

# PSPLink v0.9b

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

(c) TyRaNiD 2005/2006

(c) Julian T 2005/2006

(c) John\_K 2005

# 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](http://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 or WIFI connection from which you can load new applications, dump useful system information and then reset the PSP to start another application. The program mounts the USB device so you can copy new executables onto the memory stick for execution. 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).

## Installation

In order to build PSPLINK from source you require the very latest [pspdev.org](http://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 setup correctly the just goto 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 details 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.

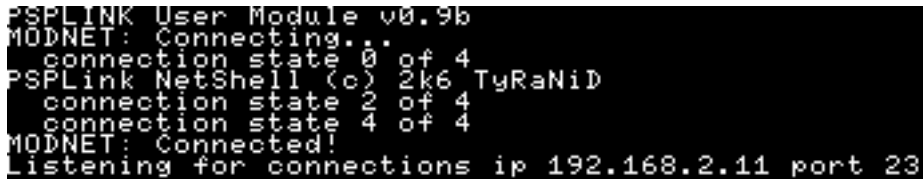
```
wifi=[0..N]
```

```
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.

When you have finished configuring PSPLINK ensure that the USB drive is unmounted and navigate to 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.



```
PSPLINK User Module v0.9b
MODNET: Connecting...
connection state 0 of 4
PSPLink NetShell (c) 2k6 TyRaNiD
connection state 2 of 4
connection state 4 of 4
MODNET: Connected!
Listening for connections ip 192.168.2.11 port 23
```

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. Ensure you have hardware and software floating point disabled as well (if the application supports it Carrier Detect monitoring). Then just type away. Recommended applications include Teraterm or Tuffy (putty for serial ports) for windows machines or minicom on a \*nix system. You can also use pterm 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 23 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 pterm to access the shell, the reasons will be explained in the pterm section later.

## 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](http://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.

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. Your 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's 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.

## **2.Address Calculator**

PSPLINK contains a number of command 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.
@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.
%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.

Operator	
?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.

#### 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.

## 3.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 (kindly provided by Hitmen). 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.

## 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.

## 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.

*usb=[0 1]* – Enables (1) or disables (0) USB mass storage on startup.

*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 of exit handler. If enabled when an application calls sceKernelExitGame PSPLINK will reboot, otherwise a message is printed to the screen but PSPLINK will stay where it is.

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

*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.

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

*pcterm*=[0 1] – Indicates whether we are communicating to PSPLINK using pterm or not.

*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.

## 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.

<b>Thread Commands (thread)</b>	
<b>Name</b>	thlist
<b>Synonym</b>	tl
<b>Arguments</b>	[v]
<b>Description</b>	List the threads in the system.



