PBASIC 言語　2.5 について

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

紹介

パララックスベーシックスタンプエディター／開発システム、改訂 2.0 は PBASIC プログラミング言語とベーシックスタンプ（BASIC Stamp®）の質を高めるために作られました。一般のベーシック言語版で考慮されるいくつかのインストラクションが PBASIC の中に組み込まれたのをこの解説の最後の部分で説明してあります。質を高めた PBASIC プログラミング言語は、PBASIC 2.5 と呼ばれます。

責任の否認

この解説は、読者への啓蒙と教育のために意図されたものです。十分に正確を期したつもりですが、パララックス社（Parallax, Inc.）及び日本マイクロボット教育社は、エラー、脱落、いかなるアプリケイションの内容の適合性などについての責任は持てませんのでご了承ください。また、本書の情報の使用及び誤使用結果によるいかなる損害も責任は持てません。なお、本書の情報を使って装置の使用、製造又は販売等に関してそれが特許、コピーの権利又はその他の権利などを侵害していないかどうかの判断は、読者に責任があります。

サポート

PBASIC 2.5 はまだ完成した商品ではありませんので、PBASIC 2.5 についてのご質問は、support@microbot-ed.com にメールを下さい。

改良されたPBASIC 2.5の概略

【１】$PBASIC directive
【２】PIN type
【３】New DEBUG control characters
【４】DEBUGIN
【５】Line continuation for comma-delimited lists
【６】IF…THEN…ELSE *
【７】SELECT…CASE
【８】DO…LOOP
【９】EXIT to terminate loops
【10】ON…GOSUB
【11】ON…GOTO
【12】READ / WRITE enhanced
【13】PUT / GET enhanced *
【14】Program labels require colon
【15】Conditional Compilation Directives *
【16】Enhanced Interface / Color Syntax Highlighting *

【１】$PBASIC Directive

（Directive（ディレクティブ）：使用する言語の Version を指定します）

新しい言語の特性使用を可能にするためと今までのプログラムをそのまま機械語に編集（Compile）するために、$ PBASIC ディレクティブが作られました。PBASIC 2.0 までのプログラムをコンパイルする時にはこのディレクティブは必要ありません。新しい特性を有効にするためには、2.5 のディレクティブが必要となります。

' {$PBASIC 2.0} ' オプション　― 2.0 の特性を使用。
' {$PBASIC 2.5} ' 改訂（version）2.5 の特性を有効にします。

利便性を図るために、新しい 2 つのボタンが希望する $ PBASIC ディレクティブを挿入するためにツールバーに加えられました。

【２】PIN Type

（Type：I/O ピンの指定）

PIN タイプの定義は、同じ I/O ピンが input と output に使われるようなプログラムを簡単にさせます。定数（例：０）なのか、input の変数ビット（例：In０）なのか、それとも output の変数ビット（例：Out０）なのか、与えられたコマンドに必要なピンの値をコンパイラーに決めさせる事によってプログラムを簡単にさせます。

構文と使用例：

```
'*****************************************************************
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
```

****

【3】DEBUG コントロールキャラクター

注意：これらの新しいキャラクターは BASIC スタンプの開発システム内だけで正しく働きます。外部の、例えば、Hyper Terminal 等では働かないかもしれませんのご注意ください。カッコの中の数字は記号列定数の数値です。

- CRSRXY（2） カーソルを X、Y（X バイトと Y バイト）の位置に動かす（この後にコマンドが続く）
- CRSRLF（3） カーソルの位置を左に動かす
- CRSRRT（4） カーソルの位置を右に動かす
- CRSRUP（5） カーソルの位置を 1 行上に動かす
- CRSRDN（6） カーソルの位置を 1 行下に動かす
- LF（10） ラインフィードキャラクター（Linefeed character）
- CLREOL（11） 現在の最後のラインに対する全てのキャラクターをクリアする
- CLRDN（12） 現在の位置からウィンドウの終わりまで全てのキャラクターをクリアする
- CRSRX（14） カーソルを X 列動かす（X バイト後にコマンドが続く）
- CRSRY（15） カーソルを Y 行動かす（Y バイト後にコマンドが続く）

