

PDFlib, PDFlib+PDI, PPS

A library for generating PDF on the fly

PDFlib 10.0.0

チュートリアル

C • C++ • Java • .NET • .NET Core • Objective-C •

Perl • PHP • Python • RPG • Ruby 用



Copyright © 1997–2022 PDFlib GmbH and Thomas Merz. All rights reserved.

PDFlib ユーザーは本マニュアルを内部利用のために印刷または電子的に複製することを許諾されます。

PDFlib GmbH

Franziska-Bilek-Weg 9, 80339 München, Germany

www.pdflib.com

電話 +49・89・452 33 84-0

jp.sales@pdflib.com

jp.support@pdflib.com (ライセンス番号をお知らせください)

この出版物およびここに含まれた情報はありのままに供給されるものであり、通知なく変更されることがあり、また、PDFlib GmbH による誓約として解釈されるべきものではありません。PDFlib GmbH はいかなる誤りや不正確に対しても責任や負担を全く負うものではなく、この出版物に関するいかなる類の（明示的・暗示的または法定に関わらず）保障をも行うものではなく、そして、いかなるそしてすべての売買可能性の保障と、特定の目的に対する適合性と、サードパーティーの権利の侵害とを明白に否認します。

PDFlib と PDFlib ロゴは PDFlib GmbH の登録商標です。PDFlib ライセンス保持者は PDFlib の名称とロゴを彼らの製品の文書内で用いる権利を与えられます。ただし、これは必須ではありません。

PDFlib は PANTONE® スポットカラーデータを含んでいます © Pantone LLC, 2019.

ソフトウェアアプリケーションやユーザー向け文書で表示される PANTONE® カラーは PANTONE 定義規格と一致しない場合があります。正確な色については最新の PANTONE Color Publication をご覧ください。

PANTONE® およびその他の Pantone, Inc. の商標は Pantone, Inc. に帰属します。© Pantone, Inc., 2003.

Pantone, Inc. は PDFlib GmbH に対して PDFlib ソフトウェアとの組み合わせでのみ使用するための頒布ライセンスされた色データおよび/またはソフトウェアの著作権者です。PANTONE カラーデータおよび/またはソフトウェアは PDFlib ソフトウェアの実行の部分として以外に他のディスク上やメモリー内へ複製してはいけません。

PDFlib は HKS Warenzeichenverband e.V からライセンスを受けた HKS® スポットカラーデータを含んでいます。

PDFlib は以下のサードパーティーコンポーネントを含んでいます：

Adobe CMap リソース群、Copyright © 1990-2019 Adobe

AES・Arcfour・SHA アルゴリズム、Copyright © 1995-1998 Eric Young

Brotli 伸長コード、Copyright © 2009, 2010, 2013-2016 by the Brotli Authors

Curl マルチプロトコルファイル転送ライブラリー、Copyright © 1996-2021 Daniel Stenberg

Expat XML パーサー、Copyright © 2001-2019 Expat maintainers

ICLlib、Copyright © 1997-2021 Graeme W. Gill

ICU International Components for Unicode、Copyright © 1991-2020 Unicode, Inc.

libjpeg、Copyright © 1991-2020, Thomas G. Lane, Guido Vollbeding

libpng、Copyright © 2006-2017 Glenn Randers-Pehrson, Copyright © 2018-2019 Cosmin Truta

Libtiff 画像ライブラリー、Copyright © 1988-1997 Sam Leffler、Copyright © 1991-1997 Silicon Graphics, Inc.

MD5 メッセージダイジェスト、Copyright © 1991-2 RSA Data Security, Inc.

NotoSans フォント群、Copyright © 2012 Google Inc. All Rights Reserved.

OpenJPEG ライブラリー、Copyright © 2002-2014, Université catholique de Louvain (UCL), Belgium

OpenSSL 暗号ライブラリー、Copyright © 1998-2018 The OpenSSL Project

sRGB v2 ICC プロファイル、Copyright © International Color Consortium 2016

WOFF2 フォント伸長、Copyright © 2013-Packard Company 2017 by the WOFF2 Authors

Zlib 圧縮ライブラリー、Copyright © 1995-2017 Jean-loup Gailly and Mark Adler

PDFlib Block Plugin は以下のサードパーティーコンポーネントをも含んでいます：

wxWidgets Cross-Platform GUI Library、Copyright © 2018 © 1998 Julian Smart、© 2018 wxWidgets



目次

o PDFlib のライセンスキーを適用 11

1 はじめに 15

- 1.1 PDFlib とは 15
- 1.2 説明書とサンプルの案内 17
- 1.3 構成の必要事項 19
- 1.4 PDFlib/PDFlib+PDI/PPS 10.0 の新機能 20
- 1.5 廃止・削除したメソッドとオプション 23
- 1.6 PDFlib の機能 24
- 1.7 PDFlib+PDI と pCOS インターフェイスの追加的機能 28
- 1.8 PPS と Block Plugin の追加的機能 29
- 1.9 製品別機能一覧 30

2 PDFlib の言語バインディング 31

- 2.1 C バインディング 31
- 2.2 C++ バインディング 35
- 2.3 Java バインディング 37
- 2.4 .NET バインディング 39
- 2.5 Objective-C バインディング 42
- 2.6 Perl バインディング 44
- 2.7 PHP バインディング 46
- 2.8 Python バインディング 48
- 2.9 RPG バインディング 49
- 2.10 Ruby バインディング 51

3 PDF 文書を作成 55

- 3.1 PDFlib プログラミングの一般的特徴 55
 - 3.1.1 例外処理 55
 - 3.1.2 ログ記録 57
 - 3.1.3 PDFlib 仮想ファイルシステム (PVF) 58
 - 3.1.4 リソース構成とファイル検索 59
 - 3.1.5 PDF 文書をメモリー内に生成 64

- 3.1.6 PDF 文書の最大サイズとその他の制限 65
- 3.1.7 Web 最適化 (線形) PDF 66
- 3.1.8 マルチスレッドプログラミング 66
- 3.1.9 EBCDIC ベースのプラットフォームで PDFlib を使う 66
- 3.2 ページ記述 68
 - 3.2.1 座標系 68
 - 3.2.2 ページ寸法 70
 - 3.2.3 直接パス 71
 - 3.2.4 パスオブジェクト 72
 - 3.2.5 テンプレート (フォームXObject) 74
- 3.3 PDF のパスワードセキュリティ 76
 - 3.3.1 PDF におけるパスワードセキュリティ 76
 - 3.3.2 PDFlib を用いて PDF 文書をパスワード保護 76
- 3.4 XMP メタデータ 79
- 3.5 連携ファイル 80

4 色と透過 83

- 4.1 デバイス色空間 83
- 4.2 ICC プロファイルによる色管理 85
 - 4.2.1 ICC プロファイルを読み込む 85
 - 4.2.2 ICC プロファイルを使用 86
 - 4.2.3 高度な色管理制御 88
- 4.3 デバイス独立な CIE L*a*b* カラー 90
- 4.4 Pantone・HKS・カスタムスポットカラー 91
- 4.5 DeviceN カラー 94
- 4.6 シェーディングとシェーディングパターン 98
- 4.7 タイリングパターン 100
- 4.8 透過ブレンドモード 101
- 4.9 オブジェクトの色を変更 104
 - 4.9.1 ブレンドモードを用いて色を変える 104
 - 4.9.2 ソフトマスクを用いて色を変える 105
- 4.10 オーバープリント制御 107

5 Unicode とレガシエンコーディング 109

- 5.1 Unicode の重要な諸概念 109
- 5.2 Unicode 対応言語バインディング 111
 - 5.2.1 ネイティブ Unicode 文字列のある言語バインディング 111

- 5.2.2 UTF-8 対応のある言語バインディング 111
- 5.2.3 デフォルト UTF-8 対応のある言語バインディング 111
- 5.3 非 Unicode 対応言語バインディング 113
 - 5.3.1 ネイティブな Unicode 文字列のない言語バインディング 113
 - 5.3.2 Unicode 処理と各種文字列 113
 - 5.3.3 シングルバイト (8ビット) エンコーディング 116
 - 5.3.4 日本語・中国語・韓国語 CMap (廃止済) 118
- 5.4 キャラクターを指定 121
 - 5.4.1 エスケープシーケンス 121
 - 5.4.2 文字参照 122

6 フォント 127

- 6.1 フォント形式 127
 - 6.1.1 TrueType フォント 127
 - 6.1.2 OpenType フォント 127
 - 6.1.3 SVG・COLR・sbix 形式の OpenType カラーフォント 128
 - 6.1.4 WOFF・WOFF2 フォント 130
 - 6.1.5 PostScript Type 1 フォント (廃止済) 131
 - 6.1.6 SING フォント (グリフレット) 131
 - 6.1.7 Type 3 フォント 131
- 6.2 Unicode のキャラクターとグリフ 133
 - 6.2.1 グリフ ID 133
 - 6.2.2 グリフに対する Unicode マッピング 133
- 6.3 フォントを読み込む 135
 - 6.3.1 テキストフォントに対するエンコーディングを選ぶ 135
 - 6.3.2 記号フォントに対するエンコーディングを選ぶ 137
 - 6.3.3 例: Wingdings 記号フォント内のグリフを選択 138
 - 6.3.4 フォントを検索 141
 - 6.3.5 Windows・macOS 上のホストフォント 144
 - 6.3.6 フォールバックフォント 146
- 6.4 フォントの埋め込みとサブセット化 149
 - 6.4.1 埋め込み 149
 - 6.4.2 サブセット化 149
- 6.5 フォント情報をクエリー 151
 - 6.5.1 フォント非依存のエンコーディング・Unicode・グリフ名クエリー 151
 - 6.5.2 フォント依存のエンコーディング・Unicode・グリフ名クエリー 152
 - 6.5.3 コードページ網羅性とフォールバックフォントをクエリー 153

7 テキスト出力 155

- 7.1 テキスト出力方式 155

- 7.2 フォントメトリックとテキストスタイル 157
 - 7.2.1 フォントとグリフのメトリック 157
 - 7.2.2 カーニング 157
 - 7.2.3 テキストスタイル 158
- 7.3 絵文字シーケンス 161
- 7.4 OpenType レイアウト機能 163
 - 7.4.1 対応している OpenType レイアウト機能 163
 - 7.4.2 テキスト行・テキストフローで OpenType レイアウト機能 166
 - 7.4.3 用字系・言語固有の OpenType レイアウト機能 167
- 7.5 複雑用字系の出力 170
 - 7.5.1 複雑用字系 170
 - 7.5.2 シェーピング処理を制御 171
 - 7.5.3 双方向組版 172
 - 7.5.4 アラビア語テキスト組版 173
- 7.6 日本語・中国語・韓国語テキスト出力 175
 - 7.6.1 TrueType・OpenType 日中韓フォントを用いる 175
 - 7.6.2 横書きと縦書き 175
 - 7.6.3 EUDC・SING フォントによる外字キャラクター 176
 - 7.6.4 OpenType レイアウト機能と日中韓テキスト出力 177
- 7.7 Unicode 異体字セレクター (UVS) 179
 - 7.7.1 各種の異体字シーケンス 179
 - 7.7.2 PDFlib で異体字シーケンス 180
- 7.8 テキスト処理パイプライン 182
 - 7.8.1 入力文字列を Unicode へ正規化 182
 - 7.8.2 Unicode 値をグリフ ID へ変換 183
 - 7.8.3 グリフ ID を転換 184

8 画像・SVG グラフィック・PDF ページを取り込む 187

- 8.1 ラスター画像 187
 - 8.1.1 基本的な画像処理 187
 - 8.1.2 対応画像ファイル形式 189
 - 8.1.3 クリッピングパス 192
 - 8.1.4 画像透過 193
 - 8.1.5 画像にスポットカラーか DeviceN カラーで着色 197
 - 8.1.6 復号配列を用いてカラー値を変える 197
- 8.2 SVG グラフィック 199
 - 8.2.1 対応 SVG 種別 199
 - 8.2.2 SVG 処理上の考慮事項 199
 - 8.2.3 SVG グラフィックの寸法 201
 - 8.2.4 フォント選択 202
 - 8.2.5 見つからないフォント、見つからないグリフを扱う 204

- 8.2.6 SVG カラー拡張 205
- 8.2.7 ベクトルグラフィック・テキスト以外の SVG 内容 208
- 8.2.8 対応していない SVG 機能 209
- 8.3 PDF ページを PDI で取り込む 211
 - 8.3.1 PDI の機能と用途 211
 - 8.3.2 PDFlib+PDI を使用 211
 - 8.3.3 文書・ページ関連のチェック 213
 - 8.3.4 注釈・フォームフィールド・アクションを取り込む 213
 - 8.3.5 取り込んだ PDF 文書のさらなる特徴 215
 - 8.3.6 文書関連のプロパティを転写 217
 - 8.3.7 ページ関連のプロパティを転写 219
- 8.4 画像・グラフィック・取り込み PDF ページを配置 221
 - 8.4.1 単純にオブジェクトを配置 221
 - 8.4.2 オブジェクトを点上か線上か枠内に配置 221
 - 8.4.3 オブジェクトの向きを変える 223
 - 8.4.4 オブジェクトを回転 224
 - 8.4.5 ページ寸法の調整 225
 - 8.4.6 配置された画像と PDF ページに関する情報をクエリー 226

9 テキストと表の組版 229

- 9.1 テキスト行を配置・はめ込む 229
 - 9.1.1 単純なテキスト行配置 229
 - 9.1.2 テキストを枠内に位置付け 230
 - 9.1.3 テキストを枠へはめ込み 231
 - 9.1.4 テキストを文字で揃える 233
 - 9.1.5 スタンプを配置 234
 - 9.1.6 リーダーを用いる 234
 - 9.1.7 パス上テキスト 235
 - 9.1.8 影付きテキスト 236
 - 9.1.9 Acrobat で編集できる透かし 236
- 9.2 複数行のテキストフロー 238
 - 9.2.1 テキストフローをはめ込み枠に配置 240
 - 9.2.2 段落の組版のオプション 241
 - 9.2.3 インラインオプションリストとマクロ 242
 - 9.2.4 タブ位置 245
 - 9.2.5 番号付きリストと段落間隔 245
 - 9.2.6 制御キャラクターとキャラクターマッピング 247
 - 9.2.7 ハイフネーション 249
 - 9.2.8 ウィドー行・オーファン行 250
 - 9.2.9 標準改行アルゴリズムの制御 251
 - 9.2.10 高度な用字系固有の改行 254
 - 9.2.11 テキストをパス・画像に回り込ませる 255

9.3 表の組版 259

- 9.3.1 単純な表を配置 260
- 9.3.2 表セルのさまざまな内容 263
- 9.3.3 表と列の幅 265
- 9.3.4 さまざまな種類の内容を持った表 266
- 9.3.5 表インスタンス 269
- 9.3.6 表組版のアルゴリズム 271

9.4 範囲枠 275

- 9.4.1 テキスト行を装飾 275
- 9.4.2 テキストフロー内で範囲枠を用いる 276
- 9.4.3 範囲枠と画像 277

10 インタラクティブ機能 279

- 10.1 リンク・しおり・注釈 279
- 10.2 フォームフィールドと JavaScript 282
- 10.3 スクリーン注釈と表現アクションを用いたマルチメディア 285
- 10.4 地理空間 PDF 289
 - 10.4.1 地理空間 PDF を Acrobat で利用 289
 - 10.4.2 地理座標系と投影座標系 289
 - 10.4.3 座標系の例 290
 - 10.4.4 Acrobat における地理空間 PDF の制約 291

11 タグ付き PDF と PDF/UA 293

- 11.1 タグ付き PDF の基礎 293
 - 11.1.1 論理構造ツリー（構造ヒエラルキー）を作成 293
 - 11.1.2 標準構造種別 297
 - 11.1.3 PDF 1.7 の構造要素ネスティング規則 303
 - 11.1.4 PDF 2.0 の構造要素ネスティング規則 303
 - 11.1.5 カスタム構造種別・ロールマップ・カスタムタグセット 308
 - 11.1.6 ページ装飾 309
 - 11.1.7 テキスト処理 311
 - 11.1.8 代替記述・置換テキスト・略語拡張 313
 - 11.1.9 印刷ストリーム順序と論理読み取り順序 314
 - 11.1.10 Adobe Acrobat におけるタグ付き PDF の不具合 315
- 11.2 タグ付き PDF 作成の高度なトピック 317
 - 11.2.1 自動表タグ付け 317
 - 11.2.2 インタラクティブ要素にタグ付け 320
 - 11.2.3 インタラクティブ要素にタグ付けする PDF 2.0 の機能 325
 - 11.2.4 リストにタグ付け 325
 - 11.2.5 内容を順序にとらわれず作成 327

- 11.3 タグ付き PDF ページを PDI で取り込む 330
- 11.4 PDF/UA によるユニバーサルアクセシビリティ 335
 - 11.4.1 PDF/UA-1 規格 335
 - 11.4.2 タグ付けの要件 337
 - 11.4.3 内容種別ごとの追加の要件 339
 - 11.4.4 PDF/UA 文書のページを PDI で取り込む 340

12 PDF のバージョンと規格 341

- 12.1 PDF のバージョン 341
 - 12.1.1 PDF のバージョンの概観 341
 - 12.1.2 PDF 1.x の機能 342
 - 12.1.3 PDF 2.0 の機能 343
 - 12.1.4 PDF の廃止済の機能 343
- 12.2 PDF/A によるアーカイビング 346
 - 12.2.1 各種の PDF/A 規格 346
 - 12.2.2 一般的要件 347
 - 12.2.3 色と画像の要件 348
 - 12.2.4 インタラクティブ機能に対する要件 351
 - 12.2.5 レベル U 準拠のための追加の PDF/A の要件 352
 - 12.2.6 レベル A 準拠のための追加の PDF/A の要件 352
 - 12.2.7 PDF/A 文書を PDI で取り込み 354
 - 12.2.8 PDF/A のための XMP 文書メタデータ 355
- 12.3 PDF/X による印刷出力 358
 - 12.3.1 PDF/X 規格ファミリー 358
 - 12.3.2 一般的要件 359
 - 12.3.3 出力インテントと色の要件 360
 - 12.3.4 画像と透過の要件 363
 - 12.3.5 インタラクティブ機能のための要件 363
 - 12.3.6 PDF/X 文書を PDI で取り込む 365
- 12.4 PDF/VT による可変・トランザクション印刷 367
 - 12.4.1 PDF/VT 規格 367
 - 12.4.2 PDF/VT の諸概念 367
 - 12.4.3 PDF/VT-1 を生成するための規則の要約 369
 - 12.4.4 文書部分ヒエラルキーと文書部分メタデータ (DPM) 370
 - 12.4.5 反復するグラフィカル内容のためのスコープヒント 372
 - 12.4.6 カプセル化 XObject 373
 - 12.4.7 PDF/X・PDF/VT 文書を PDI で取り込む 374

13 PPS と PDFlib Block Plugin 375

- 13.1 PDFlib Block Plugin をインストール 375
- 13.2 ブロックの概念の概要 378

- 13.2.1 文書デザインとプログラムコードの分離 378
- 13.2.2 ブロックのプロパティ 378
- 13.2.3 なぜ PDF のフォームフィールドを使わないの 379
- 13.3 PDFlib Block Plugin でブロックを編集 381
 - 13.3.1 ブロックを作成 381
 - 13.3.2 ブロックプロパティを編集 384
 - 13.3.3 ページ間・文書間でブロックをコピー 386
 - 13.3.4 Block Plugin のユーザーインターフェイスを XML でカスタマイズ 388
- 13.4 PDF フォームフィールドを PDFlib ブロックに変換 389
- 13.5 ブロックを Acrobat でプレビュー 392
- 13.6 PPS でブロックに流し込み 397
- 13.7 ブロックのプロパティ 402
 - 13.7.1 管理プロパティ 402
 - 13.7.2 長方形プロパティ 403
 - 13.7.3 書式プロパティ 404
 - 13.7.4 テキスト作成プロパティ 407
 - 13.7.5 テキスト組版プロパティ 408
 - 13.7.6 オブジェクトはめ込みプロパティ 411
 - 13.7.7 デフォルト内容のためのプロパティ 414
 - 13.7.8 カスタムプロパティ 414
- 13.8 pCOS でブロック名とプロパティをクエリー 415
- 13.9 ブロックをプログラムの的に作成・取り込む 417
 - 13.9.1 POCA で PDFlib ブロックを作成 417
 - 13.9.2 PDFlib ブロックを取り込む 418
- 13.10 PDFlib ブロックの仕様 419

A 改訂履歴 423

索引 425

PDFlib のライセンスキーを適用

評価版の制約 PDFlib GmbH によって提供される PDFlib・PDFlib+PDI・PPS のすべてのバイナリーバージョンは、商用ライセンスを取得したか否かにかかわらず、完全に動作する評価版として利用できます。ただし非ライセンス版は、すべての生成されるページ上に、www.pdflib.com というデモスタンプを横断印字し、また、内蔵の pCOS インターフェイスは小容量の文書（10 ページ以下、ファイルサイズ 1 MB 以下）に制限されます。非ライセンスのバイナリーは、業務目的に使用してはならず、この製品を評価するためにのみ使用できます。PDFlib GmbH 製品はいずれも、業務目的に使用するには有効なライセンスが必要です。

PDFlib ライセンスの取得をご検討いただいている企業で、評価段階やプロトタイプ/demo 期間中に評価制約の除去をご希望の場合は、sales@pdflib.com 宛に企業情報・プロジェクト内容を簡単にご説明いただければ、一時的なライセンスキーをご提供します（評価キーの提供要請をお断りする権利を私達は保持いたします。たとえば匿名によるご希望の場合など）。

PDFlib・PDFlib+PDI・PDFlib Personalization Server (PPS) は、1 つのパッケージとして頒布されていますが、それぞれ異なる製品であり、別々のライセンスキーを必要とします。PDFlib+PDI のライセンスキーは PDFlib に対しても有効ですが、その逆は無効であり、また、PPS のライセンスキーは PDFlib+PDI と PDFlib に対して有効です。すべてのライセンスキーはプラットフォーム依存であり、購入された対象のプラットフォームでしか使用できません。

PDFlib または PDI のライセンスキーをご購入いただいたら、それを適用してデモスタンプを除去してください。ライセンスキーを設定するにはいくつかの方法があります。以下にそれを解説します。

クックブック 完全なコードサンプルがクックブックの `general/license_key` トピックにあります。

ライセンスキーを API 呼び出しで実行時に適用 スクリプトやプログラムに行を追加して、ライセンスキーを実行時に設定するようにします。PDFlib オブジェクトをインスタンス化した直後に（C の場合：`PDF_new()` の後に）、`license` オプションを設定する必要があります。具体的な文法は、使用するプログラミング言語によって異なります：

- ▶ C++・Java・.NET/C#・Python・Ruby の場合：

```
p.set_option("license=...あなたのライセンスキー ...")
```

- ▶ C の場合：

```
PDF_set_option(p, "license=...あなたのライセンスキー ...")
```

- ▶ Objective-C の場合：

```
[pdflib set_option: @"license=...あなたのライセンスキー ..."];
```

- ▶ Perl・PHP の場合：

```
$p->set_option("license=...あなたのライセンスキー ...")
```

- ▶ RPG の場合：

```
c          callp      PDF_set_option(p:%ucs2('license=...あなたのライセンスキー...'))
```

ライセンスファイルを使用 実行時の呼び出しでライセンスキーを与えるのではなく、そのライセンスキーをテキストファイルに入力しておくという方法もあります。以下の形式に従ってください (PDFlib のディストリビューションにはすべて、ライセンスファイルテンプレート *licensekeys.txt* が入っているので、それを利用することもできます)。1 個の「#」キャラクターで始まる行は、注釈を内容としており、無視されます。2 行目は、ライセンスファイル自体のバージョン情報を内容としています：

```
# Licensing information for PDFlib GmbH products
PDFlib license file 1.0
PDFlib      10.0.0 ...あなたのライセンスキー ...
```

このライセンスファイル内には、複数の PDFlib GmbH 製品に対するライセンスキー群を、それぞれ別の行に記述することもできます。また、複数のプラットフォームに対するライセンスキー群を内容として持たせることにより、同一のライセンスファイルを複数のプラットフォームで共用することも可能です。ライセンスファイルは、以下の方法で構成することができます：

- ▶ *licensekeys.txt* という名前のファイルが、すべてのデフォルト位置で検索されます (13 ページ「デフォルトファイル検索パス」を参照)。
- ▶ *licensefile* オプションを *set_option()* API メソッドで設定することもできます：

```
p.set_option("licensefile={/ファイルへの/パス/licensekeys.txt}");
```

- ▶ ライセンスファイルを指し示す環境 (シェル) 変数を設定することもできます。Windows では、システムのコントロールパネルを使って、「システム」→「詳細」→「環境変数」を選択します。Unix では、以下のようなコマンドを適用します：

```
export PDFLIBLICENSEFILE=/ファイルへの/パス/licensekeys.txt
```

- ▶ IBM System i では、ライセンスファイルは ASCII で符号化されている必要があります (*asciifile* オプションを参照)。ライセンスファイルを以下のように指定できます (このコマンドは、スタートアッププログラム *QSTRUP* 内で指定することができ、すべての PDFlib GmbH 製品で動作します)：

```
ADDENVVAR ENVVAR(PDFLIBLICENSEFILE) VALUE('/PDFlib/10.0/licensefile.txt') LEVEL(*SYS)
```

レジストリー内へライセンスキー Windows では、以下のレジストリー値にライセンスファイルの名前を書き込むという方法もあります：

```
HKLM\SOFTWARE\PDFlib\PDFLIBLICENSEFILE
```

または、以下のレジストリー値のうちのいずれか 1 つにライセンスキーを直接書き込むこともできます：

```
HKLM\SOFTWARE\PDFlib\PDFlib10\license
HKLM\SOFTWARE\PDFlib\PDFlib10\10.0\license
```

注 64 ビット Windows システム上でレジストリーを手作業で扱う際には注意してください：通常どおり、64 ビットの PDFlib バイナリーは Windows レジストリーの 64 ビットビューとともに動作し、64 ビットシステム上で動作する 32 ビットの PDFlib バイナリーはレジストリーの 32 ビットビューとともに動作します。32 ビット製品に対するレジストリーキーを手作業で追加するときは、必ず 32 ビット版の *regedit* ツールを使用してください。これは「スタート」ダイアログから以下によって起動することができます：

%systemroot%\syswow64\regedit

デフォルトファイル検索パス Unix・Linux・macOS・IBM System i では、パス・ディレクトリー名を何も指定しなくても、デフォルトでいくつかのディレクトリーでファイル群が検索されます。UPR ファイル（さらなる検索パスを含んでいる場合もある）を検索して読み込む前に、以下のディレクトリーが検索されます：

```
<rootpath>/PDFlib/PDFlib/10.0/resource/cmap  
<rootpath>/PDFlib/PDFlib/10.0/resource/font  
<rootpath>/PDFlib/PDFlib/10.0/resource/icc  
<rootpath>/PDFlib/PDFlib/10.0  
<rootpath>/PDFlib/PDFlib  
<rootpath>/PDFlib
```

Unix・Linux・macOS では、**<rootpath>** は、まず */usr/local* で、ついで HOME ディレクトリーで置き換えられます。IBM System i では **<rootpath>** は空です。

ライセンスファイルとリソースファイルに対するデフォルトファイル名 デフォルトで、以下のファイル名が、デフォルト検索パスディレクトリー群の中で検索されます：

licensekeys.txt (ライセンスファイル)
pdflib.upr (リソースファイル)

この機能を利用すると、環境変数やランタイムオプションを一切設定せずにライセンスファイルを扱うこともできます。

アップデートとアップグレード アップデート（ある製品の古いバージョンからその同じ製品の新しいバージョンへの切り換え）かアップグレード（PDFlib から PDFlib+PDI または PPS への、または PDFlib+PDI から PPS への切り換え）を購入された場合、あるいはご自分のサポート契約の一部として新しいライセンスキーを受け取った場合には、そのアップデートかアップグレードに対して受け取った新しいライセンスキーを適用する必要があります。前の製品に対する古いライセンスキーは利用できなくなります。

まだライセンスされていない機能を評価 いずれの機能も、ソフトウェアに対してライセンスキーを一切適用せずに完全に評価できます。しかし、ある特定製品に対する有効なライセンスキーの適用後は、それよりも高いカテゴリーの機能は利用できなくなります。たとえば、有効な PDFlib のライセンスキーをインストールすると、PDI の機能を試用できなくなります。同様に、PDFlib+PDI のライセンスキーをインストールした後は、パーソナライゼーション機能（ブロック関数群）を利用できなくなります。

ある製品に対するライセンスキーがすでにインストールされている時は、そのかわりにダミーのライセンス文字列「0」（数字のゼロ）を設定すれば、それよりも高い製品クラスの機能を試用できるようになります。そうすることにより、それ以前に無効にされた関数がある有効になり、また、すべてのページを横断するデモスタンプが再び有効になります。

さまざまなライセンシングオプション PDFlib の 1 台ないし複数のサーバー上での利用や、PDFlib を利用者自身の製品とともに再配布することに対しては、それぞれ異なったライセンシングオプションを利用可能です。サポート契約やソースコード契約も提供しています。ライセンシングについての詳細情報と PDFlib 購入申込フォームは PDFlib ディストリビューションの中にあります。PDFlib のライセンスの入手にご関心をお持ちいただいた場合や、ご質問につきまして何なりと、当社までご連絡ください。

1 はじめに

1.1 PDFlib とは

PDFlib は、PDF 文書を生成・操作する、定評ある開発者向けコンポーネントです。PDFlib の主要ターゲットは、Web サーバー上での動的な PDF 生成と、あらゆる種類のサーバー・デスクトップ・モバイルアプリケーションにおける PDF 生成です。PDFlib を利用すれば、データベースの内容から動的に PDF 文書を生成できます。すなわち動的な Web ページと同様のことができるのです。1997 年以来、PDFlib は幅広いユースケースと開発環境において実力を証明しています。アプリケーションのプログラマーが、PDFlib を用いて、テキスト・画像・グラフィック・インタラクティブ要素を有する PDF 出力を生成します。

PDFlib は、世界じゅうの何万ものユーザー様に現にご利用いただいています。このような大スケールでデプロイしていただいているおかげもあり、当社は API と、内部実装と、生成する PDF 出力をさらに改良・ファインチューンできています。関連する PDF・PDF/A・PDF/X・PDF/UA・PDF/VT の ISO 規格に準拠することと、普及している PDF ビューアーとの互換をとることに高い優先度を与えています。PDFlib は、PDF 形式が提供しているすべての機能に対応しています。

PDFlib の製品ファミリーは、3 つの種類として提供しています：PDFlib と、PDFlib+PDI (PDF Import) と、PDFlib Personalization Server (PPS) (Adobe Acrobat 用 PDFlib Block Plugin が付属) です。

PDFlib PDFlib は PDF 文書を生成するさまざまなメソッドを提供します。それによって、PDF 文書にテキスト・グラフィック・画像を持たせたり、注釈やしおりなどインタラクティブ要素を持たせたりすることが可能です。PDFlib の API は、PDF の生成をさまざまなレベルにおいて行えるようになっています。異なるレベルを自由に混在させることも可能です。

低レベルの API を用いると、アイテムを個別にページに配置できます。その際に整形情報をアプリケーションから与えることができます。レイアウトがすでに決まっているアプリケーション（画面表示など）において、テキスト・画像などオブジェクトを直接 PDF ページに配置できます。

高レベルの API は、強力な整形機能を提供します：

- ▶ 一行テキスト出力で、さまざまな整形制御を行えます。
- ▶ テキストフロー整形機能で、任意の量のテキストを、1 個ないし複数の段またはページに配置できます。テキストを四角形でない形状に整形したり、画像を回り込ませたり、インタラクティブ要素や、タグ付き PDF の構造情報を持たせたりすることも可能です。
- ▶ 柔軟な表整形機能で、表セル内にテキスト・画像・グラフィック・注釈・フォームフィールドなどあらゆる種類の内容を配置できます。大きな表を複数ページにわたって配置することも可能です。表整形の細かい指定も多様なオプションで制御できます。
- ▶ 画像と SVG の配置を、さまざまなはめ込み・整形メソッドによって行えます。

PDFlib+PDI (PDF Import) PDFlib+PDF は、PDFlib の機能をすべて有しているうえに、PDF Import Library PDI という、既存の PDF ページとインタラクティブ要素を、生成する PDF 出力に入れ込むライブラリーを持っています。PDFlib+PDI を利用すると、PDFlib ができることはすべてできるうえ、以下も可能になります：

- ▶ 1 枚のシートに複数の PDF ページを組み付け
- ▶ 既存の PDF ページにヘッダー・フッター・スタンプ・ページ番号などテキストを加える

- ▶ 既存のページに企業ロゴなど画像を貼り付け
- ▶ 複数の PDF ページを並べ替えたり、組み合わせたり、修正したりしながらも、PDF/A・PDF/X・PDF/UA への準拠を維持

PDFlib Personalization Server (PPS) と PDFlib Block Plugin PDFlib Personalization Server (PPS) は、PDFlib+PDI を有しているうえに、PDFlib ブロックを使って可変データ処理を行うメソッドも持っています。PPS を利用するとアプリケーションがレイアウトの変更から影響を受けなくなります。

デザイナーはレイアウトの PDF を作り、可変テキスト・画像・グラフィックのためのプレースホルダーとしたい領域を、Acrobat 用 PDFlib Block Plugin を用いてマークします。この PDFlib Block Plugin にはプレビュー機能があるので、指定したプロパティに従ってブロックへの流し込みが行われた場合の結果を見ることができます。

開発者は、PDFlib ブロックにテキスト・画像・ベクトルグラフィック・PDF ページを流し込むコードを書きます。彼はブロックの書式や位置については知らなくても大丈夫です。PPS を利用すると、PDFlib+PDI ができることはすべてできるうえ、テキストと画像からメールを自動生成したり、テンプレート流し込みで取引・明細処理を行ったり、販促資料をパーソナライズしたり、1つのデータベースから個別の部品カタログを生成したり、複数の製品に対してカスタマイズした説明書を制作したりすることが可能です。

PDFlib を使うには PDFlib は、Unix・Windows・macOS のほか、IBM System i・IBM Z など EBCDIC ベースのシステムを含む多様なプラットフォームで利用可能です。PDFlib は C と C++ で書かれています。他のいくつかの言語とプログラミング環境からも利用可能です。これらを言語バインディングと呼びます。これらのバインディングは、現在のすべての Web 開発環境とスタンドアロン開発環境をカバーしています。そのアプリケーションプログラミングインターフェイス (API) は学習が容易であり、しかもどのバインディングでも同じです。現状、以下のバインディングをサポートしています：

- ▶ C and C++
- ▶ Java
- ▶ .NET
- ▶ Objective-C (macOS・iOS)・Swift
- ▶ Perl
- ▶ PHP
- ▶ Python
- ▶ RPG (IBM System i)
- ▶ Ruby

1.2 説明書とサンプルの案内

当社では、PDFlib 製品の有効活用を支援するため、以下に挙げる資料を提供しています。

すべての言語バインディング用のミニサンプル *hello・image* ミニサンプルが、すべてのパッケージ、すべての言語バインディングで利用可能です。これらは、テキストと画像を出力する最小限のサンプルコードを示しています。このミニサンプルは主に、PDFlib が正しくインストールできたことを確認したり、PDFlib アプリケーションコードの書き方をさっと把握したりするのに便利です。

すべての言語バインディング用のスターターサンプル スターターサンプルが、すべてのパッケージに入っており、すべての言語バインディングで利用可能です。重要なトピックのための出発点として便利であり、シンプルなテキスト・画像出力、テキストフロー・表組版、PDF/A・PDF/X・PDF/UA 作成、その他多くのトピックをカバーしています。このスターターサンプルを見れば、PDFlib 製品を用いて特定の目的を達するための基本的な技法を知ることができます。スターターサンプルを見てみることを強く推奨します。

PDFlib チュートリアル PDFlib チュートリアル (このマニュアル) は、すべてのパッケージに入っていて、重要なさまざまなプログラミング概念をさらに詳しく説明しており、各種の小さなコード断片も含んでいます。コードをスターターサンプルよりも拡張していくにあたっては、この PDFlib チュートリアルの中の関連する内容を知っておくべきです。

注 この PDFlib チュートリアルでは多くの例を Java 言語で示しています。具体的な文法は言語ごとに異なりますが、PDFlib プログラミングの基本概念はどの言語バインディングでも同じです。

PDFlib API リファレンス PDFlib API リファレンスは、すべてのパッケージに入っていて、PDFlib のアプリケーションプログラミングインターフェイス (API) を構成するすべての関数・オプションを簡明に記述しています。使えるオプションや入力条件など従うべきプログラミング規則を調べるには、この PDFlib API リファレンスが絶対の規範です。PDFlib を用いて開発を行う際には必ずこの PDFlib API リファレンスを使用してください。

pCOS パスリファレンス pCOS インターフェイスを利用すると、PDF 文書からさまざまな特性をクエリーすることができます。pCOS は PDFlib+PDI・PPS に内蔵されています。この pCOS パスリファレンスは、PDF 文書内の個々の対象を指し示してその対応する値を取得するために用いられるパス文法の説明を内容としています。

PDFlib クックブック PDFlib クックブックは、さまざまなタスクのための PDFlib コーディング断片を何百も集めたものです。クックブックのトピック群は Java・PHP 用ですが、簡単に他のプログラミング言語に合わせて変えることができます。なぜなら PDFlib API は、対応するすべての言語バインディングについて等価だからです。この PDFlib クックブックは、以下の URL にあります：

www.pdflib.com/pdflib-cookbook/

pCOS クックブック pCOS クックブックは、PDFlib+PDI と PPS に含まれている pCOS インターフェイスの Java コード断片を集めたものです。以下の URL にあります：

www.pdflib.com/pcos-cookbook/

TET クックブック PDFlib TET (Text and Image Extraction Toolkit =テキスト抽出ツールキット)は、PDF 文書からテキストや画像を抽出するための別製品です。これを PDFlib+PDI と組み合わせると、PDF 文書その内容に応じて処理することができます。**TET クックブック**は、TET のための Java と XSLT のコード断片を集めています。これは、TET と PDFlib+PDI の組み合わせを演示するサンプルのグループを含んでいます。たとえば、ページ上のテキストに応じて Web リンクやしおりを追加したり、検索単語をハイライトしたり、テキストに応じて文書を分割したり、目次を作成したりなどです。この TET クックブックは以下の URL にあります：

www.pdflib.com/tet-cookbook/

1.3 構成の必要事項

以前のバージョンと異なり、PDFlib 10 では、特定の操作をする際に外部ファイルが必要となる場合があります。この必要となるファイルは PDFlib のパッケージに含まれています。これらのファイルを適切なディレクトリーへコピーしてこれらのディレクトリーへのアクセスを構成するのはユーザー側の役割です。外部ファイルへのアクセスを構成する方法は *searchpath* などいくつかありますので、詳しくは 59 ページ「3.1.4 リソース構成とファイル検索」を参照してください。外部ファイルが必要になる操作を表 1.1 に挙げます。

表 1.1 構成の必要事項

ファイル	必要な場合
resource/codepage/*.cpg	PDF <code>load_font()</code> と暗黙的フォント読み込みでレガシー 8 ビットエンコーディング iso8859-2 ~ iso8859-16、cp1250 ~ cp1251、cp1253 ~ cp1258 のいずれかを用いる場合 (116 ページ「5.3.3 シングルバイト (8 ビット) エンコーディング」を参照)
resource/font/*.ttf	PDF <code>create_annotation()</code> で <code>type=Caret</code> ・ <code>FreeText</code> ・ <code>Line</code> ・ <code>Stamp</code> のいずれかを用い、しかも <code>font</code> オプションも <code>template</code> オプションも与えない場合。 PDF <code>fit_pdi_page()</code> で <code>useannots</code> (デフォルトで有効です) か <code>usefields</code> を用いて、フォームフィールドか上記いずれかの種別の注釈を含んでいるページを取り込み、しかもその注釈かフィールドに体裁ストリームがない場合。
resource/cmap/*	PDF <code>load_font()</code> と暗黙的フォント読み込みでいずれかのレガシー日中韓エンコーディング (CMap) を用いる場合。なお、これらのエンコーディングは廃止済です (118 ページ「5.3.4 日本語・中国語・韓国語 CMap (廃止済)」を参照)

1.4 PDFlib/PDFlib+PDI/PPS 10.0 の新機能

PDFlib/PDFlib+PDI/PPS 10.0 と Block Plugin 6 の最も重要な新機能と改良機能を以下概観します。他にも多くの新機能がありますので、詳しくは表 1.2 と PDFlib API リファレンスを参照してください。

カラーフォントと絵文字 OpenType カラーフォントでは、グリフ記述で色と透過も指定できます。これは特に絵文字フォントでよく使われます。PDFlib は、SVG か Microsoft COLR グリフに基づくカラーフォントに対応したとともに、OpenType 機能・異体字セレクター・絵文字異体字シーケンス（個々の絵文字の肌の色や性別を変えたりするために活用）にも対応しています。

フォームフィールドと注釈を PDFlib+PDI で取り込み PDFlib+PDI は長らく、ページ内容のみを取り込んで、インタラクティブ要素を無視してきました。リンクなど注釈種別、フォームフィールド、アクション、JavaScript をページ内容とともに取り込むことにより、インタラクティブ要素の機能を確実に維持するようにしました。たとえ取り込んだページを並べ替えて配置したり、取り込んだ複数のページを出力の 1 ページに配置したりしても、インタラクティブ要素の機能が維持されます。ページを拡大縮小・回転した場合には、その中の注釈の長方形も然るべく変換されます。注釈とフォームフィールドの取り込みが PDF/A・PDF/UA・PDF/X と互換になっています。

注釈 あらゆる種類の注釈（コメント）の作成について、オーバーホールを行い、拡張し、かつシンプルにしました。注釈の体裁ストリームが自動的に生成され、PDF/A モードにおいてその規格が許すかぎり注釈がサポートされます。

フォーム流し込み 取り込んだフォームフィールドへの流し込みをできるようにしました。たとえば、取り込んだテキストフォームに内容を書き込んだりできます。取り込んだフォームフィールドの内容と体裁（色・枠など）を変えることも可能ですし、元の状態のままにしておくこともできます。

マルチメディア PDF のスクリーン注釈と表現アクションを用いて動画ファイル・音声ファイルを利用できるようにしました。これらはサウンド注釈・ムービー注釈よりも強力であるとともに、PDFlib 10 でも Acrobat DC でもはや対応していない廃止済の Flash テクノロジーを必要としないものです。

PDF 2.0 に対応 PDFlib は、ISO 32000-2:2017 と年月入り改訂版 ISO 32000-2:2020 に従って PDF 2.0 に対応しました。これによる新機能は、タグ付き PDF のための新たな構造エレメント種別とネ스팅規則や、グラフィック機能、暗号化、インタラクティブ要素、文書レベルとページレベルの出力インテントなど多岐にわたります。PDFlib+PDI と PPS も PDF 2.0 文書を処理します。PDF 2.0 で廃止済とされた機能は PDFlib でも廃止または削除してあります。

色管理 埋め込み ICC プロファイルを内蔵していない RGB 画像に対し、業界標準の sRGB ICC プロファイルを自動的に適用するようにしました。これにより色の再現性が担保され、PDF/A・PDF/X 文書で RGB 画像を使用できます。

透過を有する PDF ページと SVG グラフィックを取り込んだ際の色の再現性についても、その場合には透過グループを自動的に生成するようにしたので、シンプルに担保できるようにになりました。

PDF 出力の最適化 PDF の ICC プロファイルなどオブジェクトをキャッシュし、重複を検出するようにしました。これにより、内容ストリームの演算子を減らす最適化ともあいまって、PDF のファイルサイズがかなり小さくなります。10 GB を超える大きな PDF 文書を最適化・線形化しつつ生成できるようにしました。

SVG の取り込み SVG の処理について、CSS ディレクティブなどいくつかの点を改良しました。リモートのフォントや画像などリソースをネットワークから取り寄せることができます。

フォント対応とテキスト処理 カラーフォント対応以外の改良点

- ▶ WOFF2 形式のフォントを使えるようにしました。
- ▶ OpenType レイアウト機能への対応を改良しました。
- ▶ Type 3 フォントの生成を改良しました。

便利機能 既存の多くの API メソッドに便利オプションを設け、よくあるタスクの手順がシンプルになる改良を行いました。たとえばグラフィックステートプロパティを、多くの API メソッドにおいてオプションで直接与えることを可能にしましたので、gstate オブジェクトを作成し適用する手間が省けます。パスオブジェクトを、テキストフローや表整形機能ですでに作成してある範囲枠の中に描けるようにしました。

デフォルトの PDF 出力の互換性は PDF 1.7 拡張レベル 8 です。これは Acrobat X/XI/DC のファイル形式です。

Pantone® カラーライブラリーをアップデート 内蔵の Pantone スポットカラーデータベースについて、ソリッドコート紙・ソリッド上質紙カラーライブラリーへの最新の追加をカバーする拡張を行いました。ヘキサクロームカラーライブラリーなど Pantone, Inc. がもはやサポートしていないスポットカラーについては削除してあります。

pCOS インターフェイス pCOS インターフェイス (PDFlib+PDI・PPS に内蔵されている) で PDF の内容をクエリーできる疑似オブジェクトを追加しました。新たな PDF 規格・フォームフィールド・署名・ICC プロファイルなどです。

タグ付き PDF と PDF/UA 複数ページにわたっている表に対しては、自動表タグ付け機能が Table エレメントを 1 つだけタグ付けするよう修正しました。複数行にわたっているリンクについては、複数の長方形を持つリンク注釈を 1 つだけ生成します。構造エレメントをテキストフローのインラインで書式オプションとともに与えることができます。

PDF 2.0 は構造エレメントの関係について厳格なモデルを導入しました。PDFlib は新たな PDF 2.0 の構造種別と属性に対応しており、新たな構造エレメントネスティング規則を強制します。タグ名前空間 (タグセット) と構造移動先に対応しています。

タグのない文書を PDF/UA にページ装飾として取り込めるようにしましたので、PDF/UA として使用できない既存の装飾的な内容をシンプルに処理できます。

PDFlib Personalization Server (PPS) と Block Plugin Acrobat 用 PDFlib Block Plugin について、使いやすくするさまざまな改良を行いました。たとえば、画面とプレビューで見えるブロックの数をフィルターで一時的に減らせます。ブロックを小刻みに動かすか大幅に動かすかを簡単に切り替えられます。ブロックのプロパティを追加してオーバープリント設定などあらゆるグラフィックステートオプションに対応させました。構造エレメント (タグ) をブロックプロパティとして利用でき、プレビューが PDF/UA の転写に対応しています。

ネットワーク リモートのリソースをネットワークから取り寄せることのできる PDFlib の API メソッドを追加しました。環境で提供されている手段を使わずに取り寄せが可能です。SVG の処理が、フォントや画像などリモートのリソースを自動的に取得します。

言語バインディング すべての言語バインディングを最新バージョンへアップデートしてあります。Perl・PHP・Ruby バインディングはデフォルトで UTF-8 有効化しました。C++ バインディングは、C++11 と C++20 で導入された Unicode 文字列型 `u8string/u16string/u32string` に対応しています。

その他改良 さまざまな領域の新機能：

- ▶ Type 3 フォント内に着色ありのグリフとなしのグリフを混用可能。
- ▶ バスオブジェクトに書式オプションを追加。

コードベースを現代化、コードセキュリティーを向上 コードをレビューし、より堅牢にするため改良してあります。Acrobat の期限切れのバージョンや RIP への対処など不要なコードを削除しました。廃止済の機能を削除した結果、コードが小さくてクリーンにもなりました。PDFlib 内で使用している各種のサードパーティーのライブラリーについて、セキュリティー改良と脆弱性保護の恩恵を受けるために最新バージョンへアップデートしてあります。

1.5 廃止・削除したメソッドとオプション

PDFlib 9 やもっと前のバージョンでは、数多くの PDFlib の API メソッドを廃止済と宣言しておりましたが、API メソッドを削除したことはこれまで全くなく、マイグレーションを考慮して残してありました。PDFlib 10 では、PDFlib 9 ないしそれ以前のバージョンで廃止済と宣言していた API メソッドをすべて削除しました。たいていの場合は単純に 1 対 1 の置き換えをすればよいようにしてあります。

同様に、廃止済のオプションを削除してあります。たいていの場合は機能自体は残しており、単純に移行先のオプションへ置き換えればよいようにしてあります。

完全になくした機能もあります。市場で需要がないか（一例として参照 XObject）、OPI や Flash など PDF 2.0 で廃止済と宣言されていることが理由です。

PDFlib 9 のアプリケーションは、もし廃止済の機能を全く使用していなければ、修正せずに PDFlib 10 で使用できます。御自身のアプリケーションが廃止済の機能を使用している場合には、PDFlib Migration Guide を御一読下さい。この文書には、廃止・削除された API メソッドとオプションの具体的な一覧と、アプリケーションコードをどう変更する必要があるかに関する説明があります。

PDFlib Migration Guide では、PDFlib 10 へ移行する前に PDFlib 9 で廃止済の機能を特定する方法も説明しています。

1.6 PDFlib の機能

表 1.2 に挙げる PDF 生成機能があります。新たな機能と改良した機能には脚注を付しています。ただし PDF 出力の改良（ファイルが小さくなったなど）とプログラミング便利機能については個別に挙げていません。

表 1.2 PDFlib の機能一覧

分類	機能
各種 PDF	PDF 1.4 ~ PDF 1.7 拡張レベル 8 ・ PDF 2.0 ¹
	線形化（Web 最適化）PDF による、Web 上でのバイトサービング 大容量出力、10 GB を超える PDF ファイルサイズ
PDF のための ISO 規格群	ISO 32000-1 : PDF 1.7 の標準化バージョン
	ISO 32000-2 : 年月入り改訂版 ISO 32000-2:2020 を含む ¹ : PDF 2.0
	ISO 15930 : PDF/X-3/4/5。グラフィックアート業界向け
	ISO 19005-1/2/3 : PDF/A-1/2/3。アーカイビング用
	ISO 16612-2 : PDF/VT-1。可変・トランザクション印刷用 ISO 14289-1 : PDF/UA-1。ユニバーサルアクセシビリティ用
フォント	TrueType フォント（TTF・TTC）
	PostScript か TrueType のアウトラインを持つ OpenType フォント（TTF・OTF・OTC）
	SVG・COLR（Microsoft）・sbix（Apple）形式の OpenType カラーフォント
	WOFF・WOFF2 ¹ フォント（Web Open Font Format = Web オープンフォント形式）
	ユーザー定義（Type 3）フォントによるビットマップ・ロゴ・カラーフォント
	欧文・日中韓テキストのための OpenType レイアウト機能に対応 ¹ : 合字・スモールキャップス・オールドタイプ数字・スワッシュキャラクター・簡体 / 繁体・縦書き字体など
	Windows または macOS システムにインストールされているフォントにアクセス
	フォントの埋め込みとサブセット化
	EUDC・SING フォント（グリフレット）による日中韓外字キャラクター
	フォールバックフォント : 足りないグリフを他のフォントのグリフへ置き換え
テキスト出力	さまざまなフォントでテキスト出力。テキストに下線・上線・取り消し線
	フォント内のグリフを数値・Unicode 値・グリフ名のいずれかで指定可能
	カーニングによる文字間隔の改善
	異体字セレクターと絵文字異体字シーケンス ¹
	テキストを太字化・斜体化・影付き
テキストフロー組版	パス上テキスト
	柔軟なスタンプメソッド
	テキストを、1 個ないし複数の長方形内に、または任意形状領域内に組版。ハイフネーション（ユーザー定義ハイフネーション位置が必要）、フォントと色の変更、揃え方式、タブ、リーダー、制御コマンド、タグ ¹ を指定可能
	言語固有の処理を用いて高度な改行 : 対象はタイ語など 柔軟な画像貼り付け・組版

表 1.2 PDFlib の機能一覧

分類	機能
	画像または画像のクリッピングパスにテキストを回り込み
表組版	さまざまなユーザー設定に従って表行・表列の寸法を自動計算して配置する表組版機能。複数ページにわたる表も可能。 表セル内にテキスト・画像・SVG グラフィック・PDF ページ・パスオブジェクト・注釈・フォームフィールドを配置可能 表セルを、枠線・背景色オプションを指定して組版可能 貼り付け画像など各種オブジェクトの座標を参照する範囲枠の概念
インタラクティブ要素	フォームフィールド ¹ ：すべてのフィールドオプションと JavaScript を設定可能 PDF の注釈 ¹ （コメント）：PDF リンク・起動リンク（他の文書種別）・Web リンクなど PDF 2.0 の注釈の属性：不透明など ¹ 長方形でない注釈、複数の長方形を持つリンク注釈 ¹ しおり・注釈・ページを開く / 閉じるなど各種イベントに対するアクションを作成 しおりを作成。さまざまなオプション・制御を設定可能 ページ遷移効果（シェード・モザイクなど） リンク・しおり・文書を開くアクションに名前付き移動先 ページラベル（ページのシンボリック名）を作成
マルチメディア	PDF 内に 3D アニメーションを埋め込み、それを JavaScript で制御 音声と動画を埋め込み、または外部ファイルか Web サーバーを参照：スクリーン注釈と表現アクションを使用 ¹
国際化	基本多言語面（BMP）外の追加キャラクターを含む Unicode 14、Unicode 正規形に対応 日本語・中国語・韓国語テキストのための日中韓フォント 日中韓異体字グリフのための表意文字異体字シーケンス（IVS） 日本語・中国語・韓国語テキストの縦書き アラビア語・タイ語・デーヴァナーガリーなどの複雑用字系のキャラクターシェーピング アラビア語・ヘブライ語などの右書き用字系の双方向テキスト組版
SVG ベクトルグラフィック	SVG 形式のベクトル画像を取り込み。SVG・CSS ¹ ・ネットワークリソース ¹ 内の ICC プロファイル、CMYK・スポットカラー
画像	BMP・GIF・PNG・TIFF・JBIG2・JPEG・JPEG 2000・CCITT ラスター画像を読み込み 画像情報をクエリー（ピクセル寸法・解像度・ICC プロファイル・クリッピングパスなど） TIFF・JPEG 画像内のクリッピングパスを使用 TIFF・PNG 画像内のアルファチャンネル（透過）を使用 画像マスク（透過画像に色を適用）、スポットまたは DeviceN カラーを用いて画像に着色
色	グレースケール・RGB（数値・16 進・HTML カラー名）・CMYK・CIE L*a*b* カラー PANTONE®（PANTONE+ も）・HKS® スポットカラー内蔵 ¹ プロセスまたはスポットカラーに基づく DeviceN（n 色）色空間

表 1.2 PDFlib の機能一覧

分類	機能
	ユーザー定義スポットカラー
	複数のプロセスカラーまたはスポットカラーの間のカラーシェーディング（スムーズシェーディング）。パターン塗り・描線
色管理	ICC プロファイルによる ICC ベースカラー ¹
	テキスト・図形・ラスター画像のレンダリングインテント
	ICC プロファイルを埋め込むか外部参照して PDF 2.0・PDF/A・PDF/X での出カインテントとして使用
	多色（xCLR）PDF/X-5n で外部参照出カインテント ICC プロファイル
	PDF 2.0 の文書レベルまたはページレベルの出カインテント ICC プロファイル
アーカイビング	PDF/A-1a/1b・PDF/A-2a/b/u・PDF/A-3a/b/u：XMP 拡張スキーマを含む
グラフィックアート	PDF/X-3・PDF/X-4・PDF/X-4p・PDF/X-5n
	オーバープリント・テキストノックアウト・黒点補償 ¹ などの設定
アクセシビリティ	タグ付き PDF によるアクセシビリティ
	注釈・フォームフィールドなどインタラクティブ要素のタグ付け ¹
	表 ¹ ・ページ装飾の自動タグ付け
	PDF/UA-1 によるユニバーサルアクセシビリティ
	テキストフロー内のタグ付け ¹
	PDF 2.0 のタグ名前空間（タグセット） ¹
	PDF 2.0 の構造種別・属性・タグネ스팅規則 ¹
	構造移動先、すなわち、特定の構造エレメントを指すリンク ¹
可変文書印刷 (VDP)	可変・トランザクション印刷のための PDF/VT-1
ベクトルグラフィック	一般的な基本ベクトルグラフィック要素 ¹ ：線・曲線・円弧・楕円・長方形など
	透過（不透明）・ブレンドモード
	再利用可能なパスオブジェクト ¹ 、クリッピングパスを画像から取り込み
レイヤー	表示を切り替えられるオプションなページ内容
	注釈・フォームフィールドをレイヤー上に配置可
セキュリティ	PDF 文書または添付を暗号化
	Unicode パスワード
	印刷不可・コピー不可など、文書権限設定の指定
地理参照付き PDF	地理空間参照情報を持つ PDF
メタデータ	文書情報：標準フィールド（タイトル・サブタイトル・作成者・キーワード）・ユーザー定義フィールド
	XMP メタデータを文書情報フィールドと同期
	ユーザー定義のカスタム XMP メタデータ

表 1.2 PDFlib の機能一覧

分類	機能
	TIFF・JPEG・JPEG 2000・SVG グラフィック内の XMP 画像メタデータを処理
プログラミング	C・C++・Java・.NET・Objective-C・Perl・PHP・Python・RPG・Ruby の言語バインディング ¹ インターネットからコンテンツをダウンロードするヘルパーメソッド ¹
	仮想ファイルシステムによるインメモリー処理（データベースの画像処理など） PDF 文書をディスクファイルか直接メモリー上へ生成
埋め込みシステム	埋め込みシステムのための、必要リソースを削減した PDFlib Mini Edition (ME)

1. PDFlib 10 の新機能またはかなり改良した機能

1.7 PDFlib+PDI と pCOS インターフェイスの追加的機能

PDFlib+PDI と PPS には、表 1.2 に示した基本的な PDF 生成機能に加えて、表 1.3 に挙げる機能があります。

表 1.3 PDFlib+PDI の追加的機能

分類	機能
PDF 入力 (PDI)	既存 PDF 文書からページを取り込む
	PDF 2.0 までのすべてのバージョンの PDF を取り込み可能
	アクションを含む注釈とフォームフィールドを取り込み ¹ 。注釈の長方形を、配置したページに合わせて変換、すなわち拡大縮小・回転。リンク注釈の機能を、ページを並べ替えても維持
	取り込んだフォームフィールドに流し込み ¹
	暗号化文書を取り込み
	複数の取り込まれた PDF 文書にわたる冗長なオブジェクト（同一フォントなど）を削除
	異常な入力 PDF 文書を修復
pCOS インターフェイス	取り込まれた PDF 文書から PDF/A・PDF/X 出力インテントをコピー
	タグ付き PDF コンテンツ集約：タグ付き PDF 文書のページ群を、構造エレメント群を含めて取り込み
	レイヤー定義（オプションな内容）を取り込み
	非常に大きな文書から内容を取り込む際のパフォーマンス向上 ¹
	pCOS インターフェイスを用いて PDF 文書・ページに関する詳細をクエリー
	PDF 規格・フォームフィールド・署名・ICC プロファイルなど PDF の詳細をクエリーするための追加的な pCOS 疑似オブジェクト ¹

1. PDFlib+PDI 10 の新機能

1.8 PPS と Block Plugin の追加的機能

PDFlib Personalization Server (PPS) でのみ、表 1.2 に示した基本的な PDF 生成機能と、表 1.3 に示した PDF 取り込み機能に加えて、表 1.4 に挙げる機能も利用可能です。

表 1.4 PDFlib Personalization Server (PPS) の追加的機能

分類	機能
可変文書印刷 (VDP)	PDF をパーソナライズ : PDFlib ブロックにテキスト・画像・PDF データ・SVG ベクトルグラフィックを流し込み
	PPS を用いて PDFlib ブロックをプログラマ的に作成
	取り込まれた文書から PDFlib ブロックをコピー
PDFlib Block Plugin	PDFlib Block Plugin で、Windows・macOS 版 Acrobat 上で PDFlib ブロックを対話的に作成
	PPS ブロック流し込みを Acrobat 上でプレビュー
	表示・プレビューするブロックを、ブロックの名称・種別・寸法・位置に基づいてフィルター ¹
	ブロックをプレビューファイルへコピー
	Acrobat 上でブロックを対話的に作成・編集する際のスナップグリッド
	ブロックを小幅または大幅に動かして、シンプルながら正確な位置合わせ ¹
	オーバープリント設定などすべてのグラフィックステートプロパティをブロックプロパティとして利用可能 ¹
	ブロックコンテナの PDF/A・PDF/UA・PDF/X ステータスを転写
	PDF フォームフィールドを PDFlib ブロックに変換して、自動流し込み可能に
	テキストフローブロックを連結して、ブロックであふれたテキストを次のブロックへ
ブロックプラグインに PANTONE®・HKS® スポットカラー名を内蔵 ¹	
ブロックの内容にタグ付け ¹	

1. Block Plugin 6 の新機能・改良機能

1.9 製品別機能一覧

PDFlib ファミリーのどの製品にどの機能があるかを表 1.5 に具体的に挙げます。

表 1.5 製品別機能一覧

機能	API メソッド・オプション	PDFlib	PDFlib+PDI	PPS
基本的な PDF 生成	以下に挙げるもの以外すべて	○	○	○
線形化 (Web 最適化) PDF	<code>PDF_begin_document()</code> の <code>linearize</code> オプション	○ ¹	○	○
PDF を最適化 (効果的でないクライアントコードと、最適化されていない取り込み PDF 文書に対して)	<code>PDF_end_document()</code> の <code>optimize</code> オプション	○ ¹	○	○
PDF/A 準拠のために添付を解析	PDF/A-2 モードにおいて <code>PDF_load_asset()</code> で <code>type=Attachment</code> を使用	○ ¹	○	○
ポートフォリオ作成のために PDF 文書を解析	<code>PDF_add_portfolio_file()</code> の <code>password</code> オプション	○ ¹	○	○
既存の PDF 文書を取り込み	すべての PDI 関数	—	○	○
pCOS を用いて PDF から情報をクエリー	すべての pCOS 関数	—	○	○
ブロックに変換データを流し込み	<code>PDF_fill_*block()</code>	—	—	○
ブロックをプログラマ的に作成	<code>PDF_poca_new()</code> : オプション <code>usage=blocks</code> <code>PDF_begin/end_page_ext()</code> : オプション <code>blocks</code>	—	—	○
ブロックを生成出力へコピー	<code>PDF_process_pdi()</code> : オプション <code>action=copyblock</code> または <code>action=copyallblocks</code>	—	—	○
PPS で利用するための PDFlib ブロックを対話的に作成	Acrobat 用 PDFlib Block Plugin	—	—	○

1. この機能には内部的に PDI を必要とするので、PDFlib ソースコードパッケージでは利用できません

2 PDFlib の言語バインディング

注 すべての PDFlib パッケージに入っているスターターサンプルに目を通すことを強く推奨します。アプリケーション開発の出発点として有用であり、PDFlib プログラミングの重要な点を多数網羅しています。

2.1 C バインディング

PDFlib は、C 言語にいくつかの C++ モジュールを加えたもので書かれています。PDFlib の C バインディングを利用するには、静的または共有ライブラリー（Windows・MVS 上の DLL）を使うことができ、中心となる PDFlib インクルードファイル *pdflib.h* を PDFlib のクライアントのソースモジュールにインクルードする必要があります。あるいは、*pdflibdl.h* を用いて PDFlib DLL を実行時に動的に読み込ませることもできます（詳しくは次項参照）。

注 PDFlib の C バインディングを用いたアプリケーションは、C++ リンカーでリンクを行う必要があります。なぜなら、PDFlib では、C++ で実装された部分をインクルードしているからです。C リンカーを用いると、C++ ランタイムライブラリーに対してアプリケーションが明示的にリンクされていない限り、未解決の外部参照エラーが出る可能性があります。

C におけるエラー処理 PDFlib は、try ~ catch 節による構造化例外処理に対応しています。ですので、C と C++ のクライアントでは、PDFlib で発生した例外をキャッチして、その例外に対して適切に反応することができます。catch 節でクライアントは、問題の正しい性質を記した文字列と、一意な例外番号と、例外を発生させた PDFlib API メソッドの名前とを得ることができます。例外処理を持つ PDFlib C クライアントプログラムの一般的構造は以下のようになります：

```
PDF *p;

/* 新規PDFlibオブジェクトを作成 */
if ((p = PDF_new()) == (PDF *) 0)
{
    printf("PDFlibオブジェクトを作成できませんでした (メモリー不足) \n");
    return(2);
}
PDF_TRY(p)
{
    ...PDFlib命令群...
}
PDF_CATCH(p)
{
    printf("PDFlib例外が発生しました:\n");
    printf("[%d] %s: %s\n",
        PDF_get_errnum(p), PDF_get_apiname(p), PDF_get_errmsg(p));
    PDF_delete(p);
    return(2);
}

PDF_delete(p);
```

`PDF_TRY/PDF_CATCH` はトリッキーなプリプロセッサーマクロとして実装されています。このうちのどちらかをうっかり入れ忘れると、コンパイラから出るエラーメッセージは原因理解が困難なものになってしまう可能性があります。マクロは上記と正確に同じように用いるようにしてください。`TRY` 節と `CATCH` 節の間には一切コードを入れてはいけません (`PDF_CATCH()` 以外)。

`catch` 節の重要な仕事は、`PDF_delete()` と PDFlib オブジェクトへのポインターとを用いて PDFlib の内部構造を解放することです。`PDF_delete()` は必要に応じて出力ファイルも閉じます。例外の後には PDF 文書は使用不能であり、不完全・不整合な状態で取り残されます。もちろん、例外発生時にどのような動作をさせるのが適切かはアプリケーションに依存します。

C と C++ のクライアントで例外をキャッチしない場合、例外発生時のデフォルト動作は、適切なメッセージを標準エラー出力などに出力して抜けるというものです。PDF 出力ファイルは未完成の状態で残されます！ これはライブラリーのルーチンとして適切ではありませんので、エラー処理が重要なアプリケーションでは PDFlib の例外処理機能を活用することを強く推奨します。ユーザー定義の `catch` 節ではたとえば、エラーメッセージを GUI ダイアログボックスに表示し、停止以外の措置をとることができるでしょう。

volatile 変数 `PDF_TRY()` ブロックと `PDF_CATCH()` ブロックの両方で使う変数については特に注意が必要です。1つのブロックから別のブロックへ制御が移ることについてコンパイラは知らないのです、このような場合には不適切なコードを生成してしまう可能性があります (レジスタ変数の最適化など)。幸い、こうした問題を避けるには簡単な規則があります：

注 `PDF_TRY()` ブロックと `PDF_CATCH()` ブロックの両方で使う変数は、`volatile` 宣言するべきです。

`volatile` キーワードを使うと、変数に最適化 (危険をはらむ) を適用してはいけないということをコンパイラに知らせることができます。

try/catch ブロックをネストして例外を再 throw `PDF_TRY()` は、任意の深さにネストすることが可能です。エラー処理をネストした場合、内側の `catch` ブロックで例外を再 throw することによって外側の `catch` ブロックをアクティブにすることができます：

```
PDF_TRY(p)                                /* 外側のtryブロック */
{
    /* ... */

    PDF_TRY(p)                              /* 内側のtryブロック */
    {
        /* ... */
    }
    PDF_CATCH(p)                            /* 内側のcatchブロック */
    {
        /* エラーをクリーンアップ */
        PDF_RETHROW(p);
    }
    /* ... */
}
PDF_CATCH(p)                              /* 外側のcatchブロック */
{
    /* さらにエラーをクリーンアップ */
}
```



```
    PDF_delete(p);
}
```

内側のエラーハンドラーで `PDF_RETHROW()` を呼び出すと、プログラムの実行はただちに外側の `PDF_CATCH()` ブロックの先頭ステートメントへ移ります。

try ブロックを途中で抜ける `PDF_TRY()` ブロックを、たとえば `return` ステートメントによって抜きたい場合、すなわちそれに対応する `PDF_CATCH()` マクロへの呼び出しをバイパスする場合は、以下のように `PDF_EXIT_TRY()` マクロを使って例外機構に通知する必要があります。このマクロから `try` ブロック末尾までの間、他のライブラリー関数は一切呼び出してはいけません：

```
PDF_TRY(p)
{
    /* ... */

    if (error_condition)
    {
        PDF_EXIT_TRY(p);
        return -1;
    }
}
PDF_CATCH(p)
{
    /* エラーをクリーンアップ */
    PDF_RETHROW(p);
}
```

実行時に読み込まれる DLL として PDFlib を利用 たいていのクライアントでは、PDFlib を、静的に結合したライブラリーとして用いるか、リンク時に結合されるダイナミックライブラリーとして用いると考えられますが、それ以外の利用法として、PDFlib DLL を実行時に読み込み、全 API メソッドへのポインターを動的に取得することもできます。この方法は特に、PDFlib DLL を必要時にのみ読み込みたい場合に有用です。PDFlib ではこのような動的な利用法を可能にするためのしくみを特に設けてあります。これは以下の規則にしたがって動作します (`hellodl.c` サンプルを例に示します)：

- ▶ `pdflib.h` のかわりに `pdflibdl.h` をインクルードする。
- ▶ `PDF_new()`・`PDF_delete()` のかわりに `PDF_new_dl()`・`PDF_delete_dl()` を用いる。
- ▶ `PDF_TRY()`・`PDF_CATCH()` のかわりに `PDF_TRY_DL()`・`PDF_CATCH_DL()` を用いる。
- ▶ それ以外のすべての PDFlib 呼び出しについては関数ポインターを用いる。
- ▶ `PDF_get_opaque()` は使ってはいけません。
- ▶ 追加のモジュール `pdflibdl.c` をコンパイルして、アプリケーションをそれにリンクさせる。

Linux で静的ライブラリーから共有ライブラリーを作成するには、以下のコマンドを利用します：

```
mkdir tmp
cd tmp
ar x ../libpdf.a
g++ -shared -o libpdf.so *
```

これで共有ライブラリーができますので、アプリケーションを C++ ランタイムライブラリーにリンクする必要はもうありません。

C で Unicode 文字列 文字列を初期化するには以下の方法があります：

- ▶ 対応するサロゲートペア (UTF-16 符号単位が2つ) を4バイトとしてビッグエンディアンのバイト順序で与える：

```
PDF_fit_textline(p, "\\x3C\\xD8\\xCF\\xDC", 4, 100, 100,  
    "fontname=EmojiOneColor textformat=utf16be fontsize=50");
```

あるいはバイト列はリトルエンディアンのバイト順序で与えることもできます：

```
PDF_fit_textline(p, "\\xD8\\x3C\\xDC\\xCF", 4, 100, 100,  
    "fontname=EmojiOneColor textformat=utf16le fontsize=50");
```

- ▶ PDFlib 文字参照 (122 ページ「5.4.2 文字参照」を参照) は、開発環境から独立であり、以下のように *charref* オプションを用いて有効化できます：

```
PDF_fit_textline(p, "&#x1F0CF;", 0,  
    100, 100, "charref=true fontname=EmojiOneColor fontsize=50");
```

2.2 C++ バインディング

C++ のためのオブジェクト指向なラッパーを PDFlib クライアントから利用可能です。これは `pdflib.hpp` ヘッダーファイルが必要としており、またこのファイルは `pdflib.h` をインクルードしています。テンプレート駆動な PDFlib インターフェイスは、PDFlib とデータをやり取りするための文字列型を複数サポートしています。

wstring に基づくデフォルトインターフェイス API のすべての文字列に対して、C++ の `std::wstring` 型の文字列が使用されます。後述の型のいずれかを選択していない限りはこれがデフォルトの文字列型です。

このデフォルトインターフェイスでは、PDFlib メソッドとやり取りするすべての文字列はネイティブの `wstring` であると前提します。`wstring` の内容は、`wchar_t` データ型のサイズによって、UTF-16 (2 バイトキャラクター群) か UTF-32 (4 バイトキャラクター群) のどちらかで符号化された Unicode 文字列であると見なされます。ソースコード内のリテラル文字列については、ワイド文字列であることを示すには接頭辞 `L` をつける必要があります。`wstring` リテラルを初期化するには以下の方法があります：

- ▶ Unicode コード点を、接頭辞 `u` と 16 進 4 桁か、接頭辞 `U` と 16 進 8 桁で与える。後者では、有効なキャラクターの頭 3 桁は必ず 0 となります (サロゲートペアには対応していません)：

```
wstring unistring = L"U0001FOCF";
```

- ▶ Unicode キャラクターをソースコード内で直接与えて、そのソースファイルを UTF-8 や UTF-16 など然るべき Unicode エンコーディングで保存。このソースのエンコーディングをコンパイラーに伝えておく必要があります。たとえば GCC では UTF-8 がソースファイルのデフォルトエンコーディングですが、他のエンコーディングをコマンドラインオプションで `-finput-charset=UTF-16LE` などと指定することも可能です。Visual Studio の場合はオプションは `/source-charset:utf-8` などとなります。
- ▶ PDFlib 文字参照 (122 ページ「5.4.2 文字参照」を参照) は、開発環境から独立であり、以下のように `charref` オプションを用いて有効化できます：

```
p.fit_textline(L"&#x1FOCF;",  
100, 100, "charref=true fontname=EmojiOneColor fontsize=50");
```

