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

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

BASIC Stamp Editor / Development System
Version 2.0 Beta – Release 2

Introduction

The Parallax BASIC Stamp Editor / Development System, Version 2.0 was created to enhance the PBASIC programming language and programming the BASIC Stamp®. To this end, many features considered standard in other (desktop) versions of BASIC have been incorporated into PBASIC (a purely embedded language). The enhancements to the PBASIC programming language are collectively called PBASIC 2.5.

Disclaimer

Parallax, Inc. is not responsible for special, incidental, or consequential damages resulting from any breach of warranty, or under any legal theory, including lost profits, downtime, goodwill, damage to or replacement of equipment or property, and any costs of recovering, reprogramming, or reproducing any data stored in or used with Parallax products.

Purpose

The purpose of this document is to describe and demonstrate the new features of the PBASIC 2.5 as supported BASIC Stamp Editor / Development System Version 2.0 (Beta). By design, this document is concise, yet complete enough for the experienced PBASIC programmer to take advantage of the new language features.

If you are new to PBASIC and BASIC Stamp programming, we suggest you wait for the final, fully documented release of this product. The full release will provide context-sensitive online help of all features.

Support

For answers to your questions and to give feedback about the new PBASIC 2.5 features, please send an e-mail to:

pbasic25beta@parallax.com

As this product is NOT FULLY RELEASED, please do not call Parallax customer service or post your questions or comments to any of the public BASIC Stamp lists (on Yahoo! Groups or Parallax web site). The only way to ensure getting you question answered or to have your feedback evaluated is send it to the e-mail address above.

PBASIC 2.5 Update Overview

New / Updated

- $PBASIC directive
- PIN type
- New DEBUG control characters *
- DEBUGIN
- Line continuation for comma-delimited lists
- IF…THEN…ELSE *
- SELECT…CASE
- DO…LOOP
- EXIT to terminate loops
- ON…GOSUB
- ON…GOTO
- READ / WRITE enhanced
- PUT / GET enhanced *
- Program labels require colon
- Conditional Compilation Directives *
- Enhanced Interface / Color Syntax Highlighting *

$PBASIC Directive

In order to enable new language features and to allow existing programs to compile as-is, a $PBASIC directive has been created. To compile programs that use only PBASIC 2.0 features, this directive may be omitted. To enable new features, the 2.5 directive is required.

```
' {$PBASIC 2.0}                                  ' optional - use 2.0 features
' {$PBASIC 2.5}                                  ' enable version 2.5 features
```

For convenience, new two buttons have been added to the toolbar that will allow you to insert the desired $PBASIC directive.

PIN Type

The PIN type definition simplifies programs where the same I/O pin is used as an input and output by allowing the compiler to determine whether the pin value required for a given command is a numeric constant (i.e., 0), input variable bit (i.e., In0) or output variable bit (i.e., Out0).

Example Syntax and Use:

```
SDA             PIN     8
SCL             PIN     9
```

```
I2C_Start:
  INPUT SDA                                      ' make pins inputs
  INPUT SCL
  LOW SDA
```

```
Clock_Hold:
  DO : LOOP WHILE (SCL = 0)                    ' monitor input bit
  RETURN
```

New DEBUG Control Characters

Note: These new control characters may only work properly within the BASIC Stamp development system, and not with external applications (i.e., HyperTerminal). The number in parenthesis is the numeric value of the string constant.

- CRSRXY (2) Move to cursor to X, Y (X-byte and Y-byte follow command)
- CRSRLF (3) Move cursor position left
- CRSRRT (4) Move cursor position right
- CRSRUP (5) Move cursor position up one line
- CRSRDN (6) Move cursor position down one line
- LF (10) Linefeed character
- CLREOL (11) Clear all characters to the end of current line
- CLRDN (12) Clear all characters from the current position to the end of window
- CRSRX (14) Move to cursor to column X (X-byte follows command)
- CRSRY (15) Move to cursor to line Y (Y-byte follows command)

Example:

```
  DEBUG CRSRXY, 1, 1
  DEBUG "BASIC Stamp Editor Version 2.0", CR
```

This example moves the cursor to the second line, second column and prints a string. The values that follow CRSRXY are zero-indexed (0, 0 is the first line, first column).

DEBUGIN

To facilitate user input from the DEBUG terminal the DEBUGIN keyword has been added. This works identically to SERIN, but does not require the user to specify the pin (fixed to PIN 16), baud rate (fixed to 9600 baud, the standard DEBUG baud) or the use of brackets to enclose modifiers or input variables.

