PSPLink v0.9f

A general purpose loader tool for V1.0 and V1.5
PSP homebrew development

Table of Contents

1.Introduction

PSPLINK is licensed under the BSD license, see LICENSE in PSPLINK root for details.

This manual contains the basic information necessary to setting up PSPLINK for development and a short guide on the various features the tool provides. It is assumed that the reader understands the concepts of programming, is able to setup a pspdev.org toolchain and sdk and has at least a basic grasp of how the PSP architecture works.

So what is this?

PSPLINK is an attempt at producing a loader/development application to eliminate the need to go back to the VSH every time. It provides a shell over the SIO, USB or WIFI connection from which you can load new applications, dump useful system information and then reset the PSP to start another application. You can also access some of the features of PSPLINK using a simple on screen console shell, which maps commands to button presses. The program mounts the USB device so you can copy new executables onto the memory stick for execution or you can choose to use the USB host filing system which allows you to access programs directly from your PC. It also eliminates some of the boilerplate code necessary for executables by installing stdio and kprintf handlers for TTY output as well as a built in exception handler, rudimentary debugger and GDB server.

It is designed so that normal user mode applications can be developed as easily as possible, without having to resort to loading first into kernel mode. While it is not designed for developing kernel software it is fully capable of doing so, as well as giving you a more in depth understanding on how the PSP kernel operates by allowing you to inspect much of the system in real time (such as currently loaded module or threads).

2.Installation

If you have downloaded a binary release of psplink then this first step can be skipped. In order to build PSPLINK from source you require the very latest pspdev.org toolchain and PSPSDK. Due to the rapid development of the SDK as well as PSPLINK things can change on a daily basis. As long as you have everything set up correctly the just go to the root of the source and type :

```
make release
```

When the build completes there will be a release directory containing a v1.0 and v1.5 directory. Copy the files appropriate to your revision of the PSP firmware onto the PSP's memory stick under *PSP/GAME*. Now edit the *psplink.ini* file under the *psplink* directory to suit. A more detailed description of the various options available are provided in the appendix, however for basic configuration the following are important.

```
sioshell=[0,1]
```

```
baud=[4800, 9600, ... 152000]
```

Setting *sioshell* to 1 enables the SIO shell, 0 disables it. The *baud* option sets the baudrate of the SIO shell, you are limited to setting it to 4800, 9600, 19200, 38400, 57600 and 115200.

```
wifi=[0..N]
```

```
wifishell=[0,1]
```

The *wifi* setting tells PSPLINK to load up the network modules and connect to a specified wireless access point. If set to 0 wifi is disabled, setting to 1 through N indicates the index of the wifi profile as configured in the VSH. The *wifishell* setting enables (1) or disables (0) the wifi network shell. If

you are using the wifi shell exclusively you will get a slight speed improvement by disabling the SIO shell entirely.

```
usbhost=[0,1]
```

```
usbshell=[0,1]
```

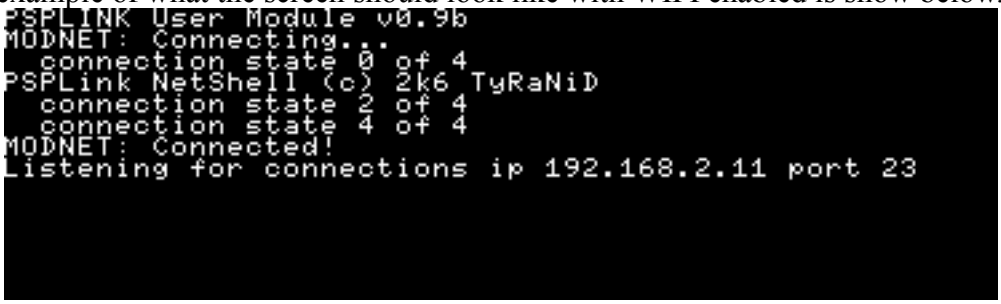
The *usbhost* setting tells PSPLINK to load up the USB HOSTFS driver, this is necessary for usbshell to operate, the *usbshell* setting enables the USB shell. If the HOSTFS driver is not enabled then setting usbshell to 1 will automatically set it.

```
conshell=[0,1]
```

The *conshell* setting tells PSPLINK to load the console shell driver. This is a really simple shell which maps the output text to the screen and you can assign commands to the PSPs buttons in the initialisation file.

When you have finished configuring PSPLINK ensure that the USB drive is unmounted and navigate using the VSH to the memory stick games and run the PSPLINK menu option.

You should now be presented with textual information on the screen indicating that PSPLINK is loading, if you are using the WIFI shell it should now start to enable the network and associate with your access point. If all goes well it will print the IP address to connect to (DHCP should also work). An example of what the screen should look like with WIFI enabled is show below.



If you do not see something like this then double check your settings are correct before continuing.

Now it is just case of connecting to the shell using which ever method you chose. For SIO then a basic terminal application will suffice, the default setup for the serial port is 115200 baud 8N1 with **NO** flow control. Then just type away. Recommended applications include Teraterm or Tutty (putty for serial ports) for windows machines or minicom on a *nix system. You can also use pcterm which comes with the PSPLINK source distribution (which will be explained in more detail later).

For WIFI access you have a number of options, normal telnet applications should suffice. Just tell the application to connect to the printed IP on port 10000 and you can start typing away. Putty is a recommended application for this as it supports a decent local echo mode, default windows telnet is apparently not quite up to the task however. Netcat is another option for the *nix people. However the WIFI shell comes with even less features than the SIO shell in terms of line editing (the technical reason is it works in a line interpreter rather than a character interpreter mode). If you can build it it is likely best to use pcterm to access the shell, the reasons will be explained in the pcterm section later.

For USB access you need to run up the usbhostfs_pc utility. This application will connect to the PSP over USB to provide HOSTFS capability, but it will also open up a socket server on your PC which you can then use using one of the options already outlined for WIFI access.

For console access you should just be able to press the appropriate buttons to load your application.

3. The Shell

PSPLINK is controlled through a text terminal, which to some might seem arcane but it provides the simplest, most portable interface there can be. The output of the PSP at this level is inherently textual so why not use that fact. There is no technical reason why a GUI tool could not be written to control PSPLINK, it is just not been done yet.

When you connect to PSPLINK you will be presented with a prompt (if you don't see one then hit enter and one should appear). From this prompt you can start typing commands to load and run modules, manipulate threads, print screen shots of what is currently on the PSP's screen and much more. The default prompt just prints the current directory you are working in, this prompt is configurable in the *psplink.ini* file or or from the shell.

The simplest command is *help*, type this and hit enter and it will provide you will a list of command categories and brief descriptions. Now just type *help category* to get a list of what commands are available under that section. If a command interests you then type *help name* and it will print some more information about that command including a brief description and parameters. Almost all commands have shorter synonyms which can be used instead of the full name (for example the synonym for *help* is the single character '?'), any synonym is included with the commands help.

The PSPLINK shell interpreter handles most of the normal features you would expect from a shell such as argument quoting (i.e. enclosing arguments in “ or ') and character escaping with \ (useful for escaping spaces or quotes inside other quotes).

