NAME

PDFJ - 日本語 PDF 生成モジュール

SYNOPSIS

```
use PDFJ qw(SJIS);
$doc = PDFJ::Doc->new($pdfversion, $paperwidth, $paperheight);
$font = $doc->new_font('Ryumin-Light', '90ms-RKSJ-H', 'Times-Roman');
$page = $doc->new_page;
$text = Text("テキスト", TStyle(font => $font, fontsize => 10));
$paragraph = Paragraph($text, PStyle(size => 100, align => 'w', linefeed => 20));
$image = $doc->new_image($jpgfile, $pixelwidth, $pixelheight, $width, $height);
$shape = Shape->ellipse($x, $y, $rx, $ry);
$block = Block('V', $paragraph, $image, $shape, BStyle(align => 'c'));
$block->show($page, $x, $y);
$doc->print('sample.pdf');
```

DESCRIPTION

■概要

このモジュールは日本語 PDF を生成する。次のような特徴がある。

- JIS X 4051「日本語文書の行組版方法」（1995）にほぼ準拠した行組版ルールを組み込んであり、禁則や行の詰め伸ばしはこのモジュールに任せることができる。
- ルビ、添え字、縦書き中の欧文、縦中横、欧文のハイフネーション、下線･傍線、圏点、網掛けといった組版処理もこのモジュールに任せることができる。
- Type1 フォントでは、和文に Ryumin-Light と GothicBBB-Medium、欧文に Times、Helvetica、Courier の各ファミリが使える。これらはフォント自体は埋め込まれないので、コンパクトな PDF を作れる。ただし表示・印刷環境にそのフォントがないと代替フォントとなる。
- 任意の TrueType フォントを使うこともできる。TrueType フォントは埋め込まれる（和文についてはサブセットで）ので、若干 PDF のサイズが大きくなるが、どんな環境でも同じように表示･印刷できる。
- 欧文に、固定ピッチの半角フォントを使うことも、プロポーショナルな欧文フォントを使うこともできる。
- 日本語文字コードとしては、シフト JIS と日本語 EUC に対応している。
- JPEG 画像（ファイルおよび URL 指定）と線画図形が扱える。画像や図形をテキストの一部として行内に配置することも可能。逆に線画図形の中にテキストや画像を配置することもできる。
- テキストを行長と行送りを指定して折り返し処理し､段落を作ることができる。段落には箇条書き

のためのラベルを付けることができる
- 段落、画像、図形などを並べてブロックというまとまりを作ることができる。ブロックには、内容の配置、周囲の余白、枠線、塗りつぶし色などを指定できる。ブロック内の並びの方向として、上→下、左→右、右→左がある。ブロックを入れ子にすることで表を作ることができる。
- 段落やブロックを指定の大きさを超えないように分割して、複数のページに分けて表示することができる。
- PDF のアウトライン情報や、ハイパーリンク（文書内および URL）を付加できる。

■表示可能なオブジェクト

PDFJ では次の表示可能な構成要素に対応するオブジェクトを組み合わせて PDF 文書を作成する。これらはみな、show というメソッドでページ上に位置を指定して表示することができる。show メソッドの具体的な使い方については後述する。

テキスト（PDFJ::Text）

指定の文字列を、指定のフォントやサイズなどの属性に従って表示するもの。フォントのエンコーディングが H であれば左から右へ横書きで、V であれば上から下へ縦書きで表示される。
ルビ、添え字、縦中横、下線・傍線、圏点、囲みといった属性の指定ができる。
テキスト自体には行長や行送りといった属性はなく、折り返して表示されることはない。
文字だけでなく、表示可能なオブジェクトを含むことができる。

段落（PDFJ::Paragraph）

テキストに対して行長と行送りと配置を指定して行の折り返しをおこない、ひとつの段落として表示するもの。行の折り返しに伴う、禁則処理、ハイフネーション、行の詰め伸ばしは自動的に処理される。
またテキストには、文字だけでなく画像や図形をひとつの文字のように扱って含むこともできる。
行頭、行末のインデント、先頭行につけるラベルを指定することもできる。
段落の前後の間隔を指定することができる。この間隔は段落を並べてブロックを作る際に適用される。

画像（PDFJ::Image）

JPEG 形式で、ファイルに保存されているものか、URL で参照できる画像のみが扱える。元のピクセルサイズとは関係なく指定の大きさで表示できる。
周囲の余白を指定することもできる。

図形（PDFJ::Shape）

直線、矩形、多角形、円、楕円、ベジエ曲線を組み合わせて図形を作成し、表示できる。線の有無、太さ、色、点線、塗りつぶしの有無、色といった属性が指定できる。
図形内にテキストを配置することもできる。
周囲の余白を指定することもできる。

ブロック（PDFJ::Block）

表示可能なオブジェクトを特定の方向に並べてひとまとめにしたもの。方向としては、H（左→右）、R（右→左）、V（上→下）の三種類がある。全体の幅や高さを指定して内容の配置を指定することもできる。内容の配置は、左右方向に、l（左）、c（中央）、r（右）、上下方向に、t（上）、m（中央）、b（下）を組み合わせて指定する。ただし、全体の幅や高さは内容によって決まる幅や高さより小さくはできない。
オブジェクトに前後の間隔の指定があれば、それに従って間隔が空けられる。また、直接数値で間

隔を指定することもできる。
ブロックには、周囲の余白、枠線、塗りつぶし色を指定することができる。
ブロックを入れ子にすることで、表を作ることができる。

■その他のオブジェクト

その他に次のようなオブジェクトが、表示可能なオブジェクトとともに使用される。

フォント（PDFJ::AFont、PDFJ::JFont）
フォントは Type1 フォントでは、和文に Ryumin-Light と GothicBBB-Medium、欧文に Times、Helvetica、Courier の各ファミリが使える。TrueType フォントは任意のものが使える。ただし PDF に埋め込まれるので、埋め込みが許可された TrueType フォントでなければならない。
和文フォントだけを指定したテキストに欧文が現れたときには、和文フォントの半角文字（文字幅は半角固定）が使われるが、組み合わせる欧文フォントを指定しておくとその欧文フォントが使われる。プロポーショナルな和文フォントには対応していない。
縦書き用エンコーディング（V）を指定した和文フォントを指定すると、そのテキストは縦書きとなる。

テキストスタイル（PDFJ::TextStyle）
フォント、フォントサイズ、文字描画モード、ベースライン調整、斜体、下線（縦書きでは傍線）、囲み箱、圏点、添え字（上・下）、ルビ、傍注、図形スタイル、が指定できる。
文字描画モードは、文字の枠線と塗りつぶしの組み合わせの指定。図形スタイルは、文字描画、下線・傍線、囲み箱における図形スタイルの指定。

段落スタイル（PDFJ::ParagraphStyle）
行長、揃え、行送り、ラベル、ラベル長、行頭インデント、行末インデント、前間隔、後間隔、が指定できる。
揃えとしては、b（行頭揃え）、m（中央揃え）、e（行末揃え）、w（両端揃え）がある。

