SN SYSTEMS
Sony Computer Entertainment Group

User Guide to

ProDG for PlayStation®3 Utilities

SN Systems Limited
Version 310.1
November 11, 2009

Copyright © Sony Computer Entertainment Inc. / SN Systems Ltd, 2003-2009.
"ProDG" is a registered trademark and the SN logo is a trademark of SN Systems Ltd.
"PlayStation" is a registered trademark of Sony Computer Entertainment Inc.
"Microsoft", "Visual Studio", "Win32" and "Windows" are registered trademarks of Microsoft Corporation. "GNU" is a trademark of the Free Software Foundation. Other product and company names mentioned herein may be the trademarks of their respective owners.

Contents

1: Introduction

Document version history

Ver.	Date	Changes
310.1	Nov. 2009	Bz78822: Updated 'Concise (-c, --concise)'. Bz78823: Added --disassemble-symbol switch and 'Disassemble address range (--disassemble-ranges)'. Bz78824: Added --compress-output. Bz78223: Updated 'Binary utilities command-line syntax'. Bz80381: Removed -dis or –disassemble switch.
300.1	Sept. 2009	Bz77912: Updated 'Binary utilities command-line syntax'.
280.1	May 2009	Bz72048: Updated 'Binary utilities command-line syntax' and 'Address to line (-a2l, --addr2line)'.

Overview of the build utilities

Build utilities are provided to assist with your game development. The following utilities are provided with the ProDG build tools suite.

ps3name	A command-line utility for demangling C++ names generated by SNC or GCC compilers for PlayStation®3.
ps3snarl.exe	The SN archive librarian. See "ps3snarl - SN Archive Librarian" on page 5.
ps3bin.exe	The SN binary utilities. See "ps3bin - SN binary utilities" on page 10.

ps3name - name demangler

ps3name is a command-line utility to demangle C++ names generated by the SNC or GCC compilers for the PlayStation®3. Mangled names are used by the compilers to encode type information into function names so as to guarantee type safe linkage. Ps3name decodes this type information into a human readable form.

The mangling schemes used by the SNC and GCC compilers for PlayStation®3 are similar, except that SNC's mangled names are typically prefixed with '_Q' and GCC's with '_Z'. A different prefix is used to prevent accidental cross linkage as SNC- and GCC-compiled C++ object modules are not link compatible.

The mangling scheme used by SNC should be considered an implementation detail of the compiler. You should not rely directly on it, or on the exact form of the output produced by ps3name.

Usage: `ps3name <options> <mangled-name>`

where <options> can be

`-h, --help`	Show help
`-s, --substitutions`	Show substitutions
`-v, --version`	Show version information

Substitutions are an internal implementation detail of the mangling scheme; this option is provided as an aid to debugging.

ps3snarl - SN Archive Librarian

The **SN AR**chive **L**ibrarian (snarl) follows the same usage as GNU ar.

You can find general information on controlling ar from the command line and from MRI scripts in the GNU documentation at www.gnu.org.

The following sections document some of the additional features that snarl offers.

Archive librarian command-line syntax

Snarl allows you to create, modify and extract from archives. In this context <archive> is usually a library, for example libc.a.

Usage:

```
ps3snarl [-]<key>[<mod> [<relpos>] [<count>]] <archive> [<files>]
[<symbols>]
```

```
ps3snarl <archive> @<file>    // use response file <file>
```

```
ps3snarl --help        // print usage info
```

where <key> must be one of the following...

d[NT]	Delete <files> from <archive>.
m[abNT]	Move <files> found in <archive>.
p[N]	Print <files> found in <archive>. See "Printing archive and object files" on page 6.
q[cfST]	Quick append <files> to <archive>. See "Super fast append" on page 7.
r[abucfST]	Replace existing or insert new <files> into <archive>.
s[GLWT]	Rebuild symbol table (ranlib) (performed by default).
t	Print table of contents for <files> in <archive>.
h[N]	Set mod times of <files> in <archive> to current time.
w[I]	Display <archive> symbol table. See "Displaying the symbol table" on page 6.
F[IN]	Display complete symbol table for <files>. See "Displaying the symbol table" on page 6.
x[oCN]	Extract <files> from <archive>.

