

Reference

COLLABORATORS

	<i>TITLE :</i> Reference		
<i>ACTION</i>	<i>NAME</i>	<i>DATE</i>	<i>SIGNATURE</i>
WRITTEN BY		January 19, 2025	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

1	Reference	1
1.1	Pure Basic Referenz-Handbuch	1
1.2	editor	2
1.3	Benutzung des CLI Compilers	3
1.4	general_rules	5
1.5	variables	6
1.6	For : Next	7
1.7	gosub_return	8
1.8	if_endif	9
1.9	Repeat : Until	10
1.10	Select : EndSelect	10
1.11	While : Wend	12
1.12	others	13
1.13	deftype	13
1.14	dim	14
1.15	NewList	14
1.16	structures	15
1.17	global	15
1.18	shared	16
1.19	procedures	16
1.20	includes	17
1.21	debugger	18

Chapter 1

Reference

1.1 Pure Basic Referenz-Handbuch

```
*****
*
*          PureBasic Referenz-Handbuch V1.30
*
*          © 2000 - Fantaisie Software -
*
*****
```

Allgemeine Themen:

Benutzung des Editors
Benutzung des CLI Compilers
Allgemeine Syntax Regeln
Variablen und Typen

Basic Schlüsselwörter:

For: Next
Gosub: Return
If: EndIf
Repeat: Until
Select: EndSelect
While: Wend
Andere

Strukturen-Optionen:

DefType
Dim
NewList
Structure: EndStructure

Prozeduren Unterstützung:

Global
Procedure: EndProcedure
Shared

Externe Bibliotheken:

Amiga
App
BitMap
Chunky
Clipboard
Commodity
2D Drawing
File
Font
Gadget
Linked List
Menu
Misc
OS
Palette
Picture
Requester
Screen
Sort
Sound
String
TagList
ToolType
WbStartup
Window

Befehls-Index:

Compiler Optionen:

Index

Eingebaute Funktionen

Debugger

1.2 editor

Einführung:

Der PureBasic Editor wurde speziell für die PureBasic Programmiersprache geschaffen und hat viele spezielle Funktionen, die extra dafür entworfen wurden. Er wird nach und nach noch mächtiger werden und wird fortgeschrittenes Editieren unterstützen, wie farbiger Syntax, Wortergänzung, Online-Hilfe...

Grundlegende Benutzung:

Der PureBasic Editor akzeptiert jedes Standard ASCII Zeichen sowie lädt und speichert die Datei im ASCII-Format. Er benutzt alle Amiga Standard Tastaturkürzel zum Editieren des Textes:

Pfeiltasten : Bewegen den Cursor in die vier Richtungen
Shift + Pfeil hoch : Eine Seite nach oben
Shift + Pfeil unten : Eine Seite nach unten
Shift + Pfeil links : Anfang der Zeile
Shift + Pfeil rechts : Ende der Zeile

Shift + Enter : Einfügen einer Zeile oberhalb der aktuellen Zeile
Shift + Del : Löscht den rechten Teil der Zeile
Shift + Backspace: Löscht den linken Teil der Zeile

Hilfe: Öffnet die PureBasic Online-Hilfe (dieses Dokument)

Jetzt, die bedeutendsten Menü Kürzel:

AmigaRechts + S: Speichert den aktuellen Programmcode ohne vorherigen Requester ↔
AmigaRechts + Q: Beendet PureBasic
AmigaRechts + L: Lädt einen neuen Programmcode

Spezielle Funktionen:

Es gibt ein Menü 'Compiler' und auf diesem Weg können Sie PureBasic kontrollieren. Menü-Einträge:

- * Compile/Run: Compiliert den aktuellen Programmcode und startet es.
- * Debugger: Auswahl, um den Debugger EIN/AUS zu schalten
- * Optionen:

- Compiliere für Prozessor: 680x0/PowerPC. Ändert das Format des generierten Executable. Die PowerPC Version ist für WarpOS.

