Power Basic to PureBasic

Reference Aid

Amílcar Matos Pérez
San Juan, Puerto Rico

About the Author

Amílcar specializes in building database and xml driven software for business enterprises. He has used PowerBasic compilers since its beginning. Amílcar Matos Pérez can be reached in the Pure Basic forum. http://www.purebasic.fr/english/index.php

About this Book

First edition: December 2014. Compiler reference: PureBasic 5.30. Written and edited in San Juan, Puerto Rico.

How is this aid organized?

The text is in three sections; the first level translations (meaning they are very easy). The second level translations (meaning they require your attention). And the third level – which is the process to follow to achieve a successful translation.

Contents

Introduction

Considering the migration of your code base to PureBasic? This booklet is intended to help you.

This is a work in progress, deliberately simple and very basic. No high level constructs or theory here. Just some issues enumeration. But I know that if at least this aid had been available for me initially, my migration would have been faster and easier. Well now it is yours! With this reference you will know which keywords can be readily exchanged without fear or testing or any doubts; which can not be exchanged; which no matter how much time you invest searching will have to be recoded. Of course this is no exhaustive enumeration nor all inclusive. I would have desired to do it better, but the usual business constraints don't let me do so. Anyway something is better than nothing. Remember that your own coding practices make a difference and with the Pure Basic community ready to help, you will farewell.

I want to contribute this writing to both the greater basic language community and the Pure Basic Community. I understand that others might be in the need to migrate their code base into another language. Re-coding into other non basic languages is a loss for all. With some help like this document that losses can be prevented. No one wants Power Basic to die. But nobody wants to loose all those talented programmers to the "dark side" either. So let's provide needed tools for them to stay in the greater basic language community. With that goal in mind I offer you dear reader this aid. Please receive it as a small gift of one of your peers. Happy holidays!

Amílcar Matos-Pérez 

San Juan, Puerto Rico

December 1st, 2014.

1. First the easy ones...

1.1 Comments

For comments Pure Basic uses the semicolon. All single quotes and REM statement can be safely substituted by the semicolon. Please remember that single quotes can appear within quoted strings.

PowerBasic	PureBasic	Note
‘ (single quote)	;	Can appear inside quoted strings
REM	;	

1.2 String functions

1.2.1 Decoration

PureBasic does not use any decoration on its string functions. Delete the dollar sign on each string function.

PowerBasic	PureBasic
Chr$	Chr
Lcase$	Lcase
Trim$	Trim
Ucase$	Ucase
Mid$	Mid

1.2.2 Not Allowed Syntax

PowerBasic	PureBasic	Not Allowed Syntax
Trim$	Trim	The ANY keyword. PureBasic allows only one character trimming.
LTrim$	LTrim	The ANY keyword. PureBasic allows only one character trimming.
RTrim$	RTrim	The ANY keyword. PureBasic allows only one character trimming.
Mid$	Mid	The assignment mode for character substitution is not allowed in PureBasic. Ex. Mid$(Var$, 5,1) = “P” <= this is not allowed.

1.3 Variables

1.3.1 Local statement

All Local statements can be safely translated as Protected. The syntax should be adjusted afterwards.

1.3.2 Variable Decorations

The dollar sign use for string variables is the same in both compilers, so no need to change them, leave them as they are.

Other variable decorations used in Power Basic must be deleted. Take care with monetary variables (@ and @@).

1.3.3 Variable Names

Some variable names may conflict with reserved words in PureBasic. In my experience this is not a big concern; any conflicts are very easily solved.

1.3.4 Variable Definitions

As Long, As String, etcetera; have to be translated to the PureBasic period-letter format. The Pure Basic Help document contains a very well designed table. Just press F1 and write variables in the index tab.

Take care that PowerBasic uses the same keywords (As Long, As String...) for Functions. In those instances the PureBasic format (period-letter) follows the Procedure keyword.

PowerBasic	PureBasic
as long	.l
as string	.s
Function One() **As Long**	Procedure**.l** One()

1.4 Commands

Some commands translate easily because they have the same syntax, although not the same name.

PowerBasic	PureBasic
Sleep	Delay
Timer	ElapsedMilliseconds
Tally	CountString
Instr	FindString
StrInsert	InsertString
Close	CloseFile
Kill	DeleteFile
Parse$	StringField
ParseCount	CountString

1.5 Structures

Power Basic uses the construct “IF...THEN”; Pure Basic uses the construct “IF...” without the THEN keyword. Simply delete every instance of the THEN keyword.

Power Basic uses the construct “SELECT CASE ...”; Pure Basic uses the construct “Select ...” without the CASE keyword in the select statement line. Note that it’s only in the select statement line, since each case of the select construct is named “Case”. Therefore the CASE keyword can be deleted in all the select statements.