...or one of the following symbol manipulation commands (see "Symbol manipulation commands" on page 8):

G	Make <symbols> global.

L	Make <symbols> local.
W	Make <symbols> weak.

...and the available 'mods' are:

a	Add <files> after <relpos> archive member.
b	Add <files> before <relpos> archive member.
c	Do not warn if <archive> had to be created.
C	Do not allow extracted files to overwrite existing files.
f	Truncate inserted filenames into 8.3 format.
l	Demangle C++ symbol names (if demangle.dll is available).
N	Specify instance <count> of same filename entries in <archive>.
o	Preserve original date.
S	Suppress building of symbol table (it is built by default).
T	Warn if symbols are multiply defined. See "Warning of multiply-defined symbols" on page 7.
u	Only replace archive members if <files> are newer.
v	Verbose mode. See "Verbose mode" on page 6.
V	Show version.
y	Do not create empty archives.

Verbose mode

Appending a 'v' onto most snarl keyletters will enable verbose mode. One of the most useful ways of using this is with the print table of contents ('`t`') command; this will force extended information to be displayed about the library's contents.

Printing archive and object files

Snarl can be used to create archives of any file type (obviously only object files will be included in the archive symbol table that is used at link time). So for example a text file containing the version history can easily be stored within the library. This could then be viewed by using the print archive files ('p') command, e.g.:

```
ps3snarl p test.lib versions.txt
```

Other implementations of library archivers do not handle **print archive files** ('p') very gracefully when object files are specified. Snarl will still display text-based files as usual, but will automatically switch to a formatted hex dump if it detects an object file. This is sent to stdout so it can be easily redirected to a file using the '>' DOS redirect command, e.g.:

```
ps3snarl p test.lib obj1.o obj2.o > out.txt
```

Displaying the symbol table

The 'w' keyletter dumps the archive symbol table to stdout. This shows which symbols are visible to the linker and what object file they reside in within the library. Demangling of C++ symbol names is supported.

The 'F' keyletter dumps out the entire object file symbol tables to stdout (from version 1.4.2.9). This enables all symbols in the library to be viewed along with their type and scope. It is recommended you pipe this output to a file as it can be quite substantial:

```
ps3snarl F test.lib > out.txt
```

Super fast append

If you are building a library in stages (i.e. not just adding all of the object files at once) then you can take advantage of the "Super Fast Append" feature. This is activated when you specify a quick append ('q') and suppress the building of the symbol table ('S'). This is only effective when working with large libraries (>20MB). It will provide no real benefit with smaller ones.

It works by appending the new file to the library without loading in the existing data. Note that to make the library functional, the symbol table must be built once you have completed all of the desired operations.

The following example shows how to speed up the appending of four object files to a large library:

Usual implementation:

```
ps3snarl q test.lib obj1.o obj2.o
...
ps3snarl q test.lib obj3.o
...
ps3snarl q test.lib obj4.o
```

Super Fast Append implementation:

```
ps3snarl qS test.lib obj1.o obj2.o
...
ps3snarl qS test.lib obj3.o
...
ps3snarl qS test.lib obj4.o
ps3snarl s test.lib            // rebuild symbol table
```

In the first example the time taken following each append is *N* (where *N* is some time delay proportional to the size of the existing library), making a total time of 3*N* seconds.

In the second example the time taken following each append is ≈0 seconds, and the time taken following the final rebuild of the symbol table is *N*, making a total time of *N* seconds.

Note: If any of the filenames of the object files to be added are more than 15 characters in length, then the fast append will be cancelled as the extended filename section will have to be rebuilt. You can get around this by using the 'f' modifier which will shorten all inserted filenames into 8.3 DOS format, e.g.: `ps3`snarl qSf test.lib "filename that is longer than 15 characters.obj".