- Sourcecode optimieren: Schaltet die Optimierungen während des Compilierens im Editor ein, um exakt die selbe Executable Größe wie beim endgültigen zu erhalten. Natürlich muß der Debugger ausgeschaltet sein, andernfalls wird diese Option ignoriert. Besser ist es, die Option während der Entwicklung auszuschalten, da sich sonst die Compilier-Zeit erhöht...
 - Kommentierte ASM Ausgabedatei: Generiert eine kommentierte ASM Ausgabedatei, wenn Sie das endgültige Executable erstellen. Die Datei befindet sich in "PureBasic:Compilers/PureBasic.asm". Sie können diese Datei verändern (optimieren) und sie nochmal mit PhxAss compilieren. Dies verlangsamt wesentlich die Executable-Erstellung, benutzen Sie es also nur wenn wirklich notwendig.
 - CLI-Ausgabe abschalten: Schaltet das CLI-Ausgabefenster aus, nützlich wenn Ihr Programm nichts ins CLI ausgibt.
 - Erstelle ein Icon: Ein Icon wird zum erstellten Executable hinzugefügt. Das Icon befindet sich im Verzeichnis "PureBasic:Compilers/Default_Icon.info". Sie können es durch ein anderes ersetzen, wenn Sie möchten.
 - Speichern: Speichert diese Einstellungen für die aktuelle Datei. Jede Datei kann ihre eigenen Einstellungen haben.
- * Erstelle Executable: Erstellt das endgültige Executable. Die Optimierungen werden natürlich automatisch eingeschaltet. Diese Funktion benutzt die oben beschriebenen Optionen.

Andere Optionen wie Datei einfügen, Drucken, Suchen sind klassische Optionen, wie in jedem anderen Editor...

1.3 Benutzung des CLI Compilers

Benutzung des CLI Compilers:

Geben Sie zum Kompilieren "PureBasic" gefolgt vom Dateinamen der Ausgangsdatei ein. ↵

PureBasic wird sie kompilieren und das Programm starten.

Compiler Optionen:

FILE

Ein String: Hierzu wird der Name der Ausgangsdatei (Source) benötigt! Dieses Argument wird benötigt oder der Compiler gibt eine Fehlermeldung aus.

TO

Ein String: Wenn dieses Argument benutzt wird, müssen Sie den Zielpfad und den Dateinamen angeben, um PureBasic zu zeigen, wo das ausführbare Programm ↵ erstellt werden soll.

Anmerkung: Auf diesem Weg wird nur die ausführbare Datei erstellt.

Das Programm wird nicht gestartet.

NR oder NORESIDENT

Auswahl: Wird dieses Argument angegeben, wird die AmigaOS Resident Datei nicht geladen. Standardmäßig lädt der Compiler diese Datei, da dadurch die Übersetzungsdauer verkürzt wird.

PPC oder POWERPC

Auswahl: Wird dieses Argument angegeben, generiert der Compiler ein Amiga PowerPC Executable für WarpOS. Zur Zeit wird als Ergebnis ein Fehler ausgegeben

Bitte speichern Sie die ASM Datei und den generierten Code, und senden Sie uns das Ergebnis! Alle Daten werden aufgezeichnet und analysiert bei unseren fortwährenden Versuchen, diese Option zu verbessern. Wir danken für Ihre Unterstützung. :)

NC oder NOCOMMENT

Auswahl: Wird dieses Argument gesetzt, wird ein unkommentierter ASM Code produziert, welcher kleiner und schneller zu compilieren ist. Dies verkürzt die Übersetzungsdauer.

PRI oder PRIORITY

Numerisch: Ein numerischer Wert zwischen -127 und +127 wird benötigt. Dies bestimmt die Priorität des Compilers. Beispiel: PureBasic PRI=10 .. wird die meiste CPU-Zeit dem Compiler reservieren.

CR oder CREATERESIDENT

Auswahl: Dies compiliert das Programm und erstellt eine Resident Datei mit allen Strukturen und Konstanten. Die compilierte Datei befindet sich in "Ram:ResidentFile" und "Ram:ResidentFile.struct".

STANDBY

Auswahl: Wird dieses Argument gesetzt, wird der Compiler in einen "Schlafmodus" versetzt und wartet auf Aufträge über den Message Port. Bitte benutzen Sie diese Funktion noch nicht, da diese für die Benutzung mit dem noch erscheinenden Editor gedacht ist.

