Flow Control

Break [Level]	Will break out of an If/Then, While/Wend or Repeat/Until code block. If the optional Level is specified, will break out of that number of nested flow blocks.
Continue	Will skip directly to the start of the code block, skipping any statements between the Continue and the end of the code block
For <variable> = <expression1> To <expression2> [Step constant]	Defines a for loop, counting variable from expression1 to expression2, optionally incrementing by Step. The "To" value can be changed inside the loop.
Next [variable]	Increments variable and loops through the For loop again. Variable is optional and only included for readability
ForEach List() | Map()	Loops through all of the elements in the specified list or map
Next [List() | Map()]	Increments the ForEach loop. List() or Map() is optional and only included for readability
Gosub label	Send program execution to the code at label, and then the program will resume on the next time
Return	Returns to the code following the Gosub call.
FakeReturn	If Goto is used inside of a subroutine, you must use FakeReturn before the Goto
Goto label	Send program execution to the code at label

Flow Control (cont)

If <expression>	Starts an If block. The code inside the block will be executed if <expression> evaluates to #True.
ElseIf <expression>	An additional If inside an If block, evaluated if the previous If returned #False.
Else	The code following Else will be executed if all previous If and ElseIf expressions evaluated to #False
EndIf	Ends an If block
Select <expression>	Start a Select block and use the value of <expression> for the comparisons
Case <constant>[, ...]	Compares the value of the Select expression to <constant> and executes the following code block if it evaluates to #True.
Default	A code block to run if none of the previous Case statements evaluated to #True
EndSelect	Ends a Select block
Repeat	Begins a Repeat block. Unlike If or While, a Repeat block is always executed at least once.
Until <expression>	Ends a Repeat block. If <expression> evaluates to #False, will run the Repeat block again until <expression> evaluates to #True
ForEver	Ends a Repeat code block and turns it into an infinite loop. Will run forever until the app is killed or a Break statement is encountered

By **DNSGeek**
cheatography.com/dnsgeek/

Published 20th July, 2023.
Last updated 15th September, 2024.

Flow Control (cont)

While <expression>	Begins a While code block. Will repeat the block until <expression> evaluates to #False.
Wend	Ends a While code block

Only use a single = in expressions, e.g. "If i=0". Do not use Then, which is not a keyword.

You can use ranges in Case statements, e.g. "Case 1 To 8"

Compiler Directives

CompilerIf <Constant Expression>	If <Constant Expression> evaluates to True, the code block will be compiled, else it will be skipped during compilation
CompilerElseIf <Constant Expression>	
CompilerElse <Constant Expression>	
CompilerEndIf	Ends the CompilerIf code block
CompilerSelect <Numeric Constant>	Starts the CompilerSelect code block.
CompilerCase <Numeric Constant>	If <Numeric Constant> is True, the code block will be compiled, else skipped during compilation
CompilerEndSelect	Ends the CompilerSelect code block
CompilerError <String Constant>	Generates a compiler error and displays <String Constant>
CompilerWarning <String Constant>	Generates a compiler warning and displays <String COnstant>
EnableExplicit	Enables Explicit mode. When enabled, all the variables which are not explicitly declared with Define , Global , Protected or Static are not accepted and the compiler will raise an error.
DisableExplicit	Disables Explicit mode
EnableASM	Enables the inline assembler
DisableASM	Disables the inline assembler

Compiler Functions

SizeOf(Type)	Returns the size of the variable type or complex structure
OffsetOf(Structure-\Field | Interface-\Function())	Returns the address offset of a Structure field or the address offset of an Interface function
TypeOf(Object)	Returns the type of a variable or structure field.
Subsystem(<Constant String Expression>)	Determine if a subsystem is in use for the program being compiled
Defined(Name, Type)	Checks if object Name has been defined or not.
InitializeStructure(*Pointer, Structure)	Initialize the specified structured memory area. It initializes structure members of type Array, List and Map
CopyStructure(*Source*, Destination, Structure)	Copies structured memory from Source to Destination
ClearStructure(*Pointer, Structure)	Clears a structured memory area
ResetStructure(*Pointer, Structure)	Clears a structured memory area and initializes it for use
Bool(<boolean expression >)	Can evaluate a boolean expression outside of a flow control block. Will return #True or #False