ブロックスタイル（PDFJ::BlockStyle）
幅、高さ、揃え、揃えフラグ、周囲余白、枠線、塗りつぶし色、前間隔、後間隔、が指定できる。

図形スタイル（PDFJ::ShapeStyle）
線幅、点線、線色、塗りつぶし色、周囲余白、前間隔、後間隔、が指定できる。

色（PDFJ::Color）
灰色指定と、ＲＧＢ指定ができる。図形属性の線色と塗りつぶし色の指定に使われる。

ページ（PDFJ::Page）
各ページの内容を保持する。テキストや画像や図形など､表示可能なオブジェクトはページに配置することで実際に表示される。

PDF **文書**（PDFJ::Doc）
一つの PDF 文書。ページ群や、リソースとしてのフォントや画像をまとめ、最終的に一つの PDF ファイルとして出力する。

■ PDFJ のインストール

管理者であれば次の標準的な手順でインストールできる。

```
perl Makefile.PL
make
make install
```

最後の make install は管理者権限で実行する。Windows では make でなく nmake を使用する。
管理者でない場合でも、PDFJ を構成する次のモジュールファイル群を Perl から利用できる（すなわち @INC にセットされた）ディレクトリにおけば利用できる。

```
PDFJ.pm
PDFJ/Object.pm
PDFJ/Unicode.pm
PDFJ/E2U.pm
PDFJ/S2U.pm
PDFJ/U2C.pm
PDFJ/TTF.pm
```

PDFJ は、欧文のハイフネーションをおこなうために、TeX::Hyphen モジュールを使用している。欧文を含むテキストを扱う場合は必要となるので、CPAN からダウンロードしてインストールしておく。管理者でない場合は、次のモジュールを Perl から利用できるディレクトリにおけばよい。

```
TeX/Hyphen.pm
TeX/Hyphen/czech.pm
TeX/Hyphen/german.pm
```

PDFJ は、フォントや画像などのデータを埋め込む際に、デフォルトでは Compress::Zlib モジュールを使用する。Compress::Zlib がない環境や、Compress::Zlib を使いたくない場合のために、Compress::Zlib を使わずにデータの埋め込みをおこなうオプションも用意されている。（文書オブジェクトの作成を参照）

■ PDFJ の使用

PDFJ を使用するには、つぎのようにして use PDFJ の引数に日本語文字コードを指定する。省略すると 'SJIS' とみなされる。

```
# Shift-JIS の場合
use PDFJ 'SJIS';

# EUC の場合
use PDFJ 'EUC';

# UTF8 の場合
use PDFJ 'UTF8';

# UNICODE（UCS2）の場合
use PDFJ 'UNICODE';
```

テキストオブジェクトを作る時に与える文字列や、フォントのエンコーディングでの日本語文字コードは、use PDFJ で指定したものと合致するようにしなければならない。
異なる文字コードを混在させたり切り替えて使用することはできない。
use PDFJ によって次のサブルーチンがエクスポートされる。

```
Doc
Text
```

```
TStyle
NewLine
Outline
Dest
Paragraph
PStyle
Block
BStyle
NewBlock
BlockSkip
Shape
SStyle
Color
```

■文書オブジェクトの作成

まず最初に文書オブジェクトを作成しなければならない。

```
$docobj = PDFJ::Doc->new($version, $width, $height);
```

ここで、$version は PDF のバージョン（※下記の注を参照）、$width はページの幅、$height はページの高さで、単位はポイント（1/72 インチ）である。（ポイントの定義は定まったものがないが、PDF では 1/72 インチとされている。1 インチは 25.4mm。）
なお、ページの幅と高さは、各ページオブジェクトを作成する時に個別に指定することもできる。
※ PDF のバージョン
PDF のバージョンは、次のように Acrobat や Acrobat Reader のバージョンと対応している。

```
PDF バージョン 1.2 … Acrobat バージョン 3
PDF バージョン 1.3 … Acrobat バージョン 4
PDF バージョン 1.4 … Acrobat バージョン 5
```

したがって Acrobat3 でも使えるようにしたければ 1.2 にしておく。ただし、日本語 TrueType フォントを使うときは 1.3 以上が必要。Acrobat4 以上で使えればよいということなら常に 1.3 にしておけばよい。
※データ埋め込み方法の指定
Compress::Zlib を使わずにデータの埋め込みをおこなう場合は、文書オブジェクトの filter メソッドを用いて次のようにデータ埋め込み方法の指定をおこなっておく。この指定をしない場合は Compress::Zlib を使った埋め込みがおこなわれる。

```
$docobj->filter('a');
```

■ページオブジェクトの作成

ページは文書オブジェクトから new_page メソッドで追加される。幅と高さを省略すると文書オブジェクトの作成の際に指定したものが使われる。

```
$pageobj = $docobj->new_page;
$pageobj = $docobj->new_page($width, $height);
```

今のところ、ページは末尾に追加できるだけで、途中に挿入する方法は用意されていない。
ページ番号は、pagenum メソッドで得られる。

```
$pagenum = $pageobj->pagenum;
```

■フォントオブジェクトの作成

フォントオブジェクトは、文書オブジェクトから、new_font メソッドで作られる。

```
$fontobj = $docobj->new_font($basefont, $encoding);
```

$basefont はベースフォント名で、Type1 フォントの場合次のいずれかを指定する。

※欧文フォント

```
Courier
Courier-Bold
Courier-BoldOblique
Courier-Oblique
Helvetic
Helvetic-Bold
Helvetic-BoldOblique
Helvetic-Oblique
Times-Bold
Times-BoldItlic
Times-Itlic
Times-Romn
```

※日本語フォント

```
Ryumin-Light
GothicBBB-Medium
```

$basefont に TrueType フォントのファイル名（拡張子が .ttf）を指定することで、TrueType フォントを指定することができる。また、TrueTypeCollection フォント（拡張子が .ttc）の場合はその中の何番目（0 から数えて）のフォントを使うかをファイル名の後ろに「: 番号」として付加する。（例：「c:\windows\fonts\msgothic.ttc:0」）

※ TrueTypeCollection フォントは固定ピッチのフォントとプロポーショナルなフォントがセットになっていることが多いが、上記のようにして指定するのは固定ピッチの方でなければならない。付属のスクリプト ttcinfo.pl で TrueTypeCollection フォントに含まれるフォント名を調べることができる。プロポーショナルなフォントはフォント名に P が付加されていることが多い。

※ TrueType フォントは PDF に埋め込まれる（日本語フォントの場合はサブセットで）が、埋め込みを許可しない TrueType フォントも存在する。PDFJ は、フォント自体の中にある埋め込みを許可するかどうかのフラグを見て、OK かどうかを判断する。ただし、別のライセンスファイルなどで使用許諾条件が示されている場合もありうるので、フォント作成者の権利を侵害しないように十分注意していただきたい。

※ TrueType フォントを埋め込む際には、フォントファイル内にユニコードに対応した cmap テーブル（platformID が 3、platformSpecificID が 1、format が 4 のもの）が必要である。古い TrueType フォントではこの cmap テーブルを持たないものも存在する。現状ではそういう TrueType フォントは埋め込むことができない。

