Tutorial: Robotics with Boe-Bot and PBASIC

This tutorial introduces you to the ***PBASIC*** programming language and the ***Boe-Bot*** development board and robotics kit. PBASIC is used to program the microcontroller that comes with the Boe-Bot robotics kit, which is sold by ***Parallax***. This microcontroller is known as the ***BASIC Stamp***.

Please note that the objective of this tutorial is to help you get started with PBASIC programming and the Boe-Bot robotics kit. It is not intended to be a complete reference.

Where to write your programs? The BASIC Stamp Editor

You need to write your PBASIC program using the ***BASIC Stamp Editor*** software. This software should already be installed on the computers in the classroom. If you want to install the software on your home PC or laptop, you can download the latest version from the Parallax website. The link for the download and install web-page for BASIC Stamp Editor is http://www.parallax.com/basicstampsoftware

BoeBot Development Board Details

The Boe-Bot development board that you will be using in this class is the "Parallax Board of Education - Rev D (USB)" and looks as shown below:

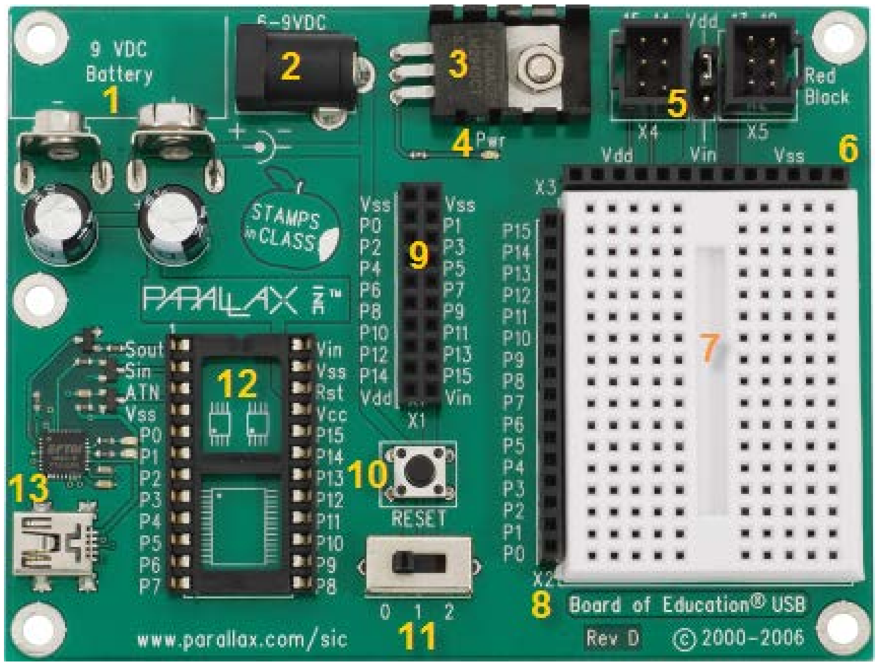


Fig. 1. Parallax Boe-Bot Development Board and its parts

P.S. Nair

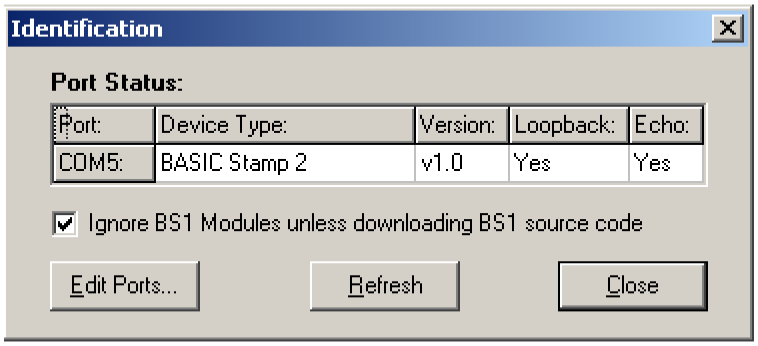
In the **Basic Stamp Editor**, click on ‘**Help→Getting Started→Identify Your Board**’. Locate ‘**Board of Education – USB**’ and click on the arrow that is labeled as ‘**NEXT**’. You will be taken to a page where you will be able to see the description of the various parts of the development board shown in Fig. 1. At the top of this page, you should be able to locate several icons. Move the cursor over the icons and when you find the icon that says ‘Expand all’, click on that icon. Read the information provided and bookmark the page for later reference.

