweise in Datenblocks zusammengefasst. Hinter jeder Zeile werden eine Reihe von Bytes reserviert, die nicht genutzt werden. Durch dieses 'Padding' können bestimmte Berechnungen schneller durchgeführt werden. Der Wert dieser Padding-Bytes ist abhängig von der Art der Erstellung des Puffers. Bei der Verwendung eines Arrays wird hier in den meisten Fällen null eingetragen, da mit DIM und REDIM initiierte Variablen i. d. R. mit null initialisiert werden. Eine Ausnahme sind Arrays, die durch = ANY initialisiert wurden; da hier der Inhalt des Speicherbereichs nicht gelöscht wird, und nur die tatsächlich

genutzen Bytes im Laufe der Bildbearbeitung überschrieben werden, findet man hier immer den 'Datenmüll', der sich vor der Reservierung des Speicherbereichs auf den entsprechenden Stellen angesammelt hat. Durch CALLOCATE wird immer ein null-initialisierter Speicherbereich reserviert.

Wie schon bei Arrays, die durch = ANY initialisiert wurden, findet man auch bei der Reservierung eines Speicherbereichs mittels ALLOCATE Datenmüll in den Padding-Bytes. IMAGECREATE arbeitet im Prinzip genauso wie ALLOCATE, berechnet jedoch selbständig den zu reservierenden Speicherplatz und legt außerdem bereits den Header an. Daher handelt es sich hier um die bevorzugte Vorgehensweise für Bildpuffer. Die Speicherbereiche, die mit IMAGECREATE reserviert wurden, werden vor Benutzung komplett mit einem Wert befüllt; dieser hängt von den Parametern, mit denen IMAGECREATE genutzt wird, und dem aktuellen Bildschirmmodus ab.

Um nun bei gegebener Startadresse des Bildpuffers die Adresse eines einzelnen Pixels zu ermitteln, bedient man sich dieser Formel:

```
p = buffer+32+(y*((w*bpp+15) AND -16)+(x*bpp)
```

- 'p' ist ein Pointer, der auf den gewünschten Pixel zeigt. Benutzen Sie bevorzugt einen UBYTE PTR, damit bei der Berechnung des Formelausdrucks kein Fehler auftritt. Wird z. B. an einem INTEGER PTR die Operation +=1 ausgeführt, so erhöht sich sein Wert tatsächlich um vier, da ein INTEGER die Länge von 4 Byte besitzt.
- 'buffer' gibt die Startadresse des Bildpuffers an.
- 'x' und 'y' sind die Koordinaten des Pixels, relativ zum linken oberen Rand des Bildpuffers.
- 'w' ist die Breite des Bildes in Pixeln.
- 'bpp' gibt die Farbtiefe des Bildpuffers in Bytes pro Pixel an.

Steht 'pitch' aus dem Bildheader zur Verfügung, so kann der Ausdruck auch vereinfacht werden:

```
p = buffer + 32 + (y * pitch) + (x * bpp)
```

Der Speicherbedarf berechnet sich also nach dieser Formel:

```
size = 32 + ( h * (( w * bpp + 15 ) AND -16 )
```

Beispiel: Ein Bildpuffer der Größe 2x2 Pixel in einem 32bpp-Modus, der einen roten, grünen, gelben und blauen Pixel enthält:

Byte	Bedeutung	Wert
0 bis 3	Version des Bildpuffers - neuer Header	7
4 bis 7	Farbtiefe in bpp	4
8 bis 11	Breite des Bildpuffers in Pixeln	2
12 bis 15	Höhe des Bildpuffers in Pixeln	2
16 bis 19	Bytes pro Zeile in diesem Puffer	16
20 bis 31	zwölf reservierte Bytes, die in zukünftigen OpenGL-Anwendungen eine Bestimmung finden könnten	0
32 bis 35	Farbe des ersten Pixels (links oben) im Format &hAARRGGBB	&h00FF0000
36 bis 39	Farbe des zweiten Pixels (rechts oben) im Format &hAARRGGBB	&h0000FF00
	Padding-Stelle - ohne Bedeutung	&hFFFF00FF

40 bis 43		
44 bis 47	Padding-Stelle - ohne Bedeutung	&hFFFF00FF
48 bis 51	Farbe des dritten Pixels (links unten) im Format &hAARRGGBB	&h00FFFF00
52 bis 55	Farbe des vierten Pixels (rechts unten) im Format &hAARRGGBB	&h000000FF
56 bis 59	Padding-Stelle - ohne Bedeutung	&hFFFF00FF
60 bis 63	Padding-Stelle - ohne Bedeutung	&hFFFF00FF

Beispiel 1:

```
"hlstring">"fbgfx.bi"
Using FB

ScreenRes 400, 300, 32

Dim buffer As Any      Ptr
Dim header As Image    Ptr
Dim Pixels As UInteger Ptr
Dim Colors As UByte    Ptr
Dim i      As UInteger

buffer = ImageCreate(2, 2)
header = buffer
Pixels = buffer + 32
Colors = buffer + 32

Pset buffer, (0, 0), &hFF0000
Pset buffer, (1, 0), &h00FF00
Pset buffer, (0, 1), &hFFFF00
Pset buffer, (1, 1), &h0000FF

With *header
  PRINT "Header des Buffers:"
  PRINT "Version ID:       "; .type
  PRINT "Bytes pro Pixel: "; .bpp
  PRINT "Breite:           "; .width
  PRINT "Hoehe:            "; .height
  PRINT "Bytes pro Zeile:  "; .pitch
End With

Draw String (0, 100), "Die Grafik:"
Put (100, 100), buffer, PSET

Open Err For Output As "hlkw0">PRINT "hlzeichen">, "Pixeldaten:"
PRINT "hlzeichen">, ""
For i = 0 To 31
  PRINT "hlzeichen">, i, Hex(Colors&"hlzeichen">], 2),
  If (i + 1) Mod 4 = 0 Then PRINT "hlzeichen">, ""
  If (i    ) Mod 4 = 0 Then
```

```
    PRINT "hlzeichen">, Hex(Pixels&"hlzahl">4], 8)
  Else
    PRINT "hlzeichen">, ""
  End If
Next

ImageDestroy buffer
Close "hlkw0">Sleep
```

Beispiel 2:
Über das UDT Image kann auch direkt auf die Header-Daten zugegriffen werden:

```
"hlstring">"fbgfx.bi"

ScreenRes 400, 300, 32

Dim As FB.Image Ptr img
Dim As UInteger Ptr Pixel
Dim As UInteger     x, y, pitch

img = ImageCreate(256, 256)
pitch = img->pitch

For y = 0 To 255
  For x = 0 To 255
     Pixel = Cast(Any Ptr, img) + 32 + (y * pitch) + (x * 4)
    *Pixel = RGB(x, y, 0)
  Next
Next

Put (0, 0), img
Sleep
```

Erkennen des Puffertyps und Bearbeitung
Je nach gewählten Ausgangsbedingungen wird FreeBASIC bei der Erstellung neuer Datenpuffer immer eine bestimmte Version des Puffers benutzen:

- Bedingungen für Version 1:
  FreeBASIC-Version <= 0.16 oder
  FreeBASIC-Version >= 0.17 und Kommandozeilenoption -lang qb oder
  FreeBASIC-Version >= 0.17 und Kommandozeilenoption -lang deprecated
- Bedingungen für Version 2:
  FreeBASIC-Version >= 0.17 und Kommandozeilenoption -lang fb

Dennoch ist es möglich, dass ein anderes Pufferformat behandelt werden soll als das unter gegebenen Bedingungen von FreeBASIC gewählte, z. B. wenn mittels BLOAD Daten in den Speicher geladen werden, die ein älteres Programm erstellt hat. Die Drawing Primitives ab FreeBASIC v0.17 können - bei jeder gewählten Kommandozeilenoption - mit beiden Pufferformaten umgehen; sie erkennen automatisch, welche Struktur angewandt werden muss. Die Drawing Primitives von FreeBASIC-Versionen vor v0.17 können nur mit Version 1 des Pufferformats umgehen; der Programierer muss selbständig Daten des jüngeren Formats umwandeln. Bei direkten Speicherzugriffen ohne die FB-eigenen Drawing Primitives kann anhand der ersten INTEGER-Stelle des Headers überprüft werden, ob es sich um das alte oder das neue Format handelt: Beim neuen Format beinhaltet diese immer den Wert 7; beim alten Format ist hier ein anderer Wert gespeichert, der sich aus Farbtiefe, Höhe und Breite des Bildpuffers zusammensetzt. Dieser Wert kann nie gleich sieben

werden. Mit dem Headertyp ändert sich auch der Startpunkt der eigentlichen Pixeldaten. Diese Formel gibt unabhängig von Pointertyp und Pufferformat den richtigen Startpunkt aus:

```
start = buffer + IIF( PEEK(INTEGER, buffer) = 7, 32, 4 ) \ SIZEOF(buffer)
```

Beachten Sie hierbei, dass Sie separat prüfen müssen, ob die Zeilen gepaddet sind. Dies ist bei Version 2 IMMER der Fall, in Version 1 hingegen NIE.

Version 1 (veraltet)
Die alte Version des Bildpuffers entspricht dem Pufferformat von QB. Sie wurde bis FreeBASIC v0.16 benutzt; seit v0.17 ist sie nur noch dann verfügbar, wenn die Kommandozeilenoption -lang deprecated oder -lang qb eingesetzt wird. Der Bildpuffer besteht aus dem Header und den eigentlichen Pixelinformationen. Der Header ist ein Bitfeld (siehe TYPE (UDT) und Bitfelder), das Angaben über Höhe, Breite und Farbtiefe des Bildes enthält; an ihn schließen sich die eigentlichen Pixeldaten im oben genannten Format an. Der Header besteht insgesamt aus 32bit, also vier Byte oder einer INTEGER-Stelle. Er ist folgendermaßen aufgebaut:

```
TYPE _OLD_HEADER FIELD = 1
  bpp      :  3 AS USHORT
  width    : 13 AS USHORT
  height        AS USHORT
END TYPE
```

Die einzelnen Elemente bedeuten dabei Folgendes:

- 'bpp' gibt die Farbtiefe in Bytes pro Pixel an. Er ist entweder gleich 1 (für 1-8bit), 2 (für 15bit und 16bit), oder gleich 4 (für 24bit und 32bit)
- 'width' gibt die Breite des Bildes in Pixeln an. Da 13 Bits verwendet werden, kann dieser Wert zwischen 1 und 8191 liegen; eine Breite von 0 ist (aus praktischen Gründen) ungültig.
- 'height' gibt die Höhe des Bildes in Pixeln an. Da hier eine volle USHORT-Stelle (16 Bit) verwendet wird, kann dieser Wert zwischen 1 und 65535 liegen; eine Höhe von 0 ist (aus praktischen Gründen) ungültig.

Dem Header schließen sich die eigentlichen Pixeldaten an; für jeden Pixel werden dabei so viele Bytes verwendet, wie in 'bpp' angegeben ist. Die Pixeldaten sind von links nach rechts und von oben nach unten geordnet. Die Größe eines Puffers nach diesem Format lässt sich also folgendermaßen berechnen:

```
size = 4 + (w * h * b)
```

wobei 'w' die Breite, 'h' die Höhe und 'b' die Farbtiefe in Bytes pro Pixel ist.

Beispiel: Ein Bildpuffer der Größe 2x2 Pixel in einem 32bpp-Modus, der einen roten, grünen, gelben und blauen Pixel enthält:

Byte	Bedeutung	Wert
0 und 1	(Breite SHL 3) OR bpp	20
2 und 3	Höhe	2
4 bis 7	Farbe des ersten Pixels (links oben) im Format &hAARRGGBB	&h00FF0000
8 bis 11	Farbe des zweiten Pixels (rechts oben) im Format &hAARRGGBB	&h0000FF00
12 bis 15	Farbe des dritten Pixels (links unten) im Format &hAARRGGBB	&h00FFFF00
16 bis 19	Farbe des vierten Pixels (rechts unten) im Format &hAARRGGBB	&h000000FF

Dies verdeutlicht auch dieses Programm:

```
"hlstring">"deprecated"
ScreenRes 400, 300, 32
Dim x As UByte    Ptr
Dim y As UInteger Ptr
Dim i As Integer
x = ImageCreate(2, 2)
y = Cast(UInteger Ptr, x)
Pset x, (0, 0), &hFF0000
Pset x, (1, 0), &h00FF00
Pset x, (0, 1), &h0000FF
Pset x, (1, 1), &hFFFF00
For i = 0 To 19
  Print i, x&"hlzeichen">],
  If ( i      Mod 4) = 0 Then
    Print Hex(y&"hlzahl">4], 8)
  Else
    Print
  End If
  If ((i + 1) Mod 4) = 0 Then Print
Next
ImageDestroy x
Sleep
```

Letzte Bearbeitung des Eintrags am 20.08.13 um 19:49:29
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

Tipps und Tricks

FreeBASIC-Referenz » Die Gfxlib » **Tipps und Tricks**
Achtung: Dieser Abschnitt richtet sich an fortgeschrittene Programmierer, die einen tieferen Einblick in die Verwaltung von Grafikpuffern erhalten wollen. Er behandelt folgende Themen:

- Double Buffering
- Arten der Grafikpuffer
- Grafikpuffer bei Drawing Primitives
- GfxPrint
- LZW-Codec in der Gfxlib
- OpenGL-Textur

Double Buffering

Vermeiden Sie Double Buffering (ein Bild zuerst in einen Puffer schreiben und es dann in den Video-RAM kopieren) soweit wie möglich. Die Gfxlib benutzt bereits ein Double-Buffering-System, um den Bildschirm zu aktualisieren. Wenn Sie Ihren eigenen Zweitpuffer verwenden, müssen nur weitere (unnötige) Bildschirmkopien durchgeführt werden, wodurch Ihr Programm langsamer wird. Stattdessen sollten Sie die bereits unterstützten Flipping-Funktionen (Verwendung mehrerer Bildschirmseiten; siehe SCREENSET und das Beispiel von MULTIKEY) benutzen.

Arten der Grafikpuffer

GET und PUT unterstützen neben normalen Arrays auch Pointer als Grafikpuffer:

```
TYPE FB_IMAGE FIELD = 1
  width AS USHORT
  height AS USHORT
  imageData(64000) AS UBYTE
END TYPE

DIM udt AS FB_IMAGE
DIM udt_ptr AS FB_IMAGE PTR
DIM array(16008) AS INTEGER
DIM array_ptr AS INTEGER PTR

udt_ptr = @udt
array_ptr = @array(0)

SCREENRES 320, 200

' Diese Anweisungen haben dieselbe Funktion:
GET (0, 0)-(319, 199), array
GET (0, 0)-(319, 199), array(0)
GET (0, 0)-(319, 199), @array(0)
GET (0, 0)-(319, 199), array_ptr
GET (0, 0)-(319, 199), @array_ptr&"hlzahl">0]

' Diese Anweisungen sind ebenfalls äquivalent zueinander:
GET (0, 0)-(319, 199), @udt
GET (0, 0)-(319, 199), udt_ptr
GET (0, 0)-(319, 199), @udt_ptr&"hlzahl">0]
```

```
SLEEP
```

In diesem Beispiel wurde GET verwendet, es funktioniert aber ebenso mit PUT.

Grafikpuffer bei Drawing Primitives

Alle Drawing Primitives (einfachste Grafikanweisungen wie LINE, CIRCLE usw.) können an Grafikpuffer angewandt werden. Dadurch ist es für Sie möglich, auf eine beliebige Anzahl von Nicht-Bildschirm-Oberflächen zu zeichnen und diese später auf den Bildschirm zu übertragen.

Beispiel:

```
DIM buffer(9602) AS USHORT
DIM sprite(1028) AS USHORT
SCREENRES 320, 200

' Da als Puffer ein Grafikpuffer angenommen wird,
' setzen wir seine ersten vier Bytes so, dass sie einen
' normalen Grafikpuffer-Header bilden.
buffer(0) = 160 SHL 3     ' Breite * 8
buffer(1) = 120       ' Höhe

CIRCLE buffer, (16, 16), 15, 12, , , 1, F
GET    buffer, ( 0,  0)-( 31,  31), sprite
PSET   sprite, (16, 16), 15
LINE   buffer, ( 0,  0)-(159, 119), 2, B
PUT    buffer, (50, 50), sprite, PSET
PUT            (80, 40), buffer, PSET
SLEEP
```

Einen einfacheren Weg, einen solchen Grafikpuffer zu erstellen stellt die Funktion IMAGECREATE dar. Durch sie wird automatisch ein Puffer der richtigen Größe reserviert und sein Header initialisiert. Das obige Beispiel könnte dadurch so aussehen:

Beispiel: Grafikpuffer mit IMAGECREATE reservieren

```
DIM buffer AS ANY PTR
DIM sprite AS ANY PTR

SCREENRES 320, 200

buffer = IMAGECREATE(160, 120)
sprite = IMAGECREATE(32, 32)

CIRCLE buffer, (16, 16), 15, 12, , , 1, F
GET    buffer, ( 0,  0)-( 31,  31), sprite
PSET   sprite, (16, 16), 15
LINE   buffer, ( 0,  0)-(159, 119), 2, B
PUT    buffer, (50, 50), sprite, PSET
PUT            (80, 40), buffer, PSET

IMAGEDESTROY buffer
IMAGEDESTROY sprite
SLEEP
```

Wenn Sie auf einen Puffer zeichnen, werden die Koordinaten durch den letzten Aufruf von WINDOW beeinflusst, jedoch nicht von VIEW. Die Clipping-Grenzen werden auf die Gesamtgröße des Puffers gesetzt. Der optionale Zielpuffer-Parameter kann bei allen Drawing Primitives angegeben werden und darf sowohl ein Array als auch ein Pointer sein, wie im Falle des Grafikpuffers.

GfxPrint

Um einen transparenten Text pixelgenau an einer beliebigen Fensterposition auszugeben, dient der Befehl DRAW STRING. Vor FreeBASIC v0.16 existierte dieser Befehl noch nicht, konnte jedoch durch eine eigene Routine ersetzt werden. Das folgende Beispiel enthält die SUB GfxPrint, die das Zeichnen von transparentem Text an jeder Position sowie Clipping-Grenzen unterstützt; ebenso ist es möglich, in einen Grafikpuffer zu schreiben.

```
DECLARE SUB GfxPrint( BYREF text AS STRING, _
  BYVAL x AS INTEGER, BYVAL y AS INTEGER, _
  BYVAL col AS INTEGER, BYVAL buffer AS ANY PTR = 0 )

TYPE fb_FontType
  w As INTEGER
  h AS INTEGER
  data AS UBYTE PTR
END TYPE

ENUM
  FB_FONT_8 = 0,
  FB_FONT_14,
  FB_FONT_16,
  FB_FONT_COUNT
END ENUM

EXTERN __fb_font(0 to FB_FONT_COUNT-1) ALIAS "__fb_font" AS fb_FontType

' Entfernen Sie das Kommentar-Zeichen vor dem
' Zeichenformat, das Sie benutzen wollen
'"hlkommentar">'#DEFINE fb_FontData (__fb_font(FB_FONT_14))
'#DEFINE fb_FontData (__fb_font(FB_FONT_16))

SUB GfxPrint( BYREF text AS STRING, _
  BYVAL x AS INTEGER, BYVAL y AS INTEGER, _
  BYVAL col AS INTEGER, BYVAL buffer AS ANY PTR = 0 )

  DIM row AS INTEGER, i AS INTEGER
  DIM bits AS UBYTE PTR

  FOR i = 1 TO LEN(text)
    bits = fb_FontData.data + (ASC(MID$(text, i, 1)) * fb_FontData.h)
    FOR row = 0 TO fb_FontData.h-1
      IF (buffer) THEN
        LINE buffer, (x + 7, y + row)-(x, y + row), col, , *bits SHL 8
      ELSE
        LINE (x + 7, y + row)-(x, y + row), col, , *bits SHL 8
      END IF
      bits += 1
```

```
    NEXT row
    x += 8
  NEXT i
END SUB

SCREENRES 320, 200
GfxPrint "Hello world!", 112, 96, 15
SLEEP
```

Achtung: Die Gfxlib speichert alle Font- und Paletten-Daten in einem LZW-komprimierten Format, um Ihre EXEs klein zu halten, und dekomprimiert sie erst mit dem ersten SCREENRES-Aufruf. Wenn Sie SCREENRES in Ihrem Programm nie aufrufen, werden Zugriffe auf die Palette- und Font-Daten unbrauchbare Daten zurückliefern.

