SN SYSTEMS
Sony Computer Entertainment Group

ユーザー ガイド

ProDG for PlayStation®3 Utilities

SN Systems Limited
バージョン 310.1
2009年11月11日

目次

1: はじめに

ドキュメント変更履歴

Ver.	日付	変更
310.1	2009年11月	Bz78822:「簡潔化 (-c, --concise)」を削除。 Bz78823:--disassemble-symbol 及び「アドレス範囲の逆アセンブル (--disassemble-ranges) シンボルの逆アセンブル (--disassemble-symbol)」'を更新。 Bz78824: --compress-output を更新。 Bz78223:「バイナリ ユーティリティのコマンドライン構文」を削除。
300.1	2009年9月	Bz77912:「バイナリ ユーティリティのコマンドライン構文」を削除。
280.1	2009年5月	Bz72048:「バイナリ ユーティリティのコマンドライン構文」及び「アドレスを行に変換 (-a2l, --addr2line) 」を削除。

概要

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 ッケージには、以下のユーティリティが含まれています。

ps3snarl.exe	SN アーカイヴライブラリアン、5 ページの「ps3snarl - SN アーカイヴライブラリアン」を参照。
ps3bin.exe	SN バイナリユーティリティ、10ページの「ps3bin SN バイナリ ユーティリティ」を参照。

ps3name - 名前デマングラ

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++ 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。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、タイプの安全なリンクが確保されます。ps3name では、この情報が人間が理解できる形式にデコードされます。

PlayStation®3 用の SNC コンパイラと GCC コンパイラで使用されるマングル方式は、SNC のマングルされた名前には通常「_Q」という接頭文字が使用され、GCC では「_Z」という接頭文字が使用されるという点を除いて同様のものとなっています。SNC- でコンパイルされた C++ オブジェクト モジュールと GCC- でコンパイルされた C++ オブジェクト モジュールとは互換性がないため、これらの偶発的な相互リンクを防止するために、異なる接頭文字が使用されます。

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。

使用形式: `ps3name <options> <mangled-name>`

<options> には以下が設定できます。

`-h, --help`	ヘルプを表示する
`-s, --substitutions`	代替項目を表示する
`-v, --version`	バージョン情報を表示する

代替項目とはマングル方式の内部処理の詳細で、このオプションはデバッグの支援を目的としています。

ps3snarl - SN アーカイヴライブラリアン

SN ARchive Librarian(SNアーカイヴライブラリアン、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snarlもしくはarをコントロ ールする方法についての情報は以下のウェブサイトでご覧いただけます。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。

アーカイヴライブラリアンのコマンドライン書式

snarlは作成、修正、アーカイヴからの抽出などを可能にします。この場合、アーカイヴとは通常ライブラリを意味します。例えばlibc.a.の場合、

使用法:

