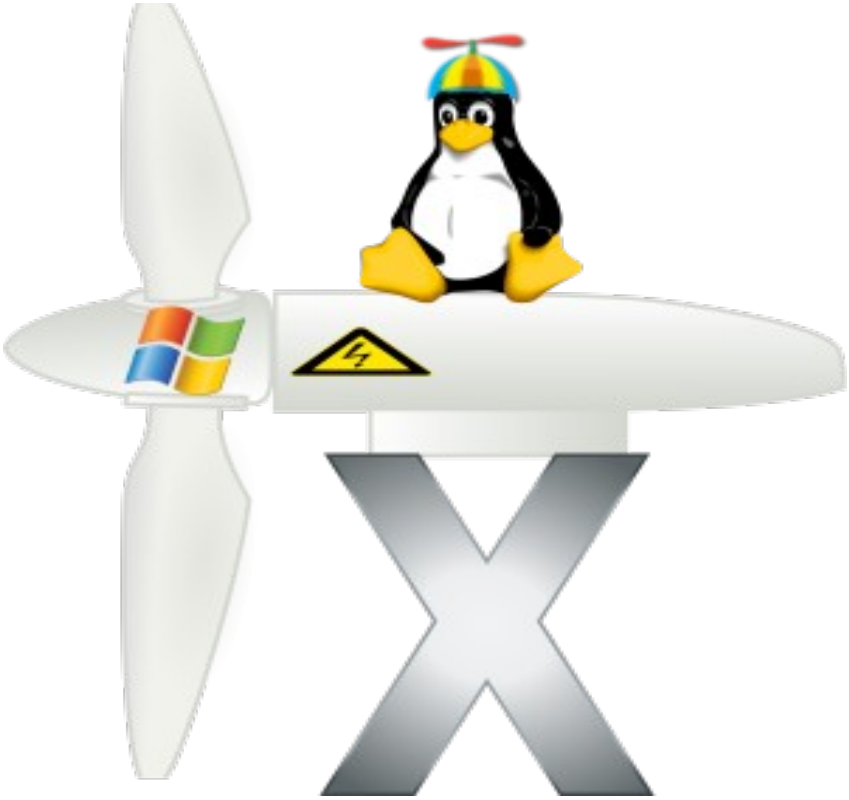
bst tools



User Manual

v 0.04

30 Apr 2010

Table of Contents

1 Introduction

The bst tool set is a multi-platform set of tools for developing with the Parallax Propeller microcontroller.

bst stands for “Brad's Spin Tool”, however it is never capitalised.

The bst tool set currently targets and supports the following architectures and operating systems :

- i386-linux-gtk2
- PowerPC-darwin (Mac OSX 10.4->10.6)
- i386-darwin (Mac OSX 10.4->10.6)
- i386-Win32 (Windows 95->Windows 7)

The current tool kit consists of three parts :

1. bstl – The command line loader / unloader
2. bstc – The command line loader / compiler
3. bst – The fully integrated GUI IDE

This manual is intended for users with a firm grasp of Propeller Chip concepts.

New users are recommended to read the manual that comes with the Parallax Propeller Tool in order to gain familiarity with the basic concepts prior to attempting use of bst.

The latest version of the bst tools can always be found linked from http://www.fnarfbargle.com/bst.html

There are often development snapshots posted to http://www.fnarfbargle.com/bst/snapshots

The code in the snapshot directory is very much a work in progress and often represents experimental features as the development progresses. It has the potential to be unstable or break badly. If it breaks, you do however get to keep both pieces.

2 bstl – Command Line Loader / Unloader

bstl was developed initially as a test for the cross-platform toolchain and to experiment with different algorithms for loading the Propeller chip. The program is dual purpose, which is to take a Propeller binary (*.binary) or eeprom (*.eeprom) file (as generated by the Propeller Tool) and load it into a Propeller chip via a serial port, or suck the program out of a Propeller chip and save it back as an eeprom file.

Asking bstl for its command line options will give you the following :

```
Program Usage :- bstl (Options) <FileName>
-d <filename> - Serial device to use (Default /dev/ttyUSB1)
-p [123] - Program Mode
    1 - Ram only (Default)
    2 - Eeprom and shutdown
    3 - Eeprom and Run
-f Load at high speed
-t Test mode (Undocumented)
-l Test mode (Undocumented)
-h  - Show this help
-u Read Propellers EEPROM into file <FileName> - WARNING will overwrite it if it exists with no prompt!
```

On Linux & OSX, it will attempt to autodetect the first USB->Serial port (as encountered when using a Parallax PropPlug, or generic PL2303 converter). On Win32 it will simply try COM2. If it does not find your Propeller, you must tell it where it is using the -d command line option.

The -f option runs the download at twice the baud rate used by the Parallax tools. This has the ability to shave 30% of the download time off when loading a large file. If you suffer reliability issues using -f, simply leave it out to use the standard download speed.

Versions 0.07-Pre1 or later have a new feature to read the contents of an eeprom connected to the Propeller. If you've lost your source code, or simply want to upload the eeprom to preserve user settings (in which case the checksum will need re-calculating before you can re-download), you can use the -u option. This will connect to the Propeller and download a small program into RAM. This copies the contents of the first 32k of eeprom into RAM, performs a 16 bit CRC on it and then uploads the contents to the PC. The PC receives the contents, checks the resulting CRC and writes it out to a file as named on the command line. If using the -u option without a file name, the whole process still occurs however there is no file written to disk. This is useful for testing communications integrity.

3 bstc – Command Line Compiler

bstc was written as a simple command line compiler to compile Propeller *.spin files. It handles generic Propeller Tool compatible spin files and aims to generate 100% bit-for-bit identical binaries.

The command line for the compiler is significantly more complex than for the loader. The current help and usage information is listed below :

```
Brads Spin Tool Compiler v0.15.4-pre8 - Copyright 2008,2009,2010 All rights reserved
Compiled for i386 Linux at 14:43:25 on 2010/04/20
Program Usage :- bstc (Options) Filename[.spin]
 -a             - Create Propeller object archive zipfile
 -b             - Write .binary file
 -c             - Write .dat file for C-Compiler (Drops a <filename.dat> file)
 -d <device>    - Device to load to (Default : )
 -D <define>    - Define a pre-processor symbol (may be used multiple times)
 -e             - Write .eeprom file
 -f             - Double download baud rate
 -h             - Display this help information
 -l[sma]          - Generate listfile (s) For source code / (m) for Machine readable - Debugger style
listing / (a) standard boring listfile
 -L <Lib Path> - Add a library path or file holding library path(s) to the searchpath (may be used
multiple times)
 -o <filename> - Output [.list/.eeprom/.binary/.zip] Filename (Defaults to input Filename without .spin)
 -O <options>  - Optimise Binary (HIGHLY EXPERIMENTAL!!!!!)
    a          - Enable all optmisations (Be careful! No, really)
    b          - Bigger constants (should be slightly faster at the expense of code size)
    c          - Fold Constants
    g          - Generic "safe" size optimisations for smaller/faster code, however not what the Parallax
compiler will generate
    r          - Remove unused Spin Methods
    u          - Fold Unary "-" Operations on Constants if it will make the code smaller
    x          - Non-Parallax compatible extensions
 -p[012]        - Program Chip on device (-d)
    0          - Load Ram and run
    1          - Load EEProm and shutdown
    2          - Load EEProm and run
  -w[012]              - Error/Warning level - 0 - Errors only / 1 - Error/Warning / 2 -
Error/Warning/Information (Default 0)
 -q             - Be silent except for GCC style errors and warnings
 -v             - Get program version information
```

3.1 Usage

bstc requires as a minimum, the filename of the source you wish to compile.

```
brad@bkmac:~$ bstc test.spin
Brads Spin Tool Compiler v0.15.4-pre8 - Copyright
2008,2009,2010 All rights reserved
Compiled for i386 Linux at 14:43:25 on 2010/04/20
Loading Object test
Program size is 3 longs
Compiled 1 Lines of Code in 0.001 Seconds
```

It will simply compile the file and tell you if you have any errors :

```
brad@bkmac:~$ bstc test.spin
Brads Spin Tool Compiler v0.15.4-pre8 - Copyright
2008,2009,2010 All rights reserved
Compiled for i386 Linux at 14:43:25 on 2010/04/20
Loading Object test

test(2,3) Error : Unresolved Symbol - X
  X := 1
__^

Compiled 2 Lines of Code in 0.001 Seconds
```

To do anything else useful, you need to use some of the optional command line parameters.

