BASIC Stamp on GNU/Linux

by Shivendra Jairam

If you're reading this there is no reason for me to introduce what the BASIC Stamp is. But what you may ask is "What is need for writing this?" Well Parallax does not have a powerful editor for Linux like they have Windows or Mac. In fact they don't even have an editor. What they do have is the Tokenizer Library, which according to their website allows you to "make your own BASIC Stamp® Development Environment". In this article I will be paying most attention to the latter part of the previous sentence. That is, I will attempt to point out a few development environments written independently and not supported by Parallax. In other words – I'll help you get your BASIC Stamp up and running. These development environments that I'll be discussing will work for BASIC Stamp 1 and all BASIC STAMP 2 models. Please note that I've tested these environments using only the BS2 under Fedora Core 3 using kernel 2.6. The developers have made the claim for support for all BASIC Stamp models.

I will walk you through compiling (if necessary), installation, tweaks, and testing. There are four development environments considered:

- **BASIC Stamp Tools for Linux (Command line)**
- **BASIC Stamp Linux Solution (Command line)**
- **PBASIC Editor for Linux (GUI)**
- **Official Windows Editor install using Crossover Office or Wine**

Before I begin, you must have a machine in which you have root access. These programs write directly to /dev/* and normal users cannot do this. If you attempt to send data to the Stamp without proper privileges, you will get errors. Note you don't need to be root to compile/tokenize. You will also need the gcc compiler and development libraries which you should have already installed on your distribution. I also assume that you have knowledge of pbasic.

It is also important to know which device in the /dev directory your Stamp is connected to. If you have one serial port, most likely you it will be connected to /dev/ttyS0. But it you have more than one serial port or using a USB to serial adapter then you can no longer safely say that it is on /dev/ttyS0. To figure this, you will need to make use of the dmesg command. This will tell you what hardware is associated with what device. More importantly it will tell you the device your Stamp is connected to. For example after executing dmesg, you may notice something similar to "device connected to /dev/ttyS0."

Throughout this article I will use /dev/ttyS0 as this is what it is actually connected to on my computer.

Please keep note which device in the /dev directory is associated with your BASIC Stamp. You will need this later on when configuring the various environments.

BASIC Stamp Tools for Linux

This can be downloaded from http://sourceforge.net/project/showfiles.php?group_id=78424 As of this writing, the newest release was 2006.05.31. This download also includes the Tokenizer library, so there is no need for a separate download.

Configuring and Installation

Once you have downloaded the file. You will need to extract it.

```
code:

tar -zxvf bstamp-2006.05.31.tar.gz
```

Now you must change to the directory where you extracted it to. By default the directory name is bstamp.

```
code:

cd bstamp
```

Before we begin to compile, it worth the mention that this development environment writes data to /dev/bstamp. This is done for an obvious reason – to allow universal compatibility because not everyone will have their Stamp on /dev/ttyS0 for example. So what they suggest you do is make a symbolic link from whatever device your stamp is connected to /dev/bstamp. And this makes a lot of sense, but it may be inconvenient. What I suggest you do is modify the code to write to /dev/ttyS0 (or whatever device the stamp is connected to) instead of a symbolic link in /dev. This may sound difficult but it is quite easy as you shall see.

A file called “bstamp_run.cpp' which is located in the bstamp folder will need to be edited. It is only one simple edit and there is nothing much to it. Open the file:

```
code:

emacs bstamp_run.cpp
```

You can use whatever text editor you wish to, but I prefer emacs. Once you have the file open, use the text editor's search function and look for the string "/dev/bstamp". For this particular release (as viewed from emacs), it appears on line 56 and looks like this:

```
char *device="/dev/bstamp";
```

You will have to change the device within quotes. For example if the stamp were connected to /dev/ttyS0, the line will now need to be changed:

```
char *device="/dev/ttyS0";
```

[If you do not wish to edit the code, you can always make a symbolic link to /dev/bstamp

code:

```
ln -s /dev/ttyS0 /dev/bstamp
```

Please note that here I'm using ttyS0 (edit as needed).

]

Save the changes and exit. Now you are finally ready to compile the code. From within the bstamp directory:

code:

```
make clean

make
```

Then as root you can install the complied code:

code:

```
make install
```

Depending on the distribution you have you may have an error when you compile with something similar to “cannot find ldconfig”. This can be solved my editing the Makefile within the bstamp directory. Look for the line ldconfig and replace it with the complete to the command with is /sbin/ldconfig. Now that you have necessary software installed, it is now time to copy the tokenizer library to where the installed program will expect to find it. You must again be root and in the bstamp directory.

code:

```
cp tokenizer.so /usr/local/lib/libbstamptokenizer.so

echo "/usr/local/lib" >> /etc/ld.so.confldconfig
```

Now we have properly configured the environment. We can now test it. The library can be loaded as by:

code:

```
bstamp_tokenize

PBASIC Tokenizer Library version 1.23
```

Usage

Before we test a program, there are a few things that I need to mention. Firstly there are two commands that you need to become familiar with. They are the bstamp_tokenize and bstamp_run. The first command takes the pbasic code and prepares it to be sent to the Stamp or it makes binary tokens that the CPU of the Stamp can interpret. The second command then sends these tokens to the Stamp. You will also need a text editor to write the code, I prefer emacs. Please note that all files containing code should have a .bs2 extension for easy recognition. Lastly, when sending data to the serial port, you must be the root user or have permission to the port. One way of getting temporary read/write permissions as a regular user is to use the “chmod 777 /dev/*” (where * is the appropriate device) command as the root user. This will ease the pain of su'ing back and forth between a regular user and root. This is however not secure on a multi-user system.

Lets consider and example program:

```
'{$STAMP BS2}

'{$PBASIC 2.5}

DEBUG “Hello world, How are you?”
```

Once the file is saved. It can be compiled/tokenized by the following command line syntax:

code:

```
bstamp_tokenize filename.bs2 filename.tok
```

This will create a file with the binary tokens with a .tok extension. It is this file that must be sent to the Stamp. The syntax is as follows:

code:

```
bstamp_run filename.tok
```

The above two step of compiling and sending can be combined into one simple command:

code:

```
cat filename.bs2 | bstamp_tokenize | bstamp_run
```

The above works as follows. The output of filename.bs2 is piped to bstamp_tokenize where it gets complied, then the output of this is piped to bstamp_run and is then sent to the Stamp.

In either method of command usage, you should see something similar to this after you have sent the data to the Stamp:

```
PBASIC Tokenizer Library version 1.23

Basic Stamp detected!Model: Basic Stamp 2

Firmware version in BCD = 16

18 characters transmitted

18 characters echoed

18 characters transmitted

18 characters echoed

18 characters transmitted

18 characters echoed

18 characters transmitted

18 characters echoed

DEBUG OUTPUT: (Press [Control]-[C] to complete sequence)

_____

Hello world, How are you?
```

You have now successfully used your BASIC Stamp under GNU/Linux with bstamp tools.

You should always place the pbasic compiler directive when writing your code. If you are using pbasic 2.x, its best you set the complier directive as 2.5. You may get some weird compile errors if you omit the directive or set it as 2.0.

There is only one downside to using bstamp tools for linux. The latest release (2006.05.31), has no (known) support for the debugin command. Emails sent to the developer yielded no response. If you figure out how to use the debugin command under bstamp tools, let me know and I'll update this article.

<incomplete>

To Add:

Basic stamp linux solution (stampbc, does not display debug info)

Basic stamp editor for linux (not all that great even though it has a GUI)

Official windows editor on crossover office (it works perfectly!)

Conclusion (you're better off using bstamp tools for linux)