`hello` サンプルなど C++ のサンプルではすべて、デフォルトの `std::wstring` インターフェイスを使った例を示しています。

注 この `wstring` インターフェイスは IBM Z システムでは使えません。かわりに `u16string`/`u32string` インターフェイスを使用してください (以下を参照)

u8string/u16string/u32string に基づくオプションな UTF-8/UTF-16/UTF-32 インターフェイス `std::u16string`・`std::u32string` 型は最低 C++11 が必須であり、`std::u8string` 型は C++20 が必須です。これらのインターフェイスは Unicode キャラクター群を UTF-8・UTF-16・UTF-32 のいずれかの形式で保管します。これらを有効化するには、`pdflib.hpp` をインクルードする前に、`PDFLIB_CPP_U8STRING`・`PDFLIB_CPP_U16STRING`・`PDFLIB_CPP_U32STRING` マクロのいずれか 1 つを設定します。それらの使用例を `hello_u8string`・`hello_u16string`・`hello_u32string` サンプルで示しています。

stringに基づくオプションな UTF-8 インターフェイス C++ のすべてのバージョンが `std::string` 型をサポートしています。このインターフェイスを有効化するには、`pdflib.hpp` をインクルードする前に、`PDFLIB_CPP_STRING` マクロを設定します。`u8string` インターフェイスと異なり、この文字列型には UTF-8 コンパイラサポートがありません。UTF-8 列を持つ文字列を適切に作成・解釈するのはクライアントコード側の役割です。

この UTF-8 ベースの `std::string` インターフェイスの使用例を `hello_string` サンプルで示しています。

C++ におけるエラー処理 PDFlib API メソッドは、エラー発生時には C++ 例外を発生させます。この例外はクライアントコード内で C++ の `try ~ catch` 節を用いてキャッチされる必要があります。詳細なエラー情報提供のため、PDFlib クラスにはパブリッククラス `PDFlib::Exception` があります。このパブリッククラスは、詳しいエラーメッセージ取得のためのメソッドと、例外番号取得のためのメソッドと、例外を発生させた API メソッドの名前を取得するためのメソッドとを公開しています。

PDFlib ルーチンが発生させる C++ 準拠の例外は規則どおりに動作します。以下のコード断片は、PDFlib が発生させた例外をキャッチします：

```
try {
    PDFlib p;
    ...PDFlib命令群...
} catch (PDFlib::Exception &ex) {
    wcerr << L"PDFlib例外が発生しました: " << endl
           << L "[" << ex.get_errnum() << L"] " << ex.get_apiname()
           << L ": " << ex.get_errmsg() << endl;
}
```

実行時に読み込まれる DLL として PDFlib を利用 C++ バインディングでは、PDFlib を自分のアプリケーションに実行時に結合させることもできます (33 ページ「実行時に読み込まれる DLL として PDFlib を利用」を参照)。動的読み込みは、`pdflib.hpp` をインクルードするアプリケーションモジュールをコンパイルする際に、以下のようにして有効にすることができます：

```
#define PDFCPP_DL 1
```

これに加えて、追加モジュール `pdflibdl.c` をコンパイルして、できるオブジェクトファイルに自分のアプリケーションをリンクさせる必要があります。動的読み込みの詳細はこの PDFlib オブジェクト内に隠蔽されていますので、これは C++ API に影響を与えません：すべてのメソッド呼び出しは、動的読み込みが有効にされているか否かにかかわらず同じに見えます。

2.3 Java バインディング

PDFlib Java 版をインストール PDFlibバインディングが動作するには、Java VMが、PDFlib Java ラッパーと PDFlib Java パッケージを利用できるようにする必要があります。PDFlib は以下のパッケージ名を持つ Java パッケージとしてまとめられています：

```
com.pdflib.pdflib
```

このパッケージは *pdflib.jar* ファイルの中にあつて、クラス *pdflib.PDFlibException* を内容としています。このパッケージをアプリケーションで利用可能にするには、*pdflib.jar* を *CLASSPATH* 環境変数に追加する必要があります。または、Java コンパイラとランタイムへの呼び出しの中に *-classpath pdflib.jar* オプションを加えるか、これと等価の手順を Java IDE 中で踏む必要があります。JDK では、Java VM が既知のディレクトリを検索するように設定することができます。具体的には、*java.library.path* プロパティにディレクトリの名前を設定します。たとえば次のように記述します。

```
java -Djava.library.path=. hello
```

このプロパティの値をチェックするには以下のようにします：

```
System.out.println(System.getProperty("java.library.path"));
```

上記に加え、プラットフォームによって以下の手順を行う必要があります：

- ▶ Unix : *libpdflib_java.so* ライブラリー (macOS の場合 : *libpdflib_java.jnilib*) は、共有ライブラリーのためのデフォルト格納場所のうちのいずれかに置くか、適切に設定したディレクトリに置く必要があります。
- ▶ Windows : *pdflib_java.dll* ライブラリーは、Windows のシステムディレクトリに置くか、PATH 環境変数で列挙されたディレクトリに置く必要があります。

Java におけるエラー処理 Java バインディングは、PDFlib のエラーをネイティブ Java 例外に翻訳します。例外が起きると PDFlib は *PDFlibException* クラスのネイティブ Java 例外を発生させます。この Java 例外は通常の try/catch 技法で処理できます：

```
try {
    p = new pdflib();
    ...PDFlib命令群...
} catch (PDFlibException e) {
    System.err.println("PDFlib例外が発生しました:");
    System.err.println("[ " + e.get_errnum() + " ] " + e.get_apiname() +
        ": " + e.get_errmsg());
} catch (Exception e) {
    System.err.println(e);
} finally {
    if (p != null) {
        p.delete();          /* PDFlibオブジェクトを削除 */
    }
}
```

PDFlib は適切な *throws* 節を宣言しますので、クライアントのコードでは、起こりうるすべての PDFlib 例外をキャッチするか、それらを自分で宣言しなければなりません。

Java で Unicode 文字列 Java の *String* オブジェクトは UTF-16 形式の文字列を表現しています。ですので、BMP 外の、すなわち U+FFFF を超える Unicode 値についてはサロゲートペアとして表現する必要があります。BMP 外の Unicode コード点を 1 個の *char* か *Character* で表すことはできません。文字列を初期化するには以下の方法があります：

- ▶ Unicode コード点を *int* 値として与えて、それを *String* へ変換：

```
String unistring = new String(Character.toChars(0x1F0CF));
```

- ▶ 対応するサロゲートペア (UTF-16 符号単位が 2 つ) を与える：

```
String unistring = new String("\uD83C\uDCCF");
```

- ▶ Unicode キャラクターをソースコード内で直接与えて、そのソースファイルを UTF-8 や UTF-16 など然るべき Unicode エンコーディングで保存。このソースのエンコーディングをコンパイラに伝えておく必要があります。そのためには *javac* オプション *encoding="utf8"* で直接伝えるか、あるいは IDE で適切な設定によって伝えます。
- ▶ PDFlib 文字参照 (122 ページ「5.4.2 文字参照」を参照) は、開発環境から独立であり、以下のように *charref* オプションを用いて有効化できます：

```
p.fit_textline("&#x1F0CF;",  
100, 100, "charref=true fontname=EmojiOneColor fontsize=50");
```

PDFlib の Javadoc 仕様書 PDFlib パッケージには、PDFlib の Javadoc 仕様書が含まれています。この Javadoc は、すべての PDFlib API メソッドの短い記述しか含んでいませんので、詳しくは PDFlib API リファレンスを参照してください。

PDFlib の Javadoc を Eclipse で設定するには以下のように操作します：

- ▶ パッケージ・エクスプローラーで Java プロジェクトを右クリックし、「**Javadoc ロケーション**」を選択します。
- ▶ 「**ブラウズ...**」をクリックして、Javadoc が置かれているパス (PDFlib パッケージに含まれています) を選びます。

この手順をふめば、「**Java 参照**」パースペクティブや「**ヘルプ**」メニューなどで PDFlib の Javadoc をブラウズできるようになります。

2.4 .NET バインディング

注 この.NETバインディングは、対応しているすべてのプラットフォームのための汎用のパッケージとして提供されます。ただし、それでも、プラットフォームごとのライセンスキーが必要であり、このライセンスを他のプラットフォームへ転用することはできません。

.NET バインディングの一般的特性を表 2.1 に挙げます。

表 2.1 .NET バインディングの一般的特性

トピック	.NET バインディングの実装
.NET 対応	.NET Standard 2.0 以上。例：.NET Core 2/3、.NET 5/6
.NET Framework 対応	.NET Framework 4.6.1 以上
ターゲットプラットフォーム	Windows・Linux・macOS。プラットフォーム/アーキテクチャーの組み合わせの正確な一覧については <code>system-requirements.txt</code> を参照してください
ダウンロードパッケージ	すべての対応プラットフォームのためのユニバーサル zip パッケージ
パッケージ内容	NuGet パッケージにマネージドとネイティブのライブラリー、説明書、サンプルを含む
実装	マネージドコードを持つ C# アセンブリー <code>PDFlib_dotnet.dll</code> と、ネイティブコードを持つプラットフォーム固有の共有ライブラリー <code>PDFlib_dotnet_native.[dll so dylib]</code>
.NET 統合	明示的 PInvoke を通じた C# Interop
バージョンング方式	バージョンング方式は .NET の規則に準拠。 .NET バージョン番号は、NuGet キャッシュや <code>.csproj</code> プロジェクトファイルの中で見ることができます。これは製品のメジャーとマイナーのリリース番号とは異なっています。両方のバージョンング方式の間のマッピングは <code>compatibility.txt</code> にあります。

PDFlib.NET をインストール 製品は NuGet パッケージとして与えられており、そのパッケージ名はバージョン番号を含んでいます。内部的には .NET 用パッケージは .NET の規則に従ったセマンティックなバージョンを使用しています。結果として、内部的なバージョン番号は、通常の製品のバージョン番号とは異なっています。両方の付番方式に対するマッピングテーブルが `compatibility.txt` にあります。

この NuGet パッケージをローカルにインストールするには以下のいずれかの方式を行います：

- ▶ `dotnet` コマンドラインツール (全プラットフォーム)。この方式については次項で詳説します。
- ▶ Visual Studio の Package Manager UI (Windows・macOS)
- ▶ Visual Studio の Package Manager Console (Windows)
- ▶ `nuget` コマンドラインツール (全プラットフォーム)

提供しているサンプル群のためのプロジェクトファイル群は、ターゲットフレームワーク .NET Core 3.1 に対して作成されています (ターゲットフレームワークモニカー `TFM=netcoreapp3.1`) ので、.NET Core 3.1 以上で動作します。他のフレームワークをターゲットにするには、たとえば以下のようにプロジェクトファイル内の TFM を合わせる必要があります。

```
<PropertyGroup>  
  <OutputType>Exe</OutputType>
```

```
<TargetFramework>net6.0</TargetFramework>
</PropertyGroup>
```

dotnet コマンドラインツールを用いて .NET 用 PDFlib をインストール 提供している *hello* プロジェクトを例に、*dotnet* ユーティリティを用いてインストール・構成・ビルドを行うプロセスを説明します：

- ▶ 製品パッケージを、好きなディレクトリーへ解凍します。
- ▶ コマンドシェルで、*hello* プロジェクトディレクトリーへ *cd* します：

```
cd <installdir>\bind\dotnet\csharp\hello
```

- ▶ (このステップは、提供しているサンプル群では必要ありません。ローカルの *NuGet.Config* ファイルを用いてパッケージを参照しているからです) NuGet パッケージを、アプリケーションのプロジェクトディレクトリーへ複製します：

```
<installdir>/bind/dotnet/PDFlib_dotnet.X.Y.Z.nupkg
```

- ▶ (このステップは、提供しているサンプル群では必要ありません。すでに PDFlib への参照を含んでいるからです) 以下のコマンドを、適切なバージョン番号とともに入力します (バージョン番号は *.nupkg* ファイルの名前の中にあります)：

```
dotnet add package PDFlib_dotnet.X.Y.Z
```

このコマンドは PDFlib 参照を *.csproj* プロジェクトファイルに追加します。これはまた、PDFlib をローカル NuGet パッケージ内に、もしまだ存在していなければインストールします。例：

```
~/nuget/packages/pdflib_dotnet/X.Y.Z
```

このキャッシュ化があるので、**.nupkg* を複製する必要があるのは最初のプロジェクトに対してのみです。それより後のプロジェクト群では、パッケージファイルはキャッシュから取られるので必要ありません。

- ▶ これで、*hello* プロジェクトをビルドして実行して実行することができます：

```
dotnet build
dotnet run
```

結果として、アプリケーションディレクトリーに *hello.pdf* 出力文書が生成されているのが見つかるはずです。

そのまま使える構成を持った完全な作成例がすべてのパッケージに入っています。*.NET* バインディングを適切に参照したら、それ以後は *PDFlib_dotnet.PDFlib*・*PDFlib_dotnet.PDFlibException* クラスを使用できます。

.NET Framework プロジェクト内に PDFlib.NET をインストール Visual Studio で *.NET Framework* プロジェクトに *PDFlib.NET* を追加するには以下のとおり操作してください(この方式は *C++*・*ASP.NET* プロジェクトでは使えません)：

- ▶ NuGet パッケージマネージャーで *PDFlib_dotnet.x.y.z.nupkg* への参照を追加。すると、*PDFlib_dotnet*への参照と*packages.config*管理ファイルがプロジェクトに追加されます。
- ▶ *packages.config* のコンテキストメニューで「*package.config* を *PackageReference* に移行する ...」を選択することによって、この参照 (初期状態では *PDFlib_dotnet.dll* だけを指しています) を NuGet パッケージ参照へ変換。

.NET におけるエラー処理 .NET バインディングは .NET の例外に対応しており、実行時に問題が発生すると、詳細なエラーメッセージを持った例外を発生させます。こうした例外をキャッチして適切に対処するのはクライアント側の役目です。そうしない場合、.NET Framework が例外をキャッチして、たいていはアプリケーションを停止させます。

PDFlib は、例外関連情報を伝えるために、メンバー `get_errnum`・`get_errmsg`・`get_apiname` を持つ自前の例外クラス `PDFlib_dotnet.PDFlibException` を定義しています。PDFlib は `IDisposable` インターフェイスを実装しているので、クライアントは `Dispose()` メソッドを呼び出してクリーンアップが可能です。

クライアントコードでは、PDFlib が発生させる .NET の例外を、通常の `try` ~ `catch` 構造を用いて処理できます：

```
try {
    p = new PDFlib();
    ...PDFlib命令群...
} catch (PDFlibException e) {
    // pdflibが投げた例外をキャッチした
    Console.WriteLine("PDFlib例外が発生しました:\n");
    Console.WriteLine("[{0}] {1}: {2}\n",
        e.get_errnum(), e.get_apiname(), e.get_errmsg());
} finally {
    if (p != null) {
        p.Dispose();
    }
}
```

C# で Unicode 文字列 C# の `String` オブジェクトは UTF-16 形式の文字列を表現しています。ですので、BMP 外の、すなわち U+FFFF を超える Unicode 値についてはサロゲートペアとして表現する必要があります。BMP 外の Unicode コード点を 1 個の *char* か *Character* で表すことはできません。文字列を初期化するには以下の方法があります：

- ▶ Unicode コード点を *int* 値として与えて、それを *String* へ変換：

```
String unistring = Char.ConvertFromUtf32(0x1F0CF);
```

- ▶ 対応するサロゲートペア (UTF-16 符号単位が 2 つ) を与える：

```
String unistring = "\uD83C\uDCCF";
```

- ▶ Unicode キャラクターをソースコード内で直接与えて、そのソースファイルを UTF-8 や UTF-16 など然るべき Unicode エンコーディングで保存。このソースのエンコーディングをコンパイラーに伝えておく必要があります。そのためには直接伝えるか、あるいは IDE で適切な設定によって伝えます。
- ▶ PDFlib 文字参照 (122 ページ「5.4.2 文字参照」を参照) は、開発環境から独立であり、以下のように *charref* オプションを用いて有効化できます：

```
p.fit_textline("&#x1F0CF;",
    100, 100, "charref=true fontname=EmojiOneColor fontsize=50");
```

2.5 Objective-C バインディング

C・C++ 言語バインディングを Objective-C で使うこともできますが、Objective-C 用の真正な言語バインディングもあります。この *PDFlib* フレームワークは、macOS で使えるものと iOS で使えるものがあります。

これらのフレームワークは C・C++・Objective-C 用の言語バインディングを含んでいます。

PDFlib Objective-C 版をインストール PDFlib を自分のアプリケーションで使うためには、*PDFlibframework* をディレクトリー */Library/Frameworks* へコピーする必要があります。このフレームワークを別の場所にインストールすることも可能ですが、Apple の *install_name_tool* の使用が必要です。このツールについてはここでは説明しません。PDFlib メソッド定義を持つ *PDFlib_objc.h* ヘッダーファイルをアプリケーションソースコード内で取り込む必要があります：

```
#import "PDFlib/PDFlib_objc.h"
```

PDFlib フレームワークをアプリ内に埋め込むには、Xcode のコード署名はバージョン番号 *A* を持つフレームワークを受け付けるのに対して、PDFlib 製品は数字のバージョン番号を用いています。これを解決するには、以下のように適切に名づけたフレームワークフォルダーを作成します：

```
cd PDFlib.framework/Versions
mv 10.x.y A
rm Current
ln -s A Current
```

引数命名規則 PDF メソッド呼び出しにあたっては、引数を、以下の規則に従って与える必要があります：

- ▶ 1 番目の引数の値は、メソッド名の直後に、コロンキャラクターで区切って与えます。
- ▶ その後の各引数については、その引数の名前と値（これも互いにコロンキャラクターで区切って）を与える必要があります。引数名は、PDFlib API リファレンスか *PDFlib_objc.h* 内で見つけることができます。

たとえば、PDFlib API リファレンス内の以下の行は：

```
void begin_page_ext(double width, double height, String optlist)
```

以下の Objective-C メソッドに対応します：

```
- (void) begin_page_ext: (double) width height: (double) height optlist: (NSString *) optlist;
```

よって、アプリケーションから以下のような呼び出しを行う必要があることになります：

```
[pdflib begin_page_ext:595.0 height:842.0 optlist:@""];
```

コード補完のための Xcode Code Sense は PDFlib フレームワークで使用できます。

Objective-C におけるエラー処理 Objective-C バインディングは、PDFlib エラーを、ネイティブ Objective-C 例外へ翻訳します。実行時の問題が発生したときには、PDFlib は、*PDFlibException* クラスのネイティブ Objective-C 例外を発生させます。この例外は通常の *try/catch* 機構で処理できます：

```
PDFlib p = NULL;

@try {
    PDFlib *pdflib = [[PDFlib alloc] init];
    ...PDFlib命令群...
}
@catch (PDFlibException *ex) {
    NSLog(@"Error %ld in %@(): %@\n",
        [ex get_errnum], [ex get_apiname], [ex get_errmsg]);
}
@catch (NSEException *ex) {
    NSLog(@"%@: %@", [ex name], [ex reason]);
}
@finally {
    if (pdflib)
        [pdflib release];
}
```

get_errmsg メソッドのほかに、例外オブジェクトの *reason* フィールドを使ってエラーメッセージを取得することもできます。

2.6 Perl バインディング

Perl 用 PDFlib ラッパーは、C ラッパーファイル 1 個と Perl パッケージモジュール 2 個で構成されています。このモジュールのうち 1 個は各 PDFlib API メソッドの Perl 版を提供し、もう 1 個は PDFlib オブジェクトのためのものです。C モジュールは、Perl インタプリターが実行時に読み込む共有ライブラリーをビルドするために使われます。この処理には Perl パッケージモジュールも使用されます。Perl スクリプトでは `use` ステートメントで共有ライブラリーモジュールを参照します。

PDFlib Perl 版をインストール Perl の拡張機構は共有ライブラリーを、実行時に DynaLoader モジュールを通じて読み込みます。Perl の実行形式が、共有ライブラリー対応付きでコンパイルされている必要があります（大多数の Perl の設定ではそうになっていません）。

PDFlib バインディングが動作するためには、Perl インタプリターが PDFlib Perl ラッパーとモジュール `pdflib_pl.pm`・`PDFlib/PDFlib.pm` を利用できるようになっている必要があります。以下に述べるプラットフォーム依存な方式を用いることもできますし、`-I` コマンドラインオプションを用いて Perl の `@INC` モジュール検索パスにディレクトリーを追加することもできます：

```
perl -I/path/to/pdflib hello.pl
```

Unix Perl は、`pdflib_pl.so` (macOS の場合：`pdflib_pl.bundle`)・`pdflib_pl.pm`・`PDFlib/PDFlib.pm` を、カレントディレクトリーで検索するか、以下の Perl コマンドで出力されるディレクトリーで検索します：

```
perl -e 'use Config; print $Config{sitearchexp};'
```

Perl は `auto/pdflib_pl` サブディレクトリーも検索します。上記コマンドの一般的な出力は次のようになります。

```
/usr/lib/perl5/site_perl/5.32/i686-linux
```

Windows DLL `pdflib_pl.dll` とモジュール `pdflib_pl.pm`・`PDFlib/PDFlib.pm` が、カレントディレクトリーで検索されるか、以下の Perl コマンドで出力されるディレクトリーで検索されます：

```
perl -e "use Config; print $Config{sitearchexp};"
```

上記コマンドの一般的な出力は次のようになります。

```
C:\Program Files\Perl5.32\site\lib
```

Perl におけるエラー処理 Perl バインディングは、PDFlib のエラーをネイティブ Perl 例外に翻訳します。Perl の例外は、適切な言語構造を適用することにより取り扱うことができます。すなわち、問題の起こりそうな箇所を次のように挟みます：

```
eval {
    my $p = new PDFlib::PDFlib;
    ...PDFlib命令群...
};
if ($@) {
    die("$0: PDFlib例外が発生しました:\n$@");
}
```

Perl で Unicode 文字列 文字列を初期化するには以下の方法があります：

- ▶ Unicode コード点を 16 進値として与える（サロゲートペアには対応していません）：

```
$unistring="\x{1F0CF}";
```

- ▶ Unicode キャラクター名を与える：

```
unistring = "\N{PLAYING CARD BLACK JOKER}";
```

- ▶ Unicode キャラクターをソースコード内で直接与えて、そのソースファイルを然るべき Unicode エンコーディングで保存。UTF-8 のソースファイルには *use utf8* プラグマを含める必要があります。UTF-16 のソースファイルは先頭に BOM を付ける必要があります。
- ▶ PDFlib 文字参照（122 ページ「5.4.2 文字参照」を参照）は、開発環境から独立であり、以下のように *charref* オプションを用いて有効化できます：

```
p.fit_textline("&#x1F0CF;",  
100, 100, "charref=true fontname=EmojiOneColor fontsize=50");
```

2.7 PHP バインディング

PDFlib PHP 版をインストール PDFlib を PHP で使う際のさまざまな方式やオプションに関する詳細な情報は、*PDFlib-in-PHP-HowTo.pdf* 文書に記載してあります。この文書はディストリビューションパッケージに含まれており、また、PDFlib ウェブサイトにも掲載してあります。

PDFlib ライブラリーについて PHP が認識するよう PHP を構成する必要があります。*php.ini* に以下の行を追加してください：

```
extension=php_pdflib
```

PHP は、ライブラリーを検索するとき、Unix では *php.ini* の中の *extension_dir* 変数で指定されたディレクトリーの中を捜し、Windows ではその他に標準のシステムディレクトリーの中も捜します。どのバージョンの PHP PDFlib バインディングがインストールしてあるかを確認するには、次のような一行 PHP スクリプトを用います：

```
<?phpinfo()?>
```

そうすると、カレントの PHP の設定に関して、長い情報ページが表示されます。このページの中で *PDFlib* という見出しのセクションをチェックしてください。

PHP に合わせた PDFlib 関数のエラー戻り値 PHP では、メソッド内でのエラー発生時に値 0 (FALSE) を返す方式が用いられているので、PDFlib API メソッドはすべて、エラー発生時に -1 でなく 0 を返すよう変更してあります。この変更については、PDFlib API リファレンスのメソッド解説に記してあります。ただし Java のコード断片では、エラー時に -1 を返す通常の PDFlib の方式を用いているので、読む際には注意してください。

PHP におけるファイル名処理 PDF や画像などのディスク上のパスの指定の無いファイル名や相対パスのファイル名は、PHP の Unix 版と Windows 版とは異なった取り扱いを受けます：

- ▶ Unix システム上の PHP は、パス部分をまったく持たないファイルを、スクリプトが置かれているディレクトリーの中で検索します。
- ▶ Windows 上の PHP は、パス部分をまったく持たないファイルを、PHP DLL が置かれているディレクトリーの中でのみ検索します。PDFlib は UTF-8 符号化されたファイル名を受け付けますが、PHP の *dirname()* メソッドは Windows ではカレントのシステムコードページを使用します。この場合には、ディレクトリーかファイルの名前を UTF-8 へ変換する必要があります。詳しくは 111 ページ「5.2.3 デフォルト UTF-8 対応のある言語バインディング」を参照してください。例：

```
$searchpath = dirname(dirname(__FILE__)).'/data';  
$searchpath = $p->convert_to_unicode("auto", $searchpath, "outputformat=utf8");
```

プラットフォームに依存しないファイル名の取り扱いを行うには、SearchPath 機能の利用を強く推奨します (59 ページ「3.1.4 リソース構成とファイル検索」参照)。

PHP における例外処理 PHP は構造化例外処理に対応しているため、PDFlib 例外は PHP 例外として発生します。PDFlib はクラス *PDFlibException* の例外を発生させます。このクラスは PHP の標準の Exception クラスを派生させたものです。標準的な *try ~ catch* 技法を用いて PDFlib 例外を扱うことができます：

```

try {
    $p = new PDFlib();
    ...PDFlib命令群...
} catch (PDFlibException $e) {
    print "PDFlib例外が発生しました:\n";
    print "[" . $e->get_errnum() . "] " . $e->get_apiname() . ": "
        $e->get_errmsg() . "\n";
}
catch (Throwable $e) {
    die("PHP例外が発生しました: " . $e->getMessage() . "\n");
}

```

Eclipse と Zend Studio を用いて開発 PHP Development Tools (PDT) は Eclipse と Zend Studio による PHP 開発に対応しています。PDT は、以下に示す手順で、状況依存ヘルプに対応するよう設定することができます。

PDFlibをEclipse設定に追加して、すべてのPHPプロジェクトに知られるようにします:

- ▶ 「Window」→「Preferences」→「PHP」→「Source Paths」→「PHP Libraries」→「New...」を選択してウィザードを起動します。
- ▶ 「User library name」に *PDFlib* と入力し、「Add External folder...」をクリックしてフォルダー *bind\php\Eclipse PDT* を選びます。

既存または新規の PHP プロジェクトで、PDFlib ライブラリーへの参照を以下のように追加することができます:

- ▶ PHP Explorer で PHP プロジェクトを右クリックし、「Include Path」→「Configure Include Path...」を選択します。
- ▶ 「Libraries」タブへ行き、「Add Library...」をクリックし、「User Library」→「PDFlib」を選択します。

これらの手順をふめば、PHP Explorer ビューの *PHP Include Path/PDFlib/PDFlib* ノード配下で PDFlib メソッドの一覧を閲覧できるようになります。新しい PHP コードを書いている時、Eclipse は、すべての PDFlib メソッドに対するコード補完と状況依存ヘルプで支援します。

PHP で Unicode 文字列 文字列を初期化するには以下の方法があります:

- ▶ Unicode コード点を 16 進形式で与える (サロゲートペアには対応していません):

```
$unistring = "\u{1F0CF}";
```

- ▶ Unicode キャラクターを PHP ソース内で直接与えて、そのソースファイルを UTF-8 形式で保存。
- ▶ PDFlib 文字参照 (122 ページ「5.4.2 文字参照」を参照) は、開発環境から独立であり、以下のように *charref* オプションを用いて有効化できます:

```

p.fit_textline("&#x1F0CF;",
    100, 100, "charref=true fontname=EmojiOneColor fontsize=50");

```

2.8 Python バインディング

PDFlib Python 版をインストール Pythonの拡張機構は共有ライブラリーを実行時に読み込むことによって動作します。PDFlib バインディングを動作させるには、以下のように、Python インタプリターがPython用PDFlibライブラリーを利用できるようにする必要があります。このラッパーは、PYTHONPATH 環境変数に列挙されたディレクトリー内で検索されます。Python ラッパーの名前はプラットフォームによって異なります：

- ▶ Unix・macOS : `pdflib_py.so`
- ▶ Windows : `pdflib_py.pyd`

PDFlib ライブラリーのほかに、以下のファイルもライブラリーと同じディレクトリーに入れておく必要があります：

- ▶ `PDFlib/PDFlib.py`
- ▶ `PDFlib/__init__.py` (Python 2.7 の場合のみ)

Python におけるエラー処理 Pythonバインディングはエラー時に `PDFlibException` を発生させます。Python 例外は通常の `try/catch` で取り扱うことができます：

```
try:
    p = PDFlib()

    ...PDFlib命令群...
except PDFlibException:
    print("PDFlib例外が発生しました:
        print("[%d] %s: %s" % (ex.errnum, ex.apiname, ex.errmsg))

except Exception:
    print(ex)

finally:
    if p:
        p.delete()
```

Python で Unicode 文字列 文字列を初期化するには以下の方法があります：

- ▶ Unicode コード点を 16 進 8 桁で与える：

```
$unistring = "\U0001F0CF";
```

- ▶ サロゲートは以下のように与えることもできます：

```
unistring = "\uD83C\uDCCF".encode('utf-16', 'surrogatepass').decode('utf-16')
```

- ▶ Unicode キャラクターをPythonソース内で直接与えて、そのソースファイルをUTF-8形式で保存。
- ▶ PDFlib 文字参照 (122 ページ「5.4.2 文字参照」を参照) は、開発環境から独立であり、以下のように `charref` オプションを用いて有効化できます：

```
p.fit_textline("&#x1F0CF;",
    100, 100, "charref=true fontname=EmojiOneColor fontsize=50");
```


2.9 RPG バインディング

PDFlib は、PDFlib 関数を埋め込まれた ILE-RPG プログラムをコンパイルするために必要なすべてのプロトタイプといくつかの有用な定数とを定義した */copy* モジュールを提供しています。

Unicode 文字列処理 PDFlib が提供する関数はすべて、可変長の Unicode 文字列を引数として用いているので、*%UCS2* ビルトイン関数を使ってシングルバイト文字列を Unicode 文字列に変換する必要があります。PDFlib の関数が返す文字列はすべて可変長の Unicode 文字列です。これらの Unicode 文字列をシングルバイト文字列に変換するには *%CHAR* ビルトイン関数を使います。

注 *%CHAR*・*%UCS2* 関数は、カレントジョブの CCSID を使って文字列を Unicode と変換します。PDFlib に同梱の例では CCSID 37 (US EBCDIC) をベースとしています。このコードページは *CHGJOB CCSID(37)* を用いて設定できます。この例をこれ以外のコードページで走らせると、オプションリスト内のいくつかのキャラクター (*{[]}* など) が正しく翻訳されないことがあります。

すべての文字列は可変長文字列として渡されるので、文字列長を明示的に指定する必要のあるさまざまな関数では *length* 引数を渡してはいけません (可変長文字列の長さは、文字列の先頭 2 バイトに格納されています)。

PDFlib 用 RPG プログラムをコンパイル・バインド PDFlib 関数を RPG から利用するには、コンパイル済みの PDFLIB・PDFLIB_RPG サービスプログラムが必要です。PDFlib の定義をコンパイル時にインクルードするには、*/copy* メンバーの名前を ILE-RPG プログラムの *D* スペックの中で指定する必要があります：

```
d/copy QRPGLSRC,PDFLIB
```

PDFlib ソースファイルライブラリーがライブラリーリストの先頭にはない場合は、そのライブラリーも指定しなければなりません：

```
d/copy PDFsrc1ib/QRPGLSRC,PDFLIB
```

ILE-RPG プログラムのコンパイルを開始する前に、PDFlib に同梱の PDFLIB・PDFLIB_RPG サービスプログラムを含むバインドディレクトリーを作成しておく必要があります。以下の例は、ライブラリー PDFLIB の中の PDFLIB というバインドディレクトリーを作成したい場合の指定です：

```
CRTBNDDIR BNDDIR(PDFLIB/PDFLIB) TEXT('PDFlib Binding Directory')
```

バインドディレクトリーを作成した後は、PDFLIB・PDFLIB_RPG サービスプログラムをバインドディレクトリーに追加する必要があります。次の例は、さきに作成したバインドディレクトリーにライブラリー PDFLIB の中のサービスプログラム PDFLIB を追加したい場合の指定です。

```
ADDBNDDIRE BNDDIR(PDFLIB/PDFLIB) OBJ((PDFLIB/PDFLIB *SRVPGM))
```

これで、*CRTBNDRPG* コマンド (または PDM 中のオプション 14) を用いてプログラムをコンパイルできるようになりました：

```
ADDLIBLE LIB(PDFLIB)
CRTBNDRPG PGM(PDFLIB/HELLO) SRCFILE(PDFLIB/QRPGLESRC) SRCMBR(*PGM) DFTACTGRP(*NO)
BNDDIR(PDFLIB/PDFLIB)
```

生成されたプログラムを実行する前に、以下のコマンドを適用する必要があります：

```
chgcurdir '/pdflib/pdflib/x.y/x.y.z/bind/rpg'
```

生成された PDF 文書は、同一ディレクトリー内に見つけることができます。

引数は、表 2.2 に挙げるデータ型に従って PDFlib API へ渡される必要があります。

表 2.2 RPG バインディングにおけるデータ型

API データ型	RPG バインディングにおけるデータ型
文字列データ型	Unicode 文字列 (%ucs2 を使用)
バイナリーデータ型	data

RPG におけるエラー処理 ILE-RPG で書かれた PDFlib クライアントでは、ILE-RPG が提供する *monitor/on-error/endmon* エラー処理機構を利用することができます。あるいは、ILE-RPG の **PSSR* グローバルエラー処理サブルーチンを用いて例外をモニターする方法もあります。ジョブログにはエラー番号と、失敗した関数、および例外の原因が示されます。PDFlib は呼び出し側プログラムへエスケープメッセージを送ります。

```
c      eval      p=PDF_new
*
c      monitor
*
c      eval      doc=PDF_begin_document(p:%ucs2('/tmp/my.pdf'):docoptlist)
:
:
*      エラー処理
c      on-error
*      このエラーについて何かします
*      PDFlibオブジェクトの解放を忘れないように
c      callp     PDF_delete(p)
c      endmon
```

2.10 Ruby バインディング

PDFlib Ruby 版をインストール Ruby の拡張機構、共有ライブラリーを動作時に読み込むことによって動作します。PDFlib バインディングが動作するには、Ruby 用の PDFlib 拡張ライブラリーの場所を Ruby インタプリターが知っていて利用できる必要があります。たいていの場合、このライブラリー (Windows・Unix の場合：*PDFlib.so*、macOS の場合：*PDFlib.bundle*) がインストールされる場所は、ローカル ruby インストールディレクトリーの *site_ruby* ブランチの中であり、すなわち、次のような名前のディレクトリーの中にインストールされます：

```
/usr/local/lib/ruby/site_ruby/<バージョン>/
```

ただし Ruby は、これ以外のディレクトリーへも拡張を探しに行きます。これらのディレクトリーの一覧を取得するには、次の ruby コールを用いることができます：

```
ruby -e "puts $:"
```

この一覧はたいていの場合、カレントディレクトリーを含んでいますので、テスト目的においては、PDFlib 拡張ライブラリーとスクリプトを同じディレクトリーに入れるだけでよいでしょう。

Ruby におけるエラー処理 Ruby バインディングは、PDFlib 例外をネイティブ Ruby 例外に翻訳するエラーハンドラーをインストールします。こうした Ruby 例外は、通常の *rescue* 技法で扱うことができます：

```
begin
  p = PDFlib.new
  ...PDFlib命令群...
rescue PDFlibException => pe
  print "PDFlib例外が発生しました:\n"
  print "[" + pe.get_errnum.to_s + "]" + pe.get_apiname + ": "+pe.get_errmsg+ "\n"
rescue Exception => e
  print e.backtrace.join("\n") + "\n" + e.to_s
ensure
  p.delete() if p
end
```

Ruby で Unicode 文字列 文字列を初期化するには以下の方法があります：

- ▶ Unicode コード点を 16 進 4 桁か 5 桁か 6 桁で与える (サロゲートペアには対応していません)：

```
$unistring = "\u{1F0CF}";
```

- ▶ Unicode キャラクターを Ruby ソース内で直接与えて、そのソースファイルを UTF-8 形式で保存。
- ▶ PDFlib 文字参照 (122 ページ「5.4.2 文字参照」を参照) は、開発環境から独立であり、以下のように *charref* オプションを用いて有効化できます：

```
p.fit_textline("&#x1F0CF;",
  100, 100, "charref=true fontname=EmojiOneColor fontsize=50");
```

Ruby on Rails Ruby on Rails は、Ruby による Web 開発を実現するオープンソースフレームワークです。Ruby 用 PDFlib 拡張は Ruby on Rails で利用可能です。Ruby on Rails 用 PDFlib 作成例を実行するには以下の手順に従ってください：

- ▶ Ruby と Ruby on Rails をインストール。
- ▶ コマンドラインから新規コントローラーをセットアップ：

```
$ rails new pdflibdemo
$ cd pdflibdemo
$ cp <PDFlibディレクトリー>/bind/ruby/<バージョン>/PDFlib.so vendor/ # .so/.dll/.bundleを使用
$ rails generate controller home demo
$ rm public/index.html
```

- ▶ `config/routes.rb` を編集：

```
...
# public/index.htmlを削除することを忘れないようにしてください
root :to => "home#demo"
```

- ▶ `app/controllers/home_controller.rb` を以下のように編集して、PDF 内容を作成するための PDFlib コードを挿入。PDF 出力はメモリー内に生成しなければならないことに留意してください。すなわち、空のファイル名を `begin_document()` に与える必要があります。出発点として、`hello-rails.rb` サンプル内のコードを利用できます：

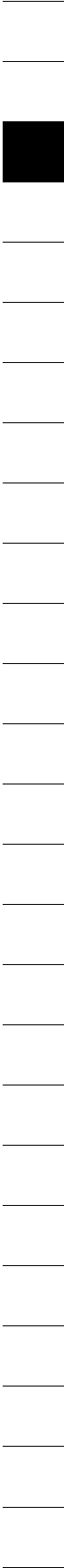
```
class HomeController < ApplicationController
  def demo
    require "PDFlib"
    begin
      p = PDFlib.new
      ...
      ...PDFlibアプリケーションコード、hello-rails.rbを参照...
      ...
      send_data p.get_buffer(), :filename => "hello.pdf",
        :type => "application/pdf", :disposition => "inline"
      rescue PDFlibException => pe
      # エラー処理
    end
  end
end
```

- ▶ インストールを試験するには、以下のコマンドで WEBrick サーバーを起動し、

```
$ rails server
```

そしてブラウザで `http://0.0.0.0:3000` を表示させます。生成された PDF 文書がブラウザ内に表示されます。

ローカル PDFlib インストール PDFlib を Ruby on Rails だけで使いたいにもかかわらず、PDFlib を Ruby で一般利用できるようにグローバルにインストールすることができない場合は、Rails ツリー内の `vendors` ディレクトリーの中に PDFlib をローカルにインストールするという方法があります。この方法は特に、Ruby 拡張を一般利用できるようにインストールする権限を持たない場合に、それでも PDFlib を Rails で扱いたいというときに役立ちます。



3 PDF 文書を作成

3.1 PDFlib プログラミングの一般的特点

クックブック プログラミングの一般的な諸側面に関するコードサンプルが PDFlib クックブックの general カテゴリーにあります。

3.1.1 例外処理

ある種のエラーは、多くの言語で例外と呼ばれています。この呼び方は理にかなっていません。そうしたエラーはまさに例外であって、プログラムの動作中にそんなに頻繁には起こらないだろうと予想される類のものだからです。一般に採られる方針としては、エラーが頻繁に起きそうな呼び出しについては従来型のエラー伝達方式（すなわち、-1 など特殊な戻り値でエラーを示す）を用いるけれども、まれにしか起こらないエラーで、こんな特例にまで場合分けを増やしてコードをややこしくするのもどうかと思われるような場合については例外方式を用いる、というのが普通です。PDFlib のやり方もまさにこれと同じです：たとえば、かなりの頻度でエラーが起きそうな操作としては：

- ▶ 出力ファイルを、権限がないのに開こうとする
- ▶ PDF を読み込みたいとき、間違ったファイル名で開こうとする
- ▶ 画像ファイルが壊れているのに開こうとする

PDFlib では、このようなエラーを特殊な戻り値で示します。それぞれの値は PDFlib API リファレンスに示してあります（たいていは -1。ただし PHP バインディングでは 0）。エラー時に -1 を返すと解説に記されている関数についてはすべて、アプリケーション開発者の側でこのエラーコードをチェックする必要があります。

これに対して、以下に挙げるようなエラーは、もっと深刻かもしれない、しかしさほど頻繁には起こらないエラーです。

- ▶ 仮想メモリ不足
- ▶ スコープ違反（たとえば文書を開く前に閉じようとする）
- ▶ PDFlib API メソッドに対する引数指定の誤り（たとえば円を負の半径で描こうとする）、またはオプション指定の誤り

PDFlib がこのような状況を認識したときには、特殊なエラー戻り値が呼び出し側に渡るのではなく、例外が発生します。知っておくべき重要なこととして、例外が発生すると、それまで生成させていた PDF 文書は完了できなくなります。例外の後で安全に呼べるメソッドは `PDF_delete()`・`PDF_get_apiname()`・`PDF_get_errnum()`・`PDF_get_errmsg()` だけです。それ以外の PDFlib のメソッドを例外の後で呼んだ場合の結果は予測不能です。例外の中には以下の情報が含まれています：

- ▶ 一意なエラー番号。
- ▶ 例外を起こした PDFlib API メソッドの名前。
- ▶ 問題の詳細などを述べたテキスト。

失敗した関数呼び出しの原因をクエリー さきに述べたように、生成中の PDF 出力文書は、例外が起こればかならず放棄しなければなりません。ただし、メソッドが、エラー値を返すことによって、致命的でない問題を報告した場合には、文書を続行することもできます。クライアントアプリケーションは自由に、プログラムの流れを調整したり、別のデータを与えたりすることで、文書を続行することができます。たとえば、あるフォント

の読み込みができないとき、たいていのクライアントは文書を放棄してしまうでしょうが、クライアントによっては、別のフォントを使って何とかしたいという場合もありえます。そのようなときは、問題がより詳しく記されたエラーメッセージを取得したい場合もあるでしょう。そのような場合には、`PDF_get_errnum()`・`PDF_get_errmsg()`・`PDF_get_apiname()` 関数を、問題発生した呼び出しの直後に呼び出すことができます。具体的には、エラー値 -1 (PHP の場合 : 0) を返してきた呼び出しの直後に呼び出します。

エラーポリシー PDFlib はエラー状況を検出すると、いくつかの戦略のうちの 1 つに従って反応します。これは `errorpolicy` オプションで構成できます。エラーコードを返す可能性のあるメソッドはすべて、`errorpolicy` オプションにも対応しています。以下のエラーポリシーに対応しています：

- ▶ `errorpolicy=return` : エラー状況が検出されたとき、メソッドがエラー値 -1 (PHP の場合 : 0) を返します。アプリケーション開発者はこの戻り値をチェックして問題を突きとめ、その問題に対してアプリケーションごとに適切な方法で対処する必要があります。このアプローチではエラー処理に統一的なアプローチができるので、推奨します。
- ▶ `errorpolicy=exception` : エラー状況が検出されたとき、例外を発生させます。出力文書は不完全で使えないものになりますので破棄する必要があります (これは `removefragments` 文書オプションを用いて自動化することも可能です)。この方式を使うと、エラー条件分岐をさぼって手軽にプログラミングができますが、そのかわり問題が起きると出力文書は、たとえそれが致命的でない、アプリケーション側で対処できる可能性のある問題であっても失われてしまいます。

以下のコード断片群は、例外処理に関するさまざまな戦略を演示します。このさまざまな例は、存在するかしないかわからないフォントを読み込もうとしています。

`errorpolicy=return` のときは、戻り値がエラーかどうかをチェックする必要があります。それが失敗を示しているときは、失敗の原因をクエリーすることで、状況を適切にさばくことができます：

```
font = p.load_font("MyFontName", "unicode", "errorpolicy=return");
if (font == -1)
{
    /* フォントハンドルが無効。しかしPDF出力は継続可能。*/
    errmsg = p.get_errmsg();
    /* 別のフォントを試すか諦める */
    ...
}
/* フォントハンドルは有効。継続 */
```

`errorpolicy=exception` のときは、エラーが発生したら文書は放棄しなければなりません：

```
font = p.load_font("MyFontName", "unicode", "errorpolicy=exception");
/* 例外が発生しなければフォントハンドルは有効。
 * 例外が発生したら、PDF出力は継続できない
 */
```

クックブック 完全なコードサンプルがクックブックの `general-programming/error_handling` トピックにあります。

警告 問題の状況によっては、PDFlib がそれを内部的に検出しても、例外を発生させてプログラムの流れをさえぎる正当な理由にはならないものもあります。例外を発生させるの

ではなく、状況の説明がログ記録されます(ログ記録機能についてさらに詳しくは 57 ページ「3.1.2 ログ記録」を参照)。警告に関しては以下のアプローチを推奨します：

- ▶ 開発局面では警告のログ記録を有効にして、そのログファイル内のすべての警告メッセージを注意深く研究します。こうした警告は、コードやデータにひそむ問題を示している可能性がありますから、その原因を理解ないし除去するよう努めなければなりません。
- ▶ 運用局面では警告のログ記録を無効にして、問題が起きたときだけ再び有効にします。

3.1.2 ログ記録

PDFlib は、以下の項目を記録するログファイルを生成できます：

- ▶ すべての API 呼び出し、およびその引数群とオプション群。なお、PDFlib 言語ラッパーによって発されたさらなる API 呼び出しもログ記録に含まれる場合があります(文字列変換のためなど)。
- ▶ API メソッドによって返された戻り値とハンドル。
- ▶ 各呼び出しに対するタイムスタンプ。
- ▶ 廃止済の API メソッド・オプションの使用に関する情報。
- ▶ 例外を発生させるほどではないが、アプリケーションの開発フェーズの間に調べるべき警告メッセージ。
- ▶ サポートケースの調査に有用でありうる内部動作に関する詳細。

ログファイルの内容は、アプリケーション開発者にとって、プログラムフローの中の問題を特定するうえで重要な助けとなりえます。ログ記録は実行時に以下のように有効化できます：

```
p.set_option("logging={filename={mylogfile.log}}");
```

あるいは、ログ記録を環境変数を通じて有効化させることも可能です。ログ記録される情報の量は、さまざまなログ記録クラスと、各クラスに対するレベルを用いて制御できます。詳しくは PDFlib API リファレンスを参照してください。

ログ記録は通常、本番環境においては有効化すべきではなく、開発フェーズ中のみ、かつ問題を解析する必要がある時に有効化されるべきものです。PDFlib GmbH のサポートでは、ユーザーの問題を話し合うためにログファイルを要請することがあります。

以下は、デフォルトのログ記録クラスを用いた典型的なログファイルから数行を抜粋したものです：

```
PDF_load_image(p_0x2201c20, "auto", "nesrin.jpg", /*c*/0, "")  
[0]
```

```
PDF_begin_page_ext(p_0x2201c20, 10.000000, 10.000000, "")  
[Begin page #1]  
PDF_fit_image(p_0x2201c20, 0, 0.000000, 0.000000, "adjustpage")  
PDF_close_image(p_0x2201c20, 0)  
PDF_end_page_ext(p_0x2201c20, "")  
[End page #1]
```

```
PDF_end_document(p_0x2201c20, "")  
[Full product name: "PDFlib Personalization Server"]  
[Document ID: <C98301CB2D4EAC2972D34CAAAE929021> <C98301CB2D4EAC2972D34CAAAE929021>]  
[Size of document: 34842 bytes]  
[End document #1]
```

3.1.3 PDFlib 仮想ファイルシステム (PVF)

クックブック 完全なコードサンプルがクックブックの `general/data_from_memory` トピックにあります。

ディスク上のファイルとは別に、クライアントは、*PDFlib Virtual File System* (PVF) というしくみを利用することもできます。これを用いると、メモリー内のデータを直接供給することができるので、ディスクのファイルを扱う必要がまったくありません。これには速度向上という利点があります。PVF はデータベースから取り出したデータなどのように、ファイルとしてディスク上に存在していない時にも活用できます。またそれ以外にも、クライアントの必要とするデータが、何らかの処理の結果としてメモリー上にすでに存在している場合に対して一般に有用です。

PVF の基本コンセプトは、仮想の読み取り専用ファイルに名前を付けて、それを通常のファイル名とまったく同じように、あらゆる API メソッドで使用できるようにするというものです。UPR の構成ファイルでも使用することができます。仮想ファイル名は、クライアントが任意に名づけることができます。もちろん、仮想ファイル名は、通常のディスクのファイルと名前衝突が起こらないようなものにならなければなりません。そのため、以下のような体系的な命名規則を推奨します (*filename* は、クライアントが名づける部分で、各カテゴリー内で一意な名前です)。また、ファイル名の拡張子についても、標準に従うことを推奨します：

- ▶ ラスター画像ファイル : `/pvf/image/filename`
- ▶ フォントのアウトラインファイル (実際のフォント名をファイル名の先頭部分として用いることを推奨) : `/pvf/font/filename`
- ▶ ICC プロファイル : `/pvf/iccprofile/filename`
- ▶ PDF 文書 : `/pvf/pdf/filename`

ファイル名を指定されて探す時、PDFlib はまず、その指定されたファイル名が既知の仮想ファイルのなかにあるかどうかをチェックします。その後、指定されたファイルをディスク上で開こうとします。

仮想ファイルの継続期間 仮想ファイルでデータが提供されたときには、それをすぐに消費する関数もあるでしょうし、そのファイルの一部分だけをまず読んで、残りの部分は後の時点で使うという関数もあるでしょう。そのため、各仮想ファイルの継続期間については細心の注意を払う必要があります。PDFlib は、それぞれの仮想ファイルに内部ロックをかけておき、その内容がもう必要なくなった時にはじめてロックを外します。そのデータをただちにコピーしておくようクライアントが PDFlib に要求した場合 (`PDF_create_pvf()` で `copy` オプションを指定) を除いては、仮想ファイルの内容をクライアントが変更・削除・解放することが許されるのは、PDFlib がロックを外した後に限られます。PDFlib は、`PDF_delete()` が実行されたときには、自動的にすべての仮想ファイルを削除します。しかし、ファイルの実際の内容 (仮想ファイルの元データ) はつねにクライアントが解放しなければいけません。

さまざまな戦略 PVF では、仮想ファイルのための必要メモリーの管理アプローチは複数ありえます。どのアプローチでも考慮の対処とすべきことは、PDFlib は、仮想ファイル名を用いた API 呼び出しが終わった後でもまだその仮想ファイルの内容へのアクセスを必要とするかもしれないということと、`PDF_end_document()` の後であればもうアクセスを必要とすることは全くないということです。留意する必要があるのは、`PDF_delete_pvf()` を呼んでも実際のファイルの内容が解放されるわけではなく (`copy` オプションを指定した場合は例外)、その PVF ファイル名を管理していた対応するデータ構造が解放されるだけだということです。よって、以下のような戦略がありえます：

- ▶ メモリー使用を最小にする：API 呼び出しで仮想ファイル名を用いたら、その直後に `PDF_delete_pvf()` を呼ぶのがよいでしょう。そして、`PDF_end_document()` の後にまた `PDF_delete_pvf()` を呼ぶことを推奨します。この 2 度目の呼び出しがどうしても必要かという、最初の呼び出しの時点ではまだ PDFlib からデータへアクセスが必要だったかもしれない、そのような場合はその時点では仮想ファイルのロック解除が拒否されてしまっているためです。しかし、最初の呼び出しですでにデータが解放できている場合もあるわけで、そのような場合でも 2 度目の呼び出しは何も害にはなりません。クライアントがファイルの内容を解放できるのは `PDF_delete_pvf()` が成功した時だけなのです。
- ▶ 仮想ファイルを再利用して速度を向上させる：クライアントによっては、同じデータ（たとえばフォント定義）を複数の出力文書に対して使いまわしたいこともあるでしょう。そのような場合には、同じファイル内容に対して何度も作成・削除を繰り返すのは賢明ではありません。その仮想ファイルを使ってほかにもまだ PDF 出力文書を生成したいという場合は、`PDF_delete_pvf()` は呼び出さなくて済むのがよいでしょう。
- ▶ 手抜きプログラミング：メモリー使用量が気にならなければ、クライアントが `PDF_delete_pvf()` を全然呼ばないようにしてもかまいません。この場合 PDFlib は、`PDF_delete()` が呼ばれた時点で、すべての開かれたままの仮想ファイルを内部的に削除します。

どの場合でも、クライアントが対応するデータを解放できるのは、`PDF_delete_pvf()` が成功裏に帰ってきた時か、`PDF_delete()` の後だけです。

PDF 出力を仮想ファイル内に作成 PVF は、ユーザーデータを PDFlib に与えるだけでなく、PDFlib が生成した PDF 文書データを保持することもできます。これは、`PDF_begin_document()` に `createpvf` オプションを与えることによって実現できます。その PVF ファイル名は以後、別の PDFlib API メソッドに与えることができます。これはたとえば、PDF 文書を PDF ポートフォリオへ入れ込むために生成するときには有用です。PDFlib が作成した PVF データを直接取得することはできませんので、メモリーから PDF データを取り出すには、能動または受動インコア PDF 生成を用います（64 ページ「3.1.5 PDF 文書をメモリー内に生成」参照）。

3.1.4 リソース構成とファイル検索

高度な応用アプリケーションでは、PDFlib にさまざまなリソースを利用させる必要があります。たとえばフォントファイル・ICC カラープロファイルなどです。PDFlib のリソース管理を、プラットフォームに依存しない、カスタマイズの容易なものにするためには、構成ファイルを作成しておくことができます。構成ファイルの中には、利用可能なさまざまなリソースとその対応するディスクファイル名を記述しておくのです。このような静的な構成ファイルだけではなくて、`PDF_set_option()` によるリソースの追加を用いた動的な実行時構成を行うことも可能です。構成ファイルに関しては PDFlib は、*Unix PostScript Resource* (UPR) という簡単なテキスト形式を用いています。この、PDFlib で用いる形の UPR ファイル形式については後述します。

`enumeratefonts` オプションを用いると、PDFlib に、検索パス上に存在するすべてのフォントを収集させることができます（141 ページ「フォントリソースリストを自動生成」を参照）。

リソースのカテゴリ PDFlib の対応しているリソースカテゴリを表 3.1 に挙げます。多くのカテゴリは、リソース名（PDFlib API 内で用いるための）を、仮想かディスク

ベースのファイルへマップするものです。表 3.1 にあるもの以外のリソースカテゴリーは無視されます。カテゴリー名は大文字・小文字区別されます。その値は名前文字列として扱われます。すなわち、ASCII か UTF-8（行頭に BOM 付き）で、あるいは IBM Z の場合には EBCDIC-UTF-8 で符号化することができます。Unicode 値は、*HostFont* リソースで各国語のフォント名を指定するのに有用でしょう。

表 3.1 PDFlib で使えるリソースカテゴリー

カテゴリー	形式	説明
<i>SearchPath</i>	pathname	データファイルのあるディレクトリーの相対または絶対パス名
<i>CMap</i>	cmapname=filename	(廃止済) 日中韓エンコーディングのための CMap ファイル
<i>FontAFM</i>	fontname=filename	(廃止済) AFM 形式の PostScript フォントメトリックファイル
<i>FontPFM</i>	fontname=filename	(廃止済) PFM 形式の PostScript フォントメトリックファイル
<i>FontOutline</i>	fontname=filename	PostScript・TrueType・OpenType・WOFF・WOFF2 のうちいずれかのフォントファイル
<i>Encoding</i>	encodingname=filename	8 ビットエンコーディングまたはコードページのテーブルを持ったテキストファイル
<i>HostFont</i>	fontname=hostfontname	システムにインストールされているフォントの名前（通常、両方のフォント名は等しい）
<i>FontnameAlias</i>	aliasname=fontname	PDFlib がすでに知っているフォントに対してエイリアスを作成
<i>ICCProfile</i>	profilename=filename	ICC カラープロファイル名

UPR ファイル形式 UPR ファイルはテキストファイルであり、その構造は非常に単純で、テキストエディターで簡単に書くことができますし、自動生成させることもできます。まず、その文法を見てみましょう：

- ▶ それぞれの行は最大 1023 キャラクターまで。
- ▶ 行末のバックスラッシュキャラクター「\」は、行終端をキャンセルします。これは行を延長したいときに使えます。
- ▶ パーセント「%」キャラクターは、行末までの注釈を開始させます。行データの一部である（すなわち注釈を開始させない）パーセントキャラクターは、直前にバックスラッシュキャラクターを付けて保護する必要があります。
- ▶ 行終端を保護するバックスラッシュと、パーセントキャラクターを保護するバックスラッシュキャラクターとの直前のバックスラッシュキャラクター群は、行データの一部であるなら二重にする必要があります。
- ▶ ピリオド「.」を単独で用いると、セクションの終了を意味します。
- ▶ すべてのエントリーは、大文字・小文字を区別します。
- ▶ スペースは、リソース名中とファイル名中をのぞくあらゆる箇所で無視されます。
- ▶ リソースの名前を値は、等号「=」を一切含んではいけません。
- ▶ 1 個のリソースが複数回定義された場合は、最後の定義がそれ以前の定義を上書きします。

UPR ファイルは以下の部分から成っています：

- ▶ ファイルの種類を示すおまじない行。次の形をとります：

```
PS-Resources-1.0
```

- ▶ ファイル中で記述されるすべてのリソースカテゴリーを一覧にしたセクション。省略可能です。各行に 1 つずつリソースカテゴリーを記述します。この一覧は、ピリオド 1 個だけの行によって終了します。利用可能なリソースカテゴリーについては後述します。この省略可能なセクションが存在しない場合であっても、1 個のピリオドキャラクターは存在する必要があります。
- ▶ ファイルのはじめに挙げられたリソースカテゴリーそれぞれについてセクションが 1 つずつ。各セクションは、リソースカテゴリーを示す 1 行で始まり、その後、利用可能なリソースを記述する行が任意の行数つづきます。この一覧は、ピリオド 1 個だけの行によって終了します。各リソースデータ行にはリソースの名前を書きます（等号はクォートする必要があります）。そのリソースがファイル名を必要とする場合には、等号の後にこの名前を付け加える必要があります。リソースエントリーに列挙されたファイルを PDFlib がさがす時には *SearchPath*（以下を参照）が適用されます。

ファイル検索と SearchPath リソースカテゴリー PDFlib はさまざまなデータアイテムをディスク上のファイルから読み込みます。たとえばラスター画像・フォントアウトライン・フォントメトリック情報・PDF 文書・ICC カラープロファイルなどです。相対パス名でも絶対パス名でもない、パス指定をまったくつけないファイル名を用いることもできます。*SearchPath* リソースカテゴリーを使うと、必要なデータファイルのあるディレクトリーのパス名の一覧を指定することができます。何かファイルを開かなければならないとき、PDFlib は、まずそのままのファイル名でファイルを開こうとします。この試みが失敗すると、PDFlib は、*SearchPath* リソースカテゴリーで指定されたディレクトリー群の中でそのファイルが開けないかどうか、成功するまで一つ一つ試みます。*SearchPath* 項目を蓄積させることもでき、それらは逆順に検索されます（後の時点で設定されたパスほど、もっと早くに設定されたものよりも先に検索される）。この機能を使うと、PDFlib のアプリケーションを、プラットフォーム依存なファイルシステム体系から解放することができます。検索パスエントリーを設定するには以下のようにします：

```
p.set_option("SearchPath={{/パス/パス/ディレクトリー 1} {/パス/パス/ディレクトリー 2}}");
```

検索パスを複数設定することもでき、また、複数のディレクトリー名を 1 回の呼び出しの中で与えることもできます。空白キャラクターを含んだディレクトリー名による問題を避けるために、エントリーが 1 個だけの場合にも中カッコを二重に用いることを推奨します。空の文字列リスト（例：{{}}）は、既存のすべての検索パスを、デフォルト項目も含めて削除します。

この検索を無効にするには、フルパスによる指定を PDFlib 関数の中に入ります。なお、*SearchPath* リソースカテゴリーの機能は以下のようにプラットフォーム依存になっています：

- ▶ Windows の場合、PDFlib はレジストリーからの項目群で *SearchPath* を初期化します。以下のレジストリーエントリーにパス名のリストをセミコロン「;」で区切って指定することが可能です。これらは以下の順序で検索されます：

```
HKLM\SOFTWARE\PDFlib\PDFlib10\10.0.0\SearchPath
HKLM\SOFTWARE\PDFlib\PDFlib10\SearchPath
HKLM\SOFTWARE\PDFlib\SearchPath
```

デフォルトファイル検索パス Unix・Linux・macOS・IBM System i システムでは、パス・ディレクトリー名を一切指定しなくても、いくつかのディレクトリーがデフォルトでファイル検索されます。UPR ファイル（追加の検索パスを内容として持つ可能性のある）を検索して読み取る前に、以下のディレクトリーが検索されます：

```
<rootpath>/PDFlib/PDFlib/10.0/resource/cmap
<rootpath>/PDFlib/PDFlib/10.0/resource/codepage
<rootpath>/PDFlib/PDFlib/10.0/resource/font
<rootpath>/PDFlib/PDFlib/10.0/resource/icc
<rootpath>/PDFlib/PDFlib/10.0
<rootpath>/PDFlib/PDFlib
<rootpath>/PDFlib
```

Unix・Linux・macOS では、**<rootpath>** はまず HOME ディレクトリーへ置き換えられ、ついで **/usr/local** へ置き換えられます。IBM System i では、**<rootpath>** は空です。

上記の最後のエントリーはとりわけ複数の製品ののためのライセンスファイルを格納するのに有用です。

デフォルトの検索パスが必要ない場合には、空の **searchpath** を与えることによってそれを無効化することによって、ファイル操作を加速できます。

デフォルトファイル名とライセンスファイル・リソースファイル デフォルトでは、以下のファイル名が、デフォルト検索パスディレクトリー群の中で検索されます：

licensekeys.txt	(ライセンスファイル。MVSの場合: license)
pdflib.upr	(リソースファイル。MVSの場合: upr)

この機能を利用すると、ライセンスファイルを、環境変数や実行時オプションを一切設定しないで扱うことができます。

サンプル UPR ファイル UPR 構成ファイルの作成例を以下に挙げます：

```
PS-Resources-1.0
.
SearchPath
/usr/local/lib/fonts
users/kurt/my_images
.
FontOutline
ArialMT=Arial.ttf
.
HostFont
Wingdings=Wingdings
.
ICCPProfile
highspeedprinter=cmykhigspeed.icc
.
```

UPR リソースファイルを検索 組み込みリソース（たとえば PDF コアフォント・sRGB ICC プロファイル）やシステムリソース（たとえばホストフォント）だけが使われる場合には、UPR 構成ファイルは必要とはされません。なぜなら、とりたてて構成がなくても、必要なリソースをすべて PDFlib が見つけられるからです。

それ以外のリソースを使いたい場合は、そのリソースを指定するために、**PDF_set_option()**（後述）を呼び出すか、UPR リソースファイルに記述します。このファイルを、PDFlib は、最初のリソースが要求された時に自動的に読み込みます。その過程は詳しくは以下のとおりです：

- ▶ Unix システム・macOS・IBM System i では、パス・ディレクトリー名を一切指定しなくても、デフォルトでいくつかのディレクトリーでライセンス・リソースファイルを検

索されます。UPR ファイルを検索して読み取る前に、以下のディレクトリーが検索されます（この順に）：

```
<rootpath>/PDFlib/PDFlib/10.0/resource/icc  
<rootpath>/PDFlib/PDFlib/10.0/resource/fonts  
<rootpath>/PDFlib/PDFlib/10.0/resource/cmap  
<rootpath>/PDFlib/PDFlib/10.0  
<rootpath>/PDFlib/PDFlib  
<rootpath>/PDFlib
```

Unix システム・macOS では、**<rootpath>** は、まず `/usr/local` で、ついで HOME ディレクトリーで置き換えられます。IBM System i では **<rootpath>** は空です。この機能を利用すれば、環境変数や実行時オプションを一切指定せずにライセンスファイル・UPR ファイル・リソースを取り扱うこともできます。

- ▶ 環境変数 `PDFLIBRESOURCEFILE` が定義されていれば、その値を PDFlib は、読み込むべき UPR ファイルの名前とします。このファイルが読み込めないときは例外が発生します。
- ▶ 環境変数 `PDFLIBRESOURCEFILE` が定義されていない場合、PDFlib は以下の名前のファイルを開こうとします：

```
upr (MVSの場合。データセットが期待される)  
pdflib/<バージョン>/fonts/pdflib.upr (IBM System iの場合)  
pdflib.upr (Windows・Unix・その他すべてのシステムの場合)
```

このファイルが読み込めないときは例外は発生しません。

- ▶ Windows では上記以外に PDFlib は以下のレジストリーエントリーをこの順に読み込もうとします：

```
HKLM\Software\PDFlib\PDFlib10\10.0.0\resourcefile  
HKLM\Software\PDFlib\PDFlib10\resourcefile  
HKLM\Software\PDFlib\resourcefile
```

そしてこれらのエントリーの値を、用いるべきリソースファイルの名前とします。このファイルが読み込めないときは例外が発生します。64 ビット Windows システム上でレジストリーを手作業で扱う際には注意が必要です：通常どおり、64 ビットの PDFlib バイナリーは Windows レジストリーの 64 ビットビューとともに動作し、64 ビットシステム上で動作する 32 ビットの PDFlib バイナリーはレジストリーの 32 ビットビューとともに動作します。32 ビット製品に対するレジストリーキーを手作業で追加する必要があるときは、必ず 32 ビット版の *regedit* ツールを使用してください。これは「スタート」ダイアログから以下によって起動することができます：

```
%systemroot%\syswow64\regedit
```

- ▶ クライアント側で `resourcefile` オプションを明示的に設定することでリソースファイルを PDFlib に実行時に読み込ませることもできます。以下のように記述します：

```
p.set_option("resourcefile={{パス/パス/pdflib.upr}}");
```

この呼び出しは任意の回数繰り返すことができます。その場合、リソースエントリーが蓄積されていきます。

リソースを実行時に構成 UPR ファイルを使った構成だけではなく、ソースコード中で `PDF_set_option()` を使って直接個々のリソースを構成することもできます。このメソッドはカテゴリー名とそれに対応するリソースエントリーをとります。リソース記述の部分

は、UPR リソースファイルでこのカテゴリーのセクションに書くのと同じように書きま
す。たとえば：

```
p.set_option("FontOutline={Foobar-Bold=foobb.otf}");
```

注 フォント構成の詳しい説明は 141 ページ「6.3.4 フォントを検索」を参照してください。

リソース値をクエリー リソースエントリーを設定するだけでなく、*PDF_get_option()* を
使ってクエリーすることもできます。カテゴリー名をキーとして、リソースの番号（1 から
開始）をオプションとして指定します。たとえば以下の呼び出し：

```
idx = p.get_option("SearchPath", "resourcenumber=" + n);  
sp = p.get_string(idx, "");
```

は、SearchPath リスト内の *n* 番目のエントリーを取得します。要求されたカテゴリーに対
して利用可能なエントリーの数よりも *n* が大きいときは、空文字列が返されます。返され
た文字列は、何らかの API メソッドが次に呼び出されるまで有効です。

3.1.5 PDF 文書をメモリー内に生成

ファイル上に PDF 文書を生成するだけでなく、PDFlib を使ってメモリー内に直接 PDF
を生成させることもできます（インコア生成といいます）。この技法は、何らディスクペ
ースの入出力が伴わないために速度の面で利点がありますし、PDF 文書をたとえば直接
HTTP で流したりすることもできます。Web 管理者が聞いて特に喜びそうなのは、自分の
サーバーにテンポラリ PDF ファイルを散らかされずに済むということでしょう。

生成されるデータは、定期的に少しずつ集めることもできますし（たとえば各ページ
ができるごとに）、最後にまるごと PDF 文書になってからひとかたまりで取り出すこと
もできます（*PDF_end_document()* の後で）。PDF データの細切れでの生成および消費には
いくつかの利点があります。第一に、データ全体がメモリー内に収まる必要がないので、
メモリー必要量が小さくて済みます。第二に、この方式では速度も上がる可能性があります。
なぜなら、遅い接続でデータを伝送している場合でも、最初の 1 チャンクを送り出し
ている間に、次の 1 チャンクがもう生成中だからです。ただし、生成されるデータの総量
は文書全体ができあがるまでわかりません。

createpvf オプションを用いると、PDF データをディスクファイルへ書き込むことなく、
メモリー内へ生成し、その後それを PDFlib に渡すことができます（59 ページ「PDF 出力
を仮想ファイル内に作成」参照）。

能動インコア PDF 生成インターフェイス PDF データをメモリー内に作成するには、
PDF_begin_document() で空のファイル名を指定し、*PDF_get_buffer()* でデータを取得しま
す：

```
p.begin_document("", "");  
...文書を作成...  
p.end_document("");  
  
buf = p.get_buffer();  
... バッファ内のPDFデータを利用 ...  
p.delete();
```

注 バッファ内の PDF データはバイナリーデータとして扱う必要があります。

これは「能動」モードと捉えられます。なぜなら、バッファ内容をいつ取り出したいかをクライアント側で決めているからです。能動モードはすべての対応言語バインディングで利用可能です。

注 C と C++ のクライアントでは、返ってきたバッファを解放してはいけません。

受動インコア PDF 生成インターフェイス 「受動」モードは、C と C++ の言語バインディングでのみ利用可能です。この場合、ユーザーは、1 つのコールバック関数をインストールします (`PDF_open_document_callback()` を通じて)。この関数は、PDFlib によって、予測不可能なさまざまな時点で呼ばれます。PDF データが消費されるのを待っている時にはいつでも呼ばれることになります。放出 (ライブラリーからクライアントへの PDF データの転送) に関するタイミング上の、ひいてはバッファ容量上の束縛条件は、クライアント側で構成することができるので、柔軟性が非常に高くなっています。環境によって、PDF 文書をまるごと一度に取り出すのが好都合な場合もあるでしょうし、複数のチャンクに分けるのがよい場合もあるでしょうし、たくさんのコマ切れに分けて PDFlib の内部文書バッファ量を抑えるのが望ましい場合もあるでしょう。放出の手法を設定するには、`PDF_open_document_callback()` で `flush` オプションの値を指定します。

3.1.6 PDF 文書の最大サイズとその他の制限

PDF 文書のサイズ 多くのユーザーはギガバイト単位の PDF 文書を扱う必要には迫られないでしょうが、業務アプリケーションのなかには、大量の請求書や明細などを含む文書を作成したり処理したりする必要があるものがあります。PDFlib 自体は生成する文書のサイズにいかなる制約も設けていませんが、PDF Reference やいくつかの PDF 規格によって課せられるいくつかの制限があります：

- ▶ 10 GB ファイルサイズ制限：PDF 1.4 の文書は、相互参照テーブルによって、10 進 10 桁すなわち $10^{10}-1$ バイトまでに制限されてきました。これはおよそ 9.3 GB にあたります。10 GB を超える出力文書を作成しようとするなら、PDF 1.5 以上を使用する必要があります。このバージョンは、10 桁制限にもはや縛られない圧縮オブジェクトストリームをサポートしていますので、10 GB を超える PDF 文書の作成が可能です。
- ▶ オブジェクトの数：一文書内のオブジェクトの数は全般的には PDF によって制限されていませんが、PDF/A・PDF/X-4・PDF/X-5 規格では、一文書内の間接オブジェクトの数を 8,388,607 個に制限しています。一文書がこの制限を超えるオブジェクトを必要とするときは、PDFlib は PDF/A-1・PDF/X-4・PDF/X-5 モードでは例外を発生させます。それ以外のモードではつねに、もっとオブジェクトの多い文書を作成できます。このチェックを、文書オプション `limitcheck=false` を用いて無効化することもできます。PDF 内のオブジェクトの数は、ページ内容の複雑さや、相互参照要素の数などに依存します。シンプルな内容の大容量文書は通常ページあたり 4～10 個のオブジェクトを持ちますので、100～200 万ページ程度の文書であれば、規格が要求するこのオブジェクト制限を超えずに作成することができます。

PDF の制限 PDFlib は、特定の実体に制約を課すことによって、PDF Reference か Acrobat、または何らかの PDF 規格によって課せられる制限に準拠する PDF 出力を作成します。これらの制限を以下に記します。

以下の制限が、値を然るべく変更することによって強制されます：

- ▶ PDF 内における最小の絶対浮動小数点値：0.000015。これより小さな絶対値を持つ数は 0 へ置換されます。

- ▶ (PDF 1.4。ただしそれより新しいPDFバージョンにはあてはまりません)PDF 内において浮動小数点数として表現できる最大の絶対値：32767.0。これより大きな絶対値を持つ数は、その最も近い整数へ置換されます。