例：

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

****

この例はカーソルを 2 行目の 2 列目に動かします。そして文字列をプリントします。CRSRXY に続く値は、起点がゼロです。（0 、 0 は 1 列目の 1 行目です。）

【4】DEBUGIN

DEBUG ターミナルからユーザーが入力し易いように、DEBUGIN キーワードが加えられました。これは、SERIN と全く同じ働きをしますが、PIN 16 に固定されているため、ユーザーがピンを指定する必要はありません。baud rate（通信速度）は DEBUG ボー標準の 9600 ボーを使うか、又は、カッコでモディファイアー（modifiers）かインプットの変数を囲って使います。

DEBUGIN は DEBUG、SERIN、SEROUT などと同じ形式のモディファイアーを使います。

例：

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

注意 : DEBUGIN を使っている時は、DEBUG ターミナルウィンドウの“Echo Off”を選ぶと良いでしょう。

【５】連続したリスト

どんなコードのラインでも、アーギュメントやリストなどをカンマ（,）で区別すれば 2 行にわたってコードを書いても構いません。

例：

```
*******************************************************************************
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
*******************************************************************************
```

【６】IF…THEN…ELSE

PBASIC 2.5 は標準の IF…THEN…ELSE を含みます。一般のシンタックスは次のような 2 つの形式のどちらかを取ります。（大カッコ［　］の中の項目はオプションです。）

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

上の例は、また次のようにも書けます。（ENDIF がないことに注意して下さい。）

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

特記：

- 複数のステートメントは、各ステートメントをコロン（:）で区別する事によって、1 行のシンタックスの THEN、ELSEIF、ELSE の中に含むことが出来ます。
- ENDIF は 1 行のシンタックスでは使われません。
- IF…THEN…ELSE 構文は最高 16 までのネスト（nest）が出来ます。

例：

```
**************************************************************************
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

**************************************************************************
```

【7】SELECT…CASE

複数の IF…THEN…ELSE 構文をすっきりとした SELECT…CASE に取り替えることが出来ます。
SELECT…CASE のための PBASIC シンタックスは、(｜このマークは共通項目の意味です)：

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

ENDSELECT
**********************************************************
```

特記：

- expression は変数、定数、別の expression を使うことが出来ます。
- condition は次のような形式にすることが出来ます；

｛condition－op｝#
ここに、condition－op はオプションの条件オペレーター：＝、＜＞、＜、＞、＞＝、＜＝　など。
は変数、定数又は expression（式）

又は、

#TO#

これは、初めの番号から次の番号の範囲を表示しています。
この形式では条件オペレーターは使えません。

- 同じ CASE 内の複数のコンディションはカンマ（,）によって区別することが出来ます。
- CASE が True の時、ディフォルトの機能は CASE のステートメントを実行します。それからプログラムは ENDSELECT のすぐ後にあるステートメントにジャンプしてそれを実行します。

例：

```
*******************************************************
SELECT irCmd
  CASE 0 TO 3
    HIGH irCmd

  CASE AllOff, Mute
    OutA = %0000

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

【８】DO…LOOP

PBASIC 2.5 は、次のような一般的な形を取る標準化したルーピング構築を含みます：

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

条件のステートメントは特定の繰り返し数だけループを実行するように設定することが出来ます。条件が True になるまでか、又はその間、或はもし条件が適用されなければ無期限にループを繰り返します。最高 16 までの DO…LOOP をネスティド（Nested）する事が出来ます。

例：

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

上の例では、条件がループのコードが実行される前にテストされます。次の例では、ループコードが少なくとも 1 回は実行されます。なぜなら、条件のテストはコードの最後になされるからです。

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

【9】EXIT

EXIT は FOR…NEXT 又は、DO…LOOP を即座に終了させます。最高 16 までの EXIT をループ構造の中に設定することが出来ます。

例：

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

上の例でループは、温度（temp）が 100 を越えなければ、10 回繰り返します。temp が 100 を越えると FOR…NEXT のループは即座に終わります。