Compiler Object Types

#PB_Byte
#PB_Word
#PB_Long
#PB_String
#PB_Structure
#PB_Float
#PB_Character
#PB_Double
#PB_Quad
#PB_List
#PB_Array
#PB_Integer
#PB_Map
#PB_Ascii
#PB_Unicode
#PB_Interface

These are the types that are returned from the compiler function TypeOf

Compiler Defined Types

#PB_Constant
#PB_Variable
#PB_Array
#PB_List
#PB_Map
#PB_Structure
#PB_Interface
#PB_Procedure
#PB_Function
#PB_OSFunction
#PB_Label
#PB_Prototype
#PB_Module

Compiler Defined Types (cont)

#PB_Enumeration

These are the types that can be specified in the compiler function Defined

Numbers

%	At the start of a number, indicates that this is a binary number
$	At the start of a number, indicates that this is a hexadecimal number

Compiler Reserved Constants

#PB_Compiler_OS	Determines on which OS the compiler is currently running. Can be one of the following:
#PB_OS_Windows	The compiler is running on Windows
#PB_OS_Linux	The compiler is running on Linux
#PB_OS_MacOS	The compiler is running on macOS
#PB_Compiler_Processor	Determines the processor type for which the program is created. Can be one of the following:
#PB_Processor_x86	Intel x86 (IA -32 or x86 -32)
#PB_Processor_x64	Intel x64 (x64, AMD64 or Intel64)
#PB_Processor_arm32	Arm 32-bit
#PB_Processor_arm64	Arm 64-bit, Apple Silicon
#PB_Compiler_Backend	Determines which kind of compiler is currently used. Can be one of the following:
#PB_Backend_Asm	The compiler generating Assembler is being used
#PB_Backend_C	The compiler generating C is being used
#PB_Compiler_ExecutableFormat	Determines executable format. Can be one of the following:

Compiler Reserved Constants (cont)	
#PB_Compiler-_Executable	A regular executable
#PB_Compiler-_Console	A console executable. Only matters on Windows
#PB_Compiler_DLL	A shared DLL in Windows. A dynlib in macOS, a shared object in Linux
#PB_Compiler-_Date	The current date at compile time, in PureBasic date format
#PB_Compiler-_File	The full path and name of the file being compiled
#PB_Compiler-_FilePath	The full path of the file being compiled
#PB_Compiler-_Filename	The filename (without path) being compiled
#PB_Compiler-_Line	The current line number being compiled
#PB_Compiler-_Procedure	The current Procedure being compiled, if applicable
#PB_Compiler-_Module	The current module being compiled, if applicable
#PB_Compiler-_Version	The compiler version, in integer format
#PB_Compiler-_Home	The full path of the PureBasic directory
#PB_Compiler-_Debugger	Set to 1 if debugger enabled, 0 otherwise
#PB_Compiler-_Thread	Set to 1 if the executable was compiled in thread safe mode, 0 otherwise
#PB_Compiler-_Unicode	Set to 1 if the executable was compiled in Unicode, 0 otherwise

Compiler Reserved Constants (cont)	
#PB_Compiler-_LineNumbering	Set to 1 if the executable was compiled with OnError line numbering support, 0 otherwise
#PB_Compiler-_InlineAssembly	Set to 1 if the executable was compiled with inline assembly, 0 otherwise
#PB_Compiler-_EnableExplicit	Set to 1 if the executable was compiled with EnableExplicit, 0 otherwise
#PB_Compiler-_IsMainFile	Set to 1 if the file being compiled is the main file, 0 otherwise
#PB_Compiler-_IsIncludeFile	Set to 1 if the file being compiled has been included by another file, 0 otherwise
#PB_Compiler-_32Bit	Set to 1 if the compiler generates 32-bit code, 0 otherwise
#PB_Compiler-_64Bit	Set to 1 if the compiler generates 64-bit code, 0 otherwise
#PB_Compiler-_Optimizer	Set to 1 if the compiler generates optimizer code, 0 otherwise

All constants start with #.

