User Guide to

Build Utilities for PlayStation®3

SN Systems Limited
Version v470.1
February 27, 2015

Contents

1: Introduction

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

`ppu-lv2-prx-fixup.exe`	Performs relocations and patches up debug entries. This contains information about loading the IRX file and running it. It only has one input, the partially linked ELF object file. See "ppu-lv2-prx-fixup".
`ppu-lv2-prx-libgen.exe`	Used for generating a stub library of a PRX. See "ppu-lv2-prx-libgen".
`ps3name.exe`	A command-line utility for demangling C++ symbol names generated by SNC or GCC compilers for PlayStation®3. See "ps3name - name demangler".
`ps3snarl.exe`	The SN archive librarian. See "ps3snarl - SN Archive Librarian".
`ps3bin.exe`	The SN binary utilities. See "snps3bin - SN binary utilities".

2: ppu-lv2-prx-fixup

The ppu-lv2-prx-fixup utility performs relocations, patching up the debug entries, adding the symbols needed for PS3 PRX execution, PRX linkage, and segment creation. This section contains information for loading the PRX file and running it. The program only has one input, which is the partially linked ELF object file.

Usage: `ppu-lv2-prx-fixup <options> <outputfile> <inputfile>`

where *<options>* can be:

```
-v                    Verbose mode
-o <outputfile>       Convert without debug symbols
-r <outputfile>       Convert with debug symbol info
-e entry_symbol       Entry point specification symbol
-m <.prx file>        Display module name and version number
```

3: ppu-lv2-prx-libgen

The ppu-lv2-prx-libgen utility creates ILB files and library stubs.

Usage: `ppu-lv2-prx-libgen <inputfile(.tbl)> <options>`

where *<options>* can be:

```
-e <entry_table_source(.s)>
-d <stub_ilb_data(.ilb)>
-s <stub_source>
```

To create the entry table source for the TBL file the syntax is as follows:

```
ppu-lv2-prx-libgen <inputfile(.tbl)> -e <entry_table_source(.s)>
```

To create the ILB calling stub for the library defined in the TBL file the syntax is as follows:

```
ppu-lv2-prx-libgen <inputfile(.tbl)> -d <stub_ilb_data(.ilb)>
```

4: ps3name - name demangler

The name demangler PS3Name is a command-line utility to demangle C++ symbol names generated by the SNC or GCC compilers for PlayStation®3.

Mangled names are used by the compiler to encode type information into function names so as to guarantee type-safe linkage. The name demangler decodes this type information into a human readable form.

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.

The name demangler can operate in three different modes:

(1) If a list of mangled names is provided on the command line, PS3Name will print the demangled version of each in the order in which they were specified.

(2) If `--file` is specified, PS3Name will interpret the parameters as file names, and will attempt to open each file in turn and print their contents, with any mangled names replaced with their demangled counterparts.

(3) If no parameters are specified, PS3Name will read mangled names from standard input.

Name Demangler command-line syntax

Syntax

```
ps3name options
ps3name mangled_names
ps3name --file filenames
```

Parameters

options

Specify one of the following:

`-h` `--help`	Show help
`-v` `--version`	Show version information

mangled_names

Provide a list of mangled names to be demangled.

`--file`

Interpret *filenames* as a list of files containing mangled names to demangle. See "Demangle File Mode (-f, --file)".

Demangle File Mode (-f, --file)

Interpret provided parameters as file names, and print the file contents with any mangled names replaced with their demangled counterparts.

Syntax

```
ps3name --file filenames
```

Arguments

filenames

The operation will be performed upon the specified files.