【10】ON…GOSUB

ON…GOSUB は他の BASIC と同一になるように加えられました。これは、PBASIC の BRANCH コマンドに似た命令で、それに続くラインのアドレスが少し違うだけです。そして、GOSUB がオフセットの値を元にして実行されます。もし、オフセット値がターゲットアドレスより大きい場合には、コードは次のラインを実行するために移ります。

一般的なシンタックスは：

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

ここに；

- オフセット（offset）は、0 から N のブランチに対するリストで、アドレスの指標（index）を特定する 0 から 255 までの変数／定数／式 などです。
- ON…GOSUB はオフセット 255 を越えるどんなリストも無視します。

例：

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

【11】ON…GOTO

ON…GOTO が他の BASIC と同一になるように加えられました。そして動作は PBASIC の BRANCH コマンドと同様です。

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

ここに；

- オフセット（offset）は、0 から N のブランチに対するリストで、アドレスの指標（index）を特定する 0 から 255 までの変数／定数／式 などです。
- ON…GOSUB はオフセット 255 を越えるどんなリストも無視します。

例：

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

上の例は次の BRANCH コマンドを同じものです：

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

【12】READ and WRITE インストラクションの増強

READ と WRITE のバイト（Bytes）又は、ワード（Words）でそれぞれ能率アップがなされました。そして、コードの 1 行の中で READ 又は WRITE の複数の変数が認められるようになりました。シンタックスは次のようになります（大カッコ［　］はオプションです。）；

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

例：

```
READ 0, hours, minutes, seconds

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

オプションのワード（Word）パラメーターが使われる時、値はローバイト（low byte）を先に読み取る又は、書き込んでそれからハイバイト（high byte）が続きます。

【13】PUT and GET インストラクションの増強

PUT と GET のバイト（Bytes）又は、ワード（Words）でそれぞれ能率アップがなされました。そして、コードの 1 行の中で PUT 又は GET の複数の変数が認められるようになりました。シンタックスは次のようになります（大カッコ｛　｝はオプションです。）;

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

例：

```
PUT 0, hours, minutes, seconds

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

オプションのワード（Word）パラメーターが使われる時、値はローバイト（low byte）を先に読み取る又は、書き込んでそれからハイバイト（high byte）が続きます。

【14】プログラムの中のラベル名にはコロン（ : ）が必要

PBASIC 2.5 は、GOTO や BRANCH などのためのラベル名にコロン（ : ）を付けなければいけません。ラベル名にコロンが付いていないと、コンパイルタイムエラーを発生します。

例：

```
Get_Hours:                                  ' good label（正しいラベル名）
    DEBUG Home, "Enter hours: ", CLREOL
    DEBUGIN DEC hrs
  IF (hrs > 23) THEN Get_Hours

Get_Mins                                    ' illegal label（コロンがないのでエラーになります）
    DEBUG Home, "Enter minutes: ", CLREOL
    DEBUGIN DEC mins
  IF (mins > 59) THEN Get_Mins
********************************************************************************
```

【15】条件付編集ディレクティブ

システムに接続した BASIC スタンプを走らせるプログラムの制作を容易にするために、いくつかの条件付編集ディレクティブが準備されました。

```
#DEFINE symbol = value
```

例：

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

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

例：

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

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

例：

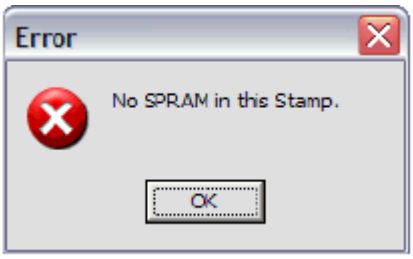
```
#SELECT $STAMP
     #CASE BS2, BS2e, BS2pe
          MidiBaud    CON    $8000 + 12

     #CASE BS2sx, BS2p
          MidiBaud    CON    $8000 + 60
#ENDSELECT

#ERROR    message
****************************************
```

この上の例のディレクティブはエディタ（editor）で表示されるエラーメッセージをプログラマーが作ることが出来ます。プログラムは#ERROR のラインをハイライトして止まります。

