Build Utilities for PlayStation®3

ユーザー ガイド

SN Systems Limited
バージョン470.1
2015年2月27日

目次

1: はじめに

ゲーム開発を支援する各種のビルド ユーティリティが用意されています。ProDG ビルド ツール パッケージには、以下のユーティリティが含まれています。

`ppu-lv2-prx-fixup.exe`	再配置を実行し、デバッグ エントリにパッチを当てる。PRX ファイルのロードおよび実行に関する情報を含む。インプットは ELF オブジェクト ファイルに部分的にリンクされた入力の 1 つのみ。「ppu-lv2-prx-fixup」を参照。
`ppu-lv2-prx-libgen.exe`	ppu-lv2-prx-libgen.exe は、PRX のスタブ ライブラリを生成するために使用する。「ppu-lv2-prx-libgen」を参照。
`ps3name.exe`	ps3name は、PlayStation®3 用の SNC コンパイラや GCC コンパイラで生成された C++ シンボルの名前をデマングルするためのコマンドライン ユーティリティです。「ps3name - 名前デマングラ」を参照。
`ps3snarl.exe`	SN アーカイブライブラリアン。「ps3snarl - SN アーカイブ ライブラリアン」を参照。
`ps3bin.exe`	SN バイナリ ユーティリティ。「snps3bin - SN バイナリ ユーティリティ」を参照。

2: ppu-lv2-prx-fixup

ppu-lv2-prx-fixup ユーティリティは、再配置、デバッグ入力内容の修正、PS3 PRX 実行に必要なシンボルの追加や、PRX リンク、セグメント作成などを実行します。このセクションには PRX ファイルのロードと実行のための情報が含まれています。インプットは ELF オブジェクト ファイルに部分的にリンクされた入力の 1 つのみです。

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

上記の *<options>* は、以下のいずれかになります。

`-v`	詳細情報モード
`-o <outputfile>`	デバッグシンボルなしで変換
`-r <outputfile>`	デバッグシンボル情報付きで変換
`-e entry_symbol`	エントリ ポイント仕様シンボル
`-m <.prx file>`	モジュール名とバージョン番号を表示

3: ppu-lv2-prx-libgen

ppu-lv2-prx-libgen ユーティリティは ILB ファイルおよびライブラリ スタブを作成します。

使用方法:`ppu-lv2-prx-libgen <inputfile(.tbl)> <options>`

上記の *<options>* は、以下のいずれかになります。

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

TBL ファイル用にエントリ テーブル ソースを作成するための構文は以下のとおりです。

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

TBL ファイルで定義されたライブラリで ILB をコールするスタブを作成するための構文は以下のとおりです。

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

4: ps3name - 名前デマングラ

名前デマングラ PS3Name は、PlayStation®3 用の SNC または GCC コンパイラで生成された C++ シンボル名をデマングルするためのコマンドライン ユーティリティです。

マングルされた名前は、タイプ情報を関数名にエンコードする際にコンパイラで使用されます。このためタイプセーフなリンクが保証されます。名前デマングラは、このタイプ情報を人間が理解できる形式にデコードします。

SNC で使用されるマングル方式は、コンパイラ処理の詳細としてのみ考慮し、この方式自体や、PS3Name の出力形式そのものに直接依存しないでください。

名前デマングラは次の 3 つの異なるモードで処理できます。

(1) マングルされた名前のリストがコマンドライン上にある場合、PS3Name はデマングルされたバージョンを、指定された順序で表示します。

(2) `--file` が指定されている場合、PS3Name はパラメータをファイル名として解釈し、マングルされた名前をそのデマングルされたものと置き換えた状態で、それぞれのファイルを順番に開き、その内容を表示します。

(3) パラメータが指定されていない場合には、PS3Name はマングルされた名前を標準入力から読み込みます。

名前デマングラのコマンドライン構文

構文

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

パラメータ

options

次の項目のうち 1 つを指定します。

`-h` `--help`	ヘルプを表示する。
`-v` `--version`	バージョン情報を表示する。

mangled_names

デマングルするマングル名のリストを指定する。

`--file`

デマングルするマングルされた名前を含むファイルのリストとして、*filenames* を解釈する。「ファイル デマングル モード (-f, --file)」を参照。

ファイル デマングル モード (-f, --file)

任意のパラメータをファイル名として解釈し、マングルされた名前をデマングルされたものと置き換えた状態で、ファイルの内容を表示します。

構文

```
ps3name --file filenames
```

引数

filenames

操作は指定されたファイル上で実行されます。

例

ファイル input.txt には次のような文字列が含まれています。

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

--file パラメータで名前デマングラを実行します。

```
>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 アーカイブ ライブラリアン

SN ARchive Librarian (snarl) は GNU ar 使用に準拠します。

コマンドライン、もしくは MRI スクリプトから ar をコントロ ールする方法については以下のウェブサイトを参照してください。**www.gnu.org**。

以下のセクションは snarl の追加機能について説明します。

アーカイブ ライブラリアンのコマンドライン構文

snarl を使うと、作成、修正、アーカイブからの抽出などができます。この場合、「アーカイブ」とは通常ライブラリを意味します。 例えばlibc.a.の場合、

使用方法:

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

ps3snarl <archive> @<file>  // 応答ファイルを使用 <file>

ps3snarl -M [mri-script]
ps3snarl -M [<mri-script]   // MRIスクリプティングを使用し、
                            // [mri-script] ファイルからのコマンドを実行
```

