599 Menlo Drive, Suite 100
Rocklin, California 95765, USA
Office: (916) 624-8333
Fax: (916) 624-8003

General: info@parallax.com
Technical: support@parallax.com
Web Site: www.parallax.com
Educational: www.stampsinclass.com

The Elements of PBASIC Style

Introduction

Like most versions of the BASIC programming language, PBASIC is very forgiving and the compiler enforces no particular formatting style. As long as the source code is syntactically correct, it will compile and download to the BASIC Stamp without trouble.

Why, then, would one suggest a specific style for PBASIC? Consider this: Over three million BASIC Stamps have been sold and there are nearly 2000 members that particpate in Parallax forums. This makes it highly likely that you'll be sharing your PBASIC code with someone, if not co-developing a BASIC Stamp project. Writing code in an organized, predictable manner will save you – and your potential colleagues – a lot of time; in analysis, in troubleshooting and especially when you return to a project after a long break.

The style guidelines presented here are just that: ***guidelines***. They have been developed from style guidelines used by professional programmers using other high-level languages such as Visual Basic®, C/C++, and Java™. Use these guidelines as-is, or modify them to suit your individual needs. The key is selecting a style the works well for you or your organization, and then sticking with it.

PBASIC Style Guidelines

1. Do It Right the First Time

Many programmers, especially new ones, fall into the "*I'll knock it out now and fix it later.*" trap. Invariably, the "*fix it later*" part never happens and sloppy code makes its way into production projects. If you don't have time to do it right, when will you find time to do it again?

Start clean and you'll be less likely to introduce errors in your code. And if errors do pop up, clean and organized formatting will make them easier to find and fix.

2. Be Organized and Consistent

Using a blank program template will help you organize your programs and establish a consistent presentation. The BASIC Stamp Editor (as of Version 2.1, Beta 1) allows you to specify a file template for the File | New function (see Edit | Preferences | Files & Directories...).

3. Use Meaningful Names

Be verbose when naming constants, variables and program labels. The compiler will allow names up to 32 characters long. Using meaningful names will reduce the number of comments and make your programs easier to read, debug and maintain.

4. Naming IO Pins

BASIC Stamp IO pins are a special case as various elements of the PBASIC language require a pin to be constant value, an input variable or an output variable. Begin IO pin names with an uppercase letter and use mixed case, using uppercase letters at the beginning of new words within the name.

BS1:
```
SYMBOL  HeaterCtrl  = 7              ' constant value
SYMBOL  AlarmLed    = PIN6           ' input or output bit
```

When using the BS2 the **PIN** definition is used. This will cause the compiler to use the correct variant for the pin (x, INx, or OUTx)

BS2:
```
HeaterCtrl    PIN   15
```

Since connections don't change during the program run, IO pins are named like constants (#5) using mixed case, beginning with an uppercase letter.

5. Naming Constants

Begin constant names with an uppercase letter and use mixed case, using uppercase letters at the beginning of new words within the name.

BS1:
```
SYMBOL  AlarmCode   = 25
```

BS2:
```
AlarmCode     CON   25
```

6. Naming Variables

Begin variable names with a lowercase letter and use mixed case, using uppercase letters at the beginning of new words within the name.

BS1 Tip: Avoid using W0 (B0 and B1) so that bit variables (BIT0..BIT15) are available for use in your programs. Bit variables 0..15 overlay W0, so the use of W0 may cause undesired effects.

```
SYMBOL  waterLevel  = W1
```

BS2 Tip: Avoid the use of internal variable names (such as B0 or W1) in your programs. Allow the compiler to automatically assign RAM space by declaring a variable of specific type.

```
waterLevel    VAR   Word
```

7. Variable Type Definitions

When using the BS1, variable type is declared by aliasing the **SYMBOL** name to an internal variable of a specific size. Note that without aliasing, the Memory Map function of the BASIC Stamp editor will not show all RAM variables used by the program.

BS1:
```
SYMBOL  status      = BIT0
SYMBOL  ovenTemp    = B2
SYMBOL  rndValue    = W2
```

For the BS2, variable types should be in mixed-case and start with an uppercase letter.

BS2:
```
status        VAR   Bit
counter       VAR   Nib
ovenTemp      VAR   Byte
rndValue      VAR   Word
```

Conserve BASIC Stamp user RAM by declaring the variable type required to hold the expected values of the variable.