3.1.1 (-p) Loading a Propeller

There are three ways to load a program into a Propeller target. These three options are specified using the command line (-p).

- -p0 – Load the program to Propeller RAM and run it
- -p1 – Load the program to an attached EEPROM and halt
- -p2 – Load the program to an attached EEPROM and run the program

The options are mutually exclusive and only one may be specified at a time.

NOTE : There may be issues currently with RS232-Propeller interfaces based on discrete Transistors. The design as published by Parallax has the DTR polarity inverted to that used by the FTDI (PropPlug) interface. This has been proved to cause issues with the ability of bstc to detect and load the Propeller. It is hoped this can be rectified prior to the release of bstc v0.16

3.1.2 (-b) Saving a Propeller binary to disk

The (-b) option allows you to save the compiled binary to disk in a Propeller Tool compatible binary format. The binary file is saved (by default) to the same directory as the original top source file, and named as the top source file.

```
brad@bkmac:~/proptest$ ls
test.spin
brad@bkmac:~/proptest$ bstc -b test.spin
Brads Spin Tool Compiler v0.15.4-pre8 - Copyright
2008,2009,2010 All rights reserved
Compiled for i386 Linux at 14:43:25 on 2010/04/20
Loading Object test
Program size is 3 longs
Compiled 2 Lines of Code in 0.001 Seconds
brad@bkmac:~/proptest$ ls
test.binary  test.spin
```

3.1.3 (-e) Saving a Propeller eeprom file to disk

The (-e) option allows you to save the compiled binary to disk in a Propeller Tool compatible eeprom format. The eeprom file is saved (by default) to the same directory as the original top source file, and named as the top source file.

The eeprom file format is simply an extended version of the binary format suitable to be loaded into an eeprom using an external programmer rather than the Propeller itself.

3.1.4 (-a) Creating a Propeller Source Archive

The (-a) option creates a zipped archive of your Propeller source files similar to the one created by the "Create Archive" function in the Parallax Propeller Tool. There are a few caveats to the bstc version however :

- The tool will not properly archive source files that are pointed to by symbolic links
- The source code must compile with no errors

If the source contains errors, bstc will refuse to create any output files in any case. If one of your source files is pointed to by a symbolic link (for example you have one object name linked to another and the spin file references the name of the link) then the archive will be created, however the linked file will be empty.

```
brad@bkmac:~/proptest$ bstc -a test.spin
Brads Spin Tool Compiler v0.15.4-pre8 - Copyright 2008,2009,2010 All rights reserved
Compiled for i386 Linux at 14:43:25 on 2010/04/20
Loading Object test
Program size is 3 longs
Compiled 2 Lines of Code in 0.001 Seconds
brad@bkmac:~/proptest$ ls
test.binary  test-bstc-archive-100420-171319.zip  test.spin
brad@bkmac:~/proptest$ unzip -t test-bstc-archive-100420-171319.zip
Archive:  test-bstc-archive-100420-171319.zip
    testing: test.spin                OK
    testing: _readme_.txt             OK
No errors detected in compressed data of test-bstc-archive-100420-171319.zip.
```

3.1.5 (-c) Creating a DAT file for use by a C compiler (advanced)

Early in the development of C for the Propeller, it was common for drivers (such as TV or VGA) to be compiled using the Propeller Tool. The binary portion of the driver (the code intended to be loaded into a COG) was then manually stripped out and inserted as a binary blob to be linked into projects developed using a C compiler.

This option was added to allow people to compile an unmodified spin file, and allow bstc to output only the desired binary portion without any other processing required. I don't think it's ever been used, however it does work.

If for some reason you want a binary image of only the DAT section of your top object. This is the command for you!

3.1.6 (-d) Specify the serial port to use to load a Propeller

If you find the default configuration does not work for you, or you have multiple Propellers connected to your machine, this allows you to manually specify which port to use.

```
brad@bkmac:~/proptest$ bstc -d /dev/ttyUSB1 -p0 test.spin
Brads Spin Tool Compiler v0.15.4-pre8 - Copyright 2008,2009,2010 All rights reserved
Compiled for i386 Linux at 14:43:25 on 2010/04/20
Loading Object test
Program size is 3 longs
Compiled 2 Lines of Code in 0.001 Seconds
We found a Propeller Version 1
Propeller Load took 0.304 Seconds
```

3.1.7 (-D) Define a pre-processor symbol (advanced)

Calling the code in bstc a pre-processor is probably a stretch. It supports basic conditional compilation only at the current time, however this option allows you to specify symbols on the command line rather than inserting #define in the files to be compiled.

3.1.8 (-f) Download at double speed

The Propeller tool talks to the Propeller at 115,200 baud and uses a fixed bit packing scheme. bstc uses a “dense packing” scheme based on one devised by “hippy” on the Parallax Forums. This increases download efficiency (and therefore speed) by about 25% on average. The (-f) option also doubles the baud rate to 230,400 baud. This shaves approximately 30% more off the download time of large files.

My tests have shown it to be 100% reliable with all the hardware I have to test on, and so I use it all the time. If you experience issues with it, I'd like to know about it, but you can always omit it and go back to the slower rate.

3.1.9 (-l) Generate compiler list files

It is often useful to be able to look at the generated output of a compiler in some form of human readable fashion, if for no other reason than to go “Ah!, that's why it does that”. List files were (are) a notable absence from the Parallax tools and have been sorely missed. bstc was written from the ground up to generate clear, concise human readable versions of compiled SPIN and PASM code to enable the programmer to understand precisely what is going on under the bonnet (hood for those across the pond).

bstc understands three possible list file options. (-la) generates a straight compiler list file. (-ls) generates a list file with the original source lines inserted above each piece of generated code. (-lm) generates a list file with additional information to assist in being parsed by a debugger.

See Section 7 for a basic explanation of the contents of the list file

3.1.10 (-L) Library Path

This option specifies where the compiler might search for library source files. It may be specified as many times as you like to list multiple library directories. The compiler will always try the same directory as the source file first, then it will search the library paths in the order given for sub-object files.

3.1.11 (-o) Output Filename

By default, all output files (*.binary / *.eeprom / *.dat / *.list) are named with the same file name as the source file. The -o parameter allows you to specify an alternate name for the output files.

The following example compiles test.spin into a binary named “john.binary”

```
rad@bkmac:~/proptest$ bstc -o john -b -ls test.spin
Brads Spin Tool Compiler v0.15.4-pre10 - Copyright 2008,2009,2010 All rights reserved
Compiled for i386 Linux at 22:00:43 on 2010/04/21
Loading Object test
Loading Object Blink
Program size is 12 longs
Compiled 16 Lines of Code in 0.035 Seconds
brad@bkmac:~/proptest$ ls
blink.spin  john.binary  john.list  test.spin
```

The compiler automatically names the extension of the filename appropriately.

3.1.12 (-O) Optimisation options

bstc has the ability to perform some basic optimisation on the source being compiled. See Section 6 for details and function of the available options.

3.1.13 (-W) Error / Warning levels

This option controls the verbosity of the compilers output. There are three levels (0/1/2).