Example

File *input.txt* contains the following text:

```
The functions in question are _Z3fooi and _Z3barii (but beware writing punctuation
next to mangled names, as in _Z3barii).
```

Run the name demangler with the `--file` parameter:

```
>ps3name --file input.txt

The functions in question are foo(int) and bar(int, int) (but beware writing
punctuation next to mangled names, as in _Z3barii).
```

5: ps3snarl - SN Archive Librarian

The **SN ARchive Librarian** (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 [switches...][-]<key>[<mod> [<relpos>] [<count>]] <archive> [<files>]
[<symbols>]
ps3snarl <archive> @<file>  // use response file <file>
ps3snarl -M [mri-script]
ps3snarl -M [<mri-script]   // use MRI scripting, taking commands from
                            // file [mri-script]
```

Where the available '*switches*' are:

`--disable-warning=`*<number>*`[,`*<number>*`[...]]`	Disable the warning referenced by *<number>*.
`--version`	Show version.
`--help`	Display this help text.

and *<key>* must be one of the following...

`d[ND]`	Delete *<files>* from *<archive>*.
`m[abND]`	Move *<files>* found in *<archive>*.
`p[N]`	Print *<files>* found in *<archive>*. See "Printing archive and object files".
`q[cfSDT]`	Quick append *<files>* to *<archive>*. See "Super fast append".
`r[abucfSDT]`	Replace existing or insert new *<files>* into *<archive>*.
`s[GLWD]`	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[l]`	Display *<archive>* symbol table. See "Displaying the symbol table".
`x[oCN]`	Extract *<files>* from *<archive>*.

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

`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.
D	Warn if symbols are multiply defined. See "Warning of multiply-defined symbols".
f	Truncate inserted filenames to less than 16 characters.
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	Output *<archive>* as a thin archive. See **Thin archives**.
u	Only replace archive members if *<files>* are newer.
v	Verbose mode. See "Verbose mode".
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.

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 $3N$ 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.

Warning: 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 truncate all inserted filenames to less than 16 characters, e.g.: `ps3snarl qSf test.lib "filename that is longer than 15 characters.obj"`.

Warning of multiply-defined symbols

The 'D' 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 qD 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 'D' as your modifier. This will rebuild the archive symbol table and warn (without changing the library contents). e.g.:

```
ps3snarl sD 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.

Thin archives

Thin archives store references to the members of the archive, rather than a copy of the binary data itself. This feature is useful when performing local builds, where relocatable object files are readily available for the final link phase, and will save on the time and space overhead it would take to construct the archive with the actual object file data.

Thin archives are created by using the Thin archive ('T') modifier in conjunction with the Quick append ('q') or Replace ('r') key commands.

> **Caution**: If the archive already exists, you must specify the appropriate archive type. An error will occur if a thin archive is used without the 'T' modifier, or a normal archive is used with the 'T' modifier.

Thin archives can be used as the input file for keys that do not output an archive file, such as Print Files ('p') and Print Table of Contents ('t').

Thin archives can also be created in an MRI script by specifying the `CREATETHIN` command. See "MRI Scripting".

> **Caution**: Thin archives are treated the same as regular archives, however the following MRI commands are not currently supported: `EXTRACT`, `GLOBAL`, `LOCAL`, `WEAK` , and `ADDLIB`.

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:

```
ps3snarl 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.]

This will create an archive "lib.a" containing the modules listed in the response file. The response file should be delimited by new lines, and blank lines will be ignored.

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").

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.

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* `CREATETHIN` *archive*	Creates an archive (or thin 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 <mri-script            // 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.

6: ps3bin - SN binary utilities

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

Syntax

Short form:

```
ps3bin -i input_file options [-o output_file]
```

Long form:

```
ps3bin --infile=input_file options [--outfile=output_file]
```

Response file:

```
ps3bin @input_file
```

Parameters

input_file

The input file. This can be an ELF or library file. You can read switches from a response file using the *@*command. Any portion of a command line can be read in from a response file.

output_file

The output file. You must specify an output file when the type differs from the input file type.

options

One or more user specified commands plus any required modifiers which will be applied to each command. The order of command execution is from left to right.

Commands	
`-a2l` `--addr2line`	Get source file correspondence for an address. See "Address to line (-a2l, --addr2line)".
`-b2e` `--bin2elf`	Convert a binary file to an ELF object file. See "Binary to ELF file (-b2e, --bin2elf)".
`-be` `--blank-elf`	Code blank an ELF file. See "Blank ELF (-be, --blank-elf)".
`-cs` `--copy-section`	Copy section(s) to a binary file. See "Copy section (-cs, --copy-section)".
`-d` `--disassemble`	Disassemble all executable code.
`--disassemble-ranges`	Disassemble code within a range of addresses. See "Disassemble address range (--disassemble-ranges)".
`--disassemble-symbol`	Disassemble code for supplied function symbol. See "Disassemble symbol (--disassemble-symbol".
`-dd` `--dump-debug-data`	Dump the debug data.
`-de`	Dump everything.

`--dump-everything`	
`-dem` `--demangle`	Demangle symbol. See "Demangle (-dem, --demangle)".
`-dh` `--dump-elf-header`	Dump the ELF header.
`-dml` `--dump-mem-layout`	Dump a virtual memory layout.
`-dph` `--dump-program-headers`	Dump program headers.
`-ds` `--dump-sections`	Dump the specified sections. See "Dump sections (-ds, --dump-sections)".
`-dsh` `--dump-section-headers`	Dump section headers.
`-dsi` `--dump-sizes`	Dump size statistics. See "Dump sizes (-dsi, --dump-sizes)".
`-dss` `--dump-stack-sizes`	Dump the sizes of stack frames for functions.
`-dsy` `--dump-symbols`	Dump symbol tables. See "Dump symbol tables (-dsy, --dump-symbols)".
`-G` `--globalize-symbol=NAME`	Mark symbol as global.
`--help`	Print command line help.
`-L` `--localize-symbol=NAME`	Mark symbol as local.
`-nd` `--no-demangle`	Do not demangle C++ symbol names.
`-r` `--reloc`	Display relocation information. See "Relocation Information (-r, --reloc)".
`-rs` `--rename-sections`	Rename sections. See "Rename sections (-rs, --rename-sections)".
`-sa` `--strip-all`	Strip all symbol and debug information.
`-sd` `--strip-debug`	Strip all debug data.
`-sse` `--strip-sections`	Strip section(s). See "Strip sections (-sse, --strip-sections)".
`-ssy` `--strip-symbols`	Strip symbol(s). See "Strip symbols (-ssy, --strip-symbols)".
`-W` `--weaken-symbol=NAME`	Mark symbol as weak.

Modifiers

`-c` `--concise`	Output a minimal amount of information. See "Concise (-c, --concise)".
`--disable-all-warnings`	Disables all warning messages.
`-gnu` `--gnu-mode`	GNU style formatted output (currently addr2line only). See "Address to line (-a2l, --addr2line)".
`-g`	Only print lines containing specified string. See "Grep (-g, --grep)".

`--grep`	
`-nd`	No demangle.
`-of fself` `--oformat=fself`	Output file will be fake signed, removing the requirement of running the output ELF file through the make_fself tool. See "Output fake signed file (-of fself, --oformat=fself)".
`-p` `--page-output`	Pause between screenfuls of information.
`-s, --display-source`	Displays source code within disassembly output. See "Display source code with disassembly (-S, --display-source)".
`-v` `--verbose`	Output all available information. See "Verbose (-v, --verbose)".
`-ver` `--version`	Display version number.
`-x` `--hex`	Perform all section dumps as plain hex dumps.

Remarks

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, for example, --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' is equivalent to '-rs'.

Some options can take multiple arguments in a comma-delimited list (for example, --dump-sections) and there is no limit on the number of arguments that can be passed in this way.

All options work for all input file types.

Address to line (-a2l, --addr2line)

Determines the corresponding line in the source files for a given address.

Syntax

```
ps3bin -i input_file -a2l [address] [-gnu] [-nd]
```

Arguments

input_file

The ELF file from which the (`--addr2line`) option will try and work out the source file correspondence.

address

The address can be entered in hexadecimal or decimal. Hex numbers must be prefixed with '`0x`' or suffixed with '`h`'.

If omitted, addresses must be specified via stdin.

`-gnu`

If `-gnu` or `--gnu-mode` are specified on the command line, the output will be displayed in the same format as the GNU addr2line tool.

`-nd`

Specify to output mangled symbol names. Can also be specified as *`--no-demangle`*.

Examples

Example 1:

```
>ps3bin –i test.elf -a2l 0x10000

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

The `-a2l` (`--addr2line`) option parses test.elf to determine the source file correspondence for the given address.

Example 2:

```
>ps3bin -i test.elf -a2l
>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)
```

No address given via the command line but an address is given via stdin.

