Reference

COLLABORATORS

	TITLE : Reference	

ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		February 8, 2025	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

Reference

1.1 Pure Basic Reference Manual

```
 ************************************************************
*                                                            *
*              Pure Basic Reference Manual V1.00             *
*                                                            *
*                © 1999 - Fantaisie Software -               *
*                                                            *
 ************************************************************


General Topics:                  External libraries:

    Using the CLI compiler                 App
     General Syntax Rules                 BitMap
     Variables and Types                ClipBoard
                            @{ "" LINK k              }              2D Drawing
Basic keywords:             @{ "" LINK k     }               Font
                            @{ "" LINK k              }                File
         For : Next                        Gadget
       Gosub : Return                    Linked List
          If : EndIf                        Menu
      Repeat : Until                        Misc
      Select : EndSelect                     OS
       While : Wend                       Palette
          Others                          Picture
                            @{ "" LINK k              }               Requester
Structures et declarations: @{ "" LINK k     }              Screen
                            @{ "" LINK k              }               TagList
   DefType                                WbStartup
   Dim                                     Window
   NewList
   Structure : EndStructure
                            @{ "" LINK k              }         ---
Procedure support:          @{ "" LINK k     }        |   |
                            @{ "" LINK k              }        |   |
   Global                          |---
   Procedure : EndProcedure        |
   Shared                          | U R E
                            @{ "" LINK k              }                --
```

```
Compiler directives:        @{ "" LINK k     }                |  |
                            @{ "" LINK k              }              |  |
   Include functions                          | ---
   Debugger                                   |____| A S I C
```

1.2 Using the CLI Compiler

Using the CLI compiler:

To have a quick look, just type 'PureBasic' followed by the source filename to compile. It will compile and launch the program.

The compilers options:

FILE

String. Need a source file name. This argument is needed or the compiler generate an error.

TO

String. If specified, you must put the destination path and filename where the executable must be created. Note: only the executable is created in this case, the program isn't runned.

NR or NORESIDENT

Switch. If set, it won't load the AmigaOS resident file. By default the compiler load this file which increase the compilation time.

PPC or POWERPC

Switch. If set, the compiler will generate Amiga PowerPC executable for WarpOS.
For now, the output is bad. You can test it and look at the asm file and send us what's wrong in the generated code :)

NC or NOCOMMENT

Switch. If set, produce a non-commented asm output, which is smaller and faster
to assemble. It will decrease the compiling stage.

PRI or PRIORITY

Numeric. Need a numeric value between -127 and +127. It will determine the priority
of the compiler. Example: PureBasic PRI=10 ... will give almost all the cpu time
to the compiler.

CR or CREATERESIDENT

Switch. Will compile the program and create a resident file with all structures and

```
    constants within. The compiled file is located in 'Ram:ResidentFile' and
    'Ram:ResidentFile.struct'

  STANDBY

    Switch. If set, the compiler is put in sleep mode and wait for order trough ↩
       its
    message port. Don't use it, as its done to be guided by the forthcoming ↩
       editor.

  DB or DEBUGGER

    Switch. If set, compile the program with the debugger support. You could use ↩
        the
    debugger later to interrupt the program, run it step by step...

  OPT or OPTIMIZATIONS

    Switch. If set, enable the optimizations for generate fast and small ↩
       executables.
    Of course, it increase a lot the compile time and should be used only when
    generating the final executable.


  Examples:

    PureBasic Sources:MypPog.pb DB PRI=10

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

1.3 general_rules

```
  General Rules

Pure Basic has etablished rules which never changes, whatever you do.
We will see toghether what their are:

* Comments are marked by a ';'. All text entered after this ';' is ignored
  by the compiler.

  Example:

    If a = 10  ; I'm a comment to indicate something

* All functions call must be followed by '(' else it's not considered
  like a function (even for null parameters functions).

  Example: WindowID() is a function.
           WindowID is a variable

* All constants are preceded by a '#'

  Example:

    #Hello = 10 is a constant
```

```
     Hello = 10 is a variable

* All labels must be followed by a ':'

  Example:

    I_am_a_label:

* An expression is something which can be evaluated. This will be used
  later in this guide. Expression can mix any variables, constants, or
  function of the same type.

  Example of valid expressions:

    a+1+(12*3)
    a+WindowHeight()+b/2+#MyConstant
    a <> 12+2
    b+2 >= c+3

* You can put any number of command on the same line by using the ':'
  operator.