LZW-Codec in der Gfxlib

Die Gfxlib speichert die Paletten- und Font-Daten in einem LZW-komprimierten Format und entpackt diese erst, wenn zum ersten mal SCREENRES aufgerufen wird. Der LZW-Codec ist in Ihren Programmen nicht aufrufbar; er wurde nicht als eigener Befehl eingebaut. Es ist dennoch möglich, auf die Kompressionsfunktionen zuzugreifen, indem Sie die Funktionen einfach deklarieren; die Gfxlib muss jedoch durch einen SCREENRES-Aufruf aktiviert sein.

Beispiel:

```
DECLARE FUNCTION LZW_Encode _
  ALIAS "fb_hEncode" ( _
  BYVAL in_buffer  AS ANY PTR, _
  BYVAL in_size    AS INTEGER, _
  BYVAL out_buffer AS ANY PTR, _
  BYREF out_size   AS INTEGER _
) AS INTEGER

DECLARE FUNCTION LZW_Decode _
  ALIAS "fb_hDecode" ( _
  BYVAL in_buffer  AS ANY PTR, _
  BYVAL in_size    AS INTEGER, _
  BYVAL out_buffer AS ANY PTR, _
  BYREF out_size   AS INTEGER _
) AS INTEGER

DECLARE SUB showData(buffer() AS UBYTE)

DIM src_buffer(100000) AS UBYTE
DIM src_size AS INTEGER
DIM dest_buffer(100000) AS UBYTE
DIM dest_size AS INTEGER

' funktioniert nur, wenn die Gfxlib verwendet wird
SCREENRES 400, 300

src_size = 0
OPEN "test.bas" FOR BINARY AS "hlkw0">WHILE NOT EOF(1)
  GET "hlzeichen">,, src_buffer(src_size)
  src_size += 1
```

```
WEND
CLOSE "hlkw0">PRINT "Data size before compression:", src_size
ShowData src_buffer()

dest_size = 100000
PRINT "Compressing...";
LZW_Encode @src_buffer(0), src_size, _
  @dest_buffer(0), dest_size
PRINT "done."

PRINT "Data size after compression:", dest_size
PRINT

src_size = 100000
PRINT "Decompressing...";
LZW_Decode @dest_buffer(0), dest_size, _
  @src_buffer(0), src_size
PRINT "done."

PRINT "Data size before decompression:", src_size
ShowData src_buffer()
SLEEP
END

SUB showData(buffer() AS UBYTE)
  DIM i AS INTEGER
  PRINT "Contents: ";
  FOR i = 3 TO 36: PRINT CHR(buffer(i)); : NEXT
  PRINT " &"
END SUB
```

OpenGL-Textur

Das Erstellen einer OpenGL-Textur mit einem Grafikpuffer ist einfach; hier ist ein kleiner Codeauszug, der diese Aufgabe erledigt:

```
"hlzahl">&h1
"hlzahl">&h2
"hlzahl">&h4
"hlzahl">&h8

"hlkw0">ONCE "GL/gl.bi"
"hlkw0">ONCE "GL/glu.bi"

FUNCTION CreateTexture( BYVAL buffer AS ANY PTR, _
  BYVAL flags AS INTEGER = 0 ) AS GLuint
  REDIM dat(0) AS UBYTE
  DIM p AS UINTEGER PTR, s AS USHORT PTR
  DIM AS INTEGER w, h, x, y, col
  DIM tex AS GLuint
  DIM AS GLenum format, minfilter, magfilter

  CreateTexture = 0
```

```
s = buffer
w = s&"hlzahl">0] SHR 3
h = s&"hlzahl">1]

IF( (w < 64) OR (h < 64) ) THEN
  EXIT FUNCTION
END IF
IF( (w AND (w-1)) OR (h AND (h-1)) ) THEN
  ' Breite oder Höhe keine Potenz von 2
  EXIT FUNCTION
END IF

REDIM dat(w * h * 4) AS UBYTE
p = CAST(UINTEGER PTR, @dat(0))

glGenTextures 1, @tex
glBindTexture GL_TEXTURE_2D, tex

FOR y = h-1 TO 0 STEP -1
  FOR x = 0 TO w-1
    col = POINT(x, y, buffer)
    ' vertausche R und B, um das GL_RGBA-Texturformat zu nutzen
    col = RGBA(col AND &hFF, (col SHR  8) AND &hFF, _
          (col SHR 16) AND &hFF, col SHR 24)
    IF( flags AND TEX_HASALPHA ) THEN
      *p = col
    ELSE
      IF( (flags AND TEX_MASKED) AND (col = &hFF00FF) ) THEN
        *p = 0
      ELSE
        *p = col OR &hFF000000
      END IF
    END IF
    p += 4
  NEXT x
NEXT y

IF( flags AND ( TEX_MASKED OR TEX_HASALPHA ) ) THEN
  format = GL_RGBA
ELSE
  format = GL_RGB
END IF

IF( flags AND TEX_NOFILTER ) THEN
  magfilter = GL_NEAREST
ELSE
  magfilter = GL_LINEAR
END IF

IF( flags AND TEX_MIPMAP ) THEN
  gluBuild2DMipmaps GL_TEXTURE_2D, format, w, _
     h, GL_RGBA, GL_UNSIGNED_BYTE, @dat(0)
  IF( flags AND TEX_NOFILTER ) THEN
```

```
      minfilter = GL_NEAREST_MIPMAP_NEAREST
    ELSE
      minfilter = GL_LINEAR_MIPMAP_LINEAR
    END IF
  ELSE
    glTexImage2D GL_TEXTURE_2D, 0, format, w, _
       h, 0, GL_RGBA, GL_UNSIGNED_BYTE, @dat(0)
    minfilter = magfilter
  END IF
  glTexParameteri GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, minfilter
  glTexParameteri GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, magfilter

  CreateTexture = tex
END FUNCTION
```

Letzte Bearbeitung des Eintrags am 19.01.13 um 21:46:54
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

ASCII-Codes

FreeBASIC-Referenz » Verschiedene Themen » **ASCII-Codes**
Die ASCII-Tabellen der Codepages 437 (Standard im Gfxlib-Screen) und 850 (Deutsche Codepage, in der Konsole) finden sich im Onlineangebot des FreeBASIC-Portals.

Letzte Bearbeitung des Eintrags am 21.12.12 um 19:17:23
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

Der Compiler

FreeBASIC-Referenz » Verschiedene Themen » **Der Compiler**
Die offizielle FreeBASIC-Ausgabe enthält als Compiler den **fbc**, FreeBASICs Flagschiff. fbc kann von der Befehlszeile gestartet werden - von DOS, dem Windows-Befehlszeilenprompt oder einer Linux-Shell. IDE-verwöhnte Benutzer können sich eine IDE (z. B. FBEdit, FBIDE, Geany, wxFBE [in alphabetischer Reihenfolge]) selbst installieren, um nicht mit der Befehlszeile hantieren zu müssen. Der Aufruf von fbc von der Konsole ohne Argumente liefert eine Liste der vorhandenen Optionsschalter, die benutzt werden können, um das Verhalten des Compilers genau einzustellen.

Im weiteren Teil dienen Anführungszeichen zur Kennzeichnung der Optionsschalter im Text. Sie sind selbst nicht Bestandteil der Optionsschalter.

Im einfachsten Fall nimmt fbc eine Quelldatei als Befehlszeilen-Argument und erstellt eine ausführbare Datei. Dabei wird die Quelldatei (.bas) in eine Assembler-Datei (.asm) compiliert, die erst per GAS in eine Objekt-Datei (.o) übersetzt wird und dann vom Linker LD mit anderen Objekt-Dateien und Libraries (diese Programmbibliotheken werden im Allgemeinen und im Folgenden unter Linux und Windows "Libraries" genannt), die für den Ablauf erforderlich sind, letztendlich zur ausführbaren Datei gemacht wird. Die Assembler- und compilierten Objekt-Dateien werden dann normalerweise gelöscht. Zum Beispiel erstellt der folgende Befehl:

```
fbc foo.bas
```

die ausführbare Datei foo.exe unter DOS/Windows und ./foo unter Linux. fbc übernimmt mehrere Quelldateien auf einmal und compiliert und linkt sie alle in eine ausführbare Datei. Beispielsweise erzeugt der Befehl:

```
fbc foo.bas bar.bas baz.bas
```

die ausführbare Datei foo.exe bzw. ./foo. Da foo.bas zuerst in der Liste steht, wird es der Haupteintrittspunkt der Datei sein und auch den Namen dafür liefern. Um einen anderen Eintrittspunkt zu bestimmen, werden die Schalter "-m" und "-x" benutzt. Soll also z. B. baz.bas der Haupteintrittspunkt der Datei foobar.exe sein, würde der Aufruf:

```
fbc -x foobar.exe -m baz foo.bas bar.bas baz.bas
```

benutzt werden.
Der Schalter "-x" benennt genau die ausführbare Datei, also würde auch unter linux aus dem obigen Befehl die ausführbare Datei ./foobar.exe erstellt werden.

Syntax:

```
fbc [ Optionen ] [ Eingabeliste ]
```

'Eingabeliste' ist eine Liste von Dateinamen. Der Compiler entscheidet abhängig von der Erweiterung, wie eine Datei behandelt werden soll. Akzeptiert werden folgende Dateitypen:

Dateierweiterung	Beschreibung
.bas	FreeBASIC-Quelldatei: die zu compilierende Programmdatei
.a	Bibliothek (Library): statische Sammlung von vorcompilierten Prozeduren
.o	Objektdatei: einzelnes vorcompiliertes Modul
.rc	**nur Windows:** Ressourcen-Script, Ressourcen-Beschreibung (Dialoge, Menüs, Icons, Tastaturkürzel)

.res	**nur Windows:** Compilierte Ressourcendatei
.xpm	**nur Linux:**X PixMap (Icon)

Die folgenden Parameter können an den FreeBASIC-Compiler übergeben werden:

Erzeugung von Bibliotheken mit privaten/öffentlichen Prozeduren

-export [Name]	Exportiert Symbole für dynamisches Linken; siehe EXPORT. Der Name einer SUB/FUNCTION wird explizit für andere Module zugänglich gemacht. Ein anderes Modul kann mit Hilfe der Anweisung DECLARE solche öffentlich deklarierten Prozeduren benutzen.
-dll, -dylib	Erzeugt eine dynamische Link-Library. So wird unter Windows eine *.dll (inklusive der Import-Library *.dll.a) bzw. unter Linux eine *.so erstellt.
-lib	Erzeugt eine statische Library. Es wird eine Sammlung von Prozeduren vorcompiliert, die aber nicht wie bei den Optionen "-dll" und "-dylib" einem Programm separat mitgegeben, sondern direkt in das fertige Programm eingefügt wird. Dies nennt man daher statisches Linken.
[-a] [Name]	Fügt eine vorcompilierte Objekt-Datei (*.o, *.a) zur Linker-Liste hinzu. Das "-a" ist optional, wenn die vorcompilierte Objektdatei die Endung *.o oder *.a besitzt.
-c	Compiliert nur (erzeugt eine Objektdatei), ohne zu linken. Es werden Sammlungen von Prozeduren vorcompiliert, aber nicht zu einem ausführbaren Programm verbunden. Werden mehrere Dateien gleichzeitig oder hintereinander compiliert, so muss die Hauptdatei mit "-m" markiert werden.
-C	Behält Objektdateien bei. Während des Compilierens werden Objektdateien mit der Endung *.o erzeugt und am Ende wieder gelöscht. Mit der Option -C werden die Objektdateien nicht gelöscht. *Die Option existiert seit FreeBASIC v0.20.*
-l [Name]	Fügt eine Library-Datei zur Linker-Liste hinzu. Eine fertige vorcompilierte statische Sammlung von Prozeduren mit dem Namen lib[Name].a wird statisch dem Programm hinzugefügt. Die ausführbare Programmdatei und das Mitgeben von DLLs (*.dll) oder Shared Librarys (*.so) kann entfallen.

Behandlung von Programmdateien (*.bas und *.bi)

[-b] [Name]	Fügt eine Quelldatei zur Compilierung hinzu. Das "-b" ist optional. Im Allgemeinen ist diese Option nicht nötig, kann aber benutzt werden, wenn die hinzuzufügende Datei nicht die Namenserweiterung .bas besitzt oder aus einem anderen Verzeichnis stammt.
-i [Name]	Fügt einen Suchpfad für include-Dateien (*.bi) hinzu. Wird die Option "-i" mehrfach verwendet, bestimmt die Reihenfolge ihres Auftretens die Reihenfolge, in der die Verzeichnisse durchsucht werden.
-include [Name]	Gibt eine Datei an, die eingebunden wird, bevor die Quelldateien übersetzt werden. Wird die Option "-include" mehrfach verwendet, bestimmt die Reihenfolge ihres Auftretens die Reihenfolge, in der die Dateien eingebunden werden. Die Option bewirkt, dass die mit 'Name' angegebenen Dateien an den Anfang des Hauptmoduls eingefügt werden, als wäre dort eine entsprechende INCLUDE-Anweisung.
-forcelang [fb|fblite|qb|deprecated]	Stellt die Dialektform ein, in der compiliert werden soll. Die Option überschreibt die Anweisung #LANG bzw. '$LANG im Quelltext. Siehe "-lang"
-lang [fb|fblite|qb|deprecated]	Stellt die Dialektform ein, in der compiliert werden soll. Um alte GW-BASIC- oder QuickBasic-/QBasic-Quellcodes ohne größere Änderungen zu compilieren, kann auf der Befehlszeile die Option "-lang qb"

	benutzt werden. Sie bietet eine verbesserte Kompatibilität zu QuickBasic-/QBasic-Code. Um FreeBASIC 0.16 oder noch ältere FreeBASIC-Quellcodes zu compilieren, kann die Option "-lang deprecated" benutzt werden. Sie wurde aus Kompatibilitätsgründen beibehalten, wird aber nicht weiterentwickelt. Wird die Option ausgelassen, so wird "-lang fb" verwendet. Die in der Option eingestellte Dialektform wird von einem im Quelltext verwendeten #LANG überschrieben. Siehe auch "-langforce".
-[m|entry] [Quelldatei]	Bestimmt den Haupteintrittspunkt einer Quelldatei; das Argument ist der Name einer Quelldatei ohne ihre Erweiterung. Die Datei [Quelldatei] wird als Startdatei angenommen. Sie enthält den Programmstart Main. Wird "-m" bzw. "-entry" nicht verwendet, wird die erste angegebene Datei mit der Namenserweiterung *.bas als Startdatei angesehen. Wird die Option "-c" oder "-r" verwendet, muss "-m" bzw. "-entry" angegeben werden, wenn eine Hauptquelldatei compiliert wird.
-o [Name]	Setzt den Namen der Ausgabedatei der compilierten Quelldatei oder Objektdatei (.o). "-o" muß unmittelbar nach der Quelldatei stehen, die mit "-b" angegeben wurde. Die Option wirkt allerdings nur zusammen mit der Option "-c", da Objektdateien normalerweise nicht gespeichert werden. Wird "-o" für eine übergebene Datei nicht benutzt, erhält die Ausgabedatei denselben Namen wie die Quelldatei, nur mit der Dateierweiterung *.o Unabhängig vom Namen der Programmdateien mit der Endung *.bas wird ein Programm mit dem Namen ./[Name] bzw. [Name].exe erzeugt.

Bedingtes Compilieren und Präprozessor

-d [Name=Wert]	Fügt ein Präprozessor-Makro allen Quelldateien hinzu. Die Option bewirkt dasselbe wie die Verwendung der Präprozessordirektiven #DEFINE oder #MACRO. Für bedingtes Compilieren kann man einen Wert definieren, z. B. DEBUG=1, der dann mit #IF oder #IFDEF bzw. #IFNDEF abgefragt werden kann.
-pp	Nur die vorcompilierte ('preprocessed') Quelldatei ausgegeben; keine komplette Compilierung

Fehlerbehandlung

-e	Einfache Fehlerunterstützung einschalten. Es werden Funktionen zur Fehlererkennung in das fertige Programm eingefügt, die zum Programmabbruch führen können, wenn ein schwerer Fehler auftritt.
-ex	Erweiterte Fehlerunterstützung einschalten. Die Option arbeitet wie "-e", aber mit der Möglichkeit, eigene Fehlerbehandlung zu implementieren. Mit Hilfe der Anweisung RESUME kann das Programm auf einen Fehler reagieren und es muss nicht zwangsläufig beendet werden.
-exx	Wie "-e" und "-ex", aber mit zusätzlicher Prüfung auf gültige Array-Grenzen und korrekte Verwendung von Zeigeradressen (Nullpointer-Prüfung).
-g	Debugger-Symbole in die Ausgabedateien einfügen, die von GDB-kompatiblen Debuggern verwendet werden können. Informationen zur Fehlersuche werden in das Programm integriert. Aufgrund der zusätzlichen Informationen ist es möglich, zur Laufzeit das Programm Schritt für Schritt auf seine Funktionsweise hin zu überprüfen. Das schrittweise Ausführen von Programmen wird mit Hilfe von Debuggerprogrammen realisiert. FreeBASIC liegt der GNU Debuger gdb bzw. gdb.exe bei. Es gibt aber auch frei erhältliche grafische Oberflächen, die das Debuggen erheblich vereinfachen und die Debug-Information visuell aufbereiten.
-noerrline	Eine fehlerhafte Stelle im Quellcode nicht anzeigen; nützlich, wenn eine IDE die Fehlermeldungen auswertet.