PDF 形式は特定の諸制限を課しています。以下のいずれか 1 つの制限を超えると例外が発生します：

- ▶ PDF 内において許容される最大の数値：2,147,483,647
- ▶ ハイパーテキスト文字列の最大長：65535
- ▶ ページ上のテキスト文字列の最大長： *Kerning=false* かつ *wordspacing=0* の場合には 32,763 バイト (すなわち、CID フォントの場合には 16,381 キャラクター)。そうでないなら 4095 キャラクター
- ▶ 以下のオプションのリストエントリは最大 8191 個に制限されています：
 views, namelist, polylinelist, fieldnamelist, itemnamelist, itemtextlist, children, group
- ▶ PDF/A-1/2/3 と PDF/X-4/5 内における最大間接オブジェクト数：8,388,607

3.1.7 Web 最適化 (線形) PDF

PDFlib は、線形化という処理を PDF 文書に対して適用することができます (PDF の線形化は「最適化」・「Web 表示用に最適化」ともいいます)。線形化は、PDF ファイル内部のオブジェクト群を再配列するとともに、アクセス高速化のために活用される補足情報を追加します。

非線形化 PDF は、クライアントへまるごと転送される必要がありますが、Web サーバーは線形化 PDF 文書を、バイトサービングという処理を用いて、1 ページずつ転送することが可能です。これによって、Web ブラウザーは、PDF 文書の個別のページを別々に取得することが可能になります。その結果として、その文書の最初のページが、文書全体がサーバーからダウンロードされ終わるのを待つことなく、ユーザーに提示されます。

PDF をバイトサービングする利点を活かすには、PDF 文書を線形化しておく必要があります。これは `PDF_begin_document()` の `linearize` オプションで実現できます：

```
p.begin_document(outfilename, "linearize");
```

3.1.8 マルチスレッドプログラミング

PDFlib そのものはシングルスレッドですが、マルチスレッドアプリケーションで安全に使用することができます。1 つの PDFlib オブジェクトが 1 つのスレッド内でのみ使用されるというよくある状況においては、マルチスレッディングに関する特別な注意は一切必要ありません。同一のオブジェクトが複数のスレッドで使用される場合には、その PDFlib オブジェクトが複数のスレッドによって同時にアクセスされることのないよう、アプリケーションはそれらのスレッドを同期させる必要があります。典型的なシナリオは PDFlib オブジェクト群のプールを利用するもので、そこでは各スレッドが、新規 PDFlib オブジェクトを作成するのではなく、既存の PDFlib オブジェクトを 1 つ取り出して、そして文書を作成した後に、そのオブジェクトがもう必要ない場合には、それをプールへ返します。同一の PDFlib オブジェクトを、その出力文書が完了する前に別のスレッドで使用することは、いかなる利点をも与えることはめったになく、推奨されません。

3.1.9 EBCDIC ベースのプラットフォームで PDFlib を使う

PDF ファイル形式の中のエディターと制御構造は ASCII ベースであり、EBCDIC ベースのプラットフォーム IBM System i・IBM Z では正しく動作しません (ただし zLinux は ASCII ベースですのでこの限りではありません)。しかし、特別なメインフレームバージョンの

PDFlib を利用すれば、ASCII ベースの PDF オペレーターと EBCDIC の（またはそれ以外の）テキスト出力とを混ぜることができます。こうした EBCDIC セーフなバージョンの PDFlib は、さまざまなオペレーティングシステムやマシンアーキテクチャーで利用可能です。

PDFlib のさまざまな機能を EBCDIC ベースのプラットフォームで活用するためには、以下のアイテムは EBCDIC テキスト形式で与えられることが期待されます（より具体的には、IBM System i ではコードページ 037 で、IBM Z ではコードページ 1047 で）：

- ▶ UPR 構成ファイル
- ▶ エンコーディングファイル・コードページファイル
- ▶ PDFlib 関数の文字列引数
- ▶ 入出力ファイル名
- ▶ 環境変数（実行環境が対応している場合）
- ▶ PDFlib のエラーメッセージも EBCDIC 形式で生成されます（Java 以外）。

ASCII 形式の入力テキストファイルを使いたい場合は、*asciifile* オプションを *true* に設定します（デフォルトは、IBM Z では *false*、IBM System i では *true*）。すると PDFlib は、これらのファイルが ASCII エンコーディングで書かれていると期待するようになります。ただしその場合でも、文字列引数はやはり EBCDIC エンコーディングであると期待されます。

これに対し、以下のアイテムはつねにバイナリーモードで取り扱う必要があります（すなわち、いかなる変換も行ってはいけません）：

- ▶ PDF 入出力ファイル
- ▶ PFB フォントアウトラインファイル・PFM フォントメトリックファイル
- ▶ TrueType・OpenType フォントファイル
- ▶ 画像ファイル・ICC プロファイル

3.2 ページ記述

3.2.1 座標系

PDF のデフォルト座標系が PDFlib の内部では用いられています。デフォルト座標系（デフォルトユーザースペースともいう）では、ページの左下隅に原点があり、DTP ポイントを単位として用いています：

1 pt = 1/72 inch = 25.4/72 mm = 0.3528 mm

1 番目の座標は右へ向かって増加し、2 番目の座標は上へ向かって増加します。PDFlib のクライアントプログラムでは、このデフォルトユーザースペースを回転・拡大縮小・並行移動・斜形化させることによって、新しいユーザー座標を作ることもできます。こうした変形に対応する関数はそれぞれ `PDF_rotate()`・`PDF_scale()`・`PDF_translate()`・`PDF_skew()` です。座標系を変更した場合、グラフィック・テキスト関数の中の座標はすべて新しい座標系に従って指定しなければなりません。座標系は、各ページの最初でデフォルト座標系に再設定されます。

メートル座標を用いる メートル座標を、座標系を拡大縮小することによって簡単に使えます。縮尺は、上記の DTP ポイントの定義より導かれます：

```
p.scale(28.3465, 28.3465);
```

この呼び出しの後には、PDFlib はすべての座標をセンチメートル単位として解釈します（インタラクティブ機能については例外、後述）。これは $72 \div 2.54 = 28.3465$ だからです。

関連する機能として、`PDF_begin/end_page_ext()` で `userunit` オプションを指定して（PDF 1.6）ページ全体に対する縮尺を与えることもできます。ただしユーザー座標は、Acrobat での最終的なページ表示に対してのみ効力を持つものであり、PDFlib で座標の拡大縮小を行うものではありません。

クックブック 完全なコードサンプルがクックブックの `general/metric_topdown_coordinates` トピックにあります。

インタラクティブ要素の座標 インタラクティブ関数には、作成したいテキスト注釈・リンク・ファイル注釈の長方形の座標を与える必要があります。PDF では、ハイパーテキストの関数のための座標はつねにデフォルト座標系で記述されていると見なされます。ユーザー座標系（変形されているかもしれない）で記述されていると見なされることはありません。これは非常にやっかいですので、PDFlib には、ユーザー座標が指定されてもそれを PDF が認める形式に自動変換する機能があります。この自動変換を有効にするには、`usercoordinates` オプションを `true` に設定します：

```
p.set_option("usercoordinates=true");
```

リンク・フィールドの長方形としては、その縁がページの縁に平行なものにしか PDF では対応していないので、拡大縮小・回転・並行移動・斜形化によって座標系が変形しているときには与えられた長方形は形を調整しなければなりません。このような場合、PDFlib では、その長方形を囲う、かつ縁がページの縁と平行な最小の長方形を計算します。そしてこれをデフォルト座標に変換し、その結果の値を、与えられた座標のかわりに用います。

要するに大局的に言ってどんな効果があるかといえば、`usercoordinates` オプションが `true` に設定されていれば、ページ内容に対してもインタラクティブ要素に対しても同じ座標系が使えるということです。

座標を視覚化 PDFlib のユーザーが PDF の座標系を扱うのを支援するために、PDFlib のディストリビューションには *grid.pdf* という PDF ファイルが含まれています。この PDF ファイルは、よく使われるいくつかのページ寸法の座標を視覚化するものです。望みの寸法のページを何か透明なものに印刷すれば、PDFlib での開発のために有用な道具になるかもしれません。

Acrobat DC でページ座標を視覚化するには以下のとおり操作します：

- ▶ カーソル座標を表示するには以下を用います：
「表示」→「表示切り替え」→「カーソル座標」
- ▶ 座標は、Acrobat で現在選択されている単位で表示されます。表示単位を変更するには、次のとおり操作します：「編集」→「環境設定」（→「一般...」）→「単位とガイド」を選択して、ポイント・インチ・ミリ・パイカ・センチメートルのうちのいずれかを選びます。

ただし、表示される座標系はページの左上隅が原点であり、PDF のデフォルトである左下隅の原点とは異なるので注意が必要です。Acrobat の座標表示と合わせた座標系を選ぶ方法については 69 ページ「下向き座標を用いる」を参照してください。

オブジェクトを回転 重要なことは、一度ページ上に描いたものは変更ができないということです。PDFlib には、回転・並行移動・拡大縮小・斜形化の関数がありますが、こうした関数は、すでに存在しているものに対しては効力を持たず、それ以降に描かれるものに対してだけ効力があります。

テキスト・画像・取り込み PDF ページを回転させるのは簡単で、*PDF_fit_textline()*・*PDF_fit_textflow()*・*PDF_fit_image()*・*PDF_fit_pdi_page()* で *rotate* オプションを指定します。こうしたオブジェクトをそれぞれのはめ込み枠内で 90 度の倍数だけ回転させるのは、これらの関数の *orientate* オプションで可能です。以下の例は傾き 45 度のテキストを生成します：

```
p.fit_textline("回転テキスト", 50.0, 700.0, "rotate=45");
```

クックブック 完全なコードサンプルがクックブックの *textflow/rotated_text* トピックにあります。

ベクトルグラフィックの回転は、一般の変形関数 *PDF_translate()*・*PDF_rotate()* を利用すれば可能です。以下の例は、左下隅を (200, 100) に持つ、回転された長方形を作成します。描きたい長方形の隅へ座標原点を変更し、座標系を回転させて、長方形を (0, 0) に配置しています。save と restore ではさむことにより、縦置きテキストの作成を完了した後、簡単に元の座標系に戻ってオブジェクトの配置を継続できます：

```
p.save();
    p.translate(200, 100);          /* 原点を長方形の隅へ移動*/
    p.rotate(45.0);              /* 座標を回転させる */
    p.rect(0.0, 0.0, 75.0, 25.0); /* 回転された長方形を描く */
    p.stroke();
p.restore();
```

下向き座標を用いる PDF の上向き座標系とは違って、グラフィック環境のなかには下向き座標を用いているものもあるので、そちらを採用したい開発者もいるでしょう。そのような座標系は PDFlib の変換関数で簡単に設定することができます。ところが、この変換はテキスト出力に対しても効力を持つので（テキストが簡単に上下ひっくり返ります）、テキストが裏返しになってしまうようにするには、ほかにも何らかの呼び出しを行うことが必要になります。

下向き座標が簡単に利用できるようにするため、PDFlib では、ある特殊なモードに対応しています。このモードでは、すべての関連する座標に対してそれぞれ異なる解釈が適用されます。この **topdown** 機能は、PDFlib ユーザーが下向き座標系でごく自然に作業が行えるようにするために設けられています。具体的には、ページの左下隅に原点 $(0, 0)$ があって y 座標が上向きに増加するデフォルト PDF 座標系を扱うのではなく、ページの左上隅に原点があって y 座標が下向きに増加する修正座標系を用います。ページでこの下向き座標系を利用するには、**PDF_begin_page_ext()** で **topdown** オプションを指定します：

```
p.begin_page_ext(595.0, 842.0, "topdown");
```

説明の完全を期するため、下向き座標系を設定した場合の効果を以下に詳しく挙げます。

以下のような「絶対座標」は、通常と何も変わらないやり方でユーザー座標に翻訳されます：

- ▶ メソッドの引数のうち、各メソッドの説明の中で「座標」と書かれているものすべて。
例：**PDF_moveto()** の $x \cdot y$ 。**PDF_circle()** の $x \cdot y$ 。**PDF_rect()** の $x \cdot y$ （しかし **width**・**height** は含まず！）。**PDF_add_note()** の $llx \cdot lly \cdot urx \cdot ury$ 。

「相対座標」の値は、下向き座標に合うよう内部変換されます：

- ▶ テキスト（正の文字サイズを持つもの）は、ページ上端に向かって配置されます。
- ▶ マニュアルの中で、長方形や枠などについて「左下隅」と言っている場合は、ページ上でもそうなるように翻訳されます。
- ▶ 回転角が指定されている場合は、その回転の中心は依然としてユーザー座標系の原点 $(0, 0)$ です。右回り回転はやはり右回りとして表示されます。

クックブック 完全なコードサンプルがクックブックの `general/metric_topdown_coordinates` トピックにあります。

3.2.2 ページ寸法

クックブック 完全なコードサンプルがクックブックの `pagination/page_sizes` トピックにあります。

規格ページ寸法 **PDF_begin/end_page_ext()** の **width**・**height** オプションには、絶対値か、またはシンボリックなページ寸法名を指定できます。後者は、`<規格>.width`・`<規格>.height` の形をとります。ここで `<規格>` は標準判型のいずれかです（小文字で。例：`a4.width`）。

ページ寸法の限界 PDF や PDFlib では、利用できるページ寸法についてはいかなる制約も課していませんが、Acrobat の実装のほうで、ページ寸法に関するプログラムの限界が存在してしまっています。他の PDF インタプリターではもっと大きな寸法やもっと小さな寸法の文書を扱うこともできる可能性があります。Acrobat のページ寸法制限を表 3.2 に示します。PDF 1.6 以上では、**PDF_begin/end_page_ext()** で **userunit** オプションを用いて、ページに対するグローバルな縮尺を指定することもできます。

さまざまなページ寸法枠 PDFlib の開発者の多くは、ページの幅と高さを指定するだけで済みますが、なかには高度なアプリケーションで（とりわけプリプレス業務では）、それ以外の PDF の枠エントリーを記述したいときもあるでしょう。PDFlib では、PDF のすべての枠エントリーに対応しています。PDFlib のクライアントで指定できるエントリーを以下に挙げます（それぞれの定義は PDF リファレンスより）。こうした項目はある種の環境で有用です：

表 3.2 Acrobat の最小・最大ページ寸法

PDF 表示ソフト	最小ページ寸法	最大ページ寸法
userunit オプションなし (デフォルト)	1/24" = 3 pt = 0.106 cm	200" = 14400 pt = 508 cm
userunit オプションあり	3 ユーザー単位	14 400 ユーザー単位 userunit の最大値は 75 000 なので、可能なページ寸法は最大 14 400 × 75 000 = 1 080 000 000 ポイント = 381 km

- ▶ MediaBox: ページの幅と高さを指定するために用いられ、通常私達がページ寸法としてとらえているものを記述します。
- ▶ CropBox: ページの内容が切り抜かれる領域。Acrobat はこの寸法を画面表示と印刷の際に利用します。
- ▶ TrimBox: 完成ページの領域を指定 (裁ち切り後の)。
- ▶ ArtBox: ページ上で意味のある内容が占める領域。これがアプリケーションソフトウェアで利用されることは稀です。
- ▶ BleedBox: 印刷所環境で出力されるときにページの内容が切り抜かれる領域。印刷所工程での裁ち切りの不正確さを考えに入れて少しゆとりを持たせて囲んでもよい。

PDFlib では、上記のどの値も利用することはなく、ただ出力ファイルに記録する機能を持つだけです。デフォルトでは PDFlib は、ページの幅と高さの指定から MediaBox を生成しますが、それ以外のエンタリーを生成しません。以下のコード断片は、新しいページを開始させた後、CropBox の 4 つの値を設定します：

```
/* カスタムCropBoxを持つ新規ページを開始 */
p.begin_page_ext(595, 842, "cropbox={10 10 500 800}");
```

3.2.3 直接パス

パスとは、任意の数の直線・長方形・円・ベジエ曲線・楕円弧でできた輪郭です。パスは、つながっていない部分を複数含むことができます。こうした部分をサブパスといいます。パスに対して適用できる操作はいくつかあります：

- ▶ 描線。パスに沿って線を描きます。クライアントが与えた、描画に関するオプション (たとえば色や線幅) を用います。
- ▶ 塗り。パスで囲われた領域全体を塗ります。クライアントが与えた、塗りに関するオプションを用います。
- ▶ 切り抜き。以後の描画の可視領域を限定します。具体的には、カレント切り抜き領域 (デフォルトでは無限定) が、カレント切り抜き領域とパスで囲われた領域との交差部分にとって替わられます。
- ▶ ただパスを終了。見えないパスができます。それでも PDF ファイルの中には存在しています。これが有用な場合は稀です。

直接パス 関数 `PDF_moveto()`・`PDF_lineto()`・`PDF_rect()` などを用いて、カレントページやその他の内容ストリーム (テンプレート・Type 3 グリフ記述など) へ書かれる直接パスを構築することもできます。パスを構築した直後に、それを `PDF_stroke()`・`PDF_fill()`・`PDF_clip()` のいずれか 1 つおよび関連する関数で処理する必要があります。これらの関数はパスを消費し削除します。パスを複数回使う唯一の方法は、`PDF_save()` と `PDF_restore()` を用いることです。

直接パスを作成しておいて上記の操作を何も適用しないとエラーになります。PDFlib のスコープ体系に従えば、クライアントはこの制約に自然に従うことになります。パスの書式属性(色・線幅など)を変えたい場合はかならず、描画操作の開始前に行う必要があります。この規則を一言で言えば「パス記述の途中で、書式を変えてはいけません」。

パスをただ作成しただけではページには何も現れません。塗りか描線をパスに適用しなければ目に見える結果は得られません：

```
p.set_graphics_option("strokecolor=red");
p.moveto(100, 100);
p.lineto(200, 100);
p.stroke();
```

たいていのグラフィック関数では、カレント点という概念を利用しています。これは、描画に用いているペンの位置と捉えることができます。

クックブック 完全なコードサンプルがクックブックの `graphics/starter_graphics` トピックにあります。

3.2.4 パスオブジェクト

パスオブジェクトは、直接パスよりも便利で強力なものです。パスを構築するためのすべての描画操作をカプセル化します。直接パスと異なり、パスオブジェクトは、`PDF_delete_path()` を用いて明示的に破棄するまで複数回使用できます。パスに関する情報については `PDF_info_path()` を用いて取得できます。パスオブジェクトを作成する方法は複数あります：

パスオブジェクトを直接作成 `PDF_add_path_point()` を呼び出すと、点群および紐付いたパス要素群がパスオブジェクトに追加されます。このメソッドは、新規に構築されたパスに対して、既存のパスオブジェクトへの参照を追加することもできます。`PDF_add_path_point()` では、パス構築を実現するためのいくつかの便利なオプションを使えます。以下のコード断片は、円 1 個を持つシンプルなパスオブジェクトを作成し、それをページ上の別々の 2ヶ所に描線し、最後にそれを削除します：

```
path = p.add_path_point( -1,  0, 100, "move", "");
path = p.add_path_point(path, 200, 100, "control", "");
path = p.add_path_point(path,  0, 100, "circular", "");

p.draw_path(path,  0,  0, "stroke");
p.draw_path(path, 400, 500, "stroke");
p.delete_path(path);
```

クックブック 完全なコードサンプルがクックブックの `graphics/starter_graphics` トピックにあります。

パスオブジェクトを画像のクリッピングパスから作成 TIFF または JPEG ラスター画像内に含まれているクリッピングパスを、`PDF_info_image()` とキーワード `clippingpath` を用いて取得できます：

```
image = p.load_image("auto", "image.tif", "clippingpathname={path 1} infomode");

path = (int) p.info_image(image, "clippingpath", "");
if (path == -1)
    throw new Exception("エラー：クリッピングパスが見つかりません!");

p.draw_path(path, 0, 0, "stroke");
```


これは、任意のパス群を取り込むための簡便な方式を提供します。画像本体のピクセルデータが必要ない場合には、*infomode* オプションを用いて画像を読み込むことによって出力ファイルサイズを減らすことも可能です。

クックブック 完全なコードサンプルがクックブックの `graphics/import_path_from_image` トピックにあります。

パスオブジェクトを SVG パス文法から作成 SVG パス文法を用いてパスオブジェクトを構築することも可能です。この文法は、*path* エレメントの *d* 属性に対して必要なものと同じです (www.w3.org/TR/SVG11/paths.html#PathData を参照)。SVG グラフィックファイル全体が必要なわけではなく、*PDF_add_path_point()* の *svgpath* オプションに与えるパス記述を内容とする文字列があれば足够了。パスが SVG ファイルから直接由来している場合には下向きの座標を使用しているため、PDFlib で使うために合わせる必要があります (*topdown* ページオプションを指定している場合であっても)。第一に、*scale={1 -1}* を与えることによってパスをアンミラーします。第二に、以下のコード断片のように、*PDF_info_path()* を使ってパスの外接枠の隅の座標を決定し、望む位置にそれを配置します。ここで (*x1, y1*) は左下隅です：

```
String svgPath="M 100 100 L 300 100 L 200 300 z";

/* 上記のSVGパス記述からパスオブジェクトを作成 */
int path = p.add_path_point(-1, 0, 0, "addpath", "svgpath = {" + svgPath + "} close");
String mirroropt = "scale= {1 -1}";

double corner_x = p.info_path(path, "x1", mirroropt);
double corner_y = p.info_path(path, "y1", mirroropt);
double target_x = 200, target_y = 500;

/* 選択した隅を、望む位置に配置。「topdown」ページオプションを用いている場合には
   (target_x-corner_x, target_y+corner_y)をかわりに使用。
*/
p.draw_path(path, target_x-corner_x, target_y-corner_y,
            mirroropt + " stroke linewidth=3");
```

クックブック A full code sample can be found in the Cookbook topic `graphics/svg_path`.

クックブック 完全なコードサンプルがクックブックの `graphics/svg_path` トピックにあります。

パスオブジェクトをオブジェクトの外接枠から作成 配置された PDF ページ・SVG グラフィック・パス・ラスター画像・表組み・範囲枠・テキストフロー・テキスト行を内包する長方形（外接枠）を、対応する *PDF_info_**() メソッドを用いて抽出できます：

```
optlist = "boxsize={400 300} fitmethod=clip matchbox={name=border}";
p.fit_image(image, 200, 150, optlist);

int path = (int) p.info_matchbox("border", 1, "boundingbox");
p.draw_path(path, 0, 0, "close stroke linewidth=10 strokecolor=red");
p.delete_path(path);
```

パスオブジェクトを使用 パスオブジェクトを作成または取得した後は、それをさまざまな目的に使えます：

- ▶ *PDF_draw_path()* を用いて、ページ上でそのパスの塗り・描線を行ったり、パスをクリッピングパスとして使用したりすることができます。

- ▶ `PDF_fit_textflow()` を用いてそのパスに複数行テキストを回り込ませる：テキストが任意の形状の内部または外部を回りこむように組まれます (255 ページ「9.2.11 テキストをパス・画像に回り込ませる」参照)。
- ▶ `PDF_fit_textline()` を用いてそのパスの上にテキストを配置、すなわち、キャラクター群がそのパスの曲線に沿って並びます (235 ページ「9.1.7 パス上テキスト」参照)。
- ▶ `PDF_add_table_cell()` を用いてパスオブジェクト群を表セル内に配置。

3.2.5 テンプレート (フォームXObject)

テンプレート (PDF 内のフォーム XObject) PDFlib では、技術用語で**フォーム XObject** と呼ばれる PDF の機能に対応しています。しかしこの用語は、対話的なフォームとまぎらわしいため、私達はこの機能を「テンプレート」と呼びます。PDFlib のテンプレートは、ページ外のバッファと捉えることができ、そこではテキスト・ベクトル・画像の操作が行えます (ページ上で直接操作するのではなく)。テンプレートができれば、それはラスター画像のように使うことができ、任意の回数、任意のページに貼り付けることが可能です。画像同様、テンプレートには拡大縮小や斜形化などの幾何学的変形を施すことができます。1 つのテンプレートを複数のページで使った場合には (ないしは同じページで複数回)、テンプレートを構成する PDF オペレーターは実際には PDF ファイル中に 1 回しか書かれていないので、PDF 出力ファイルサイズの節約になります。テンプレートは、複数のページに繰り返し現れるものに対して推奨されます。たとえば、毎ページ同じ背景や、企業ロゴや、CAD ソフト・地図作成ソフトの吐き出す図記号などです。テンプレートは、クリッピングパスを持ったラスター画像を複数回配置する場合にも推奨されます。テンプレートを作成するには以下の方法があります：

- ▶ `PDF_begin_template_ext()` で直接。
- ▶ ベクトルグラフィックから `PDF_load_graphics()` と `templateoptions` オプションで間接的に。
- ▶ ラスター画像から `PDF_load_image()` と `templateoptions` オプションで間接的に。このオプションがないと、`PDF_load_image()` は、画像 XObject という同様の PDF 構造を作成します。

注 `PDF_open_pdi_page()` で取り込まれた PDF ページも PDF フォーム XObject を作成しますが、これはテンプレート関数ではなく PDI 関数で扱われます。

テンプレートは以下のように利用できます：

- ▶ `PDF_fit_image()` を用いてそのテンプレートをページ上または他のコンテンツストリーム上に配置 (後述)：
- ▶ そのテンプレートによって定義した輝度ソフトマスクを用いてグラフィックステートを作成 (`PDF_create_gstate()` のオプション `softmask` のサブオプション `template`、105 ページ「4.9.2 ソフトマスクを用いて色を変える」参照)。
- ▶ SVG グラフィックを読み込む際のフォールバック (背景) として (`PDF_load_graphics()` のオプション `fallbackimage`)。
- ▶ 注釈の体裁として (`PDF_create_annotation()` のオプション `template` のサブオプション `normal/rollover/down`)。
- ▶ 押しボタンフォームフィールドの体裁として (`PDF_create_field()` のオプション `icon/icondown/iconrollover`)。

テンプレートを PDFlib で作成・利用 テンプレートは、ラスター画像とまったく同様に、`PDF_fit_image()` メソッドでページ上または他のテンプレート上に貼り付けることができ

まず (221 ページ「8.4 画像・グラフィック・取り込み PDF ページを配置」参照)。一般に、PDFlib でテンプレートを作成・活用する場合には以下のようなコードになります：

```
/* テンプレートを定義 */
template = p.begin_template_ext(template_width, template_height, "");
...いろいろなテキスト・ベクトル・グラフィック関数を用いてテンプレート上に描画...
p.end_template_ext(0, 0);
...
p.begin_page(page_width, page_height);
/* テンプレートを利用 */
p.fit_image(template, 0.0, 0.0, "");
...いろいろなページ描画操作を追加...
p.end_page();
...
p.close_image(template);
```

あらゆるテキスト・グラフィック・色関数がテンプレート上では使えます。ただし、以下の関数は、テンプレートを作成している間には使ってはいけません：

- ▶ **PDF_begin_item()** と、さまざまな関数の **tag** オプション：構造エレメントはテンプレート内では作成できません。
- ▶ すべてのインタラクティブ関数：これらは、配置したい文書ページ上で定義しなければならず、テンプレートの一部として生成することはできません。

クックブック 完全なコードサンプルがクックブックの `general/repeated_contents` トピックにあります。

3.3 PDF のパスワードセキュリティ

3.3.1 PDF におけるパスワードセキュリティ

PDF のパスワードセキュリティは、以下の保護機能を提供します：

- ▶ ユーザーパスワード（開くパスワードとも呼ばれます）が、ファイルを閲覧するために開くために必要。ユーザーパスワードを持つファイルのみが、クラッキングから安全です！
- ▶ マスターパスワード（所有者または権限パスワードとも呼ばれます）が、権限、ユーザーまたはマスターパスワードといったセキュリティ設定を変更するために必要です。ユーザーパスワードとマスターパスワードを持ったファイルは、いずれかのパスワードを与えることによって、閲覧するために開くことができます。
- ▶ 権限設定が、PDF 文書に対する、印刷やテキスト抽出といった特定の操作を制限します。
- ▶ 添付パスワードを、文書自体の本体内容ではなく、ファイル添付だけを暗号化するために与えることができます。

PDF 文書がこのいずれかの保護機能を使用する場合には、その文書は暗号化されます。

権限制限 PDF は、文書の操作に関するさまざまな制約を符号化することができ、これは個別に許可したり禁止したりすることができます：

- ▶ **印刷を許可**：印刷が許可されていなければ、Acrobat の印刷ボタンは無効のままになります。Acrobat は、**低解像度印刷（150 dpi）**と**高解像度印刷**の区別に対応しています。低解像度印刷は、ページのラスター画像を生成し、これは私的利用にのみ適してはいますが、高解像度での複製ができないようになっています。画像ベースの印刷は、低い出力品質となるのみならず、印刷処理をかなり遅くさせることにも留意してください。
- ▶ **変更を許可**：以下の一覧は、さまざまな文書変更操作に対する制御を提供します：

ページの挿入、削除、回転

フォームフィールドへの入力と既存の署名フィールドに署名

注釈の作成、フォームフィールドの入力と既存の署名フィールドに署名

ページの抽出を除くすべての操作

Adobe Reader が表示する権限制限は信頼できないことに留意してください。たとえば、それはアセンブリ機能を持っていないので、文書内の実際の権限設定にかかわらず、つねに「**文書アセンブリ：許可しない**」と表示します。

- ▶ 内容のコピーは、「**テキスト、画像、およびその他の内容のコピーを有効にする**」を通じて制御されます。これは、「**スクリーンリーダーデバイスのテキストアクセスを有効にする**」を用いて有効化することもできますが、PDF 2.0 では、PDF リーダーはつねにアクセシビリティに対応するべきなので、この設定は廃止済です。

文書に対して、「**印刷を許可：なし**」といったアクセス制限を指定すると、Acrobat 内の各機能も無効になります。ただし、これは他の PDF ビューアーでもそうとは限りません。アクセス権限に従うかどうかは、PDF ツールの開発者次第なのです。

3.3.2 PDFlib を用いて PDF 文書をパスワード保護

PDFlib は、PDF 文書を生成する際に、標準のセキュリティ機能を適用することができます。保護された PDF 文書から、PDFlib+PDI か PDFlib Personalization Server (PPS) でページを取り込むには、マスターパスワードか *shrug* オプションが必要です。文書のプロパ

ティを pCOS インターフェイスでクエリーする際には、pCOS モードで制御されます。たとえば、XMP 文書データ・文書情報フィールド・しおり・注釈内容は、その文書がユーザーパスワードを必要としない場合には（またはユーザーパスワードのみが与えられた場合には）、マスターパスワードなしで取得できます。これについては pCOS パスリファレンスでさらに詳しく説明しています。

注 PDFlib GmbH 製品で PDF 文書を Reader 有効化する（たとえば Acrobat Reader で注釈を許す）ことはできません。

暗号化アルゴリズムとキー長 パスワードを用いて文書を保護するために使用される暗号化アルゴリズムとキー長は、`PDF_begin_document()` の `compatibility` オプションに従って生成される文書の PDF バージョンに依存します：

- ▶ PDF 1.4・1.5：利用可能なセキュアな暗号化アルゴリズムがないので、暗号化をサポートしていません。
- ▶ PDF 1.6・PDF 1.7・PDF 1.7ext3：AES-128 が使用されますが、これらの PDF バージョンでの暗号化は廃止済です。PDF 1.7ext3（Acrobat 9）に従った AES-256 は、その既知の脆弱性があるので、全く使用されません。
- ▶ PDF 1.7ext8・PDF 2.0：Acrobat X/XI/DC に従った AES-256 が使用されます。これが推奨されるアルゴリズムです。デフォルトの PDF 出力バージョンは PDF 1.7ext8 なので、これがデフォルトで使用されます。

パスワードを PDFlib で設定 パスワードは、`PDF_begin_document()` の `userpassword`・`masterpassword` オプションで設定できます。PDFlib は、生成される文書のためにクライアントが与えたパスワードと、以下の方式で相互作用します：

- ▶ ユーザーパスワードか何らかの権制限が与えられていながら、マスターパスワードが与えられていない場合には、通常のユーザーが簡単にセキュリティー設定を変更することができるので、いかなる保護も無効となります。この理由から、PDFlib はこの状況をエラーと見なします。
- ▶ ユーザーパスワードとマスターパスワードが同じ場合には、そのファイルのユーザーと所有者の区別がもはやつかないので、この場合も効果的な保護は無効となります。PDFlib はこの状況をエラーと見なします。
- ▶ AES-256 では Unicode のパスワードが許されますが、AES-128 のパスワードは Latin-1 文字集合に限られます。
- ▶ パスワードは、AES-256 の場合には 127 UTF-8 バイトで、AES-128 の場合には 32 キャラクターで切り捨てられます。

権限を PDFlib で設定 操作制限は `PDF_begin_document()` の `permissions` オプションで設定することができます。それは操作制限の入った 1 個ないし複数の文字列で構成されます。`permissions` オプションを設定するには `masterpassword` オプションも設定しなければなりません。なぜならそうでなければユーザーが簡単に権限設定を取り除けてしまうからです。デフォルトではすべての操作が許可されています。操作制限を指定すると Acrobat のその機能は無効になります。操作制限はユーザーパスワードなしで適用することができます。以下の例のようにスペースで区切れば、複数の制限キーワードを指定することもできます：

```
p.begin_document(filename, "masterpassword=abcd1234 permissions={noprint nocopy}");
```

表 3.3 に、使えるすべての操作制限キーワードを挙げます。

クックブック 完全なコードサンプルがクックブックの `general/permission_settings` トピックにあります。

表 3.3 `PDF_begin_document()` の `permissions` オプションに対する操作制限キーワード

キーワード	解説
<code>noprint</code>	Acrobat が、ファイルの印刷を拒みます。
<code>nomodify</code>	Acrobat が、ユーザーによるフォームフィールドの追加やその他あらゆる変更を拒みます。
<code>nocopy</code>	Acrobat が、テキストやグラフィックのコピーや抽出を拒み、アクセシビリティインターフェイスを無効にします。
<code>noannots</code>	Acrobat が、コメントやフォームフィールドの追加・変更を拒みます。
<code>noforms</code>	(<code>noannots</code> を暗黙に前提します) Acrobat が、 <code>noannots</code> が指定されていない場合でも、フォームフィールドへの記入を拒みます。
<code>noaccessible</code>	(PDF 2.0 では廃止済であり利用できません) Acrobat が、アクセシビリティを目的としたテキストやグラフィックの抽出を拒みます。
<code>noassemble</code>	(<code>nomodify</code> を暗黙に前提します) Acrobat が、 <code>nomodify</code> が指定されていない場合でも、ページの挿入・削除・回転やしおり・サムネールの作成を拒みます。
<code>nohiresprint</code>	Acrobat が、高解像度印刷を拒みます。 <code>noprint</code> が指定されていない場合は、印刷は「画像として印刷」機能に制限されます。すなわちその場合、ページが低解像度に変換されたものを印刷することしかできません。
<code>plain-metadata</code>	暗号化文書でも、文書のメタデータを暗号化しないうまにします。これは XMP メタデータにのみ影響し、文書情報フィールドには影響しません。

暗号化ファイル添付 添付ファイルは、それ以外は暗号化されていない文書の中にあっても、暗号化することが可能です。これを実現するには、`PDF_begin_document()` で `attachmentpassword` オプションを与えます。

3.4 XMP メタデータ

文書情報フィールドのかわりとして、あるいはそれに加えるものとして、PDFlib はメタデータを指定するためのフレームワークとして、XMP (*Extensible Metadata Platform*) に対応しています。XMP は ISO 16684-1:2012 として標準化されています。PDFlib における XMP 対応にはいくつかの側面がありますので、以下説明します。

クックブック シンプルな XMP サンプルが、クックブックトピック `interchange/embed_xmp` 内にあります。

多くの場合、XMP は文書全体にメタデータを紐付けるために用いられます。文書レベルメタデータのほかに、XMP はページ・フォント・画像・グラフィック・画像・取り込み PDF ページに対して与えることもできます。これは、さまざまなメソッドの *metadata* オプションで実現できます。たとえば：

```
metadata={filename=info.xmp inputencoding=winansi}
```

この *metadata* オプションは、完全な XMP メタデータストリームないしはその一部分を受け付けます。PDFlib は、ユーザーが与えた XMP メタデータを、XML 規則と XMP/RDF 規則に従って検証します。PDF/A の場合には、カスタム XMP プロパティに対する追加規則が適用されます：355 ページ「12.2.8 PDF/A のための XMP 文書メタデータ」を参照してください。

内部・予約 XMP プロパティ PDFlib は、いくつかの XMP プロパティを内部的に作成します。たとえば *CreationDate* です。他に、PDF/A や PDF/X といったさまざまな PDF 企画への準拠をシグナルするために必須の XMP プロパティ群があります。内部プロパティと規格関連識別プロパティは、ユーザーが与える XMP でオーバーライドすることはできません。

文書情報フィールドに対する自動 XMP 同期 `PDF_begin/end_document()` の *autoxmp* オプションが *true* の場合には、PDFlib は、`PDF_set_info()` に与えられた文書情報フィールド群と、いくつかの内部的に生成されたエンタリー (*CreationDate* など) とを、その対応する、文書レベル XMP メタデータ内のエンタリー群へ同期します。

標準 XMP スキーマ群のうちの中にあるよく知られたプロパティに対応する文書情報フィールドは、適切なスキーマ内に配置されます。カスタム情報フィールドは拡張 PDF (*pdfx*、PDF/X とは無関係です) XMP スキーマ内に配置されますが、PDF/A モードでは、PDF/A ではカスタム XMP プロパティには拡張スキーマ記述が必須なので、無視されます。

XMP メタデータを転写 PDF 文書の大半ないしすべてのページが取り込まれたものである場合には、XMP メタデータがもし入力の中に存在していれば、それを転写することを推奨します。XMP メタデータを転写するには、以下のコード断片を使います：

```
if (p.pcos_get_string(indoc, "type:/Root/Metadata").equals("stream"))
{
    xmp = p.pcos_get_stream(indoc, "", "/Root/Metadata");
    p.create_pvf("/xmp/document.xmp", xmp, "");
    optlist += " metadata={filename=/xmp/document.xmp}";
}

p.end_document(optlist);
p.delete_pvf("/xmp/document.xmp");
```

3.5 連携ファイル

注 連携ファイルに関するさらなる情報が、PDF 協会が発行した Application Note 0002 にあります (www.pdfa.org/resource/pdf-2-0-application-note-002-associated-files/)。

連携ファイルはファイル添付と似ていますが、PDF 文書内の個別のオブジェクトにデータを接続するために使えます。連携ファイルを使うと、任意の形式のコンテンツを PDF のオブジェクトに紐付けることができ、関係の種類も指定できます。1 つないし複数のファイルを 1 つのオブジェクトと連携させることが可能です。これはさまざまな API メソッドのオプション *associatedfiles* を用いて実現できます。

連携ファイルは、*PDF_load_asset()* を用いて読み込む必要があります。通常のファイル添付と異なり、連携ファイルはデフォルトでは Acrobat の添付ペーンにリストされていません。アセットをオプション *documentattachment* を用いて読み込むと、その連携ファイルは文書添付のリストに入ります。その結果、この連携ファイル (群) は添付の一覧で見えるようになります。これらのファイルは、オプション *external=true* を与えていない限り、生成される PDF 文書の中に埋め込まれます。

連携ファイルは PDF 2.0 の一部ですが、PDF/A-3 で導入されており、そちらは連携ファイルに以下の制限を課します：

- ▶ 埋め込んでいない外部ファイルへの参照、すなわち *external=true* は許されません。
- ▶ 連携ファイルを構造ヒエラルキー (文書オプション *structureassociatedfiles*) と XMP メタデータに対して使うことは許されません。

連携ファイルに対応しているすべての API メソッドとオプションを表 3.4 に挙げます。用途では *relationship* オプションのよくある値にも言及しています。

表 3.4 associatedfiles オプションまたはサブオプションに対応している API メソッド

API メソッド	ファイルの連携対象	用途の例
<i>PDF_end_document()</i>	文書全体	ZUGFeRD と Fatur-X は、そのインボイスの XML 版を PDF/A-3 文書内へ埋め込みます (<i>relationship=Alternative</i>)。 その文書に対して、ワードプロセッサかスプレッドシートの元ファイル (<i>relationship=Source</i>)
<i>PDF_begin/end_page_ext()</i>	ページ	ベンダー独自の PieceInfo 辞書に対する代替として連携ファイルが機能します
<i>PDF_begin/end_document()</i> ・ <i>PDF_begin/end_page_ext()</i> など オプション <i>metadata</i> を持つ API メソッド	XMP メタデータ	XMP メタデータに対してスキーマ記述 (<i>relationship=Schema</i>)
<i>PDF_load_graphics()</i> で <i>templateoptions</i> <i>PDF_open_pdi_page()</i> <i>PDF_begin_template_ext()</i>	テンプレートが取り込んだページのために作ったフォーム XObject、またはグラフィック	図表に対して生データ、表に対して CSV データ、グラフに対して生の測定データ (<i>relationship=Data</i>)。 数式に対して MathML 表現 (<i>relationship=Alternative</i>)
<i>PDF_load_image()</i>	取り込んだ画像のために作った 画像 XObject	画像に対してソースデータ (<i>relationship=Source</i>)

表 3.4 associatedfiles オプションまたはサブオプションに対応している API メソッド

API メソッド	ファイルの連携対象	用途の例
PDF_create_annotation()	注釈 (type=FileAttachment でオプション attachment を用いた場合には、埋め込んだファイルはかわりにページに連携します)	リッチメディア注釈か 3D 注釈に対してソースデータ (relationship=Source)
PDF_begin_item(), またはさまざまなメソッドの tag オプション	論理構造ヒエラルキー内の構造エレメント	レスポンス PDF は、HTML・SVG・JavaScript・CSS を埋め込むことによって、他の形式へ変換された際にその構造エレメントの内容にスタイル付けをします (relationship=Alternative か Supplement)
PDF_end_document() で structureassociatedfiles	論理構造ヒエラルキー全体	その構造ツリーを他の形式へ変換するための補足情報 (relationship=Supplement)
PDF_begin/end_dpart()	DPart ヒエラルキー内の文書部分	その文書パートに関する追加情報 (relationship=Supplement)

クックブック 連携ファイルを文書内のさまざまなオブジェクトにつなぐ完全なコードサンプルが、PDFlib クックブックのトピック interchange/associated_files 内にあります。PDF 文書から連携ファイルを取得するコードサンプルが、PDFlib クックブックのトピック interchange/retrieve_associated_files 内にあります。

以下のコード断片は、ファイルアセットを読み込んで、それをページに連携し、その際にその関係を *Data* と指定しています：

```
af = p.load_asset("Attachment", af_filename, "relationship=Data");

if (af == -1) {
    throw new Exception("エラー : " + p.get_errmsg());
}

p.begin_page_ext(0, 0, "associatedfiles={" + af + "} width=a4.width height=a4.height");
```



4 色と透過

クックブック 色空間は、塗り操作と描線操作に対してそれぞれ別個に指定できます。たとえば `fillcolor`・`strokecolor` オプションを使用します。PDF において色空間は画像に対しても使用されますが、PDFlib は通常、画像の色空間を自動的に決定します。ただしカラー関連のオプションが `PDF_load_image()` で与えられた場合を除きます。

クックブック カラー処理のコードサンプルがPDFlibクックブックのcolorカテゴリーにあります。color/starter_color サンプルにすべての色空間の使用が演示されています。

4.1 デバイス色空間

デバイス独自の色空間は、おそらく、色を記述するために最も広く用いられる手段です。名前のおと、これは、ある特定のデバイスによって解釈されるものとして色を記述します。デバイスの例としてはモニターやプリンターや印刷機などです。デバイス独自カラーは、どのデバイスで出力を行うかによって、見た目が異なります。この望ましくない性質があるので、デバイスカラーは、PDF/A と PDF/X においては、直接的には許されておらず、然るべき出力インテント ICC プロファイルかデフォルト色空間とともにのみ使用が許されます。

グレースケール色空間 グレースケールカラーは、色空間キーワード `gray` を用いて要求することができ、1 個のグレー値です。グレー値は 0 = 黒から 1 = 白までの範囲です。例：

```
p.set_graphics_option("fillcolor={{ gray 0.5 }}");
```

RGB 色空間 加法的な RGB 色空間の色は、赤・緑・青の成分を混ぜて作られます。RGB カラーは、色空間キーワード `rgb` を用いて要求することができ、赤・緑・青の割合を指定する範囲 0 ~ 1 の 3 個の RGB 値であり、`(0, 0, 0)` = 黒、`(1, 1, 1)` = 白になります。広く用いられている範囲 0 ~ 255 の RGB カラー値を、PDFlib が要求する範囲 0 ~ 1 へスケールするには、255 で割る必要があります。

数値で RGB 値を指定せず、その HTML 名または 16 進値で RGB カラーを指定することもできます。例：

```
p.set_graphics_option("fillcolor={{ rgb 1 0 0 }}");  
p.set_graphics_option("fillcolor=pink");  
p.set_graphics_option("fillcolor={{#FFC0CB}}");
```

CMYK 色空間 減法的な CMYK 色空間の色は、シアン・マゼンタ・イエロー・黒（「キー」）の成分を混ぜて作られます。CMYK 値は、色空間キーワード `cmyk` を用いて要求することができ、シアン・マゼンタ・イエロー・黒を表す範囲 0 ~ 1 の 4 個の CMYK 値であり、0 = 色なし、1 = 色フルを表します。`(0, 0, 0, 0)` = 白、`(0, 0, 0, 1)` = 黒になります。CMYK カラーの極性はグレースケール・RGB カラーとは異なっていることに注意してください。例：

```
p.set_graphics_option("fillcolor={{ cmyk 0 1 0 0 }}");
```

この CMYK 色空間の特質として、CMYK オブジェクトのオーバープリント動作を、`overprintmode` オプションを用いて制御できるということがあります（107 ページ「CMYK カラーに対するオーバープリントモード」参照）。

デバイスカラーをデフォルト色空間を用いてデバイス独立カラーへマップ デバイス色空間は、デバイス独立色空間へ、デフォルト色空間を用いてマップすることが可能です：詳しくは 87 ページ「デバイスカラーを ICC ベースの色空間へマップ」を参照してください。これは、RGB または CMYK の画像またはグラフィックなど、デバイス独自カラーを有するデータが、PDF/A または PDF/X の中で、合致する出力インテントを持たずに使用されている場合に有用です。

4.2 ICC プロファイルによる色管理

国際標準化機構 (ISO) の国際カラー Consortium (ICC) は、入力デバイスや出力デバイスの色の特性を指定するためのファイル形式を定義しました。PDFlib は、ICC プロファイルとレンダリングインテントによる色管理に対応しています。ICC プロファイルは、色管理されたワークフローと、PDF/X・PDF/A といった多くの PDF 規格において、重要な役割を担います。

クックブック 完全なコードサンプルが、`starter_color` サンプルとクックブックの `color/iccprofile_to_image` トピックにあります。

4.2.1 ICC プロファイルを読み込む

注 よくある印刷条件のための ICC カラープロファイル群への推奨とリンクが www.pdflib.com で入手可能です。

受け入れ可能な ICC プロファイル ICC プロファイルを読み込むには `PDF_load_iccprofile()` を用います。プロファイルは、そのプロファイルの ICC バージョン番号・デバイスクラス・データ色空間によって定まる条件を満たす必要があります。PDF 1.4 は ICC プロファイルバージョン 2.x に対応しており、PDF 1.5 以上は ICC プロファイルバージョン 4.x に対応しています。表 4.1 に、ICC プロファイルに関するデバイスクラスとデータ色空間に関する、その用途に応じた追加の要請を詳述します。

表 4.1 さまざまな用途に対する ICC プロファイルの要請

ICC プロファイルの用途 (詳しくは表 4.2 を参照)	デバイスクラス	データ色空間
PDF/X-3/4 のための出力インテント	prtr	グレー・RGB・CMYK
PDF/X-5n のための出力インテント	prtr	xCLR (n 色)
PDF 2.0・PDF/A のための、文書レベルかページレベルの出力インテント	prtr・mnrtr	グレー・RGB・CMYK
透過グループ色空間	prtr・mnrtr・scnr・spac	グレー・RGB・CMYK
ICC プロファイルのその他すべての用途	prtr・mnrtr・scnr・spac	グレー・RGB・CMYK・Lab

ICC プロファイルを検索 PDFlib は、`PDF_load_iccprofile()` に与えられた `profilename` 引数を用い、以下の手順を踏んで ICC プロファイルを検索します：

- ▶ `profilename=sRGB` ならば、PDFlib はその内部 sRGB プロファイル (後述) を用い、検索は打ち切られます。
- ▶ `ICCProfile` リソースカテゴリー内に `profilename` という名前のリソースがあるかどうかをチェックします。もしあれば、その値をファイル名として以下の手順で用います。そのようなリソースがない場合は、`profilename` をファイル名として直接用います。
- ▶ 前の手順で決定されたファイル名を用い、以下の組み合わせを 1 つずつ順に試してみるにより、ディスク上のファイルを検索します：

<ファイル名>

<ファイル名>.icc

<ファイル名>.icm

<colordir>/<ファイル名>

(Windows・macOSのみ)

<colordir>/<ファイル名>.icc (Windows・macOSのみ)
<colordir>/<ファイル名>.icm (Windows・macOSのみ)

Windows では *colordir* は、オペレーティングシステムがデバイス依存 ICC プロファイル
を格納しているディレクトリーを示します (たとえば *C:\Windows\System32\spool*
drivers)。macOS では *colordir* として以下のパスが試みられます :

```
/System/Library/ColorSync/Profiles  
/Library/ColorSync/Profiles  
/Network/Library/ColorSync/Profiles  
~/Library/ColorSync/Profiles
```

sRGB 色空間と sRGB ICC プロファイル PDFlib は、sRGB という標準的な RGB 色空間に
対応しています。これは、デジタルカメラのような消費者向け RGB デバイスやカラープ
リンター・モニターのような事務機器における簡単な色管理のために広く利用されていま
す。PDFlib はそれに必要な ICC プロファイルデータを内蔵しています。ですから sRGB プ
ロファイルを明示的に構成する必要はありません。これを利用するには、*PDF_load_*
iccprofile() を *filename=sRGB* で呼び出します。便利なショートカットとして、*PDF_load_*
iccprofile() で作成された ICC ハンドルが期待されるすべての場所において、かわりにキー
ワード *srgb* を与えることもできます。画像ファイルの中に ICC プロファイルが埋め込ま
れていなければ、この sRGB プロファイルが RGB 画像に対してデフォルトで適用されます。
sRGB プロファイルは、デバイスクラス *mntr* (出力デバイス) に属します。すなわち、
これは PDF/A に対する出力インテントとしては使用できませんが、PDF/X に対する出力イ
ンテントとしては使用できません。

4.2.2 ICC プロファイルを使用

PDFlib は表 4.2 に挙げる用途のための ICC プロファイルに対応しています。

表 4.2 ICC プロファイルのさまざまな用途

用途	関係する API メソッド・オプション
ページ上のテキストとベクトルグラフィックに ICC ベースの色空間を設定	<i>PDF_set_option()</i> で <i>iccprofilegray/rgb/cmyk</i> および <i>PDF_setcolor()</i> で <i>colorspace=iccbasedgray/rgb/cmyk</i> 色オプションで <i>iccbased</i> キーワード
取り込んだ画像に ICC プロファイルを適用	<i>PDF_load_image()</i> : オプション <i>iccprofile</i>
画像に埋め込まれた ICC プロファイルを処理ま たは無視	<i>PDF_load_image()</i> : オプション <i>honoriccprofile</i>
画像に埋め込まれた ICC プロファイルをクエ リー	<i>PDF_info_image()</i> で <i>keyword=iccprofile</i>
グレースケースか RGB か CMYK データを ICC ベースの色空間へマップするためのデフォルト 色空間を設定	<i>PDF_begin_page_ext()</i> ・ <i>PDF_begin_template_ext()</i> ・ <i>PDF_begin_</i> <i>pattern_ext()</i> ・ <i>PDF_begin_font()</i> : オプション <i>defaultgray/</i> <i>defaulttrgb/defaultcmyk</i>
PDF/X または PDF/A 文書の出力インテントを、 参照または埋め込み ICC プロファイルで指定	<i>PDF_load_iccprofile()</i> : オプション <i>usage=outputintenc</i>
PDF 2.0 のページレベルの出力インテントを、埋 め込み ICC プロファイルで指定	<i>PDF_begin_page_ext()</i> : オプション <i>outputintents</i>

表 4.2 ICC プロファイルのさまざまな用途

用途	関係する API メソッド・オプション
透過・ブレンドモードのためのブレンド色空間を指定	<code>PDF_begin/end_page_ext()</code> ・ <code>PDF_open_pdi_page()</code> ・ <code>PDF_begin_template_ext()</code> ・ <code>PDF_load_graphics()</code> で <code>templateoptions</code> : オプション <code>transparencygroup</code> 、サブオプション <code>colorspace</code>
スポットカラーのための代替色空間	<code>PDF_set_graphics_option()</code> : 色オプションでキーワード <code>spotname</code> と <code>iccbased</code> 代替色、または <code>PDF_set_graphics_option()</code> でオプション <code>fillcolor</code> の後に <code>PDF_makespotcolor()</code> 。
DeviceN カラーのための代替色空間	<code>PDF_create_devicen()</code> : オプション <code>alternate</code> でサブオプション <code>iccbased</code>
ICC プロファイル内の色成分の数をクエリー	<code>PDF_get_option()</code> でキーワード <code>icccomponents</code>



画像 (ICC タグ付き画像) 内の埋め込みプロファイルを用いる 画像のなかには、その画像のカラー値の特徴を記述した ICC プロファイルが埋め込まれているものがあります。たとえば、埋め込まれた ICC プロファイルは、画像データの生成に用いられたスキャナーの色特性を記述することができます。PDFlib は、JPEG・JPEG 2000・PNG・TIFF 画像ファイル形式の中に埋め込まれた ICC プロファイルを処理します。`honoriccprofile` オプションが `true` に設定されている場合 (デフォルトではそうになっています)、画像内に埋め込まれている ICC プロファイルはその画像から抽出され、PDF 出力内に埋め込まれます。

`PDF_info_image()` のキーワード `iccprofile` を使うと、画像内に埋め込まれているプロファイルに対する ICC プロファイルハンドルを得ることができます。これは、同じプロファイルを複数の画像に適用する必要がある場合に有用です。

未知の ICC プロファイル内の色成分数をチェックするには `icccomponents` オプションを用います。

外部 ICC プロファイルを画像に適用 画像に埋め込まれている ICC プロファイルを使うのではなく、外部プロファイルを各画像に適用することもできます。そのためには `PDF_load_image()` の `iccprofile` オプションでプロファイルハンドルを与えます。

ページ記述に対して ICC ベース色空間 テキストやベクトルグラフィックのカラー値は、ICC ベースの色空間で直接指定することができます。色オプションで、ICC プロファイルをハンドルで指定し、その後続けて色要素を然るべき数だけ (たとえば CMYK プロファイルなら 4 要素) 指定します :

```
icchandle = p.load_iccprofile("myCMYK", "usage=iccbased errorpolicy=exception");
p.set_graphics_option("fillcolor={iccbased=" + icchandle + " 0 1 0 0}");
```

あるいは、色空間を、グローバルな `iccprofilegray`・`iccprofilergb`・`iccprofilecmyk` オプションのうちいずれか1つへの値として ICC プロファイルハンドルを与えることによって設定することも可能です。その後、ICC ベースのカラー値を、色オプションか `PDF_setcolor()` に対して、`iccbasedgray`・`iccbasedrgb`・`iccbasedcmyk` のいずれかの色空間キーワードとともに与えることができます。

デバイスカラーを ICC ベースの色空間へマップ PDFには、デバイス依存なグレー・RGB・CMYK カラーをデバイス独立な ICC ベースの色へマップする機能があります。これを利用すると、そのままではデバイス依存なカラー値に対して、正確な測色指定を与えること

ができます。これを実現するには、`PDF_begin_page_ext()`・`PDF_begin_template_ext()`・`PDF_begin_pattern_ext()`・`PDF_begin_font()`の`defaultgray`・`defaultrgb`・`defaultcmyk` オプションに、然るべき ICC プロファイルハンドルを与えます。以下の例は、sRGB 色空間を、ページ上のテキスト・画像・ベクトルグラフィックのデフォルト RGB 色空間として設定します：

```
p.begin_page_ext(595, 842, "defaultrgb=srgb");
```

デフォルト色空間が外部 ICC プロファイルに由来する場合には、プロファイルハンドルをまず作成する必要があります：

```
/* ICCプロファイルハンドルを作成 */
icchandle = p.load_iccprofile("myRGB", "usage=iccbased errorpolicy=exception");
p.begin_page_ext(595, 842, "defaultrgb=" + icchandle);
```

出力インテントプロファイル 出力デバイスのプロファイルを用いて、PDF 2.0 または PDF/X または PDF/A 文書のための出力条件を指定することができます。そのためには、`PDF_load_iccprofile()` で `usage=outputintent` を指定します。PDF/A に対しては、プリンターまたはモニタープロファイルを出力インテントとして指定でき、PDF/X ではプリンタープロファイルのみ許されます。詳しくは 358 ページ「12.3 PDF/X による印刷出力」と 346 ページ「12.2 PDF/A によるアーカイビング」を参照してください。

4.2.3 高度な色管理制御

PDFlib アプリケーションはデバイス独立なカラー値を指定できますが、ある特定の出力デバイスにおいて、その色を正確に再現できないことがあるかもしれません。この場合には、何らかの妥協が行われる必要があります。この処理を色域圧縮といいます。これは、色の範囲を、そのデバイスによって再現可能な、より狭い範囲へ縮める処理です。PDF はこの処理を細かく調整する制御をいくつか提供しています。

レンダリングインテント レンダリングインテントは、個別の画像に対して、`PDF_load_image()` に `renderingintent` オプションを与えることによって指定できます。レンダリングインテントは、テキストとベクトルグラフィックに対しても、`PDF_create_gstate()` に `renderingintent` オプションを与えることによって指定できます。利用可能なすべてのレンダリングインテントを表 4.3 に挙げます。

表 4.3 利用可能なレンダリングインテント

インテント	説明	典型的用途
Auto	レンダリングインテントを PDF ファイル内で一切指定せず、デバイスのデフォルトインテントをかわりに使用	ページ内容に関して十分な情報が得られません
AbsoluteColorimetric	デバイスの白色点（紙色の白など）に対し補正を一切行わない。色域外の色は、デバイスの色域の中で最も近い値へマップされます。	ベタ色の正確な再現。それ以外の用途には推奨されません
RelativeColorimetric	色データをデバイスの色域へ圧縮。白色点を互いにマップします（すなわち、最も明るい色はたいてい白紙色によって実現されます）。それ以外の色は若干シフトされることになります。	ベクトルグラフィック
Saturation	色の彩度を温存。カラー値はシフトされる場合があります。	ビジネスグラフィック

表 4.3 利用可能なレンダリングインテント

インテント	説明	典型的用途
<i>PerceptualColor</i>	色の関係を温存。見た目の美しさを優先して、色域内の色も色域外の色も変更します。	現実画像

黒点補償 黒点補償は、最も暗い色が、出力デバイスが出せる最も暗い色へマップされるよう、色変換を調整します。これは *RelativeColorimetric* レンダリングインテントが白について出すのと同様の効果を黒について出すものです。それは最も明るい色を、出力デバイスが出せる最も明るい色へマップするものだからです。

黒点補償は、コントラストを高めることによって画像の質を向上させる可能性があります。PDF 2.0 でのみ利用可能であり、*PDF_create_gstate()* の *useblackptcomp* オプションでのみ有効化できます。

4.3 デバイス独立な CIE L*a*b* カラー

クックブック 完全なコードサンプルが `starter_color` サンプル内にあります。

デバイス独立なカラー値を、CIE 1976 L*a*b* 色空間 (略して *Lab*) において指定することができます。この L*a*b* 色空間における色は、3 個の値 $L \cdot a \cdot b$ によって指定されます (図 4.1 参照)。輝度 (または明度) L は 0 ~ 100 の範囲にわたります。値 a と b は範囲 -128 ~ 127 で色彩を記述します。値 a は緑 (-128) からマゼンタレッド (+127) までの範囲にわたり、値 b は青 (-128) から黄色 (+127) までの範囲にわたります。正の数は暖色 (黄色・マゼンタレッド) を記述し、負の数は寒色 (緑・青) を記述します。 $a \cdot b$ 軸の交点 ($a=b=0$) は無彩色のグレー値を記述し、黒 ($0, 0, 0$) から白 ($100, 0, 0$) までの範囲にわたります。

この Lab 色空間の重要な特性として、RGB・CMYK と異なり、知覚的に一様であるということがあります。すなわち、色どうしの数値の差が等しければ、その視覚的な差も等しくなります。これによって、複数の色の混ぜ合わせの計算がはるかに簡単になります。各色の Lab 値の加重平均をとればよいからです。96 ページ「Lab 代替空間を持ったスポットカラー群に基づく DeviceN 色空間」ではこのことを利用して PostScript 変換関数を生成します。

Lab 値は、白色点もあわせて指定されている場合にのみ、絶対的な色を指定します。PDFlib は、白色点として、標準光源 D50 (日中光 5000 K、2° 測定者) を使用します。

PDFlib は Lab 色空間に以下のように対応しています：

- ▶ Lab カラーをテキスト・ベクトルグラフィックに使用するには、色オプションと `PDF_setcolor()` において、色空間キーワード `lab` と、 $L \cdot a \cdot b$ 値の 3 個の数値を用います。例：

```
p.set_graphics_option("fillcolor={lab 100 0 0}");
```

PDF/A モードでは、デフォルトの黒の塗り・描線色は、出力インテントが指定されていない場合は、デバイス独立な Lab ($0, 0, 0$) として指定されます。

- ▶ PDFlib は内蔵データベースにおいて、すべての HKS・Pantone スポットカラーの代替カラー値に Lab 色空間を使用しています。
- ▶ Lab 色空間を用いた TIFF 画像を取り込みます。
- ▶ Lab 色空間を用いた ICC プロファイルを、`PDF_load_iccprofile()` を用いて読み込みます。ただし、出力インテントとして使用しようとする場合、または `transparencygroup` オプションに使用しようとする場合を除きます。なぜなら PDF はこれらの箇所では Lab を許さないからです。

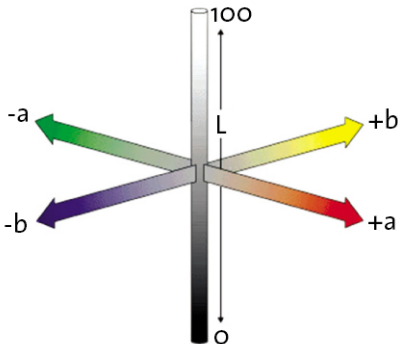


図 4.1 CIE L*a*b* 色空間

4.4 Pantone・HKS・カスタムスポットカラー

クックブック 完全なコードサンプルが、`starter_color` サンプルとクックブックの `color/spot_color` トピックにあります。

スポットカラーのための Separation 色空間 PDFlib はスポットカラーに対応しています (技術的には PDF では Separation 色空間として知られます)。スポットカラーとは、プロセスカラーの混色領域外にあるカスタムカラーの印刷に用いられるものです。スポットカラーは名前指定され、PDF では代替色を伴う必要があります。この代替色は、そのスポットカラーに近い色が選ばれますが、まったく同じ色ではありません。PDF ビューアーや RIP は、この代替色を用いて、画面表示やスポットカラー非対応機 (事務用プリンターなど) への印刷を行います。印刷機では、要求されたスポットカラーが適用され、文書内で用いられているその他すべてのプロセスカラーとともに印刷されます。

PDFlib は、さまざまな組み込みスポットカラーライブラリーに対応しており、かつ、カスタム (ユーザー定義) スポットカラーにも対応しています。スポットカラー名が `PDF_makespotcolor()` によって要求されると、PDFlib はまず、その要求されたスポットカラーが PDFlib 組み込みライブラリーのうちのいずれかの中で見つかるかどうかをチェックします。もし見つければ、PDFlib は代替色に関して、内蔵の Lab 値を用います。見つからない場合、そのスポットカラーはユーザー定義色と見なされるので、クライアント側でそれに対して適切な代替カラー値を、カラーオプション群の `spotname` カラーキーワードで (あるいは `PDF_setcolor()` を使用の際にはカレント塗り色を通じて) 与える必要があります。スポットカラーには濃度を指定することもできます。すなわち、0 = 色なし = 白から 1 = 最大濃度までの色濃度を指定するパーセント値とともに用いることができます。Separation 色空間はつねに減法的です (RGB のような加法的な色空間では 0 = 黒になると異なり)。スポットカラーが定義されたら、それを用いて、テキスト・ベクトルグラフィックを描くこともできますし、グレースケール画像に着色することもできますし、シェーディングを構築することもできます。なお、画像にスポットカラーで着色する際には、PDFlib はその色の極性を自動的に反転させます (197 ページ「グレースケール画像にスポットカラーで着色」参照)。

デフォルトでは、組み込みスポットカラーはカスタム代替値で再定義することはできません。この動作は、`PDF_set_option()` の `spotcolorlookup` オプションを用いて変更することもできます。これを使うと、古いアプリケーションが異なる色定義を用いているような場合との互換性がとれますし、また、Pantone カラーに対する PDFlib の Lab 代替値を扱うことのできないワークフローにおいても有用です。

注 組み込み Pantone®・HKS® スポットカラーのデータとその対応する商標については、PDFlib ソフトウェアでの使用のためのライセンスを PDFlib GmbH は各商標権者から取得しています。

Pantone® カラー Pantone カラーは世界的に有名で広く利用されています。PDFlib は、Pantone Matching System® (名前付きカラー数千色を有する) に完全対応しています。表 4.4 に示すデジタルカラーライブラリーにあるすべてのカラーズウォッチ名が利用できます。

スポットカラー名は大文字・小文字を区別します。上記の例と同様に大文字を使ってください。例示のように、必ず接頭辞 **PANTONE** をスウォッチ名につける必要があります。Pantone カラー名は次の方式に従って構成されます：



PANTONE <id> <paperstock>

ここで <id> は色の識別子であり (たとえば 185)、<paperstock> はペーパーストックなど指定の頭文字です (たとえば C は coated = コート紙)。スウォッチ名を構成する各成分の間にはスペースを 1 つずつ入れる必要があります。PANTONE 接頭辞ではじまる名前のスポットカラーが要求されたにもかかわらず、その名前が有効な Pantone カラーを表していなかった場合には、警告がログ記録されます。以下のコードスニペットは、ある Pantone カラーを濃度値 70 パーセントで用いた例です：

```
p.set_graphics_option("fillcolor={ spotname {PANTONE 281 U} 0.7 }");
```

表 4.4 PDFlib に内蔵の Pantone スポットカラーライブラリー

カラーライブラリー名	カラー名の例	注
PANTONE ソリッド／コート紙	PANTONE 185 C	2012 年追加の範囲 2001 ~ 2336 の 336 色。 2014 年追加の範囲 2337 ~ 2427 の 84 色。 2016 年追加の範囲 2428 ~ 3599 の 112 色。 2019 年追加の範囲 4001 ~ 4294 の 294 色。
PANTONE ソリッド／上質紙	PANTONE 185 U	2012 年追加の範囲 2001 ~ 2336 の 336 色。 2014 年追加の範囲 2337 ~ 2427 の 84 色。 2016 年追加の範囲 2428 ~ 3599 の 112 色。 2019 年追加の範囲 4001 ~ 4294 の 294 色。
PANTONE 拡張色域／上質紙 (XGC)	PANTONE 185 XGC	2015 年追加の 1736 色
PANTONE パステル／コート紙	PANTONE 9461 C	
PANTONE パステル／上質紙	PANTONE 9461 U	
PANTONE メタリック／コート紙	PANTONE 871 C	

HKS® カラー HKS カラーシステムはドイツなどの欧州諸国で広く利用されています。PDFlib は HKS カラーに完全対応しています。表 4.5 に挙げるデジタルカラーライブラリーにあるカラースウォッチ名がすべて利用可能です。

スポットカラー名は大文字・小文字を区別します。上記の例と同様に大文字を使ってください。HKS という接頭辞は必ず、例示したようにスウォッチ名につける必要があります。HKS カラー名は次の方式に従って構成されます：

HKS <id> <paperstock>

ここで <id> は色の識別子であり (たとえば 43)、<paperstock> は利用するペーパーストックの頭文字です (たとえば N は natural 紙)。スウォッチ名を構成する HKS・<id>・<paperstock> 各要素の間にはスペースを 1 つずつ入れる必要があります。上記 2 番目の方式はライブラリーかライブラリーの中の色に対してのみ用いることができます。HKS 接頭辞ではじまる名前のスポットカラーが要求されたにもかかわらず、その名前が有効な HKS カラーを表していなかった場合には、警告がログ記録されます。以下のコードスニペットは、ある HKS カラーを濃度値 70 パーセントで用いた例です：

```
p.set_graphics_option("fillcolor={ spotname {HKS 38 E} 0.7 }");
```



表 4.5 PDFlib に内蔵の HKS スポットカラーライブラリー

カラーライブラリー名	カラー名の例	注
HKS K	HKS 43 K	グロスアート紙用 88 色 (Kunstdruckpapier)
HKS N	HKS 43 N	ナチュラル紙用 86 色 (Naturpapier)
HKS E	HKS 43 E	連続ステーションリ / コート用 88 色 (Endlospapier)
HKS Z	HKS 43 Z	ニュースプリント用 50 色 (Zeitungspapier)

カスタムスポットカラー 上述の組み込みスポットカラーのほかに、PDFlib ではカスタムスポットカラーにも対応しています。これは、任意の名前と（ただし組み込みの色とち合う名前は使えません）代替色を持つことができます。この代替色は、画面表示と低品位印刷に用いられますが、高品位の色分版には用いられません。カスタムスポットカラーに適切な代替色を与えるのはクライアント側の役割です。

スポットカラーは、*fillcolor/strokecolor* テキストまたはグラフィック書式オプションとその他の色関連オプションで設定できます。代替色は、スポットカラー定義内で直接与えることができます：

```
fillcolor={spotname {CompanyRed} 1.0 {cmyk 0 0.78 0.88 0}}
```

あるいは、カレント塗り色を用いてスポットカラーを定義することも可能です。この場合には、カレント塗り色が代替色として用いられます。カレント塗り色を代替色として設定するために呼び出しが 1 つ多く必要であるという点を除けば、カスタムスポットカラーの定義と利用は、組み込みスポットカラーの利用と同様に行うことができます：

```
/* 代替色として使いたいカレント塗り色を設定 */
p.setcolor("fill", "cmyk", 0 0.78 0.88 0);
/* カレント塗り色からスポットカラーを生成 */
spot = p.makespotcolor("CompanyRed");

/* 塗りにスポットカラーを設定 */
p.set_graphics_option("fillcolor={spotname {CompanyRed} 0.7 }");
```

CMYK プロセカラーに基づく Separation 色空間 スポットカラー名は一般に、任意に選ぶことができますが、インキ名 *Cyan*・*Magenta*・*Yellow*・*Black* はつねに CMYK プロセカラーを指します。このことを利用して、特定の CMYK チャンネル（たとえばマゼンタ）のみに描画を行い、他のチャンネルに影響を与えないようにすることも可能です。これは、フル CMYK カラーで *Cyan=Yellow=Black=0* によって描画を行うのとは異なります。後者はシアン・イエロー・黒チャンネルの既存の内容を消去するからです（この動作はデバイス依存設定を用いて変えることも可能です。107 ページ「4.10 オーバープリント制御」参照）。たとえば、次の呼び出しは：

```
fillcolor={spotname Magenta 0.5 {cmyk 0 1 0 0}}
```

次のように CMYK カラーを直接使用するのは異なります：

```
fillcolor={cmyk 0 0.5 0 0}
```

なぜなら前者の呼び出しではシアン・イエロー・黒チャンネル内の既存の内容は変更されませんが、後者の呼び出しではそれらを 0% に設定するからです（オーバープリント設定に依存）。

4.5 DeviceN カラー

クックブック 完全なコードサンプルが、`starter_color` サンプルとクックブックの `color/devicen_color` トピックにあります。

DeviceN 色空間は、任意の数の名前付き色成分に対応しており、スポットカラーを一般化したものととらえることができます。そのインキ群は、プロセスカラーの集合から採ることもできますし（通常 CMYK ですが、それ以外の色空間も可）、任意のスポットカラー群とすることもできます。DeviceN カラーの応用は以下のとおりです：

- ▶ 4色よりも多くのプロセスカラーを使うことによって色域（印刷可能な色の集合）を拡げている印刷系もあります。たとえば、CMYK に加えてオレンジ・緑・紫に基づく 7色印刷系があります。
- ▶ CMYK プロセスカラー群の部分集合（たとえばシアンとマゼンタのみ）を内容とする DeviceN 色空間を用いると、CMYK カラーのオブジェクトを他の CMYK オブジェクト群にオーバープリントするべきときに有用です。
- ▶ DeviceN カラーを用いて、スポットカラーとプロセスカラーの間のシェーディング（なめらかな色遷移）を構築したり、複数のスポットカラーの間のシェーディングを構築したりすることも可能です。
- ▶ パッケージ印刷では、伝統的なスポットカラー群を使用できないので、n色を使用することが多いです。追加のカラーチャンネルを用いて、目に見える色には直接関係しない情報を記録することも行われています。たとえばワニスやダイラインなどです。