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

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

ps3snarl --help               // コマンドリストの印刷
```

<キー>は以下のいずれか、

d[NT]	ファイルをアーカイヴから消去
m[abNT]	ファイルをアーカイヴから移動
p[N]	アーカイヴのファイルを印刷、6 ページの「アーカイヴファイルとオブジェクトファイルの印刷」を参照。
q[cfST]	アーカイヴにファイルをすばやく追加、7 ページの「高速追加」を参照。
r[abucfST]	アーカイヴに現存もしくは新規ファイルを挿入
s[GLWT]	シンボル表(ranlib)をリビルド(デフォルトで実行)
t	アーカイヴのファイル用コンテンツ表を印刷
h[N]	アーカイヴのファイルのmod時を現在時刻にセット
w[l]	アーカイヴシンボル表を表示、7 ページの「シンボル表の表示」を参照。
F[lN]	ファイル用シンボル表を全て表示、7 ページの「シンボル表の表示」を参照。
x[oCN]	アーカイヴからファイルを抽出

…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 (8 ページの「シンボル操作コマンド」を参照)。

G	シンボルをグローバルにする
L	シンボルをローカルにする
W	シンボルを非重要項目にする

…使用可能なモジュールは、

a	アーカイヴメンバー[relpos]後にファイルを追加
b	アーカイヴメンバー[relpos]前にファイルを追加
c	アーカイヴ作成が必要な時に警告しない
C	解凍ファイルの既存ファイルオーバーライトを許可しない
f	8.3ファーマットに挿入ファイル名をトランケート
l	C++ シンボル名をデマングル（demangle.dllが使用可能な時）
N	アーカイヴの同じファイル名[count]インスタンスを定義
o	元の日付けを保存
S	シンボル表のビルドを抑制する（デフォルトでビルドされるため）
T	複数回シンボルが定義された時警告、7 ページの「複数回定義されたシンボルの警告」を参照。
u	ファイルが新しい時のみアーカイヴメンバーを置換
v	Verboseモード、6 ページの「Verboseモード」を参照。
V	バージョンを表示
y	空のアーカイヴを作成しない

Verbose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verboseモードに入ります。もっとも有用な方法として、目次('t')コマンドの印刷があげられます。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。

アーカイヴファイルとオブジェクトファイルの印刷

snarlは、いかなるファイルタイプのアーカイヴ作成に使用できます（リンク時ごとにオブジェクトファイルのみアーカイヴシンボル表に含まれることは言うまでもありません）。例えばバージョン史を含むテキストファイルは簡単にライブラリにストアできます。

これは**アーカイヴファイルの印刷**('p')コマンドを使用して閲覧できます。例えば、

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

その他のライブラリアーカイバの扱わない事項として、オブジェクトファイルが指定されている時は、**アーカイヴファイルの印刷**('p')は 正常に動作しない、ということが上げられます。それでもsnarlはテキストベースファイルは通常通り表示しますが、オブジェクトファイルが無効になっているときは、自動的に16進法フォーマットでダンプするようスイッチします。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、ファイルを移動できます。例えば、

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

シンボル表の表示

キー文字'w'はstdoutにアーカイヴシンボル表をダンプします。これはどの シンボルがリンカに閲覧可で、ライブラリ内のどのオブジェクトファイルに属しているのかを示します。

バージョン1.3.3.95以降はC++シンボル名のデマングルをサポートしています。

キー文字'F'はオブジェクトファイルの全てをシンボル表からstdout にダンプします（バージョン1.4.2.9以降）。これにより、ライブラリのシンボル全てがタイプやスコープと同様、閲覧可能になります。かなりの量になりますので、ファイルを配列することをお勧めします。

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

高速追加

ライブラリを段階に分けてビルドする場合、（すなわち全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）「高速追加」機能をご利用いただけます。簡便追加 ('q')が指定され、シンボル表のビルドが抑制('S')されている場合、この機能は有効となります。ただし、大規模のライブラリ（20MB以上）で作業中のみ有効です。小規模ライブラリの時には、それほどの効果がないためです。

新規ファイルを現存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、シンボル表は一度任意の操作完了後にビルドしておく必要があります。

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
```

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             // シンボル表をリビルド
```

一つめの例では各々の追加時に費やされる時間は*N*です（*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。

二つめの例では各々の追加時にかかる時間は≈0秒、シンボル表の最終ビルドにかかる時間が*N*秒となっています。かかる時間*N*秒です。

注: もし オブジェクトファイルのファイル名を15文字以上で追加したい時は一つめの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リビルドが必要なためです。解決方法として、f文字を使えば、ファイル名を8.3 DOS フォーマットに短縮します。`ps3snarl qSf test.lib`のファイル名は15字以上です。

複数回定義されたシンボルの警告

'T'は複数回定義されたシンボルが現れた際に警告を出し、既存のキーコマンドのいずれに追加できます。例えば、高速追加を実行した際、複数回定義されたシンボルの使用を警告します。

```
ps3snarl qT 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のみしたい場合は's'をキーコマンドとして指定、'T'を修飾語句として指定して下さい。アーカイヴシンボル表をリビルドし、ライブラリコンテンツは変えることなく、警告を出します。例:

```
ps3snarl sT test.lib
```

Visual Studio 統合を使用してライブラリをビルドすればこの機能は自動的に作動されます。警告はVisual Studioのビルドウィンドウに表示されます。

シンボル操作コマンド

3つのシンボル操作コマンドがあります。ライブラリの内部でシンボルプロパティを修正する時に使う、"グローバルにする" (G)、"非重要項目にする"(W)、および"ローカルにする"(L)の3つです。(バージョン1.4.2.9以降)