DB oder DEBUGGER

Auswahl: Wird dieses Argument angegeben, wird das Programm mit Unterstützung des Debuggers übersetzt. Der Debugger kann einfach dazu benutzt werden, das laufende Programm zu unterbrechen. Bitte benutzen Sie ihn behutsam und Schritt für Schritt...

OPT oder OPTIMIZATIONS

Auswahl: Wird dieses Argument gesetzt, wird die maximale Optimierung ↵
eingeschalten
und kleine und schnelle Executables produziert.

Beispiele:

```
PureBasic Sources:MeinProgramm.pb DB PRI=10
```

```
PureBasic Sources:Beispiel.pb TO Ram:Beispiel.exe OPT PRI=10
```

1.4 general_rules

Allgemeine Regeln

PureBasic hat ein paar Syntax Regeln eingeführt, welche sich niemals ändern ↵
werden.
Diese sind:-----

- * Kommentare sind gekennzeichnet mit ; . Der gesamte Text nach einem ;
wird vom Compiler ignoriert.

Beispiel:

```
If a = 10; Dies ist ein Kommentar, um auf etwas hinzuweisen.
```

- * Alle Funktionen müssen von () gefolgt werden, oder sie werden nicht als ↵
Funktion
erkannt, dies gilt auch für parameterlose Funktionen.

Beispiel: WindowID() ist eine Funktion.
WindowID ist eine Variable.

- * Allen Konstanten geht ein # voraus.

Beispiel:

```
#Hello = 10 ist eine Konstante.  
Hello = 10 ist eine Variable.
```

- * Alle Sprungmarken müssen von einem : gefolgt werden.

Beispiel:

```
Ich_bin_eine_Sprungmarke:
```

- * Ein Ausdruck ist etwas, was berechnet werden kann. Ein Ausdruck kann jede
Variable, Konstante oder Funktionen desselben Typs mischen.

Beispiele gültiger Ausdrücke:

```
a+1+(12*3)  
a+WindowHeight()+b/2+#MeineKonstante  
a <> 12+2
```



```
b+2 >= c+3
```

- * Eine beliebige Zahl an Befehlen kann mit der : Option auf derselben Zeile zusammengefügt werden.

Beispiel:

```
If OpenScreen(0,320,200,8,0) : PrintN("Ok") : Else : PrintN("Fehler") : ↵
  EndIf
```

- * Begriffe, die in diesem Guide benutzt werden:

```
<variable>  : eine Basic Variable.
<expression>: ein Ausdruck, wie oben beschrieben.
<constant>  : eine numerische Konstante.
<label>     : ein Programmlabel (Sprungmarke)
<type>      : jeder Typ, (Standard oder in einer Struktur).
```

- * In diesem Guide sind alle Themen in alphabetischer Reihenfolge aufgeführt, um evtl. Suchzeiten zu verkürzen.

1.5 variables

Variablen Definition:

Um eine Variable zu definieren, geben Sie ihren Namen ein oder den Typ den die Variable annehmen soll. Variablen müssen nicht ausdrücklich deklariert werden, sie können auch als Variablen "on-the-fly" benutzt werden. Das "DefType" Schlüsselwort kann benutzt werden, um eine ganze Reihe von Variablen zu definieren.

Beispiel:

```
a.b  ; Deklariert eine Variable.
c.l  ;

c = a*d.w ; "d" wird hier inmitten des Ausdrucks deklariert!
```

Um einen Zeiger zu benutzen, geben Sie einen * vor dem Variablennamen ein. Ein Zeiger ist eine Long-Variable, welche eine Adresse speichert. Sie werden generell in Verbindung mit Strukturen benutzt. Auf die Struktur wird mit Hilfe des Zeigers zugegriffen.

Beispiel:

```
*MyScreen.Screen = OpenScreen(0,320,200,8,0)

mouseX = *MyScreen\MouseX
```

Basic Typen

PureBasic erlaubt Variablen Typen. Es unterstützt jetzt vorzeichenbehaftete Variablen. Vorzeichenlose Variablen können benutzt werden, aber daraus kann

ein Fehler resultieren, da diese Option sich noch in einem recht frühen Stadium befindet.

Typen:

Byte: `.b`, benutzt 1 Byte im Speicher. Bereich: -128 bis +127.