<b>Name</b>	thinfo
<b>Synonym</b>	ti
<b>Arguments</b>	uid @name
<b>Description</b>	Print more detailed information about a thread.
<b>Name</b>	thsusp
<b>Synonym</b>	ts
<b>Arguments</b>	uid @name
<b>Description</b>	Suspend a thread (so it stops executing). To resume a thread use the <i>thresm</i> command.
<b>Name</b>	thresm
<b>Synonym</b>	tr
<b>Arguments</b>	uid @name
<b>Description</b>	Resume a thread from where it was suspended.
<b>Name</b>	thwake
<b>Synonym</b>	tw
<b>Arguments</b>	uid @name
<b>Description</b>	Wakeup a thread from a sleep state (for example due to it calling <i>sceKernelSleepThread</i> ).
<b>Name</b>	thterm
<b>Synonym</b>	tt
<b>Arguments</b>	uid @name
<b>Description</b>	Terminate a thread, this stops the thread from running although it will not remove it from the thread list.
<b>Name</b>	thdel
<b>Synonym</b>	td
<b>Arguments</b>	uid @name
<b>Description</b>	Delete a thread, this removes the thread and any associated allocated memory from the system.
<b>Name</b>	thtdel
<b>Synonym</b>	tx
<b>Arguments</b>	uid @name

<b>Description</b>	Terminates a thread then deletes it. Equivalent to calling <i>threrm</i> then <i>thdel</i> .
<b>Name</b>	thctx
<b>Synonym</b>	tt
<b>Arguments</b>	uid @name
<b>Description</b>	Prints the current context (i.e. the saved context due to the thread being swapped out) of a thread.
<b>Name</b>	evlist
<b>Synonym</b>	el
<b>Arguments</b>	[v]
<b>Description</b>	List the event flags in the system.
<b>Name</b>	evinfo
<b>Synonym</b>	ei
<b>Arguments</b>	uid @name
<b>Description</b>	Print more detailed information about an event flag.
<b>Name</b>	smlist
<b>Synonym</b>	sl
<b>Arguments</b>	[v]
<b>Description</b>	List the semaphores in the system.
<b>Name</b>	sminfo
<b>Synonym</b>	si
<b>Arguments</b>	uid @name
<b>Description</b>	Print more detailed information about a semaphore.
<b>Name</b>	mxlist
<b>Synonym</b>	xl
<b>Arguments</b>	[v]
<b>Description</b>	List the message boxes in the system.
<b>Name</b>	mxinfo
<b>Synonym</b>	xi
<b>Arguments</b>	uid @name
<b>Description</b>	Print more detailed information about a message box.

<b>Name</b>	cblist
<b>Synonym</b>	cl
<b>Arguments</b>	[v]
<b>Description</b>	List the callbacks in the system.
<b>Name</b>	cbinfo
<b>Synonym</b>	ci
<b>Arguments</b>	uid @name
<b>Description</b>	Print more detailed information about a callback.
<b>Name</b>	vtlist
<b>Synonym</b>	tl
<b>Arguments</b>	List the vtimers in the system.
<b>Description</b>	
<b>Name</b>	vtinfo
<b>Synonym</b>	ti
<b>Arguments</b>	uid @name
<b>Description</b>	Print more detailed information about a vtimer.
<b>Name</b>	vpllist
<b>Synonym</b>	vl
<b>Arguments</b>	[v]
<b>Description</b>	List the VPL's in the system
<b>Name</b>	vplinfo
<b>Synonym</b>	vi
<b>Arguments</b>	uid @name
<b>Description</b>	Print more detailed information about a VPL
<b>Name</b>	fpllist
<b>Synonym</b>	fl
<b>Arguments</b>	[v]
<b>Description</b>	List the FPL's in the system
<b>Name</b>	fplinfo
<b>Synonym</b>	fi

<b>Arguments</b>	uid @name
<b>Description</b>	Print more detailed information about a FPL
<b>Name</b>	mpplist
<b>Synonym</b>	pl
<b>Arguments</b>	[v]
<b>Description</b>	List the message pipes in the system
<b>Name</b>	mppinfo
<b>Synonym</b>	pi
<b>Arguments</b>	uid @name
<b>Description</b>	Print more detailed information about a message pipe
<b>Module Commands (module)</b>	
<b>Name</b>	modlist
<b>Synonym</b>	ml
<b>Arguments</b>	[v]
<b>Description</b>	List the modules in the system.
<b>Name</b>	modinfo
<b>Synonym</b>	mi
<b>Arguments</b>	uid @name
<b>Description</b>	Print more detailed information about a module
<b>Name</b>	modstop
<b>Synonym</b>	ms