使用可能な「*switches*」は以下のとおりです。

`--disable-warning=<number>[,<number>[...]]`	*<number>* で参照される警告を無効にする。
`--version`	バージョンを表示する。
`--help`	このヘルプ テキストを表示する。

ここで、*<key>* は以下のいずれかである必要があります。

`d[ND]`	*<files>* を *<archive>* から削除する。
`m[abND]`	*<files>* を *<archive>* から移動する。
`p[N]`	*<archive>* 内で検出される *<files>* を印刷する。「アーカイブファイルとオブジェクトファイルの出力」を参照。
`q[cfSDT]`	*<archive>* に *<files>* をクイック アペンドする。「超高速追加」を参照。
`r[abucfSDT]`	既存 *<files>* を置き換えるか、*<archive>* に新規 <*files*> を挿入。
`s[GLWD]`	シンボル テーブル (ranlib) をリビルド (デフォルトで実行)。
`t`	*<archive>* の *<files>* 用目次を出力する。
`h[N]`	*<archive>* 内の *<files>* の mod 時を現在時刻に設定する。
`w[]`	*<archive>* シンボル テーブルを表示 する。「シンボル テーブルの表示」を参照。
`x[oCN]`	*<archive>* から *<files>* を抽出する。

...もしくは以下のシンボル操作コマンドである必要があります (「シンボル操作コマンド」を参照):

G	*<symbols>* をグローバルにする。
L	*<symbols>* をローカルにする。
W	*<symbols>* を weak にする。

...使用可能なモジュールは、

a	アーカイブメンバ *<relpos>* 後に *<files>* を追加。
b	アーカイブメンバ *<relpos>* 前に *<files>* を追加。
c	*<archive>* 作成が必要な時に警告しない。
C	抽出されたファイルの既存ファイルへの上書きを許可しない。
D	複数回シンボルが定義されたとき警告する。「複数回定義されたシンボルの警告」を参照。
f	挿入ファイル名を 16 文字未満に切り詰める。
l	C++ シンボル名をデマングル (demangle.dll が使用可能なとき)。
N	*<archive>* にある同じファイル名 *<count>* インスタンスを指定。
o	元の日付けを保持。
S	シンボル テーブルのビルドを抑制する (デフォルトでビルドされるため)。
T	*<archive>* をシン アーカイブとして出力。「シン アーカイブ」を参照してください。
u	*<files>* が新しいときのみアーカイブメンバを置換。
v	詳細モード。「詳細モード」を参照。
V	バージョンを表示する。
y	空のアーカイブを作成しない。

詳細モード

Snarl のほとんどのキー文字に「v」を追加することによって詳細 (verbose) モードに入ります。便利な方法としては、TOC の出力 (「t」) コマンドの表示が挙げられます。これにより拡張情報にライブラリのコンテンツが強制的に表示されます。

アーカイブファイルとオブジェクトファイルの出力

Snarl は、いかなるファイルタイプのアーカイブ作成にも使用できます (リンク時にはオブジェクト ファイルのみがアーカイブ シンボル テーブルに含まれることは言うまでもありません)。例えばバージョン履歴を含むテキスト ファイルは、ライブラリに簡単にストアできます。これはアーカイブ ファイルの印刷 (「p」) コマンドを使用して閲覧できます。 例：

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

オブジェクト ファイルが指定されているときは、ライブラリ アーカイバを実装すると、**アーカイブ ファイルの画面表示** (「p」) は正常に動作しません。それでも snarl はテキストベース ファイルを通常どおり表示しますが、オブジェクトファイルを検出したときは、自動的に 16 進法ダンプに切り替えます。これは stdout に送られるので、「>」DOS リダイレクト コマンドを使えば、簡単にファイルを移動できます。 例：

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

シンボル テーブルの表示

キー文字「w」を使用すると、stdout にアーカイブ シンボル テーブルをダンプします。これにより、どのシンボルがリンカで閲覧可能で、シンボルが格納されているどのオブジェクトファイルがライブラリにあるのかがわかります。C++ シンボル名のデマングルはサポートされています。

超高速追加

すべてのオブジェクト ファイルを一度に追加するのではなく、ライブラリを段階に分けてビルドする場合、「超高速追加」機能が利用できます。クイック アペンド (「q」) が指定され、シンボル テーブルのビルドが抑制されている場合 (「s」)、この機能が有効となります。これは大規模のライブラリ (20MB以上) で作業しているときのみその効力を発揮します。小規模ライブラリでは、それほど効果がありません。

これは新規ファイルを既存のデータにロードすることなくライブラリに追加することで機能します。ライブラリを機能させるため、シンボル テーブルはすべての任意の操作が完了した後でビルドする必要があります。

以下は、4 つのオブジェクト ファイルをすばやく大規模ライブラリに追加する方法を示した例です。

通常作業：

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

超高速追加の場合：

```
ps3snarl qS test.lib obj1.o obj2.o
...
ps3snarl qS test.lib obj3.o
...
ps3snarl qS test.lib obj4.o
ps3snarl s test.lib            // シンボル テーブルをリビルド
```