DEBUGIN uses the same serial formatting modifiers as DEBUG, SERIN, and SEROUT.

Example:

```
Get_Hours:
  DEBUG Home, "Enter hours: ", CLREOL
  DEBUGIN DEC hrs
  IF (hrs > 23) THEN Get_Hours
```

Note: You may wish to select "Echo Off" on the DEBUG terminal window when using DEBUGIN.

Line Continuation

Any line of code can be continued onto the next line by breaking the first line just after the comma (,) separating arguments or list items.

Examples:

```
  DEBUG "Hello, World", CR,
        "PBASIC 2.5 is ready for action!"
```

```
  BRANCH idx, [Target1, Target2, Target3,
               Target4, Target5, Target5]
```

```
  SELECT idx
    CASE  1,  2,  6,
         10, 11, 12
      HIGH 0

    CASE  3,  4,  5,
         13, 14, 15
      HIGH 1
  ENDSELECT
```

IF…THEN…ELSE

PBASIC 2.5 includes a standardized IF…THEN…ELSE decision structure. The general syntax may take either of two forms as follows (items in curly braces {} are optional):

```
  IF condition THEN
    statement(s)
{ ELSEIF condition THEN
    statement(s) }
{ ELSE
    statement(s) }
  ENDIF
```

```
  IF condition THEN statement(s) { ELSEIF statement(s) } { ELSE statement(s) }
```

Notes:

- Multiple statements may be included in the THEN, ELSIF or ELSE blocks of the single-line syntax by separating each statement with a colon (:).
- ENDIF is not used when using the single-line syntax.
- Up to 16 IF…THEN…ELSE structures may be nested.

Examples:

```
  IF (score > 90) THEN
    DEBUG "Your grade is an A!", CR
  ELSE
    DEBUG "Perhaps more study is in order...", CR
  ENDIF
```

```
IF (idx = 1) THEN HIGH 10 : LOW 11 ELSE LOW 10 : HIGH 11
```

SELECT…CASE

SELECT…CASE can be used to cleanly replace multiple IF…THEN…ELSE structures. The PBASIC syntax for SELECT…CASE is (| denotes mutually-exclusive items):

```
SELECT expression
  CASE condition | ELSE
    statement(s)
ENDSELECT
```

Notes:

- expression can be a variable, a constant or an expression.
- condition can be in the form:

  {condition-op} #

  -- where condition-op is an optional conditional operator: =, <>, <, >, >= or <=
  -- # is a variable, a constant or an expression

  or…

  # TO #

  -- Indicates a range of the first number to the next number, inclusive
  -- Conditional operators are not allowed in this form.

- Multiple conditions within the same case can be separated by commas (,).
- When a CASE is True, the default function is for the CASE's statement(s) to be executed, then program execution jumps to the first statement following ENDSELECT.

Example:

```
SELECT irCmd
  CASE 0 TO 3
    HIGH irCmd

  CASE AllOff, Mute
    OutA = %0000

  CASE ELSE
    DEBUG "Bad Command", CR
ENDSELECT
```

DO…LOOP

PBASIC 2.5 now includes a standardized looping construct that takes the general form:

```
DO { WHILE | UNTIL condition }
  statement(s)
LOOP { UNTIL | WHILE condition }
```

The condition statement can be setup to cause the loop to run for a specific number of iterations, while or until a condition becomes true, or indefinitely if no condition is applied. Up to 16 DO…LOOPs may be nested.

Examples:

```
DO
  TOGGLE AlarmPin
  PAUSE 100
LOOP
```

```
DO WHILE (Status = Okay)
  StatusLED = IsOn
  PAUSE 100
LOOP
```

In the example above, the condition will be tested before the loop code is executed.

In the next example, the loop code will run at least once because the condition test happens at the end.

```
DO
  AlarmLED = IsOn
  PAUSE 1000
LOOP UNTIL (ovenTemp < ResetThreshold)
```

EXIT

EXIT causes the immediate termination of a FOR…NEXT or DO…LOOP. Up to 16 EXITs may be placed in a loop structure.

Example:

```
FOR samples = 1 TO 10
  GOSUB Read_Temp
  GOSUB Display_Temp
  IF (temp > 100) THEN EXIT
  PAUSE 1000
NEXT
```

In the example above the loop will run 10 times unless the temperature reading exceeds 100 which will cause the FOR…NEXT loop to terminate.

ON…GOSUB

ON…GOSUB has been added for compatibility with other versions of BASIC and is very similar to the PBASIC BRANCH command except that the address of the following line is stored and a targeted GOSUB is executed based on an offset value. If the offset value is greater than the range of target addresses, code will continue on the next line.