Word: `.w`, benutzt 2 Bytes im Speicher. Bereich: -32768 bis -32767

Long: `.l`, benutzt 4 Bytes im Speicher. Bereich: -2147483648 bis 2147483647

Vorzeichenloses Byte: `.ub`, benutzt 1 Byte im Speicher. Bereich: 0 bis 255

Vorzeichenloses Word: `.uw`, benutzt 2 Bytes im Speicher. Bereich: 0 bis 65535

Vorzeichenloses Long: `.ul`, benutzt 4 Bytes im Speicher. Bereich: 0 bis ←
4294967295

String: `.s`, belegt die Länge des Strings im Speicher.

Strukturierte Typen

Erstellt strukturierte Typen, mittels der Struktur Option. Mehr Informationen finden Sie im Kapitel "Strukturen".

1.6 For : Next

Syntax:

```
For <Variable> = <Ausdruck1> To <Ausdruck2> [Step <Konstante>]
```

```
... Schleifeninhalt
```

```
Next [<Variable>]
```

Beschreibung:

Die "For/Next" Funktion wird benutzt, um mitten im Programm eine Schleife mit den vorgegebenen Parametern zu starten. In jeder Schleife wird die <Variable> um den Faktor 1 erhöht (oder um den <Step Wert>, wenn ein solcher definiert wurde). Wenn die <Variable> gleich dem <Ausdruck2> ist, endet die Schleife.

Beispiel 1:

```
For k=0 To 10
...
Next
```

In diesem Beispiel wird die Schleife 11 mal (0 bis 10) ausgeführt, dann beendet.

Beispiel 2:

```
a = 2
b = 3

For k=a+2 To b+7 Step 2
```

```
...  
Next k
```

Hier führt das Programm 4 Schleifendurchläufe vorm Beenden aus (k wird in jeder Schleife um den Wert 2 erhöht, so ergeben sich folgende Werte von k: 4-6-8-10). Das "k" nach dem "Next" kennzeichnet, dass "Next" die "For k" Schleife beendet. Wird eine andere Variable angegeben, quittiert dies der Compiler mit einer Fehlermeldung. Es ist nützlich, mehrere "For/Next" Schleifen zu verschachteln.

Beispiel 3:

```
For x=0 To 320  
  For y=0 To 200  
    Plot(x,y)  
  Next y  
Next x
```

1.7 gosub_return

Syntax:

```
Gosub <label>  
  
<label>:  
  
... Programmcode der Sub-Routine  
  
Return
```

Beschreibung:

"Gosub" steht für "Go to sub routine." Nach "Gosub" muß eine Sprungmarke angegeben werden, dann wird die Programmausführung ab der Sprungmarke bis zum nächsten "Return" fortgesetzt. Wenn das Return erreicht wurde, wird die Programmausführung nach dem aufrufenden Gosub fortgesetzt.

"Gosub" ist sehr nützlich, um schnell strukturierten Code zu erstellen.

Beispiel:

```
a = 1  
b = 2  
  
Gosub KomplexeOperation  
  
PrintNum(a)  
End  
  
KomplexeOperation:  
  
a=b*2+a*3+(a+b)
```

```
a=a+a*a
```

```
Return
```

Syntax:

```
FakeReturn
```

Beschreibung:

Wenn Sie aus der Sub-Routine in einen anderen Programmteil außerhalb dieser Sub-Routine springen möchten (mit dem Befehl 'Goto'), müssen Sie FakeReturn benutzen, um ein Return zu simulieren, ohne es wirklich auszuführen. Wenn Sie diesen Befehl nicht benutzen, wird Ihr Programm abstürzen.

Diese Funktion sollte nutzlos sein, da ein ordentlich aufgebautes Programm kein 'Goto' benutzt. Aber manchmal, aus Geschwindigkeitsgründen, kann es ein etwas helfen.

Beispiel:

```
Main_Loop:
    ...

SubRoutine1:
    ...
    If a = 10
        FakeReturn
        Goto Main_Loop
    Endif

Return
```

1.8 if_endif

Syntax:

```
If <Ausdruck>
    ...
[Else]
    ...
EndIf
```

Beschreibung:

Die "If" Struktur wird zu Testzwecken benutzt und/oder um die Richtung der weiteren Programmausführung zu ändern, abhängig davon ob der Test wahr oder falsch ergibt. Das optionale "Else" wird benutzt, um einen Teil des Programm-Codes auszuführen, wenn der Test falsch ergibt.