There are some special character sequences when using the shell over the serial port that allow you to perform certain operations quickly. These are not available over the WIFI shell, however PCTERM provides mapping to allow you to do the same type of thing. The special mappings are CTRL+R (i.e. hold the control key and press r) which resets PSPLINK, CTRL+S which does a debug instruction step, CTRL+K which does a debug instruction skip, CTRL+N to do a *memdump* command from the current address, CTRL+P to do a memdump in the reverse direction and CTRL+D to exit the shell. See the section on exceptions and debugging for more information about debug stepping. The serial shell also has a simple command line history. By pressing the up and down cursors you can select a previous command to rerun. The size of the history is very limited but it is better than nothing at all.

Loading and Running Modules

The core feature of PSPLINK is the ability to load and run modules, a module in this case refers to either a PRX (a relocatable ELF file) or a PFX (a non-relocatable ELF file, built by default with the pspdev.org toolchain). PSPLINK's design ensures that very little user memory is wasted so almost any application can be run, if all user memory is needed it is possible to disable the user mode part of PSPLINK (with the *pluser* initialisation file option).

There are a number of commands PSPLINK's shell provides for manipulating modules, however only a subset will be described here (see the full command list for the other possible commands).

The simplest way to load and start a module is to pass its name as a path to the shell. The PSPLINK shell maintains a current working directory (which is shown in the prompt) so if the file is in the current directory you can just type ./filename.prx just like you would do on most *nix shells. The PSP file-system works in a similar way to Windows's drive hierarchy. Each type of device be it memory stick, flash or host has a drive prefix *nameX:* where name is the name of the device (e.g. ms) X is the device number (normally 0 unless special). PSPLINK treats paths as relative to the current device, so if you are in ms0:/PSP/GAME and you pass /sample.prx you will get ms0:/sample.prx. You can however pass absolute paths with the device prefix to load from another location. PSPLINK also supports a global path. If you specify a program at the prompt without any directory slashes it will try and search your path for you. If the program is found it will be executed.

Once you start a module the shell will print some useful information such as the UID and name of the module you have just started. This information can be used later to inspect the module

information. You application should now be running and doing what ever you coded it to do. PSPLINK also supports passing command line arguments to the module, this works just as you would expect with any other shell.

When you have finished with your current application you have a few choices. The simplest is to just reset PSPLINK to get back to a nice clean environment. This is done with the *reset* command. You can also stop a module (with *modstop*) then unload it from memory (with *modunld*) however that can leave threads lying about which will cause exceptions if not stopped. An alternative is the *kill* command which will stop your module, hunt out it's threads deleting them in the process and finally unloading the module. This can leave your PSP in an unknown state so it normally best to just reset. It is also important to note that by default PSPLINK will reset itself if the application calls sceKernelExitGame() (such as in an exit callback). This is configurable in the psplink.ini file.

4. Address Calculator

PSPLINK contains a number of commands which take memory addresses as arguments, while you could just use absolute addresses (and you still can) it would be nice to be able to pass a calculated result. PSPLINK supports a number of standard operators and special operators to make calculating addresses as simple as possible. It is worth noting now that the calculator does not have any operator precedence, except for parenthesis, it operates strictly left to right. The upshot of this is a calculation like *1+2*3* would equal 7 in C but would actually equal 9 in PSPLINK. If you need to put in precedence then enclose parts of the calculation in parenthesis, so *1+(2*3)* would get you the answer you would be expecting. Numbers can be in decimal, octal (0 prefix) and hex (0x prefix). Following is a list of the operators and special commands.

Operator	
+	Addition
-	Subtraction
*	Multiplication
/	Division
&	Bitwise AND
|	Bitwise OR
^	Bitwise XOR
<<	Bitwise left shift
>>	Bitwise right shift
~	Bitwise NOT
*	Dereference address
<	Logical less than
>	Logical greater than
<=	Logical less than or equal
>=	Logical greater than or equal
&&	Logical AND
||	Logical OR
==	Logical EQUALS
!=	Logical NOT EQUALS
()	Parenthesis, specifies precedence.

Operator	
`@Modulename@`	Returns the text address of the module Modulename.
`@Modulename:N@`	Returns the address of the section in Modulename this corresponds to N (number 1 to 4) (from SceKernelModuleInfo).
`@Modulename:sN@`	Returns the size of the section in Modulename this corresponds to N (number 1 to 4) (from SceKernelModuleInfo).
`@ModuleName:text@`	Returns the text address (synonym for *@ModuleName@*)
`@ModuleName:stext@`	Returns the text size
`@ModuleName:sdata@`	Returns the data size
`@ModuleName:sbss@`	Returns the bss size.
`@ModuleName:lib,name@`	Return the address of the function name (either a NID or a name) in the library lib from the specified module.
`%ThreadName%`	Extracts ThreadName, finds the thread corresponding to this and returns the entry address.
`%ThreadName:stack%`	Extracts ThreadName, finds the thread corresponding to this and returns the stack address.
`%ThreadName:sstack%`	Extracts ThreadName, finds the thread corresponding to this and returns the stack size.
`?Module:Symbol?`	Extract Module:Symbol, finds the corresponding symbol definition and returns the address of it.
`$reg`	If an exception has occurred extracts reg and returns the value of the specified CPU register. reg is the usual mnemonic MIPS names, e.g. $a0, $v0 plus $epc for the exception program counter and $fsr for the FPU status/control register.

Examples:
`memdump @PSPLINK@+0x100` – Dump memory starting at the base of the PSPLINK module plus 256.
`bp ?SampleMod:main?` - Set a breakpoint at the start of the main function in the SampleMod module.
`savemem $a0 100 memdump.bin` – Save 100 bytes to memdump.bin from address specified in the a0 register of the last exception.

5. Exceptions and Debugging

It is a sad fact of programming C/C++ that you will inevitably crash your application. The default operation of the PSP is to just shut down after 4 seconds when an exception occurs which doesn't really help you track your problem. PSPLINK installs an exception handler for you to catch common errors such as accessing illegal memory addresses. When an exception occurs PSPLINK will catch it and display a register list as well as other useful information. An example is shown below.

```
Exception - Breakpoint
Thread ID - 0x04D2091D
Th Name   - NetSample
Module ID - 0x04D3430F
Mod Name  - NetSample
EPC       - 0x088215DC
Cause     - 0x00000024
```

```
Status    - 0x20008613
BadVAddr  - 0x00000000
zr:0x00000000 at:0x0008FF00 v0:0x00000000 v1:0x00000001
a0:0x00000001 a1:0x09FBFE40 a2:0x00000000 a3:0x00000000
t0:0x0882E45C t1:0x00000006 t2:0x00000000 t3:0x0882E450
t4:0x0000000E t5:0x0882C6F0 t6:0x08823D54 t7:0x00008600
s0:0x00000008 s1:0x09FBFE44 s2:0x00000001 s3:0x09FBFEF0
s4:0x00000008 s5:0x00000013 s6:0xDEADBEEF s7:0xDEADBEEF
t8:0x00000000 t9:0x0005D470 k0:0x09FBFF00 k1:0x00000000
gp:0x08834B30 sp:0x09FBFE40 fp:0x09FBFEB0 ra:0x08820DC8
```

This exception dump tells you the state of the processor when it crashed, the type of exception, the thread it crashed in (if possible) and the module address space it crashed in (if possible). Once an exception has occurred you can use the $reg options in addressing mode to look around at why your code crashed.