Programmerstellung

-arch [Typ]	Setzt den Ziel-CPU-Typ (Standard: 486 falls nicht angegeben) Als 'Typ' kann 386, 486, 586, 686, athlon, athlon-xp, athlon-fx, k8-sse3, pentium-mmx, pentium2, pentium3, pentium4, pentium4-sse3 oder native angegeben werden.
-asm att|intel	Setzt das verwendete Asm-Format (betrifft nur "-gen gcc")
-fpu [X87|SSE]	Wird die Option ausgelassen oder "-fpu X87" angegeben, benutzt fbc die normalen 387-Assembleranweisungen für mathematische Operationen. Bei der Option "-fpu SSE" werden SSE2-Assembleranweisungen zur Berechnungen von SINGLE- und DOUBLE-Variablen benutzt. Es wird zuvor geprüft, ob der Prozessor fähig ist, SSE2 Anweisungen auszuführen. Wenn nicht wird die Option ignoriert. *Die Option existiert seit FreeBASIC v0.20.*)
-fpmode [FAST | PRECISE]	Gibt die Genauigkeit von Berechnungen mit Nachkommazahlen an. Die Option ist nur in Verbindung mit "-fpu SSE" wirksam. *Die Option existiert seit FreeBASIC v0.21.*)
-gen [gas | gcc | llvm]	fbc übersetzt den Quelltext für x86-GAS-Assembler (gas), in C für GNU C (gcc - *Die Option existiert seit FreeBASIC v0.21.*) oder für die Low Level Virtual Machine (llvm - *Die Option existiert seit FreeBASIC v0.90.*).
-r	Nur Assemblerdateien mit der Endung *.asm erzeugen, nicht compilieren. Seit FBC 0.21 und unter Verwendung von "-gen gcc" werden Dateien mit der Endung *.c erstellt. Werden mehrere Dateien gleichzeitig oder hintereinander compiliert, so muss die Hauptdatei mit "-m" markiert werden. *Die Option existiert seit FreeBASIC v0.20.*
-R	Während des Compilierens werden Assemblerdateien mit der Endung *.asm erzeugt und am Ende wieder gelöscht. Mit der Option "-R" (Retain=behalten) werden die Assemblerdateien nicht gelöscht und können mit einem Editor eingesehen werden. Dies kann unter anderem dann nützlich sein, wenn man mit der Option "-g" Debug-Informationen erzeugt und sehen will, welche Programmanweisungen der Präprozessor generiert hat. Diese werden in den Assembler-Kommentaren abgelegt. Präprozessor-Makros in diesen Dateien werden ausgewertet. Seit FreeBASIC v0.21 werden unter Verwendung von "-gen gcc" Dateien mit der Endung *.c erstellt.
-RR	erhalte die endgültige .ASM-Datei *Die Option gibt es seit FreeBASIC v0.90.0*
-s [gui|console]	**Nur Windows:** Gibt an, ob ein Programm im Fenster oder in der Konsole ausgeführt werden soll. Standardmäßig wird "-s console" benutzt. Wird "-s gui" angegeben, erscheint beim Programmstart **kein Konsolenfenster**; bei Bedarf muss ein gfx-Grafikfenster initialisiert werden (siehe SCREENRES).
-t [Wert]	**Nur Windows/DOS:** Gibt die Größe des verwendeten Stackspeichers in KByte (1024 Byte-Einheiten) an (Standard: 1024 KBytes). Lokale Arrays werden auf dem Stack erzeugt, deshalb ist 1 MB möglicherweise nicht immer genug. Problematisch können auch umfangreiche Programme sein, die reichlich Gebrauch von rekursiven Funktionsaufrufen machen, wie z. B. Raytracer oder Fraktalgeneratoren.
-target [dos|cygwin]	**Nur Windows:** Erstellt ausführbare Dateien für andere Systeme. Die bin- und lib-Verzeichnisse müssen die Unterverzeichnisse /dos bzw. /linux der entsprechenden Distribution enthalten. Um die Option einzusetzen, um für andere Betriebssysteme zu compilieren, müssen weitere Voraussetzungen erfüllt sein.
-titel [Name]	**Nur XBOX:** Setzt den XBE-Anzeigetitel
-mt	Erzwingt das Linken mit 'Thread-sicherer' Laufzeit-Library für Applikationen mit mehreren Threads. Normalerweise wird automatisch immer die 'Thread-sichere' Version benutzt, wenn die eingebauten FreeBASIC-'threading'-Funktionen verwendet werden. Deshalb kommt diese Option nur bei eigenen 'Thread'-Routinen zum Einsatz.

-nodeflibs	Keine der Standard-Libraries beim Linken einbinden Bindet Standard-Bibliotheken nicht ein; alle Libraries müssen manuell mit #INCLIB geladen werden, und die Prozeduren darin mit DECLARE deklariert werden.
-O [Wert]	Setzt die Optimierungsebene(Standard ist 0); kann einen der Werte 0, 1, 2, 3 oder max (=3) annehmen
-p [Librarypfad]	Fügt einen Pfad zum Suchen von Libraries hinzu. Standardmäßig werden Libraries im Verzeichnis der FreeBASIC-Systemlibraries und im aktuellen Verzeichnis gesucht.
-vec [Wert]	Setzt die automatische Vektorisierungsebene (Standard ist 0), mögliche Werte sind 0 bzw. NONE, 1 und 2. *Die Option ist seit FreeBASIC 0.21 verfügbar.*)
-Wa [Optionen]	Übergabe von Optionen an den Assembler GAS(-gen gas oder -gen llvm). Optionen müssen durch Kommata getrennt werden.
-Wc [Optionen]	Übergabe von Optionen an GCC bei Verwendung von "-gen gcc" oder "-gen llvm". Optionen müssen durch Kommata getrennt werden.
-Wl [Optionen]	Übergabe von Optionen an den Linker LD. Optionen müssen durch Kommata getrennt werden.
-x [Name]	Setzt den Namen der EXE-Datei/Library einschließlich mit Erweiterung. Standardmäßig wird der Name der ersten Quell-Datei verwendet, die auf der Befehlszeile übergeben wurde. Beim Compilieren von Libraries muß die Datei das Präfix "lib" im Namen haben, sonst kann der Linker sie vielleicht nicht finden. Beim getrennten Compilieren und Linken darf diese Option nur vom Linker gesetzt werden. Die Option legt demnach den Namen der Enddatei fest, also den der EXE, LIB oder DLL.
-z gosub-setjmp	Benutzt setjmp/longjmp zum Implementieren von GOSUB

Informationen

-v	Aktiviert den ausführlichen Modus; der Compiler zeigt dann seine Aktionen Schritt für Schritt an.
-version	Zeigt die Programmversion des verwendeten Compilers an.
-w ["0"|"1"|"all"|"pedantic"]	Setzt die Mindest-Warnebene: "-w 0" zeigt alle Compiler-Warnungen (Standard), während "-w 1" sie versteckt.
-prefix [Pfad]	Setzt das Compiler-Präfix (den Ort, wo der Compiler die bin-, lib- und inc-Verzeichnisse sucht). Standardmäßig ist das der Pfad, in dem sich fbc befindet, falls das bestimmt werden kann.
-print host|target	Anzeige von Host-|Ziel-Systemname
-print x	Anzeige von Binär-|Bibliotheks-Dateiname(wenn bekannt)
-profile	Der Compiler erstellt nach Beendigung die Datei profile.txt, die alle Timing-Ergebnisse für Funktionsaufrufe enthält. Diese Technik nennt man auch Profiling.
-maxerr [Anzahl|"inf"]	Setzt die Anzahl der Fehler, die der Compiler finden darf, bevor der Vorgang abgebrochen wird. Standard ist 10. Wenn "inf" (unendlich) angegeben wird, macht der Compiler bis zum Ende der Quelldatei weiter. Dies ist nützlich, wenn eine IDE die Fehlermeldungen auswertet.
-map [Name]	Link-Map als Datei [Name] speichern

Voreinstellungen aus Dateien einlesen

@Dateiname	Häufig benutzte Compiler-Optionen lassen sich als Textdatei mit dem Namen [Dateiname] abspeichern und können unter Angabe von @Dateiname zum Compilieren benutzt werden. *Die Option ist seit FreeBASIC v0.21.0 verfügbar.*

Beispiele:

```
fbc meinprog.bas
```

Mit der DOS-Version von fbc: Compilieren und linken einer ausführbaren DOS-Datei MEINPROG.EXE

```
fbc -s gui meinprog.bas
```

Mit der Windows-Version von fbc: Compilieren und linken einer ausführbaren Windows-Datei meinprog.exe. Bei diesem Programm wird das Konsolenfenster ("MS-DOS Prompt") nicht angezeigt.

```
fbc -lib modul1.bas modul2.bas modul.bas -x libmeinelib.a
```

Compilieren und linken einer statischen Bibliothek libmeinelib.a aus den drei Quelldateien.

```
fbc -m haupt_modul -c haupt_modul.bas
```

Compilieren einer Objekt-Datei haupt_modul.o und Markieren als Eintrittspunkt.

```
fbc -c unter_modul.bas
```

Compilieren einer Objekt-Datei unter_modul.o

```
fbc -x anwendung.exe haupt_modul.o unter_modul.o
```

Linken einer ausführbaren anwendung.exe

Anmerkungen:
Bei Komplett-Downloads (die nicht vom FreeBASIC-Portal angeboten werden), also einem Komplettpaket mit Compiler und IDE, sollte nach der Installation zuerst geprüft werden, ob auch wirklich der aktuelle Compiler enthalten ist (s.o. fbc -version), sonst kann es böse Überraschungen geben, wenn eine veraltete Compiler-Version enthalten sein sollte.

Unterschiede zu QB:
Der FreeBASIC-Compiler erstellt 32-Bit- bzw. 64-Bit-Anwendungen anstelle von 16-Bit-Anwendungen oder nur interpretierten Programmen. Der Compiler ist für mehrere Plattformen verfügbar wie z. B. Linux, DOS und Windows. Auch auf der XBox sind spezielle FreeBASIC-Programme ausführbar. FreeBASIC ist 100% OpenSource und verwendet ausschließlich OpenSource-Software. Siehe auch die englischsprachige Entwicklerseite http://fbc.sf.net.
Der Compiler wird immer noch weiterentwickelt und verbessert.

Letzte Bearbeitung des Eintrags am 11.01.14 um 22:45:17
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

FB-Dialektformen

FreeBASIC-Referenz » Verschiedene Themen » **FB-Dialektformen**
Seit FreeBASIC v0.17 existiert die Kommandozeilenoption "-lang". Diese wird verwendet, um den Kompatibilitätsmodus auf verschiedene Versionen von FreeBASIC einzustellen. Die Option "-lang" benötigt einen Parameter, der angibt, mit welchem Kompatibilitätsmodus compiliert werden soll:

Option	Beschreibung
fb	FreeBASIC-Kompatibilität (Standard)
deprecated	Kompatibilität zu früheren Versionen von FreeBASIC
fblite	Kompatibilität zu FreeBASIC, aber mit einem Programmierstil, der kompatibler zu QBASIC ist
qb	QBASIC-Kompatibilität

Seit FreeBASIC v0.20 ist auch die Optionsangabe als Metabefehl #LANG "Sprachversion" am Anfang des Quelltextes möglich.
Die Option "-lang" wurde nötig, um Objektorientierung und andere Features in FreeBASIC zu ermöglichen, ohne die Abwärtskompatibilität zu QB oder zu Quellcodes für ältere Versionen von FreeBASIC aufgeben zu müssen oder viele verschiedene Compiler-Versionen entwickeln zu müssen. Um alte GW-BASIC- und QBasic-Quellcodes ohne größere Änderungen compilieren zu können, geben Sie die Option "-lang qb" an.

Beispiel: Um eine QBasic-Quelltextdatei mit dem Namen myprog.bas zu compilieren, geben Sie in der Befehlszeile

```
fbc myproc.bas -lang qb
```

ein (evtl. muss der Pfad zum Compiler ebenfalls angegeben werden).

Um Codes zu compilieren, die für FreeBASIC v0.16 oder älter geschrieben wurden, geben Sie die Kommandozeile "-lang deprecated" an. Der Modus "-lang fb" ist der Standard-Modus. Diese Option muss nicht explizit angegeben werden. "-lang fb" unterscheidet sich von "-lang deprecated" durch diese Dinge:

-lang fb
Nicht mehr unterstützt werden:

- Implizite Variablen: Alle Variablen müssen explizit deklariert werden, d. h. eine Variable kann nur benutzt werden, wenn sie zuvor mit DIM, COMMON, STATIC oder EXTERN deklariert wurde.
- Suffixe (! # $ % &): Suffixe sind nur noch in numerischen Konstanten erlaubt, allerdings wird empfohlen, stattdessen die Type-Casting-Funktionen (siehe CAST) oder die Suffixe f (single), d (double), ll (longint), ul (uinteger), ull (ulongint) zu verwenden.
- DEFxxx: Da keine impliziten Variablen mehr existieren, sind auch diese Schlüsselwörter obsolet; sie wurden entfernt. 'AS Typ' muss also **immer** angegeben werden (mit Ausnahme von Konstanten, bei denen der Datentyp automatisch gewählt wird; siehe dazu CONST.
- OPTION: Die Festlegung von Standards mithilfe von OPTION steht nicht mehr zur Verfügung.
    - OPTION BYVAL: Alle Variablen außer Arrays und UDTs werden standardmäßig BYVAL übergeben. Geben Sie die Kommandozeilenoption "-w pedantic" an, um Warnungen für jede Deklaration zu bekommen, bei der nicht explizit festgelegt wurde, ob eine Variable BYVAL oder BYREF übergeben wird.
    - OPTION NOKEYWORD: Benutzen Sie stattdessen #UNDEF.
    - OPTION ESCAPE: Um einen String mit ESCAPE-Sequenzen zu behandeln, stellen Sie einem String ein Ausrufezeichen ! voran. Benutzen Sie die Option "-w pedantic", um auf mögliche Escape-Sequenzen zu prüfen.
    - OPTION EXPLICIT wird nicht mehr benötigt, da Variablen immer explizit deklariert werden müssen.

    - ♦ OPTION DYNAMIC, OPTION STATIC: Arrays sind standardmäßig statisch. Um dynamische Arrays zu erstellen, verwenden Sie REDIM oder geben Sie bei der Dimensionierung keine Indizes an.
    - ♦ OPTION BASE: Wird die untere Arraygrenze bei der Deklaration nicht explizit angegeben, so wird der Wert 0 verwendet. Geben Sie ggf. die untere Arraygrenze explizit an.
    - ♦ OPTION PRIVATE: Prozeduren sind standardmäßig PUBLIC. Um private Prozeduren zu deklarieren, müssen diese explizit mit PRIVATE gekennzeichnet werden.
    - ♦ OPTION GOSUB, OPTION NOGOSUB: GOSUB ist nicht mehr zulässig. RETURN dient ausschließlich zur Rückkehr aus Prozeduren.
- Punkte in Symbolnamen sind nicht mehr zulässig. Verwenden Sie stattdessen NAMESPACE.
- ON...GOSUB und ON...GOTO wurden entfernt. Verwenden Sie SELECT CASE [AS CONST] anstelle dieser Anweisungen.
- RESUME wurde entfernt. Viele Anweisungen können als Funktionen genutzt werden, um Laufzeitfehler abzufragen (z. B. OPEN)
- In Kommentare eingebundene Meta-Befehle: Die Metabefehle '$DYNAMIC, '$STATIC, '$INCLUDE, '$INCLIB existieren nicht mehr. Statt '$INCLUDE und '$INCLIB benutzen Sie #INCLUDE bzw. #INCLIB.
- CALL und LET existieren nicht mehr und können einfach weggelassen werden. LET besitzt eine neue Bedeutung für die mehrfache Variablenzuweisung.
- Numerische Labels, die nur aus einer Zahl bestehen, werden nicht mehr unterstützt.
- Globale Symbole, die den Namen eines internen Schlüsselworts besitzen, können nur genutzt werden, wenn sie innerhalb eines NAMESPACE deklariert werden.

-lang deprecated
Unterstützt wird:
alles, das bereits in FreeBASIC v0.16 unterstützt wurde, mit Ausnahme von ON...GOSUB, ON...GOTO und ON...GOSUB (auch auf der Modulebene)

Nicht unterstützt werden:

- Klassen: Da Punkte in Symbolnamen erlaubt sind, ist es zu schwierig, dieses Feature zu implementieren.
- Operator overloading: Da Punkte in Symbolnamen erlaubt sind, ist es zu schwierig, dieses Feature zu implementieren.

Die Option "-lang deprecated" wurde bereitgestellt, um ein einfaches Upgrade des Codes von einer Entwicklungsversion in die nächste zu ermöglichen. Sie wird möglicherweise in einem Major-Release wie FreeBasic 1.0 entfernt.

-lang fblite
Unterstützt wird:
alles, was in "-lang deprecated" erlaubt ist, außer SCOPE-Blöcken: Variablen gehören dem aktuellen Scope (Programmblock) an.

-lang qb
Unterstützt wird:
alles, was in "-lang fb" nicht erlaubt ist. Außerdem:

- CALL kann verwendet werden, um Forward-Referencing bei Funktionen zu realisieren.
- Sichtbarkeit von Variablen: Alle Variablen, ob implizit oder explizit deklariert, werden wie in QB im Scope der Prozedur alloziert.

Nicht unterstützt wird:

- Multi-Threading: Die Anweisungen THREADxxx, MUTEXxxx und CONDxxx können nicht eingesetzt werden. Siehe Multithreading.
- Klassen und NAMESPACEs
- Prozedur- und Operator-Overloading (siehe OVERLOAD, OPERATOR)
- SCOPE-Blöcke
- EXTERN-Blöcke
- Variablen-Initiatoren: Alle Variablen werden auf den Scope der Prozedur verschoben (wie in QB), sodass lokale Initiatoren schwierig zu implementieren sind.

FB-eigene Befehle im QB-Kompatiblitätsmodus

Befehle, die in FreeBASIC neu hinzugekommen sind, können auch im QB-Kompatiblitätsmodus benutzt werden: Versehen Sie diese mit einem doppelten Unterstrich am Wortanfang, um sie zu benutzen. Beispiel: __INSTRREV

Beachten Sie, dass bei Befehlen, deren Verhalten sich in FreeBasic geändert hat (z. B. SLEEP oder INKEY), auch das FreeBASIC-Verhalten übernommen wird. So wartet __SLEEP 1 im QB-Kompatibilitätsmodus eine Millisekunde, während SLEEP 1 - wie aus QB gewohnt - eine Sekunde wartet. Bei __INKEY wird dementsprechend CHR(255) statt CHR(0) als erweitertes Zeichen benutzt.

Letzte Bearbeitung des Eintrags am 20.06.14 um 01:08:07
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

Obsolete Schlüsselwörter

Die folgenden Befehle werden nicht unterstützt, weil sie spezifisch für QuickBASIC waren; sie werden als bisher unwichtig eingestuft, da es effizientere Programmiermöglichkeiten gibt oder sie ihre Umsetzung nicht wert sind. Bei einigen Befehle wurden die Gründe angegeben, warum sie nicht unterstützt werden, oder es wird eine mögliche Alternative vorgeschlagen.

CALL ABSOLUTE
Grund: Die Anweisung wird wegen des Inline-Assembler nicht mehr benötigt.

CALL INTERRUPT
Grund: Die Anweisung ist nur unter DOS auf Intel-x86-Plattformen verfügbar. In anderen Betriebssystemen gibt es keinen Grund, die Anweisung zu unterstützen.

COM
Grund: Programme, die auf Benutzerebene laufen, können auf IO-Ports nicht direkt zugreifen.

CVDMBF / CVSMBF
Alternative: binäre Speicherkopien können mit CVD bzw. CVS umgesetzt werden.

DEF FN
Alternative: Funktionen können nur über FUNCTION deklariert werden.

DEF SEG

ERDEV / ERDEV$

FIELD
Alternative: Die Anweisung kann über binären Dateizugriff ersetzt werden; siehe BINARY.

FILES
Alternative: Die Anweisung kann durch DIR ersetzt werden.

IOCTL / IOCTL$

KEY [ON|OFF|(Modus)]
Grund: Die Interrupt-Routine ON KEY wird nicht unterstützt.

LIST
Grund: Im compilierten Programm ist der Quellcode nicht mehr auslesbar.

LPOS / LPRINT (steht nur in der Dialektform -lang qb zur Verfügung)
Alternative: Mit OPEN LPT kann ein Druckerport geöffnet werden.

MKDMBF$ / MKSMBF$
Alternative: binäre Speicherkopien können mit MKD bzw. MKS umgesetzt werden.

ON xxx
Die Interrupt-Befehle ON COM [(n)], ON KEY(n), ON PEN, ON PLAY(n), ON STRIG(n), ON TIMER(n) und ON UEVENT werden nicht unterstützt.

PEN [ON|OFF|STOP|(Modus)]
Grund: Die Verwendung von Lichtgriffel ist veraltet.

PLAY
Grund: Musikausgabe ist zu plattformspezifisch. Stattdessen können Musik-Bibliotheken verwendet werden, die wesentlich mehr Funktionalität bieten.

SETMEM

SIGNAL
Grund: Diese Funktion ist nur unter DOS auf Intel-x86-Plattformen verfügbar und kann durch effizientere Netzwerk-Modelle ersetzt werden..

SOUND
Grund: Musikausgabe ist zu plattformspezifisch. Stattdessen können Musik-Bibliotheken verwendet werden, die wesentlich mehr Funktionalität bieten.

STICK / STRIG (steht nur in der Dialektform -lang qb zur Verfügung)
Alternative: Zur Abfrage des Joystick kann GETJOYSTICK verwendet werden.