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

Converts a binary file into an ELF object file. The generated object file can then be linked into the project and the resource can be referenced by the labels supplied to the `-b2e` option.

Syntax

`ps3bin -i` *`input_file`* `-b2e [`*`elf_type,start_label,size_label,alignment,end_label`*`] -o` *`output_file`*

Arguments

`input_file`

The input binary file.

`elf_type`

This determines the output ELF file type. Valid options are `PS3PPU` and `PS3SPU`. If not explicitly specified the default type is `PS3PPU`.

`start_label,end_label,size_label`

Specify labels for the entry points in the generated object file.

Automatic labels are generated for any arguments not specified by the user. Automatic labels are similar in format to *objcopy* generated labels, for example:

`_binary_`*`<formatted_elf_file_name>_<label_type>`*

`formatted_output_file_name`	The output file name, formatted as a valid C identifier, with all invalid characters converted to an underscore.
`label_type`	Will be `start`, `stop`, or `size`, based on the label type.

Sample automatic labels:

```
>ps3bin -i data.bin -b2e -o test.elf
```

```
_binary_test_elf_start
_binary_test_elf_stop
_binary_test_elf_size
```

alignment

The alignment argument specifies the alignment requirement of the binary data to be embedded into the .data section. If this value is not specified, it will default to 16 bytes.

output_file

The output filename.

Remarks

The output and input files are of different types, therefore you must specify an output and an input file. All other arguments are optional, and can be left unspecified.

Examples

Example 1:

```
ps3bin -i data.bin -b2e -o test.elf
```

No `-b2e` arguments specified; ELF type defaults to PS3 PPU format, automatic labels are generated.

Example 2:

```
ps3bin -i data.bin -b2e PS3PPU -o test.elf
```

PS3 PPU format ELF type specified explicitly, automatic labels are generated.

Example 3:

```
ps3bin -i data.bin -b2e ,,,4,MyEndLabel -o test.elf
```

PS3 PPU format defaults to PS3PPU, labels for start and size are automatically generated, alignment is explicitly set to 4, end label is explicitly set.

Example 4:

```
ps3bin -i data.bin -b2e PS3PPU,MyStartLabel,MySizeLabel,4,MyEndLabel -o
test.elf
```

All arguments are explicitly specified.

Example 5:

```
#include <stdio.h>

extern unsigned char const MyStartLabel[];
extern unsigned char const MyEndLabel[];
extern void * const MySizeLabel;
int main()
{
    printf("Start address = %p\n", MyStartLabel);
    printf("End address = %p\n", MyEndLabel);
    unsigned int const DataSize = (unsigned int) &MySizeLabel;
    printf("Size = %u bytes\n", DataSize);
    /* or: (MyEndLabel - MyStartLabel) */
}
```

This example shows how to access the binary object within C/C++ code.

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

This option blanks all code sections in an ELF file.

Syntax

`ps3bin –i` *input_file* `–be [-o` *output_file*`]`

Arguments

input_file

Code sections will be blanked for the specified ELF file.

output_file

An optional output filename.

Concise (-c, --concise)

This switch reduces the level of output but with a corresponding loss of information. For example, pipeline analysis information that is calculated and printed alongside disassembly output is disabled. This improves the disassembly output times by up to 40%.

Syntax

To use this option append it to any command line which outputs disassembly.

Example

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

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

Copies a binary section to an output file. This is often used when using overlays. To use this switch you must specify an output file name, a section name, and an input file.

Syntax

```
ps3bin –i input_file -cs section_name –o output_file
```

Arguments

input_file

The name of the file containing the specified section.

section_name

The name of the section to be copied.

output_file

The destination file for the copied section.

Example

```
ps3bin –i test.elf -cs .text -o new.bin
```

Demangle (-dem, --demangle)

This switch demangles C++ names.

Syntax

```
ps3bin –dem mangled_name
```

Arguments

mangled_name

The operation will be performed upon this symbol name.

Example

>ps3bin --demangle=__0fEfredEblahi

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

Disassemble address range (--disassemble-ranges)

Disassembles code within a range of addresses. Note that there is no short form for this switch.