1 つ目の例では、*N* は各々の追加に費やされる時間です (ここで *N* は現存ライブラリのサイズに応じて生じる時間の遅延です)。 合計、3*N*秒かかることになります。

2 つ目の例では各々の追加にかかる時間は ≈0 秒、シンボル テーブルの最終リビルドにかかる時間が *N* 秒となっています。 かかる時間は合計 *N* 秒です。

警告：オブジェクト ファイルのファイル名を 15 文字以上で追加する場合は、高速追加はキャンセルされます。 拡張ファイル名セクションの再ビルドが必要なためです。解決方法として、「f」修飾子を使えば、ファイル名が 16 文字未満に切り詰められます。例：`ps3snarl qSf test.lib "filename that is longer than 15 characters.obj"`.

複数回定義されたシンボルの警告

修飾子「D」は複数回定義されたシンボルが現れた場合に警告を出します。修飾子「D」は既存のすべてのキーコマンドに追加できます。たとえば、クイック アペンドを実行し、複数回定義されたシンボルを警告するときは以下を利用します。

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

ライブラリに手を入れることなく、単に複数回定義されたシンボルをリストしたい場合は「s」をキーコマンドとして、「D」を修飾子として指定してください。これによりアーカイブ シンボル テーブルがビルドし直され、ライブラリ コンテンツを変えることなく、警告を出します。例：

```
ps3snarl sD test.lib
```

Visual Studio Integration を使用してライブラリをビルドしている場合、この機能は自動的に作動します。 警告は Visual Studio のビルド ウィンドウに表示されます。

シン アーカイブ

シン アーカイブには、バイナリ データ自体のコピーではなく、アーカイブのメンバへの参照がストアされます。この機能は、再配置可能なオブジェクト ファイルが最終リンク段階で利用可能であるようなローカル ビルドを実行するとき便利です。この機能により、実際のオブジェクト ファイル データをもつアーカイブを構築するのにかかる時間とメモリのオーバーヘッドが節約できます。

シン アーカイブは、シン アーカイブ修飾子 (「T」) を、クイック アペンド (「q」) か、または Replace (「r」) キー コマンドと一緒に使用することで作成できます。

> **注意**：アーカイブがすでに存在する場合は、適切なアーカイブの種類を指定しなければなりません。シン アーカイブを「T」修飾子を使わずに使用した場合、または通常のアーカイブを「T」修飾子とともに使用した場合には、エラーが発生します。

シン アーカイブは、たとえばファイルの出力 (「p」) や TOC の出力 (「t」) といったように、アーカイブ ファイルを出力しないキーの入力ファイルとして使用することもできます。

CREATETHIN コマンドを指定すれば、シン アーカイブを MRI スクリプト内に作成することもできます。「MRI スクリプティング」を参照してください。

> **注意**：シン アーカイブは標準アーカイブと同じように扱いますが、以下の MRI コマンドは現在サポートされていません：EXTRACT, GLOBAL, LOCAL, WEAK , and ADDLIB.

シンボル操作コマンド

シンボル操作コマンドは 3 つあります。3 つのコマンドとは、"グローバルにする" （「G」） 、"weakにする"（「W」） 、および"ローカルにする"（「L」） で、ライブラリ内でシンボル プロパティを修正するときに使います。たとえば、以下の例は test.lib の「sym1」を weak なシンボルに変換します。

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

複数のシンボルを 1 つのコマンドラインで指定することもできます。例：

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

メモ：コマンドが個別に使われたときは、ライブラリ内のシンボルを含有するオブジェクトファイルのシンボルプロパティだけが変更されます。アーカイブ シンボル テーブルは更新されません。 すなわち、ローカル シンボルをグローバル化した場合、アーカイブ シンボル テーブルを再ビルドするまで、リンカからはアーカイブ シンボル テーブルは見えません。もちろん、「s」引数を指定すれば、この操作を同時に行うことができます。

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

クロスプラットフォーム ライブラリのビルド

snarl は、ほとんどすべてのプラットフォームのライブラリの読み込み、作成が可能です (Win32を除く)。snarl は、PlayStation®2 と PlayStation®3 にてテスト済みです。

異なるプラットフォーム用にビルドされたオブジェクトファイルを含むライブラリの作成も可能です。たとえば、ライブラリ sky.a はオブジェクト ps2sky.o およびオブジェクト ps3sky.o を内包できます。さらに、これらのオブジェクトにはそれぞれ PS2_getskycoords() および PS3_getskycoords() のような関数も内包できるため、クロスプラットフォーム ゲームでのライブラリ管理が容易になります。

応答ファイル スクリプティング

Snarl はシンプルな応答ファイルをサポートしています。 たとえば、

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

ここで response.txt は次のような形式のテキストファイルです。

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

これにより、応答ファイルに記載されるモジュールを含む、アーカイブ「lib.a」が作成されます。応答ファイルは復帰改行で区切り、空行は無視されます。

レスポンス ファイルを使うと、同じ名称の既存のアーカイブは自動的に削除され、新規にビルドし直されますのでご注意ください (追加は不可能です)。より複雑な処理には MRI スクリプティングをご使用ください (詳しくは「MRI スクリプティング」を参照)。