The general syntax is:

```
ON offset GOSUB Target0 {, Target1, Target2, ...TargetN }
```

Notes:

- offset is a variable/constant/expression (0 - 255) that specifies the index of the address, in the list, to branch to (0 - N).
- ON...GOSUB will ignore any list entries beyond offset 255.

Example:

```
DO
  ON task GOSUB Update_Motors, Read_IR, Read_Light, Read_Temp
  task = task + 1 // NumTasks
LOOP
```

ON…GOTO

ON…GOTO has been added for compatibility with other versions of BASIC and is the functional equivalent of the PBASIC BRANCH command, with this syntax:

```
ON offset GOTO Target0 {, Target1, Target2, ...TargetN }
```

Notes:

- offset is a variable/constant/expression (0 - 255) that specifies the index of the address, in the list, to branch to (0 - N).
- ON...GOTO will ignore any list entries beyond offset 255.

Example:

```
ON alarmLevel GOTO Code1, Code2, Code3
```

which is the same as:

```
BRANCH alarmLevel, [Code1, Code2, Code3]
```

READ and WRITE Enhancements

READ and WRITE have been enhanced to allow each to work with Bytes or Words, and to allow multi-variable READs or WRITEs with one line of code. The enhanced syntax is:

```
READ location, {Word} variable {, {Word} variable, {Word} variable ... }
WRITE location, {Word} variable {, {Word} variable, {Word} variable ... }
```

Examples:

```
READ 0, hours, minutes, seconds
```

```
WRITE 3, month, day, Word year
```

When the (optional) Word parameter is used, values are read or written low-byte, then high-byte ("Little-Endian").

PUT and GET Enhancements

PUT and GET have been enhanced to allow each to work with Bytes or Words, and to allow multi-variable PUTs or GETs with one line of code. The enhanced syntax is:

```
  PUT location, {Word} variable {, {Word} variable, {Word} variable ... }
  GET location, {Word} variable {, {Word} variable, {Word} variable ... }
```

Examples:

```
  PUT 0, hours, minutes, seconds
```

```
  GET 3, month, day, Word year
```

When the (optional) Word parameter is used, values are read or written low-byte, then high-byte ("Little-Endian").

Program Labels Require Colon (:)

PBASIC 2.5 requires program labels (for GOTO, BRANCH, etc.) to be terminated with a colon (:). Labels without a colon will generate a compile-time error.

Example:

```
Get_Hours:                                        ' good label
  DEBUG Home, "Enter hours: ", CLREOL
  DEBUGIN DEC hrs
  IF (hrs > 23) THEN Get_Hours

Get_Mins                                          ' illegal label
  DEBUG Home, "Enter minutes: ", CLREOL
  DEBUGIN DEC mins
  IF (mins > 59) THEN Get_Mins
```

Conditional Compilation Directives

In order to facilitate the creation of programs that will run on the BASIC Stamp connected to the system, several condition compilation directives have been created.

```
#DEFINE symbol = value
```

Example:

```
#DEFINE NOSPRAM = ($STAMP = BS2)
```

```
#IF (condition) #THEN
  statement(s)
{ #ELSE
  statement(s) }
#ENDIF
```

Example:

```
#IF (NOSPRAM) #THEN
  DEBUG "No SPRAM in this Stamp."
  END
#ENDIF
```

```
#SELECT expression
  #CASE condition
     statement(s)
#ENDSELECT
```

Example:

```
#SELECT $STAMP

  #CASE BS2, BS2e, BS2pe
    MidiBaud        CON     $8000 + 12

  #CASE BS2sx, BS2p
    MidiBaud        CON     $8000 + 60

#ENDSELECT
```

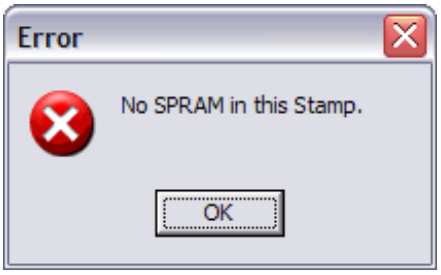
```
#ERROR message
```

This directive allows the programmer to create a custom error message that will be displayed by the editor. The program will stop and the editor will highlight the #ERROR line.