$encoding はエンコーディングで、次のいずれかの定義済みエンコーディング名を指定する。省略すると、欧文フォントに対しては'WinAnsiEncoding'、日本語フォントに対しては'90ms-RKSJ-H' が使われる。

※欧文フォントのエンコーディング

```
WinAnsiEncoding
MacRomanEncoding
MacExpertEncoding
```

```
※日本語フォントのエンコーディング
83pv-RKSJ-H … Macintosh JIS X 0208 KanjiTalk6 拡張
90pv-RKSJ-H … Macintosh JIS X 0208 KanjiTalk7 拡張
90ms-RKSJ-H … Microsoft CP932 JIS X 0208 NEC,IBM 拡張
90ms-RKSJ-V … 〃縦書き
Add-RKSJ-H  … JIS X 0208 富士通 FMR 拡張
Add-RKSJ-V  … 〃縦書き
Ext-RKSJ-H  … JIS C 6226(JIS78) NEC 拡張
Ext-RKSJ-V  … 〃縦書き
EUC-H       … JIS X 0208
EUC-V       … 〃縦書き
UniJIS-UCS2-HW-H … Unicode 横書き
UniJIS-UCS2-HW-V … Unicode 縦書き
```

日本語フォントのエンコーディングの末尾の 'H' は横書き､'V' は縦書き。'RKSJ' とつくものは Shift-JIS 用、'EUC' とつくものは EUC 用、'Uni' とつくものは Unicode 用。日本語プロポーショナルフォントは使えないことに注意。欧文部分も含めてすべての文字が全角か半角の固定ピッチとなる。
※ UTF8 の場合もフォントエンコーディングには上記の Unicode 用を指定すればよい。
new_font メソッドにはもう一つの用法があり、つぎのようにして日本語フォントと欧文フォントの組を指定する。

```
$fontobj = $docobj->new_font($jbasefont, $jencoding, $abasefont, $aencoding);
```

ここで、$jbasefont は日本語ベースフォント名、$jencoding はそのエンコーディング、$abasefont は欧文ベースフォント名、$aencoding はそのエンコーディング。$aencoding を省略すると WinAnsiEncoding。
このように日本語フォントと欧文フォントを組み合わせたフォントオブジェクトをテキストに対して指定すると、テキスト中の日本語部分と欧文部分に対してそれぞれのフォントが自動的に切り替えて適用される。これにより、欧文部分についてはプロポーショナルな表示となる。

■テキストオブジェクトの作成

テキストオブジェクトは、Text サブルーチンで作成する。

```
$textobj = Text(@list, $textstyle);
```

ここで、@list は文字列、表示可能なオブジェクト、改行オブジェクトのリストで、@list の要素が順に並べられた内容のテキストが作成される。$textstyle はテキストスタイルオブジェクト。
リストは配列参照の形で与えることもできる。

```
$textobj = Text([@list], $textstyle);
```

改行オブジェクトは NewLine サブルーチンで作成する(引数無し)。改行オブジェクトはテキストオブジェクトをそのまま表示する場合には何の効果もないが、段落オブジェクトを作る際に強制改行する効果をもたらす。
テキストスタイルオブジェクトは TStyle サブルーチンで作成する。

```
$textstyle = TStyle(%args);
```

引数にはハッシュリストの形で次のものを与える。font と fontsize は必須。その他はオプション。

font => フォントオブジェクト
fontsize => フォントサイズ（ポイント）
italic => イタリックフラグ（真を指定するとイタリックに）
bold => ボールドフラグ（真を指定するとボールドに）
slant => 斜体フラグ（真を指定すると斜体に）
render => 文字描画モード（0: 塗り潰し、1: 枠線、2: 塗り潰し＋枠線）
shapestyle => 文字描画の図形スタイルオブジェクト
rise => ベースラインの上調整値（ポイント）
vh => 縦中横フラグ（真を指定すると縦中横に）
withline => 下線または傍線フラグ（真を指定すると下線または傍線が付く）
withlinestyle => 下線または傍線の図形スタイルオブジェクト
withbox => 囲み枠指定（f: 塗り潰し、s: 枠線、sf: 塗り潰し＋枠線）
withboxstyle => 囲み枠の図形スタイルオブジェクト
withdot => 圏点フラグ（真を指定すると圏点が付く）
withnote => 注釈テキストオブジェクト
suffix => 添え字指定（'u' を指定すると上添え字、'l' を指定すると下添え字）
ruby => ルビ文字列
objalign => 表示可能オブジェクトの配置

slant による斜体は日本語文字列にのみ有効であり、欧文については Italic 系のフォントを指定することでおこなうこと。italic と bold については、イタリックとボールドを参照。
render や shapestyle の指定をしないと、文字は黒の塗り潰しで描画される。
withline を指定して withlinestyle を省略すると黒の実線となる。withbox を指定して withboxstyle を省略すると黒の実線となる。
withnote は文字の上や右に別のテキスト（そのテキストオブジェクトを withnote で指定する）を表示するものであり、suffix は指定した文字を小さくして位置を上下させる命令である。
※ withnote に文字列を与え、withnotestyle にテキストスタイルオブジェクトを与えることもできる。
objalign は、Text に画像や図形などの表示可能オブジェクトが与えられたときにどう配置するかをつぎのように指定する。objalign の指定を省略すると、横書きでは 'b'、縦書きでは 'c' とみなされる。

※横書きの場合（上下方向の配置の指定となる）
t … 文字とオブジェクトの上端をあわせる
m … 文字とオブジェクトの上下中央をあわせる
b … 文字とオブジェクトの下端をあわせる

※縦書きの場合（左右方向の配置の指定となる）
l … 文字とオブジェクトの左端をあわせる
c … 文字とオブジェクトの左右中央をあわせる
r … 文字とオブジェクトの右端をあわせる

■テキストオブジェクトの入れ子による部分スタイル指定

テキストの一部分だけに特定のスタイルを適用したい場合、テキストオブジェクトを入れ子にして部分スタイルを指定することでおこなう。入れ子になったテキストオブジェクトでは子のスタイルで指定されていないスタイルは親のものが引き継がれる。
例えば、明朝のテキストの一部をゴシックにしたい場合、つぎのようにする。

```
$mincho = $docobj->new_font('Ryumin-Light', '90ms-RKSJ-H');
$gothic = $docobj->new_font('GothicBBB-Medium', '90ms-RKSJ-H');
```

```
$textobj = Text([
    "明朝",
    Text("ゴシック", TStyle(font => $gothic)),
    "ここも明朝"
  ], TStyle(font => $mincho, fontsize => 10));
```

この場合、"ゴシック" に対するスタイルでは fontsize が指定されていないので、親スタイルの fontsize の 10 が引き継がれる。
テキストの一部に下線を引く場合は、例えばつぎのようにする。

```
$mincho = $docobj->new_font('Ryumin-Light', '90ms-RKSJ-H');
$normal_style = TStyle(font => $mincho, fontsize => 10);
$uline_style = TStyle(withline => 1);
$textobj = Text([
    "テキスト",
    Text("下線付き", $uline_style),
  ], $normal_style);
```