TROFF / TRON
Tracing kann über die Compileroption -g in Zusammenarbeit mit Debugger-Programmen umgesetzt werden.

UEVENT [ON|OFF|STOP]
Grund: Die Interrupt-Routine ON UEVENT wird nicht unterstützt.

VARPTR$

VARSEG

Letzte Bearbeitung des Eintrags am 16.02.13 um 00:01:40
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

Serial Numbers

FreeBASIC-Referenz » Verschiedene Themen » **Serial Numbers**
Serial Numbers sind Zahlen, die ein Datum und eine Zeit im selben Format enthalten, das schon bei QBX PDS und VBDOS verwendet wurde. Sie werden vorwiegend dazu verwendet, um die Zeit zwischen zwei Datums- bzw. Zeitangaben zu berechnen.

Eine Serial Number setzt sich aus zwei Teilen zusammen. Der ganzzahlige Anteil gibt die Anzahl der Tage an, die seit dem 30.12.1899 um 00:00 vergangen sind. Der Nachkommateil gibt die Uhrzeit an. Dieser Wert bedeutet, wie viel Prozent des Tages bereits vergangen ist. Der Nachkommateil 0.5 bedeutet z. B., dass die Hälfte des Tages vergangen ist; es ist also 12:00 Uhr.

Da diese Serials auf sehr genaue Nachkommawerte angewiesen sind, werden sie als DOUBLE ausgegeben. Würde man eine Serial Number in einer INTEGER-Variablen speichern, ginge die Information über die Uhrzeit verloren.

Serial Numbers sind in FreeBASIC nicht, wie in VBDOS, auf Datumsangaben zwischen 1753 und 2078 begrenzt. Um Serial-Funktionen in FreeBASIC nutzen zu können, müssen Sie die Datei datetime.bi in Ihren Quellcode einbinden, z. B. mit #INCLUDE. Alternativ dazu können Sie auch die Datei vbcompat.bi einbinden, da diese die datetime.bi automatisch in Ihr Progamm lädt.

Eine Serial Number kann erstellt werden mit:

- NOW
- TIMESERIAL - DATESERIAL
- TIMEVALUE - DATEVALUE

Die Funktionen YEAR, MONTH, WEEKDAY, DAY, HOUR, MINUTE und SECOND ermöglichen es, die einzelnen Komponenten der Serial Number wiederherzustellen. Die Funktion FORMAT unterstützt Formatierungsausdrücke, die es ermöglichen, eine Serial Number in einem Format ausgeben, das für einen Menschen lesbar ist.

Unterschiede zu QB:

- Time Serials existieren erst seit QBX 7.1.
- Serial Numbers in FreeBASIC sind nicht wie in VBDOS auf Datumsangaben zwischen 1753 und 2078 begrenzt.

Unterschiede zu früheren Versionen von FreeBASIC: existiert seit FreeBASIC v0.15

Siehe auch:
NOW, DATESERIAL, DATEVALUE, TIMESERIAL, TIMEVALUE, YEAR, MONTH, DAY, WEEKDAY, HOUR, MINUTE, SECOND, MONTHNAME, WEEKDAYNAME, DATEDIFF, DATEPART, DATEADD, FORMAT, Datum und Zeit

Letzte Bearbeitung des Eintrags am 16.02.13 um 00:36:39
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

Fehler-Behandlung in FreeBASIC

FreeBASIC-Referenz » Verschiedene Themen » **Fehler-Behandlung in FreeBASIC**
Behandlung von Laufzeitfehlern

FreeBASIC kann Laufzeitfehler auf folgende Arten behandeln:

- Normalerweise reagiert ein Programm nicht auf die auftretenden Fehler - sie werden einfach ignoriert und der Code wird weiter ausgeführt. Das Programm sollte mögliche Fehler über die Variable ERR in der nächsten Zeile abfangen.
- Einige Anweisungen können auch als Funktion eingesetzt werden und geben dann ggf. die Fehlernummer zurück (siehe z. B. OPEN (Funktion).
- Wenn mit den Compileroptionen -e, -ex oder -exx kompiliert wurde, benutzt FreeBASIC eine QB-ähnliche Fehlerbehandlung.

Wichtig: Verursacht ein FreeBASIC-Programm einen allgemeinen Schutzfehler (etwa durch Zugriffe außerhalb des eigenen Speicherbereichs), wird die Programmausführung sofort vom Betriebssystem beendet. Ein solcher Fehler lässt sich mit den Fehlerbehandlungsmethoden ("Error handling") von FreeBASIC nicht abfangen und benutzerdefiniert behandeln.

Standardfehlerbehandlung

Das Standard-Verhalten von FreeBASIC ist, bei einem Fehler die Variable ERR zu setzen und fortzufahren.

```
' Im Folgenden wird davon ausgegangen, dass die angegebene Datei
xzxwz.zwz
' nicht existiert. Dann wird nämlich in ERR ein Fehler vermeldet.
Open "xzxwz.zwz" For Input As "hlkw0">Print Err
Sleep
```

Das Programm hält, obwohl die Datei nicht existiert, nicht an, sondern setzt die Variable ERR und macht weiter. Der Fehler könnte in der nächsten Zeile ausgewertet werden.

Achtung: Bei der Verwendung von PRINT wird ERR anschließend auf 0 zurückgesetzt. Gegebenenfalls sollte ihr Wert zuvor in eine andere Variable kopiert werden.

Einige Ein- und Ausgabe-Routinen wie OPEN und PUT können als Funktion eingesetzt werden und geben eine Fehlernummer zurück. Wenn kein Fehler aufgetreten ist, wird 0 zurückgegeben.

```
Print Open ("xzxwz.zwz" For Input As "hlzahl">1)
Sleep
```

QB-ähnliche Fehlerbehandlung

Wenn mit den Schaltern -e, -ex oder -exx kompiliert wurde, wird vom Programm erwartet, dass eine QB-ähnliche Fehlerbehandlung geschieht. Erfolgt keine Fehlerbehandlung, stoppt das Programm mit einem Fehler.

Wichtig: Wenn die QB-ähnliche Fehlerbehandlung benutzt wird, muss der Programmierer sicherstellen, dass ALLE Fehlermöglichkeiten abgefangen werden. Beispielsweise erzeugt GETMOUSE einen Fehler, wenn der Cursor das aktuelle Fenster verlässt.

```
On Error Goto Ooops
Open "xzxwz.zwz" For Input As "hlkw0">On Error Goto 0
Sleep
End

Ooops:
Print Err
Sleep
End
```

ON ERROR setzt eine Fehlerbehandlungsroutine, die vom Programm aufgerufen wird, wenn ein Fehler gefunden wurde.
ON ERROR GOTO 0 schaltet diese Fehlerbehandlung ab.
In den Dialektformen -lang qb und -lang deprecated steht außerdem der Befehl RESUME zur Verfügung, um nach einem Fehler wieder mit der Programmausführung fortzufahren.

Wenn keine Fehlerbehandlungsroutine benutzt wird und ein Fehler auftritt, stoppt das Programm und sendet eine Fehlermeldung an die Konsole.

```
Aborting program due to runtime error 2 (file not found)
```

(Abbruch wegen Laufzeitfehler 2 (Datei nicht gefunden))

Die "globale" Fehlerbehandlungsroutine kann wie in QB am Programmende stehen. Bei ON LOCAL ERROR kann eine lokale Fehlerbehandlung auf Prozedurebene erfolgen. Dazu muss das Unterprogramm (SUB oder FUNCTION) eine Codesequenz zur Fehlerbehandlung besitzen, die im Fehlerfall mittels GOTO angesprungen wird.

```
Declare Sub foo
foo
Sleep

Sub foo
    Dim filename As String
    filename = ""
    On Local Error Goto Ooops
    Open "" For Input As "hlkw0">Print "Kein Fehler"
    On Local Error Goto 0
    Exit Sub
  Ooops:
    Print "Fehler " & Err & " in Function " & *Ermn & " in Zeile " & Erl
End Sub
```

Erläuterung:
Beim Schalter -e muss die Fehlerbehandlungsroutine das Programm beenden, unabhängig vom verwendeten Dialekt.
Wird das Programm mit der Compileroption -ex und -lang qb compiliert, kann die Error-Handling-Routine mit der Anweisung RESUME (wiederholt die letzte Anweisung, die den Fehler hervorrief) oder RESUME NEXT (fährt mit der nächsten Anweisung fort) die Programmausführung nach Behandlung des Fehlers fortsetzen.
Wird compiliert, ohne eine Compileroption zur Fehlerbehandlung zu verwenden, dann wird der Fehler ignoriert.

Fehlernummern

FreeBASIC kennt folgende **Laufzeit**-Fehlercodes:

Nr.	Meldung	Übersetzung
0	No error	kein Fehler
1	Illegal function call	ungültiger Funktionsaufruf
2	File not found signal	Datei nicht gefunden*
3	File I/O error	Datei-Ein-/Ausgabe-Fehler
4	Out of memory	Zu wenig Speicher
5	Illegal resume	ungültiges RESUME
6	Out of bounds array access	Array-Indexüberschreitung
7	Null Pointer Access	Nullpointer-Zugriff
8	No privileges	fehlende Berechtigungen
9	interrupted signal	Unterbrechung
10	illegal instruction signal	ungültige Anweisung
11	floating point error signal	Gleitkommafehler
12	segmentation violation signal	Speicherzugriffsfehler
13	Termination request signal	Beendigungsanforderung
14	abnormal termination signal	unnormale Beendigung
15	quit request signal	Anfragesignal beendet
16	return without gosub	RETURN ohne GOSUB
17	end of file	Dateiende

*) Der Laufzeitfehler 2 (File not found) kann auch auftreten, wenn z. B. mit ENCODING "UTF-8" eine Datei geöffnet werden soll, die nicht UTF-8-codiert ist oder bei der das Byte Order Mark (BOM) nicht gesetzt ist.

Ein Bereich für Benutzer-Fehlermeldungen ist nicht vorgegeben. Wenn also ERROR benutzt wird, um einen benutzerdefinierten Fehlercode zu setzen, ist es vernünftiger, hohe Werte zu benutzen, um Überschneidungen mit den eingebauten Fehlermeldungen zu vermeiden, falls die obige Liste zukünftig einmal erweitert werden sollte.

Siehe auch:
ON ERROR, ERR (Funktion), ERFN, ERMN, ERROR, Fehlerbehandlung, Debugging

Letzte Bearbeitung des Eintrags am 09.02.13 um 13:46:03
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

Funktionen der CRT

FreeBASIC-Referenz » Verschiedene Themen » **Funktionen der CRT**
Funktionen der C Standard Library

FreeBASIC erlaubt das Einbinden der Funktionen der C-Standard-Bibliothek. Hier finden Sie eine alphabetische Aufstellung der Funktionen, gefolgt von einer Auflistung gruppiert nach ihrer Funktionalität.

Beschreibung:
Die Kommentar-Spalte enthält eine kurze Beschreibung der Funktion. Die Liste ist nicht vollständig, gibt aber einen guten Überblick über die meisten Funktionen der C Runtime Library. Die angegebenen Funktionen werden sicher von FreeBASIC unterstützt. Für weitere Informationen siehe z. B. die C-Referenz auf cplusplus.com (engl.). Eine deutschsprachige Beschreibung vieler Funktionen inkl. verständlichen Erläuterungen und vielen Beispielen findet man auch in den Kapiteln des frei zugänglichen Buchs "C von A bis Z".

Hinweis: Die folgenden Deklarationen (zweite Tabellenspalte) sind nicht die offiziell in FreeBASIC enthaltenen, sie sollten aber genügend Informationen geben, um die Funktionen anwenden zu können.

Die Datei-Spalte beinhaltet den Namen der Datei, die eingebunden werden muss. Dies geschieht mit dem Befehl #INCLUDE am Anfang des Programmcodes. Sollte die angegebene Datei nicht eingebunden werden, wird das Programm entweder nicht compilieren, oder es wird zwar normal compilieren, aber danach falsche Werte ausgeben. Alle Header der C Runtime befinden sich im Unterordner crt. So muss, um etwa die math.bi einzubinden, #INCLUDE "crt/math.bi" bzw. #INCLUDE "crt\math.bi" angegeben werden. Alternativ kann man mit #INCLUDE "crt.bi" alle Dateien einbinden.

Die Deklaration-Spalte enthalten folgende Informationen:

- den Namen der Funktion;
- die benötigten Parameter der Funktion in Klammern, zusammen mit dem Datentyp des Parameters;
- den Datentyp des Rückgabewerts der Funktion.

So bedeutet z. B. atoi(a AS ZSTRING PTR) AS INTEGER, dass die Funktion atoi einen Wert vom Typ INTEGER zurückgibt und einen ZSTRING PTR als Parameter benötigt.

Alphabetische Liste

Name	Deklaration (mit Parametern)	Datei	Kommentar
abs_	abs_(n AS INTEGER) AS INTEGER	stdlib.bi	Gibt den Absolutwert zurück (also den nicht-negativen Wert).
acos_	acos_(a AS DOUBLE) AS DOUBLE	math.bi	Gibt den Arkuskosinus zurück (Winkel in Bogenmaß).
asin_	asin_(a AS DOUBLE) AS DOUBLE	math.bi	Gibt den Arkussinus zurück (Winkel in Bogenmaß).
atan_	atan_(a AS DOUBLE) AS DOUBLE	math.bi	Gibt den Arkustangens zurück (Winkel in Bogenmaß).
atan2_	atan2_(y AS DOUBLE, x AS DOUBLE) AS DOUBLE	math.bi	Gibt den Arkustangens zurück (Ankathete als x, Gegenkathete als y).
atoi		stdlib.bi	Konvertiert einen ZString mit Ziffern in eine Ganzzahl (Integer).

	atoi(s AS ZSTRING PTR) AS INTEGER		
atof	atof(s AS ZSTRING PTR) AS DOUBLE	stdlib.bi	Konvertiert einen ZString mit Ziffern in eine Gleitkommazahl (Double).
calloc	calloc(NumElts AS INTEGER, EltSiz AS INTEGER) AS ANY PTR	stdlib.bi	Fordert Speicher an. Gibt einen Pointer auf einen Puffer zurück, der Platz für ein Array bestehend aus NumElts Elementen der Größe EltSiz Bytes bietet.
ceil	ceil(d AS DOUBLE) AS DOUBLE	math.bi	Rundet auf die nächsthöhere Ganzzahl.
clearerr	clearerr(s AS FILE PTR)	stdio.bi	Setzt den Error-Indikator eines Filestreams (lesen oder schreiben) zurück.
cos_	cos_(ar AS DOUBLE) AS DOUBLE	math.bi	Gibt den Kosinus eines Winkels in Bogenmaß zurück.
cosh	cosh(x AS DOUBLE) AS DOUBLE	math.bi	Gibt den Kosinus Hyperbolicus eines Winkels in Bogenmaß zurück.
div	div(num AS INTEGER, denom AS INTEGER) AS div_t	stdlib.bi	Gibt den Quotienten und Rest einer Division als Struktur div_t zurück.
ecvt	ecvt(x AS DOUBLE) AS ZSTRING PTR	math.bi	Konvertiert eine Zahl in einen ZString.
exit_	exit_(status AS INTEGER)	stdlib.bi	Beendet das Programm. Dabei werden alle Dateipuffer geleert, offene Dateien geschlossen und von atexit() aufgerufene Funktionen ausgeführt.
exp_	exp_(a AS DOUBLE) AS DOUBLE	math.bi	Gibt den Wert von e^a zurück (Umkehrfunktion des natürlichen Logarithmus).
fabs	fabs(d AS DOUBLE) AS DOUBLE	math.bi	Gibt den Absolutwert (d. h. den nicht-negativen Wert) einer Double-Zahl zurück.
fclose	fclose(s AS FILE PTR) AS FILE PTR	stdio.bi	Schließt eine Datei. Gibt bei Erfolg 0 zurück, andernfalls EOF.
feof	feof(s AS FILE PTR) AS INTEGER	stdio.bi	Gibt den Wert des Dateiende-Indikators zurück. Dieser hat den Wert 0, solange das Dateiende noch nicht erreicht wurde. Der Indikator setzt sich automatisch zurück, kann aber auch manuell mittels clearerr() zurückgesetzt werden.
ferror	ferror(s AS FILE PTR) AS INTEGER	stdio.bi	Gibt die Fehlernummer für einen Stream zurück (0 wenn kein Fehler aufgetreten ist). Die Fehlernummer wird durch clearerr() oder rewind() zurückgesetzt.
fflush	fflush(s AS FILE PTR) AS INTEGER	stdio.bi	Leert einen Stream (verwenden Sie stdin, um den Stream der Tastatur zu leeren). Gibt bei Erfolg 0 zurück.
fgetc	fgetc(s AS FILE PTR) AS INTEGER	stdio.bi	Liest einzelnes Zeichen (als ASCII-Code) vom übergebenen Stream (stdin zum Lesen von der Tastatur).
fgetpos	fgetpos(s AS FILE PTR, c AS fpos_t PTR) AS INTEGER	stdio.bi	Speichert die Position des Dateizeigers von Stream s an der Stelle, auf die der Pointer c zeigt.
fgets	fgets(b AS ZSTRING PTR, n AS INTEGER, s AS FILE PTR) AS ZSTRING PTR	stdio.bi	Liest bis zu n-1 Zeichen von Stream s in den Buffer b.
floor	floor(d AS DOUBLE) AS DOUBLE	math.bi	Gibt die größte ganze Zahl zurück, die kleiner oder gleich d ist (Abrunden der Zahl d).