Warning of multiply-defined symbols

The 'T' modifier warns if multiply-defined symbols are present. It can be simply added to any existing key command. For example to perform a quick append and warn for multiply-defined symbols use:

```
ps3snarl qT test.lib obj1.o obj2.o obj3.o obj4.o obj5.o
```

If you just want to list any multiply-defined symbols without actually manipulating the library, then you can specify 's' as your key command and 'T' as your modifier. This will rebuild the archive symbol table and warn (without changing the library contents). e.g.:

```
ps3snarl sT test.lib
```

This feature is automatically activated if you are building a library with the Visual Studio Integration, and any warnings are displayed in the Visual Studio build window.

Symbol manipulation commands

There are three Symbol Manipulation Commands: "Make Global" (G), "Make Weak"(W), and "Make Local"(L), which can be used to modify symbol properties within a library. For example, the following changes 'sym1' in test.lib to a weak symbol:

```
ps3snarl W test.lib sym1
```

You can specify several symbols on one command line, for example:

```
ps3snarl G test.1ib sym1 sym2 sym3
```

Note that if these commands are used on their own then they will just alter the symbol properties in the object file containing the symbol within the library. They will not update the archive symbol table, i.e. if you change a local symbol to a global then it will be not be visible to the linker until the archive symbol table is rebuilt. You can of course do this at the same time by specifying the 's' argument:

```
snarl Gs test.1ib sym1 sym2 sym3
```

Building cross-platform libraries

Snarl is capable of reading and creating libraries for most platforms (with the exception of Win32). It has been tested with PlayStation 2 and PlayStation 3.

Libraries can also be created that contain object files built for different platforms. For example, a library sky.a could contain the objects ps2sky.o and ps3sky.o. These could then contain functions such as PS2_getskycoords() and PS3_getskycoords() respectively, making library maintenance in a cross-platform game easier to manage.

Response file scripting

Snarl supports simple response files, for example:

```
ps3snarl lib.a @response.txt
```

where response.txt is a text file in the format

```
object1.o
object2.o
object3.o
object4.o
```
[etc.]

Note that using a response file will automatically delete any existing archive of the same name and build a new one (appending is not possible). If you want to do anything more complex, use MRI scripting (see "MRI scripting" on page 8).

MRI scripting

You can control the operation of snarl with a simple command language, by invoking snarl -M from the command line. Snarl will then prompt for input, using the prompt 'SNARL >'. The snarl command language provides less control than the command-line options, but can be helpful for developers who have scripts written in MRI "librarian" format.

The -E command forces files with filename extension '.sns' to be associated with snarl:

```
ps3snarl -E     // force .sns files to be associated with snarl
```

MRI scripting commands must be formatted as follows:

- one command per line
- case is not significant
- any text after either '*' or ';' is treated as a comment
- command arguments can be separated by either commas or spaces
- the continuation character '+' causes text on the following line to be treated as part of the current command

The following table lists the commands which can be used with snarl MRI scripts:

Command	Function
ADDLIB archive ADDLIB archive (module, module, ... module)	Add all the contents of archive (or, if specified, each named module from archive) to the current archive. Requires prior use of OPEN or CREATE.
ADDMOD member, member, ... member	Add each named member as a module in the current archive. Requires prior use of OPEN or CREATE.
CLEAR	Discard the contents of the current archive, canceling the effect of any operations since the last SAVE. May be executed (with no effect) even if no current archive is specified.
CREATE archive	Creates an archive, and makes it the current archive (required for many other commands). The new archive is created with a temporary name; it is not actually saved as an archive until you use SAVE. You can overwrite existing archives; similarly, the contents of any existing file named will not be destroyed until SAVE.
DELETE module, module, ... module	Delete each listed module from the current archive; equivalent to 'snarl d archive module ... module'. Requires prior use of OPEN or CREATE.
DIRECTORY archive (module, ... module) DIRECTORY archive (module, ... module) outputfile	List each named module present in archive. The separate command VERBOSE specifies the form of the output: when verbose output is off, output is like that of 'snarl t archive module...'. When verbose output is on, the listing is like 'snarl tv archive module...'. Output normally goes to the standard output stream; however, if you specify outputfile as a final argument, snarl directs the output to that file.
END	Exit from snarl, with a 0 exit code to indicate successful completion. This command does not save the output file; if you have changed the current archive since the last SAVE command, those changes are lost.
EXTRACT module, module, ... module	Extract each named module from the current archive, writing them into the current directory as separate files. Equivalent to 'snarl x archive module...'. Requires prior use of OPEN or CREATE.
GLOBAL symbol, symbol, ... symbol	Make each listed symbol from the current archive global; equivalent to 'snarl G archive symbol ... symbol'. Requires prior use of OPEN or CREATE.
LIST	Display full contents of the current archive, in "verbose" style regardless of the state of VERBOSE. The effect is like 'snarl tv archive'. Requires prior use of OPEN or CREATE.
LOCAL symbol, symbol, ... symbol	Make each listed symbol from the current archive local; equivalent to 'snarl L archive symbol ... symbol'. Requires prior use of OPEN or CREATE.