テキストスタイルはこのように変数にセットしておいて使うこともできるし、先の例のように直接 TStyle サブルーチンを使ってもよい。

■イタリックとボールド

テキストスタイルの italic と bold を使うためには、どのフォントがどのフォントのイタリック形やボールド形である、ということをドキュメントオブジェクトに教えておいてやる必要がある。そのために、italic() と bold() メソッドを使う。例えば次のようにする。

```
$ft = $docobj->new_font('Times-Roman');
$fti = $docobj->new_font('Times-Italic');
$ftb = $docobj->new_font('Times-Bold');
$ftbi = $docobj->new_font('Times-BoldItalic');
$docobj->italic($ft, $fti, $ftb, $ftbi);
$docobj->bold($ft, $ftb, $fti, $ftbi);
```

このように、元フォント、その修飾フォント、の順で、二組以上をまとめて引数に与えることができる。組となるフォントは、欧文フォント同士、日本語フォント同士、欧文フォントと組になった日本語フォント同士、でなければならない。

```
Text('normal', Text('italic', TStyle(italic => 1)), TStyle(font => $ft))
```

このようなテキストオブジェクトを作ると、'normal' には $ft が、'italic' には $fti が使われることになる。italic() や bold() での登録がされていないフォントに対して italic や bold のスタイルを与えた場合は、何の効果ももたらさない。
なお、一般に日本語フォントにはイタリック形は存在しないので、日本語文字列に対してテキストスタイルで italic が指定された場合は、slant に置き換えて傾けて表示する。

■段落オブジェクトの作成

段落オブジェクトは Paragraph サブルーチンで作成する。

```
$paragraphobj = Paragraph($textobj, $parastyle);
```

ここで、$textobj はテキストオブジェクト、$parastyle は段落スタイルオブジェクト。複数のテキストを

与えたいときはそれを一つのテキストオブジェクトにまとめた上で与える。
段落スタイルオブジェクトは PStyle サブルーチンで作成する。

```
$parastyle = PStyle(%args);
```

引数にはハッシュリストの形で次のものを与える。size と linefeed と align は必須。他はオプション。

```
size => 段落の行方向のサイズ（ポイント）
align => 揃え（b: 行頭揃え m: 中央揃え e: 行末揃え w: 両端揃え W: 強制両端揃え）
linefeed => 行送り（ポイント）
preskip => 段落前の間隔（ポイント）
postskip => 段落後の間隔（ポイント）
beginindent => 行頭インデント
endindent => 行末インデント
beginpadding => 行頭側の余白（ポイント）
labeltext => ラベルのテキストオブジェクト
labelsize => ラベルの行方向のサイズ（ポイント）
labelskip => ラベルと本文の間隔（ポイント）
nobreak => 真だと break メソッドで分割されない
postnobreak => 真だとブロックの break でその後ろで分割されない
float => ブロックの break で、b: 先頭、e: 末尾、h: 可能ならその位置、に移動
```

linefeed で指定するのは行送りであって行間ではないことに注意。linefeed => '150%' のように 数値 % と指定すると、テキストオブジェクトのフォントサイズに対する割合とみなされる。
preskip と postskip は、ブロック内に段落を並べる時の間隔として使われる。省略すると、それぞれ行間（行送りからフォントサイズを差し引いた長さ）の半分にセットされる。
テキストは、(size - beginpadding - beginindent - endindent - labelsize) という行長を超えないように折り返し処理され、align にしたがって揃えられる。w による両端揃えの時、末尾行だけは行頭揃えとなる。W による強制両端揃えでは、末尾行も含めて両端揃えとなる。行の折り返しの際の禁則とハイフネーション、両端揃えの際の詰め伸ばしは、JIS X 4051 にほぼ則っておこなわれる。
beginindent と endindent を配列参照の形で与えると、先頭行から順に使われる。行数が要素数より大きいときは最後の要素が繰り返し使われる。
beginpadding は、ラベルも含めた段落全体の、行頭側に取る余白を指定する。（この余白は size の中に含まれる。）したがって、ラベルがないときは、beginpadding の指定と単独要素の beginindent の指定は同じ効果を持つ。
labeltext でテキストオブジェクトが指定されると、ラベルとして先頭行の前に表示される。ラベルと本文の間には labelskip だけの間隔が取られる。
labeltext には、テキストオブジェクトを返すサブルーチン参照と、そのサブルーチンに与える引数のリストを、配列参照の形で与えることもできる。これによって番号付き箇条書きが実現できる。例えばつぎのようになる。

```
$LabelNum = 1;
sub numlabel {
  my($fmt, $style) = @_;
  Text(sprintf($fmt, $LabelNum++), $style);
}
$ol_style = PStyle(size => 500, align => 'w', labelsize => 30,
  labeltext => [\&numlabel, "%d.", $normal_style]);
$para1 = Paragaph($text1, $ol_style);
$para2 = Paragaph($text2, $ol_style);
```

postnobreak と float は、この段落を含むブロックが break されるときに意味を持つ。

■段落オブジェクトの分割

段落の行方向の大きさは段落スタイルの size で指定したものになるが、それと垂直な方向の大きさは行数（と行送りとフォントサイズ）で決まる。これが一定の大きさになるように段落を分割するために、break メソッドが用意されている。例えば横書きの段落オブジェクト $para に対して、

```
@paras = $para->break(200);
```

とすると、高さが 200 ポイント以下になるように分割した段落のリストが得られる。もし、最初の段落だけは高さを 100 以下にしたければ、つぎのようにすればよい。

```
@paras = $para->break(100, 200);
```

break の引数に指定したサイズのリストは順に分割する段落のサイズとして使われ、なくなると最後のサイズが繰り返し使われる。
break の引数に指定したサイズが小さすぎて、最後のサイズでも分割できない部分が残ったときは、分割に失敗したものとして未定義値が返される。
もし、$para->break(5, 200) のように最初や途中にフォントサイズより小さなサイズを指定すると、それに対応して空の段落オブジェクトが得られる。この例では、最初に空の段落オブジェクト、続いて 200 ずつに分割された段落オブジェクトが返されることになる。
段落スタイルの nobreak が真に設定されていると、分割されない。例えば nobreak な横書き段落オブジェクト $upara があり、その高さが 150 であるときに、$upara->break(100, 200) は (空段落オブジェクト , $upara と同じオブジェクト) を返す。$upara->break(100) では分割に失敗して未定義値を返す。

■ブロックオブジェクトの作成

ブロックオブジェクトは Block サブルーチンで作成する。

```
$blockobj = Block($direction, @objlit, $blockstyle);
```

ここで、$direction はブロックの内容を並べる方向、@objlit は表示可能なオブジェクトのリスト、$blockstyle はブロックスタイルオブジェクト。
オブジェクトのリストはつぎのように配列参照の形で与えることもできる。

```
$blockobj = Block($direction, [@objlit], $blockstyle);
```

ブロックの内容を並べる方向は、次のいずれかを指定する。

```
H … 左から右
R … 右から左
V … 上から下
```

$direction で指定した方向に、@objlit の要素を順に並べたものがブロックの内容となる。その際、各要素の表示可能なオブジェクトに、preskip や postskip のスタイルが指定されていれば、それにしたがって間隔が取られる。また特殊な指定として、@objlist の中にオブジェクトでなく生の数値を入れるとその間隔が取られる。
ブロックスタイルオブジェクトは BStyle サブルーチンで作成する。

```
$blockstyle = BStyle(%args);
```