`PDF_create_devicen()` メソッドは、DeviceN 色空間ハンドルを返します。これを使って、`fillcolor/strokecolor` などの色関連オプションを用いて描画操作を行うこともできますし、`PDF_shading()` を用いてシェーディングを構築することもできますし、`PDF_load_image()` の `colorize` オプションを用いてラスター画像に着色することもできます。DeviceN カラーには濃淡をつけることもできます。すなわち、そのそれぞれのカラーチャンネルに対して、各カラーチャンネルの濃さを 0 = 色なし = 白から 1 = 最大濃度までの範囲で指定する割合値を持たせて使用することが可能です。DeviceN 色空間はつねに減法的です（RGB など加法的な色空間において 0 = 黒となるのと異なります）。

`PDF_create_devicen()` には、インキ名のリストと、代替カラー名と、濃度変換関数を実装した PostScript コードが必須です。この変換関数は、DeviceN 色空間の N 個のカラー値を、代替色空間における対応するカラー値群へ変換するものである必要があります。この代替カラー値群は、出力デバイスがその DeviceN 色空間に対応していない場合（モニター上など）に表現に用いられます。以下に、CMYK プロセスカラーかスポットカラー群から構築された DeviceN 色空間に対する、Lab を代替色として使用する場合の、然るべき PostScript 変換関数を示します。これ以外の組み合わせに対して然るべき PostScript 変換関数を作るには工夫が必要です。

PDF 1.6 以上に対する出力を生成する際には、PDFlib は DeviceN 属性辞書を出力します。この辞書には、すべての既知のスポットカラーに対する `Colorants` エントリーが含まれています。これによって PDF ビューアーは確実に、その名前付きインキ群を個別に表現するための十分な情報を得られますし、そのインキ群をブレンドすることもできるようになります。この手法の利点は、Acrobat においてオーバープリントプレビュー設定にかかわらず色表示が正しくなることにあります。かつ、Acrobat において PostScript 変換関数がなくてもそのページを表現できるようになります。ただし、この変換関数はいずれにせよ、与えなければなりません。なぜならサードパーティービューアーや RIP は、DeviceN カラーを代替色空間で表現する際に PostScript 関数を頼るかもしれないからです。

PDF/X-4/5 モードにおいては、`PDF_create_devicen()` の前に、その DeviceN 色空間において使用されるすべてのカスタムスポットカラーについて、`PDF_makespotcolor()` を呼び出す必要があります。これは一般に画面表示を向上させますので、PDF/X-4-5 を生成しない場合であっても推奨されます。

NChannel 色空間 PDF 1.6 では、DeviceN 色空間の拡張として、NChannel というものが導入されました。NChannel 色空間は、PDF ビューアーや RIP が正確な色表現を行うことを助ける情報を追加で内容として持ちます。この情報は、その DeviceN 色空間における個々のインキに対する代替色空間と（結合された濃度変換関数だけでなく）、プロセス色空間と、成分名群を含みます。NChannel はさらに、印刷インキの混ぜ合わせ動作を記述した情報を含むことも可能ですが、これは目下 PDFlib では対応していません。NChannel 色空間では、そのインキ群に対する個別の代替色を持つ `Colorants` エントリが必須です。これは PDFlib によって自動的に生成されます。NChannel 色空間ではさらに、プロセス色空間の名前とそのインキ名群も必須です。これらについては `process` オプションと `colorspace · components` サブオプションで与える必要があります。

NChannel 色空間を作成するには、`PDF_create_devicen()` でオプション `subtype=nchannel` を用います。この設定の利点は、このサブタイプを持った DeviceN 色空間は Acrobat の「オーバープリントプレビュー」設定に依存しなくなることです。場合によっては、スクリーンプレビューを正しくするためにこの Acrobat 設定が必要になることがあるのですが、NChannel は自動的に Acrobat をこの表示モードへ切り替えます。

CMYK プロセスカラーに基づく DeviceN 色空間 以下の例においては、マゼンタとイエローのインキで構成された、CMYK カラーの部分集合を使い、かつ代替色空間として DeviceCMYK を使います。その変換関数はシンプルです。なぜなら CMYK 代替色空間のシアン・黒成分に対するゼロ値を補足しているだけだからです：

```
devicen = p.create_devicen(  
    "names={Magenta Yellow} alternate=devicecmym transform={{0 0 4 1 roll}}");  
p.set_graphics_option("fillcolor={devicen " + devicen + " 0.5 1}");
```

その色を設定する際には N 個のカラー値を与える必要があります（上の例では N=2）。上の例で作成した DeviceN 色空間は、マゼンタとイエローのチャンネルにのみ描画し、シアンと黒のチャンネルを変更しません。これは、フル CMYK カラーで `Cyan=Black=0` として描画を行うとシアンとブラックのチャンネルにおける既存の内容が消去される（オーバープリント設定に依存）のと異なります。

便宜のために、表 4.6 に、CMYK インキのすべての可能な部分集合に対する、CMYK を代替色空間とした場合の、DeviceN PostScript 変換関数を挙げます。

同様の技法を用いて、CMYK 画像チャンネルの部分集合のみを表現することも可能です。たとえば以下のコード断片は、黒チャンネルのみを使う、それ以外の 3 個のチャンネルを `None` へ置き換える DeviceN 色空間を作成します。その画像データ内には 4 個のカラーチャンネルがあるため、この PostScript 変換関数はシアンとマゼンタとイエローのチャンネルをゼロ値へ置き換えています（これは、表 4.6 の PostScript 関数が既存のカラー値群をゼロへ置き換えるのではなく足りない 0 値群を補っているのと異なります）：

```
devicen = p.create_devicen(  
    "names={None None None Black} " +  
    "alternate=devicecmym transform={{4 1 roll pop pop pop 0 0 0 -1 roll}}");  
optlist = "width=4000 height=3000 bpc=8 colorize=" + devicen;  
image = p.load_image("raw", filename, optlist);
```

表 4.6 CMYK プロセスインキの部分集合群に対する、alternate=devicecmyk の場合の DeviceN PostScript 変換関数

CMYK 部分集合に対する DeviceN インキ名	PostScript 変換関数	CMYK 部分集合に対する DeviceN インキ名	PostScript 変換関数
Cyan	{0 0 0}	Magenta Black	{0 3 1 roll 0 exch}
Magenta	{0 0 0 4 1 roll}	Yellow Black	{0 0 4 2 roll}
Yellow	{0 0 0 4 2 roll}	Cyan Magenta Yellow	{0}
Black	{0 0 0 4 3 roll}	Cyan Magenta Black	{0 exch}
Cyan Magenta	{0 0}	Cyan Yellow Black	{0 3 1 roll}
Cyan Yellow	{0 exch 0}	Magenta Yellow Black	{0 4 1 roll}
Cyan Black	{0 0 3 -1 roll}	Cyan Magenta Yellow Black	{ }
Magenta Yellow	{0 0 4 1 roll}		

クックブック 完全なコードサンプルがクックブックの color/colorize_image_with_DeviceN_color トピックにあります。

Lab 代替空間を持ったスポットカラー群に基づく DeviceN 色空間 複数のスポットカラーを DeviceN インキとして用いることは、スポットカラーの間のシェーディングの基礎を提供します。デバイス依存な代替色空間に対しては、然るべき PostScript 変換関数を実装するのはかなり大変です。なぜなら RGB や CMYK は非線形な特性を有するからです。Lab 色空間の色を混ぜ合わせるのははるかに容易です。なぜなら知覚的に一様だからです。ですので、各色の Lab 値の加重平均を計算するだけで色を組み合わせることができます。以下の PostScript コードは、2 個の Pantone カラーに対してこの方式を実装しています。その 2 色の Lab 代替値をコードの先頭で与える必要があります：

```
% N=2かつLab代替を持つDeviceNに対するPostScript変換関数。
% 濃度値群を重みとして用いてL/a/b値の加重平均を計算することにより
% Labカラーどうしを混ぜ合わせます
```

```
80 28 75          % 色1=PANTONE 123 UのLab値群
31.7647 0 -17     % 色2=PANTONE 289 UのLab値群
```

```
% L値どうしを混ぜ合わせ
7 index 6 index mul      % t1*L1
7 index 4 index mul      % t2*L2
add 9 1 roll             % ボトム: L = t1*L1 + t2*L2
```

```
% a値どうしを混ぜ合わせ
7 index 5 index mul      % t1*a1
7 index 3 index mul      % t2*a2
add 9 1 roll             % ボトム: a = t1*a1 + t2*a2
```

```
% b値どうしを混ぜ合わせ
7 index 4 index mul      % t1*b1
7 index 2 index mul      % t2*b2
add 9 1 roll             % ボトム: b = t1*b1 + t2*b2
```

```
% 2個の濃度値と2x3個の入力カラー値をポップ
pop pop pop pop pop pop pop
% 結果: 混ぜ合わされた色のLab値群
```


上記の PostScript 関数 *transformFunc2* を用いて、2 個の Pantone スポットカラーに基づいて DeviceN 色空間を作成するには、以下のようにします：

```
devicen = p.create_devicen(  
    "names={{PANTONE 123 U} {PANTONE 289 U}} alternate=lab " +  
    "transform={{" + transformFunc2 + "}}");
```

Lab 代替値を持った別々のスポットカラーに基づいてシェーディングが構築されているときには、PDFlib は自動的にそのような DeviceN 色空間を構築します。

クックブック さまざまなコードサンプルと、Lab に基づくもっと大きな N 値に対する PostScript 変換関数が、クックブックの color/devicen_color トピックにあります。

4.6 シェーディングとシェーディングパターン

クックブック 完全なコードサンプルが、`starter_color` サンプルとクックブックの `color/color_gradient` トピックにあります。

スムーズシェーディングは、カラーブレンドまたはグラデーションとも呼ばれ、同じ色空間にある複数の色の間の連続的な遷移を与えます。たとえば2個のRGBカラーの間や、1個のスポットカラーの2個の濃度の間などです。シンプルなシェーディングは、2個の色の間の遷移を定義します。1番目の色は、`PDF_shading()` の `startcolor` オプションかカレント塗り色から採られます。2番目の色は、`endcolor` オプションか `c1 · c2 · c3 · c4` 引数で与えられます。任意の数の中間色を持つシェーディングを作成することもでき、そのためには `startcolor/endcolor` でなく `stopcolors` オプションを用います。

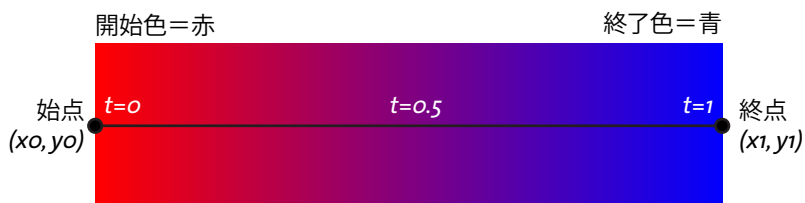
そのシェーディングカラーは変数 t によって制御されます。 t は開始色における0から終了色における1まで線形的に変化します。この線形的な遷移を指數的な遷移へ変えることもでき、そのためには `N` オプションを指数として用います。RGB・CMYK 色空間は知覚的に一様ではありませんので、その得られるブレンドは意図どおり滑らかな見えな場合があります。Lab 色空間の開始・終了色を用いてブレンドを指定すれば、より滑らかな色遷移を得られる可能性があります。PDFlib ではシェーディングに2種類の形状を使用できます：

- ▶ 直線状シェーディング (`type=axial`) : 始点と終点の間の直線に沿って色が変化します (図 4.2 参照)。そのシェーディングカラーは、始点 (x_0, y_0) から終点 (x_1, y_1) まで線形的に変化します。シェーディングを、始点・終点の向こうの境界色を用いて拡張することも可能で、そのためには `extend0 · extend1` オプションを用います。
- ▶ 放射状シェーディング (`type=radial`) : 2個の円の間で色が変化します (図 4.3 参照)。放射状シェーディングを利用すると、球の3次元風な視覚表現を作れます。そのシェーディングカラーは、中心 (x_0, y_0) と半径 r_0 を持つ開始円から、中心 (x_1, y_1) と半径 r_1 を持つ終了円まで変化します。片方の円を縮めて点にすることも可能です。

`PDF_shading()` は、シェーディングオブジェクトへのハンドルを返します。このハンドルの使い道は2つあります：

- ▶ `PDF_shfill()` を用いて領域を直接塗る。塗りたいオブジェクトの形状がシェーディングの形状と同じである場合にはこの方式が良いでしょう。このメソッドは、その名前に反して、オブジェクトの内側を塗るだけでなく、外側にも影響します。この動作は、`PDF_clip()` を用いて変えることもできます。
- ▶ もっと複雑なオブジェクトを塗るために、シェーディングパターン (タイリングパターンと混同しないでください) を定義して使用する。具体的には、`PDF_shading_pattern()` を呼び出すことによってシェーディングに基づいてパターンを作成し、このパターンを使用して任意のオブジェクトを塗ったり描線したりします。

図 4.2
2色間の直線状シェーディングの主なパラメーター



2 個のプロセッサカラーの間のシェーディング 以下のコードは、RGB 色空間における赤から青までの直線状シェーディングを作成し、それを使って円を塗ります：

```
sh = p.shading("axial", 100, 100, 500, 500, 0, 0, 0, 0, "startcolor=red endcolor=blue");
shp = p.shading_pattern(sh, "");
p.set_graphics_option("fillcolor={pattern " + shp + "}");

p.circle(300, 300, 200);
p.fill();
```

複数のスポットカラーの間のシェーディング シェーディングを使って作成できるのは、同一の色空間に属する色との遷移だけですので、開始色と終了色として、任意のスポットカラー群を使ったり、1 個のスポットカラーと 1 個のプロセッサカラーを使ったりすることはできません。ただし、Lab 代替値を持ったスポットカラー群に対しては、96 ページ「Lab 代替空間を持ったスポットカラー群に基づく DeviceN 色空間」で示しているのと同様の適切な DeviceN 色空間を用いてシェーディングを実現できます。特定の条件が満たされていれば（PDFlib API リファレンス参照）、ストップカラーとしてのスポットカラー群に対してこのような DeviceN 色空間を PDFlib は自動生成します。以下のコード断片は、2 個のスポットカラーの間のシェーディングを作成します：

```
sh = p.shading("axial", 100, 100, 500, 500, 0, 0, 0, 0,
  "stopcolors= 0% {spotname {PANTONE 123 U} 1} 100% {spotname {PANTONE 289 U} 1}");
shp = p.shading_pattern(sh, "");
p.set_graphics_option("fillcolor={pattern " + shp + "}");

p.circle(300, 300, 200);
p.fill();
```

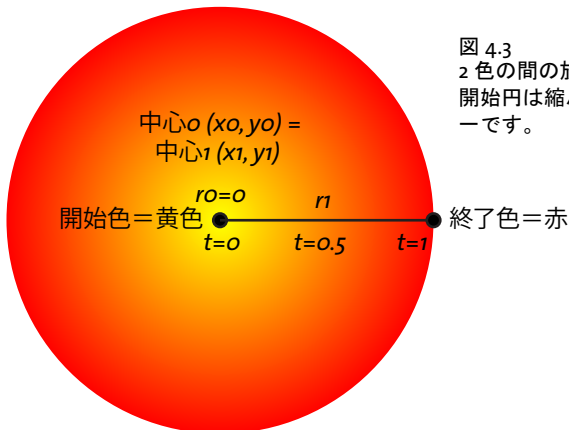


図 4.3
2 色間の放射状シェーディングの主なパラメーター。
開始円は縮んで点になっています。2 つの円の中心は同一です。

4.7 タイリングパターン

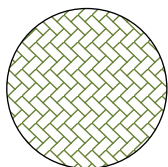
クックブック 完全なコードサンプルが、`starter_color` サンプルとクックブックの `graphics/tiling_pattern`・`images/tiling_pattern` トピックにあります。

タイリングパターンは、任意の数の塗り操作を1つの実体にまとめたものによって定義されます。このグループは任意の他の物の塗りや描線に用いることができ、塗りの場合は全域内に、描線の場合はパス上にグループがコピー（縦横に並ぶ）されます。パターンを使った作業では次の手順を踏みます：

- ▶ まず、`PDF_begin_pattern_ext()` と `PDF_end_pattern()` の間で、描画オペレーター群を用いてパターンを定義しなければなりません。パターンの定義にはたいいていの画像オペレーターが利用できます。
- ▶ `PDF_begin_pattern_ext()` によって返されたパターンハンドルは、`PDF_set_graphics_option()` または `PDF_setcolor()` でオプション `fillcolor/strokecolor` を用いて、パターンをカレントカラーとして設定するために用いることができます。

`PDF_begin_pattern_ext()` の `painttype` オプションによって、パターン定義がそれ自身の色指定群を含むことができるかどうかが決まります。`painttype=colored` の場合、パターン定義はそれ自身の色指定を含まなければならず、つねに見た目が同じになります。`painttype=uncolored` の場合、パターン定義は色指定を一切含んでいてはなりません。そのパターンが塗りや描線に使われるときには、カレントの塗りや描線の色が適用されます。

図 4.4
タイリングパターンを使って円とテキストを塗る。さらに強調のためにテキストの輪郭を描線しています。



Text filled with tiling pattern

4.8 透過ブレンドモード

クックブック コードサンプルがクックブックの color/blendmode トピックにあります。

ブレンドモードは、透過イメージングモデル（各オブジェクトがその背後にあるすべてのオブジェクトを完全に覆い隠す不透明イメージングモデルではなく）において、オブジェクトの色を背景とどのように混ぜ合わせるかを制御します。背景色は、複数の他のオブジェクトをブレンドすることによって作成されている可能性もありますので、これを**バックドロップ**と呼びます。描画されるオブジェクトを**ソース**と呼びます。ソースとバックドロップの色は、何らかのブレンド機能を用いて互いに混ぜ合わされます。表 4.7 に挙げるブレンドモードは、さまざまな芸術的效果を生み出し、また、特殊な用途に利用することもできます（104 ページ「4.9.1 ブレンドモードを用いて色を変える」参照）。利用可能なブレンドモードは以下のように分類できます：

- ▶ デフォルトのブレンドモード **Normal**：ソースを完全に不透明にします。すなわち、ソースオブジェクトが完全に背景を置き換えます。ブレンドモードを **Normal** 以外に設定すると、いかなる不透過設定や画像透過にもよらず、ソースオブジェクトは暗黙的に透過になります。
- ▶ ブレンドモード **Darken**・**Multiply**・**ColorBurn**：暗くする効果を生み出します。
- ▶ ブレンドモード **Lighten**・**Screen**・**ColorDodge**：明るくする効果を生み出します。
- ▶ ブレンドモード **Overlay**・**SoftLight**・**HardLight**：明るい領域と暗い領域を加減することによってコントラストを強くします。
- ▶ 差分ブレンドモード **Difference**・**Exclusion**：カラー値の差をとります。白いオブジェクトを白バックドロップの上に描画して結果が白にならないのはこれらのモードだけです。
- ▶ ブレンドモード **Hue**・**Saturation**・**Color**・**Luminosity**：HSL カラー表現の次元群に対する効果によって定義されます。HSL モデルにおいて、色相 (hue) は知覚される色を記述し（赤と緑は色相が異なる）、彩度 (saturation) は色がどのくらいカラフルに見えるか（その色にグレーがどのくらい混ぜられているか）を記述し、輝度 (luminosity) は色がどのくらい暗く、または明るく見えるかを記述します（白が最大輝度）。

表 4.7 ブレンドモードと、ソースオブジェクトを背景オブジェクト（バックドロップ）にブレンドさせる際の効果

ブレンドモード ISO 32000-2 に従った説明

ブレンド効果のないブレンドモード

None PDFlib 独自のブレンドモード。ブレンドモード指定のない gstate を作成します。外側の gstate で設定されているブレンドモードを確実に効かせます。

Normal ソース色を採り、バックドロップを無視します。

暗くするブレンドモード

Darken¹ バックドロップとソースのうち暗いほうの色を採ります。ソースのほうが暗い場所ではバックドロップはソースへ置き換えられます。そうでない場合には変更されません。

Multiply¹ バックドロップとソースのカラー値を乗算します。結果の色は少なくともこの 2 色のどちらかと同じ暗さになります。加法的な色を扱う場合には、任意の色を黒と乗算すると結果は黒になり、白と乗算すると元の色は変更されません。減法的な色の場合には、その色空間のすべてのインキに対して使用されている濃度の最大値が、加法空間における黒と同じ役目をします。いくつものオブジェクトが重なって重なり合っている場合には、それらが黒でも白でもなければ、だんだん暗い色が描画されます。

ColorBurn ソースカラーを反映してバックドロップカラーを暗くします。白で描画した場合には変化ありません。

表 4.7 ブレンドモードと、ソースオブジェクトを背景オブジェクト（バックドロップ）にブレンドさせる際の効果

ブレンドモード ISO 32000-2 に従った説明

明るくするブレンドモード

Lighten¹ バックドロップとソースのうち明るいほうの色を採ります。ソースのほうが明るい場所ではバックドロップはソースへ置き換えられます。そうでない場合には変更されません。

Screen¹ バックドロップとソースのカラー値の補数を乗算し、結果の補数をとります。結果の色は少なくともこの2色のどちらかと同じ明るさになります。加法的な色を扱う場合には、任意の色を白とスクリーンすると結果は白になり、黒とスクリーンすると元の色は変更されません。減法的な色の場合には、その色空間のすべてのインキに対して使用されている濃度の最大値が、加法空間における黒と同じ役目をします。複数のスライド写真を同時に1つのスクリーンへ映し出したような効果になります。

ColorDodge ソースカラーを反映してバックドロップカラーを明るくします。黒で描画した場合には変化ありません。

コントラストを強めるブレンドモード

HardLight ソースカラー値が0.5より小さければ2色を乗算し、0.5より大きければスクリーンします。バックドロップにきついスポットライトをあてたような効果になります。

SoftLight ソースカラー値が0.5より小さければ2色のうち暗いほうの色を採り、0.5より大きければ明るいほうの色を採ります。バックドロップにやわらかいスポットライトをあてたような効果になります。

Overlay バックドロップカラー値に応じて2色を乗算かスクリーンします。ソースカラーが、バックドロップのハイライトと影を温存しつつその上にかぶさります。バックドロップカラーは置き換えられるのではなく、バックドロップの明るさや暗さを反映するようにソースカラーと混ぜ合わされます。

差分ブレンドモード

Difference^{1,2} 2色のうち明るいほうの色から暗いほうの色を減算：白で描画するとバックドロップカラーが反転されます。黒で描画すると何も変化しません。減法的な色の場合には、その色空間のすべてのインキに対する濃度の最大値が、加法空間における黒と同じ役目をします。

Exclusion^{1,2} Difference モードと似た効果になりますが、コントラストはより弱くなります。白で描画するとバックドロップカラーが反転されます。黒で描画すると何も変化しません。減法的な色の場合には、その色空間のすべてのインキに対する濃度の最大値が、加法空間における黒と同じ役目をします。

HSL ブレンドモード

Hue² ソースカラーの色相を持ち、バックドロップカラーの彩度と輝度を持つ色を作成します。

Saturation² ソースカラーの彩度を持ち、バックドロップカラーの色相と輝度を持つ色を作成します。バックドロップの純粋な灰色（彩度なし）の部分にこのモードで描画しても何も変化ありません。

Color² ソースカラーの色相と彩度を持ち、バックドロップカラーの輝度を持つ色を作成します。これはバックドロップのグレーレベルを温存しますので、白黒の画像に色をつけたり、カラー画像に濃度をつけたりするのに有用です。

Luminosity² ソースカラーの輝度を持ち、バックドロップカラーの色相と彩度を持つ色を作成します。これはColor モードと同じ効果を生み出しますが、ただしソースとバックドロップが入れ替わっています。

1. このモードは対称です。すなわち、ソースカラーとバックドロップカラーを入れ替えてもブレンド操作の結果は変わりません。

2. スポットカラーに対してはこのモードは何ら影響を与えず、ブレンドモード Normal がかわりに使われます。

ブレンド色空間 PDF ビューアーは、すべての透過計算をブレンド色空間において行います。この色空間は重要な役割を果たします。なぜなら、どの色空間を選択するかによってブレンドの結果が変わってくるからです。これは以下のとおり決定されます：

- ▶ *transparencygroup* ページオプションの *colorspace* サブオプションがある場合には、そこで指定されている色空間。
- ▶ ない場合には、PDF/X または PDF/A 出力インテント ICC プロファイルがもしあるなら、それが使用されます。PDF/X では出力インテントは必ずあります。PDF/A 文書が出力インテントを含んでいない場合には、それは *transparencygroup* ページオプションの *colorspace* サブオプションを指定する必要があります。
- ▶ ないなら、出力デバイス（ソフトプルーフ／出力プレビューの場合には、シミュレートされた出力デバイス）のネイティブ色空間が使用されます。ブレンド色空間を明示的に指定することによってこの状況を避けることを推奨します。なぜなら、そうでないと透過オブジェクトの視覚表現が出力デバイスに依存してしまうからです。

以下のコード断片は、DeviceRGB をブレンド色空間として指定し、ブレンドモード *Multiply* を指定しています：

```
p.begin_page_ext(0, 0, "width=a4.width height=a4.height " +
                  "transparencygroup={colorspace=DeviceRGB}");

gstate = p.create_gstate("blendmode=Multiply");
p.set_gstate(gstate);
```

複数のブレンドモード PDF では、いつきに 1 つのブレンドモードしかグラフィックステート内で有効にすることができません。複数のブレンドモードの効果を適用したい場合には、すでにブレンドされたバックドロップとオブジェクトの上に重ねて別のオブジェクトを描画することができます。たとえば、何かほかのブレンドモードの結果を色反転させたい場合には、まずバックドロップとオブジェクトを第一のブレンドモードでブレンドしたのち、ブレンドモードを *Difference* へ切り替えて、そして最初のブレンド操作の結果の上に重ねて白い長方形を描画します。

4.9 オブジェクトの色を変更

PDFlib では、オブジェクトの色を変更する手段をいくつか用意しています。これによって、ページ上に配置されている取り込み PDF ページ、ラスター画像、SVG グラフィック、任意のテキスト・ベクトル要素の色を変更することができます：

- ▶ ラスター画像には着色することができます。197 ページ「8.1.5 画像にスポットカラーか DeviceN カラーで着色」参照。
- ▶ 任意のオブジェクトの色を、ブレンドモードを用いて変更できます。
- ▶ 輝度ソフトマスクを持った gstate を用いて、任意のオブジェクトの色をつけることができます。

4.9.1 ブレンドモードを用いて色を変える

クックブック この項で述べるすべての効果のためのコードサンプルがクックブックの `color/blendmode_effects` トピックにあります。

この項ではブレンドモードの有用な応用をいくつか紹介します。利用できるすべてのブレンドモードの説明は 101 ページ「4.8 透過ブレンドモード」にあります。すべての例において、オブジェクト（取り込み画像・PDF ページ・SVG グラフィックなど）の色が、何らかの形で変更されます。

すべての例は、以下の示す基本的なコード断片を用いて実装できます。このコード断片は、ページに対して透過グループ色空間を指定することによってデバイス独自の視覚表現を防いだうえで、何か特定のブレンドモードを設定しています。後続するオブジェクト群が、ここで指定したブレンドモードによって影響されることを防ぐために、このシーケンスは、`save/restore` ペアで囲むべきです：

```
p.begin_page_ext(width, height, "transparencygroup={colorspace=DeviceCMYK}");
```

```
// ラスター画像か取り込みPDFページかSVGグラフィックを配置  
p.fit_image(image, 200, 150, optlist);
```

```
// 望む色とブレンドすることにより既存内容を着色または脱色または色反転  
p.save();  
    // 望むブレンドモードを持ったgstateを作成  
    gs_blendmode = p.create_gstate("blendmode=Color");  
    p.set_graphics_option("fillcolor=red gstate=" + gs_blendmode);  
    p.rect(0, 0, width, height);  
    p.fill();  
p.restore();
```

オブジェクトに任意の色で着色 ブレンドモード `Color` を使うと、オブジェクトに何か新しい色を着色できます。そのオブジェクトの、知覚されるグレーレベルが、追加される色の濃度値を制御し、かつ、そのオブジェクトの元の色は無視されます。オブジェクトに着色するには、ブレンドモード `Color` を適用し、色付きの長方形（または他の形）を、そのオブジェクトの上に重ねて配置します。透過オブジェクトに着色するには、配置オブジェクトの下に白い領域を配置することを推奨します。

逆にソースオブジェクトの白い領域を通してバックドロップカラーが見えるように（暗い領域は変更されないように）したい場合には、ブレンドモード `Multiply` を使用します。

オブジェクトを脱色または色反転 場合によっては、オブジェクトを脱色（彩度をなくす。褪せさせる）したい場合もあるでしょう。すなわち、もともと色が付いていたオブ

ジェクトからグレースケール出力を作成することです。その際に、その各部分のグレーレベルは、そのオブジェクトの各部分のもともとの色の、知覚される明るさ（輝度）に対応するようにすることです。これを実現するには、ブレンドモード *Color* を用いて、そのオブジェクトの上に白い領域を重ねます。

ラスター画像の色は *invert* 画像オプションを用いて反転できます。任意のオブジェクトの色を反転することも可能です。そのためには、ブレンドモード *Difference* を用いて、そのオブジェクトの上に白を重ねます。

4.9.2 ソフトマスクを用いて色を変える

クックブック コードサンプルがクックブックの `color/softmask_effects` トピックにあります。

画像アルファチャンネルは、一般化して任意のオブジェクトと色空間で利用できます。そのためには、グラフィックステート (`gstate`) 内の輝度ソフトマスクを援用します。このソフトマスクは、取り込み PDF ページや SVG グラフィックなど任意の内容を持ったテンプレートをベースとして構築されます。このテンプレート内容の輝度、すなわち知覚されるグレーレベルまたは明度によって、その後には描かれるオブジェクト群の可視性が決定されます：テンプレート内の明るい領域は透明になり（すなわち、描かれるオブジェクト群が見える）、暗い領域はオブジェクトを見えなくします。この技法を実現するには以下の手順に従います：

- ▶ テンプレートを、*transparencygroup* オプションとサブオプション *colorspace* を用いて作成します。このテンプレートの中で、任意の描画操作を行なっていきます。
- ▶ このテンプレートをベースとして、`gstate` を、*softmask* オプションとサブオプション *type=luminosity* を用いて作成します。ソフトマスクを適用した時、ソフトマスクテンプレートの明るい（白い）領域はカレント色で着色され、一方、暗い（黒い）領域に色がつきません。*softmask* オプションの *invert* サブオプションを使うと、暗い色と明るい色の効果を反転させることができます。すなわち、暗い領域と明るい領域が維持されます。デフォルトではソフトマスクの背景は不透明で初期化されます。これは黒ベタと同等です。これは、サブオプション *backdropcolor=transparent* を用いて変えることもできます。これは白ベタと同等です。なお、*invert* サブオプションは *backdropcolor* の効果についても反転させます。
- ▶ この `gstate` を設定します。save/restore シーケンスで囲むことが多いでしょう。そうすれば、このソフトマスクの効果をその中だけにとどめることができます。この `gstate` 内のソフトマスクテンプレートの幾何情報はカレント座標系に依存します。たとえば、ソフトマスクの位置を制御するには、`gstate` を設定する前に平行移動を適用します。以上が終わったら、ページ上でマスクされる内容を描きます。

以下のコード断片は、このシーケンスを実装し、それを適用することによって、取り込み SVG グラフィックファイルに着色しています：

```
// ソフトマスクの定義に使うテンプレートを作成
tpl = p.begin_template_ext(595, 842, "transparencygroup={colorspace=devicecmymk}");
// 任意の内容を配置。例：SVGグラフィック
p.fit_graphics(graphics, 0, 0, "boxsize={595 842} position={center} fitmethod=meet");

p.end_template_ext(0, 0);

// このテンプレートをベースにした、反転したソフトマスクを用いたgstateを作成
// 「invert」オプションで黒を透過させています。すなわち暗い領域と明るい領域を維持
gstate_softmask = p.create_gstate(
    "softmask={type=luminosity template=" + tpl + " backdropcolor=transparent }");
```

```
// ソフトマスクを用いたgstateを有効にしたうえで、色を適用
// 結果: SVGグラフィックが赤く着色されます
p.save();
  p.set_graphics_option("fillcolor=red gstate" + gstate_softmask);
  p.rect(0, 0, 595, 842);
  p.fill();
p.restore();
```

複数のソフトマスク PDFは、1つの合成操作について1つまでのソフトマスクにのみ対応しています。たとえば、画像が自分のアルファチャンネル(=ソフトマスク)を持っている場合には、グラフィックステートで与えられたソフトマスクは何の効力も及ぼしません。このPDFの制約を回避するには、テンプレート内に内容を生成して、このテンプレートをページ上に配置する必要があります。これによって事実上、PDFビューアーに強制的に別々の合成操作を適用させ、ソフトマスクが使用可能になります。

4.10 オーバープリント制御

クックブック コードサンプルがクックブックの color/overprint トピックにあります。

デフォルトでは、ある特定のカラーチャンネルに描画を行うと、他のカラーチャンネル群の対応する領域は置き換わります。たとえば、CMYK デバイス上の色 0/0/1/0、すなわちイエローは、そのページ上の同じ場所にそれまでに描画されていたオブジェクト群の既存のシアン・マゼンタ・黒チャンネルを消去します。この動作を変えるにはオーバープリントという処理を用います。この処理は、印刷デバイス上における色インキの混合を制御します。オーバープリントを利用して、黒を他の暗い色の上に重ねて印刷することにより、「リッチブラック」にすることもできます。

PDF のオーバープリントフラグは CMYK 印刷を対象としています。このフラグは、ある特定のチャンネルへの描画が他のチャンネル群の既存の内容に影響するかどうかを制御します。たとえば、シアンをイエローの上に重ねて印刷すると、その結果は、オーバープリントありであれば緑になります（シアンとイエローが混合される）。逆にオーバープリントなしで同じことをすると、その結果は、最後に印刷した色であるシアンとなります。なぜなら、純粋なシアンにはイエローの成分が一切ないので、0% イエローが既存のイエローの内容を置き換えるからです。

塗り・描線操作に対するオーバープリント設定 オーバープリント動作は、描線操作とそれ以外の操作とで別々に制御できます。そのためにはグラフィック書式オプション *overprintstroke*（描線操作に対して）または *overprintfill*（描線以外のすべての操作に対して。画像の配置に対しても）を用います。どちらのオプションもデフォルトは *false* です：

- ▶ *overprintfill=false* か *overprintstroke=false* の場合には、塗りまたは描線は、どのカラーチャンネルにおいても、指定されていないインキの、対応する領域を置き換えます：前面の色が勝ちます。
- ▶ これらのオプションが *true* の場合、かつ出力デバイスがオーバープリントに対応している場合には、指定されていないインキにおけるそれまでのマーキング群に対して、変更は行われません。オーバープリントは通常、かなり暗い色に対して使用されます。

CMYK カラーに対するオーバープリントモード CMYK オブジェクトのオーバープリント動作は、さらに、*overprintmode* グラフィック書式オプションを用いて変更することもできます。このオプションは、*overprintfill/overprintstroke=true* の場合のゼロ CMYK 成分の動作を制御します：

- ▶ *overprintmode=0*（デフォルト）：それぞれの色成分が、それまでに配置されたマーク群を置き換えます（「前面の色が勝つ」）。言い換えれば、前面のオブジェクトの色のすべてのチャンネルが、下に重なっているオブジェクト群をヌキにします。
- ▶ *overprintmode=1*（「Illustrator のオーバープリントモード」と呼ばれることもあります）：濃度値 0 は、それまでに描画された色の、対応する成分を、0 に設定するのではなく、変更しません（「前面の濃度値 0 は無視される」）。言い換えれば、0 以外の値を持った前面チャンネルのみが、下に重なっているオブジェクト群をヌキにします。これは、DeviceN 色空間において、0 でない値を持った CMYK 成分のみで描画するのと同じことです。

つまり、CMYK 濃度値がわずかわるだけで（たとえば 0% か 1% か）、オーバープリントモードによって結果に大きな違いが生じる場合があります：オーバープリントモード 0 の場合にはその結果はそれぞれ 0% と 1% になります。オーバープリントモード 1 の場合には、その結果はそれぞれ背景色（それが何であれ）か 1% になります。オーバープリントモードは、その定義により、CMYK 以外の色空間においては何も影響を与えません。

オーバープリントによって影響を受けるのは DeviceCMYK 色空間の色だけです。DeviceN 色空間で Cyan・Magenta・Yellow・Black 成分のいずれかが使われていても、この CMYK オーバープリント規則は適用されません。

デバイス依存なオーバープリント表現 PDF ビューアーは、出力デバイスがオーバープリントに対応していない場合には、オーバープリント設定を無視しなければならないことになっていますので、オーバープリントオブジェクトを持った文書は本質的にポータブルではなく、意図したページ表現を実現するには、PDF ビューアーや RIP を特別に構成する必要が生じる場合があります。Acrobat が画面表示においてオーバープリント設定に従うかどうかは、「編集」→「環境設定」→「ページ表示」→「オーバープリントプレビューを使用」を通じて構成可能です。オーバープリント設定を使って作業しているときには、この環境設定を「常時」に設定することによって正しい画面表示が得られるようにすることを推奨します。さらに、Acrobat の「出力プレビュー」パネルには項目「オーバープリントをシミュレート」が設けられています。

ページ上で透過かブレンドモードが使用されている場合には、Acrobat のオーバープリントプレビューは、カレントのブレンド色空間が DeviceCMYK 色空間を使用するよう設定されている場合のみ、期待どおり動作します。そのように設定するには `PDF_begin_page_ext()` で以下のオプションを用います：

```
transparencygroup={colorspace=devicecmymk}
```

オーバープリントプレビュー対応のないサードパーティー PDF ビューアーは、期待と違う色を表示する場合があります。実際、シンプルな PDF ビューアーの多くはオーバープリントに対応していません。

デバイス独立なオーバープリント効果 上述のビューアー依存性により、オーバープリントの表現はもろくなる場合があります。オーバープリント効果をデバイス独立な方式で実現することもできます。そのためにはブレンドモード *Darken* を以下のように使用します：

```
gstate = p.create_gstate("blendmode=darken");  
p.set_gstate(gstate);
```

オーバープリントシミュレーションが有用となる状況としては、一つには、色分版の再合成が挙げられます。画像やページのシアン・マゼンタ・イエロー・黒分版を持っているとして、これらを 1 個のコンポジットな PDF ページへ再合成する必要がある場合に、*Darken* ブレンドモードを用います。

クックブック CMYK 分版を再合成しているコードサンプルが、クックブックの `color/recombine_color_channels` トピックにあります。

5 Unicode とレガシエンコーディング

この章では、Unicode やその他のエンコーディング方式に関する基礎的な情報を提供します。PDFlib におけるテキスト処理は Unicode 規格に大きく依存していますが、さまざまなレガシエンコーディングにも対応しています。

5.1 Unicode の重要な諸概念

キャラクターとグリフ テキストを扱う際には、以下の概念をはっきり区別することが大切です：

- ▶ **キャラクター**は、言語の中で情報を伝達する最小の単位です。代表的な例はラテンアルファベットの文字、中国語の表意文字、日本語の音節文字です。キャラクターは意味を持ちます：すなわちキャラクターは意味実体です。
- ▶ **グリフ**は、さまざまな視覚表現で、1 個ないし複数のキャラクターを表します。グリフは外見を持ちます：すなわちグリフは表現実体です。

キャラクターとグリフの間的一对一の対応は存在しません。たとえば合字は 1 つのグリフですが、2 つ以上のキャラクターを表現します。かと思えば、1 つのグリフが場面によって別々のキャラクターを表すこともあります（キャラクターには同じ形のものがあります。図 5.1 参照）。

BMP と PUA 以下の用語が Unicode ベースの環境では頻繁に登場します：

- ▶ **基本多言語面 (Basic Multilingual Plane = BMP)** : Unicode の範囲 U+0000 ~ U+FFFF 内のコード点から成ります。Unicode 規格はこのほかにも多くのコード点を、追加面群、すなわち範囲 U+10000 ~ U+10FFFF 内に含んでいます。
- ▶ **私用領域 (Private Use Area = PUA)** : 私用のために予約されている複数の Unicode 領域から成ります。PUA のコード点は一般的なやりとりには利用できません。なぜなら、Unicode 規格ではこの領域の中にかなるキャラクターをも指定していないからです。基本多言語面は PUA 領域 U+E000 ~ U+F8FF を含んでいます。第 15 面 (U+F0000 ~ U+FFFFFD) と第 16 面 (U+100000 ~ U+10FFFFD) は私用のためにすっきり予約されています。

キャラクタ

グリフ

U+0067 LATIN SMALL LETTER G

U+0066 LATIN SMALL LETTER F +
U+0069 LATIN SMALL LETTER I

U+2126 OHM SIGN または
U+03A9 GREEK CAPITAL LETTER OMEGA

U+2167 ROMAN NUMERAL EIGHT または
U+0056 V U+0049 I U+0049 I

図 5.1
グリフとキャラクターの関係

Unicode のエンコーディング形式 (UTF 形式) Unicode標準は各キャラクターに数(コード点)を割り当てています。この数をコンピューター処理で使うには、何らかの方式で表現しなければなりません。Unicode 標準ではこれをエンコーディング形式と呼びます(旧称:変換形式)。この用語はフォントのエンコーディングと混同してはいけません。Unicode は以下のエンコーディング形式を定義しています:

- ▶ **UTF-8**: これは可変幅の形式で、コード点は1~4バイトで表されます。範囲 U+0000 ~ U+007F の ASCII キャラクターは範囲 00 ~ 7F のシングルバイトで表されます。範囲 U+00A0 ~ U+00FF の Latin-1 キャラクターは2バイトで表され、その第一バイトはつねに 0xC2 か 0xC3 になります(これらの値は Latin-1 で \acute{A} と \tilde{A} を表します)。
- ▶ **UTF-16**: 基本多言語面 (BMP) のコード点は1つの16ビット値で表されます。補助多言語面のコード点、すなわち範囲 U+10000 ~ U+10FFFF のコード点は16ビット値のペアで表されます。このようなペアをサロゲートペアといいます。1つのサロゲート値は、範囲 D800 ~ DBFF の高位サロゲート値1つと範囲 DC00 ~ DFFF の低位サロゲート値1つから成っています。高位と低位のサロゲート値はサロゲートペアの中のみ現れ、他の場面で使われることはありません。
- ▶ **UTF-32**: 各コード点は1つの32ビット値で表されます。

Unicode のエンコーディング体系とバイト順序マーク (BOM) コンピューターアーキテクチャーは種類によって、バイトの並べ方が違います。すなわちバイトがより大きな値(16ビットや32ビット)を構成するとき、最上位バイトを最初に格納する方式(ビッグエンディアン)と、最下位バイトを最初に格納する方式(リトルエンディアン)があります。ビッグエンディアンアーキテクチャーの代表例は PowerPC であり、一方 x86/x64 アーキテクチャーはリトルエンディアンです。UTF-8 と UTF-16 はシングルバイトより大きな値を用いていますので、やはりバイトの並べ方を論じる必要が出てきます。エンコーディング体系は(上述のエンコーディング形式とは違うので注意)、エンコーディング形式に加えてバイト順序を指定します。たとえば UTF-16BE は、UTF-16 でバイト順序はビッグエンディアンという意味です。バイト順序があらかじめわからないときは、それを指定する手段としてコード点 U+FEFF があります。これをバイト順序マーク (BOM) といいます。BOM は UTF-8 では不要ですが、存在していてもよく、それを利用してバイトストリームを UTF-8 と同定することもできます。さまざまなエンコーディング形式に対する BOM の表し方を表 5.1 に示します。

表 5.1 さまざまな Unicode エンコーディング形式に対するバイト順序マーク

エンコーディング形式	バイト順序マーク (16 進)	WinAnsi における視覚表現 ¹
UTF-8	EF BB BF	ï»¿
UTF-16 ビッグエンディアン	FE FF	þÿ
UTF-16 リトルエンディアン	FF FE	ÿþ
UTF-32 ビッグエンディアン	00 00 FE FF	■ ■ þÿ
UTF-32 リトルエンディアン	FF FE 00 00	ÿþ ■ ■

1. 四角 ■ は null バイトを意味します。

5.2 Unicode 対応言語バインディング

PDFlib API の機能のなかには、使用する言語バインディングが Unicode 対応かどうかによって変化するものがあります。この概念をこの節と次の節で解説します。

5.2.1 ネイティブ Unicode 文字列のある言語バインディング

プログラミング言語が Unicode 文字列をネイティブに用いている場合に、そのバインディングを Unicode 対応と呼ぶことにします。以下の PDFlib 言語バインディングは Unicode 対応です：

- ▶ C++
- ▶ .NET
- ▶ Java
- ▶ Objective-C
- ▶ Python
- ▶ RPG

こうした環境での文字列処理は単純です：文字列はすべて、ネイティブな UTF-16 形式の Unicode 文字列として PDFlib カーネルに与えられます。こうした言語のラッパーは、クライアントから与えられる Unicode 文字列を正しく取り扱うことができ、また、いくつかの PDFlib のオプションを自動的に設定します。その際には以下の結果が生じます：

- ▶ クライアントが与える文字列はすべて、Unicode エンコーディングかつ UTF-16 形式で PDFlib にもたらされます。
- ▶ API 解説内のいろいろな文字列種別（内容文字列・ハイパーテキスト文字列・名前文字列）どうしの違いは意味を持ちません。オプション `textformat`・`hypertextformat`・`hypertextencoding` は不必要であり、かつ許されません。テキストフローオプションは強制的に `true` になります。
- ▶ ページの内容には `unicode` エンコーディングを用いることが、エンコーディングを Unicode 対応言語で取り扱ううえでもっとも簡単な方法です。ただし 8 ビットエンコーディングや、記号フォントのシングルバイトテキストも、使いたければ使うことができます。

要するに基本的には、クライアントはネイティブ Unicode 文字列を PDFlib API メソッドに与えることができ、その際に別途構成は必要ないということです。

5.2.2 UTF-8 対応のある言語バインディング

ネイティブに Unicode 文字列データ型を持たないプログラミング言語であっても UTF-8 形式の Unicode 文字列を扱うことはできます。API の文字列、すなわち名前文字列・内容文字列・ハイパーテキスト文字列・オプションリストはすべて、BOM 付きか BOM なしの UTF-8 形式を受け付けます。

Unicode 以外のエンコーディングの文字列を扱う必要がある場合には、それを PDFlib へ渡す前に UTF-8 の Unicode へ変換する必要があります。これは `PDF_convert_to_unicode()` が言語独自のメソッドで実現できます。

5.2.3 デフォルト UTF-8 対応のある言語バインディング

以下の PDFlib 言語バインディングはデフォルトで UTF-8 を通じて Unicode 対応です：

- ▶ C

- ▶ Perl
- ▶ PHP
- ▶ Ruby

C バインディングのオプションな UTF-8 対応 デフォルトでは PDFlib の C 言語バインディングは Unicode 対応ではありませんが (後述)、UTF-8 文字列とともに使用するよう構成することは可能です :

```
PDF_set_option(p, "stringformat=utf8");
```

BOM 付きの UTF-8 形式は C のユーザーに利点があります。それは PDFlib は自動的にそのような文字列を BOM を通じて認識するからです。これによって、フォントを *encoding=unicode* で読み込んだり、ハイパーテキスト文字列を *hypertextencoding=unicode* で、名前文字列を *usehypertextencoding=true* で扱ったりすることが可能になりますので、完全な Unicode ワークフローが実現されます。

ただし C インターフェイスは、関数の引数として受け渡された文字列について、*length* 引数に 0 より大きな値が与えられた場合には、UTF-16 として解釈します。

5.3 非 Unicode 対応言語バインディング

5.3.1 ネイティブな Unicode 文字列のない言語バインディング

レガシー文字列対応のある言語バインディング 以下の PDFlib の言語バインディングはデフォルトで UTF-8 を使用しますが (上述)、*stringformat=legacy* を設定することによって、レガシーな 8 ビットエンコーディングの文字列とともに使用するよう構成することも可能です:

- ▶ Perl
- ▶ PHP
- ▶ Ruby

レガシーな 8 ビットエンコーディングの文字列をこれらの言語バインディングで使用するには、文字列の形式の以下のとおり設定する必要があります:

```
p.set_option("stringformat=legacy");
```

C バインディングのレガシーな文字列へのデフォルト対応 C バインディングは、デフォルト設定 *stringformat=legacy* を通じて 8 ビットエンコーディングの文字列とともに動作します。

ただし C インターフェイスは、関数の引数として受け渡された文字列について、*length* 引数に 0 より大きな値が与えられた場合には、UTF-16 として解釈します。

5.3.2 Unicode 処理と各種文字列

Unicode 文字列は非 Unicode 対応言語でも使用できますが、文字列の処理は、複雑になるとともに、その文字列をどう使うつもりかによって異なってきます。PDFlib API は、**内容文字列**・**ハイパーテキスト文字列**・**名前文字列** (これらの呼称は歴史的な誤称です) という文字列種別を使用します。引数とオプションは、PDFlib API リファレンス内で、これらの種別のいずれか一つとして示されています。これらの文字列種別の取り扱いを、表 5.2 にまとめるとともに、以下に詳しく解説します。

表 5.2 さざまな文字列種別に対する文字列処理の概要

文字列種別	サンプル引数・オプション	関連オプション / 解釈
内容文字列	PDF_fit_textline() ・ PDF_add_textflow() の text 引数。	textformat encoding
ハイパーテキスト文字列	▶ PDF_add_table_cell() の fieldname オプション ▶ PDF_define_layer() の name オプション ▶ PDF_create_action() の destname オプション ▶ PDF_create_bookmark() の text 引数	hypertextformat hypertextencoding
名前文字列	▶ PDF_begin_document() ・ PDF_create_pvf() の filename 引数 ▶ PDF_load_font() の fontname 引数 ▶ PDF_load_iccprofile() の profilename 引数	usehypertextencoding, filenamehandling
オプションリスト内の文字列		BOM 付き : UTF-8 BOM なし : 文字列種別による

内容文字列 内容文字列は、特定のフォントに対して選んだエンコーディングに従ってページ内容を作成するために用いられます。PDFlib API リファレンスの中で、ページ内容関数における `text` という名前のメソッド引数はすべてこの種類に属します。内容文字列は特定フォント内のグリフによって表現されますので、使用可能なキャラクターの範囲はフォント / エンコーディングの組み合わせに依存します。

内容文字列の解釈は、`textformat` オプション（後述）と、選択したエンコーディングによって制御されます。`textformat=auto` ならば（これがデフォルト）、`unicode・glyphid` エンコーディングに対しては `utf16` 形式が用いられます。それ以外のすべてのエンコーディングに対しては形式は `bytes` です。

C 言語バインディングでは、UTF-16 文字列の長さを別途 `length` 引数で与える必要があります。

ハイパーテキスト文字列 ハイパーテキスト文字列は、しおりや注釈などのインタラクティブ機能のために用いられるものであり、PDFlib API リファレンスでは「**ハイパーテキスト文字列**」として示しています。インタラクティブ機能のための関数の多くの引数やオプションがこの種類に属します。

ハイパーテキスト文字列の解釈は `hypertextformat・hypertextencoding` オプション（後に詳述）によって制御されます。`hypertextformat=auto` ならば（これがデフォルト）、`hypertextencoding=unicode` の場合には `utf16` 形式が用いられ、それ以外の場合には `bytes` が用いられます。

C 言語では、UTF-16 文字列の長さを別途 `length` 引数で与える必要があります。

名前文字列 名前文字列は、外部ファイル名・フォント名・ブロック名などのために用いられるものであり、PDFlib API リファレンスでは「**名前文字列**」として示しています。これはハイパーテキスト文字列とわずかに違います。

ファイル名は特殊な場合です：オプション `filenamehandling` は、API に与えられたファイル名をローカルファイルシステムで使用できる文字列へどのように PDFlib が変換するかを指定します。

デフォルトでは名前文字列は `host` エンコーディングで解釈されます。しかし、名前の先頭に UTF-8 BOM があるときは、それは UTF-8 として（先頭に EBCDIC UTF-8 BOM があるときは EBCDIC UTF-8 として）解釈されます。`usehypertextencoding=true` なら、`hypertextencoding` で指定されたエンコーディングが名前文字列にも適用されます。これはたとえば、フォントやファイルの名前を Shift-JIS で指定するのに使えます。`hypertextencoding=unicode` の場合は、PDFlib は UTF-16 文字列を前提しますので、2 個の null バイトで終了する必要があります。

C では、UTF-8 文字列に対しては `length` 引数は 0 でなければなりません。0 以外なら文字列は UTF-16 として解釈されます。それ以外の非 Unicode 対応の言語バインディングでは名前文字列はかならず UTF-8 形式で与える必要があります。

内容文字列とハイパーテキスト文字列に対するテキスト形式 Unicode 文字列は、UTF-8 または UTF-16 形式で、任意のバイト順序で与えることができます。形式の選択は、`textformat` オプションでページ記述のテキストに対して、`hypertextformat` オプションでインタラクティブ要素群に対して制御することができます。これらのオプションで対応している値を表 5.3 に示します。`[hyper]textformat` オプションのデフォルトは `auto` です。名前文字列群に対して同じ動作を強制するには `usehypertextencoding` オプションを用います。`hypertextencoding` オプションのデフォルトは `auto` です。

表 5.3 textformat・hypertextformat オプションに対する値

[hyper]textformat	説明
bytes	文字列の 1 バイトが 1 キャラクターに対応します。これは 8 ビットエンコーディングと記号フォントで有用です。文字列の先頭に UTF-8 BOM があるときは、評価されたのち除去されます。
utf8	文字列は UTF-8 形式であると期待されます。不正な UTF-8 列があったときは、glyphcheck=error なら例外が発生し、そうでなければ削除されます。
ebcdicutf8	文字列は、EBCDIC コードされた UTF-8 形式であると期待されます (IBM System i・IBM Z のみ)。
utf16	文字列は UTF-16 形式であると期待されます。文字列の先頭に Unicode バイト順序マーク (BOM) があるときは、評価されたのち除去されます。BOM がないときは、その文字列はマシンのネイティブなバイト順序であると期待されます (たとえば Intel x86/x64 ではリトルエンディアン)。
utf16be	文字列は UTF-16 形式で、バイト順序はビッグエンディアンであると期待されます。バイト順序マークについて特別な扱いはありません。
utf16le	文字列は UTF-16 形式で、バイト順序はリトルエンディアンであると期待されます。バイト順序マークについて特別な扱いはありません。
auto	<p>内容文字列：8 ビットエンコーディングに対しては bytes と等価で、unicode・glyphid エンコーディングに対しては utf16 と等価です。</p> <p>ハイパーテキスト文字列：BOM 付きの UTF-8・UTF-16 文字列は検知されます (C では UTF-16 文字列はダブル null で終了する必要があります)。文字列の先頭に BOM がないときは、8 ビットエンコーディングの文字列として、hypertextencoding オプションに従って解釈されます。</p> <p>この設定にしておけば、Unicode をネイティブに用いないたいの環境で、適切なテキスト解釈ができるようになります。</p>

textformat の設定はあらゆるエンコーディングに対して効果がありますが、とりわけ **unicode** エンコーディングに対して有用です。エンコーディングと **textformat** のさまざまな組み合わせに対するテキスト文字列の解釈を表 5.4 で説明します。内容文字列内のコードまたは Unicode 値が、選ばれたフォント内の適切なグリフで表現できない場合については、オプション **glyphcheck** が PDFlib の動作を制御します (183 ページ「グリフ置換」参照)。

表 5.4 エンコーディングとテキスト形式の関係

[hypertext]encoding	textformat=bytes	textformat=utf8・utf16・utf16be・utf16le
すべての文字列種別：		
auto	117 ページ「自動エンコーディング」の項を参照	
unicode	8 ビットコードは U+0000 ~ U+00FF の Unicode 値	選ばれたテキスト形式に従ってエンコードされた任意の Unicode 値 ¹
内容文字列のみ：		
8 ビットエンコーディング	8 ビットコード	選ばれたエンコーディングに従って Unicode 値を 8 ビットコードに変換します ¹ 。
glyphid	8 ビットコードは 0 ~ 255 のグリフ ID	Unicode 値はグリフ ID として解釈されます ² 。

1. その Unicode キャラクターがフォント内で得られないときは、PDFlib は glyphcheck オプションに従って、例外を発生させるか、またはそれを置き換えます。

2. そのグリフ ID がフォント内で見つからないときは、PDFlib は glyphcheck 設定に従って、例外を発生させるか、またはそれをグリフ ID 0 に置き換えます。

オプションリスト オプションリスト内の文字列については、特別な注意が必要です。なぜなら非 Unicode 対応の言語バインディングでは、それは UTF-16 形式の Unicode 文字列として表現できず、バイト文字列としてしか表現できないからです。この理由から、Unicode のオプションに対しては UTF-8 が用いられます。PDFlib はオプションの先頭の BOM を見ることで、それをどう解釈するかを決定します。BOM を用いて文字列の形式が決定され、そして文字列の種類（内容文字列・ハイパーテキスト文字列・名前文字列）を用いてエンコーディングが決定されます。具体的には、文字列オプションの解釈は以下のように動作します：

- ▶ オプションの先頭に UTF-8 BOM (*0xEF 0xBB 0xBF*) があるなら、それは UTF-8 として解釈されます。EBCDIC ベースのシステムの場合：オプションの先頭に EBCDIC UTF-8 BOM (*0x57 0x8B 0xAB*) があるなら、それは EBCDIC UTF-8 として解釈されます。
- ▶ BOM が見つからないときは、文字列の解釈は文字列の種類に依存します：
 - ▶ 内容文字列は、適用可能な *encoding* オプションか、その対応するフォントのエンコーディング（どちらか存在するほう）に従って解釈されます。
 - ▶ ハイパーテキスト文字列は、*hypertextencoding* オプションに従って解釈されます。
 - ▶ 名前文字列は、*usehypertextencoding=true* なら *hypertext* の設定に従って、そうでなければ *auto* エンコーディングで解釈されます。

キャラクター{}はオプションリスト内の文字列内では特別な扱いを要し、文字列オプション内で用いるときはキャラクター\を前につける必要があります。

5.3.3 シングルバイト（8 ビット）エンコーディング

8 ビットエンコーディングは、バイト値をそれぞれ、BMP 内の Unicode 値を持つ 1 つのキャラクターへマップします。そこでキャラクターは 255 種類までです。なぜならコード 0（ゼロ）は *.notdef* キャラクターのために予約されているからです。PDFlib は、以下のエンコーディングを内蔵しています：

```
winansi (cp1252と等しい。iso8859-1のスーパーセット),
cp1252
iso8859-1
macroman (オリジナルのMacintoshの文字集合),
ebcdic (EBCDICコードページ1047),
ebcdic_37 (EBCDICコードページ037),
pdfdoc (PDFDocEncoding)
```

以下のエンコーディングは PDFlib パッケージ内で別ファイルとして利用可能です：

```
iso8859-2, iso8859-3, iso8859-4, iso8859-5, iso8859-6, iso8859-7, iso8859-8, iso8859-9,
iso8859-10, iso8859-13, iso8859-14, iso8859-15, iso8859-16,
cp1250, cp1251, cp1253, cp1254, cp1255, cp1256, cp1257, cp1258
```

これらのエンコーディングのためのリソースファイルへのアクセスについては、それがシステムのコードページとして利用可能でない限りは、以下のように構成する必要があります（59 ページ「3.1.4 リソース構成とファイル検索」を参照）

```
p.set_option("SearchPath={{/usr/pdflib/resource/codepage}}");
p.set_option("Encoding={iso8859-5=iso8859-5.cpg}");
```

ホストエンコーディング 特殊なエンコーディング *host* は固定した意味を持たず、環境のプラットフォームによって、以下の 8 ビットエンコーディングへマップされます：

- ▶ MVS か USS を持つ IBM System Z では *ebcdic* へマップされます。
- ▶ IBM System i では *ebcdic_37* へマップされます。
- ▶ Windows では *winansi* へマップされます。
- ▶ それ以外のすべてのシステムでは *iso8859-1* へマップされます。

ホストエンコーディングが有用なのは何といっても、プラットフォーム非依存のテストプログラムや、その他の単純なプログラムを書くときです。製品版でのホストエンコーディングの使用は推奨しませんので、何らかの適切なエンコーディングに置き換えるべきです。

自動エンコーディング PDFlib は、特定の環境に対してもっとも自然なエンコーディングを指定できるしくみに対応しています。エンコーディング名としてキーワード *auto* を与えると、プラットフォームや環境によって、以下のテキストフォント用 8 ビットエンコーディングを選択したことになります：

- ▶ Windows の場合：カレントのシステムコードページ（詳しくは後述）
- ▶ Unix・macOS の場合：*iso8859-1*
- ▶ IBM System i の場合：カレントジョブのエンコーディング (*IBMCCSIDoooooooooooo*)
- ▶ IBM System Z の場合：*ebcdic* (=コードページ 1047)

記号フォントでは、キーワード *auto* は *builtin* エンコーディングへマップされます (137 ページ「6.3.2 記号フォントに対するエンコーディングを選ぶ」参照)。自動エンコーディングは多くの場面で便利ですが、その反面、この方式を用いた PDFlib クライアントプログラムは他の環境では使えなくなります。

auto エンコーディングは、非 Unicode 対応の言語バインディングでは、名前文字列のデフォルトエンコーディングとして用いられています (113 ページ「5.3 非 Unicode 対応言語バインディング」参照)。

システムコードページを流用 PDFlib は、コードページ定義をシステムから取得することができます。この機能を利用すれば、コードページの実装作業をしなくてすむのでとても便利です。組み込みエンコーディングやユーザー定義エンコーディングの名前を *PDF_load_font()* に与えるのではなく、ただ、システムが知っているエンコーディング名を利用すればよいのです。この機能が利用できるのはいくつかの限られたプラットフォーム上だけであり、そのエンコーディング文字列の文法はプラットフォーム依存です：

- ▶ Windows では、エンコーディング名は *cp<番号>* です。ここで *<番号>* は、システムにインストールされている任意のシングルバイトコードページの番号です (マルチバイトの Windows コードページについては 175 ページ「7.6.1 TrueType・OpenType 日中韓フォントを用いる」参照)：

```
font = p.load_font("Helvetica", "cp1250", "");
```

シングルバイトコードページは内部の 8 ビットエンコーディングに変換されるのに対し、マルチバイトコードページは実行時に Unicode へマップされます。テキストは、選んだコードページと互換の形式で与える必要があります (たとえば *cp932* に対しては SJIS、119 ページ「日中韓フォントのためのコードページ」参照)。

- ▶ Linux では、*iconv* 機能が対応しているすべてのコードセット識別子を使えます。
- ▶ IBM System i では、任意の *Coded Character Set Identifier* (CCSID) が使えます。CCSID は文字列として与える必要があります、PDFlib は、与えられたコードページ番号に *IBMCCSID*

という接頭辞をつけます。また PDFlib は、コードページ番号が 5 文字に満たないときには頭に 0 を補います。コードページ番号として 0 (ゼロ) を与えると、カレントジョブのエンコーディングが用いられることになります：

```
font = p.load_font("Helvetica", "273", "");
```

- ▶ USS か MVS を持つ IBM System Z では、任意の *Coded Character Set Identifier* (CCSID) が使えます。CCSID は文字列として与える必要があり、PDFlib は、与えられたコードページ名に一切変更を加えずそのままシステムに渡します。

```
font = p.load_font("Helvetica", "IBM-273", "");
```

ユーザー定義の 8 ビットエンコーディング 定義済みエンコーディングのほか、PDFlib はユーザー定義 8 ビットエンコーディングにも対応しています。これを使いたいのは、PDFlib 内で利用可能でない何かの文字集合を扱いたいときです。PDFlib は、グリフ名で定義されたエンコーディングテーブルに対応しているほか、Unicode 値で定義されたテーブルにも対応しています。

ユーザー定義エンコーディングを PDFlib プログラム内で利用できるようにするには、以下の作業をあらかじめ行う必要があります (あるいはエンコーディングは、`PDF_encoding_set_char()` を使って実行時に構築することができます)：

- ▶ エンコーディングの記述を単純なテキスト形式で作成する。
- ▶ そのエンコーディングを PDFlib リソースとして (59 ページ「3.1.4 リソース構成とファイル検索」参照) 構成する。
- ▶ エンコーディングが使うすべてのキャラクターのためのグリフを持っているフォントを与える。

エンコーディングファイルは、グリフ名とコードを 1 行ずつ列挙したものです。エンコーディング定義の冒頭部分は以下のようになります：

```
% PDFlib用エンコーディング定義。グリフ名を使用
```

```
% 名前          コード  Unicode (オプション)
```

```
space          32      0x0020
```

```
exclam        33      0x0021
```

```
...
```

Unicode 値が指定されていないときは、PDFlib はその内部テーブルの中で適切な Unicode 値をさがします。グリフ名でなく Unicode 値を指定することもできます：

```
% PDFlib用コードページ定義。Unicode値を使用
```

```
% Unicode      コード
```

```
0x0020        32
```

```
0x0021        33
```

```
...
```

エンコーディングかコードページのファイルの中の Unicode 値はすべて U+FFFF よりも小さくなければなりません。

5.3.4 日本語・中国語・韓国語 CMap (廃止済)

注 この節で解説しているレガシーエンコーディングのための機能はすべて廃止済です。

注 以下の機能は、CMap を用いて読み込んだフォントに対しては利用できません：テキストフロー・グリフ置換・フォールバックフォント・シェーピング・OpenType レイアウト機能。

PDFlib は、表 5.5 に挙げる CMap を通じて日中韓レガシーエンコーディングに対応しています。レガシーエンコーディングは、非 Unicode 対応の言語バインディングでのみ使用できます。すなわち C の場合か、または Perl・PHP・Ruby で *stringformat=legacy* の場合にのみ使用できます。

表 5.5 日本語・中国語・韓国語テキストのための CMap

ロケールと文字コ レクション	CMap 名
日本語 (<i>Adobe-Japan1-7</i>)	UniGB-UTF16-H/V ¹ , GB-EUC-H/V, GBpc-EUC-H/V, GBK-EUC-H/V, GBKp-EUC-H/V, GBK2K-H/V
中国語簡体字 (<i>Adobe-GB1-5</i>)	UniCNS-UTF16-H/V ¹ , B5pc-H/V, HKscs-B5-H/V, ETen-B5-H/V, ETenms-B5-H/V, CNS-EUC-H/V
中国語繁体字 (<i>Adobe-CNS1-7</i>)	UniJIS-UTF16-H/V ¹ , 83pv-RKSJ-H, 90ms-RKSJ-H/V, 90msp-RKSJ-H/V, 90pv-RKSJ-H, Add-RKSJ-H/V, EUC-H/V, Ext-RKSJ-H/V, H/V
韓国語 (<i>Adobe-Korea1-2</i>)	(PDF 2.0 では廃止済) UniKS-UTF16-H/V ¹ , KSC-EUC-H/V, KSCms-UHC-H/V, KSCms-UHC-HW-H/V, KSCpc-EUC-H/V
韓国語 (<i>Adobe-KR-g</i>)	(PDF 2.0。非サポート) UniAKR-UTF16-H

1. PDF 1.5 以上が必要です。

CMap のリソースを構成 CMap を用いて日本語・中国語・韓国語（日中韓）テキストを作成するには、PDFlib は、その対応する CMap ファイルを必要とします。入ってくるテキストを処理して、日中韓エンコーディングを Unicode へマップするためです。CMap ファイルは別パッケージで入手できます。以下のようにインストールする必要があります：

- ▶ Windows では CMap ファイルは、PDFlib のインストールディレクトリ内の *resource/cmap* ディレクトリに置いておけば、自動的に見つけられます。
- ▶ それ以外のシステムでは CMap ファイルは、任意の都合なディレクトリの中に置くことができ、実行時に *SearchPath* を設定することで、CMap ファイルを手動で構成する必要があります：

```
p.set_option("SearchPath={{/パス/パス/resource/cmap}}");
```

日中韓 CMap ファイルの検索先を構成する方式ではなく、適切な *SearchPath* 定義を含む UPR 構成ファイルを指すよう *PDFLIBRESOURCEFILE* 環境変数を設定することもできます。

ネイティブな日中韓レガシーコードを維持 埋め込んでいないフォントに対して *keepnative* フォントオプションを *true* に設定すると、選択した CMap に従ったネイティブなレガシーコード（たとえば Shift-JIS）が PDF 出力へ書き込まれます。そうでない場合にはテキストは CID 値へ変換されます。その利点は、そのようなフォントを埋め込みなしにフォームフィールドで使用できることです。

keepnative=false の場合には、PDF 出力が生成される際に、レガシーなコード列は CID 値群へ変換されます。その利点は、OpenType 機能とテキストフロー整形機能を使用できることです。

日中韓フォントのためのコードページ PDFlib は表 5.6 に挙げるコードページに対応しています。Windows では PDFlib はさらに、システムにインストールされている任意の日中韓コードページに対応しています。

表 5.6 日中韓コードページ (textformat=auto か textformat=bytes で用いる必要があります)

ロケール	コードページ	形式	文字集合
日本語	cp932	Shift-JIS	JIS X 0208:1997 + Microsoft 拡張
中国語簡体字	cp936	GBK	GBK
中国語繁体字	cp950	Big Five	Big Five + Microsoft 拡張
韓国語	cp949	UHC	KS X 1001:1992 + 残り 8822 種のハングル拡張
	cp1361	Johab	Johab

5.4 キャラクターを指定

環境によってはプログラマーは、ソースコードを 8 ビットエンコーディング (*winansi*・*ebcdic* など) で書くことを要求されます。この場合、8 ビット符号化されたテキストの中へ、一部分だけ Unicode キャラクター群を含めるのは厄介な問題です。この状況から開発者を救うため、PDFlib では、テキストを表す補助手段がいくつか使えます。

5.4.1 エスケープシーケンス

PDFlib は、**エスケープシーケンス** (これは実際には誤称です：**バックスラッシュ置換**という用語のほうがよいでしょう) を通じてテキスト文字列内に任意の値を入れ込める方式をサポートしています。たとえば、テキストブロックのデフォルトテキスト内で `\t` シーケンスを使えば、直接キーボード入力では無理かもしれないタブキャラクターが入れられます。同様にエスケープシーケンスは、記号フォントにおけるコードを表すにも便利です。エスケープシーケンスを持たない言語バインディングにおいてはリテラル文字列をあらわすにも便利です。

エスケープシーケンスは、シーケンスを 1 個のバイト値へ置換する命令です。シーケンスは、文字列のカレントエンコーディング内のバックスラッシュキャラクター「\」に対するコードで開始します。バックスラッシュは多くのプログラミング言語において特殊な意味を持ちますので、ソースコード内でリテラルに用いる際には二重にする必要があるかもしれません。エスケープシーケンスの置換から得られるバイト値を表 5.7 に示します。ASCII と EBCDIC のプラットフォームでは違うものもあります。エスケープシーケンスで表せるのは、0 ~ 255 のバイト値だけです。

いくつかのプログラミング言語と異なり、PDFlib のエスケープシーケンスはその種類に応じてつねに固定長となります。ですのでシーケンスの終了キャラクターは必要ありません。

表 5.7 エスケープシーケンスとバイト値一覧

シーケンス	長さ	macOS・Windows・Unix	EBCDIC プラットフォーム	広く知られる解釈
<code>\f</code>	2	0C	0C	フォームフィード
<code>\n</code>	2	0A	15/25	ラインフィード
<code>\r</code>	2	0D	0D	キャリッジリターン
<code>\t</code>	2	09	05	水平タブ
<code>\v</code>	2	0B	0B	ラインタブ
<code>\\</code>	2	5C	E0	バックスラッシュ
<code>\xNN</code>	4	バイト値を表す 16 進 2 桁。例: <code>\xFF</code>		
<code>\NNN</code>	4	バイト値を表す 8 進 3 桁。例: <code>\377</code>		

エスケープシーケンスはデフォルトでは置換されません。エスケープシーケンスを文字列において使うには、*escapesequence* オプションを明示的に *true* に設定する必要があります。必要なところで選択的に *PDF_fit_textline()* など関数内でこれを行うことも可能です。あるいはすべての内容文字列 (すなわちすべてのテキスト出力操作) に対してエスケープシーケンスの展開を有効化することもでき、その方法は以下のとおりです：

```
p.set_text_option("escapesequence=true");
```

エスケープシーケンスの置換は、`PDF_set_option()` を用いてグローバルに有効化することも可能です。このグローバルオプションは、その後に使われるすべての名前文字列・ハイパーテキスト文字列・内容文字列に効力を及ぼします。環境変数の値は名前文字列として扱われますので、それもエスケープシーケンスの展開の対象となります。これはたとえばバックスラッシュキャラクターを含む Windows パスまたはファイル名などについて、望まない動作につながるおそれがあります。ですので、`PDF_set_option()` を用いてエスケープシーケンス置換をグローバルに有効化するのではなく、必要などころだけで選択的にメソッドを用いて (`PDF_fit_textline()` など)、あるいは `PDF_set_text_option()` を用いて内容文字列に対してだけ有効化することを推奨します。

`escapesequence` オプションを設定するのではなく、`PDF_convert_to_unicode()` を用い、入力と出力のエンコーディングを同じにすることによって、文字列内のエスケープシーケンスを置き換えることもできます。例：

```
String s_plain = p.convert_to_unicode("utf16",
    s.character.getBytes("UTF-16"),
    "outputformat=utf16 escapesequence=true");
```

エスケープシーケンスは BOM 検出の後に、しかしターゲット形式への変換の前に、評価されます。`textformat= utf16le` か `utf16be` ならエスケープシーケンスは、選ばれた形式に従って 2 バイト値として表す必要があります。エスケープシーケンス内の各キャラクターは 2 バイトで表現され、そのうち 1 バイトは値 0 になります。`textformat=utf8` なら、生成コードは UTF-8 に変換されません。

エスケープシーケンスが解決できないときには (1x の後の 16 進数が不正など)、例外が発生します。内容文字列についてはこの動作は、`glyphcheck · errorpolicy` 設定で制御されます。

5.4.2 文字参照

クックブック 完全なコードサンプルがクックブックの `fonts/character_references` トピックにあります。

文字参照は、文字列を Unicode 値へ置き換える命令です。この文字列は、カレントエンコーディング内のアンパサンドキャラクター「&」のコードで開始し、セミコロンキャラクター「;」で終了します。ターゲット Unicode 値を表現するためにいくつかの方式が利用可能です：

HTML 文字参照 PDFlib は、HTML 4.0 で定義されているすべての文字実体参照に対応しています。数値文字参照は 10 進・16 進記法で与えることができます。HTML 文字参照の全一覧は以下の場所にあります：

www.w3.org/TR/REC-html40/charset.html#h-5.3

例：

<code>&shy;</code>	U+00AD ソフトハイフン
<code>&euro;</code>	U+00AD ユーログリフ (実体名)
<code>&lt;</code>	U+00AD 小なり記号
<code>&gt;</code>	U+00AD 大なり記号
<code>&amp;</code>	U+00AD アンパサンド記号
<code>&Alpha;</code>	U+0391 ギリシャ文字?