- -W0 – Only display errors in the compilation
- -W1 – Display errors and warnings generated (Jmp without # - for example)
- -W2 – Display errors, warnings and information

Warnings are generated for common errors that have generated code that might not do what you want it to do. Warnings can be disabled on a particular line (for example using a variable for a jmp in PASM) by adding an '!W as the start of a comment on that source line.

The information display can be a little overwhelming at times, as it analyses your spin methods and tries to inform you of unused components that are consuming extra space (such as unused global and local variables & unused SPIN methods). For this reason, it defaults to disabled and must be explicitly enabled with -W2 to display its output.

The default error level is 0.

3.1.14 (-q) Be vewy, vewy, qwiet. I'm hunting Wabbits!

The -q option is there for the hard-core Makefile users who want the compiler to do what it does, with the minimum of fuss or noise. It will only report errors, and attempt to report them in a fashion compatible with the output of the GNU compilers. It's not as beginner friendly, but it's far more concise.

3.1.15 (-v) Display version information

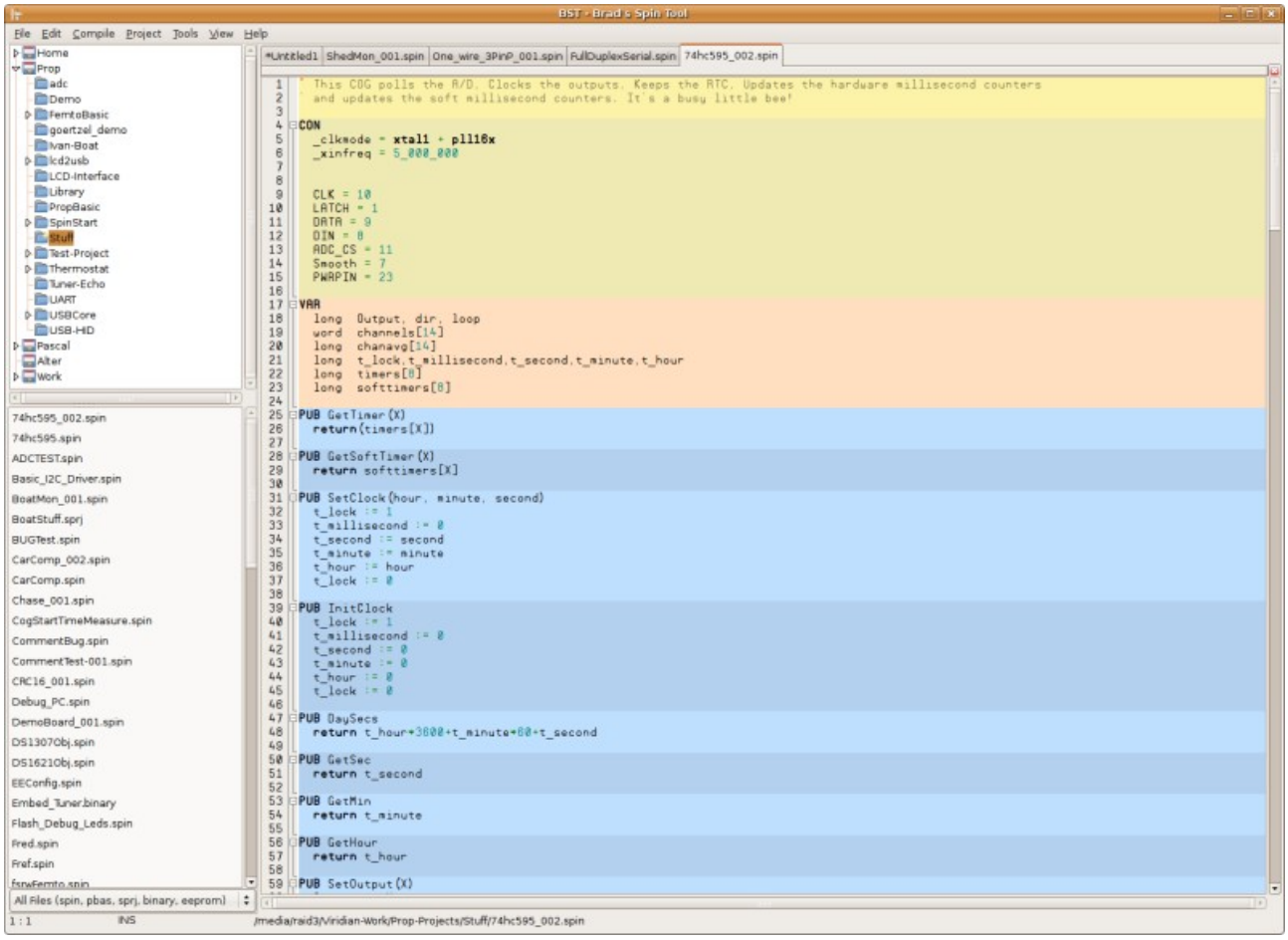
Does precisely what it says on the tin.

4 bst – The GUI IDE

bst is a full-featured GUI IDE that tries to include all the basic requirements for developing a Parallax Propeller application in the one single binary. It requires no installation, just unzip and go.

Due to the nature of the platforms bst runs on, there are some minor “behind the scenes” variations in the program with regard to system specific configuration items. In this manual, all configuration examples are for the Linux variant and valid for all platforms unless otherwise noted.

4.1 The main workspace



The main bst workspace consists of 5 main areas :

- The Directory Tree
- File Selector Box
- Tab Area
- Editor Space
- Status Bar

4.1.1 The Directory Tree

The directory tree provides a representation of each root path configured in the IDE.

Clicking on a directory in the tree causes the filtered contents to be displayed in the File Selector Box immediately below.

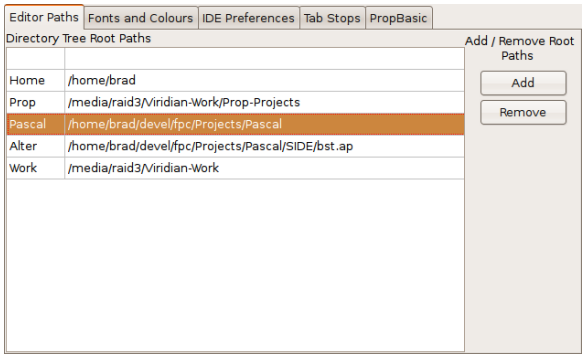
If directories have been added or removed outside of bst while it is open, a right-click on the Directory Tree box will allow you to refresh the contents.

4.1.1.1 Configuring the Directory Tree

Unlike the Propeller Tool, which displays every available storage device connected to the system and dynamically adds and removes them as filesystems come and go, bst requires you to manually configure the directories you want to be visible.

There are a number of reasons it has been implemented this way, but the foremost is speed. bst is developed in an environment with lots of network shares over unreliable links, and having the tools rescan entire directory trees at inopportune times causes significant delays. By configuring manually, you have precisely what you want, where you want and nothing else.

The directory configuration is accessed from the IDE Preferences Dialog.



It creates different initial defaults depending on the platform it is running on.

To add a directory, simply click [Add] and browse to the directory you wish to add. To rename the directory as displayed in the Tree, click on the name to select it for editing and change as desired.

To remove a directory, click it to highlight and simply click [Remove].

When you close the Preferences dialog, the directory tree will refresh with your updated preferences.

4.1.2 File Selector Box

The file selector displays the filtered contents of the directory selected in the Directory Tree. The filter can be adjusted using the box immediately at the base of the File Selector Box. Double clicking on any of the files in the list will open them using the correct part of bst.

If files have been added outside of bst, clicking again on the directory in the Directory Tree will cause the File Selector box to be refreshed.

4.1.3 The Tab Bar

Each open file in bst is given its own editor tab. The tab selector component allows you to switch, close, save, open and generally manipulate the files with the right-click context menu.