引数にはハッシュリストの形で次のものを与える。

```
width => 幅（ポイント）
```

height => 高さ（ポイント）
align => 揃え
adjust => 揃えフラグ（真を指定すると要素のブロックのサイズを揃える）
padding => 周囲余白（ポイント）
beginpadding => 行頭側の余白（ポイント）
withbox => 囲み枠指定（f: 塗り潰し、s: 枠線、sf: 塗り潰し＋枠線）
withboxstyle => 囲み枠の図形スタイルオブジェクト
preskip => 前の間隔（ポイント）
postskip => 後の間隔（ポイント）
nobreak => 真だと break で分割されない
postnobreak => 真だとブロックの break でその後ろで分割されない
repeatheader => break で分割するとき先頭で繰り返す要素数
float => ブロックの break で、b: 先頭、e: 末尾、h: 可能ならその位置、に移動
nofirstfloat => 真だと break で先頭には float 要素を置かない

width、height で幅や高さを指定した場合、内容の幅や高さがそれより大きい場合は内容に合わせられる。内容よりも指定した幅や高さが大きい場合は、align にしたがって位置が揃えられる。
align による揃えは次のいずれかを組み合わせて指定。省略すると 'tl' とみなされる。

※左右方向
l … 左寄せ
c … 中央寄せ
r … 右寄せ

※上下方向
t … 上寄せ
m … 中央寄せ
b … 下寄せ

adjust を真に指定すると、方向が H や R の場合は要素ブロックの高さ、V の場合は要素ブロックの幅を、もっとも大きいものに揃える。ブロックを並べて表を作成するときに使用する。
padding は内容の周りに取られる余白の幅であり、withbox で囲み枠を指定した場合はその余白の外側に描画される。
beginpadding は囲み枠の外の行頭側に取られる余白の指定である。
withbox では、's','f','sf' の他に、'rX'（X は数値）を付加すると角が半径 X で丸くなる。

■ブロックオブジェクトの分割

段落オブジェクトと同様に、ブロックオブジェクトも break メソッドによって分割ができる。例えば方向が 'V' のブロックオブジェクト $block を高さが 200 ポイント以下になるように分割したければ、

```
@blocks = $block->break(200);
```

とすればよい。最初のブロックだけ高さを 100 以下にしたければ、つぎのようにする。

```
@blocks = $block->break(100, 200);
```

break の引数に指定したサイズのリストは順に分割した段落のサイズとして使われ、なくなると最後のサイズが繰り返し使われる。
ブロックの分割の際、ブロック内のオブジェクトとして、方向の同じブロックや、行方向の異なる段落（'V' なら 'H'、'H' や 'R' なら 'V'）があると、そのオブジェクトも分割することでできるだけ指定のサイ

ズに合うように分割される。
指定したサイズで分割ができなかった場合、break メソッドは未定義値を返す。複数のサイズを指定した場合に、最後以外のサイズが小さすぎた場合は、それに対応するものとして空のブロックが返される。
ブロックスタイルで nobreak が真になっていると、そのブロックは分割されない。
ブロックの中に NewBlock サブルーチンで作成される改ブロックオブジェクトがあると、ブロックの分割の際にそこで強制的に分割される。
ブロックの分割の際、その要素となるブロックや段落や図形のスタイルで postnobreak が真に設定されていると、その後ろで分割されることはない。これによって、見出し段落と本文段落が別ページに分かれることを防ぐことができる。
ブロックの分割の際、その要素となるブロックや段落や図形のスタイルで float が設定されていると、その要素は分割されたブロック内で指定の位置に移動される。float 指定の意味は次のとおり。

b: 分割されたブロックの先頭
e: 分割されたブロックの末尾
h: 分割せずに可能ならその位置に、無理なら次のブロックの先頭に

ブロックスタイルで nofirstfloat を真に指定すると、break の際に全体の先頭には float 要素を置かない。
ブロックスタイルで repeatheader が指定されていると、その値の数だけの先頭要素が、分割された各ブロックの先頭で繰り返される。（ただし先頭要素の途中や後ろで分割された場合は除く。）これによって表の先頭の項目名の行を繰り返すことができる。先頭要素自体が分割されるとおかしな結果が得られるので、先頭要素が分割可能な段落やブロックの場合は nobreak を指定しておくこと（方向の違うブロックの場合は不要）。また先頭要素と次の要素が分割された場合も不適切な結果となるので、先頭要素には postnobreak の指定をしておくこと。

■画像オブジェクトの作成

画像オブジェクトは JPEG 画像についてのみ作成でき、文書オブジェクトから new_image メソッドで作成する。

```
$imgobj = $docobj->new_image($src, $pxwidth, $pxheight,
  $width, $height, $padding, $colorspace);
```

ここで、$src は URL またはファイル名（拡張子は .jpg または .jpeg であること）、$pxwidth と $pxheight は画像のピクセルサイズ、$width と $height は表示サイズ（ポイント）、$padding は周囲の余白（ポイント）、$colorspace はカラースペース（rgb,gray,cmyk のいずれかで省略すると rgb）。$padding と $colorspace は省略できる。
現在の仕様では、URL 指定した場合は生成される PDF には URL 情報だけが埋め込まれ、表示する際に AcrobatReader がその URL にアクセスして画像内容を読みとる。したがって表示に時間がかかったり、アクセスできないと画像が表示できないといったことが起こりうる。
ファイル名指定した画像の場合は、生成された PDF に画像内容そのものがデータとして埋め込まれるので、元の画像ファイルを PDF ファイルと一緒に配布したりする必要はない。
画像オブジェクトには他の表示可能なオブジェクトと違ってスタイルの指定はない。ブロックに含める際に postnobreak を指定したいというようなときは、図形オブジェクトの中に画像オブジェクトを含めて、その図形オブジェクトにスタイルを指定する。

■図形オブジェクトの作成

図形オブジェクトは Shape サブルーチンで作成する。

```
$shapeobj = Shape($shapestyle);
```

ここで $shapestyle は図形スタイルオブジェクト。$shapestyle は省略できる。これだけでは何も中味のない図形オブジェクトが作られるだけである。その後、次のメソッドを使って図形を加えていく。