fmod	fmod(x AS DOUBLE, y AS DOUBLE) AS DOUBLE	math.bi	Errechnet den Rest der Division x / y.
fopen	fopen(file AS ZSTRING PTR, mode AS ZSTRING PTR) AS FILE PTR	stdio.bi	Öffnet eine Datei. Übergeben wird der DOS-Dateiname und eine Modus-Zeichenkette. Gültige Werte für den Modus sind "r" (read) für lesenden Zugriff, "w" (write) für schreibenden Zugriff, "a" (append) für schreibenden Zugriff am Dateiende und "r+" (Datei muss existieren) bzw. "w+" (Datei wird neu angelegt) für lesenden und schreibenden Zugriff. Zusätzlich kann ein angehängtes "b" die Verwendung des Binärmodus signalisieren.
fprintf	fprintf(s AS FILE PTR, fmt AS ZSTRING PTR, ...) AS INTEGER	stdio.bi	Schreibt eine Zeichenkette mit Format fmt in einen Stream s, wobei die Zeichen % mit passenden Argumenten der Parameterliste ersetzt werden.
fputc	fputc(c AS INTEGER, s AS FILE PTR) AS INTEGER	stdio.bi	Gibt ein einzelnes Zeichen c in den Stream s aus.
fputs	fputs(b AS ZSTRING PTR, s AS FILE PTR) AS INTEGER	stdio.bi	Sendet die Zeichenkette in b an den Stream s; gibt 0 zurück, wenn die Operation fehlschlägt.
fread	fread(buf AS ANY PTR, b AS size_t, c AS size_t, s AS FILE PTR) AS INTEGER	stdio.bi	Liest c Datenelemente der Größe von b Bytes aus der Datei s in den Buffer buf. Gibt die Anzahl der Elemente zurück, die tatsächlich gelesen wurden.
free	free(p AS ANY PTR)	stdlib.bi	Gibt den für einen Zeiger p angeforderten Speicher zur anderwertigen Verwendung wieder frei.
freopen	freopen(file AS ZSTRING PTR, mode AS ZSTRING PTR, s AS FILE PTR) AS FILE PTR	stdio.bi	Öffnet eine Datei, um einen Stream umzuleiten. Z. B. freopen("myfile", "w", stdout) öffnet die Datei "myfile" und leitet die Standardausgabe dorthin um.
frexp	frexp(x AS DOUBLE, p AS INTEGER PTR) AS DOUBLE	math.bi	Berechnet den Wert m so, dass x=m*2^exponent. p ist ein Pointer auf m; exponent ist der Rückgabewert der Funktion.
fscanf	fscanf(s AS FILE PTR, fmt AS ZSTRING PTR, ...) AS INTEGER	stdio.bi	Liest vom Stream s Daten für jedes % Zeichen in fmt mit zugehörigem Zeiger in der Parameterliste.
fseek	fseek(s AS FILE PTR, offset AS INTEGER, origin AS INTEGER) AS INTEGER	stdio.bi	Setzt die Position innerhalb einer Datei. origin enthält einen der Werte SEEK_GET (0), SEEK_CUR (1) oder SEEK_END (2) und gibt an, ob der Offset vom Dateianfang, von der aktuellen Position oder vom Dateiende ausgeht.
fsetpos	fsetpos(s AS FILE PTR, p AS fpos_t PTR) AS INTEGER	stdio.bi	Setzt die Position innerhalb eines Streams s auf die von p gezeigte Position.
ftell	ftell(s AS FILE PTR) AS LONG	stdio.bi	Ermittelt die Position innerhalb eines Streams s.
fwrite	fwrite(buf AS ANY PTR, b AS INTEGER, c AS INTEGER, s AS FILE PTR) AS INTEGER	stdio.bi	Schreibt c Datenblöcke mit jeweils b Bytes aus dem Puffer buf in die Datei s. Gibt die Anzahl der tatsächlich geschriebenen Datenblöcke zurück.
getc		stdio.bi	

	getc(s AS FILE PTR) AS INTEGER		Liest ein einzelnes Zeichen (als ASCII-Wert) aus dem Stream s (stdin für das Lesen von der Tastatur).
getchar	getchar() AS INTEGER	stdio.bi	Liest ein einzelnes Zeichen aus der Standardeingabe.
gets	gets(b AS ZSTRING PTR) AS ZSTRING PTR	stdio.bi	Liest eine Zeichenkette aus der Standardeingabe bis zu einem \n oder EOF.
hypot	hypot(x AS DOUBLE, y AS DOUBLE) AS DOUBLE	math.bi	Errechnet die Hypotenuse aus den Katheten x und y.
isalnum	isalnum(c AS INTEGER) AS INTEGER	ctype.bi	Gibt einen Wert ungleich 0 zurück, wenn es sich beim Zeichen c um einen Buchstaben oder eine Ziffer handelt.
isalpha	isalpha(c AS INTEGER) AS INTEGER	ctype.bi	Gibt einen Wert ungleich 0 zurück, wenn es sich beim Zeichen c um einen Buchstaben handelt.
iscntrl	iscntrl(c AS INTEGER) AS INTEGER	ctype.bi	Gibt einen Wert ungleich 0 zurück, wenn es sich beim Zeichen c um ein Steuerzeichen handelt.
isdigit	isdigit(c AS INTEGER) AS INTEGER	ctype.bi	Gibt einen Wert ungleich 0 zurück, wenn es sich beim Zeichen c um eine Ziffer handelt.
isgraph	isgraph(c AS INTEGER) AS INTEGER	ctype.bi	Gibt einen Wert ungleich 0 zurück, wenn es sich beim Zeichen c um ein darstellbares Zeichen handelt.
islower	islower(c AS INTEGER) AS INTEGER	ctype.bi	Gibt einen Wert ungleich 0 zurück, wenn es sich beim Zeichen c um einen Kleinbuchstaben handelt.
isprint	isprint(c AS INTEGER) AS INTEGER	ctype.bi	Gibt einen Wert ungleich 0 zurück, wenn es sich beim Zeichen c um ein druckbares Zeichen handelt.
ispunct	ispunct(c AS INTEGER) AS INTEGER	ctype.bi	Gibt einen Wert ungleich 0 zurück, wenn es sich beim Zeichen c um ein Satzzeichen handelt.
isspace	isspace(c AS INTEGER) AS INTEGER	ctype.bi	Gibt einen Wert ungleich 0 zurück, wenn es sich beim Zeichen c um ein Whitespace (Leerzeichen, Tabulator, Zeilenumbruch) handelt.
isupper	isupper(c AS INTEGER) AS INTEGER	ctype.bi	Gibt einen Wert ungleich 0 zurück, wenn es sich beim Zeichen c um einen Großbuchstaben handelt.
isxdigit	isxdigit(c AS INTEGER) AS INTEGER	ctype.bi	Gibt einen Wert ungleich 0 zurück, wenn es sich beim Zeichen c um eine Hex-Ziffer handelt (0 bis F oder f).
ldexp	ldexp(x AS DOUBLE, n AS INTEGER) AS DOUBLE	math.bi	Gibt den Wert von x * 2^n zurück.
ldiv	ldiv(num AS LONG, denom AS LONG) AS ldiv_t	stdlib.bi	Gibt Quotient und Rest einer Division als Struktur ldiv_t zurück.
log_	log_(a AS DOUBLE) AS DOUBLE	math.bi	Gibt den natürlichen Logarithmus des Parameterwerts zurück.
log10	log10(a AS DOUBLE) AS DOUBLE	math.bi	Gibt den Zehnerlogarithmus des Parameterswerts zurück.
malloc	malloc(bytes AS INTEGER) AS ANY PTR	stdlib.bi	Fordert Speicher an. Gibt einen Pointer auf einen Puffer der angeforderten Größe zurück.
modf	modf(d AS DOUBLE, p AS DOUBLE PTR) AS DOUBLE	math.bi	Gibt den Nachkommateil einer Gleitkommazahl d zurück. Der Zeiger p zeigt auf den ganzzahligen Anteil als Gleitkommazahl.
perror		stdio.bi	Schreibt die Nachricht mess in die Standard-Fehlerausgabe.

	perror(mess AS ZSTRING PTR)		
pow	pow(x AS DOUBLE, y AS DOUBLE) AS DOUBLE	math.bi	Gibt den Wert von x^y zurück.
pow10	pow10(x AS DOUBLE) AS DOUBLE	math.bi	Gibt den Wert von 10^x zurück (Umkehrfunktion zu log10()).
printf	printf(fmt AS ZSTRING PTR, ...) AS INTEGER	stdio.bi	Schreibt eine Zeichenkette mit Format fmt auf die Standardausgabe (stdout), wobei die Zeichen % mit passenden Argumenten der Parameterliste ersetzt werden.
putc	putc(c AS INTEGER, s AS FILE PTR) AS INTEGER	stdio.bi	Schreibt ein einzelnes Zeichens c in den Stream s.
putchar	putchar(c AS INTEGER) AS INTEGER	stdio.bi	Schreibt ein einzelnes Zeichens c in die Standardausgabe.
puts	puts(b AS ZSTRING PTR) AS INTEGER	stdio.bi	Sendet die Zeichenkette in b an die Standardausgabe; gibt 0 zurück, wenn die Operation fehlschlägt.
rand	rand() AS INTEGER	stdlib.bi	Gibt eine Pseudozufallszahl zurück. Der Zufallsgenerator muss zuvor mit srand initialisiert werden.
realloc	realloc(p AS ANY PTR, newsize AS size_t) AS ANY PTR	stdlib.bi	Ändert die Größe des allozierten Speicherblocks, auf den p zeigt.
rewind	rewind(s AS FILE PTR)	stdio.bi	Löscht die Error-Flags eines Datei-Streams (lesen oder schreiben). Dies ist notwendig, bevor man eine veränderte Datei liest.
scanf	scanf(fmt AS ZSTRING PTR, ...) AS INTEGER	stdio.bi	Liest von der Standardeingabe (stdin) Daten für jedes % Zeichen in fmt mit zugehörigem Zeiger in der Parameterliste.
sin_	sin_(ar AS DOUBLE) AS DOUBLE	math.bi	Gibt den Sinus eines Winkels in Bogenmaß zurück.
sinh	sinh(x AS DOUBLE) AS DOUBLE	math.bi	Gibt den Sinus Hyperbolicus eines Winkels in Bogenmaß zurück.
sprintf	sprintf(p AS ZSTRING PTR, fmt AS ZSTRING PTR, ...) AS INTEGER	stdio.bi	Schreibt eine Zeichenkette mit Format fmt in p, wobei die Zeichen % mit passenden Argumenten der Parameterliste ersetzt werden.
sqrt	sqrt(a AS DOUBLE) AS DOUBLE	math.bi	Gibt die Quadratwurzel des Parameterwerts zurück. Bei einem negativen Wert a wird ein *domain error* ausgelöst.
srand	srand(seed AS UINTEGER)	stdlib.bi	Initialisiert den Pseudozufallsgenerator. Ein möglicher Initialwert für seed ist die aktuelle Zeit.
sscanf	sscanf(b AS ZSTRING PTR, fmt AS ZSTRING PTR, ...) AS INTEGER	stdio.bi	Liest vom Buffer b Daten für jedes % Zeichen in fmt mit zugehörigem Zeiger in der Parameterliste.
strcat	strcat(s1 AS ZSTRING PTR, s2 AS ZSTRING PTR) AS ZSTRING PTR	string.bi	Hängt den s2 an s1.
strchr	strchr(s AS ZSTRING PTR, c AS INTEGER) AS ZSTRING PTR	string.bi	Gibt einen Zeiger auf das erste Auftreten von c in s zurück oder NULL, falls kein solches gefunden wurde.
strcmp	strcmp(s1 AS ZSTRING PTR, s2 AS ZSTRING	string.bi	Vergleicht den s2 mit s1. Gibt 0 bei Gleichheit oder die (vorzeichenbehaftete) Differenz der ASCII Werte der ersten

	PTR) AS INTEGER		ungleichen Zeichen zurück.
strcpy	strcpy(s1 AS ZSTRING PTR, s2 AS ZSTRING PTR) AS ZSTRING PTR	string.bi	Kopiert s2 in s1.
strcspn	strcspn(s1 AS ZSTRING PTR, s2 AS ZSTRING PTR) AS INTEGER	string.bi	Gibt die Zeichenanzahl von s1 zurück, bis zu welcher kein Zeichen aus s2 vorkommt.
strerror	strerror(n AS INTEGER) AS ZSTRING PTR	string.bi	Gibt eine textuelle Fehlermeldung für den übergebenen Fehlercode zurück.
strlen	strlen(s AS ZSTRING PTR) AS INTEGER	string.bi	Gibt die Länge eines nullterminierten Strings in Bytes zurück (ohne Null mitzuzählen).
strncat	strncat(s1 AS ZSTRING PTR, s2 AS ZSTRING PTR, n AS INTEGER) AS ZSTRING PTR	string.bi	Hängt n Bytes von s2 an s1.
strncmp	strncmp(s1 AS ZSTRING PTR, s2 AS ANY PTR, n AS INTEGER) AS INTEGER	string.bi	Vergleicht n Bytes in s2 mit denselben in s1. Gibt 0 bei Gleichheit oder die (vorzeichenbehaftete) Differenz der ASCII Werte der ersten ungleichen Zeichen zurück.
strncpy	strncpy(s1 AS ZSTRING PTR, s2 AS ZSTRING PTR, n AS INTEGER) AS ZSTRING PTR	string.bi	Kopiert n Bytes von s2 nach s1.
strpbrk	strpbrk(s1 AS ZSTRING PTR, s2 AS ZSTRING PTR) AS ZSTRING PTR	string.bi	Gibt einen Zeiger auf das erste Zeichen in s1 zurück, das auch in s2 enthalten ist.
strrchr	strrchr(s AS ZSTRING PTR, c AS INTEGER) AS ZSTRING PTR	string.bi	Gibt einen Zeiger auf das letzte Auftreten von c in s zurück oder NULL, falls kein solches gefunden wurde.
strspn	strspn(s1 AS ZSTRING PTR, s2 AS ZSTRING PTR) AS INTEGER	string.bi	Gibt die Zeichenanzahl von s1 zurück, bis zu welcher kein Zeichen vorkommt, das nicht in s2 vorhanden ist.
strstr	strstr(s1 AS ZSTRING PTR, s2 AS ZSTRING PTR) AS ZSTRING PTR	string.bi	Sucht die Position von s2 innerhalb s1 und gibt einen Zeiger auf das erste Zeichen zurück.
strtod	strtod(s AS ZSTRING PTR, p AS ZSTRING PTR) AS DOUBLE	stdlib.bi	Konvertiert einen ZString nach Double, sofern der ZString eine Zahl gültigen Formats enthält.
strtok	strtok(s1 AS ZSTRING PTR, s2 AS ZSTRING PTR) AS ZSTRING PTR	string.bi	Gibt Zeiger auf die durch ein Zeichen in s2 getrennten Teile von s1 zurück.
System	System(command AS ZSTRING PTR) AS INTEGER	stdlib.bi	Führt innerhalb eines Programmes einen Betriebssystembefehl aus (z. B. dir in Windows und DOS oder ls in Linux).
tan_	tan_(ar AS DOUBLE) AS DOUBLE	math.bi	Gibt den Tangens eines Winkels in Bogenmaß zurück.

tanh	tanh(x AS DOUBLE) AS DOUBLE	math.bi	Gibt den Tangens Hyperbolicus eines Winkels in Bogenmaß zurück.
tolower	tolower(c AS INTEGER) AS INTEGER	ctype.bi	Konvertiert ein Zeichen in Kleinbuchstaben (verwendet dazu ASCII-Kodierung).
toupper	toupper(c AS INTEGER) AS INTEGER	ctype.bi	Konvertiert ein Zeichen in Großbuchstaben (verwendet dazu ASCII-Kodierung).
ungetc	ungetc(c AS INTEGER, s AS FILE PTR) AS INTEGER	stdio.bi	Schiebt das Zeichen c in den Stream s zurück. Gibt EOF im Fehlerfall zurück. Schieben Sie nicht mehr als ein Zeichen in den Stream zurück!

Puffermanipulation

#INCLUDE "crt/string.bi"

Deklaration (mit Parametern)	Kommentar
memchr(s AS ANY PTR, c AS INTEGER, n AS size_t) AS ANY PTR	Sucht nach einem Zeichen in einem Buffer.
memcmp(s1 AS ANY PTR, s2 AS ANY PTR, n AS size_t) AS INTEGER	Vergleicht zwei Buffer.
memcpy(dest AS ANY PTR, src AS ANY PTR, n AS size_t) AS ANY PTR	Kopiert einen Buffer in einen anderen.
memmove(dest AS ANY PTR, src AS ANY PTR, n AS size_t) AS ANY PTR	Verschiebt eine Anzahl von Bytes von einem Buffer in einen anderen.
memset(s AS ANY PTR, c AS INTEGER, n AS size_t) AS ANY PTR	Setzt alle Bytes eines Buffers auf ein bestimmtes Zeichen.

Zeichenklassifizierung - konvertierung

#INCLUDE "crt/ctype.bi"

Deklaration (mit Parametern)	Kommentar
isalnum(c AS INTEGER) AS INTEGER	Ungleich 0, wenn c alphanumerisch ist.
isalpha(c AS INTEGER) AS INTEGER	Ungleich 0, wenn c ein Buchstabe ist.
isascii(c AS INTEGER) AS INTEGER	Ungleich 0, wenn c ein ASCII-Zeichen ist.
iscntrl(c AS INTEGER) AS INTEGER	Ungleich 0, wenn c ein Steuerzeichen ist.
isdigit(c AS INTEGER) AS INTEGER	Ungleich 0, wenn c eine Dezimalzahl ist.
isgraph(c AS INTEGER) AS INTEGER	Ungleich 0, wenn c ein sichtbares Zeichen ist.
	Ungleich 0, wenn c ein Kleinbuchstabe ist.

islower(c AS INTEGER) AS INTEGER	
isprint(c AS INTEGER) AS INTEGER	Ungleich 0, wenn c ein druckbares Zeichen ist.
ispunct(c AS INTEGER) AS INTEGER	Ungleich 0, wenn c ein Satzzeichen ist.
isspace(c AS INTEGER) AS INTEGER	Ungleich 0, wenn c ein Whitespace (Leerzeichen, Tabulator, Zeilenende) ist.
isupper(c AS INTEGER) AS INTEGER	Ungleich 0, wenn c ein großgeschriebener Buchstabe ist.
isxdigit(c AS INTEGER) AS INTEGER	Ungleich 0, wenn c eine Hexadezimalzahl ist.
toascii(c AS INTEGER) AS INTEGER	Konvertiert c zu ASCII.
tolower(c AS INTEGER) AS INTEGER	Konvertiert c zu einem Kleinbuchstaben.
toupper(c AS INTEGER) AS INTEGER	Konvertiert c zu einem Großbuchstaben.

Datenkonvertierung

#INCLUDE "crt/stdlib.bi"

Deklaration (mit Parametern)	Kommentar
atof(string1 AS ZSTRING PTR) AS DOUBLE	Konvertiert einen ZString zu einer Fließkommazahl.
atoi(string1 AS ZSTRING PTR) AS INTEGER	Konvertiert einen ZString zu einem Integer.
atol(string1 AS ZSTRING PTR) AS INTEGER	Konvertiert einen ZString zu einem Long.
itoa(value AS INTEGER, zstring AS ZSTRING PTR, radix AS INTEGER) AS ZSTRING PTR	Konvertiert einen Integer zu einem ZString einer gegebenen Gattung.
ltoa(value AS LONG, zstring AS ZSTRING PTR, radix AS INTEGER) AS ZSTRING PTR	Konvertiert einen Long zu einem ZString einer gegebenen Gattung.
strtod(string1 AS ZSTRING PTR, endptr AS ZSTRING PTR) AS DOUBLE	Konvertiert einen ZString zu einer Fließkommazahl.
strtol(string1 AS ZSTRING PTR, endptr AS ZSTRING PTR, radix AS INTEGER) AS LONG	Konvertiert einen ZString zu einem Long einer gegebenen Gattung.
strtoul(string1 AS ZSTRING PTR, endptr AS ZSTRING PTR, radix AS INTEGER) AS ULONG	Konvertiert einen ZString zu einem vorzeichenlosen ULong.

Pfadmanipulation

#INCLUDE "crt/io.bi"

Deklaration (mit Parametern)	Kommentar
_chdir(path AS ZSTRING PTR) AS INTEGER	Ändert das aktuelle Verzeichnis zu dem gegebenen Pfad.

_getcwd(path AS ZSTRING PTR, numchars AS INTEGER) AS ZSTRING PTR	Gibt das aktuelle Arbeitsverzeichnis zurück.
_mkdir(path AS ZSTRING PTR) AS INTEGER	Erstellt ein Verzeichnis anhand des gegebenen Pfades.
_rmdir(path AS ZSTRING PTR) AS INTEGER	Löscht das angegebene Verzeichnis.

Dateimanipulation

#INCLUDE "crt/sys/stat.bi"
#INCLUDE "crt/io.bi"

Deklaration (mit Parametern)	Kommentar
chmod(path AS ZSTRING PTR, pmode AS INTEGER) AS INTEGER	Ändert die Berechtigungen für eine Datei.
fstat(handle AS INTEGER, buffer AS TYPE stat PTR) AS INTEGER	Gibt die Dateistatusinformationen zurück.
remove(path AS ZSTRING PTR) AS INTEGER	Löscht die angegebene Datei.
rename_(oldname AS ZSTRING PTR, newname AS ZSTRING PTR) AS INTEGER	Benennt die angegebene Datei um.
stat(path AS ZSTRING PTR, buffer AS TYPE stat PTR) AS INTEGER	Gibt die Dateistatusinformationen der angegebenen Datei zurück.
umask(pmode AS UINTEGER) AS UINTEGER	Setzt die Dateiberechtigungsmaske.