Warning : In versions prior to 0.19.4-Pre12, the right-click menu operates on the currently selected tab, ***NOT*** the tab you are right-clicking over! The title at the top of the context menu displays the name of the tab the menu is associated with.

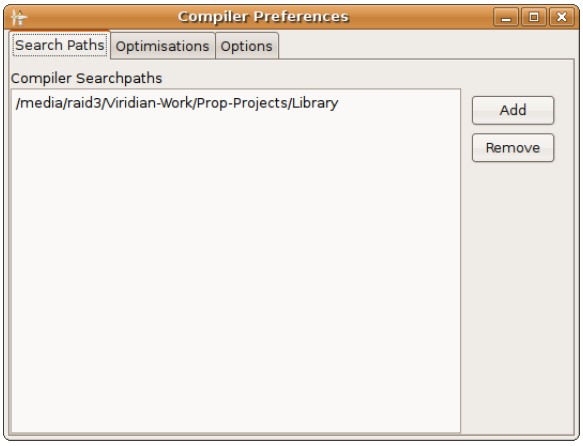
Version 0.19.4-Pre12 or later also allow you to re-order the tabs using the “Move tab left / right” menu options in the context menu.

The red close button on the far right of the tab bar closes the currently active tab (as does Ctrl-W).

4.2 Getting started with bst

bst requires some basic configuration prior to getting started with a project. The following items are the minimum requirement to ensure your bst experience is as pain-free as possible.

4.2.1 Compiler Search Paths



Unlike the Propeller Tool, which has a single library path, bst allows a list of paths to search for your Propeller Object files. It is important before you start to configure at least your basic library path to allow the bst compiler to find your library objects. These can be assigned in the Compiler Preferences dialog in the Tools menu.

The library paths are searched in logical order from top to bottom, so if you have three FullDuplex.spin files it will use the first one it finds.

4.2.2 Fonts

Each platform has a different way of managing fonts.

Windows users can install the Parallax Propeller Tool first. This will install the Propeller font for you and you should be good to go. On Windows Vista, there can sometimes be an issue with the Propeller Font not being available to all users, so you may have to locate it (Propeller.ttf) and install it in the Control Panel Fonts widget manually before bst can see it.

OSX and Linux users must download a modified version of the Propeller font. I can't stress this enough, the font supplied with the Parallax Propeller Tool is **BROKEN** on Linux and OSX. If you install it by mistake, it will cause all sorts of horrible things to occur that are hard to debug. Please install the correct font. The location of the correct font is kept up to date here : http://propeller.wikispaces.com/Propeller+Font.

OSX users can install the font simply by double clicking on it and clicking the “Install” button.

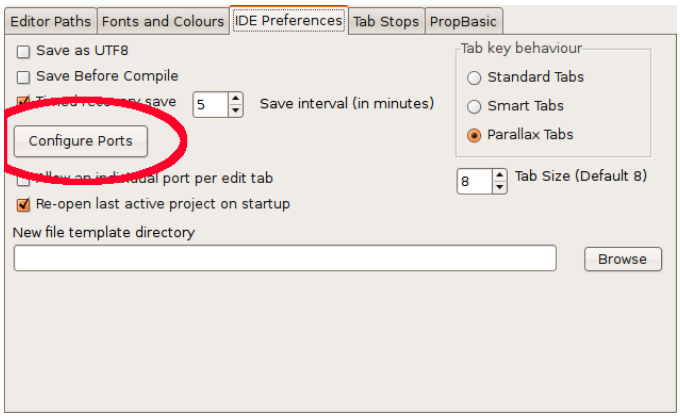
Linux users have to figure out where their particular distribution installs the fonts! On an Ubuntu system, the are located in ~/.fonts. The easy install is to copy the modified Propeller.ttf there and to log out and back in again.

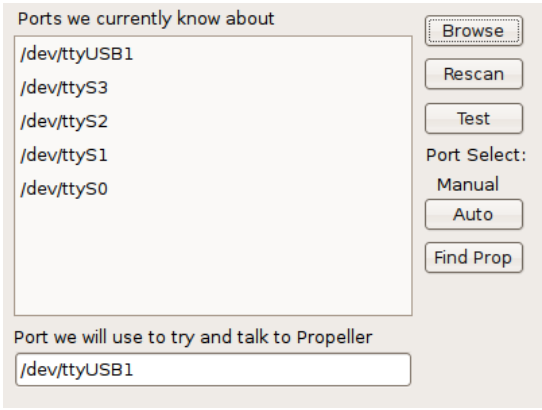
4.2.3 Serial port configuration

Windows and OSX users MUST install the ftdi serial drivers (if you are using a Parallax product like a demo board or PropPlug) before any of this is going to work. You need the FTDI VCP (Virtual COM Port) drivers. Windows users will likely have them installed already if you installed the Propeller Tool first as detailed above to get the font.

Linux users are generally good to go out of the box (you can check lsmod to see if ftdi-sio is there after you plug your device in, if you are unsure)

bst has a good go at auto-detecting relevant serial ports on the systems it runs on. In the event of it not picking up your serial port you can manually configure your port from the IDE Preferences dialog





The [Browse] button is irrelevant on Windows. On Linux & OSX it allows you to browse the filesystem looking for the port you are after (*nix – Everything is a file!).

[Rescan] will refresh the box looking for any ports that might have been added since you opened the search box (ever wondered where your Prop was and realised it was not plugged in?)

[Test] will attempt to connect to a Propeller on the selected port and query its version information.

If you have a port selected manually (there is a port name in the bottom box - /dev/ttyUSB1 above) the [Auto] button will clear that and put bst back into auto-detect mode.

[Find Prop] will try and test every detected serial port on the system sequentially until it detects a Propeller. It will then leave that port manually selected.

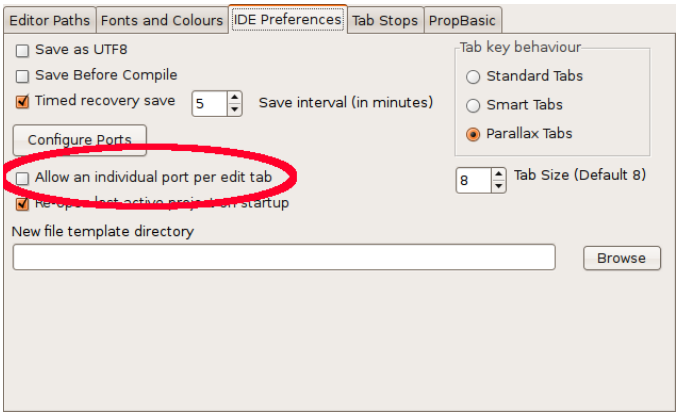
You can also type the port name or direct path into the box at the bottom to manually force bst to use a certain port.

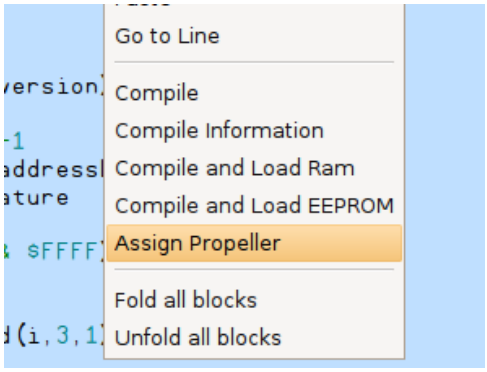
Generally, leaving things set to auto “just works”. If it should, and it doesn't we'd like to hear about it.

4.2.4 Multi-port configuration (One Propeller per editor tab)

If you are developing more complex projects and have the need to write and test code for more than one Propeller at a time, bst allows you to assign a Propeller to an editor tab individually. To compile and download you simply select the tab you want and press F10. bst will make sure the correct Propeller is selected and loaded.