直線（line メソッド）

```
$shapeobj->line($x, $y, $w, $h, $style);
```

($x,$y) から ($x+$w,$y+$h) へ直線が引かれる。$style は図形スタイルオブジェクトで、省略可能。

矩形（box メソッド）

```
$shapeobj->box($x, $y, $w, $h, $spec, $style);
```

($x,$y) と ($x+$w,$y+$h) を対角とする矩形が描かれる。$style は図形スタイルオブジェクトで、省略可能。$spec は次の描画指定。

f … 塗り潰しのみ
s … 枠線のみ
sf … 塗り潰し＋枠線
n … 描画しない
※上記の s の代わりに、l（左辺）、r（右辺）、t（上辺）、b（下辺）、の組み合わせも可
※次はオプション
rX … （X は数値）角を半径 X で丸くする

box() に対する $style では、通常の図形スタイルに加えて次のスタイルが使える。（ハイパーリンクを参照）

link => リンク先（文書内のリンク先名または、URI: を付けた URI 名）

円（circle メソッド）

```
$shapeobj->circle($x, $y, $r, $spec, $arcarea, $style);
```

($x,$y) が中心、$r が半径の円が描かれる。$arcarea は四半円指定。$style は図形スタイルオブジェクトで、省略可能。$spec は次の描画指定。

f … 塗り潰しのみ
s … 枠線のみ
sf … 塗り潰し＋枠線

楕円（ellipse メソッド）

```
$shapeobj->ellipse($x, $y, $xr, $yr, $spec, $arcarea, $style);
```

$xr が横半径、$yr が縦半径であることを除けば円と同じ。

多角形（polygon メソッド）

```
$shapeobj->polygon([@coords], $spec, $style);
```

@coords は頂点の座標の X と Y の組を順に並べたリスト。$style は図形スタイルオブジェクトで、省略可能。$spec は次の描画指定。

f … 塗り潰しのみ
s … 枠線のみ
sf … 塗り潰し＋枠線

オブジェクト（obj メソッド）

```
$shapeobj->obj($obj, @showargs);
```

図形中に表示可能なオブジェクト $obj を、$obj->show($page, @showargs) によって配置する。

以上のメソッドはみなオブジェクト自身を返すので、

```
$shapeobj = Shape->line(…)->box(…)->obj(…);
```

のように記述することも可能。
以上のメソッドで描画する場合は、結果としてその図形オブジェクトが上下左右にどれだけの範囲を占めるかという全体としての図形の大きさが内部的に管理され、幅と高さを持った表示可能オブジェクトとして扱うことができる。
これら以外のプリミティブな描画メソッドもある（PDFJ::Shape のメソッド一覧を参照）が、それらのメソッドを使った場合は図形オブジェクトの大きさの管理はおこなわれないことに注意が必要。
図形スタイルオブジェクトは、SStyle サブルーチンで作成する。

```
$shapestyle = SStyle(%args);
```

引数にはハッシュリストの形で次のものを与える。

```
fillcolor => 塗り潰し色（色オブジェクト）
strokecolor => 線色（色オブジェクト）
linewidth => 線幅（ポイント）
linedash => [$dash, $gap, $phase] または "$dash, $gap, $phase"
preskip => 前の間隔（ポイント）
postskip => 後の間隔（ポイント）
postnobreak => 真だとブロックの break でその後ろで分割されない
float => ブロックの break で、b: 先頭、e: 末尾、h: 可能ならその位置、に移動
```

linedash の指定で、$dash は破線長、$gap は隙間長、$phase は開始位置。$phase は省略可能。
preskip、postskip、postnobreak、float は、ブロックの中に図形オブジェクトを置くときに意味を持つ。
box() に対する $style では、上記の図形スタイルに加えて次のスタイルが使える。

```
link => リンク先（文書内のリンク先名または、URI: を付けた URI 名）
```

■色オブジェクトの作成

色オブジェクトは、Color サブルーチンで作成する。

```
$colorobj = Color($r, $g, $b);
$colorobj = Color('#RRGGBB');
$colorobj = Color($g);
```

三引数の場合、$r は赤、$g は緑、$b は青のそれぞれの割合（0 から 1 までの範囲の数値）。
一引数で、# で始まる 16 進 6 桁の文字列の場合、二桁ずつ赤、緑、青の割合（00 から ff まで）とみなされる。
一引数で、数値の場合は、グレーの割合（0 から 1 までの範囲の数値）。0 が黒、1 が白。

■表示可能オブジェクトのページへの配置

表示可能なオブジェクトをページ上に配置するには、show メソッドを用いる。

```
$obj->show($page, $x, $y, $align, $transtype, @transargs);
```

ここで、$page はページオブジェクト、$x、$y は表示位置、$align は配置、$transtype は変形の種類、@transargs は変形のパラメータである。$align 以降の引数は省略できる。
表示位置の座標は、ページの左下隅が原点 (0,0) となり、X 座標は右へ、Y 座標は上へ向かって増加する。単位はポイントである。
配置 $align は、($x,$y) で指定した表示位置に対して、オブジェクトをどのように配置するかを指定するもので、次の横位置と縦位置を組み合わせて指定する。

横位置
l … オブジェクトの左端を $x にあわせる
c … オブジェクトの中央を $x にあわせる
r … オブジェクトの右端を $x にあわせる
縦位置
t … オブジェクトの上端を $y にあわせる
m … オブジェクトの中央を $y にあわせる
b … オブジェクトの下端を $y にあわせる

配置 $align の指定を省略すると、そのオブジェクト固有の原点を ($x,$y) にあわせる。各オブジェクトの固有の原点は次のとおり。

横書きテキスト … 先頭文字の左端の、下端から高さの 0.125 倍だけ上の位置
縦書きテキスト … 先頭文字の上端の、左右中央の位置
段落 … 先頭行テキストの固有の原点
ブロック … 左上隅
画像 … 左下隅
図形 … 描画命令の原点がそのまま原点となる

show メソッドに、$transtype 以降の引数を与えると、表示の際に変形することができる。変形の種類 $transtype とそのパラメータ @transargs には次のいずれかを指定する。

'magnify', $mx, $my … 横方向に $mx 倍、縦方向に $my 倍、拡大・縮小する
'rotate', $rad … 反時計回りに $rad ラジアンだけ回転する
'distort', $xtan, $ytan … (1,0) を (1,$xtan) へ、(0,1) を ($ytan,1) へ移すように、横軸、縦軸をそれぞれ傾ける

各変形は、show メソッドの $x と $y の引数で決まる位置を原点としておこなわれる。

■ページのレイヤ

show メソッドで表示可能オブジェクトをページに配置していくと、後から配置したものが手前に配置されて、前に配置されたものに重なっていく。
この重なりの順序を制御したい場合のために、ページオブジェクトに layer メソッドが用意されている。

```
$pageobj->layer($layernum);
```

$layernum はレイヤ番号で、0 以上の整数値。layer メソッドを実行すると、それ以降の描画は指定したレイヤ番号のレイヤに対しておこなわれる。
ページの内容が表示されるときには、レイヤ番号の順番に配置される。

■スタイルのコピー

直接入れ子になったテキストオブジェクトのテキストスタイルに関しては、親子関係による内容の継

承がおこなわれるので、部分スタイルの指定ができる。それ以外の場合にスタイルの自動的な継承がおこなわれることはない。
既存のスタイルを元にして属性を変更したり追加したりしたスタイルを作成したい場合、clone メソッドを用いる。このメソッドはテキストスタイル、段落スタイル、ブロックスタイル、図形スタイルのすべてについて使える。

```
$newstyle = $originalstyle->clone(%newargs);
```