<b>Arguments</b>	uid @name
<b>Description</b>	Stop a module (necessary in order to unload it). Doing this will call the module_stop function on the module (if available).
<b>Name</b>	modunld
<b>Synonym</b>	mu
<b>Arguments</b>	uid @name
<b>Description</b>	Unload a module from memory (must be stopped first).
<b>Name</b>	modload

<b>Synonym</b>	md
<b>Arguments</b>	path
<b>Description</b>	Load a module into memory from the specified file path.
<b>Name</b>	modstart
<b>Synonym</b>	mt
<b>Arguments</b>	uid @name [args]
<b>Description</b>	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.
<b>Name</b>	modexec
<b>Synonym</b>	me
<b>Arguments</b>	path [args]
<b>Description</b>	Run a module using the sceKernelLoadExec call. NOTE: This will kill everything which is currently running including PSPLINK so only use if you really need to.
<b>Name</b>	modaddr
<b>Synonym</b>	ma
<b>Arguments</b>	addr
<b>Description</b>	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.
<b>Name</b>	exec
<b>Synonym</b>	e
<b>Arguments</b>	[path] [args]
<b>Description</b>	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).
<b>Name</b>	ldstart
<b>Synonym</b>	ld
<b>Arguments</b>	path [args]
<b>Description</b>	Load and start a module but do not save away the

	information. Can be used for loading modules from flash or just general use.
<b>Name</b>	kill
<b>Synonym</b>	k
<b>Arguments</b>	uid @name
<b>Description</b>	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.
<b>Name</b>	modexp
<b>Synonym</b>	mp
<b>Arguments</b>	uid @name
<b>Description</b>	Print the export libraries from a module with NID and address.
<b>Memory Commands (memory)</b>	
<b>Name</b>	meminfo
<b>Synonym</b>	mf
<b>Arguments</b>	[partitionid]
<b>Description</b>	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 for VSH applications. If partitionid is specified it will only print out the corresponding block information. Displays the amount of free memory in the system.
<b>Name</b>	memreg
<b>Synonym</b>	mr
<b>Arguments</b>	None
<b>Description</b>	Prints the accessible memory regions. These are the regions which PSPLINK consider safe to read and write to.
<b>Name</b>	memdump
<b>Synonym</b>	dm
<b>Arguments</b>	[- addr] [b h w]
<b>Description</b>	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 - 512 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.
<b>Name</b>	memblocks
<b>Synonym</b>	mk
<b>Arguments</b>	[f t]
<b>Description</b>	Print the system block tables to Kprintf. Without arguments prints a concise list. With 'f' print the full table of blocks. With 't' print the tail blocks.
<b>Name</b>	savemem
<b>Synonym</b>	sm
<b>Arguments</b>	addr size path
<b>Description</b>	Save the block of memory of size size specified to a path.
<b>Name</b>	loadmem
<b>Synonym</b>	lm
<b>Arguments</b>	addr path [maxsize]
<b>Description</b>	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.
<b>Name</b>	pokew
<b>Synonym</b>	pw
<b>Arguments</b>	addr val1 [val2..valN]
<b>Description</b>	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.
<b>Name</b>	pokeh
<b>Synonym</b>	ph
<b>Arguments</b>	addr val1 [val2..valN]
<b>Description</b>	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.
<b>Name</b>	pokeb
<b>Synonym</b>	pb
<b>Arguments</b>	addr val1 [val2..valN]
<b>Description</b>	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.
<b>Name</b>	peekw
<b>Synonym</b>	kw
<b>Arguments</b>	addr [o d x f]
<b>Description</b>	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.
<b>Name</b>	peekh
<b>Synonym</b>	kh
<b>Arguments</b>	addr [o d x]
<b>Description</b>	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.
<b>Name</b>	peekb
<b>Synonym</b>	kb
<b>Arguments</b>	addr [o d x]