To enable this feature, you need to make sure the option is checked in IDE Preferences (it can cause some confusion, so it's not enabled by default).



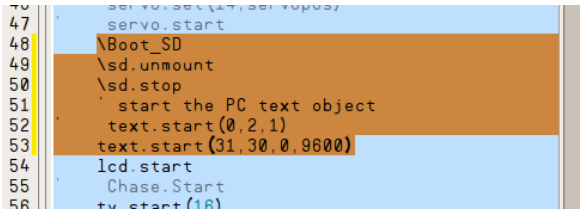


Then, from the right-click context menu inside the editor tab itself, simply select “Assign Propeller”. You are presented with the ports configuration dialog shown above and you can manually assign a port to a propeller.

On OSX and Windows, this is a bit neater as OSX names its ports with the serial number of the PropPlug, and Windows creates a new port for every individual serial converter it ever sees. Linux simply numbers them in the order they were plugged in (gurus can change that by modifying your udev rules however).

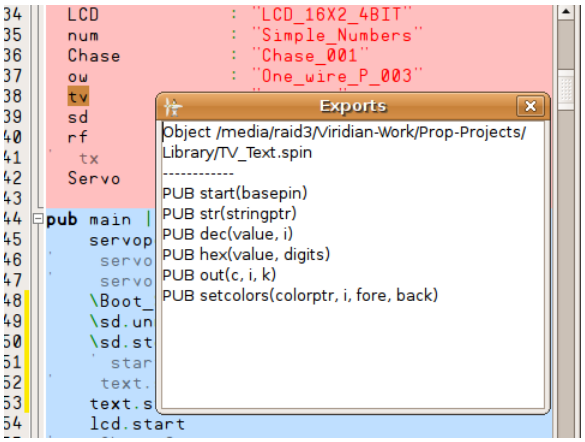
4.3 Using bst

4.3.1 (Ctrl-Shift-I/U) Block indenting



The Propeller tool allows you to indent a selected block of code using the Tab Key. In bst the code is indented using Ctrl-Shift-I and un-indented using Ctrl-Shift-U. Each press results in a change of 2 spaces forwards or back. (Cmd - [/] on OSX)

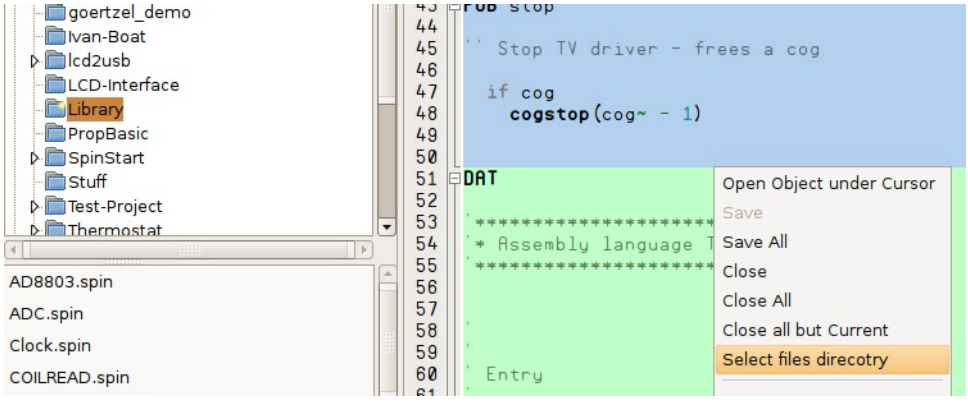
4.3.2 (Ctrl-Space) Sub object details



Once a file has been successfully compiled using any of the F8->F11 options (F9 is the quickest), its symbol table becomes available in the editor until the file is modified. bst has the ability to display the export table of sub object by pressing Ctrl-Space while the symbol is selected with the mouse. In the example shown, we clicked between the “t” and the “v” to make sure the cursor was in/on the word and pressed Ctrl-Space. The export table pops up to display the available functions in the object.

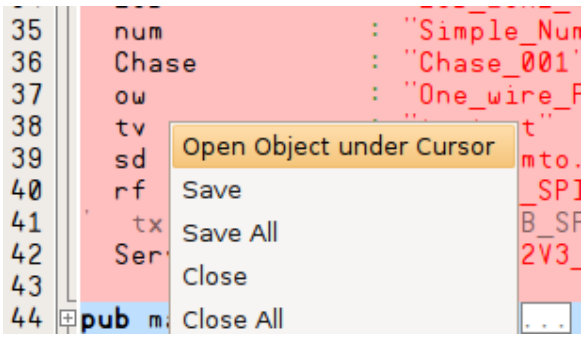
Warning : If you have “Remove unused SPIN Methods” enabled, the displayed symbols will only show methods actually compiled into the object. Currently unused methods are excluded.

4.3.3 (Context menu) Select Files Directory



Often when working on files in systems with a large quantity of library directories, it's handy to be able to browse other library files in the same directory. By selecting this item in a tab's context menu, the directory tree is immediately taken to the location of the source file, and the file selector box populated with its contents. It's also handy for checking the location of sub-objects that throw errors or warnings.

4.3.4 (Context menu) Open Object Under Cursor



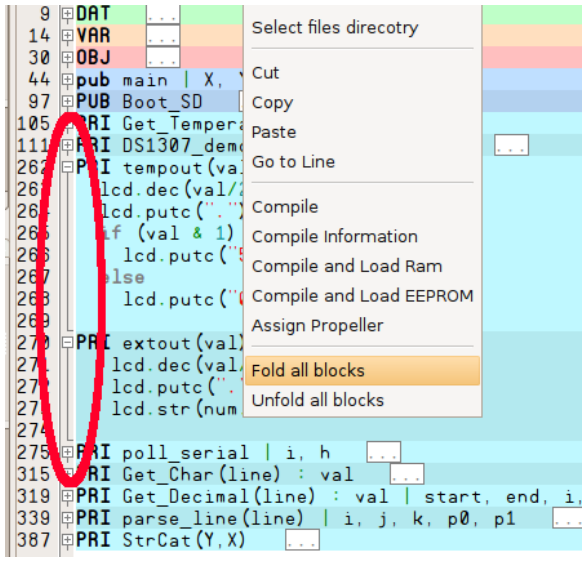
After a successful compile and prior to the source being modified, placing the cursor on the symbol of a sub-object will cause the “Open Object under Cursor” item to be enabled in the context menu. It does precisely what it says and simply opens the sub-object in a new tab, or selects the objects tab if it's already open.

4.3.5 (Ctrl-G) (Context menu) Go to line

In large source files it is convenient to be able to jump directly to a specific line. This menu opens a small dialog to allow you to enter the desired line to jump to.

4.3.6 Code Folding

Code folding is a neat addition to any source editor when working on large source files. It enables individual methods to be worked on without the clutter of nearby source code, and the convenience of the displayed adjacent code headers.



Code folding is tied to the display of line numbers (toggled with Ctrl-Shift-L) and is only enabled when numbering is visible.

Code can be folded and unfolded by clicking on the little [+] / [-] icons next to the line numbers. In addition, the entire file can be folded and unfolded using the right-click context menu and Ctrl-U will fold all blocks in a file.

The folded state of each file is stored inside the (optional) Project File for later recall.

4.3.7 (File->Create Propeller Archive) Creating a Propeller Archive.