A useful feature in these situations is the in built disassembler. If you type *disasm $epc* it will disassemble the instruction at the point of the crash. From here you can determine what register likely caused the crash and search around in memory using commands like *peekw and memdump*.

PSPLINK supports the loading of a custom symbol format so that symbolic information can be extracting during development without having to run up something like psp-objdump. These symbols can also be used in address calculation as previously explained. To generate a symbol file for your application you need a copy of *prxtool* (from subversion) and an un-stripped version of your executable. Run *prxtool* with the -y option to output the compatible format (e.g. *prxtool -y myapp.elf > myapp.sym*). Copy this to a device you can access from the PSP and type *symload myaapp.sym* in the shell. It will only print information if it fails to load the symbols for what ever reason. Now you can just use the symbol information in address calculations. If the symbols comes from a relocatable PRX then PSPLINK will ensure they are fixed up to the correct address before giving you the value no matter where your module was actually loaded.

Sometimes inducing exceptions is actually a good idea, the MIPS processor provides a breakpoint instruction which can be used to step through your code. PSPLINK provides a means of setting one shot breakpoints in your code and stepping through it at the instruction level. To use the debugger you should first load your module without starting it, this is done using the *modload* command (e.g. *modload myapp.elf*). If you have symbols then you can also load them at this point. Finally set your initial breakpoint at the area of interest using the *bpset* command (e.g. *bpset ?MyApp:main?* to break at the main function in your application if you have symbols loaded). Now start your application using the *modstart* command (e.g. *modstart @MyApp*) and if you set it correctly your breakpoint should get executed causing an exception. At this point the current instruction will be printed. You can now step through instructions using the *step* command (or on the serial shell the CTRL+S sequence) or if you want to jump over a *jal* instruction using the *skip* command (CTRL+K). The debugger doesn't work too well for debugging multi-threaded applications as PSPLINK can only track one exception at a time, if another thread errors during the time you are debugging the debug session will be destroyed. However for basic needs, where you don't want to get GDB involved) it is simple and easy to use. A very important thing to note, do not place breakpoints at the entry address of the module. If you do this then PSPLINK will hang because it is waiting for the ModuleStart call to finish which never does. Always ensure you break at the first created thread if you want to debug your application (i.e. set a breakpoint at the *main* function).

6. Debugging with GDB

Sometimes debugging with the built in instruction level debugger is not enough, this is where PSPLINK's GDB Server can come in handy. GDB (GNU Debugger) is a PC application which can connect to remote computers and perform source level debugging, you will need to build a PC version of GDB targeted for the PSP in order to use this feature, this can be done using the pspdev.org toolchain script. You can also build a version of the Insight debugger which is in effect GDB with a simple GUI.

To use GDB you need either a configured WIFI connection or a USB connection, the GDB server does not work over SIO as it cannot be easily shared. To use the USB GDB you must set *usbgdb=1* in your configuration file and restart psplink. Build your application with debugging information in it (i.e. use the -g switch when compiling and linking your code) then copy across a version (stripped or not) to the PSP (this isn't necessary if you are using USB HOSTFS) and execute the command *debug program.elf [args]*. This will start up the GDB server, load program.elf (with args if specified) and listen on your PSP's IP address on port 6999 or setup the communication over USB.

On the PC run the command *psp-gdb program.elf* and type the command *target remote ipaddress:6999* into the GDB shell for WIFI or *target remote :10001* for USB. GDB should connect and indicate the current program has been stopped in the _start routine.

Now you can use GDB as normal, setting breakpoints in your code, inspecting data etc. A word of warning, do not try and step through _start as that can only lead to pain, set a breakpoint on the main function (*break main*) and type 'c' to begin the application. It should now stop at the start of your initial thread. When you are done you can type *kill* to reset PSPLINK or you can type *detach* and the application being debugged will continue running.

You might also find GDB to be a bit slow, unfortunately there isn't a lot you can do about this due to the way that the GDB serial protocol works. It seems especially bad on Insight as that tries to do a lot of things behind the scenes.

The PSPLINK GDB server should be able to debug both ELF and PRX files (although ensure you have the latest up to date PSPSDK for PRX files to work correctly).

7.Scripting Support

PSPLINK supports a very basic script command. A script for PSPLINK is just a text file containing a list of commands to run in order. Scripts also support argument passing, arguments are specified by a $ character then the argument number. $0 refers to the path the script was run from. A special variable $! is provided which resolves to the name of the last module which was loaded. Scripts can be run by passing it to the *run* command, however if the script has the extension .sh it will be picked up by the shell parser and executed directly off the command line (same as it does for modules).

The PSPLINK distribution contains a few example scripts.

8.USB Host filing system

PSPLINK comes with a USB protocol driver to talk to a PC host to create a host filing system. The host filing system is exported as a new IO device *hostX:* (where X is 0 through 7) so once you have set up the driver and the PC side then you can just access files on your PC as if they were on ms0 or flash0.

To enable HostFS on the PSP side set the *usbhost* option in psplink.ini to 1, this setting will override the value of *usbmass*, however once PSPLINK has started you can select a different USB driver to use with the *usbmon/usbmoff/usbhon/usbhoff* commands.

For the PC side of things you first need to build the PC tool, this is under the usbhostfs_pc directory. To build you also need a compiled version of libusb which is used to talk to the device from user space. The libusb page (libusb website) indicates that it should work on most *nix like operating systems such as Linux, FreeBSD, MacOSX as well as on Windows. Building on Windows requires Cygwin, while PSPLINK comes with a pre-prepared driver and library in order to build it you can work from first principles. How to do this is explained in a dedicated section.

For unix like platforms the application can be built by typing *make*, if you are building in Cygwin this should be *make BUILD_WIN32=1* instead. If you are running on a big-endian platform (e.g. PPC MacOSX) then type *make BUILD_BIGENDIAN=1*. On unix like systems the application will need to be run as root so that libusb can access the USB bus, the recommended way of doing this is

change the executable to be owned by root and then set the SUID bit (e.g. *chown root:root usbhostfs_pc; chmod +s usbhostfs_pc*) usbhostfs_pc will drop permissions when they are not needed, it also means that any files you create will be owned by you and not root.

Note that for Windows you also need to install a driver for libusb to operate. When you plug in your PSP with PSPLINK running it should bring up the familiar Add Hardware wizard, follow the steps and when asked specify a location for the driver. Point it at the *windows/driver* directory supplied with PSPLINK and it should install.

The final step is to just run the application, if everything worked then it should print a line indicating it has connected to the device. You should be able to now go to the host0:/ directory in PSPLINK and list the directory or run applications direct from the PC.

The Host FS driver supports up to 8 drive mappings accessed through the devices host0: to host7:. By default the root of all the filing systems is the current working directory, if you want to change that you can specify the directories on the command line as arguments to the PC program. E.g. *./usbhostfs_pc /usr/local/pspdev/psp/sdk/sample /home/user/psp_work* will map host0: to the PSPSDK samples directory and host1: to /home/user/psp_work.

Other useful options are:

-*v* : Verbose mode, prints information about what files/directories are being opened.

-*vv* : Really verbose mode, prints as much information as possible.