<b>Description</b>	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.
<b>Name</b>	fillw
<b>Synonym</b>	fw
<b>Arguments</b>	addr size val
<b>Description</b>	Fills a region of memory with word values. Size indicates the number of words to write.
<b>Name</b>	fillh



<b>Synonym</b>	fh
<b>Arguments</b>	addr size val
<b>Description</b>	Fills a region of memory with half-word values. Size indicates the number of half-words to write.
<b>Name</b>	fillb
<b>Synonym</b>	fb
<b>Arguments</b>	addr size val
<b>Description</b>	Fills a region of memory with byte values. Size indicates the number of bytes to write.
<b>Name</b>	copymem
<b>Synonym</b>	cm
<b>Arguments</b>	srcaddr destaddr size
<b>Description</b>	Copys memory from one src to dest.
<b>Name</b>	findstr
<b>Synonym</b>	ns
<b>Arguments</b>	add size str
<b>Description</b>	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.
<b>Name</b>	findhex
<b>Synonym</b>	nx
<b>Arguments</b>	addr size hexstr [mask]
<b>Description</b>	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
<b>Name</b>	findw
<b>Synonym</b>	nw
<b>Arguments</b>	addr size val1 [val2..valN]
<b>Description</b>	Find a string of words (32bit) in memory. More than one value can be specified.
<b>Name</b>	findh

<b>Synonym</b>	nh
<b>Arguments</b>	addr size val1 [val2..valN]
<b>Description</b>	Find a string of half-words (16bit) in memory. More than one value can be specified.
<b>Name</b>	dache
<b>Synonym</b>	dc
<b>Arguments</b>	w i wi [addr size]
<b>Description</b>	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.
<b>Name</b>	icache
<b>Synonym</b>	ic
<b>Arguments</b>	[addr size]
<b>Description</b>	Invalidates the instruction cache of the MIPS processor. If addr and size are specified only that region of memory will be affected.
<b>Name</b>	disasm
<b>Synonym</b>	di
<b>Arguments</b>	addr [count]
<b>Description</b>	Disassemble instructions at addr, if count is specified it indicates the number of instructions to disassemble.
<b>FileIO Commands (fileio)</b>	
<b>Name</b>	ls
<b>Synonym</b>	dir
<b>Arguments</b>	[path1..pathN]
<b>Description</b>	Lists the files in a specified directory(s). If no paths are specified then will list the current directory.
<b>Name</b>	chdir
<b>Synonym</b>	cd
<b>Arguments</b>	dir
<b>Description</b>	Changes the current working directory of the PSPLINK

	shell.
<b>Name</b>	cp
<b>Synonym</b>	copy
<b>Arguments</b>	sourcepath destpath
<b>Description</b>	Copies a file from sourcepath to destpath.
<b>Name</b>	mkdir
<b>Synonym</b>	md
<b>Arguments</b>	dir
<b>Description</b>	Make a directory with the specified path.
<b>Name</b>	rm
<b>Synonym</b>	del
<b>Arguments</b>	path
<b>Description</b>	Deletes a file with the specified path.
<b>Name</b>	rmdir
<b>Synonym</b>	rd
<b>Arguments</b>	dir
<b>Description</b>	Removes a directory with the specified path.
<b>Name</b>	rename
<b>Synonym</b>	ren
<b>Arguments</b>	src dst
<b>Description</b>	Renames the file src to dst.
<b>Name</b>	pwd
<b>Synonym</b>	None
<b>Arguments</b>	None
<b>Description</b>	Print the current working directory.
<b>Debugger Commands (debugger)</b>	
<b>Name</b>	exprint
<b>Synonym</b>	ep
<b>Arguments</b>	None

<b>Description</b>	Print the last exception register information to screen.
<b>Name</b>	exresume
<b>Synonym</b>	c
<b>Arguments</b>	[addr]
<b>Description</b>	Resume from the last exception. If addr is specified sets this to epc and resumes from there.
<b>Name</b>	exprfpu
<b>Synonym</b>	ef
<b>Arguments</b>	None
<b>Description</b>	Print the last exceptions FPU registers.
<b>Name</b>	setreg
<b>Synonym</b>	str
<b>Arguments</b>	\$reg value
<b>Description</b>	Sets the last exceptions register \$reg to value.