END IF is used in both compilers in the same way, but they are written differently. Power Basic uses a space between both words while Pure Basic omits the space. Delete the space between both words. The same happens with END SELECT; the space between both words has to be deleted.

The END SUB and END FUNCTION are translated to the EndProcedure keyword.

Since Pure Basic makes no distinction between Function and Sub’s all can be translated directly to the Procedure keyword.

PowerBasic	PureBasic	Note
THEN		Delete the keyword.
Select Case	Select	Delete the CASE keyword.
End If	EndIf	Delete the space.
Else If	ElseIf	Delete the space.
End Select	EndSelect	Delete the space.
End Sub	EndProcedure	Replace with PureBasic keyword.
End Function	EndProcedure	Replace with PureBasic keyword.
Function	Procedure	Replace with PureBasic keyword.
Sub	Procedure	Replace with PureBasic keyword.

2. Translation – the second level

2.1 Constants

2.1.1

They look easy at first sight, they are not. The translations of constants should be done with some care. Power Basic uses the dollar sign; Pure Basic uses the sharp sign. But some constants are already assigned by the Pure Basic compiler with minor differences that must be taken into account.

PowerBasic	PureBasic	Notes
$CR	#CR$	Pure Basic also has #CR
$CRLF	#CRLF$	
$SPC		No equivalent use: Space(1)
$TAB	#TAB$	
%MOD_CONTROL	#MOD_CONTROL	
%VK_L	#VK_L	

2.2 Structures

2.2.1 Direct translations

PowerBasic	PureBasic	Notes
Exit Function	ProcedureReturn	Power Basic functions returns with any previously pre-assigned return value. Pure Basic does not. Meaning, if the code depended on that invisible feature, now it must be made explicit.
Exit Sub	ProcedureReturn	

2.2.2 Careful translations

PowerBasic	PureBasic	Notes
Function =	ProcedureReturn	Note the equal sign after the function keyword. This is used to pre-assign the function return value. Pure Basic uses ProcedureReturn and omits the equal sign. But ProcedureReturn returns immediately whereas Function = does not.

2.2.3 Loops

Do / Loop constructs in Power Basic can be translated to Pure Basic with the Repeat/Until construct. Substitute DO with the Repeat keyword, and in the LOOP UNTIL line delete the LOOP keyword. Care should be exercised if the LOOP line does not have the UNTIL keyword.

PowerBasic	PureBasic	Notes
DO	Repeat	
LOOP UNTIL	Until	Do not replace if the until keyword is absent in this code line.

2.2.4 Call me back!

Callback structures are procedures in Pure Basic. Translate the Callback Function keyword to Procedure. But something more has to be done: Identify that procedure to the compiler as a callback procedure with the SetWindowCallback statement.

PowerBasic	PureBasic
Callback Function	Procedure.l WinCallback(hWnd, WMsg, WParam, LParam)
Callback Function	SetWindowCallback(@WinCallback()) ; to activate the callback

2.2.5 API Functions

In Pure Basic every API function requires an ending underscore and parenthesis for the parameters. In my experience this works flawlessly. Put that underscore add the parenthesis around the parameters and away you go. The API parameters need the change to the sharp sign (explained previously), and the substitution of some Power Basic terms. For example CBHNDL which in Pure Basic is WindowId ().

Is or or, or another thing? (no typo here you read right, there are three sequential or's)

In Power Basic API functions OR is used to join two constants. Pure Basic uses the vertical bar symbol "|" (means bitwise or). So if you want to get a full night sleep remember to translate those or's, OR otherwise the bugs will not let you sleep.

PowerBasic	PureBasic
API functions	Require an ending underscore and parenthesis for the parameters
%MOD_CONTROL	#MOD_CONTROL
CBHNDL	WindowID()
OR	Use vertical bar (bitwise OR)

2.3 Commands

2.3.1 Let's deal with those commands that change completely but easily. That's the case of INCR. Power Basic uses this command to increase the value of the variable by one. Pure Basic does not have it. So every instance of INCR Var has to be rewritten as Var = Var + 1. (Pure Basic shorthand: Var+1).

PowerBasic	PureBasic	Notes
Incr	var = var + 1	
Decr	var = var – 1	
Dim(0 to Var)	Dim(Var)	Caution. If the starting number is other than zero, then conversion must be done manually.