OPEN archive	Opens an existing archive for use as the current archive (required for many other commands). Any changes as the result of subsequent commands will not actually affect archive until you next use SAVE.
REPLACE module, module, ... module	In the current archive, replace each existing module (named in the REPLACE arguments) from files in the current working directory. To execute this command without errors, both the file, and the module in the current archive, must exist. Requires prior use of OPEN or CREATE.
SAVE	Commit your changes to the current archive, and actually save it as a file with the name specified in the last CREATE or OPEN command. Requires prior use of OPEN or CREATE.
VERBOSE	Toggle an internal flag governing the output from DIRECTORY. When the flag is on, DIRECTORY output matches output from 'snarl tv '....
WEAK symbol, symbol, ... symbol	Make each listed symbol from the current archive weak; equivalent to 'snarl W archive symbol ... symbol'. Requires prior use of OPEN or CREATE.
$(ENV)	Environment variable macro expansion. Any macros used in this format will be expanded into the value of the specified environment variable upon execution, e.g.: open $(LIB_DIR)\lib.a.

To run MRI scripts from the command prompt

```
ps3snarl -M [<mri-script>]
ps3snarl -M?                 // display MRI script commands
```

In this case ps3snarl will takes its MRI commands from the MRI script file <mri-script>. The default filename extension for ps3snarl scripts is .sns.

To run MRI scripts from Windows Explorer

1. Enable file associations with ps3snarl and the .sns (snarl script) extension, using the command 'ps3snarl -E'.
2. Save all scripts in plain text with the .sns filename extension.
3. Double-click the saved script file to automatically execute it.

ps3bin - SN binary utilities

The SN binary utilities program ps3bin.exe is a tool for manipulating ELF files and library/archive files.

Features include: stripping of sections, symbols and debug data; dumping of section headers, symbol tables and program headers; copying sections to a binary file; and renaming sections.

Binary utilities command-line syntax

Usage:

```
ps3bin -i <input> <options> -o <output>
                   // short form
```

```
ps3bin --infile=<input> <options> --outfile=<output>
                   // long form
```

```
ps3bin @<file>    // use response file <file>
```

where <input> and <output> can be ELF files or library/archive files, and where <options> can be any of the following:

Exclusive options	
-a2l <address> or --addr2line=<address>	Get source file correspondence for an address <address>. See "Address to line (-a2l, --addr2line)" on page 13.
-b2e <file> or --bin2elf=<file>	Convert a binary file <file> to an ELF object file. See "Binary to ELF file (-b2e, --bin2elf)" on page 13.
-be or --blank-elf	Code blank an ELF file. See "Blank ELF (-be, --blank-elf)" on page 14.
-cs <sect> or --copy-section=<sect>	Copy section(s) to a binary file. See "Copy section (-cs, --copy-section)" on page 14.
-d or --disassemble	Disassemble all executable code.
-D or --disassemble-all	Disassemble all code.
--disassemble-ranges=<range>	Disassemble code within a range of addresses. See "Disassemble address range (--disassemble-ranges)" on page 15.
--disassemble-symbol=<symbol>	Disassemble code for supplied function symbol <symbol>. See "Disassemble symbol (--disassemble-symbol" on page 15.
-dd or --dump-debug-data	Dump the debug data.
-de or --dump-everything	Dump everything.
-dem <symb> or --demangle=<symb>	Demangle symbol <symb>. See "Demangle (-dem, --demangle)" on page 14.
-dh or --dump-elf-header	Dump the ELF header. See "Dump ELF header (-dh, --dump-elf-header)" on page 15.
-dml or --dump-mem-layout	Dump a virtual memory layout.
-dph or --dump-program-headers	Dump program headers.
-ds <sect>,<sect>,... or --dump-sections=<sect>,<sect>,...	Dump the specified sections. See "Dump sections (-ds, --dump-sections)" on page 16.
-dsh or --dump-section-headers	Dump section headers.
-dsi or --dump-sizes	Dump size statistics. See "Dump sizes (-dsi, --dump-sizes)" on page 16.
-dss or --dump-stack-sizes	Dump the sizes of stack frames for functions.
-dsy or --dump-symbols	Dump symbol tables.
-nd or --no-demangle	Do not demangle C++ symbol names.
-rs <old> <new> or --rename-sections=<old>,<new>	Rename section <old> to <new>. See "Rename sections (-rs, --rename-sections)" on page 17.
-s or --sort-data	Sort output (currently only supported by the -dsy option).