数値文字参照 Unicode キャラクターに対する数値文字参照も HTML 4.0 で定義されています。これはハッシュキャラクター「#」と 10 進または 16 進の数値を必要とし、16 進数値のは小文字か大文字の「X」キャラクターを頭につけます。例：

­	U+00AD ソフトハイフン (10進)
­	U+00AD ソフトハイフン
å	U+00AD 文字aの上に小さな丸 (10進)
å	U+00E5 文字aの上に小さな丸 (16進、小文字x)
å	U+00E5 文字aの上に小さな丸 (16進、大文字X)
€	U+20AC ユーログリフ (16進)
😂	U+1F602 FaceにTears of Joyを加えたもの

PDFlib 独自の实体名 PDFlib では、以下のグループの Unicode 制御キャラクターに対し、カスタムの文字実体参照を使うことができます：

- ▶ 表 7.6 に挙げるデフォルトシェーピング動作をオーバーライドするための制御キャラクター群。
- ▶ 表 7.7 に挙げるデフォルト双方向組版をオーバーライドするための制御キャラクター群。
- ▶ 表 9.1 に挙げるテキストフローの改行と組版のための制御キャラクター群。

例：

&#linefeed;	U+000A ラインフィード制御キャラクター
&#shortab;	U+0009 水平タブ
&#ZWNJ;	U+200C ゼロ幅非接合子

グリフ名参照 フォントの中のグリフに対応するUnicode値があらかじめわからないことが原因でそれに直接アクセスできない場合があります (なぜなら PDFlib が PUA 値を割り当てるのは実行時だからです)。このようなグリフを指定するための代替策として、グリフ名を用いた文字参照を使用することが可能です。これらの参照は、対応する Unicode 値で置き換えられます。グリフ名は以下のソースから導かれます：

- ▶ 代表的なグリフ名は内蔵リスト内で検索されます
- ▶ フォント独自のグリフ名はカレントフォント内で検索されます。この種類の文字参照はフォントを必要とするので、内容文字列でのみ動作します。

グリフ名参照を同定するために、実際の名前はアンパサンドキャラクター「&」の後にピリオドキャラクター「.」を必要とします。例：

&.three;	U+0033 数字3の代表的グリフ名
&.mapleleaf;	(PUA Unicode値) Cartaフォントにおけるカスタムグリフ名
&.T.swash;	(PUA Unicode値) 2番目のピリオドキャラクターはグリフ名の一部です

グリフ名による文字参照は以下のシナリオで有用です：

- ▶ フォント独自グリフ名による文字参照は、内容文字列内で異体字 (スワッシュキャラクターなど) や、特定の Unicode セマンティクスを持たないグリフ (記号・アイコン・装飾) を選ぶのに有用です。ただし、等幅数字をはじめとする多くの機能は、フォントが対応していれば OpenType 機能でもっと簡単に実装できます (163 ページ「7.4 OpenType レイアウト機能」参照)。
- ▶ Adobe グリフリスト内の名前 (*uniXXXX*・*uXXXX*/*uXXXXX*/*uXXXXXX* 形式のものも含め)、およびさまざまなよくある「名前誤り」グリフ名は、内容文字列とハイパーテキスト文字列でつねに受け入れられます。

バイト値参照 数値を文字参照で与えることもできます。これは記号フォント内のグリフを指定するのに有用でしょう。この方式では、ハッシュキャラクター「#」を加えた 10 進または 16 進数が必要です。ここで 16 進数は小文字か大文字の「X」キャラクターを頭に付けます。例 (Wingdings フォントを前提) :

```
&.#x9F;      Wingdingsフォントのビュレット記号
&.#159;      Wingdingsフォントのビュレット記号
```

文字参照を用いる 文字参照はデフォルトでは置換されませんので、内容文字列内で文字参照を用いるには、`charref` オプションを `true` に設定する必要があります :

```
p.fit_textline("Price: 500&euro;", x, y, "charref=true");
```

テキストオプションとして `charref` を与えると、すべての内容文字列に対して文字参照が有効化されます :

```
p.set_text_option("charref=true font=" + font + " fontsize=24");
p.fit_textline("Price: 500&euro;", x, y, "");
```

この `charref` オプションは、`PDF_set_option()` を用いてグローバルに設定することも可能です。しかし、これはすべての名前文字列・ハイパーテキスト文字列・内容文字列に影響しますので、望まない結果を引き起こすおそれがあることから、推奨されません。

`charref` オプションを設定するのではなく、`PDF_convert_to_unicode()` を用い、入力と出力のエンコーディングを同じにすることによって、文字列内の文字参照を置き換えることもできます。例 :

```
String s_plain = p.convert_to_unicode("utf16",
    s.character.getBytes("UTF-16"),
    "outputformat=utf16 charref=true");
```

文字参照を使ううえでのその他の注意点を挙げます :

- ▶ 文字参照は、内容文字列・ハイパーテキスト文字列・名前文字列のいずれでも使えます。例外として、フォント独自グリフ名参照は上述のように内容文字列でのみ動作します。
- ▶ 文字参照は `builtin` エンコーディングのテキスト内では置換されません。しかし、`unicode` エンコーディングを用いることによって記号フォントに対して文字参照を使うことはできます。
- ▶ 文字参照はオプションリスト内では置換されません。ただし、`Unichar` データ型のオプション内では認識されます。この場合、「&」・「;」修飾は外す必要があります。この認識はつねに有効であり、`charref` オプションには従いません。
- ▶ 非 Unicode 対応言語バインディングでは、`textformat=utf16 · utf16be · utf16le` のときは文字参照は 2 バイト値で表す必要があります。`encoding=unicode` かつ `textformat=bytes` のときは文字参照は ASCII で表す必要があります (EBCDIC ベースのプラットフォームでも)。
- ▶ 1 個のアンパサンドキャラクターは、文字参照の開始と見なされます。ただし、その後にセミコロン「;」が見つかり、かつ「&」と「;」の間のキャラクター群が文字参照として有効である場合に限りです。そうでない場合には、そのアンパサンドキャラクター「&」は変更されないままとなります。文字参照を開始させずにアンパサンドキャラクター「&」を表したい場合には、文字参照 `&` を使用することが推奨されます。

- ▶ 文字参照の可能性のある文字列が無効の場合(たとえば&#の後に無効な10進値群が続いている場合や、「&」の後に未知の実体名が続いている場合)には、その動作はオプション *glyphcheck* に依存します：

glyphcheck=none : その文字列はそのまま保たれます。

glyphcheck=replace : その文字列は置換キャラクターで置換されます。

glyphcheck=error : エラーが発生し、テキスト処理は停止されます。*errorpolicy=exception* の場合には例外が発生します。

6 フォント

6.1 フォント形式

6.1.1 TrueType フォント

各種 TrueType フォント形式 PDFlib は、以下のファイル形式のベクトルベースの TrueType フォントに対応しています：

- ▶ Windows と macOS の TrueType フォント (***.ttf**)。欧文・記号・日中韓フォントを含みます。
- ▶ TrueType コレクション (***.ttc**)。1つのファイルの中に複数のファイルが入っています。TTC ファイルは通常、日中韓フォントをグループ化するために用いられますが、1つの欧文フォントの複数のメンバーを1つのファイルにパッケージするのも用いられています。
- ▶ エンドユーザー定義キャラクター (EUDC) フォント (***.tte**)。Microsoft の *eudcedit.exe* ツールで作られます。



TrueType フォント名 フォントファイルを扱う際には、フォントに任意の API 名を割り当てることもできます (141 ページ「フォントデータのソース」参照)。この名前は、フォントを読み込む際に用いられ、そのフォントのファイル名やそのフォントの内部名とは異なってもかまいません。生成された PDF では、TrueType フォントの名前が PDFlib (や Windows) で用いていた名前と異なることがあります。これは正常であり、PDF は TrueType の PostScript 名を用いているという事実によるものです。PostScript 名は TrueType 名とは異なります (例： *TimesNewRomanPSMT* に対して *Times New Roman*)。

6.1.2 OpenType フォント

OpenType フォント形式 (ISO 14496-22 に従って Open Font Format ともいう) は、PostScript と TrueType 技術を結合したものです。これは TrueType ファイル形式の拡張として実装されています。OpenType フォントは、合字やスワッシュキャラクターなど、テキスト出力の改善に利用できるオプションなテーブル (163 ページ「7.4 OpenType レイアウト機能」参照) のほか、複雑用字系シェーピングのためのテーブルを含むこともできます (170 ページ「7.5 複雑用字系の出力」参照)。



OpenType フォントは、TrueType と PostScript のいずれをベースとしたグリフ記述も内容として持つことができます。PostScript ベースのほうは Compact Font Format (CFF) や Type 2 と呼ばれ、たいてい ***.otf** 接尾辞をつけて用いられます。

- ▶ CID (キャラクター ID) アーキテクチャーが日中韓フォントに対して用いられていません。現代の CID フォントは、PostScript アウトラインを持つ OpenType *.otf フォントとしてパッケージされています。
- ▶ OpenType コレクションは、OpenType 1.7 仕様で導入されました。これは、関連する OpenType フォント群を1つの合体ファイル内へまとめるという点において、TrueType コレクションと似ています。OpenType コレクションは ***.ttf** または ***.otc** 接尾辞を用います。
- ▶ バリエーションフォントは、OpenType フォントバリエーションという機構を用いており、これを用いると、同一のフォントファミリーに属する複数のフォントフェイス (ライ

ト・レギュラー・ボールドなど)を、ただ1つのフォントリソースの中へパッケージすることができます。PDFはフォントバリエーションに対応していませんのでこの機構を利用できません。TrueTypeアウトラインを持ったOpenTypeフォントバリエーションをPDFlibで読み込むことはできますが、バリエーションのないデフォルトフォントインスタンスのみが利用可能となります。CFF2テーブル内にPostScriptアウトラインを持ったOpenTypeフォントバリエーションは、PDFが対応していませんので、拒否されます。

6.1.3 SVG・COLR・sbix形式のOpenTypeカラーフォント

クックブック 完全なコードサンプルがクックブックの fonts/color_font トピックにあります。

OpenTypeカラーフォントにはさまざまな種類があり、特に絵文字のためによく利用されるようになりました。PDFlibは、以下に挙げる種類のOpenTypeカラーフォント(ISO 14496-22:2019に従って)に対応しています。

PDFはOpenTypeカラーフォントに直接は対応していないので、PDFlibはカラーグリフ群に対してType 3フォントを生成します。PDFlibは、OTF・TTF・TTC・OTC・WOFF・WOFF2形式のOpenTypeカラーフォントと、ホストフォントとしてインストールされているカラーフォントを処理します。PDF出力において、カラーフォントによるテキストは、カラーグリフがラスター画像へ変換されていないので、完全に検索・選択が可能のままとなります。

OpenType SVG フォント この種類のフォントは、SVGグリフ記述(Adobe/Mozilla)を使用します。SVGグリフは、SVG取り込みの記述に従って処理されます(199ページ「8.2 SVGグラフィック」を参照)。SVGグリフは、sRGB色空間で処理され、また、透過を含むことができます。オープンソースの *EmojiOne Color* フォントがその一例です。



OpenType COLR フォント この種類のフォントは、COLR・CPALテーブル(Microsoft)を使用し、1つないし複数のモノグリフをレイヤーすることによってカラーグリフを構築します。この各レイヤーは、sRGB色空間で着色され、また、透過の値を割り当てられることもできます。Windowsのシステムフォント *Segoe UI Emoji* がその一例です。

COLRフォントにおいてレイヤーとしてのみ使用される補助グリフは、カラーグリフを生成するために使用されますが、それ自体を使用することはできません。ただし `encoding=glyphid` を用いればすべてのグリフを指定できますので使用できます。通常の場合と同様に、このような補助グリフがフォントのサブセットに含められるのは、それが文書で実際に使われているときのみです。

OpenType sbix フォント この種類のフォントは、sbixテーブル(Apple)内のグリフビットマップを使用します。このようなフォントは、デザイン単位あたりのピクセル数がさまざまであるビットマップグリフの1つないし複数の集合を内容とします。PDFlibはデフォルトでは最大のビットマップを使用しますが、これは `strikeselect` オプションを用いて変更することも可能です。sbixテーブルによるカラーフォントは、sRGB色空間で処理され、また通常、透過マスクを持っているグリフビットマップを内容として持ちます。macOSのシステムフォント *Apple Color Emoji* がその一例です(これと同名のiOSのシステムフォントは使用できません。なぜならプロプライエタリーなビットマップ圧縮方式を使用しているため)。

さまざまなカラーフォント処理モード カラーフォントに対して意図する用途とその性質に応じて、*colormode* フォントオプションを使ってカラーフォントの処理モードを選択することができます。カラーフォントモードの比較を表 6.1 に示します。1つのフォントのカラーモードを後から変えることはできませんので、そのフォントを再度読み込んだ場合には、その *colormode* オプションは無視されて、最初の読み込み操作のカラーモードが使用されます。

表 6.1 OpenType カラーフォントのための処理モード (フォントオプション *colormode*)

	<i>colormode=combined</i> (デフォルト)	<i>colormode=ignoremono</i>	<i>colormode=ignorecolor</i>
説明	1つのフォントにモノグリフとカラーグリフが共存	モノグリフは無視されて純粋なカラーフォントが生成されます	カラーグリフは無視されて純粋なモノフォントが生成されます
利点	モノグリフとカラーグリフの扱いを一本化できます。カラーグリフもモノグリフも元の OpenType フォントと同じグリフ ID を持つこととなります。 COLR フォントのレイヤーグリフも <i>encoding=glyphid</i> を用いて使用できます。	モノグリフによる不要な OpenType フォントがスキップされます	望まないカラーグリフ・透過がなくなります
用途	モノグリフとカラーグリフをシンプルに使用	カラーフォントを他のフォントのフォールバックにする	白黒のワークフロー、透過回避
制約	—	モノグリフ・OpenType 機能・合字・シェーピングなし	カラーグリフなし

OpenType カラーフォント内のグリフの種類 カラーフォントは以下の種類のグリフを持っている可能性があります：

- ▶ カラーグリフ：色情報を内蔵しているグリフ。透過情報も持っていることがあります。色の情報がグリフ記述に含まれているので、個々のグリフの色が固定されていますから、カレントの塗り色・描線色にかかわらずグリフの色はつねに同じになります。COLR テーブルを持っている OpenType カラーフォントは、内蔵の色ではなく「前面色」を割り振られるためのグリフレイヤーを持っていることがあります。この色と、それに対応する透過を、*foregroundcolor*・*foregroundtransparency* フォントオプションを用いて与えることができ、これらの値はそのフォントのすべてのグリフに対して固定であり、かつそのようなグリフを使用するすべての場合において固定となります。カラーグリフを、*colormode=ignorecolor* を用いてスキップすることもでき、その場合には、そのフォントは色情報を何も持っていないフォントのように処理されます。
- ▶ 単色 (モノともいう) グリフ：形状のみを記述しており、色情報・透過情報を何も持っていません。モノグリフは、ページ上に配置すると、カレントの塗り色で着色できます。モノグリフは、*colormode=ignoremono* を用いてスキップすることもできます。しかし、あらゆるカラーフォントがモノグリフを持っているわけではないことに留意してください。多くの *sbix* カラーフォントは、使用可能なモノグリフを持っていないので、*colormode=ignorecolor* を用いて使用するには不適當です。

1つのフォントの中に、1つのグリフのカラー版とモノ版が共存していることもあります。

カラーフォントを使用 OpenType カラーフォントを各種アプリケーションで使用する際の注意点は：

- ▶ 多くの絵文字キャラクターがBMP外に符号化されています。すなわちUnicodeコード点が U+FFFF より大きいです。お使いの開発環境によって、このような Unicode 値を表現する方法はさまざまです。さらなる情報と例が、31 ページ「2 章 PDFlib の言語バインディング」の言語ごとの各節にあります。
- ▶ 絵文字キャラクターはさまざまに修飾・結合できます。161 ページ「7.3 絵文字シーケンス」を参照。
- ▶ 絵文字キャラクターを持っている OpenType カラーフォントはフォールバックフォント機能を利用できます。例は 147 ページ「カラー絵文字キャラクターをテキストフォントに追加」を参照してください。
- ▶ 他のフォント形式と異なり、カラーフォントに対してはサブセット化はデフォルトで有効化されます。サブセット化を無効化するにはオプション `subsetting=false` を用います。
- ▶ OpenType カラーフォントのカラーグリフは色と透過を内蔵していますので、そのフォントを使用するさいにそれらを変えることはできません。同様に、テキスト書式オプション `textrendering` はカラーグリフに対しては何ら効力を持ちませんが、ただし `textrendering=3` (不可視テキスト) と `7` (クリッピング) の場合にはテキストが不可視になります。
- ▶ OpenType カラーフォントのグリフは透明な要素を含んでいることがあります。PDF/A-1 と PDF/X-3 では透過は許されませんのでこのことがひっかかってきます。サブセット化が有効化されている場合には、透過グリフは、使用される際には検出されず、`PDF_end_document()` でフォントのサブセットが生成される時に初めて検出されます。ですので、ページオプション `transparencygroup=auto` は、サブセット化が有効化されている場合には、カラーフォント内の透過を検出しません。
- ▶ Unicode 異体字シーケンスを使って、キャラクターに対するテキストまたは絵文字スタイルグリフを選択できます(179 ページ「7.7 Unicode 異体字セレクター(UVS)」を参照)。

カラーフォントに関する情報をクエリー OpenType カラーフォントの読み込みが成功した後は、以下の色関連の情報を、`PDF_info_font()` を用いて取得できます：

- ▶ キーワード `colortype` は OpenType カラーフォントの種類を取得します。
- ▶ キーワード `colormode` は処理モードを取得します。
- ▶ キーワード `numcolorglyphs` はカラーグリフの数を取得します。

6.1.4 WOFF・WOFF2 フォント

WOFF(Web Open Font Format = Webオープンフォント形式)は、TrueType フォントと OpenType フォントのための、圧縮されたコンテナ形式です。WOFF は、Web 上での使用のために設計されたものであり、小さなフォントファイルサイズを実現するために、圧縮機能とサブセット化機能を提供しています。WOFF フォントはファイル名拡張子 `.woff` を使用しています。



WOFF2 形式は、WOFF をベースにしており、圧縮が向上しています。WOFF2 は W3C 勧告として 2018 年に発行されています。WOFF2 フォントはファイル名拡張子 `.woff2` を使用しています。

PDFlib は、背後の TrueType または OpenType フォントに対応している限りにおいて、WOFF・WOFF2 フォントに対応しています。

注 WOFF2 フォントは IBM System i・IBM Z では対応していません。

6.1.5 PostScript Type 1 フォント（廃止済）

注 PostScript Type 1 フォント対応は廃止されました。

PostScript Type 1 フォントは2つの部分に分かれています:アウトラインデータとメトリック情報です。PDFlib は、PostScript Type 1 フォントのための以下の形式に対応しています:

- ▶ プラットフォーム非依存な AFM (Adobe Font Metrics) 形式と、Windows 固有の PFM (Printer Font Metrics) フォント形式。
- ▶ プラットフォーム独立な PFA (Printer Font ASCII) 形式と、Windows 独自の PFB (Printer Font Binary) 形式。どちらも PostScript Type 1 形式のアウトライン情報用の形式です。

6.1.6 SING フォント（グリフレット）

SING フォント (*Smart Independent Glyphlets*) は OpenType ファイル形式の拡張です。SING フォントは、日中韓テキストにおける外字問題、すなわち Unicode で符号化されていないカスタムなグリフに対する解決策として開発されたものです。

SING フォントは1個のグリフだけを内容として持ちます(あわせて縦書き字体も含むこともあります)。この「メイン」グリフの Unicode 値は、PDFlib で取得することができ、それにはグリフ ID を要求し、ついでこのグリフ ID に対する Unicode 値を要求します:

```
maingid = (int) p.info_font(font, "maingid", "");  
uv = (int) p.info_font(font, "unicode", "gid=" + maingid);
```

SING フォントを、*PDF_load_font()* の *fallbackfonts* オプションの *forcechars* オプションの *gaji* サブオプションを用いてフォールバックフォントとして利用することを推奨します。176 ページ「7.6.3 EUDC・SING フォントによる外字キャラクター」を参照してください。

6.1.7 Type 3 フォント

他のすべてのフォント形式とは異なり、Type 3 フォントはファイルから取得されるのではなく、標準 PDFlib グラフィックメソッド群を用いて実行時に定義されます。Type 3 フォントは以下の用途で有用です:

- ▶ ビットマップフォント。
- ▶ 色または透過を内蔵しているフォント。
- ▶ ロゴなどのカスタムグラフィックを、シンプルなテキスト操作命令で簡単に印刷可能。
- ▶ いずれのフォントやエンコーディングでも入手できない日本語の外字(ユーザー定義キャラクター)。
- ▶ カラーフォントに対しては内部的に Type3 フォントが生成されます(128 ページ「6.1.3 SVG・COLR・sbix 形式の OpenType カラーフォント」を参照)。

Type 3 フォントは2種類のグリフを持っている可能性があります:

- ▶ デフォルトでは、グリフは無着色であり、グリフ記述はグリフの形状のみを指定しています。色は、その生成されたフォントでテキストが出力される際にカレントの塗り色が指定されます。グリフ記述内で色・透過・画像(マスクを除く)を設定するメソッドは許されません。
- ▶ グリフオプション *colorized* を *true* に設定すると、グリフ記述は色と透過を指定し、また、任意の画像を配置することもできます。グリフの色は固定されますので、テキストが配置される際に有効になっている塗り色にかかわらずそのグリフは同じ見た目になります。

Type 3 フォントの定義の中では、PDFlib のベクトルグラフィック・ラスター画像の機能がすべて使えますし、テキスト出力の機能でさえすべて使うことができるので（上述の色の制約が課されます）、Type 3 フォントのグリフの内容に関して制約は何もありません。Type 3 フォントが最も頻繁に利用されるのはビットマップグリフのためです。以下の例は、単純な Type 3 フォントを定義しています：

```
p.begin_font("Fuzzyfont", 0.001, 0.0, 0.0, 0.001, 0.0, 0.0, "");

p.begin_glyph_ext(-1, "glyphname=circle width=1000 boundingbox={0 0 1000 1000}");
p.arc(500, 500, 500, 0, 360);
p.fill();
p.end_glyph();

p.begin_glyph_ext(-1, "glyphname=ring width=400 boundingbox={0 0 400 400}");
p.arc(200, 200, 200, 0, 360);
p.stroke();
p.end_glyph();

p.end_font();
```

クックブック 完全なコードサンプルがクックブックの `type3_fonts/starter_type3font`・`type3_fonts/type3_bitmaptext`・`type3_fonts/type3_vectorlogo` トピックにあります。

こうしたフォントはPDFlibの中に登録され、その名前は `PDF_load_font()` に与えることができます。Type 3 フォントを扱う時は以下のことに留意してください：

- ▶ `.notdef` グリフが存在して必ずスロット 0 を占めている必要があります。
- ▶ 適切な Unicode 値を `PDF_begin_glyph_ext()` に与えることを強く推奨します。PDFlib は、与えられた Unicode 値から自動的に AGL (Adobe Glyph List) に従って正しいグリフ名を生成しますが、`glyphname` オプションで代替名を与えることも可能です。
- ▶ Type 3 フォント内でビットマップによるグリフを定義する際には `inline` 画像オプションを用いることを推奨します (`imagetype=jpeg`・`raw` の場合にのみ)。`interpolate` 画像に対するオプションは、Type 3 ビットマップフォントの見映えを向上させるために有用でしょう。
- ▶ ビットマップデータを用いてグリフを定義する場合、ビットマップ中の使われていないピクセルは、背景にかかわらず白く印刷されます。これを避けて背景色が透けて見えるようにするには、ビットマップ画像を作成する際に `mask` オプションを用います。
- ▶ PDF内のType 3フォントは8ビットアドレス指定を必要としますので、使用可能なグリフは 255 個に限定されます (`.notdef` グリフが必須なので)。しかし PDFlib は、Type 3 フォントに対して任意の数のグリフを受け入れて、そのグリフ集合全体を表現するために必要な数の補助的な Type 3 フォントを生成します。`unicode` エンコーディングで読み込んだ Type 3 フォントにはグリフ 255 個の制約は課されません。

6.2 Unicode のキャラクターとグリフ

6.2.1 グリフ ID

フォントはグリフの集合です。そこで各グリフは可視の形状を定義します。PDFlib は、フォント内の各グリフに番号を割り当てます。この番号をグリフ ID または GID といいます。GID 0 (ゼロ) は *.notdef* グリフを指します。*.notdef* グリフの見た目はフォント形式やベンダーによって異なりますが、よくある実装は空白グリフか白四角か四角バツテンです。最高の GID は、そのフォント内のグリフ数より 1 少ない数であり、これは *PDF_info_font()* の *numglyphs* キーワードでクエリーすることができます。

グリフ ID の割り当て方はフォント形式によって異なります：

- ▶ TrueType・OpenType フォントはすでに内部 GID を含んでいますので、PDFlib はこの GID を用います。
- ▶ CID キー付き OpenType 日中韓フォントでは、CID が GID として用いられます。
- ▶ それ以外のフォント種別については、PDFlib がグリフに、フォント内のその対応するアウトライン記述の順番に従って付番します。

PDFlib では、Unicode などのエンコーディングではなく GID でグリフを選ぶこともできます (136 ページ「グリフ ID エンコーディング」参照)。直接 GID 指定は、グリフ数をクエリして全グリフをなめることでフォントの概観表を印刷するなど、特殊な応用で有用です。

6.2.2 グリフに対する Unicode マッピング

Unicode マッピングと多義グリフ PDFlib は、1 つのフォントの中のすべてのグリフに対して Unicode 値を割り当てます。Unicode 値を利用可能でないグリフをフォントが持っていることもあります。たとえば Unicode 規格外の異体字グリフ (スウォッシュキャラクターなど)・拡張合字・非テキスト記号などです。この場合には PDFlib は PUA 値をそのようなグリフに割り当てます。

フォントによっては、ある 1 つのグリフを用いて複数の Unicode 値が表現されていることがあります。こういう多義グリフのよくある例は、空グリフが U+0020 の空白と U+00A0 のノーブレイクスペースの両方を表す場合や、1 つのグリフが U+2126 のオーム記号と U+03A9 のギリシャ文字 Ω を表す場合です。

複数の Unicode 値が同一のグリフで表されている場合には、PDFlib は、そのグリフを当該 Unicode 値群のうちの 1 つへマップする ToUnicode CMap を生成します。その他の Unicode 値は、同じグリフ ID を用いて出力されますが、その適切な Unicode 値を *ActualText* 属性で割り当てられます。これによって、正しいセマンティクスが、生成された PDF 出力の中でも保たれます。

Acrobat のテキスト選択の不具合 不幸なことに、Acrobat DC はときどき、*ActualText* 属性を持っているテキストのテキスト選択で不具合を起こします。選択・ハイライトされないグリフが生じるのです。見た目上のハイライトは欠けているのに、クリップボードへは、*ActualText* 属性のテキストが正しくコピーされます。この選択不具合を回避するには以下の手があります：

- ▶ そもそも多義グリフを避ける。
- ▶ テキストフィルターオプション *actualtext=false* を用いることによって *ActualText* 属性の生成を無効化する。

とはいえ、これらの手法は推奨されません。なぜなら、異なるキャラクターが同一 Unicode 値を持って出力されてしまうおそれがあるので、内容の再利用を妨げるからです。

マップなしグリフと私用領域 (PUA) 場合によっては、ある特定のグリフに対する Unicode 値をフォントが提供していないことがあります。この場合には、PDFlib は Unicode 私用領域 (109 ページ「5.1 Unicode の重要な諸概念」参照) 内の値をそのグリフに割り振ります。このようなグリフを**マップなしグリフ**といいます。フォント内のマップなしグリフの数は、**PDF_info_font()** の **unmappedglyphs** キーワードでクエリーすることができます。マップなしグリフは、フォントの検索性とテキスト抽出を制御する ToUnicode CMap 内では Unicode 置換キャラクター U+FFFD で表されます。結果としてマップなしグリフは、生成された PDF からテキストとして正しく抽出できなくなります。ただし、この動作は変更することもでき、それは特に日中韓キャラクターに対して有用です。詳しくは 177 ページ「外字キャラクターに対する PUA 値を温存」を参照してください。

PDFlib が PUA 値をマップなしグリフに割り振っていく際には、以下のプール内の若い値から順に用いていきます：

- ▶ 基本となるのは基本多言語面 (BMP) 内の Unicode PUA 領域、すなわち範囲 U+E000 ~ U+F8FF です。必要であればこれに加え、第 15 面 (U+F0000 to U+FFFFD) 内の PUA 値も用いられます。
- ▶ そのフォントが内部的にすでに割り振っている PUA 値は、新たな PUA 値を作成する際には用いられません。
- ▶ Adobe 領域 U+F600 ~ F8FF 内の PUA 値は用いられません。

生成される PUA 値は、1 つのフォント内で一意です。あるフォント内のグリフに対して生成される PUA 値の割り振りは、他のフォントからは独立です。

6.3 フォントを読み込む

6.3.1 テキストフォントに対するエンコーディングを選ぶ

フォントは、明示的に `PDF_load_font()` メソッドで読み込むこともでき、あるいは暗黙的に、`PDF_add/create_textflow()` や `PDF_fill_textblock()` といったメソッドに `fontname・encoding` オプションを与えることで読み込むこともできます。どのような方式を用いてフォントを読み込むのかにかかわらず、適切なエンコーディングを指定する必要があります。エンコーディングは以下を決定します：

- ▶ PDFlib が与えられるテキストをどのテキスト形式であると見なすか。
- ▶ フォント内のどのグリフが使えるか。
- ▶ ページ上のテキストとフォント内のグリフデータが PDF 出力内にどのように格納されるか。

PDFlib のテキスト処理は Unicode 規格に基づいています。今どきのあらゆる開発環境が Unicode に対応していますので、Unicode 文字列を使ってできるだけ簡単に PDF 出力を作成できるようにすることが私たちの目標です。ただし、Unicode を扱わない開発者はそのアプリケーションを Unicode へ切り替える必要はありません。レガシエンコーディングも使うことができますからです。

どのエンコーディングを選ぶかは、フォントや、得られるテキストデータや、いくつかのプログラミング的側面によって決まります。以下この項では、さまざまな分類のエンコーディングの概観を示すことで、適切なエンコーディングを選ぶための助けとします。

Unicode エンコーディング `encoding=unicode` とすると、Unicode 文字列を PDFlib に渡すことができます。このエンコーディングはすべてのフォント形式に対して使えます。利用する言語バインディングによって、そのプログラミング言語（Java など）が提供している Unicode 文字列データ型を利用できる場合もあれば、Unicode を UTF-8・UTF-16・UTF-32 のいずれかの形式でリトルエンディアン・ビッグエンディアンのいずれかのバイト順序で内容として持つバイト配列を用いる場合もあります（C など）。

`encoding=unicode` では、フォント内のすべてのグリフが指定でき、複雑用字系のシェーピングと OpenType レイアウト機能を使うことができます。PDFlib は、要求された Unicode 値に対するグリフをフォントが含んでいるかどうかをチェックします。グリフが得られないときは、代替グリフを他のフォントから持って来ることができます（146 ページ「6.3.6 フォールバックフォント」参照）。

非 Unicode 対応の言語バインディングでは、PDFlib はデフォルトではテキストが UTF-16 エンコードされていると見なします。しかし、`textformat=bytes` を指定すればシングルバイト文字列を与えることができます。この場合、このバイト値はキャラクター U+0001～U+00FF を、すなわち基本欧文字符群を持つ先頭 Unicode ブロック（ISO 8859-1 と等価）を表します。文字参照を利用すると、この範囲の外の Unicode 値をシングルバイトテキスト内で指定することが可能です。

シングルバイトエンコーディング 8ビットエンコーディングは、テキスト文字列内の各バイトを1個のキャラクターへマップしますので、同時に扱えるキャラクターは255種類までに制限されます（値0は利用できません）。この種類のエンコーディングはすべてのフォント形式に対して使えます。PDFlib は、選ばれたエンコーディングに合ったグリフをフォントが含んでいるかどうかをチェックします。使えるグリフの数が最低限の数に届かないときは、PDFlib は警告メッセージをログ記録します。選ばれたエンコーディングに対して使えるグリフがフォント内でまったく得られないときは、フォント読み込みは「font

`doesn't support encoding` というメッセージを出して失敗します。PDFlib は、要求された入力値に対するグリフをフォントが含んでいるかどうかをチェックします。グリフが得られないときは、代替グリフを他のフォントから持って来ることができます (146 ページ「6.3.6 フォールバックフォント」参照)。

非 Unicode 対応の言語バインディングでは、PDFlib はデフォルトではテキストがシングルバイトエンコードされていると見なします。しかし、`textformat=utf8` か `utf16` を指定すれば UTF-8 か UTF-16 文字列を与えることができます。

8ビットエンコーディングについて詳しくは 116 ページ「5.3.3 シングルバイト (8ビット) エンコーディング」で解説しています。これはさまざまなソースから持って来ることができます：

- ▶ 大量の定義済みエンコーディング。これはさまざまなシステムとさまざまなロケールで利用されている最も重要なエンコーディング群を網羅しています。
- ▶ ユーザー定義エンコーディング。これは、外部ファイルで与えるか、または実行時に `PDF_encoding_set_char()` で動的に構築することができます。このエンコーディングは、グリフ名か Unicode 値に基づくことができます。
- ▶ オペレーティングシステムから持ってきたエンコーディング。システムエンコーディングともいいます。この機能は、Windows と IBM System i・IBM Z で利用可能です。

ビルトインエンコーディング `encoding=builtin` を指定すると、記号フォント内の非テキストグリフに対するシングルバイトコードを使うことができます。フォントのビルトインエンコーディングはフォントの種類によって異なります：

- ▶ TrueType アウトラインを持つ OpenType フォント：このエンコーディングは、フォントのシンボリック cmap に、すなわち `cmap` テーブル内の `(3, 0)` エントリーに基づいて作成されます。このエントリーが存在しない場合には `encoding=builtin` を使用できません。
- ▶ OpenType CFF フォントはエンコーディングを CFF テーブル内に含んでいることがあります。
- ▶ Type 3 の場合、ビルトインエンコーディングはフォントの先頭 255 グリフによって定義されます。

フォントがビルトインエンコーディングを何ら含んでいないときは、フォント読み込みは失敗します (OpenType 日中韓フォントなど)。`PDF_info_font()` で `symbolfont` キーを用いることもできます。これが `false` を返したなら、そのフォントはテキストフォントであり、広く利用されているエンコーディングのいずれかで読み込むことができます。`symbolfont` キーが `true` を返したならそれはできません。記号フォント内のグリフは、各グリフに対応するコードがわかっている場合にのみ利用することが可能です (137 ページ「6.3.2 記号フォントに対するエンコーディングを選ぶ」参照)。

非 Unicode 対応の言語バインディングでは、PDFlib はデフォルトではテキストがシングルバイトであると見なします。これは、いくつかの記号フォントでの指定に伝統的に使われているシングルバイト値を使えるという利点があります。これは他のエンコーディングでは不可能です。しかし、`textformat=utf16` などを指定すればテキストを Unicode 形式で与えることができます。

`encoding=builtin` の難点は、シングルバイトエンコードされたテキスト内では文字参照が使えないことです。

グリフ ID エンコーディング PDFlib では `encoding=glyhid` をすべてのフォント形式に対して使えます。このエンコーディングでは、133 ページ「6.2.1 グリフ ID」で解説する付番方式を用いて、フォント内のすべてのグリフが指定できます。数値グリフ ID は 0 ~

65,565 の値をとれます。最大グリフ ID 値は `PDF_info_font()` で `maxcode` キーを用いてクエリーできます。

非 Unicode 対応の言語バインディングでは、PDFlib はデフォルト (`textformat=utf16`) ではテキストがダブルバイトエンコードされていると見なします。

PDFlib は、与えられたグリフ ID がフォント内で有効であるかどうかをチェックします。複雑用字系のシェーピングと OpenType レイアウト機能が使えます。

グリフ ID はフォント固有であり、場合によっては PDFlib によって作成されることもあるので、`encoding=glyphid` は一般に通常のテキスト出力には適しません。このエンコーディングの主な用途は、アプリケーションがすでにグリフ ID を使用している場合と、フォントのグリフ一覧を印刷したい場合です。

6.3.2 記号フォントに対するエンコーディングを選ぶ

記号フォントは、記号・ロゴ・ピクトグラムなどの非テキストグリフ群を内容として持つフォントです。これは、テキストフォントにはないいくつかの問題を提起します。背景にある問題は、Unicode 規格は一般にあえて記号グリフをエンコードしていないことです (ZapfDingbats フォント内のグリフなど、この規則には例外もありますが)。記号フォントを Unicode ワークフローに適合して利用できるようにするため、TrueType・OpenType フォントはたいていそのグリフに私用領域 (PUA) 内の Unicode 値を割り当てています。Unicode マッピングテーブルがないことから、PostScript Type 1 フォントはこれを行えず、一般にそのグリフを選ぶのに欧文キャラクターのコードを用いています。すべてのフォント形式において、記号グリフはたいていカスタムのグリフ名を持っています。

こうした状況から、記号フォント内のグリフの選択に関して以下のような結果が生じます：

- ▶ 記号 TrueType・OpenType フォントは `encoding=unicode` で最も良く読み込めます。グリフに割り当てられている PUA 値がわかっている場合には、記号グリフを選ぶためにテキスト内でこの値を与えることができます。そのためにはそのフォント内の PUA 割り当てがあらかじめわかっている必要があります。
- ▶ PDFlib は記号 PostScript Type 1 フォントに対して PUA 値を内部的に割り振りますので、この PUA 値は事前にはわかりません。
- ▶ 記号フォント内のグリフを 8 ビットコードを使って指定したい場合には、フォントを `encoding=builtin` で読み込んで、テキスト内で 8 ビットコードを与えることができます。たとえば、数字 4 (コード 0x34) は ZapfDingbats フォント内でチェックマーク記号を選びます。

記号フォントを `encoding=unicode` で使うためには、適切な Unicode 値をテキストで用いる必要があります：

- ▶ Wingdings・Webdings など Microsoft の記号フォントは、範囲 U+F020 ~ U+F0FF の PUA Unicode 値を用いています。
- ▶ それ以外のフォントについては、フォント内の個々のグリフに対する Unicode 値をあらかじめ知っておくか、または実行時に `PDF_info_font()` で決定する必要があります。

制御キャラクター 表 9.1 に挙げる範囲 U+0001 ~ U+001F の Unicode 制御キャラクターは、`encoding=builtin` でもテキストフロー内で使えます。0x20 未満のコードは、記号フォントがそのコードに対するグリフを持たないならば、制御キャラクターとして解釈されます。これは大多数の記号フォントにあてはまります。

ラインフィードキャラクターに対するコードは ASCII と EBCDIC とで異なっていますので、EBCDIC システム上でリテラルなキャラクター 0x0A を使うことは避け、オプショ

ン `escapesequence=true` を用いて PDFlib のエスケープシーケンス `\n` を使うことを推奨します。この `\n` は PDFlib API まで届く必要があることに留意してください。たとえば C では `\\n` と書く必要があります。

文字参照 文字参照は記号フォントに対して使うことができます。しかし記号フォントは一般に、文字参照を開始するアンバサンドキャラクター U+0026 「&」に対するグリフを含んでいません。コード 0x26 も、フォント内の既存のグリフへマップできないので使えません。ですので記号フォントは、文字参照を使う必要があるときは、`encoding=unicode` で読み込む必要があります。文字参照は `encoding=builtin` では動作しません。

6.3.3 例：Wingdings 記号フォント内のグリフを選択

記号フォント内のキャラクターを選択するにはさまざまな方法があり、なかには望みの出力が得られないものもありますので、例を見てみましょう。

フォント内のキャラクターを理解 まず、フォント内の、ターゲットとするキャラクターについていくつかの情報を収集しましょう。これには Windows の「文字コード表」アプリケーションを利用します (図 6.1 参照)：

- ▶ 「文字コード表」は、Wingdings フォント内のグリフ群を表示しますが、それは「詳細表示」内で Unicode へのアクセスを一切与えません。これは、このフォントが内容として持っている記号グリフ群に対して、標準化された Unicode 値が一切登録されていないという事実による結果です。そのかわりに、このフォント内のグリフ群は私用領域 (PUA) 内のダミー Unicode 値を使用していますが、「文字コード表」アプリケーションはこれらの値を明らかにしません。

- ▶ 「文字コード表」ウィンドウの左下隅を見ると、あるいはマウスを *smileface* キャラクターの上に乗せると、「文字コード：0x4A」と表示されます。これはこのグリフのバイトコードです。

このコードは、WinAnsi エンコーディング内の大文字の J キャラクターに対応しています。たとえば、このキャラクターをクリップボードへコピーして、そのクリップボード内容を貼り付けると、その結果は、その対応する Unicode 値 U+004A すなわちキャラクター J となります。にもかかわらず、これはこのキャラクターの Unicode 値ではありませんので、Unicode ワークフロー内で U+004A または J を用いてこれを選択することはできません。

- ▶ このフォント内で使用されている Unicode キャラクターは「文字コード表」には表示されません。しかし、Microsoft が提供している記号フォントは、以下の単純な規則を用いています：

Unicode値 = U+F000 + (「文字コード表」に表示される文字コード)

このことから、*smileface* グリフに対しては Unicode 値 U+F04A との結果が得られます。

- ▶ その対応するグリフ名は、フォントエディターや同様のツールで取得できます。この例ではそれは *smileface* です。

`PDF_info_font()` を使って、Unicode 値・グリフ名・コードのいずれかをクエリーすることもできます。152 ページ「6.5.2 フォント依存のエンコーディング・Unicode・グリフ名クエリー」を参照してください。

記号キャラクターを PDFlib で指定 ターゲットとするキャラクターに関して得られる情報に応じて、Wingdings の *smileface* グリフを選択する方法はいくつかあります：

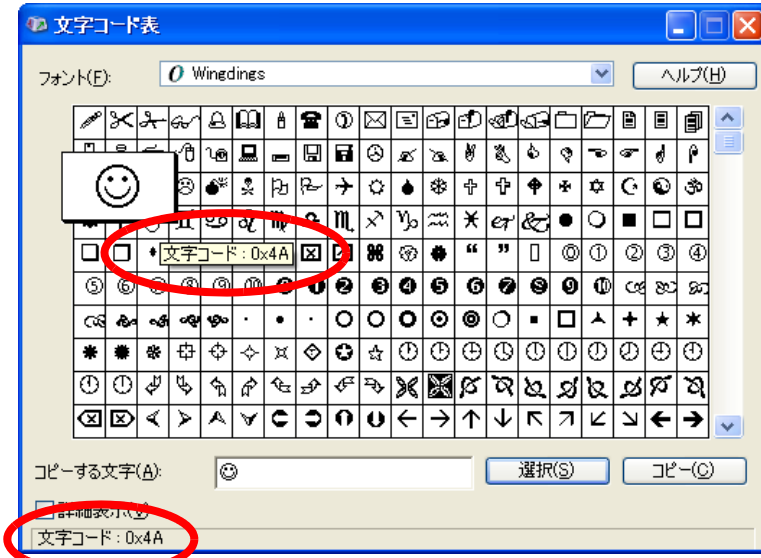


図 6.1
Windows の文字コード表
で Wingdings フォント

- ▶ そのフォント内のそのキャラクターに割り当てられている PUA Unicode 値がわかっているのなら、数値文字参照を使えます (123 ページ「数値文字参照」参照) :

``

UTF-8 で作業している場合には、その対応する 3 バイト列を使えます :

`\xEF\x81\x8A`

Unicode 値は、*encoding=builtin* と *textformat=bytes* の組み合わせでは使えません。

- ▶ 文字コードがわかっているのなら、バイト値参照を使えます (124 ページ「バイト値参照」参照) :

`&.#x4A;`

非 Unicode 対応言語バインディングでは、*encoding=builtin* かつ *textformat=bytes* ならば、文字コードを直接指定できます :

`J`

`\x4A`

- ▶ グリフ名がわかっているのなら、グリフ名参照を使えます (123 ページ「グリフ名参照」参照) :

`&.smileface;`

グリフ名は、*encoding=builtin* と *textformat=bytes* の組み合わせでは使えません。

表 6.2 に、Java・.NET など Unicode 対応言語バインディングの場合の方法を挙げます。


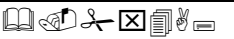
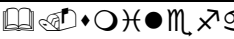
表 6.3 に、C など非 Unicode 対応言語バインディングの場合の方法を挙げます。

表 6.2 Unicode 対応言語バインディング（Java など）で Wingdings フォント内の smileface グリフを指定

エンコーディング	追加オプション	入力文字列	ページ上の印字結果
unicode		\uF04A ¹	☺
	charref		☺
	charref	&.#x4A;	☺
	charref	&.smileface;	☺
		J ²	(スペースか .notdef)
	escapesequence	\x4A	(スペースか .notdef)
builtin	(encoding=unicode で上記と同じ)		

1. U+F04A に対する、Java など多くの Unicode 対応言語での文字列文法
2. バイトコード \x4A に対する Winansi キャラクター

表 6.3 C など非 Unicode 対応言語バインディングで Wingdings フォント内の smileface グリフを指定

エンコーディング	textformat	追加オプション	入力文字列	ページ上の印字結果
unicode	utf16		\xF0\x4A ¹	☺
	utf8, bytes	charref		☺
		charref	&.#x4A;	☺
		charref	&.smileface;	☺
			textformat=utf8 なら : i•š ²	☺
			textformat=bytes なら : J ³	(スペースか .notdef)
		escapesequence ⁴	textformat=utf8 なら : \xEF\x81\x8A ⁵	☺
	escapesequence	textformat=bytes なら : \x4A	(スペースか .notdef)	
builtin	utf16, utf8	(encoding=unicode で上記と同じ)		
builtin	bytes	charref		
		charref	&.#x4A;	
		charref	&.smileface;	
			J	☺
		escapesequence	\x4A	☺

1. バイト順序に応じて、\xF0\x4Aか \x4A\xF0のいずれかで表す必要があります。なお、\xはCのエスケープ文法を示しています
2. 3 バイト列 \xEF \x81 \x8A に対する Winansi キャラクター群
3. バイトコード \x4A に対する Winansi キャラクター
4. この escapesequence オプションは、直接バイト値のための文法が何もないプログラミング言語でのみ必要です。

6.3.4 フォントを検索

フォントデータのソース 先述のように、フォントは明示的に `PDF_load_font()` メソッドで読み込むこともできますし、あるいは暗黙的に、さまざまなテキスト出力関数に `fontname` オプションを与えて読み込むこともできます。フォントのネイティブな名前を使うこともできますし、フォントデータの場所を決定するために用いられる任意のカスタム名を扱うこともできます。カスタムフォント名は文書内で一意である必要があります。`PDF_info_font()` で、このフォント名は `apiname` キーでクエリーすることができます。

同じフォント名で `PDF_load_font()` を続けて呼び出すと、すべてのオプションがこのメソッドを最初に呼び出した際に与えたものと等しければ、同じフォントハンドルが返されます (扱いが異なるオプションが若干ありますので、詳しくは PDFlib API リファレンスを参照)。そうでなければ、同じフォント名に対して新規のフォントハンドルが作成されます。PDFlib ではフォントデータのソースとして以下に対応しています：

- ▶ ディスクベースまたは仮想のフォントファイル
- ▶ Windows か macOS オペレーティングシステムから持って来たフォント (ホストフォント)
- ▶ PDF 標準フォント：これらは、よく知られた名前の少数の欧文フォントです
- ▶ `PDF_begin_font()` および関連関数群で定義された Type 3 フォント

クックブック 完全なコードサンプルがクックブックの `fonts/font_configuration` トピックにあります。

フォントリソースリストを自動生成 `enumeratefonts` オプションを用いると、検索パス上で得られるすべてのフォントを収集するよう PDFlib に命令することができます (61 ページ「ファイル検索と SearchPath リソースカテゴリー」参照)。`saveresources` オプションを用いると、PDFlib リソースのカレントリストをファイルへ書き出すことができます：

```
/* フォントを検索パスに直接追加 */
p.set_option("searchpath={{C:/fonts} {D:/fonts}}");

/* 検索パス上のすべてのフォントをなめてUPRファイルを作成 */
p.set_option("enumeratefonts saveresources={filename=C:/pdflib.upr}");
```

すると、指定したディレクトリ群の中にあるすべての利用可能なフォントが、生成される UPR ファイルの中にリストされますので、以後使用可能となります。

フォント名エイリアス設定 フォントはそれぞれ、任意の数のエイリアス名を持つことができます。これは、物理フォントへマップする必要がある擬似または仮想名を通じてフォントを要求する場面で有用でしょう。フォント名エイリアスは、以下の例のように、`FontnameAlias` リソースカテゴリー (60 ページの表 3.1 参照) で作成できます：

```
p.set_option("FontnameAlias={sans Helvetica}");
```

左のエイリアス名は、任意に選ぶことができ、そしてこれを用いて、そのフォントをその新しいエイリアス名のもとに読み込むことができます。右の名前は、フォントの有効な API 名、たとえばホストフォントや、フォントリソースカテゴリー `FontOutline` などのうちのひとつでフォントリソースへ紐付けられているフォントの名前である必要があります。

フォントに対する検索順序 PDFlib に与えられるフォント名は名前文字列です。指定された名前がフォント名エイリアスである場合には、それはその対応する API フォント名へ置き換えられます。PDFlib は API フォント名を使って、さまざまな種類のフォントを後述の順序で探します。この検索処理は、使えるフォントがステップの 1 つで見つかるとともに終わります：

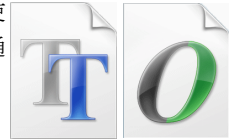
- ▶ フォント名が、**PDF_begin_font()**を用いて生成されていた Type 3 フォントの名前と一致する (131 ページ「6.1.7 Type 3 フォント」を参照)。
- ▶ フォント名が、フォント名を TrueType または OpenType フォントファイルの名前と紐づける **FontOutline** リソース内の名前と一致する。
- ▶ フォント名が、フォント名をシステムにインストールされているフォントの名前と紐づける **HostFont** リソース内の名前と一致する。
- ▶ フォント名が欧文コアフォントの名前と一致する (143 ページ「標準 Type 1 フォント (欧文コアフォント)」参照)。
- ▶ フォント名が、システムにインストールされているホストフォントの名前と一致する (144 ページ「6.3.5 Windows・macOS 上のホストフォント」参照)。
- ▶ フォント名がフォントファイルのベース名 (すなわちファイル名接尾辞を除いた名前) と一致する。

フォントが見つからなかったときは、フォント読み込みは **Font not found** というメッセージを出して止まります。

さまざまなリソースカテゴリについて詳しくは 59 ページ「3.1.4 リソース構成とファイル検索」を参照してください。以下の各項では、さまざまな分類のフォントに対するフォント読み込みについてさらに詳しく解説していきます。

TrueType・OpenType・WOFF/WOFF2 フォント フォント名は、使いたいフォントファイルの名前に対して、**FontOutline** リソースを通じて紐づける必要があります：

```
p.set_option("FontOutline={Arial=/usr/fonts/Arial.ttf}");  
font = p.load_font("Arial", "unicode", "");
```



等号の左側のフォント名 (フォントの API 名といいます) は任意に選べます：

```
p.set_option("FontOutline={f1=/usr/fonts/Arial.ttf}");  
font = p.load_font("f1", "unicode", "");
```



実行時に **PDF_set_option()** で構成するのではなく、UPR ファイル内で **FontOutline** リソースを構成することもできます (59 ページ「3.1.4 リソース構成とファイル検索」参照)。絶対ファイル名を避けるため、**SearchPath** リソースカテゴリを用いることもできます (この場合も、**SearchPath** リソースカテゴリを UPR ファイル内で構成することも可能です)。例：

```
p.set_option("SearchPath={{/usr/fonts}}");  
p.set_option("FontOutline={f1=Arial.ttf}");  
font = p.load_font("f1", "unicode", "");
```

TrueType・OpenType コレクション TrueType または OpenType コレクション (TTC。175 ページ「7.6.1 TrueType・OpenType 日中韓フォントを用いる」参照) ファイル内に含まれているフォントを選ぶには、そのフォントの名前を直接指定します：



```
p.set_option("FontOutline={MS-Gothic=msgothic.ttc}");
font = p.load_font("MS-Gothic", "unicode", "");
```

フォント名は、TTC/OTC ファイル内のすべてのフォントの名前と照合されます。あるいは、TTC/OTC ファイル内の *n* 番目のフォントを選ぶには、フォント名の後にコロンをつけて番号 *n* を指定することができます。この場合は、等号の左側の API フォント名は任意に選べます：

```
p.set_option("FontOutline={f1=msgothic.ttc}");
font = p.load_font("f1:0", "unicode", "");
```

標準 Type 1 フォント（欧文コアフォント） PDFlib は、14 種の標準フォントの集合に対応しています。これらのフォントは、メトリック情報が PDFlib に内蔵されていますので、構成する必要がありません。対応するフォントファイルは、フォントを埋め込みたい場合にのみ必要となります。コアフォントの名前は以下のとおりです：

Courier · *Courier-Bold* · *Courier-Oblique* · *Courier-BoldOblique* ·
Helvetica · *Helvetica-Bold* · *Helvetica-Oblique* · *Helvetica-BoldOblique* ·
Times-Roman · *Times-Bold* · *Times-Italic* · *Times-BoldItalic* ·
Symbol · *ZapfDingbats*

フォント名がリソースを通じていかなるファイル名へも紐づけられていないときは、PDFlib はそのフォントを標準 Type 1 フォントのリストの中で探します。

ホストフォント フォント名がリソースを通じていかなるファイル名へも紐づけられていないときは、PDFlib はそのフォントを、Windows か macOS にインストールされているフォントのリスト内で探します。システムにインストールされているフォントを**ホストフォント**といいます。ホストフォントの名前は ASCII でエンコードされている必要があります。Windows では Unicode も使えます。ホストフォントについて詳しくは 144 ページ「6.3.5 Windows · macOS 上のホストフォント」を参照してください。例：

```
font = p.load_font("Verdana", "unicode", "");
```

Windows では、フォント名の後にカンマをつけてフォントスタイルを追加することもできます（この文法は Latin コアフォントでも使えます）：

```
font = p.load_font("Verdana,Bold", "unicode", "");
```

コアフォントのいずれかの名前でホストフォントを読み込むためには、そのフォント名を、ほしいホストフォントの名前に対して、**HostFont** リソースカテゴリーを通じて紐づける必要があります。以下のコード断片は、内蔵コアフォントデータを使わずに Symbol フォントのデータをホストシステムから取って来るよう指示しています：

```
p.set_option("HostFont={Symbol=Symbol}");
font = p.load_font("Symbol", "unicode", "embedding");
```

フォントファイルを拡張子に基づいて検索 Type 3 以外のすべての種類のフォントは、指定されたフォント名をフォントファイルのベース名（ファイル接尾辞の一切ない名前）として用いて検索することができます。PDFlib は、指定された名前が見つからないときには、**SearchPath** リソースカテゴリー内にリストされているすべてのディレクトリーをなめながら、与えられたファイル名に対して、知られているすべてのファイル名接尾辞を付加することによって、フォントファイルを見つけようと試みます。この拡張子に基づく検索は具体的には以下のようなアルゴリズムです：

- ▶ 以下の接尾辞をフォント名に付加してできるファイル名のフォントファイルがあるかどうかを順番に調べます：

```
.ttf .otf .woff .ttf.woff .otf.woff .woff2 .ttf.woff2 .otf.woff2 .ttc .otc .tte .gai  
.TTF .OTF .WOFF .TTF.WOFF .OTF.WOFF .WOFF2 .TTF.WOFF2 .OTF.WOFF2 .TTC .OTC .TTE .GAI
```

- ▶ 上述の候補ファイル名群を「ありのままに」検索し、ついで、*SearchPath* リソースカテゴリー内で構成されているすべてのディレクトリー名を前につけて検索します。

これはすなわち、手作業で一切構成をしなくても、もしもその対応するフォントファイル名が、フォントの種類に従った標準的なファイル名接尾辞をフォント名に付加した名前から成っており、かつ *SearchPath* ディレクトリーのいずれかの中に置かれているならば、PDFlib はそのフォントを発見するということを意味します。

以下のステートメント群のグループ群は、フォントアウトラインファイルを見つけるうえで同等の効力を持ちます：

```
p.set_option("FontOutline={Arial=/usr/fonts/Arial.ttf}");  
font = p.load_font("Arial", "unicode", "");
```

と

```
p.set_option("SearchPath={{/usr/fonts}}");  
font = p.load_font("Arial", "unicode", "");
```

Type 3 フォント Type 3 フォントは、実行時に、PDFlib のグラフィックメソッド群を用いてグリフを定義することによって定義する必要があります (131 ページ「6.1.7 Type 3 フォント」参照)。*PDF_begin_font()* に与えられたフォント名が、*PDF_load_font()* で要求されたフォント名と一致するときは、そのフォントがフォント検索の最初の段階で選ばれます。例：

```
p.begin_font("PDFlibLogoFont", 0.001, 0.0, 0.0, 0.001, 0.0, 0.0, "");  
  
...  
p.end_font();  
...  
font = p.load_font("PDFlibLogoFont", "logoencoding", "");
```

6.3.5 Windows・macOS 上のホストフォント

macOS・Windows では PDFlib は、オペレーティングシステムにインストールされているフォントを利用することができます。こうしたフォントを**ホストフォント**といいます。手作業でフォントファイルを構成しなくても、そのフォントを単純にシステムにインストールすれば、PDFlib はそれを使用します。

ホストフォントを扱う際には、その正確な、大文字・小文字を区別したフォント名を用いることが重要です。ホストフォント検索は、*PDF_set_option()* のオプション *usehostfonts=false* を用いて無効にすることも可能です。

macOS 上のホストフォント名 フォントを取得するには、標準 TrueType/OpenType フォント名、すなわちフォントの名称テーブルの中の「完全フォント名」を使用します。FontBook アプリケーションを使うことによって、インストールされているフォント群の名前を知ることができます。

Windows 上のホストフォント名 インストールされているフォントの名前は、そのフォントファイルをダブルクリックして、現れるウィンドウのウィンドウタイトルに表示される完全フォント名を見れば知ることができます。フォントによっては、使っている Windows のバージョンに従ってその名前の一部がローカライズされていることもあります。たとえば、フォント名の一部として広く使われている **Bold** は、ドイツ語システム上では翻訳された単語 **Fett** として表示されることがあります。Windows システムからホストフォントデータを取得するには、変換された形のフォント名 (**Arial Fett** など) を PDFlib で用いるか、あるいはフォントスタイル名 (後述) を用いる必要があります。しかし、フォントデータをファイルから直接取得するには、正規の (ローカライズされていない) 形のフォント名 (**Arial Bold** など) を用いる必要があります。

注 この国際化の問題は、ローカライズされた形のフォント名を用いるのではなく、フォントスタイル名 (「**Bold**」など、後述) を付加することによって回避することができます。

Windows のフォントスタイル名 Windows オペレーティングシステムからホストフォントを読み込む際には、PDFlib ユーザーは、Windows のフォント選択機構が提供する機能を利用することができます：太さと斜体についてスタイル名を与えることができます。例：

```
font = p.load_font("Verdana,Bold", "unicode", "");
```

これは Windows に対して、ベースフォントのボールド・イタリックなどある特定のバリエーションを探すよう命令します。得られるフォントによって、Windows は、求められたフォントに最も似通っているフォントを選びます (これは新たなフォントバリエーションを作り出しません)。Windows が見つけたフォントは、求めたフォントとは異なる可能性があり、生成される PDF 内のフォント名は、求めた名前とは異なる可能性があります。PDFlib は、Windows のフォント選択に対していかなる制御もできません。フォントスタイル名はホストフォントでのみ働き、フォントファイルを通じて構成されたフォントに対しては働きません。

以下のキーワード (フォント名とカンマで区切って) は、ベースフォント名に付加してフォントの太さを指定することができます：

```
none, thin, extralight, ultralight, light, normal, regular, medium, semibold, demibold, bold, extrabold, ultrabold, heavy, black
```

このキーワードは大文字・小文字を区別しません。上記のかわりに、あるいは上記に加えて、**italic** キーワードを指定することもできます。2 つのスタイル名を指定する場合には、それぞれをカンマで区切る必要があります。例：

```
font = p.load_font("Verdana,Bold,Italic", "unicode", "");
```

フォントの太さの数値はフォントスタイル名と等価です：

```
0 (none), 100 (thin), 200 (extralight), 300 (light), 400 (normal), 500 (medium), 600 (semibold), 700 (bold), 800 (extrabold), 900 (black)
```

以下の例はフォントの bold バリエーションを選びます：

```
font = p.load_font("Verdana,700", "unicode", "");
```

注 フォントに対する Windows のスタイル名は、フォントスタイルのローカライズされた名前にかかわらずそれを選択しますので、ローカライズされたフォント名を扱う必要があるときには有用でしょう。

Windows のフォント代替 Windows は、特定のレジストリーエントリー群に基づいて、自動的にフォントを代替することがあります。この種のフォント代替は、PDFlib のホストフォント機構にも影響を与えますが、Windows オペレーティングシステムの完全制御下にあります。以下のレジストリーエントリーによって、たとえば、Helvetica フォントが必要なときに Windows がかわりに Arial を届けることもありえます：

```
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\FontSubstitutes
```

Windows のフォント代替に関する詳しい情報については、Microsoft の文書を参照してください。

6.3.6 フォールバックフォント

注 フォールバックフォントは、`encoding=glyphid` の場合には使用できません。

クックブック 完全なコードサンプルがクックブックの `fonts/starter_fallback` トピックにあります。

フォールバックフォントは、フォントとエンコーディングの不足な点を扱う強力なしくみを提供します。複数の用字系のためのフォント群を合成するのにも役立ちます。たとえばアラビア語用字系のためのフォントは、欧文キャラクターに対応していないことがあります。必要なフォント切替が PDFlib によって自動的に行われますので、フォールバックフォントはさまざまな場面でテキスト出力の実現に活用することができます。このしくみは、所与のフォント（ベースフォントといいます）を、このベースフォント内のグリフに加えて他の1つないし複数のフォント内のグリフを使用することによって強化するものです。フォールバックフォントは以下の機能を提供します：

- ▶ ベースフォント内で得られないグリフは自動的に、1 つないし複数のフォールバックフォント内で検索されます。言い換えれば、フォントにグリフを追加することが可能です。複数のフォールバックフォントをベースフォントに対して紐付けることが可能ですので、少なくとも 1 つのフォントが適切なグリフを含んでいる Unicode キャラクターをすべて有効に使うことができます。
- ▶ ある特定のフォールバックフォント内のグリフを用いて、ベースフォント内のグリフをオーバーライドすることができます。すなわち、フォント内のグリフを置き換えることが可能です。1 つないし複数の個別のグリフを置き換えることもできますし、あるいは置き換えたい Unicode キャラクター群の範囲を 1 つないし複数指定することもできます。
- ▶ フォールバックフォントからのグリフの寸法と縦位置は、ベースフォントに合うよう調整できます。

驚くかもしれませんが、ベースフォント自身をフォールバックフォントとして使うことも可能です。これを利用すると以下のトリックが実装できます：

- ▶ そのフォントの一部ないし全部のグリフの寸法か位置を調節。
- ▶ ベースフォントの実際のエンコーディングの外にキャラクターを追加。

フォールバックフォント機構は、`fallbackfonts` フォントオプションによって制御され、すべてのテキスト出力関数に対して効力を持ちます。あらゆるフォントオプションと同様に、`fallbackfonts` オプションは `PDF_load_font()` への明示的な呼び出しで与えることもできますし、あるいは暗黙的フォント読み込みのためのオプションリスト内で与えることもできます。1 つのベースフォントに対しては複数のフォールバックフォントを指定することも可能なことから、`fallbackfonts` オプションは値としてオプションリストのリストをとります。すなわち、中括弧がそのぶん必要です。

`PDF_info_font()` を利用すると、フォールバックフォント機構の結果をクエリーすることができます (153 ページ「6.5.3 コードページ網羅性とフォールバックフォントをクエリー」参照)。

注意 フォールバックフォントを扱う際には以下に留意してください：

- ▶ フォントの組み合わせは必ずしも、タイポグラフィ的に美しい結果を生み出すわけではありません。ベースフォントのグリフデザインに整合するグリフデザインを持つフォールバックフォントだけを使うよう注意を払う必要があります。
- ▶ フォールバックフォントに対するフォントオプションは、`fallbackfonts` オプションリスト内で別途指定する必要があります。
- ▶ 用字系固有のシェーピング (オプション `shaping`・`script`・`locale`) と OpenType 機能 (オプション `features`・`script`・`language`) は、同一フォント内のグリフ群に対してのみ適用され、ベースフォントとフォールバックフォントとにわたるグリフ群に対しては適用されません。
- ▶ 下線 / 上線 / 取り消し線機能は、フォールバックフォントを扱う際には注意して使う必要があります。アセnderなどのタイポグラフィ値についても同様です。ベースフォント内で決定される下線の太さ・位置は、フォールバックフォント内の値とは一致しない可能性があります。その場合、下線の位置または太さが見苦しくガタつくこととなります。この問題に対する単純な回避策は、統一的な値を、`PDF_fit_textline()`・`PDF_add/create_textflow()` の `underlineposition`・`underlinewidth` オプションで指定することです。この値は、ベースフォントとすべてのフォールバックフォントにおいてうまくいくように選ぶ必要があります。

以下の各項で、フォールバックフォントの重要な用途をいくつか解説し、その対応するオプションリストを演示します。

カラー絵文字キャラクターをテキストフォントに追加 絵文字キャラクターはたいていは独立したカラーフォントで利用可能です (128 ページ「6.1.3 SVG・COLR・sbix 形式の OpenType カラーフォント」を参照) ので、テキストフォントと絵文字フォントを切り替える必要があります。絵文字フォントをフォールバックフォントとしてテキストフォントにくっつけるとこれをシンプルにできます。カラーフォントには絵文字キャラクターのモノ版とカラー版が入っている可能性がありますので、カラーグリフが採られるよう `colormode=ignoremono` を用いましょう：

```
fallbackfonts={
  {fontname={Segoe UI Emoji} colormode=ignoremono}
}
```

複数の用字系で使えるようフォントを合体 場合によっては、入力テキストデータの用字系が事前にわからないことがあります。たとえば、データベースが欧文・ギリシャ文字・キリル文字のテキストを含んでいるのに、得られるフォントはこれらの用字系のうちの1つしか同時に網羅していないという場合があるかもしれません。用字系を決定して適切なフォントを選ぶのではなく、いくつかのフォントを結びつけたフォントを構築して、実質的にすべての用字系のスーパーセットを網羅することが可能です。`fallbackfonts` オプションに対して以下のフォントオプションを用いて、ギリシャ文字フォントとキリル文字フォントを欧文フォントに追加することができます：

```
fallbackfonts={
  {fontname=Times-Greek forcechars={U+0391-U+03F5}}
```

```
{fontname=Times-Cyrillic forcechars={U+0401-U+0490}}
}
```

フォント内の一部ないし全部のグリフを大きくする フォールバックフォントを使うと、フォント内の一部ないしすべてのグリフを、文字サイズを変えずに大きくすることができます。この場合も、ベースフォントそれ自身をフォールバックフォントとして用います。この機能は、さまざまなフォントのデザインを、コード内で文字サイズを調整せずに見た目上協調させるのに有用です。*fallbackfonts* オプションに対して以下のフォントオプションを用いて、指定した範囲内のすべてのグリフを 120% へ大きくすることができます：

```
fallbackfonts={
  {fontname=Times-Italic forcechars={U+0020-U+00FF} fontsize=120%}
}
```

拡大したピクトグラムを追加 *fallbackfonts* オプションに対して以下のフォントオプションを用いて、ZapfDingbats フォントから記号を持って来ることができます：

```
fallbackfonts={
  {fontname=ZapfDingbats forcechars=.a12 fontsize=150% textrise=-15%}
}
```

この場合も、*fontsize*・*textrise* サブオプションを用いて、記号のサイズと位置をベースフォントに合わせています。

日中韓フォント内のグリフを置き換え *fallbackfonts* オプションに対して以下のフォントオプションを用いて、ASCII 範囲内の欧文キャラクターを別フォントからのものに置き換えることができます：

```
fallbackfonts={
  {fontname=Courier-Bold forcechars={U+0020-U+007E}}
}
```

アラビア語フォントに欧文キャラクターを追加 この用途は 173 ページ「7.5.4 アラビア語テキスト組版」で解説しています。

足りないグリフをつきとめる 無償提供されているフォント *Unicode BMP Fallback SIL* は、実際のグリフでなく各 Unicode キャラクターの 16 進値を表示します。このフォントは、ワークフローにおけるフォント関連の問題を診断するのに有用なときがあります。*fallbackfonts* オプションに対して以下のフォントオプションを用いて、任意のフォントを、足りないキャラクターが視覚化されるようこの特殊なフォールバックフォントで強化することができます：

```
fallbackfonts={{fontname={Unicode BMP Fallback SIL}}}
```

フォントに外字キャラクターを追加 この用途は 176 ページ「7.6.3 EUDC・SING フォントによる外字キャラクター」で解説しています。

6.4 フォントの埋め込みとサブセット化

6.4.1 埋め込み

Acrobat における PDF のフォント埋め込みとフォント置換 PDF 文書は、正しいテキスト表現を確保するために、フォントデータをさまざまな形式で含むことができます。あるいは、グリフのメトリックといくつかの一般的なフォント情報だけを含む（グリフのアウトライン本体を含まない）フォント記述子を埋め込むこともできます。フォントが PDF 文書に埋め込まれていない場合、PDF ビューアはそれがターゲットシステムで得られ、かつ構成されていればそれを取り（Acrobat では「ローカルフォントを用いる」）、あるいはフォント記述子に従って代替フォントを組み立てようと試みます。代替フォントが使われることによってテキストは読めるようになりますが、そのグリフは元のフォントとは異なる可能性があります。同様に、代替フォントは、複雑用字系のシェーピングか OpenType レイアウト機能が使われているときには働きません。こうした理由から、一般にはフォントの埋め込みを推奨します。ただし、文書がフォントを埋め込んでいなくてもターゲットシステム上での表示が許容範囲内になるとわかっている場合は例外です。そのような PDF ファイルは本質的に非可搬ですが、すべてのワークステーション上で必要フォントが得られるとわかっている企業ネットワークなどの制御された環境においては役に立つかもしれません。一般に、フォントの埋め込みは、推奨され、PDFlib ではデフォルトの動作です。

フォントを PDFlib で埋め込む フォントの埋め込みは、デフォルトで有効ですが、*embedding* フォントオプションを用いて制御することもできます：

```
font = p.load_font("WarnockPro", "winansi", "noembedding");
```

不可視テキスト（主に OCR 出力に有用）にのみ使われるフォントに対するフォント埋め込みは、*optimizeinvisible* フォントオプションで制御することができます。

6.4.2 サブセット化

PDF 出力のサイズを減らすために、PDFlib は、あるフォントの中で実際にその文書の中で使われているグリフだけを埋め込むことができます。この処理をフォントのサブセット化といいます。サブセット化を行うと新しいフォントが作成され、その中ではグリフの数が元のフォントよりも少なく、PDF の表示に必要なフォント情報も省略されています。Type 3 フォントはサブセット化について特別な処理を要します（150 ページ「Type 3 フォントのサブセット化」を参照）。

サブセット化を要求されているフォントが文書内で使われている場合、PDFlib は実際にテキスト出力に使われているグリフを調べます。サブセット化の動作を制御するにはいくつかの方法があります：