Stream I/O

#INCLUDE "crt/stdio.bi"

Deklaration (mit Parametern)	Kommentar
clearerr(file_pointer AS FILE PTR)	Löscht die Error-Flags des Streams.
fclose(file_pointer AS FILE PTR) AS INTEGER	Schließt eine Datei.
feof(file_pointer AS FILE PTR) AS INTEGER	Prüft, ob das Dateiende eines Streams erreicht wurde.
ferror(file_pointer AS FILE PTR) AS INTEGER	Prüft, ob ein Fehler beim Dateizugriff aufgetreten ist.
fflush(file_pointer AS FILE PTR) AS INTEGER	Schreibt den Puffer-Inhalt in die Datei ("flush").
fgetc(file_pointer AS FILE PTR) AS INTEGER	Liest ein Zeichen von einem Stream.
fgetpos(file_pointer AS FILE PTR, fpos_t current_pos) AS INTEGER	Ermittelt die aktuelle Position innerhalb eines Streams.
fgets(string1 AS ZSTRING PTR, maxchar AS INTEGER, file_pointer AS FILE PTR) AS ZSTRING PTR	Liest eine Zeile von einer Datei mit höchstens maxchar Zeichen (inkl. NUL und evt. LF).
fopen(filename AS ZSTRING PTR, access_mode AS ZSTRING PTR) AS FILE PTR	Öffnet eine Datei für gepufferten Dateizugriff.
fprintf(file_pointer AS FILE PTR, fmt AS ZSTRING PTR, ...) AS INTEGER	Schreibt eine Zeichenkette mit Format fmt in eine Datei, wobei die Zeichen % mit passenden Argumenten der Parameterliste ersetzt werden.
fputc(c AS INTEGER, file_pointer AS FILE PTR) AS INTEGER	Schreibt ein Zeichen in einen Stream.

fputchar(c AS INTEGER) AS INTEGER	Schreibt ein Zeichen auf die Standardausgabe (stdout).
fputs(string1 AS ZSTRING PTR, file_pointer AS FILE PTR) AS INTEGER	Schreibt einen ZString in einen Stream.
fread(buffer AS ZSTRING PTR, size AS size_t, count AS size_t, file_pointer AS FILE PTR) AS size_t	Liest unformatierte Daten von einem Stream in einen Puffer.
freopen(filename AS ZSTRING PTR, access AS ZSTRING PTR mode, file_pointer AS FILE PTR) AS FILE PTR	Weist einen FILE PTR einer anderen Datei zu.
fscanf(file_pointer AS FILE PTR, format AS ZSTRING PTR zstring, args) AS INTEGER	Liest formatierte Daten von einem Stream.
fseek(file_pointer AS FILE PTR, offset AS LONG, origin AS INTEGER) AS INTEGER	Setzt die aktuelle Position innerhalb einer Datei auf eine neue Position.
fsetpos(file_pointer AS FILE PTR, current_pos AS fpos_t) AS INTEGER	Setzt die aktuelle Position innerhalb einer Datei auf eine neue Position.
ftell(file_pointer AS FILE PTR) AS LONG	Ermittelt die aktuelle Position innerhalb einer Datei.
fwrite(buffer AS ZSTRING PTR, size AS size_t, count AS size_t, file_pointer AS FILE PTR) AS size_t	Schreibt unformatierte Daten von einem Puffer in einen Stream.
getc(file_pointer AS FILE PTR) AS INTEGER	Liest ein Zeichen von einem Stream.
getchar() AS INTEGER	Liest ein Zeichen von der Standardeingabe (stdin).
gets(buffer AS ZSTRING PTR) AS ZSTRING PTR	Liest eine Zeile von der Standardeingabe (stdin) in einen Puffer.
printf(fmt AS ZSTRING PTR _string, ...) AS INTEGER	Schreibt eine Zeichenkette mit Format fmt in die Standardausgabe, wobei die Zeichen % mit passenden Argumenten der Parameterliste ersetzt werden.
putc(c AS INTEGER, file_pointer AS FILE PTR) AS INTEGER	Schreibt ein Zeichen in einen Stream.
putchar(c AS INTEGER) AS INTEGER	Schreibt ein Zeichen in die Standardausgabe (stdout).
puts(string1 AS ZSTRING PTR) AS INTEGER	Schreibt einen ZString in die Standardausgabe (stdout).
rewind(file_pointer AS FILE PTR)	Setzt die Position innnerhalb einer Datei wieder auf die Ausgangsposition (0).
scanf(format_string AS ZSTRING PTR, args) AS INTEGER	Liest formatierte Daten von der Standardeingabe (stdin).
setbuf(file_pointer AS FILE PTR, buffer AS ZSTRING PTR)	Setzt einen neuen Puffer für einen Stream.
setvbuf(file_pointer AS FILE PTR, buffer AS ZSTRING PTR, buf_type AS INTEGER, buf AS size_t size) AS INTEGER	Setzt einen neuen Puffer für einen Stream und regelt die Art der Pufferung.
sprintf(string1 AS ZSTRING PTR, fmt AS ZSTRING PTR, ...) AS INTEGER	Schreibt eine Zeichenkette mit Format fmt in einen ZString, wobei die Zeichen % mit passenden Argumenten der Parameterliste ersetzt werden.
sscanf(buffer AS ZSTRING PTR, format_string AS ZSTRING PTR, args) AS INTEGER	Liest formatierte Daten von einem ZString.
tmpfile() AS FILE PTR	Öffnet eine temporäre Datei.
tmpnam(file_name AS ZSTRING PTR) AS ZSTRING PTR	Ermittelt einen temporären Dateinamen.
ungetc(c AS INTEGER, file_pointer AS FILE PTR) AS INTEGER	Schiebt ein Zeichen in den Puffer eines Streams zurück.

Low-level I/O

#INCLUDE "crt/io.bi"

Nur für Windows, da für die anderen Plattformen bis jetzt die Header fehlen.

Deklaration (mit Parametern)	Kommentar
_close(handle AS INTEGER) AS INTEGER	Schließt eine Datei, die für ungepufferten Dateizugriff geöffnet wurde.
_creat(filename AS ZSTRING PTR, pmode AS INTEGER) AS INTEGER	Erstellt eine neue Datei mit den angegebenen Zugriffsrechten.
_eof(handle AS INTEGER) AS INTEGER	Prüft auf Dateiende (EOF).
_lseek(handle AS INTEGER, offset AS LONG, origin AS INTEGER) AS LONG	Setzt die Dateiposition auf eine neue Position.
_open(filename AS ZSTRING PTR, oflag AS INTEGER, pmode AS UINTEGER) AS INTEGER	Öffnet eine Datei für low-level Dateizugriff (ungepuffert).
_read(handle AS INTEGER, buffer AS ZSTRING PTR, length AS UINTEGER) AS INTEGER	Liest Binärdaten von einer Datei in einen Puffer.
_write(handle AS INTEGER, buffer AS ZSTRING PTR, count AS UINTEGER) AS INTEGER	Schreibt Binärdaten von einem Puffer in eine Datei.

Mathematik

#INCLUDE "crt/math.bi"

Deklaration (mit Parametern)	Kommentar
abs_(n AS INTEGER) AS INTEGER	Ermittelt den Absolutwert einer Ganzzahl.
acos_(x AS DOUBLE) AS DOUBLE	Errechnet den Arkuskosinus von x.
asin_(x AS DOUBLE) AS DOUBLE	Errechnet den Arkussinus von x.
atan_(x AS DOUBLE) AS DOUBLE	Errechnet den Arkustangens von x.
atan2_(y AS DOUBLE, x AS DOUBLE) AS DOUBLE	Errechnet den Arkustangens von y/x.
ceil(x AS DOUBLE) AS DOUBLE	Ermittelt den kleinsten ganzzahligen Wert größer x.
cos_(x AS DOUBLE) AS DOUBLE	Errechnet den Kosinus des Winkels x in Bogenmaß.
cosh(x AS DOUBLE) AS DOUBLE	Errechnet den Kosinus Hyperbolicus des Winkels x in Bogenmaß.
div(number AS INTEGER, denom AS INTEGER) AS div_t	Dividiert eine Ganzzahl durch eine andere.
exp_(x AS DOUBLE) AS DOUBLE	Gibt den Wert von e^x zurück (Umkehrfunktion des natürlichen Logarithmus).
fabs(x AS DOUBLE) AS DOUBLE	Ermittelt den Absolutwert einer Gleitkommazahl.
floor(x AS DOUBLE) AS DOUBLE	Gibt die größte ganze Zahl zurück, die kleiner oder gleich d ist (Abrunden der Zahl d).
fmod(x AS DOUBLE, y AS DOUBLE) AS DOUBLE	Errechnet den Rest der Division x / y.

frexp(x AS DOUBLE, expptr AS INTEGER PTR) AS DOUBLE	Berechnet den Wert m so, dass x=m*2^exponent. expptr ist ein Pointer auf m; exponent ist der Rückgabewert der Funktion.
labs(n AS LONG) AS LONG	Ermittelt den Absolutwert eines LONG.
ldexp(x AS DOUBLE, exp AS INTEGER) AS DOUBLE	Gibt den Wert von x * 2^n zurück.
ldiv(number AS LONG, denom AS LONG) AS ldiv_t	Dividiert zwei LONG und gibt Quotient und Rest einer Division als Struktur ldiv_t zurück.
log_(x AS DOUBLE) AS DOUBLE	Errechnet den natürlichen Logarithmus von x.
log10(x AS DOUBLE) AS DOUBLE	Errechnet den Zehnerlogarithmus von x.
modf(x AS DOUBLE, intptr AS DOUBLE PTR) AS DOUBLE	Gibt den Nachkommateil einer Gleitkommazahl x zurück. Der Zeiger intptr zeigt auf den ganzzahligen Anteil als Gleitkommazahl.
pow(x AS DOUBLE, y AS DOUBLE) AS DOUBLE	Gibt den Wert von x^y zurück.
rand() AS INTEGER	Gibt eine zufällige Ganzzahl im Bereich von 0 bis 32767 zurück.
Random(max_num AS INTEGER) AS INTEGER	Gibt eine zufällige Ganzzahl zwischen 0 and max_num zurück.
Randomize()	Setzt einen zufälligen Ausgangswert (seed) für den Zufallszahlengenerator.
sin_(x AS DOUBLE) AS DOUBLE	Gibt den Sinus eines Winkels in Bogenmaß zurück.
sinh(x AS DOUBLE) AS DOUBLE	Gibt den Sinus Hyperbolicus eines Winkels in Bogenmaß zurück.
sqrt(x AS DOUBLE) AS DOUBLE	Errechnet die Quadratwurzel.
srand(seed AS UINTEGER)	Setzt einen Ausgangswert (seed) für den Zufallszahlengenerator (rand).
tan_(x AS DOUBLE) AS DOUBLE	Gibt den Tangens eines Winkels in Bogenmaß zurück.
tanh(x AS DOUBLE) AS DOUBLE	Gibt den Tangens Hyperbolicus eines Winkels in Bogenmaß zurück.

Speicherallokation

#INCLUDE "crt/stdlib.bi"

Deklaration (mit Parametern)	Kommentar
calloc(num AS size_t elems, elem_size AS size_t) AS ANY PTR	Alloziert einen Speicherbereich und setzt alle Bytes auf NULL.
free(mem_address AS ANY PTR)	Gibt einen Speicherbereich frei.
malloc(num AS size_t bytes) AS ANY PTR	Alloziert einen Speicherbereich.
realloc(mem_address AS ANY PTR, newsize AS size_t) AS ANY PTR	Realloziert einen Speicherbereich (passt dessen Größe an).

Prozesskontrolle

#INCLUDE "crt/stdlib.bi"

Deklaration (mit Parametern)	Kommentar
abort()	Bricht einen Prozess ab.

execl(path AS ZSTRING PTR, arg0 AS ZSTRING PTR, arg1 AS ZSTRING PTR,..., NULL) AS INTEGER	Erstellt einen untergeordneten Kind-Prozess (übergibt Kommandozeilen-Parameter).
execlp(path AS ZSTRING PTR, arg0 AS ZSTRING PTR, arg1 AS ZSTRING PTR,..., NULL) AS INTEGER	Startet Kind-Prozess (verwendet PATH, übergibt Kommandozeilen-Parameter).
execv(path AS ZSTRING PTR, argv AS ZSTRING PTR) AS INTEGER	Startet Kind-Prozess (übergibt Argumenten-Vektor).
execvp(path AS ZSTRING PTR, argv AS ZSTRING PTR) AS INTEGER	Startet Kind-Prozess (verwendet PATH, übergibt Argumenten-Vektor).
exit_(status AS INTEGER)	Beendet den Prozess (nach flushen der Puffer).
getenv(varname AS ZSTRING PTR) AS ZSTRING PTR	Ermittelt den Wert einer Umgebungsvariable.
perror(string1 AS ZSTRING PTR)	Schreibt eine dem letzten System Error-Code (errno) zugehörige Fehlermeldung auf die Fehlerausgabe (stderr).
putenv(envstring AS ZSTRING PTR) AS INTEGER	Setzt den Wert einer Umgebungsvariable.
raise(signum AS INTEGER) AS INTEGER	Erzeugt ein C Signal (exception).
system_(string1 AS ZSTRING PTR) AS INTEGER	Führt einen residenten Betriebssystem-Befehl aus.

Suchen und Sortieren

#INCLUDE "crt/stdlib.bi"

Hinweis: Die Callback-Funktion compare, die von bsearch und qsort benötigt wird, muss mit CDECL deklariert werden. Sie muss kleiner 0 sein, wenn der Wert des ersten Parameters der Funktion vor dem des zweiten in der Sortierung liegt und größer 0, wenn der erste Wert hinter dem zweiten liegt. Die Funktion muss 0 zurückgeben, wenn beide Werte gleich sind.

Deklaration (mit Parametern)	Kommentar
bsearch(key AS ANY PTR, base AS ANY PTR, num AS size_t, width AS size_t, compare AS function(elem1 AS ANY PTR, elem2 AS ANY PTR) AS INTEGER) AS ANY PTR	Führt eine binäre Suche durch.
qsort(base AS ANY PTR, num AS size_t, width AS size_t, compare AS function(elem1 AS ANY PTR, elem2 AS ANY PTR) AS INTEGER)	Nutzt den Quicksort-Algorithmus, um ein Array zu sortieren.

Stringmanipulation

"conttab">Deklaration (mit Parametern)Kommentarstpcpy(dest AS ZSTRING PTR, src AS ZSTRING PTR) AS ZSTRING PTRKopiert einen ZString in einen anderen und gibt einen Zeiger auf das abschließende NUL in dest zurück.strcmp(string1 AS ZSTRING PTR, string2 AS ZSTRING PTR) AS INTEGERVergleicht string1 und string2 alphabetisch.strcpy(string1 AS ZSTRING PTR, string2 AS ZSTRING PTR) AS ZSTRING PTRKopiert string2 in string1 und gibt einen Zeiger auf string1 zurück.strerror(errnum AS INTEGER) AS ZSTRING PTRGibt eine textuelle Fehlermeldung für den angegebenen Fehlercode zurück.strlen(string1 AS ZSTRING PTR) AS INTEGERErmittelt die Länge eines ZStrings.strncat(string1 AS ZSTRING PTR, string2 AS ZSTRING PTR, n AS size_t) AS ZSTRING PTRHängt n Zeichen von string2 an

string1 hinten an.strncmp(string1 AS ZSTRING PTR, string2 AS ZSTRING PTR, n AS size_t) AS INTEGERVergleicht die ersten n Zeichen zweier ZStrings.strncpy(string1 AS ZSTRING PTR, string2 AS ZSTRING PTR, n AS size_t) AS ZSTRING PTRKopiert die ersten n Zeichen von string2 nach string1.strnset(string1 AS ZSTRING PTR, c AS INTEGER, size _t n) AS ZSTRING PTRSetzt die ersten n Zeichen vom ZString auf c.strrchr(string1 AS ZSTRING PTR, c AS INTEGER) AS ZSTRING PTRSucht das letzte Vorkommnis von Zeichen c im ZString.

Zeit

#INCLUDE "crt/time.bi"

Deklaration (mit Parametern)	Kommentar
asctime(time AS TYPE tm PTR) AS ZSTRING PTR	Konvertiert die Zeit vom Type tm zu einem ZString.
clock() AS clock_t	Gibt die vergangene Prozessorzeit in Ticks zurück.
ctime(time AS time_t PTR) AS ZSTRING PTR	Konvertiert eine binäre Zeit zu einem ZString.
difftime(time_t time2, time_t time1) AS DOUBLE	Errechnet die Differenz zwischen zwei Zeiten in Sekunden.
gmtime(time AS time_t PTR) AS TYPE tm PTR	Gibt die mittlere Greenwich-Zeit (GMT) in der Form des Types tm zurück.
localtime(time AS time_t PTR) AS TYPE tm PTR	Gibt die lokale Zeit in der Form des Types tm zurück.
time_(timeptr AS time_t PTR) as time_t	Gibt die seit dem 01.01.1970 um 00:00 Uhr (GMT) in Sekunden zurück.

Siehe auch:
INCLUDE (Meta), CDECL, Externe Bibliotheken

Letzte Bearbeitung des Eintrags am 06.03.13 um 00:06:28
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

Externe Bibliotheken

FreeBASIC-Referenz » Verschiedene Themen » **Externe Bibliotheken**
Der Compiler wird mit einer Reihe von Header-Dateien ausgeliefert, mit denen externe Bibliotheken direkt eingebunden werden können. Beachten Sie, dass die Version der Bibliothek mit der Header-Version übereinstimmen muss, um einen reibungslosen Ablauf sicherzustellen.

Für nähere Informationen zu den einzelnen Bibliotheken besuchen Sie die englische Referenz. Dort finden Sie auch zu vielen Bibliotheken einen Link zur Herstellerseite.