MRI スクリプティング

コマンドラインで `snarl -M` を使用すると、snarl を簡単なコマンド言語として操作することができます。Snarl は「`SNARL >`」というプロンプトを表示して入力ができるようになります。snarl コマンド言語はコマンドラインのオプションよりも操作できる種類が少ないものの、MRI 「ライブラリアン」形式でスクリプトを作成する手間を省きます。

MRI スクリプト コマンドの形式は以下のとおりです。

- 1 行に 1 コマンド
- 大文字小文字の認識なし
- 「*」または「;」以降の文字はコメントとして扱われる
- コマンドの引数はコンマかスペースで分けられる
- 継続文字である「+」を使うと、次の行のテキストは、現在のコマンドの一部として扱われる

以下の表は snarl MRI スクリプトで使用できるコマンド一覧です。

コマンド	機能
`ADDLIB archive` `ADDLIB archive (module, module, ...module )`	現在のアーカイブに指定されたアーカイブの内容すべて (または、指定されているアーカイブ内の各モジュール) を追加。このコマンドの前に `OPEN`、もしくは `CREATE` が必要。
`ADDMOD member, member, ...member`	名前の付いたメンバをそれぞれ現在のアーカイブ内のモジュールとして追加。このコマンドの前に `OPEN`、もしくは `CREATE` が必要。
`CLEAR`	現在のアーカイブの内容を破棄し、最後の `SAVE` 以降の操作の結果をキャンセルする。現在のアーカイブに指定がなくても実行される (何も起こらない)。

`CREATE archive`
 	
`CREATETHIN archive`
	アーカイブ (またはシン アーカイブ) を作成し、それを現在のアーカイブとする (他のいろいろなコマンドで必要)。新しいアーカイブは一時的な名前で作成され、`SAVE` を行うまで実際のアーカイブとしては保存されない。既存のアーカイブは上書きすることができるが、名前のついた既存のファイルの内容は `SAVE` を行うまで破壊されない。

`DELETE module, module, ...module`	現在のアーカイブからリストされた各モジュールを削除する。これは「`snarl d archive module ... module`」と同じ効果。このコマンドの前

	に OPEN、もしくは CREATE が必要。
DIRECTORY *archive* (*module*, ...*module*) DIRECTORY *archive* (*module*, ...*module*) *outputfile*	アーカイブ中にある名前付きモジュールをリストする。別のコマンドである VERBOSE は出力の形式を指定する。verbose 出力がオフの場合には、出力は「snarl t *archive module...*」と同じ。verbose 出力がオンの場合には、リストは「snarl tv *archive module...*」と同じ。出力は通常、標準出力ストリームに出されるが、 *outputfile* を最後の引数で指定した場合には、snarl はそのファイルに出力する。
End	正常終了を示す 0 exit code で snarl から出る。このコマンドでは出力ファイルは保存されない。 最後の SAVE コマンド以降に現在のアーカイブに変更を加えている場合には、その変更は失われる。
EXTRACT *module*, *module*, ...*module*	名前付きモジュールを現在のアーカイブから取り出し、それらを現在のディレクトリにそれぞれ別のファイルとして書き出す。これは「snarl x *archive module...*」と同等。このコマンドの前に OPEN、もしくは CREATE が必要。
GLOBAL *symbol*, *symbol*, ...*symbol*	現行アーカイブからリストされた各々のシンボルをグローバルにする。「snarl G *archive symbol ... symbol*」と同等。このコマンドの前に OPEN、もしくは CREATE が必要。
LIST	VERBOSE の設定にかかわらず、現在のアーカイブの内容を「詳細」形式で表示する。この結果は「snarl tv *archive'*」と同等。このコマンドの前に OPEN、もしくは CREATE が必要。
LOCAL *symbol*, *symbol*, ...*symbol*	現行アーカイブからリストされた各々のシンボルをローカルにする。「snarl L *archive symbol ... symbol*」と同等。このコマンドの前に OPEN、もしくは CREATE が必要。
OPEN *archive*	既存のアーカイブを現在のアーカイブとして使うために開く (他のコマンドで必要な場合が多くある)。これに続いて投入するコマンドの結果としての変更内容は SAVE コマンドが使われるまでアーカイブに反映されない。
REPLACE *module*, *module*, ...*module*	現在のアーカイブにおいて、それぞれの既存のモジュール (REPLACE 引数で命名されている) を現在の作業ディレクトリ中のファイルに換える。このコマンドをエラーなしに実行するためには、そのファイルと現在のアーカイブにあるモジュールの両方が存在するものでなければならない。このコマンドの前に OPEN、もしくは CREATE が必要。
SAVE	現在のアーカイブに適用した変更を反映させ、最後の CREATE か OPEN コマンドで指定したファイル名でファイルとして保存する。このコマンドの前に OPEN、もしくは CREATE が必要。
VERBOSE	DIRECTORY からの出力を決める内部フラグを切り替える。このフラグがオンのときは、DIRECTORY の出力は「snarl tv ....」からの出力と同一になる。
WEAK *symbol*, *symbol*, ...*symbol*	現行アーカイブからリストされた各々のシンボルを weak にする。「snarl W *archive symbol ... symbol*」と同様。このコマンドの前に OPEN、もしくは CREATE が必要。
$(*ENV*)	環境変数マクロ拡張。この形式のいかなるマクロも実行時に指定の環境変数

	の値に拡張されます。例：`open $(LIB_DIR)\lib.a`.