These constants are useful for the Compiler Directives (CompilerIf, CompilerSelect).

Data Blocks	
DataSection	Starts a Data section, a predefined block of information
EndDataSection	Ends the Data section
Data.TypeName	Defines data
Restore label	Will set the start of Read to label
Read[.Type] <variable>	Read the next available data

Debugging Functions

CallDebugger	Invokes the debugger and freezes the program
Debug <expression>[, DebugLevel]	Display the Debug output window and show the result of Expression in it. The 'DebugLevel' is the priority level of the debug message.
DebugLevel <constant expression>	Set the current Debug level
DisableDebugger	Disable debugger checks in the subsequent code
EnableDebugger	Enable debugger checks in the subsequent code

Variables

Define.<type> [<variable> [= <expression>]][, ...]	Defines one or more variables of Type type and optionally initializes them
Define <variable >.<type > [= <expression >]][, ...]	Alternate form of Define
Dim name.<type>(<expression>, ...)	Creates new arrays
ReDim name.<type>(<expression>, ...)	Resize an existing array. Will only affect the last dimension of a multidimensional array
Enumeration [name] [<constant > [Step <constant >]]	Creates a new enumeration named name
EnumerationBinary [name] [<constant >]	Creates a new binary enumeration named name. Binary enumerations can be used for flags.
EndEnumeration	Ends the Enumeration definition
Global[.<type>] <variable[.<type>]> [= <expression>][, ...]	Defines the scope of the variables to be global rather than local
NewList name.<type >()	Creates a new, dynamic List of Type <type>.

Variables (cont)

NewMap name.<type>([Slots])	Create a new Map (Hash, Dictionary). The keys are always of type String, the values will be of Type <type>. If Slots is not specified, will be dynamically allocated as needed.
Protected[.<type>] <variable[.<type>]> [= <expression>][, ...]	Allows a variable to be accessed only in a Procedure even if the same variable has been declared as Global in the main program
Shared <variable>[, ...]	Allows a variable, an array, a list or a map to be accessed within a procedure
Static[.<type>] <variable[.<type>]> [= <constant expression>][, ...]	Allows creating a local persistent variable in a Procedure even if the same variable has been declared as Global in the main program
Structure <name>	Begins a Structure definition
Structure <name> Extends <struct_name>	Begins a Structure definition that add to an existing Structure
Structure <name> Align <n>	Begins a Structure definition where every element is aligned to an <n>-byte boundary. Align can be used with Extends also.
EndStructure	Ends a Structure definition block
StructureUnion	Begins a StructureUnion definition block
EndStructureUnion	Ends a StructureUnion definition block
Threaded[.<type>] <variable[.<type>]> [= <constant expression>][, ...]	Allows creating a thread persistent variable

Starting a variable with # makes it a constant, e.g. #Pi=3.14

Keywords	
Import "Filename"	Allows declaring external functions and variables from a library or object
VariableName.<type>	Declares an external variable
FunctionName.<type>(<parameter>[= DefaultValue][, ...]	Declares an external function
EndImport	Ends the Import declarations
IncludeFile "Filename"	Includes the specified source code file at the current position
XIncludeFile "Filename"	The same as IncludeFile, but with protections to avoid including the same file twice.
IncludeBinary "filename"	Will include the file at the current position. Should be done in a Data block
IncludePath "path"	Will set the default path for all subsequent Include files
Macro <name> [(Parameter [, ...])]	Starts a Macro block. A placeholder for code that can be directly inserted into the source code wherever the Macro is called
EndMacro	Ends the Macro block
UndefineMacro <name >	Undefines Macro <name>
MacroExpandedCount	The number of times the Macro has been called/expanded
DeclareModule <name >	Defines the public interface to Module <name>
EndDeclareModule	Ends the public module declaration

Keywords (cont)	
Module <name>	Starts the implementation of the Module
EndModule	Ends the Module code block
USeModule <name>	Can use any module that had a previous public declaration
UnUseModule <name>	Removes the previously used module
End [ExitCode]	Ends the program correctly. If ExitCode is specified, the program will return it as the exit code to the OS.
Swap <expression1>, <expression2>	Does an optimized swap of the expressions