- ▶ デフォルトのサブセット化の動作は *autosubsetting* オプションで制御されます。それが *true* ならサブセット化が有効になります。デフォルト値は *true* です。
- ▶ *autosubsetting=true* の場合：*subsetlimit* オプションはパーセント値を持ちます。文書内で用いられている、あるフォント内のグリフの数がこの割合を超える場合には、サブセット化は無効となり、フォント全体が埋め込まれます。これによって処理時間がある程度短縮できますが、そのかわりに出力ファイルの容量は大きくなります。以下のフォントオプションはサブセット限界を 75% に設定します：

```
subsetlimit=75%
```

`subsetlimit` のデフォルト値は 100 パーセントです。言い換えれば、クライアントが明示的に 100 パーセント未満の限界値を要求しない限り、`PDF_load_font()` で要求されるサブセット化オプションは効力を持ちます。

初期フォントサブセット フォントサブセットは、文書内で使われているすべてのグリフのアウトライン記述を含んでいます。これはすなわち、生成される文書サブセットは文書ごとに変化することを意味します。なぜなら一般に、各文書内ではそれぞれ異なるグリフの集合が用いられているからです。フォントサブセットを埋め込んだたくさんの小さな文書を大きな文書へ連結する際には、フォントサブセットがそれぞれまちまちであることは厄介です:埋め込まれているサブセットは全部互いに異なるため、除去できないのです。

こうした場合のために、PDFlib では、`PDF_load_font()` の `initialsubset` オプションでフォントサブセットの初期内容を指定することが可能です。PDFlib がデフォルトでは空のサブセットから開始し、生成されるテキスト出力からの要請に応じてグリフを追加していくのに対して、`initialsubset` オプションを利用すると、空でないサブセットを指定することができます。たとえば、Latin-1 テキスト出力のみが生成されるとわかっている場合に、フォントがその他のグリフもたくさん含んでいるならば、先頭 Unicode ブロックを初期サブセットとして指定することができます:

```
initialsubset={U+0020-U+00FF}
```

これはすなわち、指定した範囲内のすべての Unicode キャラクターに対するグリフがサブセット内に入れ込まれることを意味します。この範囲を、生成される文書内のすべてのテキストを網羅するように選んでおいたならば、生成されるフォントサブセットはすべての文書内で等しくなるでしょう。そうすれば、このような文書群を後で 1 個の PDF へ連結する際に、等しいフォントサブセット群は `PDF_begin_document()` の `optimize` オプションで除去することが可能になります。

Type 3 フォントのサブセット化 Type 3 フォントを文書で使えるようにするには、その前にまず (グリフの幅が必要なので) それを定義しなければなりません。ところがその一方でサブセット化は、すべてのテキスト出力を作成した後ではじめて可能になるものです (サブセットを生成するには、どのグリフが文書内で使われたかを知る必要があるのです)。この矛盾を避けるために PDFlib は、幅オンリー Type 3 フォントに対応しています。Type 3 フォントについてサブセット化が必要なときは、2 回に分けてフォントを定義する必要があります。詳しくは PDFlib API リファレンスを参照してください。

クックブック 完全なコードサンプルがクックブックの `type3_fonts/type3_subsetting` トピックにあります。

6.5 フォント情報をクエリー

`PDF_info_font()` を利用すると、フォント・エンコーディング・Unicode・グリフに関して有用な情報をクエリーすることができます。クエリーの種類によっては、有効なフォントハンドルがこのメソッドの引数として必要な場合もあります。以下すべての例で、表 6.4 に挙げる変数を用いることにします。

表 6.4 `PDF_info_font()` の利用例で用いる変数

変数	注釈
<code>int uv;</code>	Unicode の数値。あるいは、グリフ名参照から「&」・「:」修飾を除いたものをオプションリスト内で用いることもできます (<code>unicode=euro</code> など)。詳しくは、PDFlib API リファレンスの <code>Unichar</code> オプションリストデータ型の解説を参照してください。
<code>int c;</code>	8 ビット文字コード
<code>int gid;</code>	グリフ ID
<code>int cid;</code>	CID 値
<code>String gn;</code>	グリフ名
<code>int gn_idx;</code>	グリフ名の文字列番号。 <code>gn_idx</code> が -1 以外するとき、その対応する文字列を取得するには以下のようにします： <code>gn = p.get_string(gn_idx, "");</code>
<code>String enc;</code>	エンコーディング名
<code>int font;</code>	<code>PDF_load_font()</code> を用いて作成した有効なフォントハンドル

求められたキーワードとオプション（群）の組み合わせが得られないときは、`PDF_info_font()` は -1 (PHP では 0) を返します。これは、クライアントアプリケーション側でチェックする必要があり、またこれを用いて、求めるグリフがフォント内にあるかどうかを知ることができます。

以下のサンプルコード行は、互いに依存していませんので、独立に抜き出して利用することができます。

6.5.1 フォント非依存のエンコーディング・Unicode・グリフ名クエリー

エンコーディングクエリー エンコーディングクエリーには、有効なフォントハンドルは必要ではありません。すなわち、`PDF_info_font()` の `font` 引数に値 -1 (PHP の場合 : 0) を与えることができます。`gn` には、PDFlib が内部的に知っているグリフ名だけを与えることができ、フォント独自のグリフ名を与えることはできません。

8 ビットコードか指名したグリフの、8 ビットエンコーディング内の Unicode 値をクエリー：

```
uv = (int) p.info_font(-1, "unicode", "code=" + c + " encoding=" + enc);
uv = (int) p.info_font(-1, "unicode", "glyphname=" + gn + " encoding=" + enc);
```

Unicode・グリフ名クエリー `PDF_info_font()` を利用すると、特定の 8 ビットエンコーディングに依存せず、Unicode 値と PDFlib が内部的に知っている名前との関係にかかわるクエリーを行うことも可能です。これらのクエリーはいかなるフォントにも依存しませんので、有効なフォントハンドルは必要ではありません。

内部的に知られているグリフ名の Unicode 値をクエリー :

```
uv = (int) p.info_font(-1, "unicode", "glyphname=" + gn + " encoding=unicode");
```

Unicode 値の内部グリフ名をクエリー :

```
gn_idx = (int) p.info_font(-1, "glyphname", "unicode=" + uv + " encoding=unicode");
```

```
/* 文字列番号を用いて実際のグリフ名を取得 */  
gn = p.get_string(gn_idx, "");
```

6.5.2 フォント依存のエンコーディング・Unicode・グリフ名クエリー

以下のクエリーは、特定のフォントにかかわるものですので、有効なフォントハンドルでフォントを指定する必要があります。*gn* 変数を用いて、内部的に知られているグリフだけでなく、フォント独自のグリフ名を与えることもできます。以下すべての例において、戻り値 -1 は、求めたグリフをそのフォントが含んでいないことを意味します。

フォント内のコード、グリフ ID、指名したグリフ、CID に対する Unicode 値をクエリー :

```
uv = (int) p.info_font(font, "unicode", "code=" + c);  
uv = (int) p.info_font(font, "unicode", "glyphid=" + gid);  
uv = (int) p.info_font(font, "unicode", "glyphname=" + gn);  
uv = (int) p.info_font(font, "unicode", "cid=" + cid);
```

フォント内のコード、Unicode 値、指名したグリフ、CID に対するグリフ ID をクエリー :

```
gid = (int) p.info_font(font, "glyphid", "code=" + c);  
gid = (int) p.info_font(font, "glyphid", "unicode=" + uv);  
gid = (int) p.info_font(font, "glyphid", "glyphname=" + gn);  
gid = (int) p.info_font(font, "glyphid", "cid=" + cid);
```

コード・Unicode 値・CID で指定したグリフのフォント固有の名前をクエリー :

```
gn_idx = (int) p.info_font(font, "glyphname", "code=" + c);  
gn_idx = (int) p.info_font(font, "glyphname", "unicode=" + uv);  
gn_idx = (int) p.info_font(font, "glyphname", "glyphid=" + gid);  
gn_idx = (int) p.info_font(font, "glyphname", "cid=" + cid);
```

```
/* 文字列番号を用いて実際のグリフ名を取得 */  
gn = p.get_string(gn_idx, "");
```

グリフの入手可能性をチェック *PDF_info_font()* を利用すると、自分のアプリケーションに必要なグリフをある特定のフォントが含んでいるかどうかをチェックすることができます。たとえば、以下のコードはユーログリフがフォント内に含まれているかどうかをチェックします :

```
/* "unicode=U+20AC"でもよい */  
if (p.info_font(font, "code", "unicode=euro") == -1)  
{  
    /* ユーログリフに対するグリフがフォント内で得られない */  
}
```

クックブック 完全なコードサンプルがクックブックの `fonts/glyph_availability` トピックにあります。

あるいは *PDF_info_textline()* を使って、所与のテキスト文字列におけるマップなしキャラクターの数を、すなわち文字列内の、フォント内で適当なグリフが得られないキャラク

ターの数をチェックすることもできます。以下のコード断片は、ユーロキャラクター（グリフ名参照で表現）1個だけを内容とする文字列についての結果をクエリーします。もしもマップなしキャラクターが1個見つければ、これはすなわちそのフォントがユーロ記号に対するグリフを一切含んでいないことを意味します：

```
String optlist = "font=" + font + " charref";

if (p.info_textline("&euro;", "unmappedchars", optlist) == 1)
{
    /* ユーロ記号に対するグリフはフォント内で得られない */
}
```

6.5.3 コードページ網羅性とフォールバックフォントをクエリー

PDF_info_font() を利用すると、ある特定の言語ないし用字系のテキスト出力を作成するのにフォントが適しているかどうかをチェックすることもできます。そのためには、そのテキストでどのコードページが必要かがわかっている必要があります。コードページ網羅性は、フォントの OS/2 テーブル内にエンコードされています。ただし、フォントがある特定のコードページに対応しているとは正確には何を意味するのかは、フォントデザイナーの考え方で決まります。フォントがある特定のコードページに対応しているからといって、必ずしもそれがそのコードページ内のすべてのキャラクターに対するグリフを含んでいるとは限りません。より正確な網羅性情報が必要な場合は、152 ページ「6.5.2 フォント依存のエンコーディング・Unicode・グリフ名クエリー」で示したようにしてすべての必要なキャラクターの入手可能性をクエリーすることができます。

フォントがコードページに対応しているかどうかをチェック 以下のコード断片は、フォントがある特定のコードページに対応しているかどうかをチェックします：

```
String cp="cp1254";

result = (int) p.info_font(font, "codepage", "name=" + cp);

if (result == -1)
    System.err.println("コードページ網羅性不明");
else if (result == 0)
    System.err.println("コードページはこのフォントでは対応していません");
else
    System.err.println("コードページはこのフォントで対応しています");
```

対応している全コードページのリストを取得 以下のコード断片は、TrueType または OpenType フォントが対応しているすべてのコードページのリストをクエリーします：

```
cp_idx = (int) p.info_font(font, "codepagelist", "");

if (cp_idx == -1)
    System.err.println("コードページリスト不明");
else
{
    System.err.println("コードページリスト:");
    System.err.println(p.get_string(cp_idx, ""));
}
```

これは、広く用いられている Arial フォントに対しては以下のリストを作成します：

cp1252 cp1250 cp1251 cp1253 cp1254 cp1255 cp1256 cp1257 cp1258 cp874 cp932 cp936 cp949
cp950 cp1361

予備グリフをクエリー `PDF_info_font()` を利用すると、フォールバックフォント機構の結果をクエリーすることができます (フォールバックフォントについて詳しくは 146 ページ「6.3.6 フォールバックフォント」を参照)。以下のコード断片は、指定した Unicode キャラクターを表すのに用いられているベースフォントかフォールバックフォントの名前を調べます：

```
result = p.info_font(basefont, "fallbackfont", "unicode=U+03A3");
/* result==ベースフォントならば、ベースフォントが使われフォールバックフォントは必要なかった */
if (result == -1)
{
    /* キャラクターはベースフォントでもフォールバックフォントでも表示できない */
}
else
{
    idx = p.info_font(result, "fontname", "api");
    fontname = p.get_string(idx, "");
}
```

7 テキスト出力

7.1 テキスト出力方式

PDFlib は、テキスト出力にいくつかのレベルで対応しています：

- ▶ **PDF_show()** や類似の関数群による低レベルテキスト出力。
- ▶ **PDF_fit_textline()** による一行に組まれたテキスト出力。このメソッドはパス上テキストにも対応しています。
- ▶ テキストフローによる複数行テキスト組版出力 (**PDF_fit_textflow()** および関連する関数群)。テキストフロー組版機能は、ベクトルベースの形状の内側または外側にテキストを回りこませることもできます。
- ▶ 表内のテキスト。表組版機能は、表セル内のテキスト行・テキストフロー内容に対応しています。

低レベルテキスト出力 **PDF_show()** のような関数群を使うと、テキストをページ上のある特定の場所に、いかなる組版支援をも利用せずに配置することができます。これは、非常に基本的な出力要請を持つアプリケーション（プレーンテキストファイルを PDF へ変換するなど）、あるいはすでに完全なテキスト配置情報を持っているアプリケーションの場合にのみ推奨します（別形式のページを PDF へ変換するドライバーなど）。以下のコード断片は、低レベル関数群でテキスト出力を作成します：

```
font = p.load_font("Helvetica", "unicode", "");  
  
p.setfont(font, 12);  
p.set_text_pos(50, 700);  
p.show("Hello world!");  
p.continue_text("says Java");
```

テキスト行で組まれた一行テキスト出力 **PDF_fit_textline()** は、一行だけのテキスト出力を作成し、さまざまな組版機能を提供します。ただし、テキスト行ごとの位置はクライアントアプリケーションが決定する必要があります。以下のコード断片は、テキスト行でテキスト出力を作成します。フォント・文字サイズはオプションとして指定できますので、これに先立って **PDF_load_font()** を呼び出しておく必要はありません：

```
p.fit_textline(text, x, y, "fontname=Helvetica fontsize=12");
```

詳しくは 229 ページ「9.1 テキスト行を配置・はめ込む」を参照してください。

テキストフローによる複数行テキスト出力 **PDF_fit_textflow()** は、任意の行数のテキスト出力を作成し、また、テキストを複数の段組みまたはページにわたらせることもできます。テキストフロー組版機能はたくさんの組版機能に対応しています。以下のコード断片は、テキストフローを用いてテキスト出力を作成します：

```
tf = p.add_textflow(tf, text, optlist);  
result = p.fit_textflow(tf, llx, lly, urx, ury, optlist);  
p.delete_textflow(tf);
```

詳しくは 238 ページ「9.2 複数行のテキストフロー」を参照してください。

表内のテキスト テキスト行とテキストフローを使って、表セル内にテキストを配置することもできます。詳しくは 259 ページ「9.3 表の組版」を参照してください。

7.2 フォントメトリックとテキストスタイル

7.2.1 フォントとグリフのメトリック

テキスト位置 PDFlib はテキスト位置を、グラフィック描画のカレント点とは独立に保持します。前者は `textx/texty` オプションでクエリーでき、後者は `currentx/currenty` でクエリーできます。

グリフのメトリック PDFlib では、PDF で用いられているグリフとフォントのメトリックの体系を用いています。ここで簡単に説明しておきましょう。

PDFlib のユーザーが指定する必要がある文字サイズというのは、テキストの行と行の間で文字が重なりあわないために必要な最小間隔のことです。文字サイズは一般にフォント内の各文字よりも大きくなっています。なぜならその中にはベースラインより上の部分も下の部分も含んでいるからであり、また、それに加えて行と行の間の間隔をもっと広くとっていることもあるからです。

leading (行送り) は、テキストの 1 つの行のベースラインと次の行のベースラインとの間の縦の間隔を指定します。デフォルトではこれは文字サイズと同じ値に設定されています。**capheight** (キャップハイト) は、多くの欧文フォントでは *T* や *H* のような大文字の高さです。**xheight** (*x* ハイト) は、多くの欧文フォントでは *x* のような小文字の高さです。**ascender** (アセンダー) は、多くの欧文フォントでは *f* や *d* のような小文字の高さです。**descender** (ディセンダー) は、多くの欧文フォントでは、ベースラインから *j* や *p* のような小文字の下端までの間隔です。ディセンダーはふつう負の値です。**xheight · capheight · ascender · descender** の値は文字サイズに対する割合として表されているので、必要な文字サイズを掛けてから用いる必要があります。

gaplen プロパティは TrueType・OpenType フォントでのみ得られます (それ以外のフォントでは算出されます)。**gaplen** 値はフォントファイルから読み出され、ベースライン間の推奨間隔とアセンダー+ディセンダーとの差を示します。

PDFlib は、これらの値のうちの 1 つないし複数を、推算で求めなければならない場合もあります。なぜならこれらの値は、フォントファイルの中に存在しているという保証がないためです。用いられている値が本当の値なのか、それとも推測値なのかを知るには、**PDF_info_font()** を呼び出してオプション **faked** で **xheight** をクエリーします。あるフォントのキャラクターメトリックを PDFlib でクエリーするには以下のように記述します：

```
font = p.load_font("Times-Roman", "unicode", "");  
  
capheight = p.info_font(font, "capheight", "");  
ascender = p.info_font(font, "ascender", "");  
descender = p.info_font(font, "descender", "");  
xheight = p.info_font(font, "xheight", "");
```

注 上付き・下付きの位置とサイズを、PDFlib を用いてクエリーすることはできません。

クックブック 完全なコードサンプルがクックブックの `fonts/font_metrics_info` トピックにあります。

7.2.2 カーニング

さまざまな文字の組み合わせのなかには、望ましくない見ばえになってしまうものがあります。たとえば、2 つの *V* が隣り合うと *W* のように見えてしまいますし、*T* と *e* の間の間隔は縮めないで広くあきすぎて不恰好になってしまいます。このような補正のことをカーニングといいます。多くのフォントが、問題となる文字の組み合わせそれぞれに対する間

隔調整を指定した包括的なカーニング情報を持っています。PDFlib には、カーニングの動作を制御する方式が 2 種類あります：

- ▶ デフォルトでは、フォント内のカーニング情報はそのフォントを読み込む際に読み取られます。カーニングが必要でない場合は、`PDF_load_font()` で `readkerning` オプションを `false` に設定します。
- ▶ テキスト出力に対するカーニングは、テキスト出力関数群が対応している `kerning` テキスト書式オプションで有効にする必要があります。

一時的にカーニングを無効にすることがたとえばどんなときに有用かということ、数表を組みたいときに、カーニングデータが数字どうしのペアを含んでいる場合というのが挙げられます。カーニングされた数字は表内できれいに並ばないからです。なお、今どきの TrueType・OpenType フォントはこの目的のための特殊な数字を含んでおり、これは等幅数字レイアウト機能とオプション `features={tnum}` で利用できます。

カーニングは、どのような字間・単語間隔・横伸縮が指定されていたとしても、それに加えて適用することができます。PDFlib では、1 つのフォント内のカーニングペアの数についてはまったく無制限です。

カーニングは、フォントオプション `readkerning` とテキストオプション `kerning` によって制御されます。デフォルトではカーニングは有効になっています。

7.2.3 テキストスタイル

擬似ボールドフォント PDFlib は、個別のテキスト文字列について、`fakebold` オプションを通じて擬似的なボールドテキストを作成する機構をサポートしています。この方式は、グリフの輪郭を描線することによってボールドフォントを擬似表現しています：Type 3 フォントの場合には、テキストがさまざまな変位で複数回配置されます。強調のためには本当のボールドフォントを使用することを強く推奨します。`fakebold` オプションは、本当のボールドテキストよりも劣るテキスト出力を作成しますし、また、テキスト抽出を阻害するおそれもあります。

クックブック 完全なコードサンプルがクックブックの `fonts/simulated_fontstyles` トピックにあります。



Tele Vaso

図 7.1 カーニング

カーニングなし

Tele Vaso

カーニング適用

Te Va

カーニングによる文字移動

擬似斜体フォント `italicangle` オプションを使って、レギュラーフォントしか利用できないときにイタリックフォントのような効果を出すこともできます。この方式は、偽イタリックフォントを作成するために、レギュラーフォントをユーザーから与えられた角度だけ傾けるものです。負の値でテキストは右に傾きます。もちろん、本物のイタリックや斜体フォントを使ったほうがはるかにきれいな出力が得られることを忘れてはいけません。しかしイタリックフォントが入手できないときに `italicangle` オプションを使えば、簡単にそれに似た効果を出すことができます。この機能は特に日中韓フォントで便利です。`italicangle` オプションの値は普通 -12 から -15 度の範囲です。

注 PDFlib はグリフ幅を、斜体化したグリフの新しい外接枠には合わせません。たとえばテキストを均等配置するとき、斜体化したグリフははめこみ枠からはみ出す可能性があります。

影付きテキスト PDFlib は、同じテキストを場所を少しずつずらして複数個印字することによって影付き効果を作成することができます。影付きテキストは `PDF_fit_textline()`・`PDF_add/create_textflow()` の `shadow` オプションで作成できます。影の色や、その主テキストに対する相対位置、図形ステータスオプション群は、サブオプションで指定できます。

下線・上線・取り消し線付きテキスト PDFlib では、テキストの下や上や中央に線をひくことができます。線幅やベースラインからの間隔は、フォントのメトリック情報に基づいて計算されます。また、横伸縮やテキストマトリックスのカレント値も線幅の計算には加味されます。下線・上線・取り消し線の有無を切り替えるには、それぞれ `PDF_set_text_option()` で `underline`・`overline`・`strikeout` オプションを、またはテキスト出力関数群でその対応するオプションをオンオフします。`underlineposition`・`underlinewidth` オプションを使って微調整もできます。

線の色には `strokecolor` オプションが用いられます。しかし、カレントの `linecap` オプションは無視されます。`decorationabove` オプションは、線がテキストの上と下のいずれに引かれるかを制御します。体裁上の注意：多くのフォントでは、下線は文字のベースラインより下の部分とぶつかってしまい、また、上線は文字の上の付加記号とぶつかってしまいます。

クックブック 完全なコードサンプルがクックブックの `text_output/starter_textline` トピックにあります。

テキスト表現モード PDFlib は、テキストの見栄えを変更する表現モードにいくつか対応しています。これを用いると、テキストの輪郭を描いたり、テキストをクリッピングパスとして利用したりすることができます。また、テキストを不可視にすることもでき、これはたとえば、スキャン画像の上にテキストを乗せてテキスト検索や索引生成を可能にしつつ、テキスト自体は隠す、といった活用ができるでしょう。表現モードの一覧は *PDFlib API リファレンス* に挙げてあります。表現モードは、`textrendering` テキストオプションで設定することができます。

テキストを描線すると、`strokewidth`・`strokecolor` といったテキストステータスのオプション群がグリフの輪郭に適用されます。表現モードは、Type 3 フォントを用いて印字されるテキストには何の効果も持ちません。

クックブック 完全なコードサンプルがクックブックの `text_output/text_as_clipping_path`・`text_output/invisible_text` トピックにあります。

テキストの色 テキストは通常、*fillcolor* オプションで指定された色で印字されます。塗り色は *PDF_setcolor()* を用いて設定できます。ただし、表現モードで 0 以外が選択されている時は、その選択されている表現モードによって、*strokecolor* と *fillcolor* はどちらもテキストに対して効力を持ちます。

クックブック 完全なコードサンプルがクックブックの `text_output/starter_textline` トピックにあります。

7.3 絵文字シーケンス

クックブック 絵文字シーケンスの例がクックブックの `text_output/emoji_sequences` トピックにあります。

絵文字キャラクターの見目はさまざまなシーケンスを用いて修飾できます。たとえば絵文字キャラクターの肌の色を変えたりすることが可能です。その基礎となっているコンセプトは Unicode Technical Standard #51 『Unicode Emoji』で解説されており、網羅的なテーブルが以下にあります：

unicode.org/Public/emoji/14.0/emoji-zwj-sequences.txt
unicode.org/Public/emoji/14.0/emoji-sequences.txt

その実装は OpenType 機能定義に基づいていますので、絵文字シーケンスに対応しているかどうかはフォントによって異なります。無料で利用可能な *EmojiOneColor* フォントや Windows フォント *Segoe UI Emoji* など今どきのカラーフォントは多様な絵文字の組み合わせに対応しています。絵文字シーケンスは単色フォントでも使用できますが、最良の効果はカラーフォントによって実現できます。

あるフォントがある特定のシーケンスに対応している場合には、PDFlib のテキスト出力メソッド群は、それに対応する効果を、オプションが特に指定されていなくてもデフォルトでレンダーします（シーケンスは *ccmp* OpenType 機能を用いて実現されており、それはデフォルトで有効になっています）：

- ▶ 絵文字のゼロ幅接合子シーケンス：複数人のグループ化、役割、性別、髪など修飾を行います。U+200D ZERO WIDTH JOINER (ZWJ) を使用します。このキャラクターは PDFlib ではテキストオプション *charref=true* を設定している場合には文字参照 *‍* として指定できます。
- ▶ 肌色修飾子：絵文字の肌の色を調節します。U+1F3FB EMOJI MODIFIER FITZPATRICK TYPE 1-2 ~ U+1F3FF EMOJI MODIFIER FITZPATRICK TYPE 6 を使用します。これらは肌色シーケンス外で使用された場合にはたいがいカラーパッチとしてレンダーされます。
- ▶ 絵文字の旗シーケンス：地域標識キャラクター U+1F1E6 REGIONAL INDICATOR SYMBOL LETTER A などから旗を生成します。
- ▶ 絵文字のキーキャップシーケンス：数字と U+FE0F EMOJI PRESENTATION SELECTOR (VS16) と U+20E3 COMBINING ENCLOSING KEYCAP を使用します。

絵文字シーケンスの例を表 7.1 にいくつか示します。

表 7.1 絵文字修飾シーケンス

シーケンスの種類	絵文字シーケンス	単離状態のキャラクター	合体されたキャラクター
ZWJ のグループ化	👨‍👩‍👧		
肌の色	👦🏻 👦🏽 👦🏿		
絵文字の旗	🇲🇹 🇬🇧		
絵文字のキーキャップ	o️⃣ 9️⃣		
絵文字の性別	🧜‍♂ 🧜‍♀		
肌の色と性別	🧜🏿‍♂ 🧜🏿‍♀		
絵文字の髪構成要素	👨‍🦰 👨‍🦳		
ZWJ の役割 : 男性	👨‍🌾		
ZWJ の役割 : 女性	👩‍🌾		

7.4 OpenType レイアウト機能

クックブック 完全なコードサンプルがクックブックの `fonts/starter_opentype`・`fonts/opentype_feature_tester` トピックにあります。

7.4.1 対応している OpenType レイアウト機能

PDFlib は、いくつかのフォント内の追加情報に従った高度なテキスト出力に対応しています。これらのフォント拡張を OpenType レイアウト機能といいます。たとえば、フォントが *liga* 機能を含んでいるかもしれません。この機能は、*f・f・i* グリフは結合して合字を形作れるという情報を含んでいます。他によく利用される例としては、*smcp* 機能によるスモールキャップス、すなわち通常の大文字キャラクターよりも小さな大文字キャラクターや、*onum* 機能によるオールドスタイル数字、すなわちアセンダーとディセンダーを持つ数字（ベースライン上にすべて配置される横並びの数字とは異なる）などが挙げられます。合字は非常に広く利用される OpenType 機能ではありますが、多数の機能のうちの 1 つにすぎません。PDFlib は以下のグループの OpenType 機能に対応しています：

- ▶ 欧文タイポグラフィーのための OpenType 機能群を表 7.2 に挙げます。これらは *features・script・language* オプションで制御されます。
- ▶ 複雑用字系のための OpenType 機能群。これらは *shaping・script* オプションで制御されます（170 ページ「7.5 複雑用字系の出力」を参照）。
- ▶ カーニングのための OpenType 機能テーブル群。PDFlib は *kerning* を OpenType 機能としては扱いません。なぜならカーニングデータは OpenType 機能テーブル以外の手段でも表現することができるからです。カーニングを制御するには、*readkerning* フォントオプションと *kerning* テキストオプションを用いてください（157 ページ「7.2.2 カーニング」を参照）。
- ▶ 以上に加え、日中韓テキストのための OpenType 機能については 177 ページ「7.6.4 OpenType レイアウト機能と日中韓テキスト出力」で解説します。

以下の Web サイトを使って、フォントの持つ OpenType 機能を調べることもできます：

wakamaifondue.com/

注 カーニング以外、PDFlib は GPOS テーブルに基づく OpenType 機能に対応していません。OpenType レイアウト機能は 8 ビットエンコーディングに対しては使えません。

表 7.2 対応している、欧文タイポグラフィと絵文字のための OpenType 機能（日中韓テキスト用の OpenType 機能は表 7.9 に挙げます）

タグ	名前	説明
合字・字体		
aalt	すべての字体にアクセス	選択した1つのキャラクターのすべての異体字をアクセス可能にします。たとえば、通常の置き換えによってサポートされるコンテキストの外にあるグリフをユーザーが必要とするかもしれず、あるいは、ほしいグリフを印字させるのは何の機能かをユーザーが知らないかもしれません。
calt ¹	コンテキスト字体	デフォルトグリフを、より良い連結動作を与える字体へ置き換え。一部ないし全部のグリフを連結させるようデザインされた用字系書体で使用されます。
clig ¹	コンテキスト合字	グリフ列を、タイポグラフィ的観点からより好ましい1個のグリフへ置き換え。他の合字機能群と異なり、cligは、その合字が推奨されるコンテキストを指定します。この能力は、いくつかの用字系デザインにおいて、またスワッシュ合字に対して重要です。
csw	コンテキストスワッシュ	デフォルトグリフを、指定されたコンテキストにおいて、スワッシュへ置き換え。
cv01 ~ cv99	キャラクターの異体字 01-99	フォントの中で、1つないし複数のキャラクターに対して様式的異体字群があるが、1つのキャラクターに対する異体字群が他のキャラクター群に対する異体字群とシステムチックに関連していないことがあります。あるいは、1つの大文字キャラクターとその小文字（または、1つのキャラクターとそれにダイアクリティカルマークをあらかじめつけたキャラクター）に対して1つの異体が存在するが、それは他の無関係なキャラクター群には該当しないことがあります。
dlig	随意合字	グリフ列を、タイポグラフィ的観点からより好ましい1個のグリフへ置き換え。dligは、ユーザーの好みによって、特殊効果のために用いることができる合字をカバーします。
hist	歴史的字体	デフォルトの字体を歴史的字体へ置き換え。いくつかの字の字体は過去には広く用いられていましたが、現在では時代遅れに見えます。
hlig	歴史的合字	デフォルトの合字を歴史的合字へ置き換え。
liga ¹	標準合字	グリフ列を、タイポグラフィ的観点からより好ましい1個のグリフへ置き換え。ligaは、通常の場合において用いられるべき合字をカバーします。
locl ¹	ローカライズ字体	グリフのデフォルト字体からローカライズ字体への置換を有効にします。この機能は script・language オプションを必要とします。
salt	スタイリスティック字体	デフォルト字体をスタイリスティック字体へ置き換え。これらの異体字は、スワッシュや歴史的といった決まった分類には必ずしもあてはまりません。
ss01 ~ ss20	スタイリスティック集合 01 ~ 20	個々のグリフのスタイリスティック字体（salt 機能を参照）に加えて、あるいはそれに替えて、フォントによっては、その文字集合の1個ないし複数の一部分に対応するスタイリスティック異体字グリフ群の集合を含んでいるものがあります。例：欧文フォント内の小文字に対して複数の異体字。
subs	下付き	デフォルトグリフを下付きグリフへ置き換え。
sup	上付き	横並びまたはオールドスタイル数字を上付き数字へ置き換え（主に脚注表示用）、小文字を上付き文字へ置き換え（主にフランス語敬称の省略形用）。
swsh	スワッシュ	デフォルトグリフをスワッシュグリフへ置き換え。
titl	見出し化	デフォルトグリフを、見出し用にデザインされた字体へ置き換え。
数字・数学		
afr	別形式分数	スラッシュで区切られた数字群を別形式へ置き換え。

表 7.2 対応している、欧文タイポグラフィーと絵文字のための OpenType 機能（日中韓テキスト用の OpenType 機能は表 7.9 に挙げます）

タグ	名前	説明
dnom	分母	スラッシュの直後の数字を分母数字へ置き換え
frac	分数	スラッシュで区切られた数字群を、通用の、斜め線による分数へ置き換え
lnum	横並び数字	数字をオールドスタイルからデフォルトの横並び字体へ変えます。
mgrk	数学ギリシャ文字	ギリシャ文字グリフの標準タイポグラフィー字体を、数学的表記で広く用いられている字体へ置き換え。
numr	分子	スラッシュの直前の数字を分子数字へ置き換え、タイポグラフィースラッシュを分数スラッシュへ置き換え。
onum	オールドスタイル数字	数字を、デフォルトの横並び形式からオールドスタイル字体に変えます。
pnum	プロポーションアル数字	等幅（表形式）数字を、プロポーションアル幅を持つ数字へ置き換え。
sinf	科学的下付き	横並びまたはオールドスタイル数字を、主に化学や数学表記用の下付き数字（より小さなグリフ）へ置き換え。
tnum	等幅数字	プロポーションアル数字を等幅（表形式）数字へ置き換え。
zero	スラッシュ付きゼロ	数字ゼロに対するグリフを、中空部に斜め線を引いた字体へ置き換え。
大文字 / 小文字・序数		
c2pc	大文字からのプティットキャピタル	大文字キャラクターをプティットキャピタルに変えます。
c2sc	大文字からのスモールキャップス	大文字キャラクターをスモールキャップスに変えます。
case	ケースセンシティブ字体	さまざまなアクセント記号を上へずらして、全大文字シーケンスや横並び数字との調和を高めます。また、オールドスタイル数字を横並び数字に変えます。
ordn	序数	デフォルトのアルファベットグリフを、数字の後に用いるための序数字体へ置き換え。通常、Numero キャラクター U+2116 も作成します。
pcap	プティットキャピタル	小文字キャラクターをプティットキャピタルに、すなわち通常のスモールキャップより背の低い大文字に変えます。
smcp	スモールキャップス	小文字キャラクターをスモールキャップスに変えます。
unic	ユニケース	大文字と小文字を、小文字とスモールキャップス字体の混在する集合へマップして、高さ一定のアルファベットにします。
位置による字体		
falt	改行箇所語尾グリフ	行末グリフを代替字体（必要に応じて前幅が短かったり長かったりするもの）へ置き換えることによって、テキストの均等割り付けを助けます。
isol ² init medi fina	独立形・語頭形・語中形・語末形	続け書きされる用字系、とりわけアラビア語に対して、単語の外（独立して）・先頭・途中・末尾に出現するグリフを、各字体へ置き換え。
その他		
_none	全機能無効化	すべての OpenType 機能を無効化。

表 7.2 対応している、欧文タイポグラフィと絵文字のための OpenType 機能（日中韓テキスト用の OpenType 機能は表 7.9 に挙げます）

タグ	名前	説明
ccmp ¹	グリフ合成 / 分解	グリフ字体の数を最小化するために、1 個のキャラクターを 2 個のグリフへ分解したり、あるいはより良いグリフ処理のために 2 個のキャラクターを 1 個のグリフに合成したりすることが望ましい場合があります。この機能は絵文字シーケンスのためによく使われます。
ornm	飾り文字	ビュレットキャラクターか ASCII キャラクター群、または両方を飾り文字へ置き換え。

1. この機能はデフォルトで有効です。
2. これらの機能は、`shaping=true` の場合には自動的に有効化されます。

OpenType 機能をクエリー `PDF_info_font()` を用いて、フォント内の OpenType 機能をプログラムでクエリーすることができます。以下の命令は、そのフォントで利用可能かつ PDFlib が対応しているすべての OpenType 機能をスペース区切り列挙したものを取得します：

```
result = (int) p.info_font(font, "featurelist", "");
if (result != -1)
{
    /* 機能をスペース区切り列挙した文字列を取得 */
    featurelist = p.get_string(result, "");
}
else
{
    /* 対応機能全然見つからず */
}
```

たとえば合字 (*liga*) など特定の機能に PDFlib と対象フォントが対応しているかを調べるには以下の断片を使用します：

```
result = (int) p.info_font(font, "feature", "name=liga");
if (result == 1)
{
    /* 機能はフォントとPDFlibが対応している */
}
```

7.4.2 テキスト行・テキストフローで OpenType レイアウト機能

PDFlib は、テキスト行・テキストフロー関数群では OpenType レイアウト機能に対応していますが、`PDF_show()` など単純テキスト出力関数群では対応していません。

OpenType レイアウト機能のための要件 レイアウト機能群を用いて使用されるための OpenType フォントは、使われる OpenType 機能（群）を持つ GSUB テーブルを持っている必要があり、かつ、`encoding=unicode` か `glyphid` を用いて読み込む必要があります。

OpenType 機能を扱う際には以下に留意してください：

- ▶ OpenType 機能（オプション `features`・`script`・`language`）は、同一フォント内のグリフ群に対してのみ適用されますので、フォールバックフォントを指定している場合は、マスターフォントと 1 個ないし複数のフォールバックフォントとにわたるグリフ群には適用されません。

- ▶ 必ず、必要な場面でそのつど各機能を有効・無効を切り替えてください。OpenType 機能をうっかり全テキストに対して有効なままにしておくと、予期しない結果が生じることがあります。
- ▶ 以下の機能はデフォルトで有効化されます：*calt*・*ccmp*・*clig*・*liga*・*locl*。

OpenType 機能を制御 テキストの部分ごとに、必要に応じて OpenType 機能の有効と無効を切り替えることができます。*features* テキストオプションを使って機能を名前でも有効化し、機能名の頭に *no* をつけてそれを無効化します。たとえば、スモールキャップ機能をテキストフローに対するインラインオプションリストで有効化するには以下を使用します：

```
<features={smcp}>PDF<features={nosmcp}
```

テキスト行に対しては以下のように OpenType 機能を有効にできます：

```
p.fit_textline("PDF", x, y, "features={smcp}");
```

すべての機能を無効化するには、キーワード *_none* を用いるか、フォントオプション *readfeatures=false* を用いて関連の OpenType テーブルを無視します。OpenType 機能は、PDFlib Personalization Server (PPS) で利用するためのブロックプロパティとして有効にすることも可能です。

制御キャラクターで合字を無効に 言語によっては、ある特定の状況では合字を使ってはいけないうものがあります。ドイツ語などの言語におけるタイポグラフィ規則では、複合語の境界をまたぐ合字の使用を禁じています。たとえば *f+i* の組み合わせは、単語 *Schilfinsel* 内では合字へ置き換えてはいけません。合成された 2 つの単語の境界をまたいでいるからです。

先述のように、合字などの OpenType 機能の処理は、*features* オプションで有効・無効を切り替えることができます。上記のような場合のたびに合字をオプションで無効にしていくのは面倒です。シンプルな合字制御を提供するために、テキスト内の制御キャラクターで合字を無効にする機能がありますので、これを利用すれば、機能の有効・無効をたくさんオプションで切り替える必要はなくなります。連続するキャラクターの間に **ゼロ幅非接合子** (U+200C、*‍*、表 7.6 も参照) キャラクターを挿入すれば、合字が有効にしてあっても、その箇所は合字へ置き換わらなくなります。たとえば、以下のコードは *f+i* 合字を作成しません：

```
<features={liga charref=true}>Schilf&zwj;insel
```

7.4.3 用字系・言語固有の OpenType レイアウト機能

OpenType 機能は、あらゆる場面で適用されるものもありますし、ある特定の用字系に対して実装されているものもあります。ある特定の用字系と言語との組み合わせに対して実装されているものもあります。このため、*features* オプションとともに *script*・*language* テキストオプションを与えることが可能です。これらが効力を持つのは、その機能がフォント内で用字系または言語に固有な形で実装されている場合のみです。

たとえば、*f・i* グリフに対する合字は、いくつかのフォントにおいては、トルコ語が選ばれているときには有効化されません（なぜならトルコ語では点のない *i* が使用されるので、*i* の合字形はそれと混同しやすいからです）。そのようなフォントを用いている場合に、以下のテキストフローオプションリストは、用字系 / 言語を一切指定していないので、望まない合字を生成します：

<features={liga}>fi

しかし、以下のテキストフローオプションリストは、トルコ語オプションがあるので合字を作成しません：

<script=latn language=TRK features={liga}>fi

locl 機能は、言語固有のキャラクター字体を選択します。例：

セルビア語用キャラクター字体：

<features={locl} script=cyrl language=SRB charref>б

ウルドゥー語用数字字体：

<features={locl} script=arab language=URD charref>٢٣٤٥

script・*language* オプションは、OpenType 仕様で定義されているキーワードに対応しています。世界のさまざまな地域に対する例を表 7.3 と表 7.4 に挙げます。用字系キーワードは必ず 4 字です。それよりキーワードが短い場合にはスペースキャラクターが末尾についていますので、*script* オプションの値にもそれを含める必要があります。

表 7.3 features オプションとともに使用できる、対応している用字系。対応しているすべての用字系を挙げているわけではありません。

用字系の名前	キーワード	用字系の名前	キーワード
特殊		南・中央アジア I：インド公用用字系	
用字系無指定	_none	デーヴァナーガリー	deva
自動用字系検出	_auto	グジャラート	gujr
欧州		テルグ	telu
欧文	latn	ベンガル	beng
キリル	cyrl	オディア（以前はオリヤー）	orya
ギリシャ	grek	カンナダ	knda
中東		グルムキー	guru
ヘブライ	hebr	タミル	taml
アラビア	arab	マラーヤラム	mlym
東アジア		南・中央アジア II：その他現代用字系	
日中韓表意文字	hani	チベット	tibt
ハングル	hang	モンゴル	mong
ハングル字母	jamo	東南アジア	
注音符号	bopo	タイ	thai
彝	yi	ラーオ	lao
西夏	tang	タイ ヴィエト	tavt
ひらがな	hira	表記法	
カタカナ	kana	点字	brai

表 7.4 language オプションに対するキーワード。対応しているすべての言語を挙げているわけではありません。

キー ワード	言語	キー ワード	言語	キー ワード	言語
_none	言語指定なし	ELL	ギリシャ語	PLK	ポーランド語
ARA	アラビア語	GUJ	グジャラート語	ROM	ルーマニア語
BEN	ベンガル語	IWR	ヘブライ語	RUS	ロシア語
BGR	ブルガリア語	HIN	ヒンディー	SRB	セルビア語
CAT	カタルーニャ語	HUN	ハンガリー語	SND	シンド語
ZHP	中国語注音符号	IND	インドネシア語	SNH	シンハラ語
ZHS	中国語簡体字	JAN	日本語	ESP	スペイン語
ZHT	中国語繁体字	KAN	カンナダ語	TAM	タミル語
NLD	オランダ語	KSH	カシミール語	TEL	テルグ語
ENG	英語	KHM	クメール語	THA	タイ語
FAR	ペルシア語	KOR	韓国語	TIB	チベット語
FRA	フランス語	ORI	オリヤー語	TRK	トルコ語 ¹
DEU	ドイツ語	PAS	パシュトー語	URD	ウルドゥー語

1. いくつかのフォントはトルコ語に対して誤って TUR を用いていますので、PDFlib はこのタグを TRK と等価として扱います。

7.5 複雑用字系の出力

クックブック 完全なコードサンプルがクックブックの`complex_scripts/starter_shaping`トピックにあります。

7.5.1 複雑用字系

欧文用字系では左から右へ、1つのキャラクターの後に次のキャラクターを配していきます。他の用字系（書記系）では他にもテキスト出力のための必要事項があります。このような書記系を複雑用字系と呼ぶことにします。PDFlib は、さまざまな現代と昔の用字系のためのテキスト処理を実行します。この項ではこの処理をさらに詳しく解説します。用字系によっては他にも処理が必要なものがあります：

- ▶ アラビア語とヘブライ語では、テキストを右から左へ置いていきます。混合テキスト（アラビア文の中に欧文がはさまった）は、右書きの部分と左書きの部分の両方を含みます。これらの部分は並べ替えの必要があり、これを双方向（Bidi）問題といいます。
- ▶ いくつかの用字系、とりわけアラビア語では、キャラクターの位置（単独、語頭 / 語中 / 語尾）によって異なるキャラクター形状を用います。
- ▶ キャラクター列を必須合字へ置き換えます。
- ▶ グリフの位置を縦・横へ調整する必要があります。
- ▶ インド系用字系では、いくつかのキャラクターの並べ替えが必要です。すなわち、キャラクターはテキスト内での位置が変わることがあります。

シェーピング これらの処理ステップを1つないし複数必要とする用字系を複雑用字系と呼びます。入力された論理テキストから正しい表記を作り上げる処理をシェーピングといいます。シェーピングは、キャラクターが単語の先頭・途中・末尾、独立した位置のいずれにあるかに応じて、適切なグリフを選択します。シェーピングはアラビア語とヒンディーのテキスト組版の不可欠な要素です。シェーピングは、2つ以上のキャラクターの列を然るべき合字へ置き換えたり、キャラクターを並べかえたりすることもあります。シェーピング処理がキャラクターの字体を自動的に決定しますので、明示的な合字と Unicode 表示形（U+FB50 以降の Arabic Presentation Forms A など）を入力キャラクターとして使用してはいけません。

注意 複雑用字系のシェーピングを扱う際には以下に留意してください：

- ▶ PDFlib は *shaping·script* オプションを自動的に設定せず、ユーザーがそれらを与える前提しています。
- ▶ 用字系固有のシェーピング（オプション *shaping·script·language*）は、同一フォントからのグリフ群に対してのみ適用され、複数のフォントにわたるグリフ群には適用されません。フォールバックフォントを使っている場合は、シェーピングは同一（ベースまたは予備）フォントのテキスト区間内でのみ適用されます。
- ▶ シェーピングはテキスト内のキャラクターの順番を変えることがありますので、単語内の属性変更については注意を払う必要があります。たとえば、テキストフロー内でインラインオプションを用いて単語内の 2 番目のキャラクターに色をつけようとしている場合に、シェーピングが 1 番目と 2 番目のキャラクターを入れ替えたらどうなるのでしょうか。この理由から、シェーピングされたテキストの中での書式の変更は、単語の途中では行わずに、単語の境界でのみ行う必要があります。

シェーピングのための要件 複雑用字系のシェーピングとともに用いるためのフォントは、以下の要件を満たしている必要があります：

- ▶ 選択した用字系に対して必要な OpenType 機能テーブルをフォントが持っている必要があります。
- ▶ フォントを `encoding=unicode` か `glyphid` で読み込む必要があります。
- ▶ `PDF_load_font()` の `vertical` オプションは用いてはいけません。また、`readshaping` オプションを `false` に設定してはいけません。

7.5.2 シェーピング処理を制御

用字系の選択 複雑用字系の処理 (*shaping* オプション) には *script* オプションが必須です。さらに *language* オプションを与えることもできます。これはシェーピングの言語独自の側面を制御します。たとえばアラビア語とウルドゥー語で数字を変えるなどです。しかし、言語独自のシェーピングテーブルを持っているフォントはかなり少ないので、多くの場合には *script* オプションを与えれば充分であり、*language* オプションを用いてシェーピングを向上させることはできません。このシェーピング処理は、表 7.5 に挙げる用字系に対応しています。以下のオプションリストは、アラビア語テキストに対してシェーピング（と双方向処理）を有効化します：

```
shaping script=arab
```

表 7.5 shaping オプションが対応している用字系

地域	対応している用字系とそのキーワード
中東	アラビア (arab)・ヘブライ (hebr)
中央アジア	シンハラ (sinh)・チベット (tibet)
東南アジア	クメール (khmr)・タイ (thai)
東アジア	ハングル (hang)・日中韓表意文字 (hani)
インド	ベンガル (beng)・デーヴァナーガリー (deva)・グジャラート (gujr)・カナダ (knda)・マラーヤラム (mlym)・オディア [以前はオリヤー] (orya)・グルムキー (guru)・タミル (tam1)・テルグ (telu)

`script=_auto` にすると、PDFlib は、テキスト内のキャラクターの多数が属する用字系を自動的に割り当てます。欧文テキストはシェーピングを必要としないので、用字系を自動的に決定する際には無視されます。`PDF_info_textline()` の `scriptlist` キーワードを使うと、テキストに対して用いられている用字系をクエリーすることができます。

言語の選択 *language* オプションは、テキストが書かれている自然言語を指定します。表 7.3 に挙げるキーワードが使えます。例：

```
language=ARA
language=URD
```

注 テキストフローにおける高度な改行（254 ページ「9.2.10 高度な用字系固有の改行」を参照）でも、言語固有の処理が行われますが、これは *language* オプションによって制御されるのではなく、*locale* オプションによって制御されます。*locale* オプションは言語だけでなく、国と地域をも特定するものです。

シェーピング動作をオーバーライド 場合によって、シェーピング動作をオーバーライドしたいこともあります。PDFlib はこの目的のためにいくつかの Unicode 組版キャラク

ターをサポートしています。簡便のために、これらの組版キャラクターは、実体名を用いて指定することも可能です（参照）。

表 7.6 デフォルトのシェーピング動作をオーバーライドする Unicode 制御キャラクター

組版キャラクター	実体名	Unicode 名	機能
U+200C	ZWNJ	ZERO WIDTH NON-JOINER	隣り合う 2 つのキャラクターどうしがつながることを防ぐ
U+200D	ZWJ	ZERO WIDTH JOINER	隣り合う 2 つのキャラクターどうしを強制的につなげる

7.5.3 双方向組版

クックブック 完全なコードサンプルがクックブックの `complex_scripts/bidi_formatting` トピックにあります。

右書きのテキスト（アラビア語・ヘブライ語をはじめとするさまざまな用字系）においては、アドレスや別言語による引用などで、左書きの欧文テキスト列がネストされることが非常に頻繁にあります。このような混在テキスト列では双方向（Bidi）組版が必要になります。数字はつねに左書きされますので、双方向問題は、全くアラビア語・ヘブライ語だけで書かれたテキストにも生じます。双方向処理は、オプションで有効にする必要はなく、右書きのテキストが適切な *script* オプションとともに現れた際には、シェーピング処理の一環として自動的に適用されます。

注 双方向処理は、複数行テキストフローでは対応しておらず、テキスト行（すなわち一行テキスト出力）でのみ対応しています。

双方向アルゴリズムをオーバーライド 自動双方向処理は多くの場合において適切な結果を与えますが、場合によっては明示的なユーザー制御が必要なこともあります。PDFlib ではこの目的のためにいくつかの方向組版コードに対応しています。利用の便宜のため、これらの組版キャラクターは実体で指定することも可能です（表 7.7 参照）。これらの双方向組版コードは、以下のような場合にデフォルトの双方向アルゴリズムをオーバーライドするのに有用です：

- ▶ 右書きの段落が左書きのキャラクター群で始まる場合。
- ▶ 混在テキスト列がネストされている場合。
- ▶ 左書きテキストと右書きテキストとの間の境界に、句読点など弱いキャラクター群がある場合。
- ▶ 混在テキストを含む製品番号などの場合。

右書き文書処理の向上のためのオプション さまざまな組版オプションや Acrobat の動作のデフォルト設定は、左書きテキスト出力に合わせて設定されています。右書きのテキスト組版と文書表示のためには、以下のオプションを用います：

- ▶ テキスト行を、以下のはめ込みオプションで右揃えで配置します：

```
position={right center}
```

- ▶ リーダーを、テキストと左枠の間に作成します：

```
leader={alignment=left text=.
```

表 7.7 双方向アルゴリズムをオーバーライドするための方向組版コード

組版コード	実体名	Unicode 名	機能
U+202A	LRE	左書き埋め込み (LRE)	埋め込み左書き列を開始します
U+202B	RLE	右書き埋め込み (RLE)	埋め込み右書き列を開始します
U+200E	LRM	左書きマーク (LRM)	左書きのゼロ幅キャラクター
U+200F	RLM	右書きマーク (RLM)	右書きのゼロ幅キャラクター
U+202D	LRO	左書き上書き (LRO)	キャラクター群を強い左書きキャラクター群として扱うよう強制
U+202E	RLO	右書き上書き (RLO)	キャラクター群を強い右書きキャラクター群として扱うよう強制
U+202C	PDF	方向組版ポップ (PDF)	直前の LRE・RLE・RLO・LRO の前の双方向ステータスへ復帰

- ▶ `PDF_begin/end_document()` の以下のオプションを用いて、Acrobat での右書き文書・ページ表示を改善します：

```
viewerpreferences={direction=r2l}
```

コード内で双方向テキストを扱う 双方向テキストを扱う際には以下も有用でしょう：

- ▶ `PDF_info_textline()` の `startx/starty・endx/endy` キーワードを用いると、それぞれ論理的開始・終了キャラクターの座標を知ることができます。
- ▶ `PDF_info_textline()` の `writingdirx` キーワードを用いると、テキストの主流な書記方向を知ることができます。この方向は、テキストの先頭キャラクター群から、または表 7.7 に従った方向組版コード（テキスト内にあれば）から推定されます。
- ▶ `PDF_info_textline()` の `position` オプションで `auto` キーワードを用いると、自動的にアラビア語またはヘブライ語のテキストは右枠へ、欧文テキストは左枠へ寄せられます。たとえば以下のテキスト行オプションは、テキストをベースライン上に右寄せまたは左寄せします：

```
boxsize={width 0} position={auto bottom}
```

7.5.4 アラビア語テキスト組版

クックブック 完全なコードサンプルがクックブックの `complex_scripts/arabic_formatting` トピックにあります。

上述の双方向組版とテキストのシェーピングに加え、アラビア用字系のテキスト出力の生成に関しては、ほかにも組版上の側面がいくつかあります。

アラビア合字 アラビア用字系は合字を多用します。多くのアラビア語フォントは 2 種類の合字を含んでおり、それぞれ PDFlib では異なる扱いを受けます：

- ▶ 必須合字 (`rlig` 機能) はつねに適用する必要があります。ラーム-アリフおよびその派生形などがこれにあたります。必須合字は、`script=arab` で `shaping` オプションを有効にしている場合に自動的に使用されます。
- ▶ 任意アラビア合字 (`liga・dlig` 機能) は自動的に使用されず、他のユーザー制御の OpenType 機能と同様に、すなわち `features={dlig}` で有効にすることができます。任意アラビア合字は、複雑用字系処理とシェーピングの後に適用されます。

合字をさせない ある種の略称など、場合によっては、隣り合うキャラクターを合字にさせたくないことがあります。この場合には、表 7.6 に挙げた組版キャラクターを用いて、合字を強制したり禁止したりすることができます。たとえば、以下の例のゼロ幅非結合子は、キャラクターが合字を形成することを禁止して、正しい略称表記を生成しています：

```
&#x0623;&#x064A;&ZWJ;&#x0628;&#x064A;&ZWJ;&#x0625;&#x0645;
```

アラビア語テキストにおけるタトウィール タトウィールキャラクター U+0640 (カシーダともいう) を 1 個ないし複数挿入することによって、アラビア単語を引き伸ばすことができます。PDFlib は自動的にタトウィールキャラクターを挿入することによるテキストの均等揃えは行いませんが、このキャラクターを自分で入力テキスト内に挿入して単語を引き伸ばすことはできます。

アラビア語フォントに欧文キャラクターを追加 Google の Noto fonts macOS などいくつかのアラビア語フォントは、欧文キャラクターに対するグリフを一切含んでいません。この場合は *fallbackfonts* オプションを使って、欧文キャラクターをアラビア語フォントに連結することができます。PDFlib は、欧文またはアラビア語のテキスト入力に従って自動的に両フォントを切り替えます。すなわち、アプリケーション側でフォントを切り替える必要はなく、欧文とアラビア語が混在するテキストを 1 個のフォント指定で与えることができます。

fallbackfonts オプションに対して以下のフォント読み込みオプションリストを用いると、NotoNaskhArabic-Regular フォントへ NotoSerif-Regular フォントの欧文キャラクター群を追加することができます：

```
fontname=NotoNaskhArabic-Regular  
fallbackfonts={ {fontname=NotoSerif-Regular} }
```

7.6 日本語・中国語・韓国語テキスト出力

7.6.1 TrueType・OpenType 日中韓フォントを用いる

PDFlib は、TrueType・TrueType/OpenType Collections (TTC/OTC)・OpenType 形式の日中韓フォントに対応しています。日中韓フォントは以下のように処理されます：

- ▶ **embedding** オプションが **true** ならば、フォントは CID フォントに変換されて、PDF 出力に埋め込まれます。
- ▶ Windows 上の日中韓ホストフォント名は、BOM 付き UTF-8 形式か UTF-16 形式で **PDF_load_font()** に与えることができます。macOS では非欧文ホストフォント名に対応していません。

以下の例では、中国語テキストを ArialUnicodeMS フォントで印字しています。フォントは、システムにインストールされているか、あるいは 141 ページ「6.3.4 フォントを検索」に従って構成されている必要があります：

```
font = p.load_font("Arial Unicode MS", "unicode", "");
if (font == -1) { ... }
p.fit_textline("\u4e00\u500b\u4eba", x, y, "fontsize=24");
```

TrueType Collection 内の個別フォントにアクセス TTC/OTC ファイルは、複数の別々のフォントを持っています。各フォントを利用するにはその適切な名前を与えます。ただし、TTC/OTC ファイル内がどのフォントを持っているかを知らない場合には、各フォントを番号で指定することも可能です。具体的には、コロンとフォント番号 (0 から始まる) を追加することにより指定します。番号が 0 の場合には省略可能です。たとえば、TTC/OTC ファイル **msgothic.ttc** は複数のフォントを持っており、**PDF_load_font()** で以下のように指定することができます (各行内のフォント名は等価)：



msgothic:0	MS Gothic	msgothic:
msgothic:1	MS PGothic	
msgothic:2	MS UI Gothic	

ただし、**msgothic** (接尾辞をつけない) はフォント名としては扱われません。なぜならそれではフォントを一意に特定できないからです。フォント名のエイリアス (141 ページ「フォントデータのソース」参照) を TTC/OTC 番号と組み合わせて用いることも可能です。指定された番号のフォントが見つからないときには、メソッド呼び出しは失敗します。

TTC/OTC フォントファイルは 1 回のみ構成しなければなりません。TTC/OTC ファイル内のすべての番号づけられたフォントは自動的に発見されます。以下に、**msgothic.ttc** 内のすべての番号づけられたフォントを構成するために十分なコードを示します (141 ページ「6.3.4 フォントを検索」参照)：

```
p.set_parameter("FontOutline", "msgothic=msgothic.ttc");
```

7.6.2 横書きと縦書き

PDFlib は横書きと縦書きに対応しています。縦書きはさまざまな方法で要求できます：

- ▶ TrueType・OpenType フォントの場合は、**vertical** オプションを与えれば縦書きで利用できます。
- ▶ 「@」キャラクターで始まるフォント名はつねに縦書きで処理されます。

デフォルトでは縦書きではすべてのグリフが同じ高さを持ちます。ただし TrueType・OpenType フォントは、縦書きについてプロポーショナルなメトリックを内容とすることもできます。PDFlib では、フォントオプション `readverticalmetrics` (デフォルトは `false`) を設定することによって、そのようなプロポーショナルな縦書きメトリックを使用させることも可能です。

注 縦書きで文字の間をあけるには字間を負にする必要があります。

OpenType フォントは、表 7.8 に挙げる OpenType レイアウト機能を持っている場合があります。これらの機能は、デフォルトの字形を、縦書き用に調整された異体字へ置き換えるものです。うち `vkna` 機能については、`features` テキストオプションを用いて制御することが可能であり、また `vrt2/vert` は、縦書きでは自動的に有効になります。

表 7.8 縦書きテキストのための OpenType レイアウト機能

キー	名称	説明
<code>vert</code> ¹	縦組み	デフォルトの字形を、縦書き用に調整された異体字へ置き換え。
<code>vkna</code>	縦組みかな切替	標準のかなを、縦書きのためにデザインされた字形へ置き換え。
<code>vrt2</code> ¹	縦組み切替・回転	いくつかの等幅のグリフ（半角・三分の一幅・四分の一幅）とプロポーショナル幅のグリフ（主に欧文とカタカナ）を、縦書きに適した字形へ、すなわち 90°時計回りさせた形へ置き換え。この機能が存在しているときには、 <code>vrt2</code> の部分集合である <code>vert</code> 機能は無効化されます。

1. 縦書きにおけるフォントに対しては自動的に有効化されます

7.6.3 EUDC・SING フォントによる外字キャラクター

PDFlib は、日中韓テキストのためのカスタム外字キャラクターを利用できる Windows EUDC (エンドユーザー定義キャラクター、`*tte`)・SING フォント (`*gai`) に対応しています。最も便利なのは、カスタムキャラクターを持ったフォントをフォールバックフォント機構で他のフォントへ統合することでしょう。

外字キャラクターに対してフォールバックフォントを用いる 通常、外字キャラクターは Windows EUDC または SING グリフセットから持って来ますが、`fallbackfonts` オプションはあらゆる種類のフォントを受け付けます。ですのでこの方法は外字キャラクターに限らず、あらゆる種類の記号に対して利用することができます (例: 企業ロゴが別フォント内にある場合)。`fallbackfonts` オプションに対して以下のフォントオプションを用いれば、読み込んだフォントに対して、EUDC フォントからのユーザー定義 (外字) キャラクターを追加することができます:

```
fallbackfonts={
  {fontname=EUDC forcechars=U+E000 fontsize=140% textrise=-20%}
}
```

ベースフォントをひとたびこのフォールバックフォント構成で読み込めば、テキスト内の EUDC キャラクターは、フォントを変える必要なく使うことができます。

SING フォントの場合、Unicode 値は PDFlib によって自動的に決定されますので、与える必要はありません:


```

fallbackfonts={
  {fontname=PDFlibWing encoding=unicode forcechars=gaiji}
}

```

外字キャラクターに対する PUA 値を温存 場合によっては、たとえば Windows EUDC フォントでは、そのフォントによって外字キャラクターが私用領域 (PUA) 内の Unicode 値へマップされています。デフォルトでは、PDFlib は、生成する PDF (ToUnicode CMap) 内では PUA 値を U+FFFD (Unicode 置換キャラクター) へ置き換えます。結果として、そのようなキャラクターは、生成される PDF から正しく抽出できません。

この動作は、*preserverpua* フォントオプションを用いて変更することもできます。これが *true* に設定されている場合には、PUA 値を持つ外字キャラクターはその Unicode 値を温存します。

7.6.4 OpenType レイアウト機能と日中韓テキスト出力

163 ページ「7.4 OpenType レイアウト機能」で解説したように、PDFlib は OpenType・TrueType フォント内の高度なタイポグラフィレイアウトテーブルに対応しています。たとえば、OpenType 機能を用いて、欧文グリフのプロポーショナル幅か半角かいずれかの字体を選択したり、異体字を選択したりすることが可能です。表 7.9 に、日中韓テキストのための OpenType 機能を挙げます (他に、一般的用途のための OpenType レイアウト機能を表 7.2 に、縦書きのための機能を表 7.8 に挙げています)。

表 7.9 日本語・中国語・韓国語テキスト用の対応 OpenType レイアウト機能

キーワード	名前	説明
expt	エキスパート字形	JIS78 字と同様、この機能は、標準の和文字形を、対応する、タイポグラファーが望む字形へ置き換えます。
fwid	全角	他の幅に設定されたグリフを、全角 (たいていは em) に設定されたグリフへ置き換え。これは欧文キャラクターやさまざまな記号を含む可能性があります。
hkna	横組み用仮名	標準の仮名を、横書き専用デザインされた字形へ置き換え。
hngl	ハンゲル	(ISO 14496-22:2015/Amd.2:2017 により廃止) 韓文漢字キャラクターを、その対応するハンゲル (音素) キャラクターへ置き換え。
hojo	補助漢字字形 (JIS X 0212-1990)	JIS X 0213:2004 字形がデフォルトとしてエンコードされている場合に、JIS X 0212-1990 字形 (「補助漢字」ともいいます) を利用。
hwid	半角	プロポーショナル幅の、または em の半分以外の等幅のグリフを、em の半分の幅のグリフへ置き換え。
ital	イタリック	ローマン体のグリフを、その対応するイタリック体のグリフへ置き換え。
jp04	JIS2004 字形	(nlck 機能の部分集合) JIS X 0213:2004 グリフを利用。
jp78	JIS78 字形	デフォルト (JIS90) の和文グリフを、その対応する JIS C 6226-1978 (JIS78) の字形へ置き換え。
jp83	JIS83 字形	デフォルト (JIS90) の和文グリフを、その対応する JIS X 0208-1983 (JIS83) の字形へ置き換え。
jp90	JIS90 字形	JIS78 または JIS83 の和文グリフを、その対応する JIS X 0208-1990 (JIS90) の字形へ置き換え。
loc1 ¹	ローカライズ字形	グリフのローカライズされた字形で、デフォルト字形を置き換えることを可能にします。この機能は <i>script</i> ・ <i>language</i> オプションを必要とします。

表 7.9 日本語・中国語・韓国語テキスト用の対応 OpenType レイアウト機能

キー ワード	名前	説明
nalt	修飾字形	デフォルトのグリフを、さまざまな表記字形へ置き換え（白丸囲み・黒丸囲み・四角囲み・括弧付き・菱形囲み・角丸四角囲みなど）。
nlck	国語審議会漢字字形	日本の国語審議会（NLC）が多くの JIS キャラクターに対して 2000 年に制定した新たなグリフ形状を利用。
pkna	プロポーショナル仮名	等幅（半角または全角）に設定された仮名および仮名関連のグリフを、プロポーショナルなグリフへ置き換え。
pwid	プロポーショナルグリフ	等幅（通常、全角または em の半分）に設定されたグリフを、プロポーショナルな字送りのグリフへ置き換え。
qwid	四分の幅	他の幅のグリフを、em の 4 分の 1 の幅に設定されたグリフへ置き換え。
ruby	ルビ表記字形	デフォルトの仮名グリフを、（通常、上付きにされた）ルビ用にデザインされた、より小さいグリフへ置き換え。
smpl	簡体字	和文漢字または中文繁体字を、その対応する簡体字へ置き換え。
tnam	名前旧字体	和文新字体を、その対応する旧字体へ置き換え。これは trad 機能と等価ですが、ただし個人の名前における使用が適切であると認められる旧字体に限られます。
trad	旧字体	和文漢字または中文簡体字を、その対応する旧字体 / 繁体字へ置き換え。
twid	三分の幅	他の幅のグリフを、em の 3 分の 1 の幅に設定されたグリフへ置き換え。

1. この機能はデフォルトで有効化されます。

7.7 Unicode 異体字セレクター (UVS)

1 つの Unicode キャラクターを表現するグリフが何種類も存在することもあります。通常、そうした視覚的差異は、適切なフォント (レギュラーかイタリックかなど) か、OpenType 機能で選択されるフォント内の代替グリフ (スウォッシュキャラクターなど) を使って実現されます。場面によっては、グリフの選択がセマンティックに意味を持ち、フォント関連の組版情報一切なしでプレーンテキスト内でもそれを明確にする必要があります。Unicode はこの目的のために異体字セレクターを提供しています。

7.7.1 各種の異体字シーケンス

1 つの Unicode 異体字シーケンス (UVS) は、1 つのベース Unicode キャラクターと、その後の 1 つの異体字セレクターから成っています。このシーケンスはこのベースキャラクターの異体字を選択します。Unicode 規格は各種のシーケンスを定義しています。OpenType フォントは、異体字グリフとその異体字セレクターを指定する UVS テーブルを持つことができます。この UVS テーブルは、そのフォント内でデフォルトではどのグリフ異体字が使用され、どの代替グリフ群が利用可能かを指定します。

標準化異体字シーケンス これらは Unicode キャラクターデータベース内で定義されています。これらは *U+FE00 VS1* ~ *U+FE0D VS14* の異体字セレクターのうちの一つを使用します (目下 VS1 のみが標準化されています)。それをベースキャラクターの後に配する必要があります。標準化異体字シーケンスは、数学グリフの字体と、モンゴル語のテキストのために使用されます。

絵文字異体字シーケンス 絵文字異体字シーケンスというリストの中に、数百個の絵文字キャラクターが含まれています。これらはテキストスタイル (たいていは単色) と絵文字スタイル (たいていはカラー) で利用可能です。この選択は、Unicode キャラクターの後に 1 つか 2 つの異体字セレクターを配することによってオーバーライドすることも可能です:

- ▶ *U+FE0E VS15* テキスト表示セレクターはテキストスタイル (単色) のグリフを選択します。テキストスタイルのほうが望ましい場面としてはたとえば、グリフに内蔵されている色に依存せずにカスタムの色をグリフに適用したい場合などです。
- ▶ *U+FE0F VS16* 絵文字表示セレクターは絵文字スタイル (カラー) のグリフを選択します。

Unicode 規格では各絵文字キャラクターのデフォルトのスタイルを指定していますが、フォント内の UVS テーブルは個別のキャラクターについてデフォルトを変えることもできます。より重要なのは、これらの異体字を利用可能かどうかはフォントデザイナーに依存する点です。

表意文字異体字シーケンス (IVS) これらは、Unicode Technical Standard #37 に従った登録プロセスによって定義されており、表意文字異体字データベース内にリストされています。たとえば、日中韓互換表意文字のための異体字グリフが利用可能です。IVS は、ベースキャラクターとしての統一表意文字キャラクター 1 個と、*U+E0100 VS17* ~ *U+E01EF VS256* の 240 種の異体字セレクターのうちの一つから成ります。

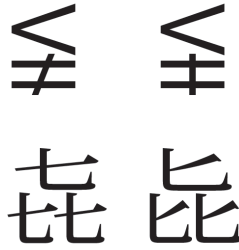


図 7.2 デフォルトグリフ（左）と異体字グリフ（右）

7.7.2 PDFlib で異体字シーケンス

異体字グリフを選択 PDFlib は、OpenType フォント内の UVS テーブルを、それがフォントオプション `readselectors=false` を用いて無効化されていない限り、解釈します。

フォントが、必要なグリフを含んでいることがわかっている場合には、異体字シーケンスに対する作業は、単に PDFlib のテキスト出力関数にそのシーケンスを与えれば済みます。以下のコード断片は、Unicode ベースキャラクターのデフォルトグリフと、セレクターによって選択された異体字を印字します。結果を図 7.2 に示します。それぞれのグリフのペアが違うことがわかります：

```
p.fit_textline("&#x2268; &#x2268;&#xFE00;", 50, 700,
              "fontname={Cambria Math} fontsize=24 charref=true");

p.fit_textline("&#x3402;&#xE0100; &#x3402;&#xE0101;", 50, 650,
              "fontname={KozMinPr6N-Regular} fontsize=24 charref=true");
```

必要なグリフがフォントにないなどの原因によって、ベースキャラクターに対する異体字セレクターに従えない場合には、それは無視されます。

フォント内の異体字セレクターをクエリー `PDF_info_font()` を使って、フォントがそもそも異体字セレクターを含んでいるのかどうかをチェックすることもできます。`selector` キーワードを `index` オプションとともに用いれば、そのフォント内で得られるすべての異体字セレクターの一覧を取得することができます：

```
for (i = 0; i < 256; i++)
{
    selectors[i] = (int) p.info_font(font, "selector", "index=" + i);
    if (selectors[i] == -1)
    {
        selectorcount = i;
        break;
    }
}
```

以下のコード断片を使うと、フォントが、Unicode 値 `uv` の後にセレクター `s` を配した特定のシーケンスに対する異体字グリフを含んでいるかどうかをチェックできます：

```
if (p.info_font(font, "unicode", "unicode=" + uv + "selector=" + s) != -1)
{
    /* このフォント内でこのシーケンスに対して異体字グリフが利用可能 */
}
```

このクエリーは、異体字が得られるかどうかをチェックすることだけを意図しています。得られる Unicode 自体は、PDFlib は異体字に PUA Unicode 値を割り当てますので、おそらく有用ではないでしょう。

7.8 テキスト処理パイプライン

クライアントアプリケーションは、ページ出力したいテキストを PDFlib に与えます。このテキストは何らかのエンコーディングと形式に従って符号化されています。しかし、PDFlib の内部処理は Unicode 規格に基づいており、また最終テキスト出力はフォント固有のグリフ ID を必要とします。ですので PDFlib は、ページ内容のために与えられる文字列をテキスト処理パイプラインの中で 3 つの過程を経て処理します：

- ▶ 入力コードを、選択されているエンコーディングに従って Unicode 値へ正規化。
- ▶ Unicode 値を、そのフォント内で利用可能なグリフに従ってフォント固有のグリフ ID へ変換。
- ▶ グリフ ID を、出力エンコーディングに従って転換。

テキスト処理パイプラインのいくつかの側面はオプションで制御できます。

7.8.1 入力文字列を Unicode へ正規化

以下のステップが、*glyphid* 以外のすべてのエンコーディングに対して実行されます：

- ▶ Unicode 対応言語バインディング：シングルバイトエンコーディングが指定されているときは、UTF-16 テキストは高次バイトを捨てることでシングルバイトテキストへ変換されます。
- ▶ エスケープシーケンス（121 ページ「5.4.1 エスケープシーケンス」参照）を、その対応する数値へ置き換えます。
- ▶ 文字参照を解決し、それをその対応する Unicode 値へ置き換えます（122 ページ「5.4.2 文字参照」および次項参照）。
- ▶ シングルバイトエンコーディング：シングルバイトテキストを、指定されたエンコーディングに従って Unicode へ変換します。
- ▶ *normalize* オプションに応じて、テキストを、Unicode 正規化形 (NFC など) のうちの一つへ正規化。