-*s port* : Sets the localhost TCP shell port to 'port'

-*g port* : Sets the localhost TCP GDB port to 'port'

-*d* : Print the transfers to and from GDB

-*c* : Enable case-insensitive mode. Attempts to match file names and directories to the closest match ignoring case. Useful for running software which is designed for Memory Stick (which is case insensitive) but the files on the PC have mixed case.

-*f filename* : Load drive mappings from a file. The format is a text file with each line being the form num=path. For example *4=/home/user/projects* will map host4 to /home/user/projects.

-*h* : Print help.

Once the usbhostfs_pc is running you can access a simple shell to control a few aspects of the program without having to close it and restart. The available commands are:

help : Print the list of commands

drives : Print the current drive mappings

mount num path : Mount the directory path as host number num. For example *mount 0 /usr/local* will map /usr/local to host0.

load mapfile : Load a drive mapping file. This file is the same form as the -f switch.

save mapfile : Save out the current drive mappings to a file.

nocase on|off : Set case insensitivity mode to on or off. If no argument is specified it will print the current mode.

gdbdebug on|off : Enable GDB protocol debugging (same as the -d switch).

verbose num : Set the verbose mode, 0 means off, 1 is the same as -v and 2 is the same as -vv.

exit : Exit usbhostfs_pc.

9.PSPLINK Configuration File

PSPLINK has a file in its startup directory called *psplink.ini* which can be used to set-up a selection of parameters before it even starts. The format of the file is similar to a Windows ini file and consists of lines containing key=value pairs. Boolean values are indicated using 0 for false and 1 for true, sometimes the value is a string or a proper number. Following is a brief description of what the commands do.

usbmass=[0 1] – Enables (1) or disables (0) USB mass storage on startup (you can still start USB once the shell has loaded with *usbmasson*).

usbhost=[0 1] – Enables or disables USB host filesystem on startup (you can still start USB host once the shell has loaded with *usbhoston*). Note that this setting overrides *usbmass* in the configuration file, if both are set to true then *usbhost* will be loaded.

pluser=[0 1] – Enables of disables the user mode psplink module. Normally this is left to 1 as certain things like exception support relies on it. However if you need all the memory you can get (and are running the SIO shell) then you can disable it.

resetonexit=[0 1] – Enables or disables the reset on exit handler. If enabled when an application calls sceKernelExitGame PSPLINK will reboot, otherwise a message is printed to the shell but PSPLINK will stay where it is.

sioshell=[0 1] – Enable or disable the SIO shell.

kprintf=[0 1] – Enables kprintf over SIO only, not the full shell. This allows the PSP to reboot faster and eliminates the tty.

wifi=[0..N] – Enables wifi support. Setting to 0 disables the wifi loader, otherwise the number indicates the network configuration to use.

wifishell=[0 1] – Enables or disables the wifi shell. If enabled but wifi is disabled it will enable wifi with the default configuration of 1.

usbshell=[0 1] – Enables or disables the USB shell. If enabled then the *usbhost* setting will be automatically set to 1.

prompt=... - Set the PSPLINK shell prompt. If the string contains %d then that will be expanded to the current working directory at run time.

path=... - Set the default path for PSPLINK. Each path should be absolute (e.g. ms0:/apps) and separated by semi-colons (;).

pcterm=[0 1] – Indicates whether we are communicating to PSPLINK using pcterm or not. Does some extra magic to make pcterm work better.

usbgdb=[0 1] – Indicates that PSPLINK should use the USB gdb stub instead of the WIFI one.

conshell=[0 1] – Enable the console shell.

consinterfere=[0 1] – If set to the 1 specifies that the console shell may interfere with the display.

conscrosscmd=... - Maps a shell command to the cross button.

conssquarecmd=...- Maps a shell command to the square button.

constrianglecmd=...- Maps a shell command to the triangle button.

conscirclecmd=...- Maps a shell command to the circle button.

consselectcmd=...- Maps a shell command to the select button.

consstartcmd=...- Maps a shell command to the start button.

consdowncmd=...- Maps a shell command to the down button.

consleftcmd=...- Maps a shell command to the left button.

consrightcmd=...- Maps a shell command to the down button.

consupcmd=...- Maps a shell command to the up button.

baud=[4800..115200] – Specify the SIO baud rate. Legal values are 4800, 9600, 19200, 38400, 57600 and 115200.

modload=path – Specify a module to auto load at start up. This command can be repeated more than once.

disopt=options – Set the options for the disassembler, see the documentation on the *disset* command for more information about what 'options' should be set to.

10.Command List

The following information is a list of commands which the current version of the PSPLINK shell supports along with more detailed description of use (as opposed to the inbuilt shell's help system). Each entry contains the full name of the command, the synonym (if available) any arguments and a description of purpose. In the list of arguments [x] indicates that the parameter 'x' is optional. A vertical bar (|) indicates you have a choice of two or more different argument types. A '..' indicates multiple values can be added (normally optional). Certain arguments are of special note, to prevent the information being repeated in every entry the following is a description of each.

uid: Indicates a hexadecimal UID is required. This UID should refer to the type of interest, e.g. thread UIDs for thread commands and module UIDs for module functions.
@name: Indicates a textual name prefixed with an '@' character is required. The name should refer to the type of interest, e.g. a module name for a module function.
module:symname: Indicates the command needs the name of a symbol (*symname)* in a specific module *(module)*.
addr: Indicates the command can accept a calculated address argument.
[v]: Used in list commands, if the character 'v' is specified it indicates that the list code should print more detailed information about each entry.

Thread Commands (thread)	
Name	thlist
Synonym	tl
Arguments	[v]
Description	List the threads in the system.
Name	thsllist
Synonym	tl
Arguments	[v]
Description	List the sleeping threads in the system.
Name	thdelist
Synonym	None
Arguments	[v]
Description	List the delayed threads in the system.

Name	thsulist
Synonym	tl
Arguments	[v]
Description	List the suspended threads in the system.
Name	thdolist
Synonym	tl
Arguments	[v]
Description	List the dormant threads in the system.
Name	thinfo
Synonym	ti
Arguments	uid|@name
Description	Print more detailed information about a thread.
Name	thsusp
Synonym	ts
Arguments	uid|@name
Description	Suspend a thread (so it stops executing). To resume a thread use the *thresm* command.
Name	thspuser
Synonym	None
Arguments	None
Description	Suspend all user threads. Do not use if you are running the WIFI shell as that works using a user mode thread which will also get suspended.
Name	thresm
Synonym	tr
Arguments	uid|@name
Description	Resume a thread from where it was suspended.
Name	thwake
Synonym	tw
Arguments	uid|@name
Description	Wakeup a thread from a sleep state (for example due to it calling sceKernelSleepThread).