MRI スクリプトをコマンド プロンプトから実行する

```
ps3snarl -M mri-script
ps3snarl -M <mri-script            // MRI スクリプト コマンドを表示
```

この場合、ps3 は MRI コマンドを *mri-script* で指定される MRI スクリプトファイルから取り出します。ps3snarl スクリプトのファイル名のデフォルト拡張子は .sns です。

6: ps3bin - SN バイナリ ユーティリティ

SN バイナリ ユーティリティ プログラム ps3bin.exe は、ELF ファイルおよびライブラリ ファイルを操作するためのツールです。

セクション、シンボル、デバッグ データのストリップ、セクション ヘッダー、シンボル テーブル、プログラム ヘッダーのダンプ、バイナリ ファイルへのセクションのコピー、セクションの名前変更などの機能を搭載しています。

バイナリ ユーティリティ コマンドライン構文

構文

短い形式：

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

長い形式：

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

応答ファイル：

```
ps3bin @input_file
```

パラメータ

input_file

入力ファイルです。ELF またはライブラリ ファイルが指定できます。@ コマンドを使うと、応答ファイルからスイッチを読み込むことができます。応答ファイルからは、コマンド ラインの任意の箇所を読み込めます。

output_file

出力ファイルです。種類が入力ファイルの種類と異なるときは、出力ファイルを指定しなければなりません。

options

1 つまたは複数のユーザー指定コマンドおよび、各コマンドに適用される、必要とされる修飾子です。コマンドの実行順序は左から右となります。

コマンド	
-a2l --addr2line	ソース ファイルの address の該当箇所を取得する。「アドレスを行に変換 (-a2l, --addr2line)」を参照。
-b2e --bin2elf	バイナリ ファイルを ELF オブジェクト ファイルに変換する。「バイナリを ELF ファイルに変換 (-b2e, --bin2elf)」を参照。
-be --blank-elf	ELF ファイルをブランクにする。「ELF のブランク化 (-be, --blank-elf)」を参照。
-cs --copy-section	セクションをバイナリ ファイルにコピーする「セクションをコピー (-cs, --copy-section)」を参照。
-d --disassemble	すべての実行可能コードを逆アセンブルする。
--disassemble-ranges	アドレス範囲内のコードを逆アセンブルする。「アドレス範囲の逆

	アセンブル (--disassemble-ranges)」を参照。
--disassemble-symbol	指定された関数シンボルに対するコードを逆アセンブルする。「シンボルの逆アセンブル (--disassemble-symbol」を参照。
-dd --dump-debug-data	デバッグ データをダンプする。
-de --dump-everything	すべてをダンプする。
-dem --demangle	シンボルをデマングルする。「デマングル (-dem, --demangle)」を参照。
-dh --dump-elf-header	ELF ヘッダーをダンプする。
-dml --dump-mem-layout	仮想メモリのレイアウトをダンプする。
-dph --dump-program-headers	プログラム ヘッダーをダンプする。
-ds --dump-sections	指定したセクションをダンプする。「セクションをダンプ (-ds, --dump-sections)」を参照。
-dsh --dump-section-headers	セクション ヘッダーをダンプする。
-dsi --dump-sizes	サイズ統計をダンプする「サイズをダンプ (-dsi, --dump-sizes)」を参照。
-dss --dump-stack-sizes	関数のスタック フレームのサイズ情報をダンプする。
-dsy --dump-symbols	シンボル テーブルをダンプする。「ダンプ シンボル テーブル (-dsy, --dump-symbols)」を参照。
-G -- globalize- symbol=NAME	シンボルをグローバルとしてマークする。
--help	コマンド ラインのヘルプを出力する。
-L --localize-symbol=NAME	シンボルをローカルとしてマークする。
-nd --no-demangle	C++ シンボル名をデマングルしない。
-r --reloc	再配置情報を表示する。「再配置情報 (-r, --reloc) 」を参照。
-rs --rename-sections	セクションの名前を変更する。「セクション名の変更 (-rs, --rename-sections)」を参照。
-sa --strip-all	すべてのシンボルおよびデバッグ情報をストリップする。
-sd --strip-debug	すべてのデバッグ情報をストリップする。
-sse --strip-sections	セクションをストリップする。「セクションの削除 (-sse, --strip-sections)」を参照。

-ssy --strip-symbols	シンボルをストリップする。「シンボルの削除 (-ssy, --strip-symbols)」を参照。
-w --weaken-symbol=NAME	シンボルを weak としてマークする。