Syntax

```
ps3bin –i input_file --disassemble-ranges=low_address..high_address[,...]
```

Arguments

input_file

The file containing the ranges to be disassembled.

low_address..high_address

Specify the range of addresses to be disassembled. The low and high address must be separated by two full-stops "..". Several ranges can be specified, separated by a comma (,).

The address can be entered in hexadecimal or decimal. Hex numbers must be prefixed with '0x' or suffixed with 'h'.

Example

```
ps3bin –i test.elf --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 also between 0x020000 and 0x030000.

Disassemble symbol (--disassemble-symbol)

Disassembles code for a supplied function symbol.

Syntax

```
ps3bin –i input_file --disassemble-symbol=function_symbol [-nd]
```

Arguments

input_file

The file containing the function to be disassembled.

function_symbol

Disassembled code will be generated for this symbol. It may be either a mangled (for example, "._Z3fooi") or demangled name.

Remarks

- If `-nd` is specified, *function_symbol* must be a mangled name.
- If *function_symbol* is a demangled name, it must match exactly the output from symbol dumps (for example, `-dsy`), including any whitespace.

Examples

```
ps3bin –i test.elf --disassemble-symbol=my_function_symbol(int)
ps3bin –i test.elf --disassemble-symbol=._Z18my_function_symbolif
ps3bin –i test.elf --disassemble-symbol=._Z18my_function_symbolif -nd
```

Display source code with disassembly (-S, --display-source)

Displays source code within disassembly output. Lines of source code are displayed above their associated instructions.

Syntax

```
ps3bin -i input_file disassembly_options -S
```

Arguments

input_file

The input file.

disassembly_options

Any valid switch that outputs disassembly, for example `-d`, `--disassemble-symbol`, `-ds .text` and so on.

Remarks

- The input file must contain debugging information. Use `-g` when compiling.
- To display source code the files containing the source code must be available on the file system. Paths embedded within the debugging information in the input file are used to find source files.

  If the source files cannot be found using the absolute paths in the debugging information, other paths will be searched based on the current working directory and the directory containing the input file.
- Source files may not be the correct version, and you will not be warned if source files have been changed since the input file was compiled.

Troubleshooting

- If source files cannot be found, place the input file somewhere within the source file directory tree.
- Alternatively, try running `ps3bin` from a directory within the source file tree, which will ensure that the current working directory is within the source tree.

Example

```
>ps3bin -i test.o -d -S

Disassembly of section .text:
---------- C:/test.cpp ----------
     1:
     2: int main ()
     3: {
```

```
main:
0x81000000: B082       sub      sp,sp,#0x8
     4:     int a = 5;
0x81000002: F04F 0005  mov      r0,#0x5
0x81000006: 9000       str      r0,[sp]
     5:     return a;
0x81000008: 9800       ldr      r0,[sp]
     6: }
0x8100000A: B002       add      sp,sp,#0x8
0x8100000C: 4770       bx       lr
---------- No source file ----------
0x8100000E: BF00       nop

_start:
0x81000010: F000 F802  bl       _initialize
0x81000014: BF00       nop
0x81000016: BF00       nop

< Output truncated >
```

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

Prints a virtual address view of the sections in the ELF file.

Syntax

```
ps3bin –i input_file -dml
```

Arguments

input_file

The name of the input ELF file.

Example

```
>ps3bin –i test.elf -dml

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)

Decodes and outputs the contents of the specified section(s).

Syntax

```
ps3bin –i input_file -ds section_name[,...] [-nd] [-c] [-v]
```

Arguments

input_file

The file containing the specified section.

section_name

Contents of the specified section will be decoded and output. Multiple sections can be specified by using a comma as a separator.

-nd

If specified, states that mangled symbol names from the specified section will be ouput.

-c

If specified, reduces the level of output.

-v

If specified, enables a detailed output.

Remarks

To use this option, a section name must be specified as well as an input file. If the application cannot decode the section then a hex dump will be displayed instead.

Example

```
>ps3bin –i test.elf -ds .strtab

.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
```

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

Prints out the sizes of the various components of the input file.

Syntax

```
ps3bin –i input_file -dsi
```

Arguments

input_file

Component sizes for this file will be printed.