  Example:

    If OpenScreen(0,320,200,8,0) : NPrint("Ok") : Else : NPrint("Failed") :  ↩
       EndIf

* Word used in this guide:

  <variable>  : a basic variable
  <expression>: an expression like explain above
  <constant>  : a numeric constant
  <label>     : a program label
  <type>      : any type (standard or structured types)

* In this guide, all the topic are sorted in the alphabetical order to decrease
  the searching time.
```

1.4 variables

```
Variables declaration:

  To declare a variable in Pure Basic you just have to type its name and
  optionnaly the type you want to this variable. Variables doesn't need to
  be explicically declared and you can use variables on-the-fly. You can
  use the DefType keyword to declare in mass variables

  Example:

    a.b   ; Declare our variable.
    c.l   ;

    c = a*d.w ; 'd' is declared here within the expression !
```

```
  If you want to use a pointer, you have to put '*' before the variable name.
  A pointer is a long variable which store an address and is generally  ↩
     associated
  to a structured type. You can access the structure via the pointer.

  Example:

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

    mouseX = *MyScreen\MouseX


Basic types

  Pure Basic allow to type variables. It support for now signed variables.  ↩
     Unsigned
  can be used but result can be wrong as its only in early stage.

  Types:

  Byte: .b, take 1 byte  in memory. Range: -128 to +127.
  Word: .w, take 2 bytes in memory. Range: -32768 to -32767
  Long: .l, take 4 bytes in memory. Range: -2147483648 to 2147483647

  Unsigned Byte: .ub, take 1 byte  in memory. Range: 0 to 255
  Unsigned Word: .uw, take 2 bytes in memory. Range: 0 to 65535
  Unsigned Long: .ul, take 4 bytes in memory. Range: 0 to 4294967295

  String: .s, take the string length in memory.


Structured types

  You can build your own structured types via the Structures. Look in the
  structures chapter for more informations.
```

1.5 For : Next

```
Syntax:

  For <variable> = <expression1> To <expression2> [Step <constant>]

     ... Loop content

  Next [<variable>]

Description:

  The For:Next function is used to cause a loop within a program with the
  given parameter. At each loop the <variable> is increase of 1 (or of the
  Step value, if Step is specified) and when the <variable> value equal
  the <expression2> the loop stop.

  Example 1:
```

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

In this example, the program will loop 11 time (0 to 10), and quit.

Example 2:

```
  a = 2
  b = 3

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

Here, the program loop 4 time before quit (k is increased by value of 2 at each loop, so k value is: 4, 6, 8, 10). The 'k' after the Next indicate that the Next is finishing the 'For k' loop. If you put another variable, the compiler will generate an error. It's useful when you nest some For:Next

Example 3:

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

1.6 gosub_return

Syntax:

```
  Gosub <label>

  <label>:

    ... Sub routine code

  Return
```

Description:

Gosub stands for 'Go to sub routine'. You have to specified a label after Gosub and the program will continue at the label position until he encounter a Return. When a return is reached, the program is tranfered back below the Gosub.

Gosub are very useful to build fast structured code.

Example:

```
a = 1
b = 2

Gosub ComplexOperation

PrintNum(a)
End


ComplexOperation:

  a=b*2+a*3+(a+b)
  a=a+a*a

Return
```

1.7 if_endif

```
Syntax:

  If <expression>
    ...
  [Else]
    ...
  EndIf


Description:

  'If' structure is used to achieve tests and change the program direction
  if the test is true or false. The 'Else' optional command is used to execute
  a part of code if the test is false.

  You can nest any number of 'If' together.

  Example 1:

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


  Example 2:

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

```
      Endif
```

1.8 Repeat : Until

```
  Syntax:

    Repeat

      ... Program ...

    Until <expression>
[or Forever]
```

```
  Description:

    This function allow to loop until the <expression> become true.
    You can nest any number of Repeat. If you need an endless loop
    you should use the 'Forever' keyword instead of 'Until'.

    Example:

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

    This will loop until 'a' take a value > to 100 (It will loop 101 time)
```

1.9 Select : EndSelect

```
    Syntax:

    Select <expression1>

      Case <expression2>

        ...Code...

      [Case <expression3>....]

        ...Code...

      [Default]

        ...Code...

    EndSelect
```

```
  Description:
```

```
Select allow you to do quick choice. First, the program
execute the <expression1> and keep its value in memory.
It compare this value to all the Case <expression> value
and if true execute the corresponding code and quit the
Select strucure. If none of the Case value are true,
it will execute the Default code (if specified).

Example:

  a = 2

  Select a

    Case 1
      NPrint("Case a = 1")

    Case 2
      NPrint("Case a = 2")

    Case 20
      NPrint("Case a = 20")

    Default
      NPrint("I don't know")

  End Select
```

1.10 While : Wend

```
Syntax:

  While <expression>

    ... Program ..

  Wend


Description:

  While loop until the <expression> become false. A thing which is good to
  keep in mind with a While test is if the first test is false, the program
  will never enter inside the loop and will skip this program part. It could
  be very useful sometimes. A 'Repeat' loop execute always at least one time
  the loop (because the test is performed after).