Connecting the Development Board to your Computer (PC)

1. Connect the battery holder (it holds 4 AA batteries) to the part number **2** on the board (see Fig. 1) using the cable attached to the battery holder.
2. Slide the power switch (part number **11** on the board) to the position 1(middle position).This will power up the board.
   When you want to run the servo motors, the switch should be in position 2
3. After you have connected the battery pack to the board, take the USB cable and connect the broad end of the cable to a USB drive on your computer. Connect the narrow end to part number **13** on the board (see Fig. 1).

   ***Important (Please note)*:** When you are done using the board, please disconnect the USB cable first before removing the power connection to the battery pack.

4. Next, you need to test the connection between the board and the computer. In order to do this, click on the functional key **F6** on your keyboard. Alternatively, you can click on ‘**Run→Identify**’ in the Basic Stamp Editor window. If the connection has been properly done, you will see the that the connection has been detected on one of the COM ports of the computer, similar to what is shown in Fig. 2 below (in this figure, it is indicated that COM5 is where the connection was detected and the device identified was BASIC Stamp 2).



Port Status:

Port:	Device Type:	Version:	Loopback:	Echo:
COM5:	BASIC Stamp 2	v1.0	Yes	Yes

[✓] Ignore BS1 Modules unless downloading BS1 source code

Fig. 2 Example result of a correct connection check

Writing your first PBASIC Program

1. In the Debug Terminal (main window of Basic Stamp Editor), type the following program:

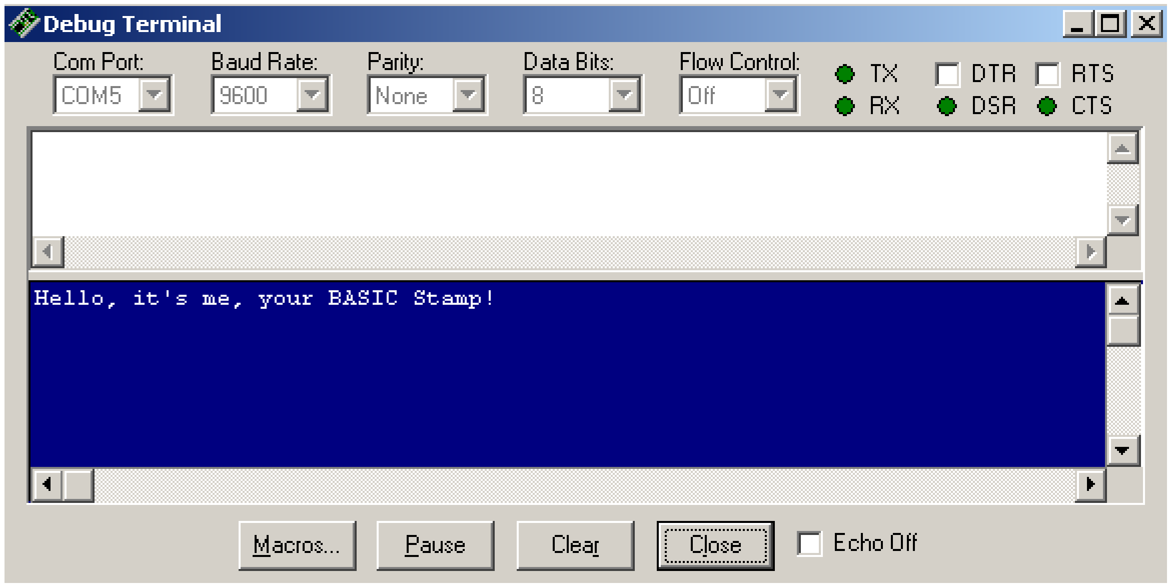
```
' First PBASIC program
' A comment is followed by the single quote symbol '
' This is a comment line and so are the next three lines shown below
' Stamps in Class - FirstProgram.BS2
' BASIC Stamp sends message to the Debug Terminal
' The next line is a Directive.

' {$STAMP BS2}
' The above directive tells the Basic Stamp Editor software that
' the part being programmed ON the board is BASIC Stamp 2
' The following directive tells that the version 2.5 of the Basic Stamp Editor software
'  is being ‘used

' {$PBASIC 2.5}

DEBUG "Hello, it's me, your BASIC Stamp!"

' The DEBUG command shown above is used to send a message to the PC
' The END command is used to end the program
END
```