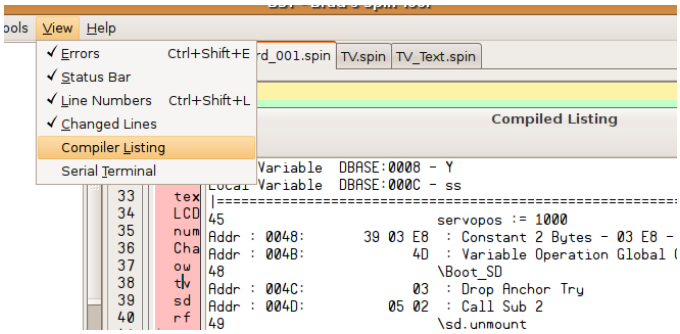
The Propeller Tool has the ability to create a zip file containing all the source files required to build the currently opened project (this includes library objects and data files). bst has the ability to do the same thing with one limitation. Your program must compile without error to allow an archive to be built. This is a limitation that stems from the archiver implementation inside the compiler (it uses the archiver built into bstc to do the dirty work). The work around if you need to send someone a broken archive is to comment out the offending code to allow the project to compile.

NOTE : bst is currently incapable of creating PropBASIC source file archives.

The archive name is titled the same as the top object file postfixed by the time and date of the archive, and it is saved automatically in the same directory as the top source file.

4.4 The List Window

Just as the list file output in bstc (-ls) presents a “compilers eye” view of the generated source, bst includes a separate List Window.



This can be opened and left open. Each time a file is compiled, the compiled listing is updated in this window. It enables a programmer to *really* see what is going on under the covers and help to understand why things do what they do. The list window is not docked or tied to the Main IDE window and can be moved around where convenient. It loses its source contents when closed, but can be closed and opened at any time.

See Section 7 for a basic explanation of the contents of the list file

4.5 The Project File

A Project is simply an extended set of preferences that defines :

- Which source files are open
- Where each source file is scrolled to
- What parts of each source file are folded
- (Optionally) Which Propeller Serial Port is assigned to each source file
- A project specific set of Pre-Processor Symbol definitions
- A project specific set of compiler options
- A project specific set of compiler search paths
- Which directory the Directory Tree is pointing to

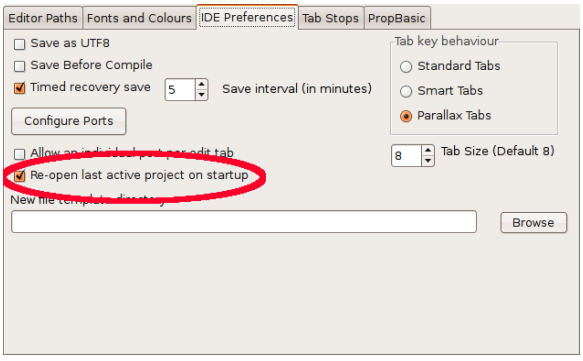
When a new project file is created, it simply takes all the settings from the current session.

Additionally, to facilitate complex or special configurations, you can define separate compiler search paths, and compiler optimisation options in the Project Options dialog. This allows you to override the global search path and options settings defined in the bst configuration. The search path is overridden if there are ***any*** search paths defined in the Project Options. The compiler optimisations require you to check the “Override global compiler optimisations” box before they take effect.

Opening a project file will see all tabs restored to precisely where they were when the project was last saved.

Warning : When you close the Project File, all files in the editor are closed simultaneously!

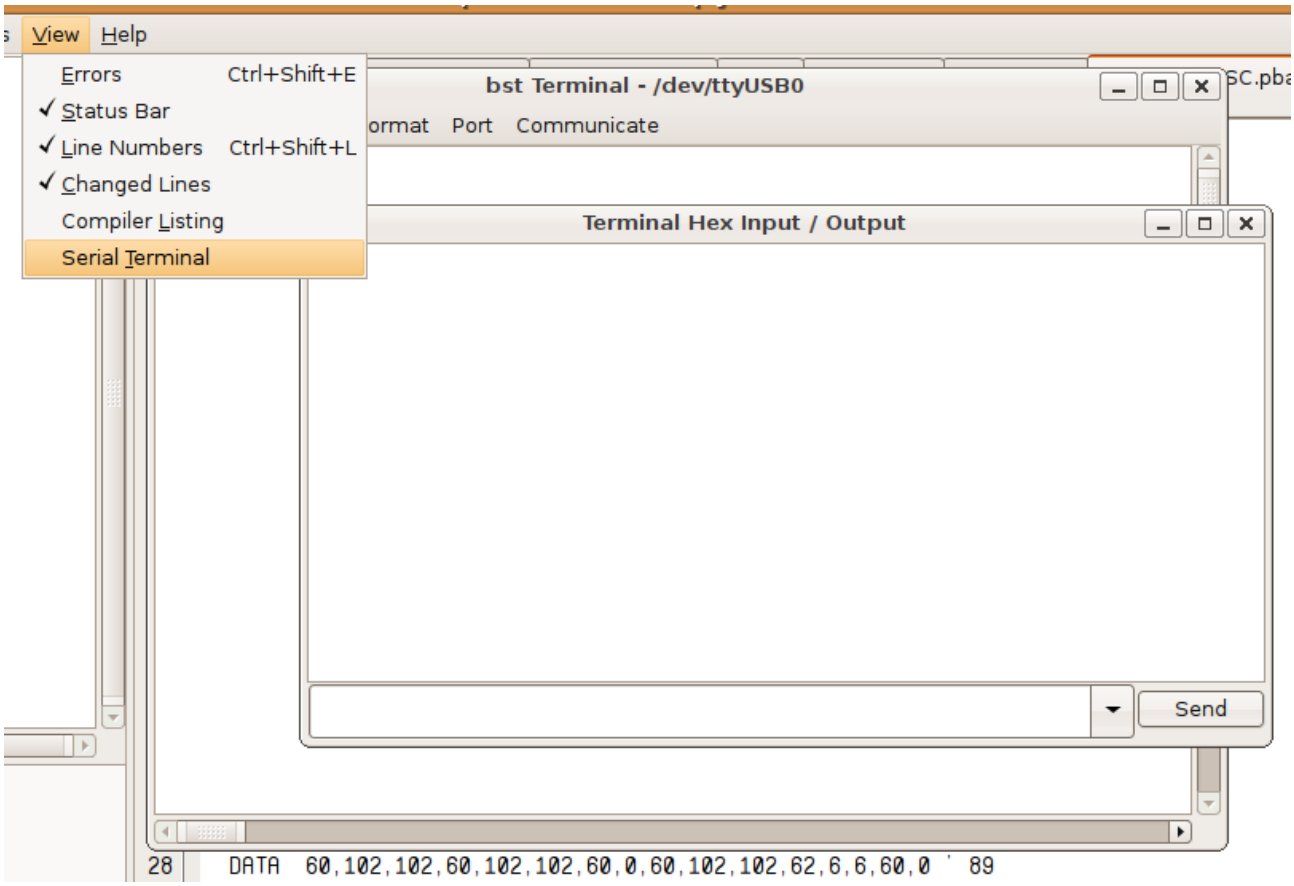
A useful option is to have bst re-open the project file that was open when it was closed last. This provides for a nice persistent workspace where you can always pick up right where you left off. This is configured in the IDE Preferences dialog.



NOTE : The open objects are currently stored in the project file with absolute path names, so moving project files across operating systems or different machines is unlikely to provide happy results. (This is scheduled for rectification prior to v0.20)

4.6 The Serial Terminal

The serial terminal built into bst is designed as a basic debugging tool. It allows communication with any devices connected to a serial port, and has the ability to send and receive in either ASCII or Hexadecimal notation. The serial terminal also has a separate Hex window which operates simultaneously with the ASCII window to allow easy monitoring of data coming into the system. The Hex window allows the composition of Hexadecimal strings in the bottom box and sends them over the wire as composed in RAW form.



The Serial terminal is fully integrated with bst, such that there is no need to manually disconnect / connect prior to downloading a propeller. bst will check to see if the terminal is using the port it needs to download to, and will manage the terminals connection in the background to ensure an uninterrupted download to the Propeller.