Grafische und zeichenorientierte Benutzerschnittstellen (GUI/TUI)

Name	Beschreibung	Header-Datei	Header-Version
CGUI	Bibliothek zur einfachen Erzeugung grafischer Benutzerschnittstellen *(Win32, Linux)*	cgui.bi	2.0.1
Curses	standardisierte zeichenorientierte Benutzerschnittstelle	curses.bi	pdcurses 3.4, ncurses von 2005
GTK+	plattformübergreifende grafische Benutzerschnittstelle *(Win32, Linux)*	gtk/gtk.bi	2.24.6, 3.2.2
IUP	portables Toolkit für grafische Benutzerschnittstellen *(Win32, Linux)*	IUP/iup.bi, IUP3/iup.bi	2.3.0, 3.5
wx-c	plattformübergreifende grafische Benutzerschnittstelle *(Win32, Linux)*	wx-c/wx.bi	0.9.0.2
Windows API	Programmierschnittstelle für Windows-Anwendungen *(Win32, Linux mit WINE)*	windows.bi	
X11	Window-System, das üblicherweise unter Linux eingesetzt wird *(Linux)*	X11/*.bi	

Grafik

Name	Beschreibung	Header-Datei	Header-Version
Allegro	Bibliothek zur Spieleprogrammierung	allegro.bi	4.0.3, 4.1
DUGL	Bibliothek für Spiele und Grafik *(DOS)*	siehe engl. Forum	
caca	Bibliothek für ASCII art	caca.bi	0.99.beta18
Cairo	2D-Grafikbibliothek mit Unterstützung mehrerer Ausgabegeräte *(Win32, Linux)*	cairo/cairo.bi	1.2.6
DISLIN	Sammlung von Prozeduren zur grafischen Darstellung von Daten *(Win32, Linux)*	dislin.bi	von 2005
freeglut	freie Alternative zu GLUT; s. u. *(Win32, Linux)*	GL/freeglut.bi	2.6.0
FreeImage	Bibliothek zur Unterstützung gängiger Bildformate *(Win32, Linux)*	FreeImage.bi	3.15.1
Freetype2	hochqualitative und portable Font-Engine *(Win32, Linux)*	freetype2/freetype.bi	2.1.9
GD	Bibliothek für die dynamische Erstellung von Bildern *(Win32, Linux)*	gd/gd.bi	von 2005

GIFLIB	Portable Routinen für die Arbeit mit GIF-Bildern	gif_lib.bi	4.1
GLUT	OpenGL Utility Toolkit zur Fenstererstellung und Eingabeverarbeitung *(Win32)*	GL/glut.bi	von 2006
GLFW	OpenGL-Bibliothek zur Erstellung eines OpenGL-Fensters und Eingabeverarbeitung *(Win32, Linux)*	GL/glfw.bi	2.7.2
GRX	2D-Grafikbibliothek	grx/grx20.bi	2.4.6
IL (DevIL)	plattformübergreifende Bibliothek zur Bildverarbeitung *(Win32, Linux)*	IL/il.bi	1.6.7
japi	GUI-Toolkit, welches das AWT-Toolkit von Java verwendet *(Win32, Linux)*	japi.bi	von 2005
jpeglib	plattformübergreifende Bibliothek zum Lesen und Schreiben von JPEG-Bildern *(Win32, Linux)*	jpeglib.bi	vermutlich 6.2
JPGalleg	kleines Add-on für Allegro, das JPEG-Unterstützung hinzufügt *(Win32, Linux)*	jpgalleg.bi	2.5
libpng	Bibliothek zum Lesen und Schreiben von PNG-Bildern *(Win32, Linux)*	png.bi	1.2.16
OpenGL	plattformübergreifende 3D-Grafikbibliothek *(Win32, Linux)*	GL/gl.bi	von Jan. 2012
PDFlib	portable Bibliothek zur dynamischen Erstellung von PDF-Dokumenten *(Win32, Linux)*	pdflib.bi	4.0.2
SDL	plattformübergreifende Multimedia-Bibliothek *(Win32, Linux)*	SDL/SDL.bi	1.2.9
TinyPTC	kleine und einfache Framebuffer-Grafikbibliothek	tinyptc.bi	

Musik/Sound, Audio/Video

Name	Beschreibung	Header-Datei	Header-Version
BASS	Audio-Bibliothek *(Win32, Linux)*	bass.bi	2.4.8
BASSMOD	Version von BASS, die nur MOD unterstützt (XM, IT, S3M, MOD, MTM, UMX) *(Win32, Linux)*	bassmod.bi	2.0
Flite	Runtime-Sprachsynthesizer-Engine *(Win32, Linux)*	flite/flite.bi	1.4
FMOD	Audio-Bibliothek *(Win32, Linux)*	fmod.bi	3.74
MediaInfo	Bibliothek für das Auslesen technischer und Tag-Informationen aus vielen Mediendateiformaten *(Win32, Linux)*	MediaInfo.bi	vom Okt. 2011*
mpg123	MPEG-Decoder einschließlich MP3 *(Win32, Linux)*	mpg123.bi	von 2010*
Ogg	Ersteller und Decoder für das Ogg-Multimedia-Container-Format *(Win32, Linux)*	ogg/ogg.bi	von 2007
OpenAL	plattformübergreifendes 3D-Audio-API *(Win32, Linux)*	AL/al.bi, AL/alut.bi	OpenAL 1.13, ALUT 1.1.0
PortAudio	plattformübergreifende Bibliothek zur Audio-Eingabe und -Ausgabe *(Win32, Linux)*	portaudio.bi	von 2010*

sndfile	Bibliothek zum Lesen/Schreiben/Konvertieren von Audiodateien in verschiedene Formate *(Win32, Linux)*	sndfile.bi	1.0.x
VLC	Bibliothek zur Audio- und Videowiedergabe *(Win32, Linux)*	vlc/*.bi	1.1.x
Vorbis	Bibliothek zur Audio-Kompression (Ogg Vorbis) *(Win32, Linux)*	vorbis/vorbisenc.bi, vorbis/vorbisfile.bi	von 2007

*) Es liegt nur eine maschinenübersetzte Version des Headers vor.

Datenbank

Name	Beschreibung	Header-Datei	Header-Version
GDBM	Datenbank-Funktionen mit erweiterbarem Hashing, vor allem zur Speicherung von Schlüsselpaaren *(Win32, Linux)*	gdbm.bi	von 2010
MySQL	Hochqualitative, weitverbreitete Datenbank-Engine *(Win32, Linux)*	mysql/mysql.bi	4.0.17
PostgreSQL	objektrelationales Datenbank-Management-System *(Win32, Linux)*	postgresql/postgres_ext.bi	von 2006
SQLite	Kleine C-Bibliothek für ein eingebettetes SQL-Datenbanksystem	sqlite2.bi, sqlite3.bi	2.8.17, 3.7.8

Entwickler-Tools

Name	Beschreibung	Header-Datei	Header-Version
CUnit	Leichtgewichtiges System zum Schreiben, Administrieren und Ausführen von Tests in C	CUnit/CUnit.bi	2.1-0
GDSL	Generic Data Structures Library; Sammlung von Routinen für generische Datenstrukturen	gdsl/gdsl.bi	von 2005
gettext (incl. libintl)	Mechanismus zur Internationalisierung	libintl.bi, gettext-po.bi	von 2010, 0.17
GNU ASpell	Open-Source-Rechtschreibprüfung *(Win32, Linux)*	aspell.bi	0.50
libbfd	erlaubt Programmen, Objektdateien in vielen verschiedenen Formaten mit Hilfe einer allgemeinen Schnittstelle auszulesen	bfd.bi	2.16 - 2.18*

*) Durch Definition des Symbols __BFD_VER__ mit dem Wert 216, 217 oder 218 kann festgelegt werden, dass der Header für binutils 2.16, 2.17 bzw. 2.18 eingebunden werden soll.

```
"hlzahl">217
"hlstring">"bfd.bi"
```

Eingebundene Sprachen

Name	Beschreibung	Header-Datei	Header-Version
JNI	standardisierte Anwendungsprogrammierschnittstelle zur Einbindung der Java Virtual Machine *(Win32, Linux)*	jni.bi	von 2006
json-c	Implementierung von JSON (JavaScript Object Notation) in C *(Win32, Linux)*	json-c/json.bi	vermutlich 0.9

libffi	Foreign Function Interface zur Einbindung von in einer fremden Programmiersprache geschriebenen Code	ffi.bi	3.0.10
libjit	Bibliothek zur Just-in-time-Compilierung	jit.bi	0.1.2
Lua	leichtgewichtige, eingebettete Lua-Engine *(Win32, Linux)*	Lua/lua.bi	5.1.1
SpiderMonkey	eingebettete JavaScript-Engine *(Win32, Linux)*	spidermonkey/jsapi.bi	von 2006

Kryptografie

Name	Beschreibung	Header-Datei	Header-Version
cryptlib	mächtiges Security-Toolkit für Verschlüsselung und Authentifizierung *(Win32, Linux)*	cryptlib.bi	von 2005
UUID	Bibliothek zum Erzeugen und Auswerten von Universally Unique Identifier *(Win32, Linux)*	uuid.bi	von 2010

Mathematik

Name	Beschreibung	Header-Datei	Header-Version
big_int	Bibliothek zur Verwendung beliebig großer Integer *(Win32, Linux)*	big_int/big_int.bi	von 2005
Chipmunk	Bibliothek für 2D-Starrkörperphysik *(Win32, Linux)*	chipmunk/chipmunk.bi	4.1.0
GMP	Bibliothek zur Berechnung mit beliebiger Genauigkeit (Ganzzahlen, rationalen Zahlen, Gleitkommazahlen) *(Win32, Linux)*	gmp.bi	4.1.4
GSL	Bibliothek mit einem großen Umfang an mathematischen Routinen *(Win32, Linux)*	gsl/*.bi	1.6
Newton	Echtzeitsimulation physikalischer Umgebungen *(Win32, Linux)*	Newton.bi	von 2005
ODE	hochperformante Bibliothek zur Simulation von Starrkörperdynamik *(Win32, Linux)*	ode/ode.bi	0.11.1

Netzwerk

Name	Beschreibung	Header-Datei	Header-Version
cgi-util	Bibliothek zur Erstellung von CGI-Programmen für Webseiten *(Win32, Linux)*	cgi-util.bi	
curl	clientseitige URL-Transfer-Bibliothek, unterstützt fast alle Protokolle	curl.bi	7.24.0
FastCGI	Erweiterung zu CGI zur Unterstützung hoher Performance ohne Einschränkung der serverspezifischen APIs *(Win32, Linux)*	fastcgi/fastcgi.bi	von 2005
ZeroMQ	hochperformante Bibliothek für asynchronen Datentransfer *(Win32, Linux)*	zmq/zmq.bi	2.1.10

eXtensible Markup Language (XML)

Name	Beschreibung	Header-Datei	Header-Version
Expat	stream-orientierter XML-Parser mit verschiedenen nützlichen Funktionen *(Win32, Linux)*	expat.bi	1.95.8
libxml	De-facto Standardbibliothek für den Zugriff auf XML-Dateien *(Win32, Linux)*	libxml/*.bi	2.6.17
libxslt	Bibliothek zum Handling von XSLT *(Win32, Linux)*	libxslt/libxslt.bi	1.1.13
Mini-XML		mxml.bi	2.7

	kleiner XML-Parser zum Lesen von XML und XML-ähnlichen Dateien *(Win32, Linux)*

Reguläre Ausdrücke

Name	Beschreibung	Header-Datei	Header-Version
PCRE	Bibliothek zur Auswertung Perl-kompatibler regulärer Ausdrücke *(Win32, Linux)*		
TRE	leichtgewichtige, robuste und effiziente POSIX-konforme Bibliothek für reguläre Ausdrücke	pcre.bi, pcre16.bi, prceposix.bi	8.31

Komprimierung

Name	Beschreibung	Header-Datei	Header-Version
bzip2	Komprimierung mit dem bzip2-Algorithmus sowie das Lesen/Schreiben von .bz2-Dateien	bzlib.bi	1.0.6
libzip	einfache Erstellung und Dekomprimierung von .zip-Dateien	zip.bi	0.10.1
liblzma	starke LZMA-basierte Komprimierung für .lzma- und .xz-Dateien	lzma.bi	5.0.2
LZO	schnelle Komprimierung und sehr schnelle Dekomprimierung	lzo/lzo.bi	2.02
QuickLZ	sehr schnelle Komprimierungs-Bibliothek	quicklz.bi	1.5.0
zlib	De-facto-Standardbibliothek für die Komprimierung mit dem Deflate-Algorithmus (eingesetzt in .zip, .gz, .png u. a.)	zlib.bi	1.2.6

System-APIs

Name	Beschreibung	Header-Datei	Header-Version
C Runtime Library	siehe Funktionen der CRT	crt.bi	
DOS API	Benutzerschnittstelle für DOS	dos/dos.bi	
disphelper	Hilfsbibliothek zur Verwendung von COM-Objekten in C *(Win32, Linux mit WINE)*	disphelper/disphelper.bi	von 2005
GLib	GNOMEs allgemeine plattformübergreifende Software Utility Library *(Win32, Linux)*	glib.bi	
Windows API	Programmierschnittstelle für Windows-Anwendungen *(Win32, Linux mit WINE)*	windows.bi	
X11	Window-System, das üblicherweise unter Linux eingesetzt wird *(Linux)*	X11/*.bi	

Letzte Bearbeitung des Eintrags am 01.02.13 um 21:01:31
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

Programmablauf

FreeBASIC-Referenz » FreeBASIC thematische Übersicht » **Programmablauf**
Schleifen:

- FOR ... NEXT, STEP
- WHILE ... WEND
- DO ... LOOP, UNTIL, WHILE

Bedingungsstrukturen:

- IF ... THEN , ELSE, ELSEIF , END IF
- SELECT CASE, CASE, IS, CASE ELSE
- IIF

Kontrollbefehle:

- CONTINUE (Fortsetzung)
- EXIT (Abbruch)

Tutorial: Kontrollstrukturen: Bedingungen, Schleifen usw.

GOTO und GOSUB

- GOTO, GOSUB, RETURN
- ON ... GOTO , ON ... GOSUB
- Tutorial: Guter Programmierstil

Programm anhalten / verzögern: SLEEP

Weitere Beispiele im *'Freebasicverzeichnis'\examples\manual\control*

Letzte Bearbeitung des Eintrags am 20.05.12 um 20:45:48
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

Prozeduren

FreeBASIC-Referenz » FreeBASIC thematische Übersicht » **Prozeduren**
Einleitung zu den Prozeduren

Deklaration und Definition

- SUB, FUNCTION
- DECLARE: Deklaration
- OVERLOAD: Prozeduren mit unterschiedlicher Parameterliste
- LIB , ALIAS: Prozedur aus einer Bibliothek
- PUBLIC , PRIVATE: öffentliche und private Prozeduren
- STATIC: Erhaltung der Variablenwerte
- NAKED: speziell für Prozeduren in (Inline-)Assembler

Aufrufkonventionen

- STDCALL: Standard-Aufrufkonvention für FreeBASIC und die Microsoft Win32-API
- CDECL: Konvention vieler C/C++ -Compilern (von rechts nach links)
- PASCAL: PASCAL-Aufrufkonvention (von links nach rechts)

Parameterübergabe:

- Informationen zur Parameterübergabe
- BYVAL (by value), BYREF (by reference)
- CONST (Klausel): auf den Parameter nur lesend zugreifen
- ...(Auslassung): variable Argumentenzahl
- VA_ARG, VA_FIRST, VA_NEXT

Aufruf einer Prozedur in älteren BASIC-Dialekten: CALL

Tutorial: Unterprogramme

Prozeduren in UDTs und CLASSes

- CONSTRUCTOR, DESTRUCTOR
- PRIVATE, PROTECTED
- PROPERTY
- THIS
- OPERATOR

Beispiele im 'Freebasicverzeichnis'\examples\manual\procs

Letzte Bearbeitung des Eintrags am 03.06.12 um 17:43:50
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

Module (Library / DLL)

FreeBASIC-Referenz » FreeBASIC thematische Übersicht » **Module (Library / DLL)**
Dynamic Link Library (.dll) / Shared Object (.so)

- DYLIBLOAD: dll/so zur Laufzeit laden
- DYLIBSYMBOL: Pointer auf eine Prozedur ermitteln
- DYLIBFREE: Speicherplatz wieder freigeben
- EXPORT: Prezedur nach außen bekannt machen
- LIB: Prozedur aus einer dll/so einbinden

Zugriff auf Variablen und Prozeduren anderer Module

- COMMON, EXTERN (Module): Zugriff auf globale Variablen anderer Module
- PUBLIC, PRIVATE: öffentliche / private Prozeduren
- IMPORT: Verwendung bei Win32-DLLs
- ALIAS: für den Linker sichtbarer Name

Verwaltung von Prozeduren

- STDCALL, CDECL, PASCAL: Aufrufkonvention
- EXTERN ... END EXTERN: interne Bezeichner umdefinieren

Tutorial: Erstellen einer DLL mit FreeBASIC

Beispiele im 'Freebasicverzeichnis'\examples\manual\module

Letzte Bearbeitung des Eintrags am 23.05.12 um 23:22:09
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

Datentypen und Deklarationen

FreeBASIC-Referenz » FreeBASIC thematische Übersicht » **Datentypen und Deklarationen**
Numerische Datentypen:

- BYTE , UBYTE
- SHORT , USHORT
- INTEGER , UINTEGER
- LONG , ULONG
- LONGINT , ULONGINT
- SINGLE , DOUBLE

Zeichenketten:

- STRING
- WSTRING
- ZSTRING

Siehe auch:
Tabellarische Übersicht der Standard-Datentypen

Konstanten:

- CONST
- Tutorial: CONST vs. "reflinkicon" href="temp0141.html">ENUM

Benutzerdefinierte Datentypen:

- TYPE (UDT) , TYPE (Funktion) , TYPE (Forward Referencing)
- UNION
- FIELD: Padding für UDTs und UNIONs
- siehe auch Bitfelder

Variablen dimensionieren:

- DIM , AS
- SHARED , STATIC , UNSIGNED
- VAR
- SCOPE
- NAMESPACE, USING (Namespace)
- siehe auch Kapitel Variablen-Initiatoren in 'Ausdrücke und Operatoren'

Tutorials:
Variablen, Datentypen und Arrays
Variablen initialisieren

Beispiele im 'Freebasicverzeichnis'\examples\manual\datatype

Letzte Bearbeitung des Eintrags am 25.12.12 um 18:02:37
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

Arrays

Arrays

- DIM: statisches Array anlegen
- REDIM: dynamisches Array anlegen oder seine Größe ändern
- REDIM PRESERVE: Array redimensionieren, ohne seine Elemente zu löschen
- ERASE: Array löschen
- LBOUND / UBOUND: untere / obere Grenze des Arrays

Tutorial: Variablen, Datentypen und Arrays

Hinweis: Werden Arrays an Unterprogramme übergeben, geschieht dies immer BYREF. Wird versucht mit der Klausel BYVAL die Übergabeart zu verändern, führt das zu einem Compilerfehler.

Beispiele im 'Freebasicverzeichnis'\examples\manual\array

Letzte Bearbeitung des Eintrags am 15.01.12 um 19:50:49
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

Speicher

Speicher reservieren:

- ALLOCATE
- CALLOCATE
- DEALLOCATE
- REALLOCATE

Objekte erstellen:

- NEW
- DELETE

Speicheradressen bearbeiten:

- PEEK
- POKE
- CLEAR

Verfügbaren Speicher ermitteln:

- FRE

Code-Beispiel: Informationen über Speicherauslastung (WinAPI)

Weitere Beispiele im 'Freebasicverzeichnis'\examples\manual\memory

Letzte Bearbeitung des Eintrags am 20.05.12 um 20:45:10
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

Pointer (Zeiger)

FreeBASIC-Referenz » FreeBASIC thematische Übersicht » **Pointer (Zeiger)**
Deklaration von Pointer-Variablen
POINTER , PTR

Zugriffsoperatoren

- @ (Pointer-Referenzierung)
- * (Pointer-Dereferenzierung)
- [ ] (Pointerindizierung)
- -> (Record-Zugriff bei UDTs)

Schlüsselwörter zum Speicherzugriff

- PEEK , POKE: direkter RAM-Zugriff
- STRPTR , SADD: Pointer zu einer String-Variablen
- VARPTR: Pointer zu einer Variablen
- PROCPTR: Pointer zu einer Prozedur

Weitere Informationen: Grundlagen zu Pointern

Letzte Bearbeitung des Eintrags am 30.05.12 um 20:59:06
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

Datentypen umwandeln

Datentypen umwandeln

- CAST , CPTR
- CBYTE , CUBYTE
- CSHORT , CUSHORT
- CINT , CUINT
- CLNG , CULNG
- CLNGINT , CULNGINT
- CSIGN , CUNSG
- CSNG , CDBL

Umwandlung zwischen Zahlen und Strings

- Text in Zahl umwandeln: VAL , VALINT , VALUINT , VALLNG , VALULNG
- Zahl in Text umwandeln: STR , WSTR

Andere Zahlensysteme

- BIN, WBIN
- OCT, WOCT
- HEX, WHEX

Erstellen einer binären Kopie

- Text in Zahl umwandeln: CVSHORT , CVI , CVL , CVLONGINT , CVS , CVD
- Zahl in Text umwandeln: MKSHORT , MKI , MKL , MKLONGINT , MKS , MKD

ASCII-Code umwandeln

- ASC
- CHR , WCHR
- Tutorial: Die ASCII-Steuerzeichen in FreeBASIC
- Code-Beispiel: ASCII-Tabelle anzeigen lassen

Formatierte Ausgabe

- FORMAT
- siehe auch: PRINT USING

Beispiele im 'Freebasicverzeichnis'\examples\manual\casting

Letzte Bearbeitung des Eintrags am 20.05.12 um 20:49:32
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

Bit-Operatoren

Bit-Operatoren

Schlüsselwort	Bedeutung	Ausdruck1	Ausdruck2	Ergebnis
AND	Sind beide Bits gesetzt?	0	0	0
		0	1	0
		1	0	0
		1	1	1
OR	Ist mind. ein Bit gesetzt?	0	0	0
		0	1	1
		1	0	1
		1	1	1
XOR	Ist genau ein Bit gesetzt?	0	0	0
		0	1	1
		1	0	1
		1	1	0
EQV	Sind beide Bits gleich?	0	0	1
		0	1	0
		1	0	0
		1	1	1
IMP	Folgt Ausdruck2 auf Ausdruck1?	0	0	1
		0	1	1
		1	0	0
		1	1	1
NOT	Ist das Bit nicht gesetzt?	0		1
		1		0
ANDALSO	Sind beide Ausdrücke wahr?	false	-	false
		true	false	false
		true	true	true
ORELSE	Ist mind. ein Ausdruck wahr?	false	false	false
		false	true	true
		true	-	true

Im Fall von ANDALSO und ORELSE bedeutet false = 0 und true <> 0; im Rückgabewert ist true = -1. Die mit - gekennzeichneten Stellen bedeuten, dass dieser Teilausdruck nicht ausgewertet wird.