2. After you have typed the above program, save the program by clicking on ‘**File → Save**’. Then name the file as **FirstProgram.bs2**. Then click on the **Save** button.

P.S. Nair

Running your first PBASIC Program

1. Now, run the program by clicking on '**Run→Run**'. Alternatively, you can click on the functional key **F9**. This will run the program on the Basic Stamp 2 that is on the development board and then display the result on the screen of your computer as shown below:



```
Hello, it's me, your BASIC Stamp!
```

Fig. 3. Output of the FirstProgram.bs2 program.

More PBASIC Programs:

I. Now try the following program and run it after saving:

```
' {$STAMP BS2}

' {$PBASIC 2.5}

DEBUG "Hello, it's me, your BASIC Stamp!", CR
' The DEBUG...CR command shown above can be used to move the program control
' to the next line and, optionally,  to display a message

DEBUG "What is 7 x 11?  ", DEC 7 * 11
' The DEBUG...DEC command can be used to evaluate an expression and, optionally,
' to display a message
END
```

P.S. Nair

II. <u>Introducing delay in the program</u>

Try the following program and run it after saving:

```
' {$STAMP BS2}

' {$PBASIC 2.5}

PAUSE 5000
' The above command delays the execution of the next command by 5000 ms = 5s

DEBUG "Hello, it's me, your BASIC Stamp!", CR

' The DEBUG...CR command shown above can be used to move the program control
' to the next line and, optionally, to display a message

END
```

III. <u>The DO…LOOP</u>

Try the following program and run it after saving:

```
' {$STAMP BS2}

' {$PBASIC 2.5}

DO
   DEBUG "Hello", CR
   PAUSE 1000
LOOP

' The commands between DO and LOOP are executed repeatedly
' In this program, the word 'Hello' will be displayed on a new line repeatedly
' with a delay of 1second between successive displays.

END
```

P.S. Nair

IV. Use of Variables and Variable initialization

Try the following program and run it after saving:

```
' {$STAMP BS2}

' {$PBASIC 2.5}

value           VAR   Word
anothervalue    VAR   Word
thirdvalue      VAR   Word

' value, anothervalue and thirdvalue  are all variables (VAR means a variable)
' All these variables are Word-sized (Range: 0 to 65535)
' We can also specify negative numbers by using DEBUG…SDEC
‘ Range when DEBUG… SDEC is used: −32768 to + 32767

' These variables are assigned initial values as shown below (Initialization)
 value = 500
anothervalue = 65535

' The following lines display the current values of  ‘value’ and ‘anothervalue’
DEBUG ? value, CR
DEBUG ? anothervalue, CR

' The following line modifies the variable ‘value’
value = value * 10

' The following line displays the new value taken by the variable ‘value’

DEBUG ? value

thirdvalue = value – 9000

DEBUG “thirdvalue = ”
DEBUG SDEC thirdvalue

' DEBUG SDEC causes negative value to be displayed

END
```

P.S. Nair

V. The FOR…NEXT LOOP

The FOR…NEXT loop is a very convenient and powerful way to control the number of times a given portion of your program is executed. It has the following syntax:

FOR *Counter* = *StartValue* **TO** *EndValue* {**STEP** StepValue}
…
…
NEXT

(The words written in bold letters above are key words; The contents of the curly brackets specify optional arguments. You can have other lines or program between **FOR** and **NEXT**, as in the DO…LOOP).