グリフ名による文字参照 グリフ名参照（122 ページ「5.4.2 文字参照」を参照）が使用されている場合には、PDFlib はその指定された名前を持つグリフの位置をフォント内で見つけようと試み、そしてその参照をそのグリフの Unicode 値へ置き換えます。指定されたグリフがフォント内で見つからないときは、PDFlib は Unicode 値を決定するためにその内蔵グリフ名テーブルを検索します。この Unicode 値はさらに、適切なグリフがフォント内に存在しているかどうかをチェックするために使われます。そのようなグリフが見つからないときは、動作は *glyphcheck · errorpolicy* 設定で制御されます。文字参照は、*glyphid · builtin* エンコーディングでは使うことができません。

Unicode 制御キャラクター 制御キャラクターは、何のグリフをも表現せず、組版情報を伝達するために使用される Unicode 値です。PDFlib は、以下のグループの Unicode 制御キャラクターを処理します：

- ▶ デフォルトのシェーピング動作をオーバーライドするための制御キャラクター群（表 7.6 に挙げています）と、デフォルトの双方向組版をオーバーライドするための制御キャラクター群（表 7.7 に挙げています）は、テキスト行とテキストフローの中において、複雑用字系のシェーピングと OpenType レイアウト機能の処理を制御します。これらの制御キャラクターは評価された後に除去されます。
- ▶ 表 9.1 に挙げる、改行とテキストフロー組版のための組版制御キャラクター群。これらの制御キャラクターは評価された後に除去されます。

- ▶ その他、範囲 U+0001 ~ U+0019 と U+007F ~ U+009F の Unicode 制御キャラクターは *replacementchar* キャラクターで置き換えられます。

フォントが制御キャラクターのためのグリフを持っている場合であっても、そのグリフは通常は見えなくなります。なぜなら PDFlib が制御キャラクターを除去するからです（この規則の例外として、*&NBSP;* と *&SHY;* は除去されません）。しかし *encoding=glyphid* を用いると、制御キャラクターはテキスト内に残されるので、出力で見せることができます。

7.8.2 Unicode 値をグリフ ID へ変換

前項で決定された Unicode 値は、いくつかの理由により変更が必要な場合があります。以下のステップはすべてのエンコーディングに対して実行されますが、ただし以下の例外があります：*encoding=glyphid* の場合、無効なグリフ ID はグリフ ID 0 へ置換され、*glyphcheck=error* のときは例外が発生します。

フォールバックフォントからのキャラクターを強制 Unicode 値を、*fallbackfonts* オプションの *forcechars* サブオプションに従って置き換えて、その対応するフォールバックフォントのグリフ ID を決定します。詳しくは 146 ページ「6.3.6 フォールバックフォント」を参照してください。

異体字シーケンスを解決 フォントによっては、Unicode キャラクターの後に、そのキャラクターの特定のグリフ異体字を選択する異体字セレクターが続くものがあります（179 ページ「7.7 Unicode 異体字セレクター (UVS)」参照）。そのフォントがその異体字シーケンスのための異体字グリフを内容として持っている場合には、元のグリフ ID ではなくその異体字グリフのグリフ ID が用いられます。

グリフ ID へ変換 Unicode 値を、133 ページ「6.2.2 グリフに対する Unicode マッピング」で決定されるマッピングに従ってグリフ ID へ変換します。Unicode 値に対応するグリフ ID がフォント内に見つからないときは、その次のステップは *glyphcheck* オプションによって異なります：

- ▶ *glyphcheck=none* : グリフ ID 0 が用いられます。すなわち、*.notdef* グリフがテキスト出力内で用いられます。*.notdef* グリフが視覚的形狀を内容として持つ場合（たいていは白四角か四角パッテン）には、問題の起きたキャラクターが PDF ページ上で目に見えるようになります。それが望ましいかどうかは場合によるでしょう。
- ▶ *glyphcheck=replace*（これがデフォルト）：警告メッセージがログ記録され、PDFlib は、マップできない Unicode 値を後述のグリフ置換機構によって置き換えようと試みます。
- ▶ *glyphcheck=error* : PDFlib はエラーを発生させます。*errorpolicy=return* の場合には、これはすなわちメソッド呼び出しがテキスト出力を一切作成せずに終了することを意味し、*PDF_add/create_textflow()* が -1 (PHP の場合 : 0) を返すことを意味します。*errorpolicy=exception* の場合は例外が発生します。

グリフ置換 *glyphcheck=replace* の場合は、マップできない Unicode 値は再帰的に以下のように置き換えられます：

- ▶ マスターフォントを読み込んだ際に指定したフォールバックフォントの中で、その Unicode 値に対するグリフが検索されます。各フォントに対してフォールバックフォントは複数指定することもできるので、ここでは任意の数のフォントが関わる可能性があります。フォールバックフォントのうちの 1 つでグリフが見つかったときはそれが使われます。

- ▶ タイポグラフィ的に類似のグリフを、PDFlib の内蔵置換テーブル内の Unicode 値に従って選びます。この内蔵リストからこれらの置換のいくつかを以下に抜粋します。リスト内の 1 番目のキャラクターがフォント内で見つからないとき、それは 2 番目のキャラクターへ置き換えられます：

U+00A0 (ノーブレイクスペース)	U+0020 (空白)
U+00AD (ソフトハイフン)	U+002D (ハイフン-マイナス)
U+2010 (ハイフン)	U+002D (ハイフン-マイナス)
U+03BC (ギリシャ文字μ)	U+00C5 (マイクロ記号)
U+212B (オングストローム記号)	U+00B5 (欧文Aの上に丸があるÅ)
U+220F (多項総乗演算子)	U+03A0 (ギリシャ文字π)
U+2126 (オーム記号)	U+03A9 (ギリシャ文字Ω)

内蔵テーブルに加えて、全角キャラクター U+FF01 ~ U+FF5E は、フォント内で全角字体が得られない場合には、その対応する ISO 8859-1 キャラクター (すなわち U+0021 ~ U+007E) へ置き換えられます。

- ▶ Unicode の合字を、その構成グリフ群へ分解します (例: U+FB00 欧文合字 *ff* をシーケンス U+0066 *f*・U+0066 *f* へ置き換え)。
- ▶ 同じ Unicode セマンティクスを持つグリフを、そのグリフ名に従って選びます。特に、ピリオドで区切られたグリフ名接尾辞はすべて、その対応するグリフが得られないときは除去されます (例: *A.swash* を *A* へ置き換え。 *g.alt* を *g* へ置き換え)。

これらの方式がいずれも Unicode 値に対するグリフを与えないときは、*replacementchar* オプションが以下のように評価されます：

- ▶ *replacementchar=auto* (これがデフォルトです) の場合には、キャラクター U+00A0 (ノーブレイクスペース) と U+0020 (空白) が試されます。これらさえも得られないときは、「ミッシンググリフ」記号が使われます (PDF/A・PDF/UA・PDF/X-4/5 では不可)。
- ▶ Unicode キャラクターが *replacementchar* として指定されている場合には、それが元のキャラクターのかわりに用いられます。
- ▶ *replacementchar=drop* の場合には、そのキャラクターは入力ストリームから除去され、出力は作成されません。
- ▶ *replacementchar=error* の場合には、例外が発生します。これを利用すると、読めないテキスト出力を避けることができます。

クックブック 完全なコードサンプルがクックブックの `fonts/glyph_replacement` トピックにあります。

7.8.3 グリフ ID を転換

決定されたグリフ ID はまだ最終的なものではありません。なぜなら最終出力を作成する前に、いくつかの転換を行わなければならない可能性があるからです。具体的にどのような転換が必要かは、フォントやいくつかのオプションによって異なります。

縦書きグリフ 縦書きモードのフォントでは、いくつかのグリフは縦書き字体へ置き換わる可能性があります。この置換は、フォント内に *vert* または *vert2* OpenType レイアウト機能テーブルが必要です。

OpenType レイアウト機能 OpenType 機能は、合字・スワッシュキャラクター・スモールキャップスをはじめとするさまざまなタイポグラフィバリエーションを、1 個ないし複数のグリフ ID を他の値へ置き換えることによって作成することができます。この件は

163 ページ「7.4 OpenType レイアウト機能」で解説しています。OpenType レイアウト機能は、適切なフォントに対してのみ有効であり（166 ページ「OpenType レイアウト機能のための要件」参照）、かつ *features* オプションに従って適用されます。デフォルトで有効化されるレイアウト機能もあります。

複雑用字系のシェーピング シェーピングは、テキストの順序を替え、また、キャラクターの位置によって適切な字体のグリフを決定します（例：アラビア語キャラクターの語頭・語中・語末・独立形）。これは適切なフォントに対してのみ有効であり（170 ページ「シェーピングのための要件」参照）、かつ *shaping* オプションに従って適用されます。

8 画像・SVG グラフィック・PDF ページを取り込む

8.1 ラスター画像

8.1.1 基本的な画像処理

PDFlib でラスター画像を貼り付けるのは簡単です。まず、その画像を PDFlib のメソッドで開く必要があります。この関数は、画像特性を解析してピクセルデータを PDF 出力へコピーします。`PDF_load_image()` は、画像記述子の役割をするハンドルを返します。このハンドルを使って `PDF_fit_image()` を呼び出すことができるので、その際に位置付け・拡張オプションも与えます：

```
image = p.load_image("auto", "image.jpg", "");
if (image == -1)
    throw new Exception("エラー : " + p.get_errmsg());

p.fit_image(image, 0.0, 0.0, "");
p.close_image(image);
```

`PDF_fit_image()` メソッドの最後の引数は、画像の位置付け・拡張・回転を指定できるさまざまなオプションを持たせることのできるオプションリストです。このオプション群については 221 ページ「8.4 画像・グラフィック・取り込み PDF ページを配置」で詳しく説明します。

画像ファイルの取り込みが失敗した場合には、`PDF_load_image()` はエラーコードを返します。この画像の失敗についてもっと詳しく知る必要がある場合には、`PDF_get_errmsg()` を呼び出すと、詳細なエラーメッセージが得られます。

クックブック 画像処理のためのコードサンプルが PDFlib クックブックの images カテゴリにあります。

画像データを再利用 ラスター画像の反復利用のための重要な PDF 最適化技法に PDFlib は対応しています。複数のページに同じロゴや背景を使うレイアウトを考えてみましょう。そのような場合には画像データ本体は一度しか PDF 内に取り込まず、その画像を使う各ページからそこへの参照だけを生成することが可能です。画像ファイルを一度読み込んでおいて、各ページにロゴや背景を配置するたびに `PDF_fit_image()` を呼び出せばよいのです。複数のページにその画像を配置したり、同じ画像でも貼り付けるたびに拡張倍率を変えたりすることもできます（画像を閉じていない限り）。画像の容量や使用回数によってはこの技法は顕著な容量節減をもたらすでしょう。

拡張と dpi 計算 取り込んだ画像のピクセル数を PDFlib が変えることはありません。拡張すると画像のピクセルは膨らんだり縮んだりしますが、ダウンサンプリングが行われることはありません。拡張倍率が 1 ならば 1 ピクセルの大きさはユーザー座標の 1 単位と同じになります。いいかえれば、ユーザー座標系が拡張されていなければ画像はその元の解像度で（その画像が解像度情報を持っていないければ 72 dpi で）取り込まれます（デフォルトでは 1 インチが 72 単位なので）。

クックブック 完全なコードサンプルがクックブックの `images/image_dimensions` トピックにあります。そこに、画像の寸法を得る方法や、それをさまざまな大きさに貼る方法を示してあります。

取り込み画像の色空間 `PDF_load_image()` で与えられたオプションに従って ICC プロファイルの追加・削除をしたりスポットカラーか DeviceN カラーを適用したりする場合を除いて、PDFlib は一般に、取り込んだ画像の元の色空間を保持します。しかし、まれにこれが不可能な組み合わせがあります。

PDFlib は、RGB と CMYK との間の変換は一切行いません。そのような変換が必要なときは、画像を PDFlib に取り込む前にその画像データに適用しておく必要があります。

埋め込み ICC プロファイルを持っていない BMP・JPG・JPX・PNG・TIFF 形式のすべての RGB 画像に対してはデフォルトで sRGB ICC プロファイルが割り当てられます。

複数ページ画像 PDFlib は、複数の画像を持つ TIFF・JBIG2 画像に対応しています。これを複数ページ画像ファイルともいいます。複数ページ画像を利用するには、`PDF_load_image()` で `page` オプションを用います：

```
image = p.load_image("tiff", filename, "page=2");
```

この `page` オプションは、複数画像ファイルが利用されることを示し、利用したい画像の番号を指定します。先頭画像の番号は 1 です。このオプションは、`PDF_load_image()` がファイル内でもう画像が得られないことを示す `-1` を返すまで値を増やしていくことができます。

クックブック 複数画像 TIFF ファイル内のすべての画像を複数ページ PDF ファイルへ変換する完全なコードサンプルがクックブックの `images/multi_page_tiff` トピックにあります。

インライン画像 再利用可能画像は Image XObject として PDF 出力へ書き出されますが、これに対してインライン画像は各コンテンツストリーム（ページかパターンかテンプレートかグリフ定義）の中に直接書き込まれます。これにより若干の容量が節約できますが、容量の小さな画像データ（4 KB まで）での利用に留めるべきです。インライン画像の主な用途は Type 3 フォントのビットマップグリフ定義であり、インライン画像を他の場面で使用することは推奨しません。

インライン画像は `PDF_load_image()` と `inline` オプションで生成できます。インライン画像の再利用はできません。すなわちその対応するハンドルを画像ハンドルとして呼び出しに与えてはいけません。そのため、`inline` オプションが与えられると `PDF_load_image()` の内部動作は以下のコードと等価になります：

```
p.fit_image(image, 0, 0, "");  
p.close_image(image);
```

インライン画像は、`imagetype=jpeg·raw` でのみ対応しています。他の種類の画像では、`inline` オプションは静かに無視されます。

画像内の XMP メタデータ 画像ファイルは XMP メタデータを含んでいることがあります。デフォルトでは PDFlib は、出力ファイルサイズを削減するため、TIFF・JPEG・JPEG 2000 画像形式内の XMP メタデータを無視します。ただし、その画像メタデータは（有効な XMP をそれが持っているなら）、`PDF_load_image()` の以下のオプションで、出力 PDF 文書内の生成画像 XObject に添付することができます：

```
metadata={keepxmp=true}
```

8.1.2 対応画像ファイル形式

PNG 画像 あらゆる種類の PNG 画像 (ISO 15948) に PDFlib は対応しています。PNG 画像が透過情報を持っている場合、その透過は生成 PDF 内で保たれます (193 ページ「8.1.4 画像透過」参照)。

PNG 画像が sRGB チャンクを含んでいる場合には、その sRGB ICC プロファイルが画像に添付されます。ただし、*honoriccprofile* オプションが *false* である場合、または、*iccprofile* オプションで他の ICC プロファイルがその画像に割り当てられている場合を除きます。この sRGB チャンク内のレンダリングインテントが使用されます。ただし *renderingintent* オプションが与えられているときを除きます。

JPEG 画像 PDFlib は、以下の種類の JPEG 画像 (ISO 10918-1) に対応しています：

- ▶ グレースケール・RGB (通常は YCbCr 符号化されていますが、直接 RGB にも対応)・CMYK カラー。
- ▶ ベースラインおよびプログレッシブ JPEG 圧縮。

これらの条件は、実用されているすべての JPEG 画像を網羅します。上記の制約は、広く用いられていない JPEG の機能を使用している画像を取り込むことはできないことを意味します。とりわけ算術符号、ロスレス圧縮、構成要素あたり 8 ビット以外のビット深度、および SmartScale・可逆 RGB 色変換など非標準的機能がこれにあたります。なお、類似形式 JPEG-LS (ISO 14495、「ロスレス JPEG」と呼ばれる時もある)・JPEG-XR (ISO 29199-2、以前 HD Photo と呼ばれていた) には対応していません。

JPEG 画像は何種類かのファイル形式に納めることができます。よく利用されるあらゆる JPEG ファイル形式と機能に PDFlib は対応しています：

- ▶ JFIF 1。さまざまな画像処理アプリケーションによって生成されます。
- ▶ Adobe Photoshop などの Adobe アプリケーションによって出力される JPEG ファイル。
- ▶ PDFlib は、Adobe Photoshop で作成された JPEG 画像からクリッピングパスを読み取ります。
- ▶ PDFlib は、*honoriccprofile* オプションが *false* に設定されていない限り、JPEG 画像内の埋め込み ICC プロファイルに従います。
- ▶ JPEG 画像が Exif マーカーを含んでいる場合には、その Exif マーカー内の色空間情報は解釈されます。それが sRGB 色空間を示している場合には、その sRGB ICC プロファイルがその画像に添付されます (ただし、その画像が明示的に埋め込まれた ICC プロファイルを含んでいる場合、または *honoriccprofile* オプションが *false* である場合、またはその画像に *iccprofile* オプションで他の ICC プロファイルが割り当てられている場合は、この限りではありません)。
- ▶ 画像の望ましい向きを指定した、Exif マーカー内の Orientation エントリーに従います。これを無視する (多くのアプリケーション同様) には *ignoreorientation* オプションを用います。

JPEG 2000 画像 JPEG 2000 画像 (ISO 15444-2) には PDF 1.5 以上が必要です。PDFlib は、JPEG 2000 画像を、以下の条件に従って受け付けます：

- ▶ JP2・JPX ベースライン画像 (通常 **.jp2* または **.jpf*) に対応しています。ただし以下の色空間条件に従う必要があります。範囲 1 ~ 38 ビットのすべての色深度値に対応しています。

次の色空間に対応しています：sRGB・sRGB グレー・ROMM-RGB・sYCC・e-sRGB・e-sYCC・CIE Lab・ICC ベース色空間群・CMYK。PDFlib は、JPEG 2000 画像ファイル内の元の色空間に変更を加えません。

- ▶ (非サポート) 1 個・3 個・4 個の色要素を持つ、JPX ラッパーのない生の JPEG 2000 コードストリーム (しばしば **.j2k*) を取り込むことができます。
- ▶ JPEG 2000 画像に埋め込まれた限定またはフルの ICC プロファイルは保持されます。すなわち、*honoriccprofile* オプションはつねに *true* です。

注 ISO 15444-6 に従った JPM 複合画像ファイル (通常 **.jpm*) には対応していません。

PDF/X-4/5 に対する追加の JPEG 2000 の制約 (JPEG 2000 は、PDF 1.4 に基づいた PDF/X-3 では許されません) :

- ▶ カラーチャンネルの数は 1・3・4 のうちのいずれかである必要があります。
- ▶ 各カラーチャンネルのビット深度は 1・8・16 のうちのいずれかである必要があります。
- ▶ すべてのカラーチャンネルが同一のビット深度を持っている必要があります。
- ▶ ちょうど 1 個の色空間定義が JPEG 2000 画像ファイル内に存在する必要があります。
- ▶ CMYK 画像は、出力条件が CMYK デバイスである場合、または *defaultcmyk* オプションが与えられている場合にのみ使用できます。

PDF/A-2 に対する追加の JPEG 2000 の制約 (JPEG 2000 は、PDF 1.4 に基づいた PDF/A-1 では許されません) :

- ▶ カラーチャンネルの数は 1・3・4 のうちのいずれかである必要があります。
- ▶ すべてのカラーチャンネルが同一のビット深度を持っている必要があります。
- ▶ その JPEG 2000 画像内の色空間指定の数が 1 より大きい場合には、*APPROX* フィールド内に値 0x01 を持つ色空間指定がちょうど 1 個存在する必要があります。

JBIG2 画像 PDFlib は、単ページ・複数ページの JBIG2 画像 (ISO 14492) に対応しています。JBIG2 画像はつねに単色ピクセルデータを内容として持ちます。

JBIG2 圧縮の性質から、複数ページ JBIG2 ストリーム内のいくつかのページが同一のグローバルセグメントを参照している場合があります。複数ページ JBIG2 ストリームの 1 つないし複数のページが変換される際、グローバルセグメントは、生成 PDF 画像間で共用できます。*PDF_load_image()* への呼び出しは互いに独立ですので、あらかじめ PDFlib に、同一 JBIG2 ストリームから複数ページを変換すると知らせておく必要があります。これは以下のように行います :

- ▶ 先頭ページを読み込む際に、すべてのグローバルセグメントを PDF へコピーします。*PDF_load_image()* に対して以下のオプションリストを用います :

```
page=1 copyglobals=all
```

- ▶ 同一 JBIG2 ストリームから、以降のページを読み込む際に、ページ 1 に対する画像ハンドル *<N>* を与えることによって、すでにページ 1 とともにコピーされているグローバルセグメントへの参照を PDFlib が作成できるようにする必要があります。*PDF_load_image()* に対して以下のオプションリストを用います :

```
page=2 imagehandle=<N>
```

クライアントアプリケーション側では必ず、同一 JBIG2 画像ストリームから抽出されたページに対してのみこの *copyglobals/imagehandle* 機構が適用されるようにする必要があります。*copyglobals* オプションがない場合は、PDFlib は自動的にカレントページに対する必要データをすべてコピーします。

TIFF 画像 PDFlib は、すべての関連する種類の TIFF 画像を処理します :

- ▶ 圧縮方式：非圧縮・CCITT（グループ 3・グループ 4・RLE）・ZIP（= Flate）・PackBits（= RunLength）・LZW・新旧方式 JPEG に加え、いくつかのまれな圧縮方式。
- ▶ 色空間：単色・グレースケール・RGB・CMYK・CIE Lab・YCbCr 画像。取り込まれる TIFF 画像内の色空間は、以下の例外を除き、変更されずに保持されます：CIE Lab カラーを持つ LZW 圧縮された TIFF 画像は RGB へ変換され、CIE Lab 色空間を保持しません。
- ▶ 色深度は、色要素あたり 1・2・4・8・16 ビットのいずれかでなければなりません。16 ビット画像は PDF 1.5 を必要とします。
- ▶ 元の TIFF 形式を、4GB を超えて拡張した BigTIFF 形式。

画像を取り込む際、以下の TIFF 機能は処理されます：

- ▶ 複数の画像を含んだ TIFF ファイル（188 ページ「複数ページ画像」参照）。TIFF ファイル内の特定の画像を選ぶには *page* オプションを 사용합니다。
- ▶ アルファチャンネルまたはマスク（193 ページ「8.1.4 画像透過」参照）に、*ignoremask* オプションが設定されていない限り従います。*alphachannelname* オプションで明示的にアルファチャンネルを選ぶこともできます。
- ▶ PDFlib は、Adobe Photoshop や互換プログラムで作成された TIFF 画像内のクリッピングパスに、*ignoreclippingpath* オプションが設定されていない限り従います。
- ▶ PDFlib は、TIFF 画像内の埋め込み ICC プロファイルに、*honoriccprofile* オプションが *false* に設定されていない限り従います。
- ▶ JPEG 画像が Exif マーカーを含んでいる場合には、その Exif マーカー内の色空間情報は解釈されます。それが sRGB 色空間を示している場合には、その sRGB ICC プロファイルがその画像に添付されます（ただし、その画像が明示的に埋め込まれた ICC プロファイルを含んでいる場合、または *honoriccprofile* オプションが *false* である場合、またはその画像に *iccprofile* オプションで他の ICC プロファイルが割り当てられている場合は、この限りではありません）。
- ▶ 画像の望ましい向きを指定した Orientation タグに従います。これを無視する（多くのアプリケーション同様）には *ignoreorientation* オプションを 사용합니다。

BMP 画像 PDFlib は、以下の種類の BMP 画像に対応しています：

- ▶ BMP バージョン 2・3。
- ▶ 色深度はコンポーネントあたり 1・4・8 ビット。3×8 = 24 ビットの TrueColor を含みます。16 ビット画像は 5+5+5 プラス未使用 1 ビットとして扱われます。32 ビット画像は 3×8 ビット画像として扱われます（残りの 8 ビットは無視されます）。
- ▶ 白黒または RGB カラー（インデックスカラーも直接カラーも）。
- ▶ 非圧縮と 4 ビット・8 ビット RLE 圧縮。
- ▶ ピクセルがボトムアップ順で格納されている場合 PDFlib は画像を反転させません（この BMP の機能はめったに使われず、アプリケーションによって異なる解釈をされます）。

RAW 画像 非圧縮処理された RAW 画像データは、さまざまな特殊な用途に有用でしょう。なお、この形式は、Adobe アプリケーション群が生成する Camera Raw ファイルとは関係ありません。

RAW 画像データを与えるには、*width · height · bpc* オプションを与える必要があります。なぜなら PDFlib は画像データからその画像の寸法を導出できないからです。その色空間は *components* オプションから導出できます：1 成分ならグレースケール画像を暗示し、3 成分なら RGB 画像を、4 成分なら CMYK 画像を暗示します。あるいは、*colorize* オ

プッシュンを通じて、スポットカラーか DeviceN カラーを画像に適用することもできます。この場合には、その色空間と、色成分の数は、その色空間ハンドルから導出されます。与える画像データ長は

$[width \times components \times bpc / 8] \times height$

バイトに等しい必要があります。ここで、カッコ内の小数は切り上げです。画像サンプルは、上から下へ、かつ左から右への順序と見なされます（座標変換は行われていないと前提されます）。16ビットサンプルは、最上位バイトを最初に与える必要があります（ビッグエンディアンバイト順序）。*bpc* が 8 より小さい場合には、各ピクセル行はバイト境界で開始し、カラー値はバイト内で左から右へ詰め込まれる必要があります。カラーチャンネル群はつねに交互配置されます。すなわち、最初のピクセルに対するすべてのカラー値を最初に与える必要があります、その後 2 番目のピクセルに対するカラー値群を与え、以下同様とする必要があります。画像に合致するオプション値群を与えるのはユーザー側の役割です。そうしない場合には、壊れた PDF 出力が生成される場合があります、Acrobat がメッセージ「*Insufficient data for an Image*」を発するおそれがあります。

カラー値群の極性は、色関連オプション群に対する場合と同じです（83 ページ「4 章色と透過」参照）。なお、Adobe Photoshop は、期待される PDF 極性を、グレースケール画像と RGB 画像に対してのみ用いており、RAW CMYK 画像と追加カラーチャンネルに対しては反転極性を用いています：PDF と PDFlib は 0 = 強度なしと期待するのに対し、Photoshop は 0 = 最大強度と見なします。この極性を調整するには、その画像を読み込む際に *invert* オプションを適用します。

8.1.3 クリッピングパス

PDFlib は、Adobe Photoshop や互換ソフトで作成された TIFF・JPEG 画像の中のクリッピングパスに対応しています。1 つの画像ファイルには、複数の名前付きパスを含む場合があります。*PDF_load_image()* の *clippingpathname* オプションを使えば、名前付きパスのうち 1 つを選ぶことができ、それがクリッピングパスとして使われます：画像は、クリッピングパスの内部だけが可視となり、それ以外の部分は不可視になります。これは背景と前景を分離したり、画像の不要部分を除去したりするのに有用です。

あるいは、画像ファイルはデフォルトクリッピングパスを含む場合があります。PDFlib は、画像ファイル内にクリッピングパスを見つけた場合、それを自動的にその画像に適用します（図 8.1 参照）。デフォルトクリッピングパスが適用されないようにするには、*PDF_load_image()* で *honorclippingpath* オプションを *false* に設定します。同じ画像のインスタ



図 8.1
クリッピングパスを利用して
前景と背景を分離

ンスが複数あって、しかもそのうち一部のインスタンスにしかクリッピングパスを適用したくないときは、`PDF_fit_image()` に `ignoreclippingpath` オプションを与えることによってクリッピングパスを無効にすることができます。クリッピングパスが適用されると、画像の配置やはめ込みに関するすべての計算は、切り抜かれた画像の外接枠をもとに行われます。

クックブック 完全なコードサンプルがクックブックの `images/integrated_clipping_path` トピックにあります。クックブックトピック `path_objects/import_path_from_image` は、クリッピングパスを視覚化する方法を示しています。

クリッピングパスを記述するためのベクトル演算は、`PDF_fit_image()` が呼び出されるたびに、PDF 出力へ書き出されます。クリッピングパスを持った画像 1 個を、その文書内に複数回配置する場合には、出力ファイルサイズを削減するために、その画像をテンプレート内にラップすることを強く推奨します。これは `PDF_load_image()` の `templateoptions` オプションで実現できます。

8.1.4 画像透過

画像透過は、さまざまな芸術的效果などに有用です。たとえば、画像の中の不要な部分を無視して、関心の対象である人物や物だけを見せることができます。PDFlib は、画像透過のためのいくつかの手法に対応しています：

- ▶ アルファチャンネル（ソフトマスクとも呼ばれる）は、各画像ピクセルに対して透過値を指定するものです。このアルファチャンネルは、取り込む画像ファイルに含まれていることもありますし、別のグレースケール画像として指定することもできます。ピクセルあたり 1 ビットより多いソフトマスクは PDF/A-1・PDF/X-3 では許されていません。
- ▶ クロマキーマスク処理は、単色値またはカラー値範囲を透過と指定します。この透過カラー値（1 つないし複数）は、取り込む画像から来ることもありますし、`chromakey` オプションを通じて指定することもできます。
- ▶ ステンシルマスク処理は、透過領域を指定する別のビットマップ画像を用います。

表 8.1 に、アルファチャンネル・クロマキーチャンネル・ステンシルマスクを用いて画像をマスクする場合に利用可能な手法をまとめました。このマスクの極性は、アルファ方式とステンシル方式とで異なります：アルファマスクでは黒い領域を通じて背景が見えますが、ステンシルマスクでは白い領域を通じて背景が見えます。

これ以外にも、`colorize` オプション（197 ページ「8.1.5 画像にスポットカラーか DeviceN カラーで着色」参照）・ブレンドモード・`gstate` ソフトマスク（104 ページ「4.9 オブジェクトの色を変更」参照）を用いて、画像の色のさまざまな効果を実現できます。比較のため、画像にスポットカラーか DeviceN カラーで着色する効果も表 8.1 に挙げています。

クックブック 完全なコードサンプルがクックブックの `images/image_mask` トピックにあります。

内蔵アルファチャンネルを持った画像 PDFlib は、以下の画像形式から内蔵アルファチャンネルを読み取ります（`ignoremask=true` の場合を除きます）：

- ▶ TIFF 画像：1 個のアルファチャンネルを内容として持っていることがあり、PDFlib はそれを使用します。あるいは TIFF 画像は、名前で識別される複数のアルファチャンネルを含むこともできます。複数のチャンネルが TIFF 画像内に存在する場合には、PDFlib はデフォルトでは、その先頭のアルファチャンネルを使用します。他のチャンネルを明示的に選択することも可能です。そのためには、`alphachannelname` オプションを用いてその名前を与えます：

表 8.1 画像透過と画像着色で使える方式

マスク/着色手法	ベース画像	補助マスク画像	マスクまたはベース画像内のピクセル群の効果
画像または masked オプションからのアルファチャンネル (ソフトマスク)			
内蔵アルファチャンネルによる透過画像	内蔵アルファチャンネルを持つ TIFF・PNG・JPEG 2000 画像	(画像ファイルから)	(TIFF/PNG/JPEG 2000 画像形式に従った透過)
画像および別のアルファチャンネル画像	任意の画像。オプション masked で補助画像内のソフトマスクを参照	ビットマップまたはグレースケール画像	黒マスク=背景 グレーマスク=透け 白マスク=ベース画像 ¹
画像または chromakey オプションからのクロマキーマスク処理			
内蔵クロマキー値を持った画像	内蔵クロマキーエントリーを持った PNG 画像	—	クロマキー色=透過 画像のその他の色=ベース画像
画像および別のクロマキー値または範囲	JPEG・JPEG 2000 以外の任意の画像。オプション chromakey で透過カラー値 (1 つないし複数) を指定	—	クロマキー色 (1 つないし複数) =透過 画像のその他の色=ベース画像
mask オプションを用いたステンシルマスク処理			
ステンシルとしてのマスク画像を通して描画	—	オプション mask を用いて読み込まれたビットマップ画像	黒マスク=塗り色 白マスク=背景 ¹
ステンシル画像を用いてベース画像をマスク	任意の画像。オプション masked で補助画像内のステンシルマスクを参照	オプション mask を用いて読み込まれたビットマップ画像	黒マスク=ベース画像 白マスク=背景 ¹
一定値の不透明度を用いた、全体的に透明な画像			
画像全体に一定値の不透明度を適用	任意の画像	—	全ピクセルが同一の透過度で表示されます
colorize オプションを用いた画像着色			
スポットカラーを用いて画像に着色	グレースケール画像 (ピクセルあたり 1 ビット以上)。オプション colorize でスポットカラーを参照	—	黒画像=ベータスポットカラー グレー画像=薄めたスポットカラー 白画像=白
DeviceN カラーを用いて画像に着色	n 色画像。オプション colorize で DeviceN カラーを参照	—	画像内の 0 = 白 中間 = 薄めた DeviceN 画像内の 1 = ベータ DeviceN カラー

1. ステンシルまたはアルファ画像に invert オプションを適用することによってマスクの効果を反転させることもできます。

```
image = p.load_image("tiff", filename, "alphachannelname={apple}");
```

- ▶ PNG 画像: 1 個のアルファチャンネルを内容として持っていることがあり、PDFlib はそれを使用します。
- ▶ JPEG 2000 画像: 1 個のアルファチャンネルを内容として持っていることがあり、PDFlib はそれを使用します。

注 Photoshop は、完全なアルファチャンネルのかわりに、TIFF 画像内に透過背景を作成することもできます。これに使用される独自形式に PDFlib は対応していません。このような

透過画像を PDFlib で使うには、Photoshop でこれを TIFF ファイル形式で保存し、その際に「TIFF オプション」ダイアログボックスで「透明部分を保持」を選択する必要があります。

別のグレースケール画像をアルファチャンネルとして使用 画像ファイル内の内蔵アルファチャンネルのかわりに、第二の画像をアルファチャンネルのソースとして使うこともできます。グレースケール画像は、すべての種類が、アルファチャンネルとして使用するのに適しています。このマスクが埋め込み ICC プロファイルの内容として持っている場合には、*honoriccprofile=false* を用いてこれを無視する必要があります。TIFF 画像に対しては、*PDF_load_image()* に *nopassthrough* オプションを与えることによってマルチストリップ画像を避けることを推奨します。BMP 画像は他の画像種別とは向きが異なっています。ですので BMP 画像をマスクとして使用できるようにするには、まず *x* 軸を軸として鏡映させる必要があります。

アルファ画像内の白いピクセルの部分においては、ベース画像の対応する領域が描画され、黒いマスクピクセルの部分においては背景を見通せます。マスクがピクセルあたり 1 ビットよりも多くを使用している場合には、中間値は、前景画像を背景に対してブレンドさせることによって、透け効果を生みます。

マスクを読み込んだ後、それをベース画像に適用するには、*masked* オプションを用います：

```
// アルファチャンネルとして使いたいマスク画像を読み込む
mask = p.load_image("png", maskfilename, "");
if (mask == -1)
    throw new Exception("エラー：" + p.get_errmsg());

// ベース画像を読み込んでそれをマスク
image = p.load_image(type, filename, "masked=" + mask)
if (image == -1)
    throw new Exception("エラー：" + p.get_errmsg());

p.fit_image(image, x, y, "");
```

画像アルファチャンネルを視覚化 ときには画像アルファチャンネルをグレースケール画像として見せたいこともあるでしょう。これはデバッグや、アルファチャンネルの再利用に役立つでしょう。これを実現するには、マスクへの画像ハンドルをメイン画像から取得します。それには *PDF_info_image()* とキーワード *imagemask* を用います：

```
image = p.load_image("auto", "image.tif", "errorpolicy");

alpha = (int) p.info_image(image, "imagemask", "");
if (alpha != -1)
    p.fit_image(alpha, 0.0, 0.0, "");
```

ここでアルファチャンネルはグレースケール画像として扱われます：透過領域が黒く見えます。

クロマキーマスク処理 画像は、マスクされるべき単色または色範囲を指定することができます。指定された範囲の色を持つピクセルは描画されず、したがって背景が透けて見えます。映像技術方面ではこの技法はクロマキーまたはブルースクリーンマスク処理と呼ばれています。これは *chromakey* 画像オプションで使えます。このオプションは、画像の色空間の色要素の数を *n* としたとき、範囲 $0 \sim 2^{\text{色要素あたりビット数}} - 1$ の *n* 個または $2 \times n$ 個の整数を受け付けます。

このリストの中の各ペアは、色要素範囲の、境界を含む下限と上限を内容とします。すべての色要素 (*decode* 値群も *invert* オプションを用いた色反転もまだ適用していない段階の) がこの指定範囲群に収まっているところのピクセルは透過として扱われ、すなわち描画されずに背景が透けて見えます。

n 個のペアでなく n 個の値を与えた場合には、各要素範囲は 1 個の単色値のみが内容となります。すなわち、各リスト値は色要素に対する下限と上限の両方を記述していることとなります。

chromakey オプションのさまざまな例を表 8.2 に挙げます。

表 8.2 chromakey 画像オプションの使用例

クロマキーオプション	画像の色空間	効果
<code>chromakey={255 255 255}</code>	RGB	白ピクセルを透過として扱う
<code>chromakey={0 255 128 255 0 255}</code>	RGB	50% を超えるグリーンを持つすべてのピクセルを透過として扱う
<code>chromakey={242 255 242 255 242 255}</code>	RGB	95% より明るいすべての色を透過として扱う
<code>chromakey={242 255 242 255 242 255}</code> <code>decode={0 0.95 0 0.95 0 0.95}</code>	RGB	95% より明るいすべての色を透過として扱い、かつ、95% で急に切り落とされるのを防ぐために残りの色を拡散させる

クロマキーマスク処理は、ただ 1 つの透過カラー値を持っている PNG 画像に対しては自動的に適用されます (*ignoremask=true* でない限り)。複数のカラー値が、対応するアルファ値とともに存在している場合には、50% 未満のアルファ値を持つ最初のもののみが用いられます。

ステンシルマスク ステンシルマスクは、白いピクセルが透過として扱われる、ビット深度 1 のビットマップ画像です：ページの既存内容が画像内の透過部分を通して見えます。0 ピクセル値を持つ領域は、カレント塗り色で描くこともできますし、それを用いて別の画像の一部を見せることもできます。

ステンシルマスクをカレントの塗り色で着色するには、その画像を *mask* オプションを用いて読み込み、そしてその画像を配置する前にカレント塗り色を設定する必要があります：黒いピクセルがカレント塗り色で着色され、白い領域は変更されません：

```
mask = p.load_image("tiff", maskfilename, "mask");
if (mask == -1)
    throw new Exception("エラー : " + p.get_errmsg());

p.set_graphics_option("fillcolor=red");
p.fit_image(mask, x, y, "");
```

もしもこのステンシル画像を使って他の画像を *masked* オプションでマスクした場合には、ステンシルマスクの黒い領域においてはそのベース画像のピクセルが見え、白い領域においては背景が見えます。

一定値の不透明度を適用することによって画像を全体的に透明にする 画像を配置する際に、全ピクセルに対して一定値の不透明度を適用すると、その画像全体を透明にすることができます (背景画像として使うなどのために)。これを実現するには *gstate* オプションと *opacityfill* サブオプションを用います。例：

```
p.fit_image(mask, x, y, "gstate={opacityfill=0.5}");
```

8.1.5 画像にスポットカラーか DeviceN カラーで着色

ステンシルマスクでは、ビットマップ画像の不透過部分に色が適用されましたが、これと同様に、PDFlib では、画像にスポットカラーか DeviceN カラーで着色することもできます。

グレースケール画像にスポットカラーで着色 白黒またはグレースケール画像にスポットカラーで着色することができます。画像にスポットカラーで着色するには、その画像を読み込む際に `colorize` オプションを与える必要があります。このオプションは、`PDF_makespotcolor()` を用いて作成されたスポットカラーハンドルを内容とします：

```
spot = p.makespotcolor("PANTONE Reflex Blue CV");  
  
String optlist = "colorize=" + spot;  
image = p.load_image("tiff", "image.tif", optlist);
```

91 ページ「4.4 Pantone・HKS・カスタムスポットカラー」で説明しているように、スポット色空間は通常、0 = 色なし = 白と期待します。これに対し、グレースケール色空間では 0 = 黒です。白を白のままにするために、PDFlib は、画像にスポットカラーで着色する際には、画像の極性を反転させます。すなわち、画像の暗い領域が最大のスポットカラー強度になります。この色極性を反転させるには `invert` 画像オプションを用います。

n チャンネル画像を DeviceN カラーで着色 `PDF_create_devicen()` を用いて作成された DeviceN カラーハンドルを、`colorize` 画像オプションで与えると、DeviceN カラーを n 色画像に適用することができます。画像を DeviceN カラーで着色するためには、その画像データが、ちゃんと N 個のカラーチャンネルを内容として持っている必要があります。

94 ページ「4.5 DeviceN カラー」で説明しているように、DeviceN 色空間は 0 = 色なし = 白と期待します。これに対し、グレースケール色空間では 0 = 黒です。画像をスポットカラーで着色した場合と異なり、PDFlib は、画像を DeviceN カラーで着色する際には、画像の極性を反転させません。ですので、スポットカラーで着色する場合と、**N=1** の DeviceN カラーで着色する場合とは、期待される色極性が異なります。この色極性を反転させるには `invert` 画像オプションを用います。

画像に DeviceN 色空間で着色するコード断片が 95 ページ「CMYK プロセッサカラーに基づく DeviceN 色空間」にあります。

注 DeviceN カラーで着色できるのは RAW 画像だけです。

クックブック コードサンプルがクックブックの `color/colorize_image_with_DeviceN_color` トピックにあります。

8.1.6 復号配列を用いてカラー値を変える

画像のカラー値は、線形の復号関数を用いてさらに変えることもできます。この方式は、`decode` 画像オプションを用いて使えます。これは、画像の色空間の要素数を n としたときに、 $2 \times n$ 個の float またはパーセント値を受け付けます。ですので、復号配列を適用するには、画像の色空間に関する情報が必要です。復号配列内の数値の各ペアは、色要素値の下限と上限である 0 と 2^{色要素あたりビット数} がマップされるターゲットカラー値を記述します：中間の値は線形に内挿されます。デフォルトでは、たいていの色空間の場合、画像の色要素値は範囲 0 ~ 1 へマップされます。復号配列を利用すると、カラー値の範囲を縮めたり広げたり限ったりすることによって、特定のカラー効果を適用することが可能で

す。この技法は、画像が着色されている場合や、マスクとして用いられている場合にも役立つでしょう。

復号値は、加法混色の色空間の場合と、減法混色の色空間の場合とで、効果が異なります。たとえば、値を大きくするほど、RGB カラーでは明るくなり、CMYK カラーでは暗くなります。たいていの色空間では (Lab・インデックスを除き)、典型的な値は範囲 0 ~ 1 の中にあります。この範囲の外の値も許されますが、ビューアーは結果のカラー値を、その色空間で適切な範囲へ切り捨てます。これを利用すると特定の効果が得られます。たとえば、ある範囲の明るさの色を強制的に白にするなどです。

別の画像へのソフトマスクとして用いられている画像に `decode` オプションを適用すると、そのマスクの効果を変えることができます。たとえばそのマスクを和らげたりすることができます。

以下の画像オプションは、ブラックのターゲット値を、デフォルト範囲 0 ~ 1 から、新しい範囲 0.2 ~ 1.2 へずらさせています。すなわち、CMYK 画像のブラックチャンネルを 20% 増加させ、画像を強めています：

```
decode={0 1 0 1 0 1 0.2 1.2}
```

`decode` オプションのさまざまな例を表 8.3 に挙げます。

表 8.3 `decode` 画像オプションの使用例

復号オプション	画像の色空間	効果
<code>decode={1 0 1 0 1 0}</code>	RGB	画像の色を反転。invert と同じ
<code>decode={-0.5 1.5 -0.5 1.5 -0.5 1.5}</code>	RGB	コントラストを強める
<code>decode={0.5 1.5 0.5 1.5 0.5 1.5}</code>	RGB	すべての色を 50% 明るくする
<code>decode={-0.2 0.8 -0.2 0.8 -0.2 0.8}</code>	RGB	すべての色を 20% 暗くする (100% = 白なので)
<code>decode={-0.2 0.8 -0.2 0.8 -0.2 0.8 -0.2 0.8}</code>	CMYK	すべての色を 20% 明るくする (100% = 暗なので)
<code>decode={0 1 0 1 0 1 0 0}</code>	CMYK	ブラックチャンネルを除去。鈍い画像になります
<code>decode={0 1 0 1 0 1 0.2 1.2}</code>	CMYK	ブラックチャンネルを 20% 大きくする。強い画像になります
<code>decode={1 0 0 1 0 1 0 1}</code>	CMYK	シアンチャンネルを反転させ、他のチャンネルは変更しない。芸術向け以外に有用性はなさそう

8.2 SVG グラフィック

8.2.1 対応 SVG 種別

PDFlib は、W3C の述語によれば「規格準拠した高品位な静的 SVG ビューアー」です。PDFlib は、SVG グラフィックを以下のように受け付けます：

- ▶ PDFlib は、W3C が発行した SVG 1.1 (Second Edition) を実装しています。
- ▶ 以下の Unicode 形式とエンコーディングに対応しています：
UTF-8・UTF-16・ISO 8859-1 ~ ISO 8859-15・ASCII
- ▶ CSS スタイル付けが可能ですが、ただし対応していない CSS 要素もあります。
- ▶ プレーンテキスト形式の SVG ファイルのほかに、Flate 圧縮された SVG ファイル (*.svgz) にも対応しています。
- ▶ 色は、SVG Color 1.2 ドラフトに従った追加の色空間を用いて指定することも可能です。スポットカラーの濃度値のための PDFlib 独自拡張の色空間も利用可能です (205 ページ「8.2.6 SVG カラー拡張」参照)。



制約について 209 ページ「8.2.8 対応していない SVG 機能」を参照してください。

8.2.2 SVG 処理上の考慮事項

基本的な SVG の取り扱い ベクトルグラフィックを PDFlib で埋め込むことは容易に実現できます。まず、そのグラフィックファイルを PDFlib のメソッドで開く必要があります。このメソッドはそのグラフィックを解釈して、内部表現をメモリー内に格納します。この関数 `PDF_load_graphics()` は、グラフィック記述子の役割を持つハンドルを返します。このハンドルは、`PDF_fit_graphics()` への呼び出しで、位置付け・拡張オプションとともに使うことができます：

```
graphics = p.load_graphics("auto", "graphics.svg", "");
if (graphics == -1)
    throw new Exception("エラー : " + p.get_errmsg());

if (p.info_graphics(graphics, "fittingpossible", optlist) == 1)
    p.fit_graphics(graphics, 0.0, 0.0, "");
else
    System.err.println("グラフィックを配置できません : " + p.get_errmsg());

p.close_graphics(graphics);
```

`PDF_fit_graphics()` の最後の引数は、位置付け・拡張・回転のためのさまざまなオプションをサポートしているオプションリストです。これらのオプションに関する詳細は、221 ページ「8.4 画像・グラフィック・取り込み PDF ページを配置」で説明します。

クックブック SVG の取り扱いのためのコードサンプルが、PDFlib クックブックの `starter_svg` ミニサンプルと `graphics` カテゴリーにあります。

SVG 内容をインライングラフィックとするかテンプレート (フォーム XObject) とするか PDFlib は、取り込んだ SVG ベクトルグラフィックについて、2 種類の PDF 出力を生成できます：

- ▶ デフォルトではPDFlibは、取り込んだSVGグラフィックそれぞれに対してテンプレート (フォームXObject) を1つずつ生成します。その中には必要なグラフィック操作がすべて入っています。このテンプレートはデフォルトのオプション群を用いて生成されますので、カスタムのオプション群が必要な場合には、`PDF_load_graphics()` の `templateoptions` オプションを用いて与えることもできます。テンプレートは透過の動作を明確に定義しています。これは、SVGグラフィックの透過の要素がその同じSVGの中の他の要素と相互作用する場合、またはこの配置したグラフィックに不透明を適用することによってそのSVGを通じて既存のページ内容が部分的に見えるようにしたい場合に重要です。また、グラフィックが複数回配置される場合には、テンプレートによってPDF出力は小さく済みます。テンプレート方式の難点としては、SVGグラフィック内のリンクはPDFの注釈へ変換されません。テンプレートが生成されるとはいえ、返されるSVGハンドルをテンプレートハンドルとして使用することはできないので注意してください。
- ▶ `PDF_load_graphics()` の `inline` オプションを `true` にすると、そのグラフィックのデータは、ページ・パターン・テンプレート・グリフ記述の内容ストリームの中にインラインで書き込まれます。このインライン方式が推奨される場合としては、SVGグラフィック内のリンクをPDFの注釈へ変換したい場合です (`PDF_fit_graphics()` の `convertlinks` オプションを参照)。しかし、これにはいくつかの難点もあります。グラフィックの中で透過なオブジェクトが重なり合っている場合には、出力の見え目がおかしくなることがあります。また、配置したグラフィックの不透過度を正しく変更できません。`PDF_fit_graphics()` を複数回呼び出すと、そのグラフィックのデータはPDF出力へ何度も書き込まれるので、出力のファイルサイズが膨らみます。

同一のグラフィックを複数のPDF文書内で使用 グラフィックは、カレント出力文書とは独立に読み込んだり閉じたりすることもできます。`PDF_load_graphics()` が呼び出された時点で、そのグラフィックの内部表現が作成されます。これは、その対応する `PDF_close_graphics()` への呼び出しまでメモリー内に保持されます。グラフィックを、文書をまたいでメモリー内に保持することは、同一のグラフィックが複数の出力文書内に配置される場面において、そのグラフィックを1回だけ読みこめば済むので、パフォーマンス上の利点があります。たとえば、アプリケーションが、シンボルや背景アートワークや企業ステーションナリを持つグラフィックを1回読み込んで、そのグラフィックが必要とされる各文書内で `PDF_fit_graphics()` を呼び出すことができるでしょう。

SVGの処理上の問題をチェック `PDF_load_graphics()` はSVGグラフィックを読み込みますが、完全な処理と分析はその後、フォームXObjectの作成と、読み込みのスコープに応じて、`PDF_fit_graphics()`・`PDF_close_graphics()`・`PDF_end_document()` のうちのいずれかの時点でしか行われません。エラー状況のなかには、完全処理が行われている間にしか検出できないものもありますので、これらの関数は、問題が発見されれば例外を発生させる可能性があります (なぜならこれらはエラー値を一切返すことができないので)。そのような例外を回避するには、グラフィックを `PDF_info_graphics()` の `fittingpossible` キーワードでチェックすることができます。これは、すべての処理ステップを実行しつつも、出力を一切作成せず、SVG処理の成功 (か否か) を報告します。もしこのチェックが成功すれば、その画像が配置される際に `PDF_fit_graphics()` は例外を発生させません。もしこの `fittingpossible` チェック中にエラーが発生すれば、`errorpolicy` にかかわらず、`PDF_info_graphics()` は0を返します。まとめると：

- ▶ この `fittingpossible` チェックは、後の `PDF_fit_graphics()`・`PDF_close_graphics()`・`PDF_end_document()` での例外を回避します。PDF 出力は例外の後では使用不能になりますので、これは推奨されるアプローチです。
- ▶ この `fittingpossible` チェックを省くと、SVG の読み込みは速くなりますが、その SVG データが引き起こす例外が後で発生するおそれがあります。この設定は、`PDF_fit_graphics()` での例外が許容できる場合に、SVG の読み込みを速めるために使用できます。たとえば、アプリケーションが 1 個の SVG グラフィックファイルを 1 個の PDF 文書へ変換し、それ以外のページ内容を追加しない場合には、これは許容できるアプローチです。

この `fittingpossible` チェックは、カレントでアクティブなグローバル・文書・ページオプション群と、カレント出力インテントを使用します。ですのでこのチェックは、`PDF_fit_graphics()` への実際の呼び出しの直前にのみ走らせることを推奨します。

ネットワークリソース SVG グラフィックは、そのグラフィックファイルの中の URL を通じて、外部のフォントや画像などリソースを参照することができます。すなわち、そのフォントや画像がローカルのマシンへダウンロードされます。ネットワーク動作の詳細は、`PDF_set_option()` の `network` オプションを用いて制御できます。以下のプロトコルに対応しています：`file`・`http`・`https`・`ftp`。すべてのネットワーク動作を抑止するには `network` オプションの `disable` サブオプションを使用します。

複数の SVG グラフィックが同一のネットワークリソースを利用していることがわかっている場合には、`PDF_load_graphics()` の `downloadlifetime` オプションを用いてキャッシングを活用できます。

注 ネットワークリソースは、IBM システム i では対応していません。

8.2.3 SVG グラフィックの寸法

SVG グラフィックは、その幅と高さを、その SVG グラフィックのターゲットビューポート（ブラウザーウィンドウや、PDF ページの一部分など）へのマッピングを定義している `svg` エレメント内で指定します。しばしば、このビューポートの寸法は絶対単位で指定されます。例：

```
<svg xmlns="http://www.w3.org/2000/svg" width="640mm" height="480mm">
```

PDFlib は、この `width`・`height` 属性をポイントへ変換し、そしてそれらを、`PDF_info_graphics()` の `graphicswidth`・`graphicsheight` キーワードを通じて得られるようにします。寸法がピクセル (`px`) で指定されている場合には、PDFlib は `1pt=1px` を用います。これらの値は、はめ込み操作のためのオブジェクト枠を計算するためにも使われます。`svg` エレメントは、ビューポート内のウィンドウを指定する `viewBox` 属性を内容として持つ場合もあります。

SVG グラフィック内で指定されている寸法値を、`forcedwidth`・`forcedheight` オプションを用いてオーバーライドすることもできます。`matchbox` オプションのサブオプション `clipping` を用いると、グラフィックをオブジェクト枠の一部へ切り出すこともできます。

絶対寸法値のない SVG グラフィック SVG グラフィックのなかには、`width`・`height` を欠いているか、あるいは以下の例のように相対値のみを内容としているために、絶対寸法情報を内容として持たないものもあります：

```
<svg xmlns="http://www.w3.org/2000/svg" width="100%" height="100%">
```

これはスタンドアロン用途を想定されていない SVG グラフィックで用いられています。PDFlib はこの場合にも対応します。それにはユーザーがオプション `boxsize` と `fitmethod=nofit` を用いてはめ込み枠を指定しておく必要があります。このはめ込み枠はオブジェクト枠として用いられます。その SVG グラフィックはこのはめ込み枠の辺で切り抜かれます。はめ込み枠を指定されていない場合には PDFlib は、`viewBox` 属性 (あれば) をオブジェクト枠として用います。SVG グラフィック内で絶対寸法情報が得られないときは、`PDF_load_graphics()` の `fallbackwidth`・`fallbackheight` オプションを与えることができます。これらのオプションが与えられていないときは、PDFlib は、その SVG グラフィックの外接枠を、`PDF_fit_graphics()` か `PDF_info_graphics()` への最初の呼び出しの際に算出します。算出される外接枠は、線幅と過大なグリフが考慮されていないので、小さすぎる場合があります。この場合には、この算出された枠を、`bboxexpand` オプションを用いて広げることができます。デフォルトでは、この算出された外接枠は、そのグラフィックが座標系の中の元の場所に位置付けられるよう移動されます (すなわち、この枠は必ずしも原点を隅として用いるわけではありません)。

8.2.4 フォント選択

フォント選択アルゴリズム SVG におけるフォント選択は、以下のプロパティによって制御されます：

```
font-family
font-style
font-weight

font-stretch
font-variant
font-size
font-size-adjust
```

これらのプロパティのうち、最初の 3 つだけが、外部フォントの選択に関連します。適切なフォントを選択するため、PDFlib は以下のフォント名を構築します：

```
<font-family>,<font-weight>,<font-style>
<font-family>-<font-weight><font-style>
<font-family>,<font-normweight>,<font-style>
<font-family>,<font-weight>,<font-normstyle>
<font-family>,<font-normweight>,<font-normstyle>
```

ここで `<font-normweight>` は

Regular, Thin, Extralight, Light, Medium, Semibold, Bold, Extrabold, Black

のうちのいずれか一つであり、また `<font-normstyle>` は

Italic

たとえば、以下の SVG フォント指定に対して：

```
font-family="Tahoma" font-weight="Bold" font-style="Italic"
```

PDFlib は、Windows ホストフォントを指定するためにも用いることができるフォントスタイルを指定するためのこのカンマ区切り PDFlib 文法を用いて、フォント *Tahoma,Bold,Italic* を検索します。

PDFlib はその後、処理が成功してフォントが読み込めるまで、上に挙げたフォント名を順次読み込み試行します。このリスト内のフォント名は、フォントリソース指定でも用いることができます。例：

```
p.set_option("FontOutline={<fontname>=<filename>}")
p.set_option("FontNameAlias={<fontname>=ArialMT}")
```

すべての試みが失敗した場合には、PDFlib は、名前 **<font-family>** を持つフォントを読み込み試行して、必要に応じて **Bold・Italic** プロパティを擬似表現します。

ブラウザによっては、指定されたフォントファミリーが見つからない場合には、フォント選択プロパティ群を無視するものがあります。PDFlib はそういうことをしませんので、PDFlib のフォント構成機構を通じて適切なフォントが得られるように配慮する必要があります (135 ページ「6.3 フォントを読み込む」参照)。

この **font-family** プロパティは、複数のフォントファミリー名を内容とする場合もあります。例：

```
font-family="Georgia, 'Minion Web', 'Times New Roman', Times, 'MS PMincho', serif"
```

この場合、PDFlib は、このリスト内の特定のフォントが読み込めなかったときには、その次のフォントを読み込み試行します。1 つの **font-family** リストに対して何らかのフォントが読み込めた場合には、PDFlib は、このリスト内の残りの **font-family** 群を、最初に読み込んだフォント (マスターフォント) に対するフォールバックフォント (146 ページ「6.3.6 フォールバックフォント」参照) として読み込もうと試みます。もしこのマスターフォントが、もっと前に読み込まれていたためにすでにフォールバックフォント群を持っているときには、新しいフォールバックフォント群は、既存のフォールバックフォントのリストの末尾に追加されます。

PDFlib が SVG グラフィックのためのフォントを読み込むと試みる際には、以下のオプションでデフォルトで用いられます：

```
embedding subsetting skipembedding={fstype latincore}
```

これらのオプションをオーバーライドするには、**PDF_load_graphics()** のオプション **defaultfontoptions** を用います。

フォント構成 Windows システムでは、PDFlib は、システムにインストールされているすべてのフォントにアクセスできます (144 ページ「6.3.5 Windows・macOS 上のホストフォント」参照)。たとえば、SVG フォント指定

```
font-family="Verdana" font-weight="bold"
```

は、PDFlib フォント名 **Verdana,Bold** となります。他のオペレーティングシステムでは、PDFlib は、**FontOutline** リソースが以下の方式で指定されていればフォントを発見します：

```
<fontnamepattern>=<filename.xxx>
```

ここで、**<fontnamepattern>** は、上述のフォント名パターン群のうちの一つであり、また **xxx** は、その対応する、そのフォントアウトラインファイルのフォント名拡張子です。

上述のフォント名パターン群のうちの一つないし複数に整合するフォント名を持つ適切な **FontOutline** リソースは、**PDF_set_option()** の **enumeratefonts** オプションで自動的に作成することもできます。フォント構成に関して詳しくは 141 ページ「6.3.4 フォントを検索」を参照してください。

総称 SVG フォントファミリーを PDF コアフォントへマップ PDFlib は、以下の形の `FontNameAlias` リソースを用いて、総称 SVG フォントファミリー `monospace · sans-serif · serif` を、総称 SVG フォントファミリーが最初に出現した時点で、自動的に Latin コアフォントへマップします：

```
p.set_option("FontNameAlias={monospace=Courier}")
p.set_option("FontNameAlias={monospace,Bold=Courier-Bold}")
```

総称フォント名マッピングの完全なリストは以下のとおりです (総称フォントファミリー `cursive` と `fantasy` に対してはデフォルトマッピングはありません)：

<code>monospace</code>	<code>Courier</code>
<code>monospace,Bold</code>	<code>Courier-Bold</code>
<code>monospace,Italic</code>	<code>Courier-Oblique</code>
<code>monospace,Bold,Italic</code>	<code>Courier-BoldOblique</code>
<code>sans</code>	<code>Helvetica</code>
<code>sans,Bold</code>	<code>Helvetica-Bold</code>
<code>sans,Italic</code>	<code>Helvetica-Oblique</code>
<code>sans,Bold,Italic</code>	<code>Helvetica-BoldOblique</code>
<code>sans-serif</code>	<code>Helvetica</code>
<code>sans-serif,Bold</code>	<code>Helvetica-Bold</code>
<code>sans-serif,Italic</code>	<code>Helvetica-Oblique</code>
<code>sans-serif,Bold,Italic</code>	<code>Helvetica-BoldOblique</code>
<code>serif</code>	<code>Times-Roman</code>
<code>serif,Bold</code>	<code>Times-Bold</code>
<code>serif,Italic</code>	<code>Times-Italic</code>
<code>serif,Bold,Italic</code>	<code>Times-BoldItalic</code>

これらのマッピングは、以前にユーザーによって他の適切なリソースが一切指定されていない場合にのみ実行されます。

8.2.5 見つからないフォント、見つからないグリフを扱う

見つからないフォントとデフォルトフォント すべてのフォント読み込み試行が失敗した場合、PDFlib は、`PDF_load_graphics()` の `defaultfontfamily` オプションで定義された `font-family` 名を持つデフォルトフォントを読み込もうと試みます。デフォルトでは、`Arial Unicode MS` フォントがもし得られるならそれが、そうでないなら `Helvetica` が用いられます。PDF のフォント構成の中で `Arial Unicode MS` フォントが得られるようにしておくか、あるいは、大きなグリフ集合を持つ別のフォントを `defaultfontfamily` で指定しておくことを強く推奨します。たとえば、`NotoSans-Regular` フォントを最終手段フォントとして構成するには、以下のオプションを用います：

```
defaultfontfamily={NotoSans-Regular}
```

個別のフォントを置換 入手できない、あるいは望ましくない (たとえば、十分なグリフを含んでいないという理由で。後述) 特定のフォントを避けるために、より適切な別のフォントへそれをマップすることができます。フォント名エイリアス機能と `PDF_set_option()` をこの目的に使用します (詳しくは 141 ページ「フォント名エイリアス設定」を参照)。たとえば、中国語のテキストが不適切に、中国語のグリフを一切含んでいない

Trebuchet MS フォントで設定されている場合に、これを以下のようにして *Arial Unicode MS* へマップすることができます：

```
p.set_option("FontnameAlias={ {Trebuchet MS}={Arial Unicode MS} }");
```

フォント属性は自動的に追加されないことに留意してください。たとえば、もしこの *Trebuchet MS* フォントが属性 *font-weight="bold"* で使われているならば、このフォントのボールド版へのエイリアスを作成する必要があります：

```
p.set_option("FontnameAlias={ {Trebuchet MS,Bold}={Arial Unicode MS} }");
```

見つかからないグリフを視覚化 選択されたフォントが、必要なグリフを含んでいない場合には、デフォルト置換グリフがかわりに用いられますので、テキストは全く見えなくなります。見つかからないグリフを視覚化するには、*defaultfontoptions* オプションを用いて、見える置換グリフを指定することができます。たとえば、*PDF_load_graphics()* に対する以下のオプションは、すべての見つかからないグリフに対して疑問符を表示します：

```
defaultfontoptions={replacementchar=?}
```

見つかからないグリフに対して特定のフォールバックフォントを指定 SVG 内で指定されているフォントが、そのテキストのための適切なグリフを含んでいない場合には、テキストが全く見えなくなります。よくある例として、中国語のテキストを、中国語のグリフを一切含んでいない欧文フォントで見せようとしている例を考えてみましょう。もちろん最善の解決策は、SVG 内でそもそも適切なフォントを用いることです。しかし、SVG 内の不適切なフォントを扱わねばならない場合には、PDFlib でフォールバックフォントを指定することができます。このフォールバックフォントは、元のフォントが特定のグリフを与えないときに用いられます。

PDF_load_graphics() に対する以下のオプションは、Arial Unicode MS をフォールバックフォントとして指定しています：

```
defaultfontoptions={fallbackfonts={{fontname={Arial Unicode MS} encoding=unicode}}}
```

なお、さきに説明した *defaultfontfamily* オプションで指定されたフォントは、フォントが見つかからないときに使用されるのに対して、この予備技法は、フォントは得られるが必要なグリフを全部は含んでいない場合にあてはまります。

グローバルなフォールバックフォントファミリーを指定 *fallbackfontfamily* ・ *fallbackfontoptions* オプションを用いると、フォールバックフォント群のファミリーと、対応するオプション群を指定することができます。

defaultfontoptions 内の *fallbackfonts* オプションは、ただ1つのフォントをフォールバックフォントとして使用するよう選択するのに対して、*fallbackfontfamily* を用いると、フォールバックフォント群のファミリーを指定することができます。すなわち、スタイル属性群がこのフォントファミリー名に適用されます。ただし、指定されたフォントのスタイル変種が実際に得られることを前提としています。例：

```
fallbackfontfamily={Arial}
```

8.2.6 SVG カラー拡張

クックブック 非sRGBカラーのためのコードサンプルがPDFlibクックブックのトピックcolor/svg_color_extensionにあります。

SVG 1.1 はデフォルトでは、テキストとベクトルグラフィックに対して sRGB 色空間を使用しています。ですので PDFlib は、PDF/X と PDF/A に対しては、SVG グラフィックをデバイス依存カラーとして扱います。しかし、SVG 内に埋め込まれている画像は他の色空間を使用している場合もあります。たとえば、埋め込まれているか参照されている JPEG 画像が CMYK 色空間を使用しているかもしれず、それは ICC プロファイルを有しているかもしれないし、有していないかもしれません。RGB カラーに対するデフォルト sRGB プロファイルは、`PDF_load_graphics()` の `iccprofilergb` オプションを用いてオーバーライドすることもできます。

sRGB に加えて、PDFlib は、SVG Color 1.2 ドラフト仕様に従った文法拡張を使用した追加の色空間に対応しています。表 8.4 に、使える SVG 色空間を、文法例とともに挙げます。これらの色空間に関する一般的な情報と、PDFlib におけるそれらの扱いについては、83 ページ「4 章 色と透過」を参照してください。

SVG 文法は、すべての種類に対して、sRGB フォールバックカラーを必須としています。PDFlib は、文法エラーのときと、スポットカラー定義が見つからないときと、ICC プロファイルが見つからないとき、またはオプション `forcesrgb` が与えられているときに、このフォールバックカラーを使用します。

ブレンドモードを援用すると SVG 画像を着色または脱色することもできます(104 ページ「4.9.1 ブレンドモードを用いて色を変える」参照)。

ICC ベースのカラー `icc-based` 色指定は、指定された ICC プロファイルを適用します。これはローカルファイルとして参照されていない場合には、`ICCprofile` リソースカテゴリに従って検索されます。`honoriccprofile=false` が与えられている場合には ICC プロファイルは無視されます。SVG グラフィック内で指定されているプロファイルを他のプロファイルでオーバーライドするにはオプション `iccprofilegray/iccprofilergb/iccprofilecmyk` を使用します。

グレイ・RGB・CMYK デバイス色空間 デバイス独自色空間 `device-gray`・`device-rgb`・`device-cmyk` は、一般的用途においては推奨されません。なぜならそれらの視覚表現は、PDF/A または PDF/A 出力インテント ICC プロファイルか然るべきデフォルト色空間によって制御されるのでない限り、特定の出力デバイスに依存してしまうからです。かわりに ICC ベースの色空間を用いて作業することを強く推奨します。例外として挙げられるのは、CMYK コントロールストリップを使って印刷機における濃度値を直接制御する場合や、トンボを印刷する場合です。

`device-gray/rgb/cmyk` 属性が参照画像によって生成されたデバイス独自カラーについては、`defaultimageoptions` の `iccprofile` サブオプションか `templateoptions` の `defaultgray/rgb/cmyk` サブオプションを通じて、ICC ベースのカラーへマップすることができます。

スポットカラー・DeviceN カラー `icc-named-color` 構文を使うと、スポットカラーを指定できます。ただし、名前付きカラーの ICC プロファイルには対応していないので、そのスポットカラーは PDFlib が知っているものである必要があります。そうでないと、sRGB フォールバックカラーが代替色として使用されます。したがって、カスタムのスポットカラーについては、`PDF_load_graphics()` を呼び出す前に自分のコードの中で定義しておく必要があります (93 ページ「カスタムスポットカラー」参照)。例：

```
// スポットカラー「CompanyRed」をLab代替値とともに定義
p.set_graphics_option("fillcolor={spotname {CompanyRed} 1.0 {lab 60 65 65}}");
```


`device-nchannel` 構文を使うと、DeviceN カラーを選択できます。この DeviceN カラーは、`PDF_create_devicen()` を用いて作成されたうえで (94 ページ「4.5 DeviceN カラー」参照)、`PDF_load_graphics()` に `devicencolors` オプションを通じて与えられている必要があります。すなわち、使える DeviceN 色空間は *N* の値ごとに 1 個だけであることとなります。

シェーディング 表 8.4 に挙げる色空間は、SVG のグラデーションの中で使用されていることもありえます。ただし、PDF は、1 つのグラデーションの中ではすべてのストップカラーが同一の色空間から採られていることを要請しています。1 つのグラデーションの中で複数の色空間からのストップカラーを使っている場合には、すべての色に対してその sRGB フォールバックカラーが用いられます。

スポットカラーから作られるシェーディングは、スポットカラーが Lab 代替値とともに作成されていれば (PDFlib 内部で既知のスポットカラーはすべてこれに該当)、スポットカラーを温存します。CMYK など他の色空間で作成されたスポットカラーは温存されず、PDF 出力内で sRGB シェーディングとなります。

注 SVG における非 sRGB カラーは、まだ完全には標準化が済んでいませんので、その文法は将来変わる可能性もあります。

表 8.4 SVG 内の拡張色空間 (forcesrgb を用いて無効化させることもできます)

色空間	SVG 構文例	注
デバイス独立色空間		
sRGB	#FF0000	SVG 1.1 のデフォルトの色空間
ICC ベースカラー	#7F7F7F icc-color(gray.icc, 0.5) #CC6633 icc-color(rgb.icc, 0.8, 0.4, 0.2) #2B7FAB icc-color(cmyk.icc, 0.8, 0.4, 0.2, 0.0)	グレースケール・RGB・CMYK のプロファイルに対して、それぞれ 1 個・3 個・4 個のカラー値を与える必要があります。
CIE Lab	#598237 cielab(50, -25, 35)	
CIE Lch	#FF007E cielch(50, 127, 0) #FF007E cielchab(50, 127, 0)	明度・彩度・色相 (度単位で) CIE Lab へ変換されます。
周知のスポットカラー	#FFB12D icc-named-color(nmcl.icc, 'PANTONE 123 U') #FFCE4C icc-named-color(nmcl.icc, 'PANTONE 123 U', 0.5) #994F5B icc-named-color(nmcl.icc, 'HKS 18 Z')	プロファイル名は無視されます。SVG 文法への拡張として、PDFlib では、オプションな濃度値を使えます。濃度値がない場合は 1.0 と見なされます。
カスタムスポットカラー	#FFE560 icc-named-color(nmcl.icc, 'CompanyColor') #FFF36D icc-named-color(nmcl.icc, 'CompanyColor', 0.5)	そのスポットカラーがそれまでにすでに定義されていない限り、sRGB フォールバックカラーが代替色として使用されます。プロファイル名は無視されます。濃度値がない場合は 1.0 と見なされます。
n 色	#FFE560 device-nchannel(0, 0.7, 0.2) #FFF36D device-nchannel(1 0 0 0 0.5 0.2 0)	DeviceN 色空間ハンドルを <code>devicencolors</code> オプションで与える必要があります。n に対する DeviceN 色空間ハンドルが与えられていない場合には、sRGB フォールバックカラーが使用されます。
デバイス依存色空間		
DeviceGray	#7F7F7F device-gray(0.5)	PDF/A と PDF/X のデバイス色空間に対する要件に従う必要があります。

表 8.4 SVG 内の拡張色空間 (forcesrgb を用いて無効化させることもできます)

色空間	SVG 構文例	注
DeviceRGB	#0000FF device-rgb(0, 0, 255)	PDF/A と PDF/X のデバイス色空間に対する要件に従う必要があります。
DeviceCMYK	#00AEEF device-cmyk(1, 0, 0, 0)	PDF/A と PDF/X のデバイス色空間に対する要件に従う必要があります。

8.2.7 ベクトルグラフィック・テキスト以外の SVG 内容

SVG 内に埋め込まれた画像 PDFlib は、SVG 内の *image* エレメントを処理して、187 ページ「8.1 ラスタ画像」で解説したすべての画像形式と、ネストされた SVG グラフィックを受け付けます。画像データは、SVG ファイル内に埋め込まれていても、外部ファイル内に存在していても、ネットワークごとにリモートの場所から取得してもかまいません。

SVG グラフィック内の画像は、自動的に処理されます。ただし、場合によっては、特定の画像処理オプション群を与えるほうがよい場合もあります。これは *PDF_load_graphics()* の *defaultimageoptions* オプションで実現できます。たとえば、以下のオプションを用いると、低解像度画像の見栄えを向上させるアンチエイリアスを適用できます (これは、画像にアンチエイリアスを適用する多くのブラウザの SVG 表示と整合します) :

```
defaultimageoptions={interpolate}
```