<b>Name</b>	bpset
<b>Synonym</b>	bp
<b>Arguments</b>	addr
<b>Description</b>	Set a one-shot breakpoint at the specified address. Once the breakpoint has occurred it will be cleared.
<b>Name</b>	bpprint
<b>Synonym</b>	bt
<b>Arguments</b>	None
<b>Description</b>	Print the current list of breakpoints.
<b>Name</b>	step
<b>Synonym</b>	s (or CTRL+S for SIO shell)
<b>Arguments</b>	None
<b>Description</b>	Step a single MIPS instruction.
<b>Name</b>	skip
<b>Synonym</b>	k (or CTRL+K for SIO shell)
<b>Arguments</b>	None

<b>Description</b>	Step a single MIPS instruction, if the next instruction is jal or jalr then break immediately after the call.
<b>Name</b>	symload
<b>Synonym</b>	syl
<b>Arguments</b>	path
<b>Description</b>	Load symbols from the specified path.
<b>Name</b>	symlist
<b>Synonym</b>	syt
<b>Arguments</b>	None
<b>Description</b>	List the currently loaded symbol files.
<b>Name</b>	symprint
<b>Synonym</b>	syp
<b>Arguments</b>	modname
<b>Description</b>	List the symbols for the specified module.
<b>Name</b>	symbyaddr
<b>Synonym</b>	sya
<b>Arguments</b>	addr
<b>Description</b>	Print the symbol information at the specified address.
<b>Name</b>	symbyname
<b>Synonym</b>	syn
<b>Arguments</b>	module:symname
<b>Description</b>	Print the address of the symbol in the specified module with the symbol name.
<b>Miscellaneous Commands (misc)</b>	
<b>Name</b>	usbon
<b>Synonym</b>	un
<b>Arguments</b>	None
<b>Description</b>	Enable the USB mass storage device drivers.
<b>Name</b>	usboff

<b>Synonym</b>	uf
<b>Arguments</b>	None
<b>Description</b>	Disable the USB mass storage device drivers.
<b>Name</b>	usbstat
<b>Synonym</b>	us
<b>Arguments</b>	None
<b>Description</b>	Print the status of the USB mass storage device connection.
<b>Name</b>	uidlist
<b>Synonym</b>	ul
<b>Arguments</b>	[root]
<b>Description</b>	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, SceSystememMemoryPartition and SceSystememHeap.
<b>Name</b>	exit
<b>Synonym</b>	quit (or CTRL+D for SIO shell)
<b>Arguments</b>	None
<b>Description</b>	Exit the shell and return back to Sony's VSH.
<b>Name</b>	set
<b>Synonym</b>	None
<b>Arguments</b>	[var=value]
<b>Description</b>	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 only valuable currently available is prompt which sets the prompt string.
<b>Name</b>	scrshot
<b>Synonym</b>	ss
<b>Arguments</b>	path
<b>Description</b>	Take a screenshot of the current display and write it to a 24bit colour Windows bitmap file.

<b>Name</b>	run
<b>Synonym</b>	None
<b>Arguments</b>	path [args]
<b>Description</b>	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.
<b>Name</b>	calc
<b>Synonym</b>	None
<b>Arguments</b>	addr [d o x]
<b>Description</b>	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.
<b>Name</b>	reset
<b>Synonym</b>	r (or CTRL+R in SIO shell)
<b>Arguments</b>	None
<b>Description</b>	Reset PSPLINK.
<b>Name</b>	ver
<b>Synonym</b>	v
<b>Arguments</b>	None
<b>Description</b>	Print the version of PSPLINK.
<b>Name</b>	pspver
<b>Synonym</b>	None
<b>Arguments</b>	None
<b>Description</b>	Print the PSP version number as returned by sceKernelDevkitVersion
<b>Name</b>	help
<b>Synonym</b>	?
<b>Arguments</b>	[command category]
<b>Description</b>	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.

## 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 should be either a network address for the PSP (in wifi mode) or a serial port file (e.g. /dev/ttyS0) for serial mode.

As pterm 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.



## **4.Thanks and Greetz**

Thanks to Hitmen for providing the disassembler code.

Thanks to pspdev.org for hosting this stuff.