Name	thterm
Synonym	tt
Arguments	uid|@name
Description	Terminate a thread, this stops the thread from running although it will not remove it from the thread list.
Name	thdel
Synonym	td
Arguments	uid|@name
Description	Delete a thread, this removes the thread and any associated allocated memory from the system.
Name	thtdel
Synonym	tx
Arguments	uid|@name
Description	Terminates a thread then deletes it. Equivalent to calling *threrm* then *thdel.*
Name	thctx
Synonym	tt
Arguments	uid|@name
Description	Prints the current context (i.e. the saved context due to the thread being swapped out) of a thread.
Name	evlist
Synonym	el
Arguments	[v]
Description	List the event flags in the system.
Name	evinfo
Synonym	ei
Arguments	uid|@name
Description	Print more detailed information about an event flag.
Name	smlist
Synonym	sl
Arguments	[v]

Description	List the semaphores in the system.
Name	sminfo
Synonym	si
Arguments	uid|@name
Description	Print more detailed information about a semaphore.
Name	mxlist
Synonym	xl
Arguments	[v]
Description	List the message boxes in the system.
Name	mxinfo
Synonym	xi
Arguments	uid|@name
Description	Print more detailed information about a message box.
Name	cblist
Synonym	cl
Arguments	[v]
Description	List the callbacks in the system.
Name	cbinfo
Synonym	ci
Arguments	uid|@name
Description	Print more detailed information about a callback.
Name	vtlist
Synonym	zl
Arguments	[v]
Description	List the vtimers in the system.
Name	vtinfo
Synonym	zi
Arguments	uid|@name
Description	Print more detailed information about a vtimer.
Name	vpllist

Synonym	vl
Arguments	[v]
Description	List the VPL's in the system
Name	vplinfo
Synonym	vi
Arguments	uid|@name
Description	Print more detailed information about a VPL
Name	fpllist
Synonym	fl
Arguments	[v]
Description	List the FPL's in the system
Name	fplinfo
Synonym	fi
Arguments	uid|@name
Description	Print more detailed information about a FPL
Name	mpplist
Synonym	pl
Arguments	[v]
Description	List the message pipes in the system
Name	mppinfo
Synonym	pi
Arguments	uid|@name
Description	Print more detailed information about a message pipe
Name	thevlist
Synonym	tel
Arguments	[v]
Description	List the thread event handlers in the system
Name	thevinfo
Synonym	tei
Arguments	uid|@name
Description	Print more detailed information about a thread event

	handler.
Module Commands (module)	
Name	modlist
Synonym	ml
Arguments	[v]
Description	List the modules in the system.
Name	modinfo
Synonym	mi
Arguments	uid|@name
Description	Print more detailed information about a module
Name	modstop
Synonym	ms
Arguments	uid|@name
Description	Stop a module (necessary in order to unload it). Doing this will call the module_stop function on the module (if available).
Name	modunld
Synonym	mu
Arguments	uid|@name
Description	Unload a module from memory (must be stopped first).
Name	modload
Synonym	md
Arguments	path
Description	Load a module into memory from the specified file path.
Name	modstart
Synonym	mt
Arguments	uid|@name [args]
Description	Start a loaded module with optional arguments. NOTE: like the exec commands on *nix you must manually supply the path to the module as the first argument if you want the usual argv[0] handling to work.

Name	modexec
Synonym	me
Arguments	[@key] path [args]
Description	Run a module using the sceKernelLoadExec call. Key is optional, it can be set to @game, @vsh or @updater which specifies the mode which the PSP reboots into. NOTE: This will kill everything which is currently running including PSPLINK so only use if you really need to.
Name	modaddr
Synonym	ma
Arguments	addr
Description	Print the module information for the module at the specified address. This will only pick up the loaded sections of a module not anything like later heap allocations.
Name	exec
Synonym	e
Arguments	[path] [args]
Description	Execute a new module under PSPLINK. This saves the path and arguments away for later use. If exec has already been run in this session PSPLINK will reboot before executing the module. If you do not specify a path then PSPLINK will reuse the ones from the current exec session (if available).
Name	ldstart
Synonym	ld
Arguments	path [args]
Description	Load and start a module but do not save away the information. Can be used for loading modules from flash or just general use.
Name	debug
Synonym	d
Arguments	path [args]
Description	Start an application under the GDB debugger. You will need to load up a copy of psp-gdb or psp-insight to continue debugging, however the shell is still active and can be used to load other things while debugging

	through GDB. See the section on GDB for more information.
Name	kill
Synonym	k
Arguments	uid|@name
Description	Kill a module, stop it (so module_stop can do a sensible cleanup) then terminate and delete any remaining threads. Finally the module will be unloaded.
Name	modexp
Synonym	mp
Arguments	uid|@name
Description	Print the export libraries from a module with NID and address.
Name	modimp
Synonym	None
Arguments	uid|@name
Description	Print the import libraries in a module with NID and address.
Name	modfindx
Synonym	mfx
Arguments	uid|@name library nid|@name
Description	Find the address of a specified module export. Three parameters are required, the module to find the export in, the library name and the nid or valid name of the function. If name is specified then the NID value will be generated automatically using the SHA1 functions. For example 'modfindx @sceIOFileManager IoFileMgrForUser @sceIoOpen' will find the address of the sceIoOpen function in IoFileMgrForUser.
Name	apihook
Synonym	None
Arguments	uid|@name library nid|@name [param]
Description	Hooks a user mode API call (i.e. one which goes through the syscall gateway). The first three parameters are the same as used in modfindx, the forth parameter allows you to specify how the arguments should be formatted. Only 6 parameters are

	supported, param is a list of characters which will format these 6 parameters. 'i' indicates print as an integer, 'x' print as hex, 'o' print as octal and 's' print as a string. For example to hook sceIoOpen use *apihook @sceIOFileManager IoFileMgrForUser @sceIoOpen sxo* which hooks sceIoOpen formatting the parameters as a string, hexadecimal integer and an octal integer.
Name	apihooks
Synonym	None
Arguments	uid|@name library nid|@name [param]
Description	Acts the same as the *apihook* command except when the hook is called the calling thread will be put into a sleep mode. This can be used to inspect the parameters in more detail. To restart the thread use the *thwake* command with the printed thread ID.
Name	apihp
Synonym	None
Arguments	None
Description	Print the currently installed API hooks.
Name	apihd
Synonym	None
Arguments	id
Description	Delete the API hook with the specified ID.
Memory Commands (memory)	
Name	meminfo
Synonym	mf
Arguments	[patitionid]
Description	Print the kernel list of memory allocation blocks. Partitions are numbers 1 through 4, 1 is kernel space, 2 is user space, 3 is a mirror of the kernel space and 4 is used by the Media Engine. If partitionid is specified it will only print out the corresponding block information. Displays the amount of free memory in the system.
Name	memreg

Synonym	mr
Arguments	None
Description	Prints the accessible memory regions. These are the regions which PSPLINK consider safe to read and write to.
Name	memdump
Synonym	dm
Arguments	[-|addr] [b|h|w]
Description	Dumps 256 bytes of data to screen (in a hexeditor style mode) from the specified address. If the address is not specified then the previous address plus 256 is displayed. If – is specified then the memory at address - 256 will be printed. If the last argument is one of b, h or w then the hexdump will be printed in bytes, half-words or words as appropriate. Bytes is the default. The value is saved for when calling memdump with no arguments.
Name	memblocks
Synonym	mk
Arguments	[f|t]
Description	Print the sysmem block tables to Kprintf. Without arguments prints a concise list. With 'f' print the full table of blocks. With 't' print the tail blocks.
Name	savemem
Synonym	sm
Arguments	addr size path
Description	Save the block of memory of size *size* specified to a path.
Name	loadmem
Synonym	lm
Arguments	addr path [maxsize]
Description	Load the block of memory to the specified address from path. If maxsize is specified then it wont load more than that value no matter how big the file is.
Name	pokew
Synonym	pw
Arguments	addr val1 [val2..valN]