-sa or --strip-all	Strip all symbol and debug information.
-sd or --strip-debug	Strip all debug data.
-sse <sect>,<sect>,... or --strip-sections=<sect>,<sect>,...	Strip section(s). See "Strip sections (-sse, --strip-sections)" on page 17.
-ssy <symb>,<symb>,... or --strip-symbols=<symb>,<symb>,...	Strip symbol(s). See "Strip symbols (-ssy, --strip-symbols)" on page 17.
Non-exclusive options	
@<file>	Read switches from a response file <file>. Any portion of a command line can be read in from file <file>.
-c or --concise	Output a minimal amount of information. See "Concise (-c, --concise)" on page 14.
--compress-output	Compresses FSELF output file. Must be used with -of fself or --oformat=fself to have any effect.
-g <string> or --grep=<string>	Only print lines containing <string>.
-gnu or --gnu-mode	GNU mode. This option formats the output of the operation to match the style of the GNU binutil output (currently only supported for addr2line).
-i <file> or --infile=<file>	Specify an input file.
-nd	No demangle.
-o <file> or --outfile=<file>	Specify an output file.
-of fself or --oformat=fself	Output file will be fake signed, removing the requirement of running the output ELF file through the make_fself tool.
-p or --page-output	Pause between screenfuls of information.
-v or --verbose	Output all available information. See "Verbose (-v, --verbose)" on page 17.
-ver or --version	Display version number.

Only one input and one output file can be specified on the command line. If no input file is specified, then the first unrecognised argument is taken to be the name of the input file. Long filenames containing spaces must be delimited by quotation marks, e.g. '--infile="C:\my long filename.elf".

The command-line options can be specified in any order, and in either a long form, or a shortened form to save time when using them from a command line. When using a short-form equivalent of a long-form option that takes an argument, then the argument can be specified after the short-form option followed by a space. For example, '--rename-section=.sect' is equivalent to '-rs .sect'.

Some options can take multiple arguments in a comma-delimited list (e.g. --dump-sections=<sect>,<sect>,...') and there is no limit on the number of arguments that can be passed in this way.

Exclusive options cannot be mixed on a command line with any other exclusive option, whereas the *non-exclusive options* can be mixed with other options.

All options work for all input file types.

Address to line (-a2l, --addr2line)

The -a2l (--addr2line) option parses the debug data to try and work out the source file correspondence. Example:

```
>ps3bin -a2l 0x10000 debug.elf

Address:        0x00010000
Directory:      V:/Build Tools/Binutil Testing/TestSuite/
File Name:      test.c
Line Number:    4
Symbol          foo(int, float)
```

The address can be entered in hexadecimal or decimal. Hex numbers must be prefixed with '0x' or affixed with 'h'. If ps3bin can find the source file then it will also attempt to print the source line.