The Replace statement is also one that changes. The conversion is the statement name and the position of one parameter, see below;

PowerBasic	REPLACE That$ WITH This$ IN String$
PureBasic	String$ = ReplaceString(String$, That$, This$)

2.3.2 For – Next Loops and One off bugs.

Pure Basic arrays always start at zero. Power basic arrays not necessarily start at zero. Usually this is a non-issue in translation (the array will have one or more empty positions), but be alert to any possible one off bugs that appear. I'm not recommending this be left as is. The first goal is to translate the code and have a working solution as fast as possible, with the less time and money investment possible. Once you have that working solution then these details can be addressed.

2.3.3 Power Basic has the following syntax for calling a function;

CALL FunctionName&(Var$, Collection$(), Parameter3) To Result&

This syntax translate easily into Pure Basic as;

Result = FunctionName(Var$, Collection$(), Parameter3)

Note the following;

a. The function name decoration (&) is gone.
b. The TO keyword is gone.
c. The CALL keyword is gone.
d. The Result variable now is on the left with an equal sign added.

2.3.4 BY REF, BY VAL keywords

Pure Basic variables always pass into procedures BY VAL. To pass a variable BY REF into a procedure pass the variable pointer.

2.3.5 The new procedure commands;

Join	Requires a new procedure. Suggested name: JoinArrayIntoString
Parse	Requires a new procedure. Suggested name: ParseStringIntoArray
StrDelete	Requires a new procedure. Suggested name: StrDelete
Mid() = Var$	Requires a new procedure. Suggested name: SubstituteCharacter
Max	Requires a new procedure. Suggested name: Maximum
Min	Requires a new procedure. Suggested name: Minimum

2.3.6 Power Basic has the following syntax for opening a read only file;

```
FileNb& = FREEFILE
OPEN Filename$ FOR INPUT AS #FileNb&
```

This syntax translate easily into Pure Basic as;

```
FileNb = ReadFile(#PB_ANY, Filename$)
```

Note the following;

a. Variable decoration (&) is gone.
b. FOR INPUT keywords are gone. The ReadFile statement is just for this input only purpose.
c. The FREEFILE statement is gone. The #PB_ANY does the job.
d. Power Basic has other file syntax, I have shown an example. Manual conversion is not to be feared if the code is understood.

2.3.7 For reading a file Power Basic uses;

```
LINE INPUT #FileNb, String$
```

This syntax translate easily into Pure Basic as;

```
String$ = ReadString(FileNb)
```

2.3.8 To write to a file Power Basic uses;

```
Print#FileNb, String$
```

This syntax translate easily into Pure Basic as;

```
WriteStringN(FileNb, String$)
```

2.3.9 Power Basic uses the following syntax for message box;

```
MSGBOX ShowThisTextToTheUser$ , , BoxTitle$
```

This syntax translate easily into Pure Basic as;

```
MessageRequester (BoxTitle$, ShowThisTextToTheUser$, #PB_MessageRequester_Ok)
```

2.3.10 Power Basic has the EXIT <structure> construct. Pure Basic uses the Break keyword. Power Basic has the ITERATE keyword. Pure Basic uses the Continue keyword. The reason they are listed last is because, in my opinion, they should be translated at the end of the process. Remember your code is going thru a lot of changes and we all want to be bug free. Both statements change the execution order and that change is easily “seen” once all the other changes have been done. This way you’ll have a clearer understanding of where the execution goes.

3. The Process - Sharing the experience!

3.1 I make a copy of the code file and save it on the translation folder.

3.2 There I make another copy of that first copy and labeled it working copy.

3.3 On that working copy change all the FUNCTION and SUB statements first. Because then the IDE procedures tab will list all of the procedures and navigation will be faster and easier.

3.4 Then change all END FUNCTION and END SUB statements. Then you can use the folding feature of the IDE.

3.5 Then save another numbered copy of work in progress. Keep making changes and saving numbered copies.

3.6 For the detail changes – get the whole procedure, paste it onto a new tab of the IDE; work there and when all changes are done bring back the whole procedure to the working copy.

3.7 Once you're done use the syntax check (Menu-> Compiler -> Syntax Check). That will show any missing details.

3.8 Testing and debugging. I follow the standard professional practices.

As you see it's nothing fancy, just plain disciplined approach to the whole process.

The process can be automated?

Partially, Yes! It depends on many factors, but the most important is the coding style used. Carefully crafted code can be automated. Sloppy or spaghetti code? Well, it can be translated but will take more time and automation might be very limited. The best way to know is to try. Translate one of your shorter programs first. With this reference aid and that first experience automate part of the process, tailoring it to your coding style and work practices.

Conclusion

I know there are many more details that need to be addressed, that is why this reference aid is a work in process. With this aid you have a format example to organize your own reference guide. You receive assurance that translation can be done within the common business constraints. And more importantly that others are available to help with the doubts and issues that arise.

As explained migrating is easier than what it looks like. Just need to be careful and organized in your approach. Happy coding!

The End