Eine beliebige Zahl von "If" Strukturen kann ineinander verschachtelt werden.

Beispiel 1:

```
If a=10
  Nprint ("a=10")
Else
  Nprint ("a<>10")
EndIf
```

Beispiel 2:

```
If a=10 and b>=10 or c=20
  If b=15
    nprint("ok")
  Else
    nprint("ok2")
  Endif
Else
  nprint("Test Fehler")
Endif
```

1.9 Repeat : Until

Syntax:

```
Repeat
    ... Programm ...
Until <Ausdruck>
[or Forever]
```

Beschreibung:

Diese Funktion durchläuft eine Schleife bis der <Ausdruck> wahr ergibt. Eine beliebige Zahl an Schleifendurchläufen ist möglich. Wird eine endlose Schleife benötigt, dann benutzen Sie das "Forever" Schlüsselwort anstelle von "Until".

Beispiel:

```
a=0
Repeat
  a=a+1
Until a>100
```

Diese Schleife wird solange ausgeführt, bis "a" einen Wert > 100 ergibt.
(Die Schleife wird 101 mal durchlaufen.)

1.10 Select : EndSelect

Syntax:

```
Select <Ausdruck1>

Case <Ausdruck2>

    ...Code...

[Case <Ausdruck3>....]

    ...Code...

[Default]

    ...Code...

EndSelect
```

Beschreibung:

"Select" erlaubt eine schnelle Auswahl. Das Programm führt den <Ausdruck1> aus und behält dessen Ergebnis im Speicher. Dieses wird mit allen Werten aus den "Case <Ausdrücken>" verglichen, und wenn dieser Vergleich wahr ergibt, wird der zugehörige Programmcode ausgeführt und die "Select" Struktur beendet. Wenn keiner der "Case" Werte wahr ist, dann wird der "Default" Code (sofern definiert) ausgeführt.

Beispiel:

```
a = 2

Select a

    Case 1
        PrintN("Case a = 1")

    Case 2
        PrintN("Case a = 2")

    Case 20
        PrintN("Case a = 20")

    Default
        PrintN("I weiß nicht")

End Select
```

Syntax:

```
FakeEndSelect
```

Beschreibung:

Wenn Sie aus einem Select-Abschnitt zu einem anderen Programmteil außerhalb des Select springen möchten (mit dem Befehl 'Goto'), müssen Sie FakeEndSelect benutzen, welches ein EndSelect simuliert, ohne es wirklich auszuführen. Wenn Sie es nicht benutzen, wird Ihr Programm abstürzen.

Beispiel:

```
Main_Loop:
...
Select a

    Case 10
        ...

    Case 20
        FakeEndSelect
        Goto Main_Loop

EndSelect
```

1.11 While : Wend

Syntax:

```
While <Ausdruck>

... Programm ..

Wend
```

Beschreibung:

Eine "Wend" Schleife wird solange durchlaufen, bis der <Ausdruck> falsch ←
ergibt.

Ein wichtiger Punkt, um eine Vorstellung von einer "While" Schleife zu ←
bekommen:

Ergibt der erste Test falsch, dann gelangt die Programmausführung niemals zum
Programmcode innerhalb der Schleife und überspringt diesen Teil. Eine "Repeat"
Schleife wird dagegen mindestens einmal ausgeführt (da der Test erst nach ←
jeder
Schleife durchgeführt wird).

Beispiel:

```
b = 0
a = 10
While a = 10
    b = b+1
    If b=10
        a=11
```

```
Endif  
Wend
```

Diese Programmschleife wird ausgeführt bis "a" <>10 ist. Eine Änderung erfolgt hier, wenn b=10 ist, dann wird die Schleife 10 mal ausgeführt.

1.12 others

Eine Liste andere Befehle:

Goto

```
Goto <Sprungmarke>
```

Dieser Befehl wird benutzt, um die Programmausführung direkt zu einer ↵
Sprungmarke
zu verlegen. Seien Sie vorsichtig mit dieser Funktion, da falsche Benutzung zu
einem Programmabsturz führen kann...