`ps3bin` allows addresses to be given for addr2line via stdin if no address is given via the command line. Example:

```
>ps3bin -a2l -i debug.elf -gnu
>0x10000

Address:      0x00010000
Directory:    V:/Build Tools/Binutil Testing/TestSuite/
File Name:    test.c
Line Number:  4
Symbol        foo(int, float)

>0x18000

Address:      0x00018000
Directory:    V:/Build Tools/Binutil Testing/TestSuite/
File Name:    test.c
Line Number:  4
Symbol        bar(int)
```

ps3bin now allows the addr2line information to be displayed in the same format as the GNU addr2line tool by specifying -gnu or --gnu-mode on the command line.

Example:

```
>ps3bin -a2l 0x10000 debug.elf -gnu
foo(int, float)

C:\example\file.c:110
```

Binary to ELF file (-b2e, --bin2elf)

The -b2e (--bin2elf) option converts a binary file into an ELF object file. Example:

```
>ps3bin -i my_resource_file -b2e PS3PPU,my_start_label,my_size_label -o
my_object_file.o
```

The first argument is ELF type and must be one of the following: "PSP", "PS3SPU", "PS3PPU" and "NGC".

This generated object file can then be linked into the project and the resource can be referenced by the label supplied to the -b2e option.

The -b2e (--bin2elf) option accepts two optional additional arguments which are specified after the size label and separated by a comma. These are an alignment value and an end label.

For example, if we wish to convert a binary file into an ELF object file and specify an alignment value of 16 then you would do the following:

```
>ps3bin -i my_resource_file -b2e PS3PPU,my_start_label,my_size_label,16 -o
my_object_file.o
```

If you wished to also specify an end label then you would do the following:

```
>ps3bin -i my_resource_file -b2e
PS3PPU,my_start_label,my_size_label,16,my_end_label -o my_object_file.o
```

It is not possible to specify an end label without specifying an alignment value.

Blank ELF (-be, --blank-elf)

The -be (--blank-elf) option blanks all code sections in an ELF file.

The command line syntax is:

```
>ps3bin -be <input file> -o <output file>
```

Concise (-c, --concise)

Some of the dump operations output a lot of data when run. This option cuts the output down but at the loss of information.

Currently the only options that support this are the 'dump symbol table', 'demangle' and 'disassembly output' options. To use this option just add it to the end of an existing operations command line. For example:

```
>ps3bin -dsy test.prx -c

0x00000000 f 0x0000 crt0.c
0x00000000 f 0x0000 crt0mark.c
           u 0x0008 __PSPEXP__module_start
           u 0x0008 __PSPREN__module_start__start
           u 0x0008 __PSPEXP__module_stop
           u 0x0008 __PSPREN__module_stop__stop
           u 0x0008 __PSPEXP__module_start_thread_parameter
0x00000000 f 0x0000 kernel_bridge.c
0x00000D3C t 0x02F8 init_all
0x00001098 t 0x0304 pad_read
0x00001450 d 0x0000 DATA.
0x00000050 r 0x0000 RDATA.
0x00001450 d 0x0480 cube_data
0x00000000 f 0x0000 libgu.c
0x00000050 r 0x0018 __psp_libinfo__
0x00000068 r 0x0378 initList
0x000003E0 r 0x0010 g_ListOptImmediate
0x000109F4 b 0x0400 g_SignalCallStack
0x000003F0 r 0x0040 dither.0
0x00010490 b 0x0030 intrParam
```

This symbol table dump attempts to emulate the minimal syntax of the GNU NM tool. See http://www.gnu.org/software/binutils/manual/html_chapter/binutils_2.html for more information.

Pipeline analysis information that is calculated and printed alongside disassembly output can be disabled with the -c, --concise switch. This can improve the disassembly output times by up to 40%. This will affect all switches that perform disassembly output.

Copy section (-cs, --copy-section)

The -cs (--copy-section) option copies a binary section to an output file. This is often used when using overlays. To use this option you must specify an output file name, a section name, and an input file. The command line syntax is:

```
>ps3bin -cs .text test.o -o new.bin
```

Demangle (-dem, --demangle)

The -dem (--demangle) option demangles C++ names. Example:

```
>ps3bin --demangle=__0fEfredEblahi

Input:      __0fEfredEblahi
Demangled: fred::blah(int)
```

Disassemble address range (--disassemble-ranges)

The --disassemble-ranges option (no short form) will disassemble code within a range of addresses where range is in the format <low_address..high_address>.

```
>ps3bin --disassemble-ranges=0x010250..0x010300,0x020000..0x030000
```

This will print a portion of the disassembly where the program address range lies between 0x010250 and 0x010300 and between 0x020000 and 0x030000.