Description	Poke 1 or more words (32bit) into memory at the specified address. If more than one word is specified then the address will be incremented accordingly.
Name	pokeh
Synonym	ph
Arguments	addr val1 [val2..valN]
Description	Poke 1 or more half-words (16bit) into memory at the specified address. If more than one half-word is specified then the address will be incremented accordingly.
Name	pokeb
Synonym	pb
Arguments	addr val1 [val2..valN]
Description	Poke 1 or more bytes (8bit) into memory at the specified address. If more than one bytes is specified then the address will be incremented accordingly.
Name	peekw
Synonym	kw
Arguments	addr [o|d|x|f]
Description	Peek at a word (32bit) in memory. By passing o,d,x or f as the last argument indicates the result should be printed in octal, decimal, hexadecimal or floating point.
Name	peekh
Synonym	kh
Arguments	addr [o|d|x]
Description	Peek at a half-word (16bit) in memory. By passing o,d or x as the last argument indicates the result should be printed in octal, decimal, hexadecimal or floating point.
Name	peekb
Synonym	kb
Arguments	addr [o|d|x]
Description	Peek at a byte (8bit) in memory. By passing o,d or x as the last argument indicates the result should be printed in octal, decimal, hexadecimal or floating point.

Name	fillw
Synonym	fw
Arguments	addr size val
Description	Fills a region of memory with word values. Size indicates the number of words to write.
Name	fillh
Synonym	fh
Arguments	addr size val
Description	Fills a region of memory with half-word values. Size indicates the number of half-words to write.
Name	fillb
Synonym	fb
Arguments	addr size val
Description	Fills a region of memory with byte values. Size indicates the number of bytes to write.
Name	copymem
Synonym	cm
Arguments	srcaddr destaddr size
Description	Copys memory from one src to dest.
Name	findstr
Synonym	ns
Arguments	add size str
Description	Finds an ASCII string in memory, all characters can be used by escaping them (\xXX for hexadecimal or \0XXX for octal), excluding the NUL character. Prints a list of matching regions.
Name	findhex
Synonym	nx
Arguments	addr size hexstr [mask]
Description	Find a string of hex digits in memory, if mask is specified each byte will be ANDed before matching. For example findhex 01234567 0FFFFFFF will find a string of bytes in memory with the pattern X1 23 45 67 where X is don't care.

Name	findw
Synonym	nw
Arguments	addr size val1 [val2..valN]
Description	Find a string of words (32bit) in memory. More than one value can be specified.
Name	findh
Synonym	nh
Arguments	addr size val1 [val2..valN]
Description	Find a string of half-words (16bit) in memory. More than one value can be specified.
Name	dcache
Synonym	dc
Arguments	w|i|wi [addr size]
Description	Performs an operation on the data cache of the MIPS processor. Setting w will writeback the cache, i invalidates the cache and wi will writeback then invalidate the cache. If addr and size are specified only that region of memory will be affected.
Name	icache
Synonym	ic
Arguments	[addr size]
Description	Invalidates the instruction cache of the MIPS processor. If addr and size are specified only that region of memory will be affected.
Name	disasm
Synonym	di
Arguments	addr [count]
Description	Disassemble instructions at addr, if count is specified it indicates the number of instructions to disassemble.
Name	disopts
Synonym	None
Arguments	None
Description	Print the current disassembler options. Each option is signified by a single character, the possible options are:

	• 'x' – Display immediate integers as hex rather than signed decimal. • 'r' – Disable printing of mnemonic CPU registers (i.e. print $2 instead of $v0) • 's' – Print the corresponding symbol for the PC of the instruction if it is available (e.g. the fourth instruction in the main function will be printed as 'main+0x10') • 'm' – Disable macro instructions such as nop and li. • 'p' – Print the real address of a load/store or for the jr instruction. This only works in debugging mode and only when it prints the first instruction on break. It converts such information as ofs($reg) or jr $reg to the real address based on the exception registers with symbol fixup if available.
Name	disset
Synonym	None
Arguments	options
Description	Set one or more disassembler options. Options is a list of 1 or more characters corresponding to the list presented in disopts.
Name	disclear
Synonym	None
Arguments	options
Description	Clear one or more disassembler options. Options is a list of 1 or more characters corresponding to the list presented in disopts.
FileIO Commands (fileio)	
Name	ls
Synonym	dir
Arguments	[path1..pathN]
Description	Lists the files in a specified directory(s). If no paths are specified then will list the current directory.
Name	chdir
Synonym	cd
Arguments	dir

Description	Changes the current working directory of the PSPLINK shell.
Name	cp
Synonym	copy
Arguments	sourcepath destpath
Description	Copies a file from sourcepath to destpath.
Name	mkdir
Synonym	None
Arguments	dir
Description	Make a directory with the specified path.
Name	rm
Synonym	del
Arguments	path
Description	Deletes a file with the specified path.
Name	rmdir
Synonym	rd
Arguments	dir
Description	Removes a directory with the specified path.
Name	rename
Synonym	ren
Arguments	src dst
Description	Renames the file src to dst.
Name	remap
Synonym	None
Arguments	devfrom: devto:
Description	Remaps the device 'devfrom:' to the device 'devto:'. This allows you to alias say fatms0: to host0: if you are not running the USB host driver. (Note that it is fatms0: not ms0:, using ms0: will fail).
Name	pwd
Synonym	None
Arguments	None

Description	Print the current working directory.
Debugger Commands (debugger)	
Name	exprint
Synonym	ep
Arguments	None
Description	Print the last exception register information to screen.
Name	exresume
Synonym	c
Arguments	[addr]
Description	Resume from the last exception. If addr is specified sets this to epc and resumes from there.
Name	exprfpu
Synonym	ef
Arguments	None
Description	Print the last exceptions FPU registers.
Name	setreg
Synonym	str
Arguments	$reg value
Description	Sets the last exceptions register $reg to value.
Name	bpset
Synonym	bp
Arguments	addr
Description	Set a one-shot breakpoint at the specified address. Once the breakpoint has occurred it will be cleared.
Name	bpprint
Synonym	bt
Arguments	None
Description	Print the current list of breakpoints.
Name	step

Synonym	s (or CTRL+S for SIO shell)
Arguments	None
Description	Step a single MIPS instruction.
Name	skip
Synonym	k (or CTRL+K for SIO shell)
Arguments	None
Description	Step a single MIPS instruction, if the next instruction is jal or jalr then break immediately after the call.
Name	symload
Synonym	syl
Arguments	path
Description	Load symbols from the specified path.
Name	symlist
Synonym	syt
Arguments	None
Description	List the currently loaded symbol files.
Name	symprint
Synonym	syp
Arguments	modname
Description	List the symbols for the specified module.
Name	symbyaddr
Synonym	sya
Arguments	addr
Description	Print the symbol information at the specified address.
Name	symbyname
Synonym	syn
Arguments	module:symname
Description	Print the address of the symbol in the specified module with the symbol name.
Miscellaneous Commands (misc)	

