PARALLAX INC
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 SX/B Style

Introduction

Like most versions of the BASIC programming language, SX/B is very forgiving and the compiler enforces no particular formatting style. As long as the source code is syntactically correct, it will usually compile and can be programmed into to the SX microcontroller without trouble.

Why, then, would one suggest a specific style for SX/B? Consider this: Millions of SX microcontrollers have been sold, SX/B makes the SX accessible to a wider audience (i.e., less-experienced programmers), and there are over 4000 members that participate in Parallax forums. This makes it highly likely that you'll be sharing your SX/B code with someone, if not co-developing a SX-based project. Writing code in an organized, predictable manner will save you – and your potential teammates – 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™. We suggest you use these guidelines as-is, or – especially if you're advanced and have been programming a while – 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.

SX/B 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 into 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 SX-Key IDE allows you to specify a file template for the **File | New (SX/B)** selection.

3. Use Meaningful Names

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

4. Naming I/O Pins

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

```
HeaterCtrl    VAR   RA.0
```

Since connections don't change during the program run, I/O pins are named like constants (#5) using mixed case, beginning with an uppercase letter. Resist the temptation to use direct pin names (e.g., RB.7) in the body of a program as this can lead to errors when making circuit changes.

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.

```
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.

```
waterLevel    VAR   Byte
```

7. Variable Types

SX/B supports byte and bit variables. To define bit variables, the byte that holds them must be defined first.

```
alarms        VAR   Byte
overTemp      VAR   alarms.0
underTemp     VAR   alarms.1
```

8. General 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.

```
Get_Tag:
  RfidEn = Active
  DO
    char = RX_RFID
  LOOP UNTIL char = $0A
  FOR idx1 = 0 TO 9
    tagBuf(idx1) = RX_RFID
  NEXT
  RfidEn = Deactivated
```

9. SX/B Keywords

All SX/B language keywords, including **CON** and **VAR** should be uppercase. The SX-Key IDE does syntax highlighting, but does not change case so this is the responsibility of the programmer.

```
Main:
  DO
    HIGH AlarmLed
    WAIT_MS 100
    LOW AlarmLed
    WAIT MS 100
  LOOP
```

10. Declare Subroutines

As of version 1.2 subroutines may be declared. This benefits the programmer in two ways: 1) the compiler creates a jump table that allows the subroutine code to be placed anywhere in the program space and, 2) the compiler does a syntax check on the subroutine call to ensure that the proper number of parameters are being passed.

```
WAIT_MS       SUB   1, 2
```

When using a declared subroutine, the use of **GOSUB** is optional

11. Declared Subroutine Labels

Declared subroutines are, in effect, added language elements and should be treated like new keywords: all uppercase. To distinguish subroutine labels from SX/B keywords use an underscore between new words. As with general program labels, subroutine labels begin in column 1 and must be terminated with a colon.

```
' Use: WAIT_MS milliseconds {, multiplier }
' -- multiplier is optional

WAIT_MS:
  temp1 = __PARAM1
  IF __PARAMCNT = 1 THEN
    temp2 = 1
  ELSE
    temp2 = __PARAM2
  ENDIF
  IF temp1 > 0 THEN
```

```
    IF temp2 > 0 THEN
      PAUSE temp1 * temp2
    ENDIF
  ENDIF
  RETURN
```

As shown above, it is good practice to document the subroutine with usage requirements, especially when optional parameters are available.

12. 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.

```
' Use: LCD_OUT [ aByte | string | label ]
' -- "aByte" is single-byte constant or variable
' -- "string" is an embedded literal string
' -- "label" is DATA statement label for stored z-String

LCD_OUT:
..temp1 = __PARAM1
..IF __PARAMCNT = 2 THEN
....temp2 = __PARAM2
....DO
......READ temp2 + temp1, temp3
......IF temp3 = 0 THEN EXIT
......SEROUT LcdTx, LcdBaud, temp3
......INC temp1
......temp2 = temp2 + Z
....LOOP
..ELSE
....SEROUT LcdTx, LcdBaud, temp1
..ENDIF
..RETURN
```

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

13. Be Generous With White Space

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