Disassemble symbol (--disassemble-symbol)

The --disassemble-symbol option will disassemble code for a supplied function symbol. Example:

```
>ps3bin --disassemble-symbol=.my_function_symbol(int, float)
```

Use the -nd flag if disassembling mangled symbol names. Example:

```
>ps3bin --disassemble-symbol=._Z18my_function_symbolif -nd
```

Dump ELF header (-dh, --dump-elf-header)

The -dh (--dump-elf-header) option dumps the ELF header for the file. Example:

```
>ps3bin -dh test.o

ELF header:
  Magic Number:       0x7F ELF
  File Class:         ELFCLASS32
  Data Encoding:      ELFDATA2LSB
  ELF Header Version: 0x01
  Type:               ET_REL
  Machine:            EM_MIPS
  Version:            EV_CURRENT
  Entry point:        0x00000000 | Program Hdr Offset:
0x00000000
  Section Hdr Offset: 0x000001D4 | Flags:
0x10A23001
  ELF Header Size:    0x00000034 | Program Hdr Size:
0x00000000
  Num Prog Hdrs:      0x00000000 | Section Hdr Size:
0x00000028
  Num Section Hdrs:   0x00000009 | Section Hdr Strings:
0x00000006
```

Dump memory layout (-dml, --dump-mem-layout)

The -dml (--dump-mem-layout) option prints out virtual address view of the sections in the ELF file.

Example:

```
>ps3bin -dml debug.elf

Program header 0 : 0x00000000 - 0x0000EAA8
0x00000000 - 0x0000B970 = .text
0x0000B970 - 0x0000B9C0 = .sceStub.text.sceGe_user
0x0000B9C0 - 0x0000B9D8 = .sceStub.text.sceDisplay
0x0000B9D8 - 0x0000B9E0 = .sceStub.text.sceCTRL
0x0000B9E0 - 0x0000B9F8 = .sceStub.text.UtilsForUser
0x0000B9F8 - 0x0000BA38 = .sceStub.text.ThreadManForUser
```

```
0x0000BA38 - 0x0000BA68 = .sceStub.text.SysMemUserForUser
0x0000BA68 - 0x0000BA80 = .sceStub.text.StdioForUser
0x0000BA80 - 0x0000BAA8 = .sceStub.text.ModuleMgrForUser
0x0000BAA8 - 0x0000BAB8 = .sceStub.text.Kernel_Library
0x0000BAB8 - 0x0000BAE8 = .sceStub.text.IoFileMgrForUser
0x0000BAE8 - 0x0000BAEC = .lib.ent.top
0x0000BAEC - 0x0000BAFC = .lib.ent
0x0000BAFC - 0x0000BB00 = .lib.ent.btm
0x0000BB00 - 0x0000BB04 = .lib.stub.top
0x0000BB04 - 0x0000BBCC = .lib.stub
0x0000BBCC - 0x0000BBD0 = .lib.stub.btm
0x0000BBD0 - 0x0000BC04 = .rodata.sceModuleInfo
0x0000BC04 - 0x0000BCF4 = .rodata.sceResident
0x0000BCF4 - 0x0000BDB0 = .rodata.sceNid
0x0000BDB0 - 0x0000C610 = .rodata
0x0000C610 - 0x0000EA90 = .data
0x0000EA90 - 0x0000EA98 = .cplinit
0x0000EA98 - 0x0000EAA0 = .ctors
0x0000EAA0 - 0x0000EAA8 = .dtors

Program header 1 : 0x0000EAC0 - 0x00027924

0x0000EAC0 - 0x00027924 = .bss
```

Dump sections (-ds, --dump-sections)