例：

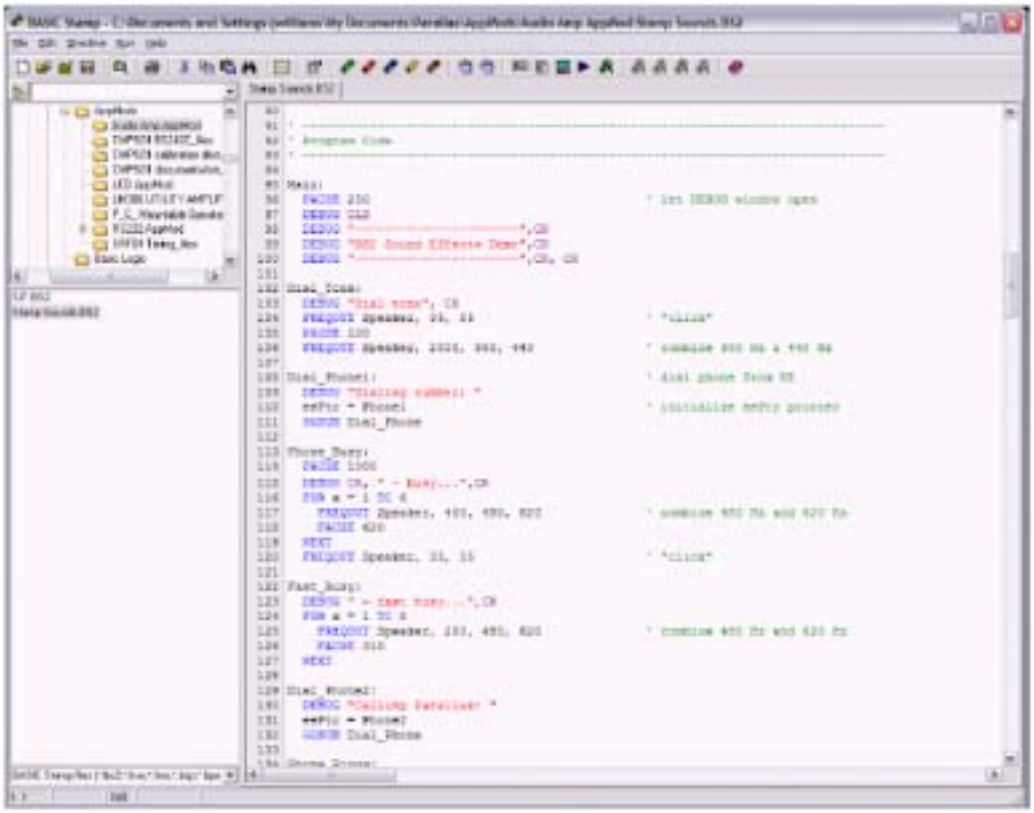
```
#IF ($STAMP = BS2) #THEN
     #ERROR "No SPRAM in this Stamp."
#ENDIF
****************************************
```

上の例では、もし BS2 の BASIC スタンプが接続されていれば次のように表示されます：



【16】インターフェイスの向上及び色付きシンタックスハイライト

BASIC スタンプのエディタ／開発システムは、ファイルの検索とファイルを開く事が簡単に出来るようにエクスプローラー形式のインターフェイスになりました。複数ファイルをファイルペイン（ファイルのリストがある所）から選択出来ます。そして、エディタペイン（ファイルを書く所）にドラッグアンドドロップが出来ます。この動作は選択したひとつ又は複数のファイルを開きます。



このリリースは色付きシンタックスハイライトを含んでいます。ディフォルトハイライトで殆どのプログラマーは間に合うでしょうが、変更することも出来ます。ハイライトの一覧を変更するには、Edit\Preferences...から Preferences ダイアログを開いて、“Editor Appearance”タブを選び、PBASIC scheme を選びます。“Copy Scheme”ボタンをクリックして、コピーを approve（承認）します。そしてそれから、希望するようにシンタックスハイライトを変更します。変更の状態をすぐ見たい場合には、“Show Preview Example”のチェックボックスを選んでください。