1.13 deftype

Syntax:

```
DefType.<Typ> [<Variable>, <Variable>, ...]
```

Beschreibung:

Wenn keine <Variablen> angegeben werden, wird "DefType" benutzt, um den ↵
Standardtyp
(default type) für zukünftige typenlose Variablen festzulegen.

Beispiel:

```
DefType.l
```

```
a = b+c
```

a, b und c werden als Long-Typen (.l) definiert, da kein anderer Typ ↵
angegeben
wurde.

Werden Variablen angegeben, deklariert "DefType" diese Variablen als " ↵
definierte
Typen" (defined type) und ändert nicht den Standardtyp.

Beispiel:

```
DefType.b a,b,c,d
```

a,b,c,d werden als Byte-Typen (.b) deklariert.

1.14 dim

Syntax:

```
Dim name.<Typ>(<Ausdruck>)
```

Beschreibung:

"Dim" wird benutzt, um die Größe neuer Arrays festzulegen. Ein Array in PureBasic kann von jedem Typ sein, einschließlich strukturierter und benutzerdefinierter Typen. Wenn ein Array einmal DIMensioniert wurde, kann es seine Größe nicht mehr ändern und ein anderes Array kann nicht mit demselben Namen definiert werden.

Beispiel:

```
Dim MyArray.l(41)

MyArray(0) = 1
MyArray(1) = 2
```

1.15 NewList

Syntax:

```
NewList Name.<Typ>()
```

Beschreibung:

"NewList" erlaubt das Verwalten von dynamisch verknüpften Listen (dynamic linked lists) in PureBasic. Jedes Element einer Liste wird dynamisch zugewiesen. Es gibt keine Einschränkungen in der Anzahl der Elemente, es können so viele wie nötig benutzt werden. Eine Liste kann jeden Standard- oder Struktur-Typ haben.

Um eine Liste aller Befehle für das Verwalten von Listen zu erhalten, klicken Sie hier .

Beispiel:

```
NewList mylist.l()

AddElement(mylist())

mylist() = 10
```

1.16 structures

Syntax:

```
Structure <Name der Struktur>

    ... Struktur-Inhalt

EndStructure
```

Beschreibung:

"Structure" ist nützlich, um Benutzertypen zu definieren und um Zugriff auf Speicherbereiche des OS zu erhalten. Strukturen können für das schnellere und einfachere Verwalten von großen Datenbeständen benutzt werden. Strukturen werden mit der "\" Option aufgerufen. Sie können auch verschachtelt werden.

Beispiel:

```
Structure Info
    Name.s
    Vorname.s
    Alter.l
    Geburtstag.l
EndStructure

Dim Freunde.Info(100)

Freunde(0)\Name = "Andersson"
Freunde(0)\Vorname = "Richard"
...
```

1.17 global

Syntax:

```
Global <Variable> [, <Variable>, ...]
```

Beschreibung:

"Global" erlaubt es, Variablen als global zu benutzen, d.h. sie können \leftrightarrow innerhalb einer Prozedur benutzt werden.

Beispiel:

```
Global a.l, b.b, c, d
```

1.18 shared

Syntax:

```
Shared <variable> [, <variable>, ...]
```

Beschreibung:

"Shared" erlaubt die Zugriffsteilung einer Variable zwischen Hauptprogramm ↔ und Prozedur sowie den Zugriff auf eine Variable innerhalb der Prozedur.

Beispiel:

```
a.1 = 10
```

```
Procedure myproc()  
  Shared a
```

```
  a = 20
```

```
EndProcedure
```

```
myproc()
```

```
PrintN(Str(a)) ; Wird 20 ausgegeben, da die Variable "geteilt" (shared) ↔  
wurde.
```

1.19 procedures

Syntax:

```
Procedure.<type> name(<variable1>[, <variable2>, ...])
```

```
... Procedure code
```

```
[ProcedureReturn Wert]
```

```
EndProcedure
```

Beschreibung:

Eine "Procedure" ist ein Programmteil, welcher unabhängig vom Hauptcode innerhalb des Programms ist und seine eigenen Parameter und Variablen haben kann. In PureBasic wird bei Prozeduren die Rekursion voll unterstützt, jede Prozedur kann sich auch selbst aufrufen. Um auf Variablen des Hauptprogramms zugreifen zu können, müssen diese mit der Prozedur "geteilt" (shared) werden. Dies ist mit den "Shared" und "Global" Schlüsselworten möglich.