次の例では、test.libのシンボルを非重要シンボルに変換します。

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

複数のシンボルを一つのコマンドラインに指定することもできます。

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

注:コマンドが個別に使われた時は、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を変更します。アーカイヴシンボル表は更新されません。すなわち、ローカルシンボルをグローバルとした場合、アーカイヴシンボル表をリビルドするまで、リンカには不詳となります。もちろん、's'引数を使えば、同時に指定できます。

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

クロスプラットフォーム

snarlは、ほとんど全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)。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。例えば、ライブラリsky.aはps2sky.oおよびps3sky.oのオブジェクトを内包できます。また、クロスプラットフォームのゲームライブラリの監理を容易にするため、PS2_getskycoords() および PS3_getskycoords()のような関数も漸次内包できます。

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。例えば、

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

ここでresponse.txtはフォーマットのテキストファイルです。

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

応答ファイルは同じ名称のアーカイヴは自動的に削除し、新規ビルドし直しますのでご注意下さい(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8ページの「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"librarian"形式でスクリプトを作成する手間を省きます。

-E コマンドは、ファイル名に拡張子 '.sns' の付いたファイルを強制的に snarl に関連付けます。

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

MRIスクリプトコマンドの形式は以下の通りです。

- 1行に1コマンド
- 大文字小文字の認識無し
- '*'か';'以降の文字はコメント
- コマンドの引数はカンマかスペースで分けられる
- '+'は継続を表し、このマークがついている行の文字は、現在のコマンドと同等に扱われる

以下の表は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)	現在のアーカイヴに指定されたアーカイヴの内容全て(または、指定されているアーカイヴ内の各モジュール)を追加。このコマンドの前にOPENかCREATEコマンドが必要。
ADDMOD member, member, ... member	各名前付メンバーを現在のアーカイヴ中のモジュールとして追加。このコマンドの前にOPENかCREATEコマンドが必要。
CLEAR	現在のアーカイヴの内容を消去し、最後のSAVE以降の操作の結果をキャンセルする。現在のアーカイヴに指定がなくても実行はされる(何も起こらない)。
CREATE archive	アーカイヴを作り、それを現在のアーカイヴとする(他のいろいろなコマンドで必要となる)。新しいアーカイヴは一時的な名前で作られ、SAVEを行なうまで実際にアーカイブとして書き込まれない。既にあるアーカイヴを上書きすることができるが、名前のついたファイルの内容はSAVEを行なうまでは破壊されない。
DELETE module, module, ... module	現在のアーカイヴからリストされた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かCREATEコマンドが必要。
DIRECTORY archive (module, ... module) DIRECTORY archive (module, ... module) outputfile	アーカイヴ中にある名前付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。オンの場合には、リストは'snarl tv archive module...'と同じ。出力は普通は標準出力に出されるが、出力ファイルを最後のパラメータ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exit code 0でsnarlから出る。このコマンドは出力ファイルをセーブしない。最後のSAVEコマンド以降に現在のアーカイヴ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	名前付モジュールを現在のアーカイヴから取り出し、それらを現在のディレクトリにそれぞれファイルとして書き出する。これは'snarl x archive module...'と同じ。このコマンドの前にOPENかCREATEコマンドが必要。
GLOBAL symbol, symbol, ... symbol	現行アーカイヴ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 'と同様。「開く」、もしくは「作成」に先立って使用のこと。
LIST	VERBOSEの設定にかかわらず、現在のアーカイヴの内容を"verbose"形式で表示する。この結果は'snarl tv archive'と同じ。このコマンドの前にOPENか

	CREATEコマンドが必要。
LOCAL symbol, symbol, ... symbol	現行アーカイヴ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'と同様。「開く」、もしくは「作成」に先立って使用のこと。
OPEN archive	既存のアーカイヴを現在のアーカイヴとして使うためにオープンする(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はSAVEコマンドが使われるまではアーカイヴに反映されない。
REPLACE module, module, ... module	現在のアーカイヴにおいて、既存のモジュール(REPLACEコマンドで指定されている)を現在のワーキング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ヴ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かCREATEコマンドが必要。
SAVE	現在のアーカイヴに適用された変更を反映させ、最後のCREATEかOPENコマンドで指定されたファイル名でセーブする。このコマンドの前にOPENかCREATEコマンドが必要。
VERBOSE	DIRECTORYの出力を決める内部フラグを切り替える。このフラグがオンだと、DIRECTORYの出力は'snarl tv 'の出力と同一になる。
WEAK symbol, symbol, ... symbol	現行アーカイヴからリストされた各々のシンボルを非重要項目にする。'snarl W archive symbol... symbol'と同様。「開く」、もしくは「作成」に先立って使用のこと。
$(ENV)	環境変数マクロ拡張。このフォーマット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.aj