  Example:

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

```
  Wend

This program will loop until the a value is <> of 10. Here, a change when b ↩
   =10,
so the program will loop 10 time.
```

1.11 others

```
Here is the list of other commands:

Goto

  Goto <label>

  This command is use to transfert directly the program to the given
  label position. Be very cautious while using this function as incorrect
  use could cause program crash...
```

1.12 deftype

```
Syntax:

  Deftype.<type> [<variable>, <variable>,...]


Description:

  If no <variables> are specified, DefType is used to change the
  Default type for future untyped variables.

  Example:

    DefType.l

    a = b+c

    a, b and c will be signed long typed (.l) as no type is specified.


  But, if variables are specified, DefType only declare these variables
  as the defined type and don't change the default type.

    Example:

    DefType.b a,b,c,d

    a,b,c,d will be signed byte typed (.b)
```

1.13 dim

Syntax:

```
Dim name.<type>(<expression>)
```

Description:

Dim is used to 'Dimension' the new arrays. An array in Pure Basic can be of any types, included structured and user defined types. Once an array is dim'ed you can't change its time and you can't redim another array with the same name.

Example:

```
Dim MyArray.l(41)

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

1.14 NewList

Syntax:

```
NewList name.<type>()
```

Description:

NewList allow you to manage dynamic linked list in Pure Basic. Each element of the list is allocated dynamically. There is no element limits, so you can have as many as you need. A list can have any standard or structured type.

To see all the command needed to manage a list, click here

Example:

```
NewList mylist.l()

AddElem(mylist())

mylist() = 10
```

1.15 structures

Syntax:

```
Structure <name of structure>
```

```
    ... Structure content

  EndStructure
```

```
Description:

  Structure are useful to define user type and access some OS memory
  areas. Structures can be used to have a faster and easier handle
  of big datas. Structures ae accessed with the '//' operator.
  You can nest Structures.

  Example:

    Structure Info
      Name.s
      ForName.s
      Age.l
      Birthday.l
    EndStructure

    Dim myfriends.Info(100)

    myfriends(0)\Name = "Andersson"
    myfriends(0)\Forname = "Richard"
    ...
```

1.16 global

```
Syntax:

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

```
Description:

  Global allow you to declare the variables as Global, ie: you can
  access them inside a procedure.

  Example:

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

1.17 shared

```
Syntax:

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

```
Description:
```

Shared allow you to share a variable to access it in your procedure.

Example:

```
a.l = 10

Procedure myproc()
  Shared a

  a = 20

EndProcedure

myproc()

NPrint(Str(a)) ; Will print 20, as the varaible has been shared.
```

1.18 procedures

Syntax:

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

  ... Procedure code

EndProcedure
```

Description:

A Procedure is a part of code independant of the main code which can have parameters and have its own variables. In Pure Basic, the recursivity is fully supported for the Procedures and a procedure can call it itself. To access your main code variables, you have to share them by using 'Shared' or 'Global' keywords

A procedure can't return a result for now. It will be added later. You can use a global variable to simulate it.

Example:

```
Global Result.l

Procedure Maximum(nb1.l, nb2.l)

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

EndProcedure
```

```
Maximum(15,30)

NPrint(Str(Result))

End
```

1.19 includes

```
Syntax:

   IncludeFile "filename"
  XIncludeFile "filename"

Description:

  IncludeFile will include the named source file at the current place
  in the code. XIncludeFile is same exept it avoid you to include many times
  the same file.

  Example:

    XInclude "Sources:myfile.pb" ; This one will be inserted
    XInclude "Sources:myfile.pb" ; This one will be ignored and all next calls..


Syntax:

  IncludeBinary "filename"

Description:

  IncludeBinary will include the named source file at the current place
  in the code.

  Example:

    IncludeBinary "Sources:myfile.data"
```

1.20 debugger

```
The Pure Basic Debugger

  The debugger is an external program which can control the execution of
  a program. The provided debugger is very limited and has very few functions.
  Nevertheless it's enough to debug correctly a program. Hopefully, it will be
  regulary updated and better, if someone want do its own debug utility, please
  contact us. The debugger is 100% OS friendly and don't use interrupts or trap
  vectors.

  For now, you can stop a program execution, and analyze what's going wrong,
  which is very useful in case of your program fall in an endless loop.
```

Functions:

Stop

It will halt the execution and display the current code position.

Cont

It will continue a previously stopped program.

Step

With this button you can walk in your code step by step, ie: line after line. It's very handy to know where the fault is.

Trace

Trace is the same as Step exept it doesn't freeze the code execution, so you watch the program lines which are displayed.

Exit

Exit quit the debugger, the compiler and the program in case of you have an endless loop you can use it to quit.

The debugger's keywords in Pure Basic:

STOP: invoke the debugger and freeze the program immediately:

Example:

```
If a=10
  Stop   ; The debugger will be invoked.
Else
  Ok=1
Endif
```