Ausdrücke und Operatoren
siehe logische Operatoren

Bit-Manipulationen

- BIT , BITSET , BITRESET
- LOBYTE , HIBYTE
- LOWORD , HIWORD

Beispiele im 'Freebasicverzeichnis'\examples\manual\bits
sowie in 'Freebasicverzeichnis'\examples\manual\operator

Letzte Bearbeitung des Eintrags am 20.05.12 um 20:49:56
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

String-Funktionen

FreeBASIC-Referenz » FreeBASIC thematische Übersicht » **String-Funktionen**
Teilstring ausgeben

- LEFT , RIGHT: erste bzw. letzte Zeichen ausgeben
- MID (Funktion): Teilstring ausgeben

String verändern

- MID (Anweisung): Teilstring ändern
- LCASE , UCASE: String in Klein- bzw. Großbuchstaben umwandeln
- TRIM , LTRIM , RTRIM: Zeichen am Anfang bzw. Ende entfernen
- LSET , RSET: String links- bzw. rechtsbündig setzen

String erzeugen

- SPACE , WSPACE: String aus Leerzeichen erzeugen
- STRING (Funktion) , WSTRING (Funktion): String aus vorgegebenen Zeichen erzeugen

Weiteres:

- LEN: Länge des Strings ausgeben
- INSTR , INSTRREV: Teilstring suchen

Tutorial:
Stringmanipulationen

Code-Beispiele:

- StringEx
- StrReplace
- SubStr - Fast ein SplitString
- StringSplit - Fast ein bisschen SubStr
- Sehr schnelle Replace-Funktion

Weitere Beispiele im 'Freebasicverzeichnis'\examples\manual\strings

Letzte Bearbeitung des Eintrags am 17.05.12 um 21:45:01
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

Mathematik

FreeBASIC-Referenz » FreeBASIC thematische Übersicht » **Mathematik**
Rechenoperationen: + , - , * , / , \
Ausdrücke und Operatoren siehe mathematische Operatoren

Schreibweise: Dezimaltrennzeichen . ; Exponent ^ ; Klammern: ( )

Zahlenformate: siehe Datentypen

Algebraische Funktionen:

- ABS (Absolutbetrag), SGN (Vorzeichen)
- EXP (Potenz zur Basis e), LOG (natürlicher Logarithmus)
- SQR (Quadratwurzel)
- Rundung, Nachkommateil, Rest: INT , CINT , FIX , FRAC , MOD
- Tutorial: Runden in FreeBASIC

Trigonometrie:

- SIN , COS , TAN
- ASIN , ACOS , ATN , ATAN2
- Tutorial: Winkelmodus und Winkelfunktionen

Zufallszahlen: RND , RANDOMIZE

Datum und Uhrzeit: Serial Numbers

Code-Beispiele:
Mathematik
Die Zahl Pi aus der CPU?

Tutorial:
Rechnen mit Vergleichen

Weitere Beispiele im *'Freebasicverzeichnis'\examples\manual\math*
sowie in *'Freebasicverzeichnis'\examples\real10*

Letzte Bearbeitung des Eintrags am 20.05.12 um 20:50:27
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

Datum und Zeit

FreeBASIC-Referenz » FreeBASIC thematische Übersicht » **Datum und Zeit**

Systemzeit und -datum des PCs

- Systemzeit lesen und setzen: TIME , SETTIME
- Systemdatum lesen und setzen: DATE , SETDATE

Zeitmessung

- TIMER
- CPU-Takte: Code-Beispiel Counter.Bas von MichaelW

Zeitverzögerung: SLEEP

Spezielle Berechnungen von Datum und Uhrzeit

- TIMESERIAL , DATESERIAL
- ISDATE , DATEVALUE , TIMEVALUE
- DATEADD , DATEDIFF
- NOW , DATEPART
- siehe auch: Date Serials

Datums- und Zeitformat

- SECOND , MINUTE , HOUR
- DAY , WEEKDAY , WEEKDAYNAME
- MONTH MONTHNAME
- YEAR

Ausgabeformat: FORMAT

Code-Beispiele:
Windows-Uptime ermitteln
Wochentagsausgabe in verschiedenen Sprachen
UNIX-Timestamp

Weitere Beispiele im 'Freebasicverzeichnis'\examples\manual\dates
sowie in 'Freebasicverzeichnis'\examples\manual\proguide\dates.bas

Letzte Bearbeitung des Eintrags am 20.05.12 um 20:50:52
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

Grafik

FreeBASIC-Referenz » FreeBASIC thematische Übersicht » **Grafik**
Grafikbildschirm

- SCREENRES, SCREEN: Grafikbidschirm initialisieren
- SCREENINFO: Informationen über den aktuellen Videomodus
- SCREENLIST: Liste der verfügbaren Auflösungen
- SCREENCONTROL: Einstellungen der Gfxlib
- SCREENSET: aktive und sichtbare Bildschirmseite setzen
- SCREENCOPY, FLIP, PCOPY: Bildschirmseite kopieren
- SCREENLOCK, SCREENUNLOCK: Zugriff auf Bildschirmseite (ent)sperren
- SCREENSYNC: auf Bildschirmaktualisierung warten
- SCREENPTR: Pointer auf den Datenbereich der Bildschirmseite
- SCREENEVENT: Informationen über ein Systemereignis
- SCREENGLPROC: Adresse einer OpenGL-Prozedur

Farbe

- COLOR (Anweisung), COLOR (Funktion): Farbe setzen / auslesen
- PALETTE, PALETTE GET: Farbpalette bearbeiten / auslesen
- RGB , RGBA: RGB(A)-Farbton umrechnen

zeichnen

- PSET (Grafik), PRESET (Grafik): einzelnen Punkt setzen
- LINE (Grafik): Strecke oder Rechteck zeichnen
- CIRCLE: Kreis oder Ellipse zeichnen
- PAINT: Bildausschnitt füllen
- DRAW (Grafik): Zeichnen mittels Grafikcursorbefehlen
- DRAW STRING: Text ausgeben
- POINT: Pixelinformationen abrufen

Methoden (für PUT und DRAW STRING)

- PSET
- PRESET
- AND
- OR
- XOR
- ADD
- TRANS
- ALPHA
- CUSTOM

Image

- IMAGECREATE, IMAGEDESTROY: Grafikpuffer erstellen bzw. zerstören
- IMAGECONVERTROW: Pixel kopieren und konvertieren
- IMAGEINFO: Informationen über einen Grafikpuffer
- GET (Grafik), PUT (Grafik): Grafik speichern und setzen

Sonstiges

- CLS: Bildschirm löschen
- VIEW (Grafik): Clipping-Grenzen setzen
- WINDOW: physischen Darstellungsbereich festlegen
- PMAP: Bildschirmkoordinaten umrechnen

Weitere Informationen:

- Interne Pixelformate
- Standard-Paletten
- Interne Treiber
- Geschwindigkeit und Größe

Code-Beispiele:
Grafik und Fonts

Tutorials:
Grafikprogrammierung mit FreeBasic

Weitere Beispiele im 'Freebasicverzeichnis'\examples\manual\gfx

Letzte Bearbeitung des Eintrags am 21.12.12 um 19:22:18
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

Multithreading

FreeBASIC-Referenz » FreeBASIC thematische Übersicht » **Multithreading**
Threads organisieren

- THREADCREATE
- THREADWAIT
- THREADCALL

Mutexe

- MUTEXCREATE
- MUTEXDESTROY
- MUTEXLOCK
- MUTEXUNLOCK

conditional variables

- CONDCREATE
- CONDDESTROY
- CONDSIGNAL
- CONDWAIT
- CONDBROADCAST

Tutorial: Arbeiten mit Threads

Beispiele im 'Freebasicverzeichnis'\examples\manual\threads

Letzte Bearbeitung des Eintrags am 18.01.12 um 13:10:17
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

Hardware-Zugriffe

FreeBASIC-Referenz » FreeBASIC thematische Übersicht » **Hardware-Zugriffe**
Gerät zum Datenzugriff öffnen

- OPEN COM
- OPEN LPT

Port-Zugriff

- INP
- OUT
- WAIT

Zugriff auf den Drucker

- LPOS
- LPRINT
- siehe auch PRINT # in Zusammenhang mit OPEN LPT

Code-Beispiele:

- Statusleitungen der seriellen Schnittstelle über WIN-API steuern
- Schnittstellen ermitteln

Weitere Beispiele im 'Freebasicverzeichnis'\examples\manual\hardware

Letzte Bearbeitung des Eintrags am 09.08.10 um 23:24:54
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

Fehlerbehandlung, Debugging

FreeBASIC-Referenz » FreeBASIC thematische Übersicht » **Fehlerbehandlung, Debugging**
Fehlerbehandlung

- ERR (Funktion) , ERR (Anweisung)
- ERFN , ERMN
- ERROR (Anweisung) , #ERROR
- deprecated: ON ERROR , LOCAL , RESUME , ERL
- siehe auch: Fehler-Behandlung in FreeBASIC

Debugging
Die folgenden Debugging-Befehle funktionieren nur, wenn mit der Kommandozeilenoption -g compiliert wird.

- ASSERT
- ASSERTWARN

Code-Beispiele:

- Debug-Info anzeigen
- Log-Funktion für Debugging

Weitere Beispiele
zur Fehlerbehandlung: im 'Freebasicverzeichnis'\examples\manual\error
zum Debugging: im 'Freebasicverzeichnis'\examples\manual\debug

Letzte Bearbeitung des Eintrags am 25.08.10 um 23:54:42
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

Benutzereingaben

FreeBASIC-Referenz » FreeBASIC thematische Übersicht » **Benutzereingaben**
Tastatureingabe:

- INPUT (Anweisung), INPUT (Funktion): Zeicheneingabe
- WINPUT: Zeicheneingabe mit Unterstützung von *wide characters*
- LINE INPUT: Einlesen einer kompletten Zeile
- INKEY: Taste aus dem Tastaturpuffer holen
- MULTIKEY: Status einer Taste überprüfen
- GETKEY: auf Tastendruck warten

Maus:

- GETMOUSE: Maus abfragen
- SETMOUSE: Maus setzen

Joystick:

- GETJOYSTICK: Joystick abfragen
- STICK, STRIG: Joystickabfrage in QB

Code-Beispiele

- Geheime Passworteingabe
- Passworteingabe Konsole
- Graphical User Interface für die Konsole
- inputUsing: erweiterte INPUT-Maske
- mehr Code-Beispiele im Forum: Maus und Tastatur

Weitere Beispiele im 'Freebasicverzeichnis'\examples\manual\input

Letzte Bearbeitung des Eintrags am 20.05.12 um 20:51:58
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

Dateien (Files)

Dateibehandlung

- BLOAD
- BSAVE
- OPEN (Datei)
- CLOSE # , RESET

Dateimodus

- INPUT
- OUTPUT
- APPEND
- RANDOM
- BINARY

Weitere Parameter beim Öffnen einer Datei

- Zugriffsrechte: ACCESS { READ | WRITE | READ WRITE }
- Zugriffssperre: LOCK { READ | WRITE | READ WRITE }; siehe auch LOCK , UNLOCK
- Datei(text)format: ENCODING { "ASCII" | "UTF-8" | "UTF-16" | "UTF-32" }

Standard-Datenströme öffnen

- OPEN CONS: Standardeingabe und -ausgabe
- OPEN SCRN: Standardausgabe
- OPEN ERR: Standardfehlerausgabe
- OPEN PIPE: gepufferter Datenstrom

Daten lesen

- GET #
- INPUT #
- LINE INPUT #

Daten schreiben

- PUT #
- PRINT #, PRINT USING
- WRITE #

Lese- / Schreibposition

- SEEK (Funktion) , SEEK (Anweisung)
- LOC

Sonstiges

- FREEFILE: nächste freie Dateinummer finden
- EOF: überprüfen, ob das Ende der Datei erreicht wurde
- LOF: Länge einer geöffneten Datei zurückgeben

weitere Funktionen: siehe Betriebssystem-Anweisungen

Code-Beispiele: Dateien und Laufwerke

Weitere Beispiele im 'Freebasicverzeichnis'\examples\manual\fileio

Letzte Bearbeitung des Eintrags am 26.08.12 um 11:52:01
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

Konsole

FreeBASIC-Referenz » FreeBASIC thematische Übersicht » **Konsole**
Hinweis: Auch wenn es sich hier um typische Konsolenbefehle handelt, können sie auch in einem Grafikfenster (siehe SCREENRES) verwendet werden.

Konsolenfenster

- SCREEN 0: Konsolenfenster aufrufen (Standard)
- SCREEN (Funktion): Zeichen an einer bestimmten Stelle abfragen
- WIDTH (Funktion) , WIDTH (Anweisung): Größe des Konsolenfensters (Zeilen/Spalten)
- VIEW (Text): Grenzen des Textanzeigebereichs

Ausgabe

- CLS: Fensterinhalt löschen
- PRINT , PRINT USING , WRITE: Textausgabe
- COLOR (Funktion): Farbinformationen abfragen
- COLOR (Anweisung): Vorder-/Hintergrundfarbe setzen

Cursorposition

- LOCATE (Funktion) , LOCATE (Anweisung): Cursorposition abfragen/setzen
- CSRLIN , POS: Zeile/Spalte des Cursors abfragen
- TAB , SPC: Texteinrückung

Anhang: Titel eines Konsole-Fensters ändern (WinAPI)

Code-Beispiel: Schriftart in der Konsole ändern

Beispiele im 'Freebasicverzeichnis'\examples\manual\console

Letzte Bearbeitung des Eintrags am 21.05.12 um 14:20:27
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

Betriebssystem-Anweisungen

FreeBASIC-Referenz » FreeBASIC thematische Übersicht » **Betriebssystem-Anweisungen**
Programm starten / beenden

- END , SYSTEM , STOP: Programm beenden
- CHAIN , EXEC , RUN: externes Programm starten

aktuellen Ordnerpfad anzeigen und ändern

- CURDIR: aktuelles Arbeitsverzeichnis anzeigen
- EXEPATH: Verzeichnis des Programms anzeigen
- CHDIR: Arbeitsverzeichnis wechseln

Dateiinformationen

- FILEATTR: Dateiattribute abfragen
- FILEDATETIME: Datum der letzten Änderung ermitteln
- FILELEN: Dateilänge ermitteln
- FILEEXISTS: Datei auf Existenz prüfen
- DIR: Ordner nach Dateien durchsuchen

Dateien / Ordner erstellen und löschen

- FILECOPY: Datei kopieren
- NAME: Datei umbenennen
- KILL: Datei löschen
- MKDIR: Ordner erstellen
- RMDIR: Ordner löschen

Verschiedenes:

- SETENVIRON: Umgebungsvariable setzen
- ENVIRON: Umgebungsvariable abfragen
- COMMAND: Kommandozeilenparameter abfragen
- SHELL: Systemkommando ausführen
- WINDOWTITLE: Fenstertitel ändern

weitere Funktionen: siehe Dateien (Files)

Code-Beispiele: System

Weitere Beispiele im 'Freebasicverzeichnis'\examples\manual\system

Letzte Bearbeitung des Eintrags am 20.05.12 um 20:52:35
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

Präprozessor-Anweisungen

FreeBASIC-Referenz » FreeBASIC thematische Übersicht » **Präprozessor-Anweisungen**
Einbindung anderer Dateien:

- #INCLUDE [ONCE]: externen Quellcode einbinden
- #INCLIB: Bibliothek einbinden
- #LIBPATH: Pfad für Bibliotheken hinzufügen

Symbole und Makros:

- #DEFINE: Symbol definieren
- #UNDEF: Symbol löschen
- #MACRO, #ENDMACRO: Makro definieren
- DEFINED: Überprüfen, ob ein Symbol definiert wurde
- Präprozessoren: vordefinierte Symbole

Bedingungen:

- #IF, #ELSEIF, #ELSE, #ENDIF: Bedingung abprüfen
- #IFDEF , #IFNDEF: Definition eines Symbols testen

Debugging:

- #ERROR: Fehlermeldung ausgeben
- #LINE: Zeilennummer und Modulnamen festlegen
- #PRINT: Compilermeldung ausgeben
- #ASSERT: Fehlermeldung beim Nicht-Eintreffen einer bestimmten Bedingung ausgeben

Sonstiges:

- #LANG: Dialektform festlegen
- #PRAGMA: Compiler-Optionen ändern

Letzte Bearbeitung des Eintrags am 30.06.13 um 13:03:30
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

Verschiedenes

Verschiedenes

Verwaltung konstanter Daten

- DATA (Daten festlegen)
- READ (Daten einlesen)
- RESTORE (Datenzeiger setzen)

Programmsteuerung

- LET (Wertzuweisung)
- END (Programmende oder Ende eines Blockes)
- TO (Festlegung eines Bereichs)
- SWAP (Werte zweier Variablen vertauschen)
- ASM (Verwendung des Inline-Assemblers)

Variablenorganisation

- SIZEOF, LEN (Speichergröße einer Variablen oder eines Datentyps)
- OFFSETOF (Offset eines Records innerhalb eines UDTs)
- TYPEOF (Datentyp einer Variablen)

Programmorganisation

- OPTION (deprecated; steht unter -lang fb nicht zur Verfügung):
  - OPTION BASE: Startindex von Arrays setzen
  - OPTION PRIVATE: Prozeduren als PRIVATE deklarieren
  - OPTION BYVAL: Parameter standardmäßig by value übergeben
  - OPTION DYNAMIC, '$DYNAMIC: Arrays dynamisch anlegen
  - OPTION STATIC, '$STATIC: Arrays statisch anlegen
  - OPTION EXPLICIT: explizierte Deklaration der Variablen erzwingen
  - OPTION ESCAPE: Escape-Characters in Strings aktivieren
  - OPTION GOSUB: Unterstützung von GOSUB akivieren
  - OPTION NOGOSUB: Unterstützung von GOSUB deaktivieren
  - OPTION NOKEYWORD: Schlüsselwort freigeben
- Option(): Setzen zusätzlicher Attribute und Merkmale

Sonstiges
BEEP: Ausgabe eines Tonsignals

Beispiele im 'Freebasicverzeichnis'\examples\manual\misc

Letzte Bearbeitung des Eintrags am 17.08.12 um 22:26:54
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags

Kommentare

FreeBASIC-Referenz » FreeBASIC thematische Übersicht » **Kommentare**
einzeiliger Kommentar

- REM
- Hochkomma '

mehrzeilige Kommentare

```
/'
   Kommentare, mehrzeilig
'/
```

Nähere Informationen finden Sie im Referenzeintrag zu REM.

Sonstiges
Bei Verwendung eines Zeilenfortsetzungszeichens _ behandelt der Compiler alle Inhalte nach dem Unterstrich als Kommentar. Da es sich hierbei um ein ungewolltes Verhalten handelt, wird es sich vermutlich in zukünftigen Versionen ändern.

Letzte Bearbeitung des Eintrags am 29.07.11 um 17:13:53
FreeBASIC-Portal.de • Zur Onlinefassung des Eintrags