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?                    // display MRI script commands
```

この場合、ps3snarl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。

Windows ExplorerからMRIスクリプトを実行する方法

1. ps3snarlと.sns (snarlスクリプト) 機能拡張をコマンド'ps3snarl -E'を使用して結合させる
2. .sns ファイル名機能拡張で全てのスクリプトをプレーンテキスト形式で保存
3. 保存したファイルをダブルクリックして自動実行

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/アーカイブ ファイルを操作するためのツールで、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、シンボル テーブル、プログラム ヘッダのダンプ、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 ユーティリティのコマンドライン構文

使用方法

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

```
ps3bin --infile=<input> <options> --outfile=<output>
                    // long form
ps3bin @<file>   // use response file <file>
```

この <input> と <output> は ELF ファイルまたはライブラリ/アーカイブ ファイル、<options> は以下のいずれかになります。

排他的オプション	
-a2l <address>または --addr2line=<address>	ソース ファイルの <address> の箇所を取得する (13 ページの「アドレスを行に変換 (-a2l, --addr2line)」を参照)。
-b2e <file>または--bin2elf=<file>	バイナリ ファイル <file> をELF オブジェクト ファイルに変換する (13 ページの「バイナリを ELF ファイルに変換 (-b2e, --bin2elf)」を参照)。
-beまたは--blank-elf	ELF ファイルをブランクにする (14 ページの「ELF のブランク化 (-be, --blank-elf)」を参照)。
-cs <sect>または --copy-section=<sect>	セクションをバイナリ ファイルにコピーする (14 ページの「セクションをコピー (-cs, --copy-section)」を参照)。
-dまたは--disassemble	すべてのコードを逆アセンブルする。
-ddまたは--dump-debug-data	デバッグ データをダンプする。
-deまたは--dump-everything	すべてをダンプする。
-dem <symb>または --demangle=<symb>	シンボル <symb> をデマングル (デコード) する (15 ページの「デマングル (-dem, --demangle)」を参照)。
-dhまたは--dump-elf-header	ELF ヘッダをダンプする (15 ページの「ELF ヘッダのダンプ (-dh, --dump-elf-header)」を参照)。
-D or --disassemble-all	すべてのコードを逆アセンブルする。
--disassemble-ranges=<range>	アドレス範囲内のコードを逆アセンブルする(15 ページの「アドレス範囲の逆アセンブル (--disassemble-ranges)」を参照)。
--disassemble-symbol=<symbol>	指定された関数シンボル <symbol> に対するコードを逆アセンブルする (15 ページの「シンボルの逆アセンブル (--disassemble-symbol)」を参照)。.
-dmlまたは--dump-mem-layout	仮想メモリのレイアウトをダンプする。
-dphまたは--dump-program-headers	プログラム ヘッダをダンプする。
-ds <sect>,<sect>,...または --dump-sections=<sect>,<sect>,...	指定したセクションをダンプする (16 ページの「セクションをダンプ (-ds, --dump-sections)」を参照)。
-dshまたは--dump-section-headers	セクション ヘッダをダンプする。
-dsiまたは--dump-sizes	サイズ情報をダンプする (17 ページの「サイズをダンプ (-dsi, --dump-sizes)」を参照)。
-dss or --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または--dump-symbols	シンボル テーブルをダンプする。
-nd or --no-demangle	C++ シンボル名をデマングルしない。
-rs <old> <new>または --rename-	セクションの名前を <old> から <new> に変更する (17 ページの「セクション名の変更 (-rs, --rename-sections)」を参照)。

sections=<old>,<new>	
-s or --sort-data	出力情報を並べ替える (現在は、-dsy オプションでのみサポート)。
-saまたは--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または--strip-debug	すべてのデバッグ データをストリップする。
-sse <sect>,<sect>,...または --strip-sections=<sect>,<sect>,...	セクションをストリップする (17 ページの「セクションの削除 (-sse, --strip-sections)」を参照)。
-ssy <symb>,<symb>,...または --strip-symbols=<symb>,<symb>,...	シンボルをストリップする (17 ページの「シンボルの削除 (-ssy, --strip-symbols)」を参照)。
非排他的オプション	
@<file>	レスポンス ファイル <file> からスイッチを読み込む。ファイル <file> からは、コマンドラインの任意の箇所を読み込めます。
-cまたは--concise	最小限の情報を出力する (14 ページの「簡潔化 (-c, --concise)」を参照)。
--compress-output	FSELF 出力ファイルを圧縮する。効果を得るためには、-of fself または --oformat=fself と共に使用する必要があります。
-g <string>または --grep=<string>	<string> を含む行のみをプリントする。
-gnu or --gnu-mode	GNU モード。処理の出力を、GNU binutil 出力のスタイルに合わせてフォーマットする (現在は addr2line のみをサポート)。
-i <file>または--infile=<file>	入力ファイルを指定する。
-nd	デマングルしない。
-o <file>または--outfile=<file>	出力ファイルを指定する。
-of fselfまたは--oformat=fself	出力ファイルには電子署名が付加されるため、出力された ELF ファイルに対して make_fself ツールを使う必要がなくなります。
-pまたは--page-output	情報を、1 画面ごとにポーズする。
-vまたは--verbose	存在するすべての情報を出力する (17 ページの「詳細情報 (-v, --verbose)」を参照)。
-verまたは--version	バージョン番号を表示する。

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=.sect」は「-rs .sect」となります。

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

「排他的オプション」はコマンドライン上で別の排他的オプションと併用することができませんが、「非排他的オプション」は、別のオプションと併用できます。

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)

-a2l (--addr2line) オプションでは、デバッグ データが解析され、ソース ファイルの該当箇所が試行および処理されます (以下を参照)。