%newargs を指定しなければ単にコピーが作られる。%newargs で指定した属性は元の属性を上書きする（元の属性がなければ追加される）。

■スタイル指定のさまざまな方法

各種のスタイルオブジェクトを作成したり、clone() でコピーしたりするときに、その引数はハッシュリストで与えるが、ハッシュ参照や css ライクな文字列で与えることもできる。次は同等である。

```
PStyle(size => 200, align => 'w', linefeed => '150%')
PStyle({size => 200, align => 'w', linefeed => '150%'})
PStyle('size:200; align:w; linefeed:150%')
```

文字列で与えるときは、「属性名 : 属性値」を「;」で区切って並べて指定する。ただし、ハッシュ参照や文字列で指定するときは引数はその一つだけでなければならない。
属性値としてオブジェクトを与えるときは文字列による方法は取れないが、属性値もスタイルオブジェクトである時には、次のように {} で指定が可能である。

```
TStyle('withline:1; withlinestyle:{linewidth:0.5; linedash:2,2; strokecolor:#ff0000}')
```

引数にハッシュリストを与えるときにも、属性値としてスタイルオブジェクトを取る属性（属性名が style で終わる）や、Color オブジェクトを取る属性（属性名が color で終わる）については、属性値をオブジェクトでなくハッシュ参照や文字列で与えることができる。

```
TStyle(withline => 1, withlinestyle =>
  {linewidth => 0.5, linedash => '2,2', strokecolor => '#ff0000'})
```

■ PDF 文書の出力

作成した PDF 文書をファイルに出力するには、文書オブジェクトの print メソッドを用いる。

```
$docobj->print($filename);
```

ファイル名 $filename の拡張子は、通常は .pdf とする。
ファイル名として '-' を指定すると標準出力に出力される。

■アウトライン

PDF にはアウトラインという目次機能がある（しおりとも言う）。アウトラインの項目をマウスでクリックするとその項目で指定された位置が表示される。アウトラインは階層的に構成され、章や節などの見出しをアウトラインに対応させることが多い。
PDFJ で文書にアウトラインを付加するには、テキストオブジェクトを作成する際に、対象の文字列やオブジェクトの前にアウトライン指示オブジェクトを置く。アウトライン指示オブジェクトは Outline() サブルーチンで作成する。例えば、「はじめに」という見出しをアウトラインのトップレベル

に加えたい場合、つぎのようにする。

```
Text(Outline('はじめに'),'はじめに',$midasi_style)
```

このテキストオブジェクトがページに配置されると、文書のアウトラインに「はじめに」という項目が作られてこの「はじめに」というテキストの左上の位置が指定される。この例ではアウトライン項目とテキストの文字列を同じにしているが、異なる文字列を指定してもよい。
１レベル下の「本書の内容」という見出しをアウトラインに加えたい場合、つぎのようにする。

```
Text(Outline('本書の内容',1),'本書の内容',$midasi_style)
```

Outline() の２番目の引数には、アウトラインの階層レベルを指定する。レベル 0 は上記の「はじめに」の例のように省略できる。
アウトラインは、Outline() を含んだテキストオブジェクトがページに配置されるときに順に追加されて作られていく。いまのところ、既存のアウトラインの途中に挿入する手段は用意されていない。レベル 0 の項目の次にレベル 2 の項目を作るなど、階層のギャップが生じると、ギャップを埋めるための空文字列によるアウトライン項目が作られる。

■ハイパーリンク

PDF にはハイパーリンク機能があり、ページ上のリンク元に指定された領域をクリックすると、そのリンク先が表示される。リンク先としては、同じ文書内の場所、別の文書の場所、URI（http: などで始まるインターネット上の場所と考えればよい）があるが、今のところ PDFJ では同じ文書内の場所と URI に対応している。
同じ文書内でのリンクを作るには、リンク先のテキストに Dest() サブルーチンで名前を指定して作成した PDFJ::Dest オブジェクトを配置する。例えば

```
Text(Dest('dest'),'リンク先',TStyle(…))
```

とすると、'リンク先' というテキストの前に 'dest' という名前のリンク先が作られる。Dest() で作られる PDFJ::Dest オブジェクト自体は、表示には現れない。リンク先の名前は任意の文字列が使えるが、「URI:」で始まるものは URI へのリンクのために使われる。
リンク元では矩形の図形オブジェクトの図形スタイルの link でリンク先名を指定するか、テキストスタイルの withboxstyle で同様に link 指定をする。例えば

```
Shape->box(0,0,100,50,'s',SStyle(link => 'dest'))
```

とすると、横 100 ポイント、縦 50 ポイントの矩形が作られて、その内部をクリックすると名前が 'dest' のリンク先に飛ぶ。テキストの場合は、

```
Text('ここをクリック',TStyle(withbox => 'n',
  withboxstyle => SStyle(link => 'dest')))
```

のようにすればよい。このように withbox => 'n' とすると矩形は描画されない。リンクであることを示すために色を変えるとか下線を付けるとかいった工夫はユーザーに任されている。（withbox => 'b' で下線を付けることができる。）
URI リンクの場合は、Dest() によるリンク先の設定は必要なく、リンク先の名前として、「URI:」に続けて URI を書けばよい。例えば

```
Text('米アドビ',TStyle(withbox => 'n',
  withboxstyle => SStyle(link => 'URI:http://www.adobe.com/')))
```

のようにする。
URI はすでに URI エンコードされていない限り URI エンコードされる。

■ PDFJ::Doc クラスのメソッド

```
  PDFJ::Doc->new($version, $pagewidth, $pageheight)
  PDFJ::Doc->new({version => $version, pagewidth => $pagewidth, pageheight =>
$pageheight})
  filter($filter)
  filter({filter => $filter})
  print($file)
  print({file => $file})
  new_page($pagewidth, $pageheight)
  new_page({pagewidth => $pagewidth, pageheight => $pageheight})
  get_page($pagenum)
  get_page({number => $pagenum})
  get_lastpagenum
  new_font($basefont, $encoding, $abasefont, $aencoding)
  new_font({basefont => $basefont, encoding => $encoding, abasefont => $abasefont,
aencoding => $aencoding})
  italic($font1, $font2, ...)
  italic({base => $font1, decorated => $font2})
  bold($font1, $font2, ...)
  bold({base => $font1, decorated => $font2})
  new_image($src, $pxwidth, $pxheight, $width, $height, $padding, $colorspace)
  new_image({src => $src, pxwidth => $pxwidth, pxheight => $pxheight, width => $width,
height => $height, padding => $padding, colorspace => $colorspace})
```

■ PDFJ::Page クラスのメソッド

```
  PDFJ::Page->new($docobj, $pagewidth, $pageheight)
  pagenum
  layer($layernum)
  layer({layer => $layernum})
```

■ PDFJ::AFont クラスのメソッド

```
  PDFJ::AFont->new_std($docobj, $basefont, $encoding)
  PDFJ::AFont->new_ttf($docobj, $ttffile, $encoding)
```

■ PDFJ::JFont クラスのメソッド

```
  PDFJ::JFont->new_std($docobj, $basefont, $encoding, $afontobj)
  PDFJ::JFont->new_ttf($docobj, $ttffile, $encoding, $afontobj)
```