修飾子	
-c --concise	最小限の情報を出力する。「簡潔化 (-c, --concise)」を参照。
--disable-all-warnings	すべての警告メッセージを無効にする。
-gnu --gnu-mode	GNU スタイル形式で出力する (現時点では addr2line のみ)。「アドレスを行に変換 (-a2l, --addr2line)」を参照。
-g --grep	指定した文字列を含む行のみ出力する。「Grep (-g, --grep)」を参照。
-nd	デマングルしない。
-of fself --oformat=fself	出力ファイルは仮の署名付きで、make_fself ツールから出力 ELF ファイルを実行する必要性をなくします。「仮の署名付きファイルの出力 (-of fself, --oformat=fself)」を参照。
-p --page-output	情報を、1 画面ごとにポーズする。
-S, --display-source	逆アセンブリ出力内にソース コードを表示する。「逆アセンブリと共にソース コードを表示 (-S, --display-source)」を参照。
-v --verbose	利用できるすべての情報を出力する。「Verbose (-v, --verbose)」を参照。
-ver --version	バージョン番号を表示する。
-x --hex	すべてのセクション ダンプを、プレーンな 16 進ダンプとして実行する。

コメント

コマンドラインでは、入力ファイルと出力ファイルをそれぞれ 1 つずつしか指定できません。入力ファイルを指定しない場合は、識別できない最初の引数が入力ファイルの名前として使用されます。スペースを含む長いファイル名は、引用符で囲んでください (例：「--infile="C:\my long filename.elf"」)。

コマンドラインのオプションは任意の順番で指定でき、コマンドラインからは、長い形式を使用、または短い省略形式を使用して時間を節約することができます。引数を使用するオプションの省略形を使用する場合は、省略形オプションの後にスペースを挿入して引数を指定します。たとえば、「--rename-section=」は「-rs」と同じです。

一部のオプションでは、コンマで区切って複数の引数を使用でき (例：「--dump-sections」) 、このような場合は引数の数に制限はありません。

また、すべてのオプションは、入力ファイルの全タイプに対応しています。

アドレスを行に変換 (-a2l, --addr2line)

任意のアドレスについて、ソース ファイル中の対応する行を決定します。

構文

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

引数

input_file

ELF ファイル。ここから、(--addr2line) オプションが試行され、ソース ファイルの該当箇所が処理されます。

address

アドレスには、16 進数と 10 進数が使用できますが、16 進数には前に「0x」または後ろに「h」を付ける必要があります。

これを省略する場合には、アドレスは stdin から指定しなければなりません。

-gnu

コマンドラインで -gnu または --gnu-mode が指定されている場合、出力は GNU addr2line ツールと同じ形式で表示されます。

-nd

マングルされたシンボル名の出力を指定します。*--no-demangle* として指定することもできます。

例

例 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)
```

-a2l (--addr2line) オプションにより test.elf が解析され、指定されたアドレスについて、ソース ファイルの該当箇所が決定されます。

例 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)
```

コマンド ラインからはアドレスは指定されていませんが、アドレスは stdin を通じて指定されます。

バイナリを ELF ファイルに変換 (-b2e, --bin2elf)

-b2e (--bin2elf) オプションを使うと、バイナリ ファイルが ELF オブジェクト ファイルに変換されます。この生成されたオブジェクト ファイルはその後プロジェクトにリンクすることができます。また、リソースは -b2e オプションによって指定されたラベルによって参照できます。

構文

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

引数

input_file

入力バイナリ ファイルです。

elf_type

これにより出力 ELF ファイルの種類が決定されます。有効なオプションは PS3PPU と PS3SPU です。これが明示的に指定されていない場合は、デフォルトタイプはPS3PPU となります。

start_label,end_label,size_label

生成されたオブジェクト ファイルのエントリ ポイントに対してラベルを指定します。

ユーザーが指定しなかった引数に対しては自動ラベルが生成されます。自動ラベルは *objcopy* により生成されたラベルとその形式が似ています。例を以下に示します。

binary<formatted_elf_file_name>_<label_type>

formatted_output_file_name	有効な C 識別子として書式化された出力ファイル名。無効な文字はすべてアンダースコアに変換されます。
label_type	ラベル タイプにより start、stop または size となります。

自動ラベルの例：

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

alignment

アライメント引数は .data セクションに埋め込まれるバイナリ データのアラインメント要件を指定するものです。この値が指定されていない場合には、デフォルトで 16 バイトに指定されます。

output_file

出力ファイル名です。

コメント

出力ファイルと入力ファイルの種類は異なります。このため、出力ファイルと入力ファイルを指定しなければなりません。その他の引数はオプションです。このため指定しないでおくこともできます。

例

例 1：

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

-b2e 引数が指定されていません。ELF タイプはデフォルトで PS3 PPU 形式に指定され、自動ラベルが生成されます。

例 2：

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

PS3 PPU 形式の ELF タイプが明示的に指定され、自動ラベルが生成されます。

例 3：

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

PS3 PPU 形式はデフォルトで PS3PPU に指定され、start および size のラベルが自動的に生成されます。アラインメントは明示的に 4 にセットされ、end ラベルは明示的にセットされます。

例 4 :