```
>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)
```

アドレスは、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。Snbin でソース ファイルが検出されると、ソース行がプリントされます。

ps3bin では、コマンド ラインでアドレスが指定されない場合、stdin による addr2line へのアドレス提供が認められています。以下はその例です。

```
>ps3in -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 では、コマンドラインで -gnu または --gnu- モードを指定することにより、GNU addr2line ツールと同じフォーマットで addr2line 情報を表示することができます。

例:

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

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

バイナリを ELF ファイルに変換 (-b2e, --bin2elf)

-b2e (--bin2elf) オプションでは、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 (以下を参照)。

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

この最初の引数は ELF のタイプとなり、「PSP」、「PS3SPU」、「PS3PPU」、「NGC」のいずれかを指定します。

生成されたこのオブジェクト ファイルはプロジェクトにリンクすることができ、ソースは、-b2e オプションで使用されたラベルからの参照が可能になります。

-b2e (--bin2elf) オプションでは、アライメント値とエンド ラベルの 2 つの追加引数を使用できます。この引数は、サイズ ラベルの後にカンマで区切って使用します。

たとえば、バイナリ ファイルを ELF オブジェクト ファイルに変換し、アライメント値に 16 を指定する場合は、以下のように記述します。

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

エンド ラベルも指定する場合は、以下のように記述します。

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

エンド ラベルを指定する場合は、アライメント値も指定する必要があります。

ELF のブランク化 (-be, --blank-elf)

-be (--blank-elf) 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できます。

コマンドライン構文は、以下のようになります。

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

簡潔化 (-c, --concise)

一部のダンプ処理では、実行時に大量のデータが出力されます。このオプションでは出力を制限できますが、情報も省略されます。

現在これに対応するオプションは、「シンボル テーブルのダンプ」、「デマングル」、「逆アセンブリ出力」オプションのみとなります。このオプションを使用する場合は、従来のコマンド行の後に追加します (以下を参照)。

```
>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
```

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がエミュレートされます。詳細は、http://www.gnu.org/software/binutils/manual/html_chapter/binutils_2.html を参照してください。