■ PDFJ::Showable クラスのメソッド

次のメソッドは、PDFJ::Text、PDFJ::Paragraph、PDFJ::Block、PDFJ::Image、PDFJ::Shape の各クラスで

共通して使える。

```
  show($page, $x, $y, $align, $transtype, @transargs)
  show({page => $page, x => $x, y => $y, align => $align, transtype => $transtype,
transargs => [@transargs]})
  width
  height
```

■ PDFJ::Text クラスのメソッド

```
  PDFJ::Text->new($text, $style)
  PDFJ::Text->new(@texts, $style)
  PDFJ::Text->new([@texts], $style)
  PDFJ::Text->new({texts => $text, style => $style})
  PDFJ::Text->new({texts => [@text], style => $style})
```

■ PDFJ::Paragraph クラスのメソッド

```
  PDFJ::Paragraph->new($text, $style)
  PDFJ::Paragraph->new({text => $text, style => $style})
  linesnum
  break($size)
  break(@sizes)
  break({sizes => $size})
  break({sizes => [@sizes]})
```

■ PDFJ::Block クラスのメソッド

```
  PDFJ::Block->new($direction, $object, $style)
  PDFJ::Block->new($direction, @objects, $style)
  PDFJ::Block->new($direction, [@objects], $style)
  PDFJ::Block->new({direction => $direction, objects => $object, style => $style})
  PDFJ::Block->new({direction => $direction, objects => [@objects], style => $style})
  adjustwidth($size)
  adjustwidth({size => $size})
  adjustheight($size)
  adjustheight({size => $size})
  break($size)
  break(@sizes)
  break({sizes => $size})
  break({sizes => [@sizes]})
```

■ PDFJ::BlockSkip クラスのメソッド

```
  PDFJ::BlockSkip->new($skip)
```

```
PDFJ::BlockSkip->new({skip => $skip})
```

■ PDFJ::Image クラスのメソッド

```
PDFJ::Image->new($docobj, $src, $pxwidth, $pxheight, $width, $height, $padding)
setsize($width, $height)
setpadding($padding)
```

■ PDFJ::Shape クラスのメソッド

```
PDFJ::Shape->new($style)
PDFJ::Shape->new({style => $style})
```

マクロ命令（描画範囲の管理がおこなわれる）

```
line($x, $y, $w, $h, $style)
line({x => $x, y => $y, w => $w, h => $h, style => $style})
box($x, $y, $w, $h, $spec, $style)
box({x => $x, y => $y, w => $w, h => $h, spec => $spec, style => $style})
circle($x, $y, $r, $spec, $arcarea, $style)
circle({x => $x, y => $y, r => $r, spec => $spec, arcarea => $arcarea, style => $style})
ellipse($x, $y, $xr, $yr, $spec, $arcarea, $style)
ellipse({x => $x, y => $y, xr => $xr, yr => $yr, spec => $spec, arcarea => $arcarea,
style => $style})
polygon([@coords], $spec, $style)
polygon({coords => [@coords], spec => $spec, style => $style})
```

オブジェクト配置命令

```
obj($obj, @showargs)
obj({obj => $obj, showargs => [@showargs]})
```

プリミティブ命令

```
setboundary($x, $y)
gstatepush
gstatepop
linewidth($w)
linedash($dash, $gap, $phase)
ctm(@array)
fillcolor($color)
strokecolor($color)
fillgray($g)
strokegray($g)
fillrgb($r, $g, $b)
strokergb($r, $g, $b)
moveto($x, $y)
lineto($x, $y)
curveto($x1, $y1, $x2, $y2, $x3, $y3)
```

```
rectangle($x, $y, $w, $h)
closepath
newpath
stroke
closestroke
fill
fill2
```

■ PDFJ::Style クラスのメソッド

次のメソッドは PDFJ::TextStyle、PDFJ::ParagraphStyle、PDFJ::BlockStyle、PDFJ::ShapeStyle のすべてで使える。

```
clone(%args)
clone({%args})
clone($argstr)
```

■ PDFJ::TextStyle クラスのメソッド

```
PDFJ::TextStyle->new(%args)
PDFJ::TextStyle->new({%args})
PDFJ::TextStyle->new($argstr)
```

■ PDFJ::ParagraphStyle クラスのメソッド

```
PDFJ::ParagraphStyle->new(%args)
PDFJ::ParagraphStyle->new({%args})
PDFJ::ParagraphStyle->new($argstr)
```

■ PDFJ::BlockStyle クラスのメソッド

```
PDFJ::BlockStyle->new(%args)
PDFJ::BlockStyle->new({%args})
PDFJ::BlockStyle->new($argstr)
```

■ PDFJ::ShapeStyle クラスのメソッド

```
PDFJ::ShapeStyle->new(%args)
PDFJ::ShapeStyle->new({%args})
PDFJ::ShapeStyle->new($argstr)
```

■ PDFJ::Color クラスのメソッド

```
PDFJ::Color->new($r, $g, $b)
```

```
PDFJ::Color->new($rgb)
PDFJ::Color->new($g)
PDFJ::Color->new({value => $rgb})
PDFJ::Color->new({value => $g})
```

■ PDFJ::Outline クラスのメソッド

```
PDFJ::Outline->new($title, $level)
PDFJ::Outline->new({title => $title, level => $level})
```

■ PDFJ::Dest クラスのメソッド

```
PDFJ::Dest->new($name)
PDFJ::Dest->new({name => $name})
```

■内部クラス

以下は、通常はユーザーが直接扱う必要のない、PDFJ 内部で使われるクラス。

PDFJ::File クラス

PDFJ::Doc の下請け。PDF 文書＝ PDFJ::Doc オブジェクトを PDF ファイルに書き出す際に、PDF オブジェクトの索引情報などのメタデータを付加して、規定に従ったファイル構造を作る役割をする。

PDFJ::ObjTable クラス

PDFJ::Doc の下請け。PDFJ::Doc オブジェクトに含まれる PDF オブジェクトを管理する。

PDFJ::TextSpec クラス

PDFJ::Text の下請けとして、テキスト属性を保持する。

PDFJ::Object::* クラス群

PDF 文書は Adobe 社が規定した PDF の文法に沿って構成されており、その構成単位もまた「オブジェクト」と呼ばれる。この PDF レベルのオブジェクトを、上記で説明したような PDFJ における Perl オブジェクトと区別するために、「PDF オブジェクト」と呼ぶことにする。PDFJ は、低レベルでプリミティブな PDF オブジェクトを隠蔽し、ユーザーが直接扱わなくてよいようにしている。もし自分で PDF オブジェクトを操作したいときは、PDFJ::Object::* クラス群を使えばよい。

PDFJ::TTF クラス

TrueType フォントファイル (.ttf) の内容を読みとったり、サブセットを作成したりする。

PDFJ::TTC クラス

TrueTypeCollection フォントファイル (.ttc) を読みとって、指定した番号のフォントについての PDFJ::TTF オブジェクトを得る。

AUTHOR

中島 靖 nakajima@netstock.co.jp http://hp1.jonex.ne.jp/~nakajima.yasushi/

SEE ALSO

「JIS X 4051（日本語文書の行組版方法）」(JIS、1995)
「PDF リファレンス 第２版」(アドビシステムズ、2001)