Name	usbmon
Synonym	umn
Arguments	None
Description	Enable the USB mass storage device drivers.
Name	usbmoff
Synonym	umf
Arguments	None
Description	Disable the USB mass storage device drivers.
Name	usbhon
Synonym	uhn
Arguments	None
Description	Enable the USB HostFS device drivers.
Name	usbhoff
Synonym	uhf
Arguments	None
Description	Disable the USB HostFS device drivers.
Name	usbstat
Synonym	us
Arguments	None
Description	Print the status of the USB device connection.
Name	uidlist
Synonym	ul
Arguments	[root]
Description	Print the list of system UIDs. If root is specified only the specific section of the uid list will be printed, e.g. 'uidlist Thread' will only print thread uids. Possible values for root are Vtimer, Alarm, Delay, Timer, Fpl, Vpl, MsgPipe, Mbx, EventFlag, Semaphore, Callback, ThreadEventHandler, Thread, WaitQ, SceModule, SceSysMemoryBlock, SceSysmemMemoryPartition and SceSysmemHeap.
Name	exit
Synonym	quit (or CTRL+D for SIO shell)

Arguments	None
Description	Exit the shell and return back to Sony's VSH.
Name	set
Synonym	None
Arguments	[var=value]
Description	If passed with no arguments will print the list of PSPLINK variables. If the argument is passed then will set the variable var to the value. The currently available variables are *prompt* which sets the PSPLINK prompt and *path* which sets the path.
Name	scrshot
Synonym	ss
Arguments	path
Description	Take a screen shot of the current display and write it to a 24bit colour Windows bitmap file.
Name	run
Synonym	None
Arguments	path [args]
Description	Run a PSPLINK script. Scripts are just a textual file with a list of shell commands to run. See the section on scripts for more information.
Name	cop0
Synonym	c0
Arguments	None
Description	Prints the current COP0 registers for the MIPS CPU.
Name	calc
Synonym	None
Arguments	addr [d|o|x]
Description	Calculates the address as per the section on address calculation. If d, o or x is specified print the result in decimal, octal or hex.
Name	reset
Synonym	r (or CTRL+R in SIO shell)
Arguments	None

Description	Reset PSPLINK.
Name	wifi
Synonym	None
Arguments	[ap]
Description	Enables the WIFI drivers if they have not been already. ap is a number 1..N which indicates the WIFI configuration to use.
Name	wifishell
Synonym	None
Arguments	[ap]
Description	Enables WIFI and the WIFI shell. ap is a number 1..N which indicates the WIFI configuration to use.
Name	ver
Synonym	v
Arguments	None
Description	Print the version of PSPLINK.
Name	pspver
Synonym	None
Arguments	None
Description	Print the PSP version number as returned by sceKernelDevkitVersion
Name	config
Synonym	None
Arguments	None
Description	Prints the current psplink.ini settings.
Name	confset
Synonym	None
Arguments	name [value]
Description	Set a psplink.ini configuration setting. If name doesn't exist then it is added, it is then set to value. If value is not specified then the setting is empty. Note that once this is run any extract information in the configuration file (for example comments) will be stripped out.

Name	confdel
Synonym	None
Arguments	name
Description	Deletes a configuration option from psplink.ini.
Name	tty
Synonym	None
Arguments	None
Description	Enter tty mode. In this mode all shell input is redirected to the stdin handler so an interactive application can be written. To exit from tty mode back to shell mode you must type ~. at the start of the line.
Name	power
Synonym	None
Arguments	None
Description	Print information about the current power state (e.g. whether AC is connected, batter life, CPU speed etc.).
Name	clock
Synonym	None
Arguments	cpu ram bus
Description	Sets the clock speed of the PSP. For example calling *clock 333 333 166* will set the PSP to 333Mhz CPU speed, 333Mhz ram speed and 166Mhz bus speed.
Name	tonid
Synonym	None
Arguments	name
Description	Converts a textual name to a NID.
Name	profmode
Synonym	None
Arguments	[t|g|o]
Description	Sets the current profiler mode. The profiler is a hardware device which can give you information about things like the number of cache misses or the number of clock cycles used in your application. By default

	the profiler is only available in kernel mode but it can be set to operate in a global or thread mode where it can be accessed by user mode applications. Passing t as an argument sets thread mode where the profiler is context switched so only the information reflects the current thread, passing g sets global mode which runs constantly for all threads and passing o will disable the profiler mode. You must reset PSPLINK to get the profiler into the specified mode, from then on it will stay as set. To access the profilers either call sceKernelReferGlobalProfiler or sceKernelReferThreadProfiler in your application and set the enable member to 1 to start the counters. Specifying no arguments will print the current mode.
Name	debugreg
Synonym	None
Arguments	[val]
Description	Set or print the kernel's debug register value.
Name	help
Synonym	?
Arguments	[command|category]
Description	Print some simple help. If a command name is specified more detailed information will be printed. If a category is specified then the commands under that will be listed.

11.PCTerm

PCTerm is a simple command line tool to access the PSPLINK shell. It works on both serial and wifi connections and uses readline to give a fully featured history and command line editing.

To build it you need to install libreadline 5, then just type *make* to build the tool.

Using PCTerm is also pretty easy, the options you can pass are as follows.

-s – Enable serial mode.

-b baud – Specify the serial baud rate.

-p port – Specify the network port.

-r retries – Specify how many times the network code should retry a connection.

-h history – Specify a new history file

The final argument is optional (it will default to localhost for WIFI or /dev/ttyS0 for serial), if your settings are not the same then it should be either a network address for the PSP (in WIFI/USB mode) or a serial port file (e.g. /dev/ttyS0) for serial mode.

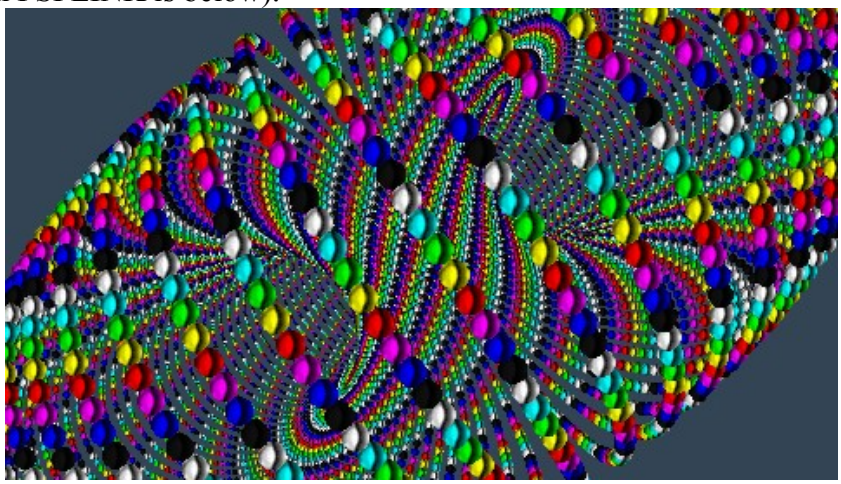
As pcterm works in a line by line mode some of the shortcut keys for the shell are not available. These are remapped to ALT+ the character used by the serial shell. e.g. CTRL+S for step instruction

is mapped to ALT+S.