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

すべての引数は明示的に指定されます。

例 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) */
}
```

この例は C/C++ コード内でバイナリ オブジェクトにアクセスする方法を示したものです。

ELF のブランク化 (-be, --blank-elf)

このオプションでは、ELF ファイルのすべてのコード セクションをブランクにします。

構文

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

引数

input_file

指定した ELF ファイルについて、コード セクションはブランクにされます。

output_file

オプションの出力ファイル名です。

簡潔化 (-c, --concise)

このスイッチを使うと、出力のレベルが低減されますが、それに伴いデータも失われます。たとえば、逆アセンブリ出力と一緒に計算・出力されるパイプライン解析情報は無効にされます。これにより逆アセンブリ出力にかかる時間が 40% 程度改善されます。

構文

このオプションを使うときは、逆アセンブリを出力するコマンド ラインにこのオプションを追加してください。

例

```
>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.
>psp2bin --disassemble-ranges=0x010250..0x010300,0x020000..0x030000
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
```

このシンボル テーブルのダンプでは、GNU NM ツールの最低限の構文のエミュレートが試行されます。詳細は、http://www.gnu.org/software/binutils/manual/html_chapter/binutils_2.html を参照してください。

セクションをコピー (-cs, --copy-section)

バイナリ セクションを出力ファイルにコピーします。これは、オーバーレイを使うとき、よく使用します。このスイッチを使用するには、出力ファイル名、セクション名と入力ファイル名を指定してください。

構文

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

引数

input_file

指定したセクションを含むファイルの名前です。

section_name

コピーするセクションの名前です。

output_file

コピーしたセクションのターゲット ファイルです。

例

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

デマングル (-dem, --demangle)

このスイッチは C++ の名前をデマングルします。

構文

```
ps3bin –dem mangled_name
```

引数

mangled_name

操作はこのシンボル名上で実行されます。

例

```
>ps3bin --demangle=__0fEfredEblahi

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

アドレス範囲の逆アセンブル (--disassemble-ranges)

アドレス範囲内のコードを逆アセンブルします。このスイッチについては、短い形式はありません。ご注意ください。

構文

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

引数

input_file

逆アセンブリする範囲を含んでいるファイルです。

low_address..high_address

逆アセンブリするアドレス範囲を指定します。下位アドレスと上位アドレスは 2 つのフルストップ「..」で分離してください。複数の範囲を指定できます。コンマ（,）で分離してください。

アドレスには、16 進数と 10 進数が使用できますが、16 進数には前に「0x」または後ろに「h」を付ける必要があります。

例

```
ps3bin –i test.elf --disassemble-ranges=0x010250..0x010300,0x020000..0x030000
```

これにより、0x010250 と 0x010300、0x020000 と 0x030000 の間にあるプログラム アドレス範囲の逆アセンブリの部分が出力されます。

シンボルの逆アセンブル (--disassemble-symbol)

指定された関数シンボルに対するコードが逆アセンブルされます。

構文

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

引数

input_file

逆アセンブリする関数を含むファイルです。

function_symbol

このシンボルに対して逆アセンブリ コードが生成されます。名前は、マングルされたもの (たとえば "._Z3fooi" など) またはデマングルされたものとなります。

コメント

- -nd が指定されている場合、*function_symbol* はマングル名としなければなりません。
- *function_symbol* がデマングルされた C++ 名である場合、空白文字も含め、シンボル ダンプからの出力に完全に一致しなければなりません (たとえば 「-dsy」など)。

例

```
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
```

逆アセンブリと共にソース コードを表示 (-S, --display-source)

逆アセンブリ出力内にソース コードを表示する。関連する命令の上にソース コードの数行が表示されます。

構文

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

引数

input_file

入力ファイルです。

disassembly_options

逆アセンブリを出力する任意の有効なスイッチです (例：-d、--disassemble-symbol、-ds .text など)。

コメント

- 入力ファイルにはデバッグ情報が必要です。コンパイル中には -g を使用してください。
- ソース コードを表示するときは、ソース コードを内包しているファイルがファイル システム上で利用できなければなりません。入力ファイルのデバッグ情報内に埋め込まれているパスはソース ファイルの検索に使用されます。

  デバッグ情報内の絶対パスを使用してもソース ファイルが見つからない場合、現在の作業ディレクトリと入力ファイルを持つディレクトリに基づいてその他のパスが検索されます。
- ソース ファイルは正しいバージョンでない場合があり、その場合、入力ファイルがコンパイルされてからソース ファイルが変更されていても警告は表示されません。

トラブルシューティング

- ソース ファイルが見つからない場合、入力ファイルをソース ファイル ディレクトリ ツリー内のどこかに配置してください。
- または、ソース ファイル ツリー内のディレクトリから ps3bin の実行を試みてください。これにより現在の作業ディレクトリがソース ツリー内にあることが保証されます。

例

```
>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

< 出力はこれ以降、切り捨てられています >
```

メモリ レイアウトのダンプ (-dml, --dump-mem-layout)

ELF ファイル内のセクションの仮想アドレス ビューが出力されます。

構文

```
ps3bin –i input_file -dml
```

引数

input_file

入力 ELF ファイルの名前です。

例

```
>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
```

セクションをダンプ (-ds, --dump-sections)