BS1:
```
SYMBOL  bitValue    = BIT0            ' 0 - 1
SYMBOL  byteValue   = B2              ' 0 - 255
SYMBOL  wordValue   = W2              ' 0 - 65535
```

BS2:
```
bitValue      VAR   Bit               ' 0 - 1
nibValue      VAR   Nib               ' 0 - 15
byteValue     VAR   Byte              ' 0 - 255
wordValue     VAR   Word              ' 0 - 65535
```

8. Program Labels

Begin program labels with an uppercase letter, used mixed case, separate words within the label with an underscore character and begin new words with a number or uppercase letter. Labels should be preceded by at least one blank line, begin in column 1 and must be terminated with a colon (except after **GOTO** and **THEN** [in classic PBASIC] where they appear at the end of the line and without a colon).

BS1:
```
Print_String:
  READ eeAddr, char
  IF char = 0 THEN Print_Done
  DEBUG #@char
  eeAddr = eeAddr + 1
  GOTO Print_String

Print_Done:
  RETURN
```

9. PBASIC Keywords

All PBASIC language keywords, including **SYMBOL**, **CON**, **VAR**, **PIN** and serial/debugging format modifiers (**DEC**, **HEX**, **BIN**) and constants (**CR**, **LF**) should be uppercase. The BASIC Stamp editor will correctly format PBASIC keywords automatically, and allow you to set color highlighting by category to suit your personal tastes.

BS1/BS2:

```
Main:
  DEBUG "BASIC Stamp", CR
  END
```

10. Indent Nested Code

Nesting blocks of code improves readability and helps reduce the introduction of errors. Indenting each level with two spaces is recommended to make the code readable without taking up too much space.

BS2:

```
Main:
..DO
....FOR testLoop = 1 TO 10
......IF (checkLevel < threshold) THEN
........lowLevel = lowLevel + 1
........LEDokay = IsOff
......ELSE
........LEDokay = IsOn
......ENDIF
......PAUSE 100
....NEXT
..LOOP WHILE (testMode = Yes)
```

Note: The dots are used to illustrate the level of nesting and are not a part of the code.

11. Condition Statements

Enclose condition statements in parenthesis for clarity (BS2 only – parenthesis are not allowed when using the BS1).

BS2:

```
Check_Temp:
  IF (indoorTemp >= setPoint) THEN
    AcCtrl = IsOn
  ELSE
    lowLevel = lowLevel + 1
  ENDIF
```

```
Fill_Water_Tank:
  DO WHILE (waterLevel = IsLow)
    TankFill = IsOn
    PAUSE 250
  LOOP
```

```
Get_Delay:
  DO
    DEBUG HOME, "Enter time (5 – 30)... ", CLREOL
    DEBUGIN DEC2 tmDelay
  LOOP UNTIL ((tmDelay >= 5) AND (tmDelay =< 30))
```

12. Be Generous With White Space

White space (spaces and blank lines) has no effect compiler or BASIC Stamp performance, so be generous with it to make listings easier to read. As suggested in #8 above, allow at lease one blank line before program labels (two blanks lines before a subroutine label is recommended). Separate items in a parameter list with a space.

BS2:
```
Main:
  DO
    ON task GOSUB Update_Motors, Scan_IR, Close_Gripper
  LOOP

Update_Motors:
  PULSOUT leftMotor, leftSpeed
  PULSOUT rightMotor, rightSpeed
  PAUSE 20
  task = (task + 1) // NumTasks
  RETURN
```

An exception to this guideline is with the bits parameter used with **SHIFTIN** and **SHIFTOUT**, the **REP** modifier for **DEBUG** and **SEROUT**, and the byte count and terminating byte value for **SERIN**. In these cases, format without spaces.

```
  SHIFTIN A2Ddata, A2Dclock, MSBPOST, [result\9]

  DEBUG REP "*"\25, CR

  SERIN IRbSIO, IRbBaud, [buffer\8\255]
```

13. Use Conditional Compilation for Compatibility

Some commands such as **SERIN** and **SEROUT** use different parameters based on the target BASIC Stamp. Use conditional compilation for maximum compatibility of your programs.

BS2:
```
#SELECT $STAMP
  #CASE BS2, BS2E, BS2PE
    T1200       CON     813
    T2400       CON     396
    T9600       CON     84
  #CASE BS2SX, BS2P
    T1200       CON     2063
    T2400       CON     1021
    T9600       CON     240
#ENDSELECT
```