12.Example of Use (USB Mass Storage)

This is a simple example for compiling a PSPLINK compatible executable and running it inside the shell utilising USB Mass Storage.

1. Compile up the gu/sprite sample from PSPSDK. Assuming you have correctly installed the toolchain this is just a case of changing to the /usr/local/pspdev/psp/sdk/sample/gu/sprite directory and type *make*.
2. Start up PSPLINK and ensure the USB mass storage driver is enabled (either from setting *usbmass*= to 1 in *psplink.ini* or at the shell typing *usbmon*).
3. Mount the USB mass storage device in your operating system. On Windows this is likely to be automatic, under *nix you might need to use the *mount* command.
4. Copy the sprite.elf file built in the sample directory to somewhere on the mass storage device, we assume from here on in it was copied to the root of the memory stick. Now unmount the drive to ensure all pending writes are flushed.
5. Connect to the PSPLINK shell using your preferred method, for example with pcterm using *pcterm -s /dev/ttySX* for the serial shell or *pcterm 1.2.3.4* for the network shell.
6. Reset PSPLINK (using *reset* or CTRL+R or ALT+R depending on the connection) to ensure that the internal write have been flushed to the memory stick.
7. When PSPLINK starts up again you should already be in the memory stick root (indicated in the prompt with the string *ms0:/*). Type *./sprite.elf* at the shell and hit the enter key. PSPLINK should now display a bit of information about the module it is running and on the PSP screen there should be the sprite sample running (if you have not seen it a screenshot taken with PSPLINK is below).



8. Type *modinfo "@Sprite Sample"* to print the module information. Note the value of TextAddr is 0x8900000, this is a pretty clear indication this is a PFX file.
9. Now you can either kill the sample with the *kill* command (type *kill "@Sprite Sample"*) or reset PSPLINK again to get back to a clean start.
10. Try and build the sprite sample now as a PRX file. Edit the Makefile and add the line *BUILD_PRX=1* after the *ASFLAGS* line and type *make clean; make*. You should now have a sprite.prx file in the sample directory.
11. Copy this file to the memory stick using the previous process and run it with *./sprite.prx*. Run the *modinfo* command again and note that TextAddr has completely changed.
12. Finally type *exit* and the PSP should exit back to the PSP's VSH.

13.Example of Use (USB HostFS)

This is a simple example for compiling a PSPLINK compatible executable and running it inside the shell utilising the USB host filing system.

1. Compile up the gu/sprite sample from PSPSDK. Assuming you have correctly installed the toolchain this is just a case of changing to the /usr/local/pspdev/psp/sdk/sample/gu/sprite directory and type *make*.
2. Start up PSPLINK and ensure the USB host driver and USB shell is enabled (setting *usbhost=1* and *usbshell=1*).
3. Start up the PC side of the hostfs driver with the by running the command *usbhostfs_pc /usr/local/pspdev/psp/sdk/samples/gu/sprite.* You should see a line saying it connected to the device.
4. Connect to the PSPLINK shell using pcterm by just running it with no arguments. You should now see the prompt *host0:/>* displayed on your terminal. Type *ls* and it should print the contents of the sprite sample directory, including the executable sprite.elf.
5. Type *./sprite.elf* at the shell and hit the enter key. PSPLINK should now display a bit of information about the module it is running and on the PSP screen there should be the sprite sample running.
6. Type *modinfo "@Sprite Sample"* to print the module information. Note the value of TextAddr is 0x8900000, this is a pretty clear indication this is a PFX file.
7. Now you can either kill the sample with the *kill* command (type *kill "@Sprite Sample"*) or reset PSPLINK again to get back to a clean start.
8. Try and build the sprite sample now as a PRX file. Edit the Makefile and add the line *BUILD_PRX=1* after the *ASFLAGS* line and type *make clean; make*. You should now have a sprite.prx file in the sample directory.
9. Run it with *./sprite.prx*. Run the *modinfo* command again and note that TextAddr has completely changed.
10. Finally type *exit* and the PSP should exit back to the PSP's VSH.

14.USB HostFS for Windows

Note that this section is only for people who want to set up the USB driver from scratch on Windows. Building the PC side of HOSTFS (and consequently the shell and gdb servers) is slightly more involved than it is on Unix systems. Follow the steps below.

1. Ensure you have Cygwin installed along with compilers etc. Download the latest snapshot of the libusb-win32 device driver package from http://libusb-win32.sourceforge.net/, it will be called something like libusb-win32-device-bin-XXXXXX.tar.gz.
2. Extra the libusb package to a directory somewhere, extract the psplink source code to somewhere Cygwin can access. From the extracted directory copy include/usb.h and lib/gcc/libusb.a to the usbhostfs_pc directory.
3. Install PSPLINK on your PSP, ensure that the *usbhost* setting is set to 1, start PSPLINK on the PSP with a USB cable plugged in.
4. Windows should now ask to install a driver for a 'PSP Type-B' device, ignore this for now. Go to the bin directory in the extracted libusb directory and run the inf-wizard.exe program. Click next and you should see the following dialog.

5. Ensure that PSP Type B is show, select it and click Next.
6. On the next dialog confirm the information looks correct and click Next again.
7. Now it will ask you for the output location for the .inf file. Navigate to the libusb bin directory and name the file psp.inf, this will create the files needed for the device driver to install.
8. Now go back to the Windows device installer dialog which we ignored earlier. Choose to browse for the driver yourself and point it at the libusb bin directory, Windows should now pick up your device correctly and install it.
9. Go to the usbhostfs_pc directory in Cygwin and type *make*. You will get some format warnings but it should generate a executable. If it complains about not finding usb.h or libusb then ensure you have copied them across as per step 2. Note there is no need to set SUID root or anything on Windows, however you will almost certainly need to be running as an Administrator.
10. With PSPLINK running run up usbhostfs_pc.exe, the text 'Connected to device' should now be printed on the terminal. Now using your preferred shell method connect and you should be able to list the host directories. If you want to use the usbshell (and you have set the *usbshell* option to 1) you can now use pcterm to connect to the local network address and you should see the USB shell. If you do not get a prompt printed then hit enter and one should appear.

15.Glossary

Term	*Description*
ELF	Executable and Linkable Format. A file format specified by the Tool Interface Standards Committee (TIS) used in the PSP for its executable objects.
PRX	Playstation Relocatable eXecutable. A specially formatted ELF file which has not been linked to a fixed location. The PSP kernel is capable to change the position of the executable in memory at load time.
PFX	Playstation Fixed eXecutable. A specially formatted ELF file which has been linked to a fixed location. When the PSP kernel loads the executable it will try and place it at the specified address. If this is not possible the load will fail. This is the default output format of PSPSDK, i.e. a plain ELF file.
Module	A module in this context refers to any executable (PRX or PFX) which the PSP kernel is able to load into memory and execute.
SIO	Serial IO, a term to refer to communicating with a device over a serial port.

16.Legal

17.Thanks and Greetz

Thanks to Hitmen for providing some very useful information.

Thanks to pspdev.org for hosting this stuff.

Nil for implementing the original SIO cable and Marcus Comstedt for the hardware information.

And thanks to the people who have bothered to test the code and provide suggestions, in no particular order Oobles, dot_blank, ginka, sharkus, Shazz, nullp01nter, johnmph.