NOTE : bst has issues on OSX with the control over the DTR line (used to reset the Propeller). The terminal will stand one connection to the Propeller after it has been loaded. The next disconnect/reconnect cycle will see the Propeller reset. This has not proven to be much of an issue, but it's worth noting.

4.7 Under the bonnet (hood)

What follows is some basic background information on some of the less visible parts of bst. It may assist in debugging or understanding what is going on when something goes wrong.

4.7.1 Persistent Configuration Files

bst stores persistent configuration information on your system in a single location. This is to allow configuration details, specific preferences and niceties like recent files lists to persist across bst sessions.

On Linux the configuration resides in your home directory (~/.bst.ini)

On OSX the configuration resides in your home directory (~/Library/Preferences/bst.ini)

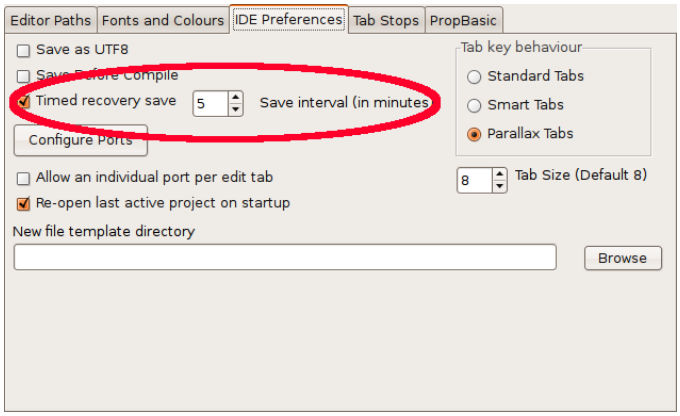
On Win32 the configuration resides in the System Registry (HKEY_CURRENT_USER\software\camsoft\bst\0.1)

In the case that something does go horribly wrong, it's sometimes quite helpful if you can send these files to the developer to assist in tracking down the problem.

If all else fails you can always delete these and start clean.

4.7.2 Recovery Files

bst is very much beta software and from time to time can crash with ugly error messages (although less frequently these days). To this end, bst has a “recovery save” mechanism whereby the contents of any tab that is not completely saved to disk is saved into a single recovery file stream prior to each attempt at compilation. Additionally there is an option in the IDE Preferences to allow a periodic recovery save to occur while other activity is taking place.



Like the configuration files, each platform has a different location for the recovery file :

On Linux (~/.bst.recover)

On OSX (~/Library/Preferences/bst.recover)

On Win32 it's wherever your particular version of Windows wants to place your user files. The file is called .bst.recover but you are on your own locating it!

To work around an issue with a particular operating system not properly saving the recovery file before the entire OS crashed trying to access the Propellers serial port, bst saves each recovery file with the -new postfix. It then removes the old recovery file and renames the new one. This ensures that the file is properly on disk or in the filesystems journal before the old one is removed and provides an additional safety net.

The contents of the recovery file is relatively simple, but is composed of binary streams and metadata. We've not yet had to manually parse one, so the recovery mechanism seems relatively stable.

5 SPIN Language Extensions

NOTE : bst uses the bstc compiler internally, so where bstc is mentioned, bst is implied.

Like Michael Park's “homespun” compiler, bstc has some extensions to the reference spin language. These are minor additions that are seen to enhance the usability of the language under certain circumstances, however they render the source files incompatible with the Parallax Spin Compiler. All of these extensions require at a minimum the (-Ox) command line parameter to enable the “Non Parallax Compatible Extension” option in the compiler

5.1 #define and friends

bst has a very basic conditional pre-processor built into the tokeniser. This allows the conditional compilation of code based on symbol definitions. It behaves similarly to most other conditional compilation directives in other languages but has a few important points worth noting.

- Defined symbols are global, and therefore passed down to any source file compiled after the file in which the symbol was defined
- Symbols are not allowed to contain spaces (the first encounter whitespace delimits the symbol)
- Any text after a symbol on a pre-processor line is ignored
    - #define I_am_a_symbol fred woz 'ere ← “fred woz 'ere” is completely ignored

bstc supports #define / #undef / #ifdef / #ifndef / #elseifdef / #elseifndef / #else / #endif as conditional statements

NOTE : All pre-processor statements **must** begin at the start of a line (Column 0)

bstc also supports #info / #warn / #error to allow comments to be inserted in the list file, or compilation to be halted with a user definable error message. Unlike the conditional statements, the #info / #warn / #error lines accept any message and will continue until the end of line.

5.2 @@@ - The absolute address operator

In SPIN methods, there are the @ & @@ operators. In PASM you have @. In all contexts they mean different things.

In SPIN, @ means “Give me the HUB address of that variable”. The interpreter does this at runtime as it only knows the absolute address when it know where it is in the Propeller. This applies to global (VAR), Local (PUB/PRI) and PASM (DAT) variables. The @ always returns the address of that variable in the HUB.

In SPIN, @@ means give me the compile-time offset (It's only used when referencing variables in a DAT block) plus the object base address (this results in the correct absolute HUB address). Again, its a runtime operator only.

In PASM, @ means “Give me the offset of this symbol into the DAT section”. It is a relative value with its base as the start of the objects DAT block.

The @@@ operator results in a compile-time constant that gives the absolute hub address of the symbol in question. It's a special use symbol and it's not widely used. If you know you need it, you know what it is.

Warning : The Propeller Tool will compile the @@@ operator as valid code in a SPIN method, however it will give a result far from what you want it to do!

5.3 Bytecode() (advanced)

Have you seen the generated SPIN bytecode in the list files and though, “Gee, I'd like to be able to do something funky but the compiler won't let me” ? If so, bytecode() is your friend. It simply inserts raw bytecode into the SPIN method. It's also useful for those developing compilers, debuggers, de-compilers or manipulation tools to be able to have complete control over the code being inserted.

The following code example writes 12 to the first local variable, then writes 157 to the return value (Local variable 0). It's perfectly valid spin and the Bytecode($38,$9D,$61) is the equivalent of “Result:=157”

Code :

```
PUB Fred | X
  X := 12
  Bytecode($38,$9D,$61)
```

List:

```
Spin Block Fred with 0 Parameters and 1 Extra Stack Longs. Method 1
PUB Fred | X

Local Parameter DBASE:0000 - Result
Local Variable  DBASE:0004 - X
|=============================================================================|
2                       X := 12
Addr : 0018:          38 0C  : Constant 1 Bytes - 0C - $0000000C 12
Addr : 001A:             65  : Variable Operation Local Offset - 1 Write
3                       Bytecode($38,$9D,$61)
Addr : 001B:          38 9D  : Constant 1 Bytes - 9D - $0000009D 157
Addr : 001D:             61  : Variable Operation Local Offset - 0 Write
Addr : 001E:             32  : Return
```

5.4 The VARX block (advanced)

SPIN has a quirk that from time to time seems to drive people nuts. For reasons of efficiency (we'd guess anyway) it re-orders your variables as declared in a VAR block in the order declared, but sorted into LONG / WORD / BYTE order (to best preserve alignment and waste no space).

bstc has a nasty hack whereby you can prevent this from occurring. Rather than declare a VAR block, you can declare the block using VARX. This will pack the variables in precisely the order you specify them in that object. It will generate errors if you break the alignment rules to prevent you from generating broken code, so you will have to add manual padding to the array to ensure the variables are properly aligned.

> Warning : There is an important caveat with the use of VARX. It affects the ***ENTIRE*** object. Once you declare a VARX block in an object (even if you have multiple VAR blocks), **All** blocks become un-packed.

This is not a modification we expect to see regular use, but it was more of an academic exercise to gauge the usefulness of the concept.

6 Compiler optimisation options

The Parallax compiler is a relatively straight forward compiler. It simply translates what you have in your source file into object code, and compiles the object code into a single binary file. bstc has a few extra tweaks that can potentially save some space in the resulting memory map and maybe make your code go imperceptibly faster.