The -ds (--dump-sections) option prints the contents of the specified section(s). The program decodes the sections and prints them in a human readable format. To use this option, a section name must be specified as well as an input file. The command line syntax is:

```
>ps3bin -ds .strtab test.o

.strtab:
Type:         SHT_STRTAB
Flags:        None
Address:      0x00000000 | Offset:      0x00000144
Size:         0x0000002E | Link:        0x00000000
Info:         0x00000000 | Align:       0x00000001
Entry Size:   0x00000000

0x00000001 - DATA
0x00000007 - RDATA
0x0000000E - SDATA
0x00000015 - a
0x00000017 - b
0x00000019 - c
0x0000001B - d
0x0000001D - e
0x0000001F - f
0x00000021 - x
0x00000023 - main
0x00000028 - _main
```

If the application cannot decode the section then a hex dump will be displayed instead. Multiple sections can be specified by using a comma as a separator.

Dump sizes (-dsi, --dump-sizes)

The -dsi (--dump-sizes) option prints out the sizes of the various components of the input file.

Example:

```
>ps3bin -dsi debug.elf
```

```
Text Size   Data Size   Debug Size   BSS Size   Total   Filename
252         1124        467          0          1843    debug.elf
```

Rename sections (-rs, --rename-sections)

The -rs (--rename-sections) option renames a section. To use this option you must specify the old section name, a new section name, an input file and an output file name.

The command line syntax is:

```
>ps3bin -rs my_old_section_name my_new_section_name test.elf -o new.elf
```

Strip sections (-sse, --strip-sections)

The -sse (--strip-sections) option removes a section from an input file. To use this option you must specify a section name, an input file, and an output file name. The command line syntax is:

```
>ps3bin -sse .data test.o -o new.o
```

You can specify multiple sections to remove. These must be comma separated. For example:

```
>ps3bin -sse .data,.text,.symtab ...
```

Strip symbols (-ssy, --strip-symbols)

The -ssy (--strip-symbols) option removes a symbol from an input file. To use this option you must specify a symbol name, an input file and an output file name. The command line syntax is:

```
>ps3bin -ssy main test.o -o new.o
```

You can specify multiple symbols to remove. These must be comma separated. For example:

```
>ps3bin -ssy main,exit,printf,hello_world
```

Verbose (-v, --verbose)

Some of the dump operations only output a minimal set of data when run. This is to help readability. If you desire a more thorough output then the -v (--verbose) option can be used. Currently the only options that support this are the 'dump symbol table' and the 'dump section header' options. To use this option just add it to the end of an existing operations command line. For example:

```
>ps3bin -dsh test.o

Index Name      Size  Type         Address
0     SHN_UNDEF (0)   SHT_NULL     0x00000000
1     .text     88    SHT_PROGBITS 0x00000000
2     .rodata    0    SHT_PROGBITS 0x00000000
3     .data      0    SHT_PROGBITS 0x00000000
4     .sdata     0    SHT_PROGBITS 0x00000000
5     .symtab  272    SHT_SYMTAB   0x00000000
6     .strtab   46    SHT_STRTAB   0x00000000
7     .shstrtab 73    SHT_STRTAB   0x00000000
8     .reginfo  24    Unknown type 0x00000000
9     .rel.text 64    SHT_REL      0x00000000

>ps3bin -dsh test.o -v

test.o - Section headers:

0 - SHN_UNDEF:
  Type:         SHT_NULL
  Flags:        None
  Address:      0x00000000 | Offset:      0x00000000
  Size:         0x00000000 | Link:        0x00000000
  Info:         0x00000000 | Align:       0x00000000
  Entry Size:   0x00000000

1 - .text:

  Type:         SHT_PROGBITS
  Flags:        SHF_WRITE, SHF_ALLOC, SHF_EXECINSTR
  Address:      0x00000000 | Offset:      0x00000178
  Size:         0x00000058 | Link:        0x00000000
  Info:         0x00000000 | Align:       0x00000008
  Entry Size:   0x00000000

< Output truncated >
```

GNU mode (-gnu, --gnu-mode)

The -gnu (--gnu-mode) option formats the output of the operation to match the style of the gnu binutil output. This option is currently only supported for addr2line.

Example:

```
 >ps3bin -a2l 0x10000 debug.elf -gnu

foo(int, float)

C:\example\file.c:110
```

2: Index