Example

```
>ps3bin –i test.elf -dsi

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

Dump symbol tables (-dsy, --dump-symbols)

Dumps and outputs symbol tables.

Syntax

ps3bin –i *input_file* –dsy -s

Arguments

input_file

Symbol tables will be dumped for this file.

-s

Sorts the symbol table output.

Grep (-g, --grep)

Filters the printed output.

Syntax

ps3bin –i *input_file options* -g *filter_string*

Arguments

input_file

The input file.

options

Any valid ps3bin switch which outputs data.

filter_string

Only print lines containing the specified string.

Output fake signed file (-of fself, --oformat=fself)

The output file can be fake signed, removing the requirement of running the output ELF file through the make_fself tool.

Syntax

ps3bin –i *input_file* –of fself [--compress-output]

Arguments

input_file

The input file.

--compress-output

Optionally compresses FSELF output.

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

Renames a section.

Syntax

```
ps3bin –i input_file -rs old_section_name new_section_name [-o output_file]
```

Arguments

input_file

The input file.

old_section_name

The name of the original section.

new_section_name

The name of the new section.

output_file

An optional output file. If not specified, the operation will be performed on the input file.

Remarks

To use this option you must specify the old section name, a new section name and an input file name.

Example

```
ps3bin –i test.elf -rs my_old_section_name my_new_section_name
```

Renames "my_old_section_name" to "my_new_section_name" in test.elf.

Relocation Information (-r, --reloc)

When specified in isolation, displays the relocation entries.

When combined with a switch which dumps disassembly, displays the relocation information inline with the disassembly.

Syntax

```
ps3bin –i input_file -r [disassembly_options]
```

Arguments

input_file

The input file.

disassembly_options

Any valid ps3bin switch which outputs data, for example `-d`, `--disassemble_symbol`, or `-ds .text`.

Remarks

When displaying the relocation information inline with disassembly the relocations are printed on the line below the instruction to which they pertain.

Use cases for displaying relocations inline with disassembly include:

- The symbol referenced from an instruction is shown in the relocation information. The functions called can be interpreted from this information.
- Relocations are matched to the source code that generated them, this is important if a tool such as the linker issues an error message pertaining to a specified relocation. Examine which instruction is modified by the specified relocation to ascertain what modifications need to be made to the source code in order to fix the problem.

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

Removes a section from a file.

Syntax

```
ps3bin –i input_file -sse section_name[,...] [-o output_file]
```

Arguments

input_file

The operation will be performed on this file unless an output file is specified.

section_name

The specified section name will be removed. You can also specify multiple sections to remove; these must be comma separated.

output_file

An optional output file. If not specified, the operation will be performed on the input file.

Remarks

Stripping sections can yield an invalid output file.

Examples

Example 1:

```
ps3bin –i test.o -sse .data -o new.o
```

File test.o will be output as new.o, minus the .data section.

Example 2:

```
ps3bin –i test.o -sse .data,.text,.symtab
```

The .data, .text and .symtab sections will be removed from test.o.

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

Removes a symbol from a file.

Syntax

```
ps3bin –i input_file -ssy symbol_name[,...] [-o output_file] [-nd]
```

Arguments

input_file

The input file.

symbol_name

All symbols that match the specified symbol name will be removed from the input file. You can also specify multiple symbols to remove; these must be comma separated.

output_file

An optional output file. If not specified, the operation will be performed on the input file.

-nd

If specified, the user specified argument *symbol_name* is mangled.

Remarks

Stripping symbols can yield an invalid output file.

Examples

Example 1:

```
ps3bin –i test.o -ssy main -o new.o
```

File test.o will be output as new.o, minus the main symbol.

Example 2:

```
ps3bin –i test.o -ssy main,exit,printf,hello_world
```

All the specified symbols will be removed from test.o.

Verbose (-v, --verbose)

Enables a detailed output from certain switches.

Syntax

To use this option append it to any command line which contains `–dsh` or `–dsy`.

Remarks

The only switches that support this are dump symbol headers (`–dsh`) and dump symbol table (`-dsy`).

Examples

```
>ps3bin –i test.o -dsh

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 –i test.o -dsh -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 >
```

7: Index