6.1 (-Oa) Enable all optimisations

This option is often the only one I use. It simply enables **all** the options detailed below simultaneously.

6.2 (-Ob) Bigger Constants

The Parallax compiler has a trick to save space, whereby large constants can be written as smaller constants preceded by a "!" negative operator. So $FFFABCD would be represented as !$5432. This is a great way to save a couple of bytes here and there and makes for much more compact code, **however** it turns out it's actually *faster* to just load the large value (at the expense of code space). It's not much of an optimisation, but it's there "because we can".

6.3 (-Oc) Fold Constants

The Propeller Tool compiler provides a constant() operator, which allows the use of complex constant expressions to be folded down to it's resulting value, saving space in the object and ensuring a faster run speed. bstc takes this to its next logical progression and does it for you. Any expressions in your code that have compressible constant expressions will be folded down to the smallest possible size during the compilation process.

```
Con
  Seven = 7
PUB Fred | X
  X := 12 * 6 + 5 / Seven // 9
```

Without -Oc

```
4                         X := 12 * 6 + 5 / Seven // 9
Addr : 0018:           38 0C  : Constant 1 Bytes - 0C - $0000000C 12
Addr : 001A:           38 06  : Constant 1 Bytes - 06 - $00000006 6
Addr : 001C:              F4  : Math Op *
Addr : 001D:           38 05  : Constant 1 Bytes - 05 - $00000005 5
Addr : 001F:           37 22  : Constant Mask Y=34 Decrement 00000007 7
Addr : 0021:              F6  : Math Op /
Addr : 0022:           38 09  : Constant 1 Bytes - 09 - $00000009 9
Addr : 0024:              F7  : Math Op //
Addr : 0025:              EC  : Math Op +
Addr : 0026:              65  : Variable Operation Local Offset - 1 Write
```

With -Oc

```
4                         X := 12 * 6 + 5 / Seven // 9
Addr : 0018:           38 48  : Constant 1 Bytes - 48 - $00000048 72
Addr : 001A:              65  : Variable Operation Local Offset - 1 Write
```

“X := Constant(12 * 6 + 5 / Seven // 9)” would achieve the same result in the Parallax Compiler.

6.4 (-Og) Generic safe optimisations

Currently there is only one additional optimisation in this category. Again, for the purposes of much tighter code, the Parallax compiler has a clever bytecode pair for encoding even X^2 constants. This can pack large constants in a word of code quite neatly, however when encoding constants < $FF it's much slower than a simple constant load. In this instance bstc substitutes the operation with a straight constant load.

6.5 (-Or) Remove unused SPIN methods/objects

With this option enabled, bstc iteratively removes all unused SPIN object code from your program. For example, when writing code I often include the “simple_numbers” object, however I rarely use more than one method from that object in a program. In this instance, the remaining object code is simply consuming space that could otherwise be better used. In addition, each method or object consumes an additional 4 bytes (long) in the object method table. By removing methods and objects that are not referenced, significant space savings may be realised without having to manually strip down objects or customise library components.

6.6 (-Ou) Fold Unary

Occasionally in SPIN code constants are described as negative numbers. The compiler can optionally store these as the “-” operator on the raw number. If this case occurs, and there is the ability to make the resulting code more compact, bstc will discard the “-” operator and negate the constant in the compiler.

6.7 (-Ox) Non-Parallax compatible extensions

This option must be enabled to allow bstc to parse source files that would fail the Parallax Compiler. Without this option, every file parsed by bstc **should** also compile cleanly with the original Parallax tools.

7 Anatomy of a list file

Lets look at a basic spin program and its associated list file

test.spin

```
CON
  _clkmode      = xtal1 + pll16x
  _xinfreq      = 5_000_000

OBJ
  Blink : "Blink"

VAR
  long Fred
  byte Ada, Jean, May

PUB Start | X
  X := 1
  Blink.Go
  Ada := Jean
```

blink.spin

```
PUB Go | Y
  Y := 1
```

Compilation

```
brad@bkmac:~/proptest$ bstc -ls test.spin
Brads Spin Tool Compiler v0.15.4-pre8 - Copyright 2008,2009,2010 All rights reserved
Compiled for i386 Linux at 14:43:25 on 2010/04/20
Loading Object test
Loading Object Blink
Program size is 9 longs
Compiled 11 Lines of Code in 0.005 Seconds
```

Now, we have a new file (test.list) in the same directory as the source files.

Let's break the list file down into pieces to see what it tells us :

7.1.1.1 Global header

This is the global file header. It details the initial parameters for the SPIN interpreter, and the layout of the object files within the compiled binary.

```
|============================================================================|
Objects : -
test
  |
  +-Blink

Object Address : 0010 : Object Name : test
Object Address : 0028 : Object Name : Blink

Binary Image Information :
PBASE : 0010
VBASE : 0034
DBASE : 0044
PCURR : 001C
DCURR : 004C
|============================================================================|
```

We have two objects. Blink is a sub-object of test. The object code of the “test” object is located at $0010 in the binary image, and the object code of “Blink” is located at $0028.

PBASE is the start of the object code for the first object to run after the Propeller starts up.

VBASE is where the variables (anything declared in a VAR block) start after all the object code.

DBASE is where the stack (used by the SPIN interpreter) starts. (The stack grows from low addresses to high)

PCURR is the address of the first piece of bytecode to execute when the Propeller boots.

DCURR is the initial stack pointer (always 8 bytes higher than the start of the stack)

So in this example, the first piece of bytecode in the “Start” method in object “test” resides at $001C in the hub, and this is where execution will commence.

In the list file, from here down each object is given its own separate section.

7.1.1.2 Object header

```
|============================================================================|
Object test
Object Base is 0010
|============================================================================|
Object Constants
|============================================================================|
Constant _clkmode = 00000408 (1032)
Constant _xinfreq = 004C4B40 (5000000)
|============================================================================|
|============================================================================|
VBASE Global Variables
|============================================================================|
VBASE : 0000 LONG Size 0004 Variable Fred
VBASE : 0004 BYTE Size 0001 Variable Ada
VBASE : 0005 BYTE Size 0001 Variable Jean
VBASE : 0006 BYTE Size 0001 Variable May
|============================================================================|
```

The object header details where the object resides, any constants it contains or exports and what and where its variables are. Constants are displayed with their contents in HEX then brackets containing the Decimal representation. If the constant is a floating point value the representation in brackets is the value in Decimal, while the HEX is what is passed to the compiler.

7.1.1.3 Spin Method

```
|============================================================================|
Spin Block Start with 0 Parameters and 1 Extra Stack Longs. Method 1
PUB Start | X

Local Parameter DBASE:0000 - Result
Local Variable  DBASE:0004 - X
|============================================================================|
13                          X := 1
Addr : 001C:              36  : Constant 2 $00000001
Addr : 001D:              65  : Variable Operation Local Offset - 1 Write
14                          Blink.Go
Addr : 001E:              01  : Drop Anchor
Addr : 001F:       06 02 01   : Call Obj.Sub 2 1
15                          Ada := Jean
Addr : 0022:           88 05  : Memory Op Byte VBASE + READ Address = 0005
Addr : 0024:           89 04  : Memory Op Byte VBASE + WRITE Address = 0004
Addr : 0026:              32  : Return
```

Each SPIN method has at least one local variable (the default RESULT variable). The header for each SPIN method details the location and size of each declared local variable (or parameter).

8 Warranty Statement

As much as I hate to have to do this, we live in a litigious society full of people ready to blame anyone else for their own stupidity (Coffee is HOT people!). Therefore I have to include the following statements. If and when people resume taking responsibility for their own actions, I look forward to being able to remove them.

8.1 Disclaimer of Warranty

THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

8.2 Limitation of Liability.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.