a.Next, try the following program and run it after saving:

```
' {$STAMP BS2}

' {$PBASIC 2.5}

Val_count          VAR    Word
FOR Val_count = 0  TO 18 STEP 6
   DEBUG ? Val_count
   PAUSE 500
   ' Half second delay is created by the above line
NEXT

END
```

b.Now try running the following program

```
' {$STAMP BS2}

' {$PBASIC 2.5}

Val_count          VAR    Word
FOR Val_count = 0 TO 18
   ' There is no STEP in this program → STEP = 1

   DEBUG ? Val_count
   PAUSE 500
   ' Half second delay is created by the above line
NEXT

END
```

An Important PBASIC Command:

The PULSOUT Command

- Physical movement of the robot is achieved in the Boe-Bot robotics kit through the use of servo motors.
- These servo motors are controlled by sending pulses. In the case of Basic Stamp 2, a pulse means a voltage of 5V lasting for a short duration
- Varying the duration or width of this 5V voltage results in the pulse width being varied
- The width of the pulse controls the duration of rotation operation of the servo motors; the rotation of the servo movements could be used to cause movements
- Therefore, the servo motors require high (5V) and low (0V) voltages that last for precise durations. Such precise-duration signals can be provided to the board by using the **PULSOUT** command
- Syntax: PULSOUT *PinNumber*, *Argument*
- This command sends out a precise high pulse on the Basic Stamp pin specified by Pin in the *PinNumber*. The duration (width) of the pulse depends on the argument
- Note: The *Argument* does not specify the duration directly. The <u>actual duration</u> and the <u>*Argument*</u> are related to each other by the following formulae:

Actual Duration = *Argument* * 2 µs,

or

***Argument* = Actual Duration in seconds * 500,000**

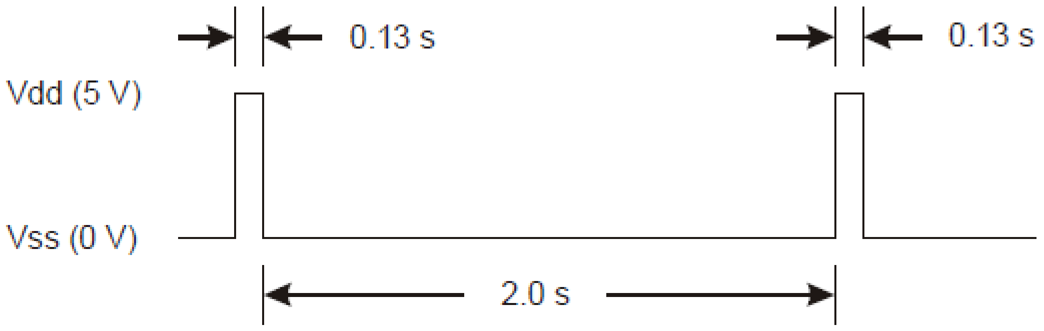


Fig. 4. Two pulses, each having a width of 0.13 s.

Fig. 4 Source: *Robotics with Boe-Bot Student Guide Version 3.0 by Andy Lindsay, available at http://www.parallax.com/Portals/0/Downloads/docs/books/edu/Roboticsv3_0.pdf*

Examples of PULSOUT

1. *PULSOUT 13, 80000*
   Here a high pulse is sent to pin number 13 of the Basic Stamp 2. The actual pulse duration is 80000 * 2 μs = (80000 * 2) / 1,000,000 s = 0.16 seconds

2. A program using **PULSOUT** and **FOR…NEXT**

```
' {$STAMP BS2}

' {$PBASIC 2.5}
L_counter        VAR    Word
FOR L_counter = 1 TO 100

   PULSOUT 13, 850
   PAUSE 20
' PULSOUT command lasts for 850 * 2 µs  = 1.7ms
' PAUSE command causes 20 ms delay
' Time to execute the loop commands themselves (PULSOUT and PAUSE) = 1.3 ms

' Therefore, time for one loop iteration = 1.7 + 20 + 1.3 = 23 ms
' Therefore, total servo run time (due to 100 iterations) = 23 ms * 100 = 2.3 seconds

NEXT

END
```

P.S. Nair