Example:

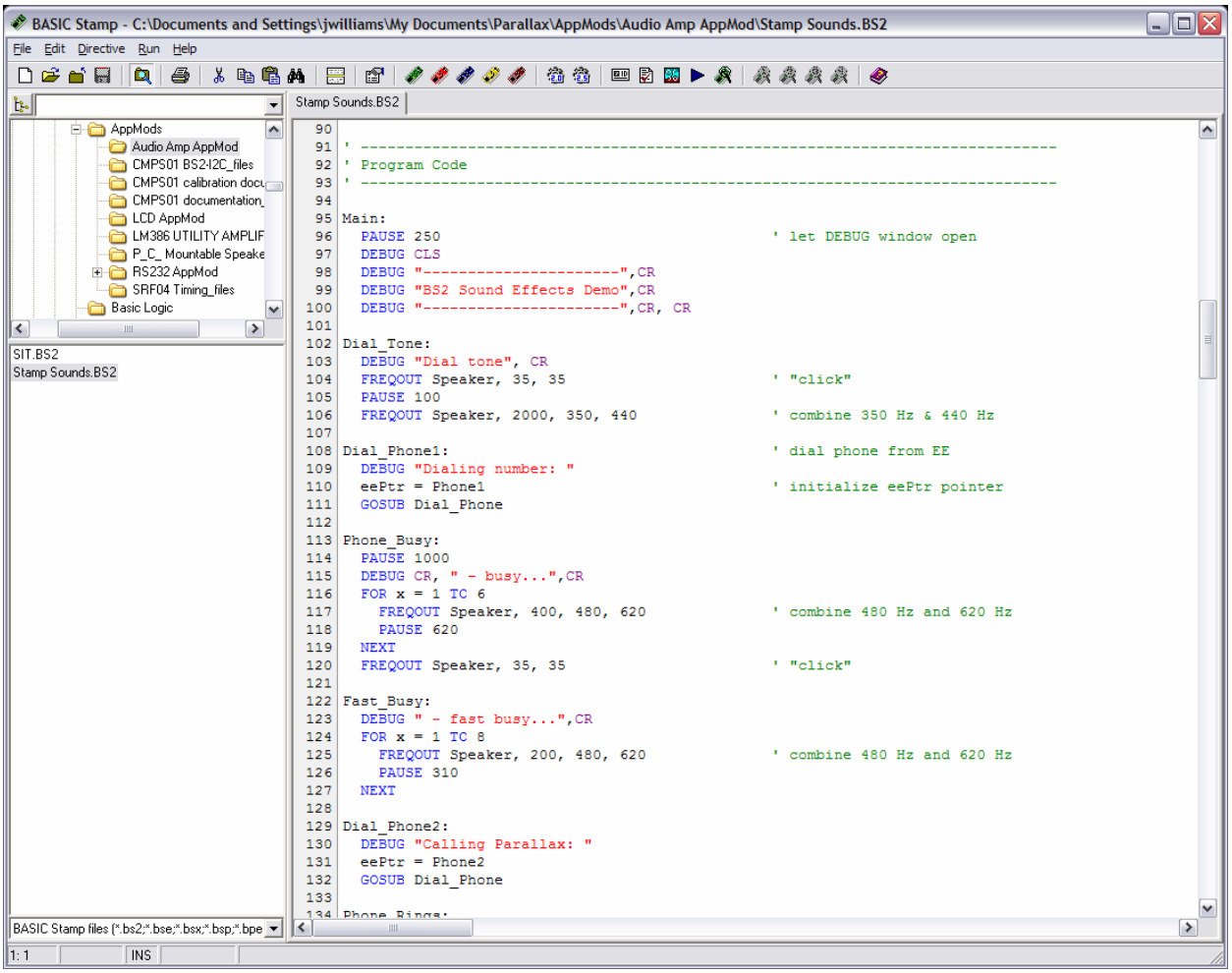
```
#IF ($STAMP = BS2) #THEN
  #ERROR "No SPRAM in this Stamp."
#ENDIF
```

Will display the following dialog if the connected BASIC Stamp is a BS2:



Enhanced Interface / Color Syntax Highlighting

The BASIC Stamp Editor / Development system now features an Explorer-type interface to make file navigation and opening easier. One or more files may be selected from the file pane, then dragged and dropped onto the editor pane; this action will case the selected file(s) to be opened.



Included in this release is color syntax highlighting. While the default highlight scheme will meet the needs of most programmers, it may be modified to suit personal preferences. To modify the highlight scheme, open the Preferences dialog (Edit \ Preferences...), then select the "Editor Appearance" tab and select the PBASIC scheme. Click the "Copy Scheme" button, approve the copy, and then modify the syntax highlighting for the Custom scheme as desired. To see the changes dynamically, make sure the "Show Preview Example" checkbox is selected.