Eine Prozedur kann wenn benötigt ein Ergebnis zurückliefern. Sie müssen

nach 'Procedure' den Typ (.type) festlegen und das 'ProcedureReturn' Schlüsselwort an einem beliebigen Punkt innerhalb der Prozedur benutzen.

Beispiel:

```
Procedure.l Maximum(nb1.l, nb2.l)

  If nb1>nb2
    Result = nb1
  Else
    Result = nb2
  Endif

  ProcedureReturn Result

EndProcedure

Result.l = Maximum(15,30)

PrintNumberN(Result)

End
```

1.20 includes

Syntax:

```
IncludeFile "Dateiname"
XIncludeFile "Dateiname"
```

Beschreibung:

"IncludeFile" fügt die genannte Programmdatei an der aktuellen Stelle in den Programmcode ein. "XIncludeFile" macht genau dasselbe, außer dass es vermeidet ↵
,
dieselbe Datei mehrfach einzufügen.

Beispiel:

```
XInclude "Sources:MeineDatei.pb" ; Diese wird eingefügt.
XInclude "Sources:MeineDatei.pb" ; Dieser Aufruf wird ignoriert, genauso wie
                                alle nachfolgenden Aufrufe...
```

Syntax:

```
IncludeBinary "Dateiname"
```

Beschreibung:

"IncludeBinary" fügt die genannte Datei an der aktuellen Stelle in das ↵
Programm
ein.

Beispiel:

```
IncludeBinary "Sources:MeineDatei.data"
```

Syntax:

```
IncludePath "Pfad"
```

Beschreibung:

"IncludePath" legt den Standard-Dateipfad für alle nach diesem Befehl einzufügenden Dateien fest. Dies kann sehr nützlich sein, wenn Sie viele Dateien einfügen, die sich im selben Verzeichnis befinden.

Beispiel:

```
IncludePath "Sources:Data/"

IncludeFile "Sprite.pb"
XIncludeFile "Music.pb"
...
```

1.21 debugger

Der PureBasic Debugger

Der Debugger ist ein externes Programm, welches die Ausführung eines Programms kontrolliert. Der mitgelieferte Debugger ist eingeschränkt und hat wenige Funktionen. Dennoch ist dieser ausreichend, um ein Programm korrekt zu debuggen.

Er wird regelmäßig verbessert und erweitert. Falls irgendjemand ein eigenes Debugging-Tool schreiben möchte, bitte kontaktiert uns. Der Debugger ist 100% OS-freundlich und benutzt keine Interrupts oder Trap-Vektoren.

Die Ausführung eines Programms kann angehalten werden und eine Analyse zum Aufspüren von Fehlern kann erfolgen! Dies kann sehr nützlich sein, wenn ein Programm in eine Endlosschleife verfällt.

Funktionen:

Stop

Dies stoppt die Ausführung und zeigt die aktuelle Stelle im Programmcode.

Cont

Dies setzt ein zuvor angehaltenes Programm fort.

Step

Dieser Schalter ermöglicht den Code nach und nach abzuarbeiten, d.h. Zeile für

Zeile. Sehr nützlich, um Fehler aufzuspüren.

Trace

Dieser Schalter ermöglicht dem Benutzer, den Programmcode zu verfolgen, da die einzelnen Programmzeilen bei der Ausführung angezeigt werden.

Exit

Exit: Dies beendet den Debugger, den Compiler, und jedes Programm im Falle von Problemen oder wenn eine "Endlosschleife" auf keinem anderen Weg gestoppt werden kann. ←

Die Debugger Schlüsselwörter in PureBasic:

STOP:

Stop: Dies ruft den Debugger auf und hält sofort die Programmausführung an.

Beispiel:

```
If a=10
    Stop    ; Der Debugger wird aufgerufen.
Else
    Ok=1
Endif
```