逆アセンブリ出力と並行して計算および出力されるパイプライン分析情報は、-c、--concise スイッチで無効化することができます。これにより、逆アセンブリ出力時間が最高 40% 削減可能です。これは、逆アセンブリ出力を実行するすべてのスイッチに影響を与えます。

セクションをコピー (-cs, --copy-section)

-cs (--copy-section) オプションでは、バイナリ セクションが出力ファイルにコピーされます。これは、オーバーレイを使用する際によく使用します。このオプションを使用する場合は、出力ファイル名、セクション名、入力ファイルを指定します (以下を参照)。

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

デマングル (-dem, --demangle)

-dem (--demangle) オプションでは、C++ の名前がデマングル (デコード) されます (以下を参照)。

```
>ps3bin --demangle=__0fEfredEblahi

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

アドレス範囲の逆アセンブル (--disassemble-ranges)

--disassemble-ranges オプション (短縮形なし) では、アドレス範囲内 (<low_address..high_address> 形式における範囲) のコードが逆アセンブルされます。

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

これにより、0x010250 と 0x010300 の間、および 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が出力されます。.

シンボルの逆アセンブル (--disassemble-symbol)

--disassemble-symbol オプションでは、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。以下はその例です。

```
>sns3in --disassemble-symbol=.my_function_symbol(int, float)
```

マングル化されたシンボル名を逆アセンブルする場合は、-nd フラグを使用します。以下はその例です。

```
>sns3in --disassemble-symbol=._Z18my_function_symbolif -nd
```

ELF ヘッダのダンプ (-dh, --dump-elf-header)

-dh (--dump-elf-header) オプションでは、ファイルの ELF ヘッダをダンプできます (以下を参照)。

```
>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
```

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

-dml (--dump-mem-layout) オプションでは、ELF ファイル内のセクションの仮想アドレス ビューがプリントされます (以下を参照)。

```
>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
```

セクションをダンプ (-ds, --dump-sections)

-ds (--dump-sections) オプションでは、指定したセクションの内容がプリントされます。このプログラムでは、セクションがデコードされ、人間が理解できる形式でプリントされます。このオプションを使用する場合は、セクション名と入力ファイルを指定する必要があります (以下を参照)。

```
>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
```

指定したセクションをデコードできない場合は、代わりに 16 進数のダンプが表示されます。またカ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。

サイズをダンプ (-dsi, --dump-sizes)

-dsi (--dump-sizes) オプションでは、入力ファイルの各種コンポーネントのサイズがプリントされます (以下を参照)。

```
>ps3bin -dsi debug.elf

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

セクション名の変更 (-rs, --rename-sections)

-rs (--rename-sections) オプションでは、指定したセクションの名前を変更できます。このオプションを使用する場合は、現在のセクション名、新しいセクション名、入力ファイル、出力ファイル名を指定します (以下を参照)。

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

セクションの削除 (-sse, --strip-sections)

-sse (--strip-sections) オプションでは、入力ファイルの指定セクションを削除できます。このオプションを使用する場合は、セクション名、入力ファイル、出力ファイル名を指定します (以下を参照)。

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

複数のセクションを指定して削除する場合は、以下のようにカンマで区切ります。

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

シンボルの削除 (-ssy, --strip-symbols)

-ssy (--strip-symbols) オプションでは、入力ファイルからシンボルを削除できます。このオプションを使用する場合は、シンボル名、入力ファイル、出力ファイル名を指定します (以下を参照)。

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

複数のシンボルを指定して削除する場合は、以下のようにカンマで区切ります。

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

詳細情報 (-v, --verbose)

一部のダンプ処理では、実行時に、最低限のデータのみが出力されます。これは読みやすさを目的としているためですが、完全な出力が必要な場合は、-v (--verbose) オプションを使用します。現時点では、「シンボル テーブルのダンプ」オプションと「セクション ヘッダのダンプ」オプションのみがこの機能に対応しています。このオプションを使用する場合は、従来のコマンド行の後に追加します (以下を参照)。

```
>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 モード (-gnu, --gnu-mode)

-gnu (--gnu-mode) オプションでは、GNU binutil 出力スタイルに合うように処理の出力形式を整えます。このオプションは現在、addr2line にのみ対応しています。

例::

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

foo(int, float)

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

2: インデックス