指定したセクションのコンテンツをデコード/出力します。

構文

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

引数

input_file

指定したセクションを含むファイルです。

section_name

指定したセクションの内容がデコードされ、出力されます。また、コンマで区切ることによって、複数のセクションを指定することもできます。

-nd

指定されていれば、指定したセクションからマングルされたシンボル名が提示されます。

-c

指定されていれば、出力のレベルを低減します。

-v

指定されていれば、詳細出力を有効にします。

コメント

このオプションを使用する場合は、入力ファイルだけでなく、セクション名も指定しなければなりません。アプリケーションでセクションをデコードできない場合は、16 進ダンプが代わりに表示されます。

例

```
>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
```

サイズをダンプ (-dsi, --dump-sizes)

入力ファイルのさまざまなコンポーネントのサイズを出力します。

構文

ps3bin –i *input_file* -dsi

引数

input_file

このファイルのコンポーネント サイズが出力されます。

例

```
>ps3bin –i test.elf -dsi

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

ダンプ シンボル テーブル (-dsy, --dump-symbols)

シンボル テーブルをダンプし、出力します。

構文

```
ps3bin –i input_file –dsy -s
```

引数

input_file

シンボル テーブルはこのファイルに対してダンプされます。

-s

シンボル テーブルの出力を並べ替えます。

Grep (-g, --grep)

プリントした出力をフィルタします。

構文

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

引数

input_file

入力ファイルです。

options

データを出力する、任意の有効な ps3bin スイッチです。

filter_string

指定した文字列を含む行のみを表示します。

仮の署名付きファイルの出力 (-of fself, --oformat=fself)

make_fself ツールから出力 ELF ファイルを実行する必要性をなくすため、出力ファイルは仮の署名付きにすることができます。

構文

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

引数

input_file

入力ファイルです。

--compress-output

オプションとして FSELF 出力を圧縮します。

再配置情報 (-r, --reloc)

隔離された状態で指定された場合、再配置エントリを表示します。

逆アセンブリをダンプするスイッチと併用した場合、その逆アセンブリに一致する再配置情報を表示します。

構文

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

引数

input_file

入力ファイルです。

disassembly_options

データを出力する任意の有効な ps3スイッチです（例：`-d`, `--disassemble_symbol`, or `-ds .text`など)。

コメント

逆アセンブリと一致する再配置情報を表示する際、その再配置場所は関連する命令の下の行に表示されます。

逆アセンブリと一致する再配置場所の表示に関するユース ケースには次のようなものが挙げられます。

- 命令から参照されたシンボルは、再配置情報内に表示されます。呼び出された関数は、この情報から解釈することができます。
- 再配置情報は、再配置が生成されたソース コードに一致します。指定された再配置に関するエラー メッセージがリンカなどのツールから発行される場合にこの点が重要になります。指定された再配置のためにどの命令が修正されているのかを検討し、この問題を修正するため、どういった修正をソース コードに加える必要があるのかを突き止めてください。

セクション名の変更 (-rs, --rename-sections)

セクション名を変更します。

構文

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

引数

input_file

入力ファイルです。

old_section_name

オリジナル セクションの名前です。

new_section_name

新しいセクションの名前です。

output_file

オプションの出力ファイルです。指定されていない場合には、操作は入力ファイル上で実行されます。

コメント

このオプションを使用する場合は、古いセクション名、新しいセクション名、入力ファイル名を指定してください。

例

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

test.elf 内で、「my_old_section_name」を「my_new_section_name」という名前に変更します。

セクションの削除 (-sse, --strip-sections)

ファイルからセクションを削除します。

構文

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

引数

input_file

出力ファイルが指定されていない限り、操作はこのファイル上で実行されます。

section_name

指定したセクション名は削除されます。複数のセクションを指定して削除することができます。その場合は、コンマで区分してください。

output_file

オプションの出力ファイルです。指定されていない場合には、操作は入力ファイル上で実行されます。

コメント

セクションのストリップを行うと、無効な出力ファイルが生成される場合があります。

例

例 1 :

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

ファイル test.o は new.o として出力され、.data セクションが削除されます。

例 2 :

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

.data セクション、.text セクション、.symtab セクションは test.o から削除されます。

シンボルの削除 (-ssy, --strip-symbols)

ファイルからシンボルを削除します。

構文

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

引数

input_file

入力ファイルです。

symbol_name

指定したシンボル名と一致するすべてのシンボルが入力ファイルから削除されます。複数のシンボルを指定して削除することができます。その場合は、コンマで区分してください。

output_file

オプションの出力ファイルです。指定されていない場合には、操作は入力ファイル上で実行されます。

-nd

指定されていれば、ユーザー指定の引数 *symbol_name* がマングルされます。

コメント

シンボルのストリップを行うと、無効な出力ファイルが生成される場合があります。

例

例 1 :

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

ファイル test.o は new.o として出力され、main シンボルが削除されます。

例 2 :

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

指定されたシンボルがすべて test.o から削除されます。

詳細情報 (-v, --verbose)

特定のスイッチからの詳細な出力を有効にします。

構文

このオプションを使うときは、-dsh または -dsy を含むコマンド ラインにこのオプションを追加してください。

コメント

このオプションをサポートしているスイッチは、シンボル ヘッダーのダンプ (-dsh) とシンボル テーブルのダンプ (-dsy) のみです。

例

```
>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
```

```
< 出力はこれ以降、切り捨てられています >
```

7:インデックス