画像が入手できない場合 (参照された外部画像ファイルが見つからないなどの原因で)、かつ、オプション *fallbackimage={}* (すなわち空のオプションリスト) が与えられている場合には、PDFlib は透明な灰色の市松模様を作成します。オプション *fallbackimage* の各種サブオプションを用いると、このパターンをカスタマイズしたり、カスタムの画像かテンプレートをフォールバック視覚効果として与えたりすることができます。見つからない画像が自動的に置換されることを避けたい場合には以下のオプションを用います :

```
errorconditions={references={image}}
```

SVG のリンク *inline* オプションを用いて取り込まれた SVG グラフィック内のリンクは、生成される PDF 出力内のインタラクティブなリンク注釈へ変換されます。ただし、リンクの作成を無効化する条件がいくつかあります (PDFlib API リファレンス参照)。PDF 注釈の *contents* オプションに、SVG リンクの *xlink:title* 属性がもしあればそれが、なければターゲット URI が記入されます。タグ付き PDF モードでは、生成されたリンクに対して、*Link* エレメント 1 個と、関連する *OBJR* エレメント 1 個が作成されます。ただし、カレントでアクティブなアイテムが直接ページ装飾か擬似エレメントである場合を除きます。

SVG リンクの PDF リンクへの変換は、*PDF_fit_graphics()* の *convertlinks* オプションで無効にすることもできます。

SVG 内のメタデータ SVG グラフィックは、XMP メタデータを含んでいることもあります。デフォルトでは PDFlib は、出力ファイルサイズを削減するために、SVG グラフィック内の XMP メタデータを無視します。SVG からテンプレートが作成される場合には (これがデフォルトの動作)、*PDF_load_graphics()* の以下のオプションで、その XMP メタデータを、生成されるフォーム XObject に添付することもできます :

```
templateoptions={metadata={keepxmp=true}}
```


SVG グラフィックの `metadata`・`desc`・`title` エレメントの内容は、`PDF_info_graphics()` で、以下のパターンに従って取得できます：

```
idx = (int) p.info_graphics(svg, "description", "");
if (idx != -1)
    description = p.get_string(idx, "");
```

8.2.8 対応していない SVG 機能

対応していない機能の扱い デフォルトでは、対応していない SVG 機能は無視されます。結果として、出力は作成されますが、そのグラフィックのいくつかの側面は失われているか、あるいは誤った状態となります。この動作は、`PDF_load_graphics()` の `errorconditions` オプションで変更できます。そのサブオプション群は、無視されるのではなくエラーを発生させる条件を指定します。たとえば、以下のオプションリストを用いると、`PDF_load_graphics()` は、SVG グラフィックがアニメーションエレメントまたはスクリプティングエレメントを含む場合には失敗します。例：

```
errorconditions = {elements={animate script}}
```

対応していない SVG エレメント 以下の SVG エレメントは、対応されておらず、無視されます：

- ▶ アニメーションとスクリプティングのためのエレメント群：

```
animate, animateColor, animateMotion, animateTransform, script, mpath, set
```

- ▶ フィルターのためのエレメント群：

```
feBlend, feColorMatrix, feComponentTransfer, feComposite, feConvolveMatrix,
feDiffuseLighting, feDisplacementMap, feDistantLight, feFlood, feFuncA, feFuncB,
feFuncG, feFuncR, feGaussianBlur, feImage, feMerge, feMergeNode, feMorphology,
feOffset, fePointLight, feSpecularLighting, feSpotLight, feTile, feTurbulence, filter
```

- ▶ その他のエレメント群：

```
cursor, foreignObject, vkern
```

制約のある SVG エレメント・属性・プロパティ 以下の属性とプロパティには制約があります：

- ▶ いくつかの CSS ディレクティブに対応していません。
- ▶ フォント選択プロパティ `font-variant` には、キーワード `small-caps` でのみ、かつ OpenType 機能 `smcp` を含むフォントに対してのみ対応しています。
- ▶ テキスト体裁プロパティ `text-decoration` の複数の値の組み合わせには対応していません。PDFlib は文字飾り要素を、別個の塗り・描線色を持つ領域として描くのではなく、文字飾り要素を直線として描きます。この直線は、塗り色があればそれで、なければ描線色で描かれます。
- ▶ `textPath` エレメントに対する `rotate` 属性には対応していません。
- ▶ プロパティ `unicode-bidi` には、双方向テキストレイアウトのために必要なテーブルを含む TrueType/OpenType フォントについてのみ従います。PDFlib は、`PDF_fit_textline()` のオプションリスト内で、オプション `shaping` と `script=_auto` を設定します。
- ▶ プロパティ `glyph-orientation-vertical` は SVG 1.1 に従って対応していますが、ただし角度 $180^\circ \cdot 270^\circ$ には対応していません。

- ▶ **view** エレメントに対する属性 *preserveAspectRatio* は無視されます。
- ▶ フォーム XObject (テンプレート) が生成されている場合、すなわちみ出した内容が切り落とされている場合には、**svg** エレメントのプロパティ *overflow* は値 *visible* に対応しません。これは *inline* オプションを用いて回避することもできます。

対応していない SVG プロパティ 以下の SVG プロパティは、対応されておらず、無視されます：

alignment-baseline, color-interpolation, color-interpolation-filters,
color-rendering, cursor, dominant-baseline, enable-background, filter, flood-color,
flood-opacity, glyph-orientation-horizontal,
image-rendering, lighting-color, pointer-events, shape-rendering, text-rendering

対応している SVG エレメントの対応していない属性 対応している SVG エレメントの、以下の属性は、対応されておらず、無視されます：

baseProfile (svg)
contentScriptType (svg)
contentStyleType (svg)
externalResourcesRequired (すべてのエレメント)
method (textPath)
on* (すべてのエレメント)
spacing (textPath)
縦書きテキストに対するtextLength・lengthAdjust (text・textpath・tref・tspan)
version (svg)
zoomAndPan (svg)
xlink:role (すべてのエレメント)
xlink:show (すべてのエレメント)
xlink:type (すべてのエレメント)

8.3 PDF ページを PDI で取り込む

注 この節で解説するすべての関数は、PDFlib+PDI か、または PDFlib Personalization Server PPS（これは PDI を含んでいます）を必要とします。PDF 取り込みライブラリー（PDI）は PDFlib 基本製品には含まれていません。PDI は PDFlib のすべてのバイナリー版に含まれていますが、それを利用するには PDFlib+PDI か PPS のためのライセンスキーが必要です。

8.3.1 PDI の機能と用途

PDI（PDF 取り込みライブラリー）が得られる場合には、既存 PDF 文書内のページを取り込むことができます。PDI は、既存 PDF 文書内のページを PDFlib で利用できるようにします。概念的に、取り込まれた PDF ページは、取り込まれたラスター画像と同様に扱われます：PDF 文書を開き、取り込むページを選び、それを出力ページ上に配置します。取り込んだページに PDFlib の変形関数を適用して並行移動・拡大・回転・斜形化させることもできます。取り込んだページは新しい内容と組み合わせることができます。そのためには、取り込んだ PDF ページを出力ページ上に配置した後に PDFlib のテキスト・グラフィック関数を使えばよいのです（取り込んだページは新しい内容の背景となると捉えられます）。PDFlib と PDI を活用すれば以下のような課題が簡単に実現できます：

- ▶ 複数の PDF 文書内の複数のページを重ね合わせ（たとえば、既存文書に便箋を追加して印刷済み用紙のようにする）。
- ▶ 既存文書内に PDF 広告を配置。
- ▶ PDF のページの表示領域を切り抜いて、見せたくない要素（トンボなど）を取り除く。または、ページの拡大縮小。
- ▶ 複数ページを 1 枚の紙に印刷。
- ▶ 複数の PDF/X か PDF/A の文書を処理して、新しい PDF/X か PDF/A のファイルを作成する。
- ▶ ファイルの PDF/X か PDF/A の出力インテントをコピー。
- ▶ 既存 PDF のページにテキスト（ヘッダー・フッター・スタンプ・ページ番号など）や画像（企業ロゴなど）を追加。
- ▶ 入力文書内の全ページを出力文書にコピーして、各ページにバーコードを配置。
- ▶ pCOS インターフェイスを使って、PDF 文書の任意のプロパティをクエリー（詳しくは pCOS パスリファレンスを参照）。

PDF の背景ページを配置してそこに動的なデータを入れ込みたい場合には（たとえばメールのマーჯや、Web 上のパーソナライズされた PDF 文書や、フォーム記入など）、PDI を PDFlib ブロックとあわせて利用されることをおすすめします（375 ページ「13 章 PPS と PDFlib Block Plugin」参照）。

8.3.2 PDFlib+PDI を使用

クックブック PDF 取り込みの諸側面に関するコードサンプルが PDFlib クックブックの pdf_import カテゴリにあります。

一般的に考慮すること PDF 文書からはページ内容だけでなく以下のアイテムも取り込めます：

- ▶ 注釈（213 ページ「注釈を取り込む」を参照）
- ▶ フォームフィールド（214 ページ「フォームフィールドを取り込む」を参照）
- ▶ アクション（214 ページ「アクションと JavaScript を取り込む」を参照）

- ▶ 構造エレメントタグ (詳しくは 215 ページ「タグ付き PDF 文書の構造情報を取り込む」を参照)
- ▶ レイヤー定義 (215 ページ「レイヤーを持つページを取り込む」を参照)
- ▶ (PPSのみ)PDFlibブロックを、`PDF_process_pdi()` とオプション `action=copyallblocks` または `copyblock` で取り込めます (418 ページ「13.9.2 PDFlib ブロックを取り込む」参照)。

取り込んだページ内の個々のリソースを他のPDFlib関数で再利用することはできません。たとえば、取り込んだ文書内のフォントを他の何らかの内容のために再利用することは不可能です。必要なフォントはすべてPDFlib内で構成する必要があります。取り込んだ複数の文書が同じフォントの埋め込みフォントデータをそれぞれ持っていたとしても、フォントデータの重複をPDIはデフォルトでは解消させませんが、これは `optimize` 文書オプションを用いて実現することもできます。他方、取り込んだPDF内で欠けているフォントがあれば、生成されるPDF出力ファイル内でもそのフォントは欠けたままです。最適な方法としては、取り込む文書はなるべく開いたままにしておいたほうが、同じフォントが何度も出力文書内に埋め込まれずに済みます。

取り込んだPDFのページを出力ページ上に配置するためにPDFlib+PDIはテンプレート機能(フォームXObject)を利用します。

PDF ページを取り込むためのコード断片 既存PDF文書内のページの取り込みは非常に単純なコード構造で実現可能です。以下のコードスニペットは、既存文書のページを開き、そのページ内容を出力PDF文書内にコピーします(出力PDF文書はあらかじめ開いている必要があります)：

```
int doc, page, pagecount, pageno = 1;
String filename = "input.pdf";

if (p.begin_document(outfilename, "") == -1) {...}
...

doc = p.open_pdi_document(infile, "");
if (doc == -1)
    throw new Exception("エラー：" + p.get_errmsg());

/* 入力文書内のページ数。全ページを取り込むのに有用 */
pagecount = (int) p.pcos_get_number(doc, "length:pages");

page = p.open_pdi_page(doc, pageno, "");
if (page == -1)
    throw new Exception("エラー：" + p.get_errmsg());

/* ダミーのページ寸法。この後adjustpageオプションによって変更される */
p.begin_page_ext(20, 20, "");
p.fit_pdi_page(page, 0, 0, "adjustpage");
p.close_pdi_page(page);
...ページに内容を追加するPDFlib関数群...
p.end_page_ext("");
p.close_pdi_document(doc);
```

`PDF_fit_pdi_page()` の最後の引数は、取り込むページの位置付け・拡大縮小・回転を指示するさまざまなオプションを持ちうるオプションリストです。このオプションについては詳しくは 221 ページ「8.4 画像・グラフィック・取り込み PDF ページを配置」で解説しています。

8.3.3 文書・ページ関連のチェック

文書関連のチェック PDFlib+PDI は、すべての種類の PDF 文書を、PDF バージョン番号や、そのファイル内で使用されている機能にかかわらず、処理します。PDFlib+PDI は、ある種の破損した文書でも開くことができるよう、破損 PDF のための修復モードを実装しています。ただし、まれに、PDF 文書が、あるいは文書の特定のページが、PDI によって拒絶されることもあります。

PDF 文書またはページがうまく取り込めないときは、`PDF_open_pdi_document()` と `PDF_open_pdi_page()` はエラーコードを返します。失敗に関してもっと詳しく知る必要がある場合には、その理由を `PDF_get_errmsg()` でクエリーすることができます。あるいは、`errorpolicy` オプションを `exception` に設定することによって、文書を開くことができなかつたときには例外が発生するようにすることもできます。

ページ関連のチェック `PDF_open_pdi_page()` 内で以下のチェックが行われます：

- ▶ 生成されつつある PDF 出力文書よりも高い PDF バージョン番号を用いている PDF 文書の中のページは、取り込めません。なぜなら、より高いバージョン番号を持つ PDF が取り込めたとすると、要求された PDF バージョンに出力が準拠しているかどうか PDFlib はもはや確信が持てないことになるからです。解決策：出力 PDF のバージョンを、`PDF_begin_document()` 内の `compatibility` オプションを用いて、必要なレベルに設定します。`PDF 1.7ext 3` と `PDF 1.7ext 8` の文書は、PDI に関する限り、PDF 1.7 と互換です。PDF/A モードでは、入力 PDF バージョン番号は意味を持ちません。なぜなら PDF バージョンヘッダーは PDF/A では無視される必要があるからです。文書が、より古い PDF バージョンに準拠していることがわかっているにもかかわらず、より高い PDF バージョンヘッダーを使用している場合には、`PDF_open_pdi_document()` の `ignorepdfversion` オプションを用いることができます。
- ▶ PDF/A ・ PDF/X ・ PDF/VT ・ PDF/UA の文書が、その対応する、カレント出力文書の PDF/A ・ PDF/X ・ PDF/VT ・ PDF/UA ステータスと非互換の場合には、拒否されます。詳しくは以下の項を参照してください：
 - ▶ 354 ページ「12.2.7 PDF/A 文書を PDI で取り込み」。
 - ▶ 365 ページ「12.3.6 PDF/X 文書を PDI で取り込む」。
 - ▶ 374 ページ「12.4.7 PDF/X ・ PDF/VT 文書を PDI で取り込む」。
 - ▶ 340 ページ「11.4.4 PDF/UA 文書のページを PDI で取り込む」。
- ▶ 文書が、矛盾した PDF/A または PDF/X 出力インテントを含んでいる場合には、ページは一切取り込まれません。

8.3.4 注釈・フォームフィールド・アクションを取り込む

注釈を取り込む デフォルトでは PDI は、取り込まれるページに存在しているページ内容とすべての注釈を取り込みます。`PDF_fit_pdi_page()` の `useannots` オプションを与えると、リンク注釈など選択した種類の注釈だけを取り込めます。注釈に紐付いているアクションも取り込まれます。文書内のリンクの移動先は、ページの新しい配置に合わせたものになります。たとえば、取り込んだページの順序を変えても文書内のリンクはなお動作します。リンクの移動先は、そのターゲットページが拡張か回転されている場合などには再計算されます。しかし、生成する出力文書の内容と構造によっては、インタラクティブ要素を再現できないこともあります。たとえば、取り込んだページに次ページへのリンクがあったとして、なのにそのターゲットページを取り込まなかったら、そのリンクはどこを指せばよいのでしょうか？ 一般に PDFlib は、できるだけ多くのインタラクティブ要素が

新たな文書でも機能するように試みますが、あらゆる場合にそれを保証できるものではないわけです。

注釈の取り込みはページスコープでのみ動作し（たとえば PDF ページがテンプレートに配置される際には動作しない）、以下の制約が課されます：

- ▶ リンクのターゲットが得られない場合（たとえばターゲットページが取り込まれていないせいで）には、何のアクションもないリンクが生成されます。
- ▶ 名前付き移動先は解決されますが、名前は温存されません。ですので、リンクはターゲットページへ飛びますが、移動先のシンボリック名は失われます。
- ▶ **PrinterMark**・**Watermark** 注釈の取り込みには対応していません。
- ▶ 注釈辞書内の、動画起動キー **Start**・**Duration**・**Rate**・**Synchronous** などいくつかのプロパティは取り込まれません。
- ▶ PDF 2.0 の構造移動先の取り込みには対応していません。

注釈種別 **Caret**・**FreeText**・**Line**・**Stamp** のいずれかを内容として持っているページを取り込む場合には、その注釈が体裁ストリームを持っていないならば、Noto フォント群へのアクセスを構成する必要があります。フォントは 2 つとも PDFlib ディストリビューションに含まれています（19 ページ「1.3 構成の必要事項」を参照）。

フォームフィールドを取り込む デフォルトでは PDI はページ内容のみを取り込み、フォームフィールドを無視します。`PDF_fit_pdi_page()` の `usefields` オプションを与えると、すべてのフィールドを取り込んだり、テキストフィールドなど選択した種類のみを取り込んだりすることができます。フォームフィールドは、ただ 1 つの PDF 入力文書からのみ取り込むことができ、複数の文書からは取り込めません。取り込んだページをフォームフィールドとともに配置できるのは 1 回だけです。また、フォームフィールドの作成は、ただ 1 つの PDF 文書から取り込むか、それとも `PDF_create_field()` を用いるか、そのどちらかのみで行えます。両方の方式を組み合わせることはできません。しかし、取り込んだフォームフィールドに `PDF_create_field()` で流し込みを行うことは可能です。

フォームフィールドの取り込みはページスコープでのみ動作し（たとえば PDF ページがテンプレートに配置される際には動作しない）、以下の制約が課されます：

- ▶ フォームフィールドの長方形の位置と寸法は、ページ上の配置に合わせて拡張などしたものにになります。
- ▶ 署名フィールドの暗号署名データは消去されますので、この場合には、視覚表現も再生成されます。不可視の署名は無視されます。

アクションと JavaScript を取り込む PDF 文書は、さまざまなトリガーイベントによって起動されるアクションを内容として持っていることがあります。アクションの取り込みについてまとめると以下のとおりです：

- ▶ 注釈とフォームフィールドに位置しているアクションは、自動的にその注釈かフォームフィールドと一緒に取り込まれます。
- ▶ 以下の種類のアクションの取り込みには対応していません：
GoToE・**GoToDp**・**Thread**・**Sound**・**Trans**。
- ▶ 文書アクションは、`PDF_open_pdi_document()` の `useactions` オプションを用いると取り込めます。
- ▶ ページアクションは、`PDF_fit_pdi_page()` の `useactions` オプションを用いると取り込めます。
- ▶ 注釈・フォームフィールド・ページ・文書のための JavaScript アクションは、文書レベルに格納されている JavaScript ルーチンに依存していることがあります。これらのスクリプト

リプトは、`PDF_open_pdi_document()` の `usejavascript` オプションを用いて取り込めます。文書レベルの JavaScript に依存しているフィールドを取り込む場合にはこのオプションを `true` に設定することを推奨します。文書レベルの JavaScript は、`PDF_create_action()` の `script · scriptname` オプションを用いて作成することも可能です。

8.3.5 取り込んだ PDF 文書のさらなる特徴

取り込んだ PDF ページの寸法 取り込んだ PDF ページは取り込んだラスター画像と同様に扱われ、`PDF_fit_pdi_page()` を用いて出力ページ上に配置することができます。デフォルトでは、Acrobat での表示とまったく同じ形で PDI はページを取り込みます。とりわけ次のような動作をします：

- ▶ クロッピングは保持されます（技術的にいえば：CropBox が存在する場合には、PDI は MediaBox よりも CropBox を優先採用します。70 ページ「3.2.2 ページ寸法」参照）。
- ▶ ページに適用されている回転は保持されます。

`cloneboxes` オプションは PDFlib+PDI に対して、取り込みページのすべてのページ枠を生成出力ページへコピーするよう指示し、その結果、すべてのページ寸法情報が転写されます。

あるいは、`pdiusebox` オプションを用いて明示的に、ページの MediaBox · CropBox · TrimBox · ArtBox エントリーのうちのいずれかを（もしあれば）用いて取り込みページの寸法を決めるよう PDI に指示することもできます。

そして、`PDF_open_pdi_page()` の `transform` オプションを用いて、取り込みページに対して、拡大 · 縮小や回転など、任意の変換を適用することもできます。

色の扱い PDFlib+PDI は、取り込んだ PDF 文書の色を一切変更しません。たとえば、PDF が ICC カラープロファイルを含んでいる場合、これは出力文書内に保持されます。ページが透過グループエントリーを含んでいる場合、それは、生成されるフォーム XObject へ複製されます。ただし、`PDF_open_pdi_page()` の `transparencygroup` オプションが別の扱いを要請している場合を除きます。デフォルトでは、取り込んだページに然るべき透過グループがなければ生成されます。その際には、PDF/A · PDF/X · PDF/VT のあてはまる要請がすべて考慮されます。

タグ付き PDF 文書の構造情報を取り込む デフォルトでは、タグは、入力文書と出力文書の両方がタグ付けされていれば取り込まれます。ただし、タグ取り込みは、`PDF_open_pdi_document()` と `PDF_open_pdi_page()` の `usetags` オプションで無効にすることもできます。詳しくは 330 ページ「11.3 タグ付き PDF ページを PDI で取り込む」を参照してください。

レイヤーを持つページを取り込む PDFlib はつねに、ページ上のすべてのレイヤー（技術的には「オプション内容」といいます）の内容を取り込みます。レイヤーの可視状態を含むレイヤー定義も、そのレイヤーが、取り込んだページ群のうちのいずれかで使用されていれば、取り込まれます。ただし、レイヤー定義の取り込みは、`PDF_open_pdi_document()` の `uselayers` オプションで無効にすることもできます。`uselayers=false` で作業をするためには、生成される文書はレイヤーを一切含んではいけません。すなわち、レイヤーを持った PDF 文書はすべて `uselayers=false` を用いて開く必要があり、かつ、`PDF_define_layer()` を呼び出してはいけません。

取り込んだレイヤー群の整頓をさらに制御するには、取り込んだレイヤー群に付加されるレイヤーリストの中に階層構造的なタイトルレイヤーを作成する `PDF_open_pdi_`

`document()` の `parenttitle` を用いることもできます(たとえばファイル名を与えるために)。`parentlayer` オプションは、同様に動作しますが、ただしユーザー定義レイヤーのハンドルをとりません。

地理参照付き PDF を取り込む 地理参照付き PDF を PDI で取り込む際には、地理空間情報は、それが以下のいずれかの方式で作成されていれば保持されます(画像ベースの地理空間参照)：

- ▶ PDFlib で `PDF_load_image()` の `georeference` オプションで
- ▶ Acrobat で地理空間情報を持つ画像を取り込んで。

地理空間情報はページを取り込んだ後、それが以下のいずれかの方式で作成されていた場合には失われます(ページベースの地理空間参照)：

- ▶ PDFlib で `PDF_begin/end_page_ext()` の `viewports` オプションで
- ▶ Acrobat で手作業で PDF ページを地理登録して。

複数の取り込み文書をまたぐ最適化 PDFlib 自体は、高度に最適化した PDF 出力を作成しますが、取り込んだ PDF にはもしかすると冗長なデータ構造があつて、場合によっては最適化の余地があるかもしれません。さらに、取り込む PDF がもし複数であれば、その複数のファイルが等価なリソース(ファイルなど)を含む場合には、出力するファイルのサイズはふくれあがる可能性があります。このような場面では、`PDF_begin_document()` の `optimize` オプションを使うことができます。これは取り込んだファイル内の冗長なオブジェクトを検知して、生成する出力の体裁や品質をそこなうことなくそうしたオブジェクトを削除します。

暗号化された PDF 文書と「シュラッグ」機能 暗号化文書内(すなわち、権限設定またはパスワードを持ったファイル)のページを取り込むためには、その対応するマスターパスワードを与える必要があります。暗号化された PDF 文書でパスワードがないものは、デフォルトでは拒絶されます。ただしこれは、`PDF_open_pdi_document()` の `infomode` オプションを `true` に設定することにより、pCOS で情報をクエリーするために(ページを取り込むためではなく)開くことはできます。(この `infomode` 規則に対する例外：Distiller の設定「オブジェクトレベルの圧縮：最高」を用いて作成された文書は、情報モードでも開くことができません)。

「シュラッグ」機能を利用すると、保護された文書の中のページを、その文書の作成者の権利を尊重する責任をユーザーが受け入れることを前提として、マスターパスワードなしで取り込むことができます。「シュラッグ」機能を利用することにより、ユーザーは、彼または彼女がいかなる文書作成者の権利をも侵害していないことを宣明します。PDFlib GmbH の契約条件は、ユーザーが文書作成者の権利を尊重することを求めています。

以下のすべての条件が真であるときに、「シュラッグ」機能は有効になります：

- ▶ `PDF_open_pdi_document()` に `shrug` オプションが与えられている。
- ▶ その文書はマスターパスワードを必要としているが、それが `PDF_open_pdi_document()` に与えられていない。
- ▶ その文書がユーザー(開く)パスワードを必要としている場合には、それが `PDF_open_pdi_document()` に与えられている必要があります。

「シュラッグ」機能には以下の効果があります：

- ▶ マスターパスワードが与えられていなくてもページを取り込むことができる。
- ▶ pCOS 擬似オブジェクト `shrug` が `true/1` に設定される。

- ▶ pCOS が完全モードで動作する (限定モードではなく)。すなわち、`pcosmodename` 擬似オブジェクトが値 `full` に設定される。

8.3.6 文書関連のプロパティを転写

PDF 文書は多様なプロパティを持っており、ページ内容以外にも多くの機能を含んでいることがあります。PDFlib+PDI を用いて PDF 文書を取り込む際には、取り込んだ文書の特徴を、生成する出力文書において再現するかを、詳細に制御できます。ある特徴を転写するようシンプルなオプションで PDFlib+PDI に命令できる場合もありますし、そうでない場合には、pCOS インターフェースを使って入力文書から何らかのデータを取得したのちにこのデータを、対応する PDFlib の API メソッドとオプションに与えることができます。

表 8.5 取り込んだ PDF 文書の文書関連の特徴を転写

PDF のアイテム	pCOS パス	PDFlib を用いて転写または作成
以下のプロパティは、オプションを用いて転写できます：		
文書の出カインテント	/Root/OutputIntents	PDF_process_pdi() : オプション action=copyoutputintent
文書レベルのアクション	destpage・/Root/OpenAction・ /Root/AA	PDF_open_pdi_document() : useactions オプション
JavaScript 名称ツリー	names/JavaScript	PDF_open_pdi_document() : usejavascript オプション
名前付き移動先	names/Dests	注釈フィールドのアクションによって使用されている名前付き移動先は解決されますが、その名前は取り込まれません。 PDF_add_nameddest()
ベース URI	/Root/URI	ベース URL は、取り込まれたリンクアクションに自動的に適用されます。 PDF_begin/end_document() : uri オプション
以下のプロパティは、pCOS を用いてオブジェクトを取得してそれを PDFlib に与えることによって転写できません：		
文書の添付	names/EmbeddedFiles	PDF_begin/end_document() : attachments オプション
タグ付き PDF のステータス	tagged	PDF_begin_document() : tagged オプション
文書が開かれた際のページレイアウト	/Root/PageLayout	PDF_begin/end_document() : pagelayout オプション
文書を開くモード	/Root/PageMode	PDF_begin/end_document() : openmode オプション
ビューアの実環境設定	/Root/ViewerPreferences	PDF_begin/end_document() : viewerpreferences オプション
文書の言語	/Root/Lang	PDF_end_document() : lang オプションか lang タグ付けオプション

表 8.5 取り込んだ PDF 文書の文書関連の特徴を転写

PDF のアイテム	pCOS パス	PDFlib を用いて転写または作成
暗号化	encrypt/algorithm encrypt/master encrypt/user encrypt/attachment encrypt/plainmetadata encrypt/noaccessible など	PDF_begin_document() : attachmentpassword・masterpassword・ permissions・userpassword オプション
文書レベル XMP	/Root/Metadata	PDF_begin/end_document() : metadata オプション
文書情報フィールド	/Info	PDF_set_info()
線形化	linearized	PDF_begin_document() : linearize オプション
PDF/A・PDF/UA・PDF/X・PDF/VT 規格のステータス	pdfa・pdfua・pdfx・pdfvt	PDF_begin_document() : pdfa・pdfua・pdfx・pdfvt オプション
PDF のバージョン	pdfversionstring	PDF_begin_document() : compatibility オプション
文書レベルの連携ファイル	/Root/AF	PDF_end_document() : associatedfiles オプション
構造ヒエラルキー上の連携ファイル	/Root/StructTreeRoot/AF	PDF_end_document() : structureassociatedfiles オプション
以下のプロパティは簡単に転写できません :		
しおり	bookmarks[...]	PDF_create_bookmark()
DPart と文書部分メタデータ (DPM)	/Root/DPartRoot	PDF_begin/end_dpart() : dpm オプション
ポートフォリオ	/Root/Collection	PDF_begin/end_document() : portfolio オプション、 PDF_add_portfolio_file()、 PDF_add_portfolio_folder()
電子署名とタイムスタンプ	signaturefields[...]	署名を転写しても無効になるので意味がありません。新たな署名とタイムスタンプは、PDFlib PLOP DS を用いて作成できます。
Reader で編集ツールを有効化するための使用権限	usagerights	使用権限は、会社独自の電子署名があるので、Adobe ツールを用いてのみ生成できます。
タグ付き PDF に対するクラスマップ	/Root/StructTreeRoot/ClassMap	非対応
アーティクルのスレッドとビード	articles	非対応
文書レベルの PieceInfo	/Root/PieceInfo	非対応

8.3.7 ページ関連のプロパティを転写

表 8.6 取り込んだ PDF 文書のページ関連の特徴を転写

PDF のアイテム	pCOS パス	PDFlib を用いて転写または作成
以下のプロパティは、オプションを用いて転写できます：		
ページ内容	pages[...]	PDF_fit_pdi_page()
ページ寸法	pages[...]/MediaBox と pages[...]/CropBox	PDF_fit_pdi_page() : adjustpage オプション
ページの幾何情報 (枠と回転)	pages[...]/MediaBox など pages[...]/Rotate	PDF_open_pdi_page() ・ PDF_fit_pdi_page() : cloneboxes オプション
注釈	pages[...]/annots	PDF_fit_pdi_page() : useannots オプション
フォームフィールド	pages[...]/fields	PDF_fit_pdi_page() : usefields オプション
ページを開くアクション、 閉じるアクション	pages[...]/AA/O と pages[...]/AA/C	PDF_fit_pdi_page() : useactions オプション
PDFlib Personalization Server (PPS) のためのブ ロック	pages[...]/blocks	PDF_process_pdi() : オプション action=copyblock ・ action=copyallblocks
レイヤー	/Root/OCProperties/OCGs	PDF_open_pdi_document() : uselayers オプション
ページ内容に対数する構造工 元素タグ	PDF_info_pdi_page() : キーワード topleveltag	PDF_open_pdi_document() ・ PDF_open_pdi_page() : usetags オプション
ページの出カインテント	pages[...]/OutputIntents	PDF_process_pdi() : オプション action=copypageoutputintent
以下のプロパティは、pCOS を用いてオブジェクトを取得してそれを PDFlib に与えることによって転写できま す：		
ページレベル XMP	pages[...]/Metadata	PDF_begin/end_page_ext() : metadata オプション
注釈とフィー無フィールド のタブ順序	pages[...]/Tabs	PDF_begin/end_page_ext() : taborder オプション
ユーザー単位	pages[...]/UserUnit	PDF_begin/end_page_ext() : userunit オプション
ビューポート	pages[...]/VP	PDF_begin/end_page_ext() : viewports オプション
ページ遷移	pages[...]/Trans	PDF_begin/end_page_ext() : duration ・ transition オプション
ページレベルの連携ファイ ル	pages[...]/AF	PDF_begin/end_page_ext() : associatedfiles オプション
画像レベルの連携ファイル	pages[...]/images[...]/AF	PDF_load_image() : associatedfiles オプション
フォーム XObject レベルの 連携ファイル	pages[...]/templates[...]/AF	PDF_load_graphics() ・ PDF_open_pdi_page() ・ PDF_begin_template_ext() : associatedfiles オプション
注釈 XObject レベルの連携 ファイル	pages[...]/annots[...]/AF	PDF_create_annotation() : associatedfiles オプション
ページラベル	pages[...]/label	PDF_begin_page_ext() : label オプション

表 8.6 取り込んだ PDF 文書のページ関連の特徴を転写

PDF のアイテム	pCOS パス	PDFlib を用いて転写または作成
以下のプロパティは転写できません：		
ページレベルの PieceInfo	pages[...]/PieceInfo	非対応
プレゼンテーションステップ	pages[...]/PresSteps	非対応

8.4 画像・グラフィック・取り込み PDF ページを配置

ラスター画像とテンプレートを配置するための関数 `PDF_fit_image()` と、グラフィックを配置するための関数 `PDF_fit_graphics()` と、取り込み PDF ページを配置するための関数 `PDF_fit_pdi_page()` は、ページ上への配置を制御するためのさまざまなオプションを提供しています。この節では、いくつかの代表的な応用作業をしてみることで、もっとも重要ないくつかのオプションの動作を示します。すべてのオプションの完全な一覧と説明については、[PDFlib API リファレンス](#)を参照してください。

この節に載せる作成例はすべて、ラスター画像でもテンプレートでもグラフィックでも取り込み PDF ページでも、同じく動作します。コードの作成例はラスター画像のためのものしか載せませんが、オブジェクト一般の配置方法をここでは語っているのです。あらゆる `fit` 関数はすべて、それを呼び出す前にはかならず、`PDF_load_image()` か `PDF_load_graphics()` か `PDF_open_pdi_document()` と `PDF_open_pdi_page()` への呼び出しを行う必要があります。簡潔のため、これらの呼び出しはここでは繰り返しません。

クックブック 画像・グラフィック・取り込み PDF ページに関するコードサンプルが `PDFlib` クックブックの `images・graphics・pdf_import` カテゴリにあります。

8.4.1 単純にオブジェクトを配置

画像を参照点に位置付け デフォルトでは、オブジェクトはその元の寸法で、左下隅を参照点に配置されます。この例では、画像の下端中央を参照点(`o, o`)に配置してみましょう：

```
p.fit_image(image, 0, 0, "position={center bottom}");
```

同様に、`position` オプションをキーワード `left・right・center・top・bottom` から別の組み合わせで使うことによって、オブジェクトそれぞれ左端・右端・中央・上端・下端を参照点に配置することができます。

画像を拡張して配置 コードを以下のように変えると、画像を配置する際にその大きさも変更できます：

```
p.fit_image(image, 0, 0, "scale=0.5");
```

このコード断片は、オブジェクトの左下隅をユーザー座標系の点 (`o, o`) に配置します。と同時に、オブジェクトは `x・y` 方向に拡張倍率 0.5 で拡張されるので、元の 50 パーセントの大きさになります。

クックブック 完全なコードサンプルがクックブックの `images/starter_image` トピックにあります。

8.4.2 オブジェクトを点上か線上か枠内に配置

オブジェクトを位置付けるために、あらかじめ定義した幅と高さの枠をあわせて使うこともできます。以下の図内の灰色の枠と線は、枠の大きさを視覚化するために描き足してあるだけで、実際の出力にはありません。

オブジェクトを枠内に配置することは、`fitmethod=nofit` の場合には意味がありません。なぜならこの場合、そのオブジェクトは位置付けられるだけであり、拡張は行われなからです。`boxsize` オプションを用いて、オブジェクト配置のための横線か縦線、あるいは実際の枠を指定することができます：

boxsize={100 0}	横線
boxsize={0 100}	縦線
boxsize={100 200}	枠

以下のさまざまな例では、オブジェクトを枠内に、さまざまなはめ込み方式ではめ込んでいきましょう。

枠内へ自動はめ込み `fitmethod=auto` を用いると、PDFlib は画像を、枠内にはめ込めるように、ただしそれをつぶさずに拡張します：もしそれがその枠内にはめ込めた場合には、拡張は一切行われません。そうでない場合には、幅と高さの縦横比を保ちながら、寸法が縮小されます。

図 8.2a・図 8.2b・図 8.2c では、はめ込み枠の寸法が最初 `boxsize={70 45}`、次に `boxsize={70 30}`、さらに `boxsize={30 30}` と減少していくにつれて、PDFlib が画像寸法を縮小していくさまを演示しています。

図 8.2 さまざまなはめ込み方式に従って画像を枠内にはめ込む

生成される出力	PDF_fit_image に与えるオプションリスト
a) 	<code>boxsize={70 45} position=center fitmethod=auto</code> (拡張は必要でない)
b) 	<code>boxsize={70 30} position=center fitmethod=auto</code> (枠の高さが小さくなったのに合わせて縮小される)
c) 	<code>boxsize={30 30} position=center fitmethod=auto</code> (枠の高さと幅が小さくなったのに合わせて縮小される)
d) 	<code>boxsize={70 45} position=center fitmethod=meet</code>
e) 	<code>boxsize={35 45} position=center fitmethod=meet</code>
f) 	<code>boxsize={70 45} position=center fitmethod=entire</code>
g) 	<code>boxsize={30 30} position=center fitmethod=clip</code>
h) 	<code>boxsize={30 30} position={right top} fitmethod=clip</code>

画像を枠の中央へはめ込み あらかじめ定義した長方形の中央に画像を置きたいときは、自分で計算をする必要は全くなく、適切なオプションを使えば実現できます。`position=center` を使って、幅 70・高さ 45 単位の枠 (`boxsize={70 45}`) の中央に画像を配置しましょう。`fitmethod=meet` を使うと、画像は縦横比を保ちつつ、その上下が枠に収まりきるまで拡大縮小されます (図 8.2d 参照)。

枠の幅を 70 から 35 単位の縮めると、画像はその左右が枠に収まりきるまで縮小されます (図 8.2e 参照)。

`fitmethod=meet` では、画像がつぶされないことが保証されるとともに、枠内に、できるだけ大きく配置されます。

画像を枠全体へはめ込み 画像をもっと枠に合わせて、枠全体を画像が埋めるようにすることもできます。これは `fitmethod=entire` で実現できます。しかし、この組み合わせは画像をつぶすおそれがあるので、有用な場合はまれでしょう (図 8.2f 参照)。

画像を枠へはめ込む際に切り抜き 別のはめ込み方式 (`fitmethod=clip`) を使うと、オブジェクトがはめ込んだ枠からはみ出したときに、そのオブジェクトを切り抜くことができます。枠の大きさを縦横とも 30 単位の縮めて、画像をその枠の中央に元の大きさのまま位置付けてみましょう (図 8.2g 参照)。

画像を枠の中央に位置付けることによって、画像はすべての端が均等に切り落とされます。同様に、画像の右上部分をすべて見せたいならば、`position={right top}` で位置付ければよいでしょう (図 8.2h 参照)。

8.4.3 オブジェクトの向きを変える

画像の向きを変えて配置 今度の例としては、画像の向きを西向きに変えてみましょう (`orientate=west`)。これはすなわち、画像が 90° 反時計回りに回転され、その回転後のオブジェクトの左下隅が参照点 (o, o) へ並行移動されることを意味します。画像はその場で回転します (図 8.5a 参照)。はめ込み方を指定していないので、画像は元の大きさのまま出力されて、枠からはみ出します。

画像の向きを変えて縦横比を保ちつつ枠ちょうどへはめ込み 今度は、画像を西に向けたうえで、あらかじめ定義した大きさにすることに挑戦してみましょう。求める大きさの枠を定義して、画像の縦横比を変えずにその枠にはめ込みます (`fitmethod=meet`)。向きは `orientate=west` と指定します。デフォルトでは、画像は枠の左下隅に配置されます (図 8.5b 参照)。東に向けた画像を図 8.5c に、南向きを図 8.5d に示します。

`orientate` オプションは、図 8.4 に示すとおり、向きのキーワードとして `north`・`east`・`west`・`south` に対応しています。

なお、`orientate` オプションは、座標系全体には一切影響せずに、配置するオブジェクトにだけ影響を及ぼします。

図 8.5 画像の向きを変える


生成される出力	PDF_fit_image に与えるオプションリスト
a) 	<code>boxsize={70 45} orientate=west¹</code>

図 8.3
rotate オプション

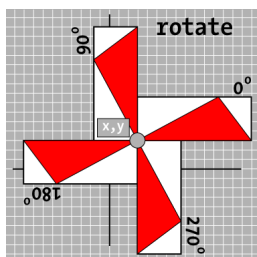
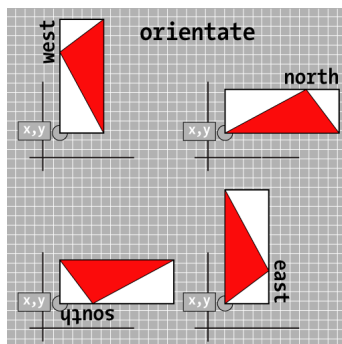
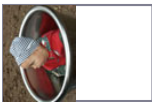





図 8.4
orientate オプション



生成される出力	PDF_fit_image に与えるオプションリスト
b) 	<code>boxsize={70 45} orientate=west fitmethod=meet</code>
c) 	<code>boxsize={70 45} orientate=east fitmethod=meet</code>
d) 	<code>boxsize={70 45} orientate=south fitmethod=meet</code>
e) 	<code>boxsize={70 45} position={center bottom} orientate=east fitmethod=clip</code>

1. デフォルト `fitmethod=nofit` ですので、この `boxsize` オプションは実際には必要ありません。

向きを変えた画像を枠へはめ込んで切り抜き 画像を東に向けて (`orientate=east`)、枠の下端中央に位置付けましょう (`position={center bottom}`)。さらに、画像を元の大きさのまま配置し、もし画像が枠からはみ出したら切り落とします (`fitmethod=clip`) (図 8.5e 参照)。

8.4.4 オブジェクトを回転

`rotate` オプションは、参照点を中心に座標系を回転させることによって、オブジェクトを回転させます。結果として、はめ込み枠も回転されます。図 8.3 に、`rotate` オプションの一般的な動作を図示します。

画像を回転させて配置 まずはじめに、画像を 90° 反時計回りに回転させることに挑戦してみましょう。オブジェクトを配置する前に、座標系は参照点で 90° 反時計回りに回転されます。回転後のオブジェクトの右下隅 (回転前のオブジェクトの左下隅だった所) が、最終的に参照点の位置になります。この場合を示したのが図 8.6a です。

回転は座標系全体に影響するので、枠の回転されます。同様に、画像を 30° 反時計回りに回転することができます (図 8.6b 参照)。

画像を回転してはめ込み 今度は、画像を 90° 反時計回りに回転させて、縦横比を保ちつつ枠にはめ込むことに挑戦してみましょう。これは `fitmethod=meet` を使えば実現できます (図 8.6c 参照)。同様に、画像を 30° 反時計回りに回転させて、その画像を縦横比を保ちつつ枠にはめ込むことができます (図 8.6d 参照)。

図 8.6 画像を回転

生成される出力	PDF_fit_image に与えるオプションリスト
a) 	<code>boxsize={70 45} rotate=90¹</code>
b) 	<code>boxsize={70 45} rotate=30¹</code>
c) 	<code>boxsize={70 45} rotate=90 fitmethod=meet</code>
d) 	<code>boxsize={70 45} rotate=30 fitmethod=meet</code>

1. デフォルト `fitmethod=nofit` ですので、この `boxsize` オプションは実際には必要ありません。

8.4.5 ページ寸法の調整

ページ寸法を画像に合わせる 今度の例としては、ページの大きさをオブジェクトの大きさに自動的に合わせましょう。これはたとえば、さまざまな画像を PDF 形式でアーカイブしておきたいときなどに有用です。参照点 `(x,y)` を使えば、ページをオブジェクトの寸法とちょうど同じにするか、それとも多少大きめや小さめにするかを、指定することができます。ページ寸法を大きめにする (図 8.7 参照)、画像のまわりにふちが残ります。ページ寸法を画像より小さくすると、画像の一部は切り落とされます。まずは、ページ寸法をオブジェクトの大きさとちょうど同じにしましょう：

```
p.fit_image(image, 0, 0, "adjustpage");
```

次のコード断片は、ページ寸法を `x・y` 方向に 40 単位ずつ増やしていますので、オブジェクトのまわりに白ふちが作成されます：

図 8.7
ページ寸法の調
整。左から、ちよ
うど・大きめ・小
さめ



```
p.fit_image(image, 40, 40, "adjustpage");
```

次のコード断片は、ページ寸法を $x \cdot y$ 方向に 40 単位ずつ減らしています。オブジェクトはページの端で切り落とされますので、オブジェクトの一部（幅 40 単位の）は見えなくなります：

```
p.fit_image(image, -40, -40, "adjustpage");
```

$x \cdot y$ 座標を手段として配置する方法（ページの端、または一般には座標軸からの、オブジェクトまでの間隔を指定する方法）のほかに、はめ込み枠を指定する方法もあります。これは、さまざまな組版規則に従ってオブジェクトが配置される長方形の領域です。この組版規則は、`boxsize · fitmethod · position` オプションで制御することができます。

取り込み PDF ページのページ枠群を転写 取り込み PDF ページのすべてのページ枠（MediaBox・CropBox など）を、カレント出力ページへコピーすることができます。`cloneboxes` オプションを、すべての関連する枠の値を読み取るために `PDF_open_pdi_page()` に与える必要があり、そしてその枠の値をカレントページに適用するために `PDF_fit_pdi_page()` にも与える必要があります：

```
/* ページを開き、ページ枠エンタリー群を転写 */
inpage = p.open_pdi_page(indoc, 1, "cloneboxes");
...
/* 出力ページをダミーページ寸法で開く */
p.begin_page_ext(10, 10, "");
...
/*
* 取り込みページを出力ページ上に配置し、入力ページ内にある
* すべてのページ枠を転写。これは、begin_page_ext() で用いた
* ダミー寸法をオーバーライドします。
*/
p.fit_pdi_page(inpage, 0, 0, "cloneboxes");
```

この技法を活用すると、生成 PDF のページ寸法や裁ち落としなどが必ず入力文書のページと同じになるようにすることができます。これは特にプリプレス分野において重要です。

8.4.6 配置された画像と PDF ページに関する情報をクエリー

配置された画像とテンプレートに関する情報 `PDF_info_image()` メソッドを用いて、画像・テンプレート情報をクエリーすることができます。このメソッドで使えるキーワードは、一般的な画像情報（例：幅・高さをピクセル単位で）のみならず、その画像の出力ページ上への配置に関する情報も網羅しています（例：はめ込み計算実行後の幅・高さを絶対値で）。

以下のコード断片は、ピクセル寸法と、ある特定のはめ込みオプションにより画像を配置した後の絶対寸法の両方を取得します：

```
String optlist = "boxsize={300 400} fitmethod=meet orientate=west";
p.fit_image(image, 0.0, 0.0, optlist);

imagewidth = (int) p.info_image(image, "imagewidth", optlist);
imageheight = (int) p.info_image(image, "imageheight", optlist);
System.err.println("画像寸法 (ピクセル単位) : " + imagewidth + " x " + imageheight);

width = p.info_image(image, "width", optlist);
height = p.info_image(image, "height", optlist);
System.err.println("画像寸法 (ポイント単位) : " + width + " x " + height);
```

配置された PDF ページに関する情報 `PDF_info_pdi_page()` メソッドを用いて、配置 PDF ページに関する情報をクエリーすることができます。このメソッドで使えるキーワードは、元のページに関する情報（例：その幅・高さ）のみならず、その取り込み PDF の出力ページ上への配置に関する情報も網羅しています（例：はめ込み計算実行後の幅・高さ）。

以下のコード断片は、取り込みページの元の寸法と、ある特定のはめ込みオプションによりそのページを配置した後の寸法の両方を取得します：

```
String optlist = "boxsize={400 500} fitmethod=meet";
p.fit_pdi_page(page, 0, 0, optlist);

pagewidth = p.info_pdi_page(page, "pagewidth", optlist);
pageheight = p.info_pdi_page(page, "pageheight", optlist);
System.err.println("元のページ寸法: " + pagewidth + " x " + pageheight);

width = p.info_pdi_page(page, "width", optlist);
height = p.info_pdi_page(page, "height", optlist);
System.err.println("配置ページ寸法: " + width + " x " + height);
```


9 テキストと表の組版

9.1 テキスト行を配置・はめ込む

一行のテキストをページ上に配置するためのメソッド `PDF_fit_textline()` にはさまざまな組版オプションがあります。この節ではもっとも重要なオプションをいくつかよく使われる応用例を用いて解説します。こうしたオプションの完全な説明は *PDFlib API リファレンス* にあります。`PDF_fit_textline()` のオプションの多くは `PDF_fit_image()` のオプションと同じです。ですのでここではテキスト関連の利用例のみを示します。画像の組版については、221 ページ「8.4 画像・グラフィック・取り込み PDF ページを配置」にある作成例を参照するよう推奨します。

以下の利用例では `PDF_fit_textline()` への呼び出しのみを示します。必要なフォントはすでに読み込まれて希望の文字サイズに設定されているものとします。

`PDF_fit_textline()` は、仮想的なテキスト枠を用いてテキストの位置付けを決定します：このテキスト枠の幅はテキストの幅と同じであり、枠の高さはフォントの大文字の高さと同じです。このテキスト枠を変更するには `matchbox` オプションを用います。

以下の作成例では、参照点の座標は `PDF_fit_textline()` の `x・y` 引数として与えられます。テキスト行に対するはめ込み枠は、テキストが配置される領域です。それは `PDF_fit_textline()` の `x・y` 引数と適切なオプション (`boxsize・fitmethod・position・rotate`) で指定される長方形領域として定義されます。はめこみ枠は、`margin` オプションを用いて、左/右または上/下へ縮めることもできます。

クックブック テキスト出力の諸側面に関するコードサンプルが *PDFlib クックブック* の `text_output` カテゴリにあります。

9.1.1 単純なテキスト行配置

テキストを参照点に位置付け デフォルトでは、テキストは左下隅を参照点に合わせて配置されます。しかしこの例では、テキストの下端中央を参照点に合わせて配置したいのです。以下のコード断片は、テキスト枠の下端中央を参照点(30, 20)に合わせて配置します。

```
p.fit_textline(text, 30, 20, "position={center bottom}");
```

図 9.1 に、中央揃えのテキスト配置の様子を図示します。同様に、キーワード `left・right・center・top・bottom` の組み合わせを変えた `position` オプションを使って、参照点にテキストを配置することができます。

図 9.1
テキストの
中央揃え

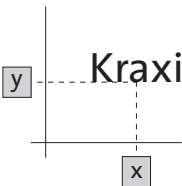
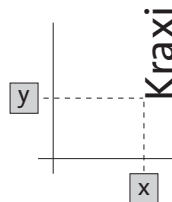


図 9.2
西向きの
単純なテキスト



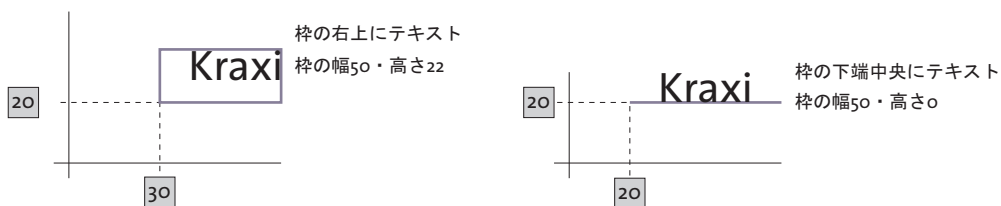


図 9.3 テキストを枠内に位置付け

テキストの向きを変えて配置 次は、テキストを回転させて、その左下隅（回転後の）を参照点に合わせて配置してみましょう。以下のコード断片は、テキストを西（90°反時計回り）に向けた後、その回転したテキストの左下隅を参照点 $(0, 0)$ へ並行移動させます。

```
p.fit_textline(text, 0, 0, "orientate=west");
```

図 9.2 に、単純なテキストの向きを変えて配置した様子を图示します。

9.1.2 テキストを枠内に位置付け

テキストを位置付けるには、幅と高さをあらかじめ定義した枠をあわせて使うこともでき、テキストはその場合、この枠に対して相対的に位置付けることが可能です。図 9.3 に、その一般的なはたらきを图示します。



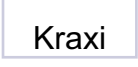

テキストを枠内に位置付け 長方形の枠を定義して、この枠内の右上にテキストを配置しましょう。以下のコード断片は、幅 50 単位・高さ 22 単位の枠を、参照点 $(30, 20)$ に定義します。図 9.4a のように、テキストは枠の右上に配置されます。

同様に、下端中央にテキストを配置することもできます。この場合を图示したのが図 9.4b です。

枠とテキストの間に間隔をあけるには、*margin* オプションを付け加えます（図 9.4c 参照）。

なお、図中の枠や線は、枠の寸法を視覚化するために描いたものであり、実際の出力には現れません。

図 9.4 さまざまな位置付けオプションに従ってテキストを枠内に配置

生成される出力	PDF <code>fit_textline()</code> に与えるオプションリスト
a) 	<code>boxsize={50 22} position={right top}</code>
b) 	<code>boxsize={50 22} position={center bottom}</code>
c) 	<code>boxsize={50 22} position={center bottom} margin={0 3}</code>
d) 	<code>boxsize={50 0} position={center bottom}</code>

生成される出力

PDF_fit_textline() に与えるオプションリスト

e)

Kraxi

```
boxsize={0 35} position={left center} orientate=west
```

テキストを横線や縦線で整列させる テキストの位置付けを、横線や縦線（すなわち高さや幅がゼロの枠）に沿って行うというのは、若干極端なケースではありますが、有用な場合もあります。図 9.4d は、テキストを枠に対して下端中央揃えで配置したものです。幅を 50、高さを 0 としたことで、枠はまるで横線のようにになっています。

テキストを縦線に沿って中央揃えするためには、西向きにして、枠に対して左端中央に位置付けましょう。この場合を図 9.4e に示します。

9.1.3 テキストを枠へはめ込み

この項では、さまざまなはめ込み方式を用いて、テキストを枠へはめ込みましょう。カレントのフォントと文字サイズはどの例でも同じとしておき、さまざまなはめ込み方式にもなって文字サイズなどのプロパティが変わる様子がわかるようにします。

デフォルトの場合から始めましょう：何のはめ込み方式も用いないで、切り落としも拡張も一切されないようにする場合です。テキストは、幅 100 単位・高さ 35 単位の枠の中央に配置されます（図 9.5a 参照）。

枠の幅を 100 から 50 単位に縮めても、出力には全く影響を与えません。テキストは元の文字サイズを保ち、枠からはみ出します（図 9.5b 参照）。

テキストを小さな枠内へ縦横比を保ってはめ込み それでは、テキスト全体を枠の中へ、縦横比を保ってはめ込んでみましょう。これはオプション `fitmethod=auto` で実現できます。図 9.5c では、枠の幅が充分広いので、テキストはまったく元の大きさのまま、変わらずに枠内に配置されています。

枠の幅を 100 から 58 に縮小すると、テキストは長すぎて収まりきらなくなります。はめ込み方式 `auto` はテキストに長体を、`shrinklimit` オプション（デフォルト：0.75）を限度としてかけようとしています。図 9.5d は、テキストが元の長さの 75 パーセントに縮んだ様子を示します。

枠の幅をさらに 30 単位まで縮めると、テキストは長体をかけても収まりきらなくなります。すると、`meet` 方式が適用されます。これは、テキスト全体が枠に収まるまで文字サイズを下げます。この場合を示したのが図 9.5e です。

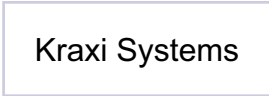
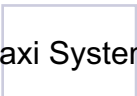
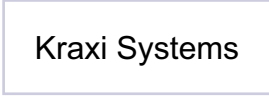
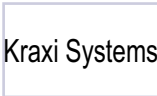

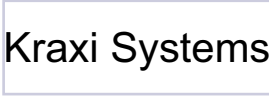

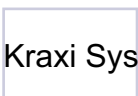
テキストの文字サイズを上げて枠へはめ込み テキストを枠の幅（または高さ）いっぱいに広げて、ただし縦横比は保ったまま、はめ込みたい時があるかもしれません。テキストより大きい枠に対して `fitmethod=meet` を用いると、テキストの幅が枠の幅と一致するまでテキストが大きくなります。この場合を図示したのが図 9.5f です。

テキストを枠いっぱいにはめ込み さらに、テキストが枠の中を埋めつくすようにはめ込むことも可能です。この場合は、`fitmethod=entire` を使います。しかしこの設定は、テキストをほぼ確実にゆがめてしまうので、使うことはめったにないでしょう（図 9.5g 参照）。

テキストを枠で切り落としてはめ込み これもレアなケースですが、テキストを元のサイズのままはめ込んで、もしも枠からはみ出た部分は切り落としたい時もあるかもしれま

せん。その場合は `fitmethod=clip` が使えます。図 9.5h では、テキストを枠の左端に配置していますが、枠の幅が足りません。テキストは右側が切り落とされます。

図 9.5 さまざまなオプションに従ってテキストをページ上の枠へはめ込む

生成される出力	PDF <code>fit_textline()</code> に与えるオプションリスト
a) 	<code>boxsize={100 35} position=center fontsize=12</code>
b) 	<code>boxsize={50 35} position=center fontsize=12</code>
c) 	<code>boxsize={100 35} position=center fontsize=12 fitmethod=auto</code>
d) 	<code>boxsize={58 35} position=center fontsize=12 fitmethod=auto</code>
e) 	<code>boxsize={30 35} position=center fontsize=12 fitmethod=auto</code>
f) 	<code>boxsize={100 35} position=center fontsize=12 fitmethod=meet</code>
g) 	<code>boxsize={100 35} position=center fontsize=12 fitmethod=entire</code>
h) 	<code>boxsize={50 35} position={left center} fontsize=12 fitmethod=clip</code>

テキストの縦中央揃え PDF `fit_textline()` におけるテキストの高さは、デフォルトではキャップハイト、すなわち大文字の H の高さです。テキストを枠の中央に位置付けると、縦方向にはそのキャップハイトにもとづいて中央揃えされます (図 9.6a 参照)。

それ以外の高さをテキスト枠に対して指定するには、範囲枠機能を使います (275 ページ「9.4 範囲枠」も参照)。PDF `fit_textline()` の `matchbox` オプションは、テキスト行の高さを定義しており、与えられた文字サイズのキャップハイトがそのデフォルトになっています。この範囲枠の高さは、その `boxheight` サブオプションにもとづいて算出されます。`boxheight` サブオプションは、ベースラインからテキスト上端・下端までの距離を決定します。デフォルトの設定は `matchbox={boxheight={capheight none}}`、すなわち範囲枠の上

端はベースラインより上にあってキャップハイトに一致し、範囲枠の下端はベースラインを下へ越えません。

範囲枠の大きさを図示するため、ここでは赤く色を塗りましょう (図 9.6b 参照)。図 9.6c では、テキストが *xheight* に基づいて縦中央揃えされるように、その対応する高さの範囲枠を定義しています。

図 9.6d ~ f に、有用なさまざまな *boxheight* 設定の範囲枠 (赤) と、その決める高さにもとづいて枠の中で中央揃えされたテキストを示します。

図 9.6 さまざまな範囲枠高さに従ってテキストを枠へはめ込む

生成される出力	PDF_fit_textline() に与えるオプションリスト
a)	<code>boxsize={80 20} position=center fitmethod=auto</code>
b)	<code>boxsize={80 20} position=center fitmethod=auto matchbox={boxheight={capheight none} fillcolor=mistyrose}</code>
c)	<code>boxsize={80 20} position=center fitmethod=auto matchbox={boxheight={xheight none} fillcolor=mistyrose}</code>
d)	<code>boxsize={80 20} position=center fitmethod=auto matchbox={boxheight={ascender none} fillcolor=mistyrose}</code>
e)	<code>boxsize={80 20} position=center fitmethod=auto matchbox={boxheight={ascender descender} fillcolor=mistyrose}</code>
f)	<code>boxsize={80 20} position=center fitmethod=auto matchbox={boxheight={fontsize none} fillcolor=mistyrose}</code>

9.1.4 テキストを文字で揃える

テキストを文字で揃える テキストを、ある特定の文字で揃えたい場合があるかもしれません。たとえば数値の小数点揃えなどです。図 9.7a に示すように、テキストははめ込み枠の中央に位置付けられます。PDF_fit_textline() で *alignchar=.* オプションを用いることで、数値が点で揃います。

点を枠の中央に配置している *position* オプションを省くこともできます。そうすれば、デフォルトの *position={left bottom}* が用いられるので、点は参照点に配置されます (図 9.7b 参照)。一般に、揃え文字はその右下隅が参照点に配置されます。

図 9.7 テキスト行を点で揃える


生成される出力	PDF_fit_textline() に与えるオプションリスト
a)	<code>boxsize={70 8} position={center bottom} alignchar=.</code>
b)	<code>boxsize={70 8} position={left bottom} alignchar=.</code>

9.1.5 スタンプを配置

クックブック 完全なコードサンプルがクックブックの `text_output/simple_stamp` トピックにあります。

テキストの回転を指定しなくても、スタンプという便利な機能を使えば、テキストを枠内に対角線に沿って配置することができます。このスタンプ機能は洗練された自動計算を行い、テキストが枠内いっぱいにはたがるよう文字サイズと回転角を決定します。対角線スタンプを、たとえばページの背景などに配置するには、`PDF_fit_textline()` で `stamp` オプションを指定します。`stamp=llzur` を指定すると、テキストははめ込み枠の左下隅から右上隅へ配置されます。これに対し、`stamp=ulzlr` を指定すると、テキストははめ込み枠の左上隅から右下隅へ配置されます。`fitbox` オプションは無視されます。図 9.8 では、`showborder=true` を用いてはめ込み枠とスタンプの外接枠を図示しています。

図 9.8 左下から右上へのスタンプのようにテキスト行をはめ込む

生成される出力	PDF <code>fit_textline()</code> に与えるオプションリスト
	<code>fontsize=8 boxsize={160 50} stamp=llzur showborder=true</code>

9.1.6 リーダーを用いる

リーダーを使うと、はめ込み枠の端とテキストとの間の余白を埋めることができます。たとえば点リーダーは、目次の各エントリとその対応するページ番号をつないで見やすいようによく利用されます。

目次のリーダー `PDF_fit_textline()` で `leader` オプションと `alignment={none right}` サブオプションを用いると、テキスト行の右にリーダーが付加されて、テキスト枠の右端まで繰り返されます。リーダー右端と右枠との間隔は一定ですが、テキストとリーダー左端との間隔は変動の可能性があります (図 9.9a 参照)。

クックブック テキスト行の中での点リーダーの使用例を示す完全なコードサンプルがクックブックの `textoutput/leaders_in_textline` トピックにあります。

クックブック テキストフローの中での点リーダーの使用例を示す完全なコードサンプルがクックブックの `textflow/dot_leaders_with_tabs` トピックにあります。

ニュース電光掲示板のリーダー 別の用例としては、ニュース電光掲示板効果を作成したい場合もあるかもしれません。ここではプラスとスペース「+」をリーダーに使いましょう。テキスト行は中央に配置し、リーダーはそのテキスト行の前と後に印字させます (`alignment={left right}`)。リーダーの左右端は左右の枠に揃い、テキストとの間隔は変動の可能性があります (図 9.96b 参照)。

図 9.9 リーダーを用いたテキスト行をはめ込む

生成される出力	PDF_fit_textline() に与えるオプションリスト
<pre> a) Features of Giant Wing Description of Long Distance Glider..... Benefits of Cone Head Rocket..... </pre>	<pre> boxsize={200 10} leader={alignment={none right}} </pre>
<pre> b) ++++++++ Giant Wing in purple!+++++++ ++ Long Distance Glider with sensational range! ++ ++++++ Cone Head Rocket incredibly fast! ++++++ </pre>	<pre> boxsize={200 10} position={center bottom} leader={alignment={left right}} text={+ } </pre>

9.1.7 パス上テキスト

テキストを直線上に配置するのではなく、任意のパス上に配置することもできます。PDFlib は個々のキャラクターをパス上に、テキストがそのパスの曲線に沿うように配置します。パス上テキストを作成するには `PDF_fit_textline()` の `textpath` オプションを用います。パスはそれ以前に作成済みである必要があり、パスハンドルで表します。パスハンドルは、`PDF_add_path_point()` および関連するパスオブジェクト関数群で明示的にパスを構築するか、あるいは既存のラスター画像内のクリッピングパスのハンドルを取得することによって作成できます。以下のコード断片は、1 個のパスを作成し、テキストをそのパス上に配置します (図 9.10 参照) :

```

/* パスを原点に定義 */
path = p.add_path_point( -1, 0, 0, "move", "");
path = p.add_path_point(path, 100, 100, "control", "");
path = p.add_path_point(path, 200, 0, "circular", "");

/* テキストをパス上に配置 */
p.fit_textline("Long Distance Glider with sensational range!", x, y,
    "textpath={path=" + path + "} position={center bottom}");

/* 例ですのでパスも描いてみせましょう */
p.draw_path(path, x, y, "stroke strokecolor=dodgerblue");
    
```

クックブック 完全なコードサンプルがクックブックの `text_output/text_on_a_path` トピックにあります。

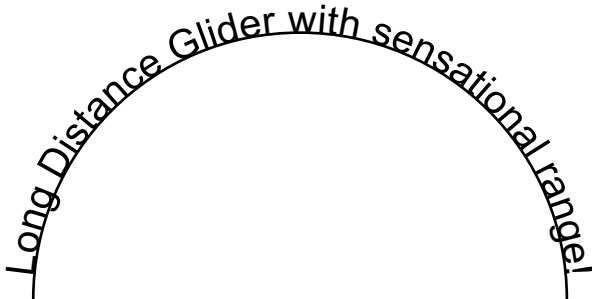


図 9.10
パス上テキスト

画像のクリッピングパスを用いてテキストを配置 パス関数群でパスオブジェクトを手作業で構築するのではなく、画像からクリッピングパスを抽出して、そのパス上にテキストを配置することもできます。この画像は `honorclippingpath` オプションとともに読み込んである必要があります、かつ、求めるパスがその画像のデフォルトのクリッピングパスでない場合には `clippingpathname` も `PDF_load_image()` に与える必要があります：

```
image = p.load_image("auto", "image.tif", "clippingpathname={path 1}");

/* 画像のクリッピングパスからパスオブジェクトを作成 */
path = (int) p.info_image(image, "clippingpath", "");
if (path == -1)
    throw new Exception("エラー：クリッピングパスが見つかりません!");

/* テキストをパス上に配置 */
p.fit_textline("Long Distance Glider with sensational range!", x, y,
    "textpath={path=" + path + "} position={center bottom}");
```

パスとテキストの間にアキを作る デフォルトでは、PDFlib は個々のキャラクターをパス上に直接配置します。すなわち、グリフとパスの間にはアキは全くありません。パスとテキストの間にアキを作りたいときは、キャラクター枠を延長することができます。これは、`matchbox` オプションの `boxheight` サブオプションでキャラクター枠の縦延長を指定することで実現できます。以下のオプションリストはアセンダーを考慮に入れています (図 9.11 参照)：

```
p.fit_textline("Long Distance Glider with sensational range!", x, y,
    "textpath={path=" + path + "} position={center bottom}" +
    "matchbox={boxheight={capheight descender}}");
```

9.1.8 影付きテキスト

`shadow` オプションを用いると、テキストの影付き効果を生み出すことができます。影の色と、メインテキストからの横・縦距離を、サブオプション内で指定できます：

```
p.fit_textline("Long Distance Glider", x, y,
    "fillcolor=rosybrown shadow={offset={3, -3}}");
```

9.1.9 Acrobat で編集できる透かし

テキスト行機能を使うと、テキストに対してさまざまな組版オプションを適用することができます。このテキストは、ページに組み込まれた一部分となり、最終 PDF 文書内で簡単に編集することはできません。しかし、Acrobat は「透かし」機能を提供しています。

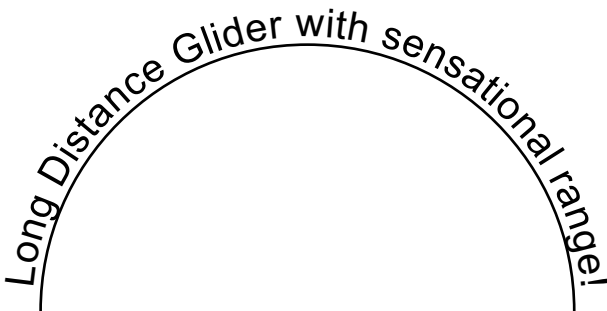


図 9.11
パス上テキストで、パスとテキストの間にアキをあげた

この機能を利用すると、Adobe Acrobat で後で変更や削除できるページ内容を作成できません（無料の Reader では不可）。ただし、この機能は PDF 標準 ISO 32000 に含まれているものではなく、Acrobat に実装されている独自拡張です。ですのでこの機能は、すべての標準準拠 PDF ビューアーで動作することが保証されているものではありません。

Acrobat の透かしは、1 行ないし複数のテキスト行か、1 つの画像か、1 つの PDF ページを、1 つのテンプレート（フォーム XObject）内にラップして、その透かしの組版特性群の XML 記述を加えることによって動作します。既存の透かしを Acrobat DC で編集する操作は以下のとおりです：「ツール」→「PDF を編集」→「透かし」→「更新 ...」または「削除 ...」。すると「透かしを更新」ダイアログが開きますので、そこで透かしのテキストや、その配置・体裁や、対象とするページ範囲を指定できます。このテキストの体裁は、このダイアログを用いた後に変わる可能性もあります。なぜなら、このダイアログは、文字間隔や下線、テキスト表現モードなど、あらゆるテキスト組版オプションに対応しているわけではないからです。

この透かしは、画面表示上で、または印刷ページ上で、あるいは両方で印字されるよう指定することが可能です（この Acrobat のダイアログの中の「表示方法オプション ...」）。これは、適切な画面・印刷設定を施された「透かし」というレイヤーを通じて実現されます。ただし PDF/A・PDF/X・PDF/UA においては、レイヤーとレイヤーオプションにはさまざまな制約が課せられています。

PDFlib は、そのテンプレート機能を拡張することによってこの透かし機能に対応しています。`PDF_begin_template_ext()` を用いてテンプレートを作成する際に、その *watermark* オプションを用いてこれに標識することが可能です。その場合、このテンプレートは、以下のアイテムのうち 1 つを内容として持っている必要があります：

- ▶ `PDF_fit_textline()` を用いて配置されたただ 1 行のテキスト。
- ▶ `PDF_fit_textflow()` を用いて配置された複数行のテキスト。
- ▶ `PDF_fit_image()` を用いて配置された画像。
- ▶ `PDF_fit_pdi_page()` を用いて配置された PDF ページ。

そうして生成された透かしは、自動的にすべてのページに加えられ（あるいは一部のページにも可）、後で Acrobat を用いて変更または削除することが可能です。

以下のコード断片は、編集可能な透かしの定義を演示しています。この透かしは、この透かしを定義した後に作成されるすべてのページに自動的に追加されます。なお、指定している文字サイズは意味を持ちません。なぜなら、こうしてできたテンプレートはいずれにせよそのページ内に収められるからです：

```
p.begin_template_ext(0, 0, "watermark={location=ontop opacity=60%}");

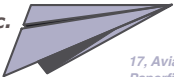
p.fit_textline("Preliminary", 0, 0,
    "fontsize=12 fontname=Helvetica-Bold fillcolor=red " +
    "boxsize={595 842} stamp=ll2ur");

p.end_template_ext(0, 0);
```

9.2 複数行のテキストフロー

1行のテキストをページ上に配置する機能だけでなく、PDFlib は、任意の長さのテキストを組版できるテキストフローという強力な機能をサポートしています。テキストは何行にも、何段にも、あるいは何ページにもわたることができ、またその見ばえはさまざまなオプションで制御することができます。フォント・サイズ・色といった文字属性を、テキストのどの部分にでも適用することができます。テキストの両端揃え・片端揃えや、段落のインデントや、タブ位置といったテキストフロー属性を指定できます。テキストの中にソフトハイフンで示した改行機会が考慮されます。構造情報を持ったタグを与えることも可能です。図 9.12・図 9.13 は、請求書のさまざまな部分がページ上にテキストフロー機能を用いて配置できているさまを例示したものです。こうした出力を制御するためのオプションについて、以下の項で詳しく説明します。

Kraxi Systems, Inc.
Paper Planes



17, Aviation Road
Paperfield
Phone 7079-4301
Fax 7079-4302
info@kraxi.com
www.kraxi.com

Kraxi Systems, Inc. 17, Aviation Road Paperfield

John Q. Doe
255 Customer Lane
Suite B
12345 User Town
Everland

INVOICE **14.03.2004**

	ruler right 30	left 45	right 275	right 375	right 475
	ITEM	DESCRIPTION	QUANTITY	PRICE	AMOUNT
	1	Super Kite	2	20,00	40,00
	2	Turbo Flyer	5	40,00	200,00
	3	Giga Trash	1	180,00	180,00
	4	Bare Bone Kit	3	50,00	150,00
	5	Nitty Gritty	10	20,00	200,00
	6	Pretty Dark Flyer	1	75,00	75,00
	7	Free Gift	1	0,00	0,00
					845,00

leftindent = 55 → Terms of payment: 30 days net. 30 days warranty starting at the day of sale. This warranty covers defects in workmanship only. Kraxi Systems, Inc., at its option, repairs or replaces the product under warranty. This warranty is not transferable. Returns or exchanges are not possible for wet products. **alignment = left**

parindent = 7% → Have a look at our new paper plane models!
 → Our paper planes are the ideal way of passing the time. We offer revolutionary new developments of the traditional common paper planes. If your lesson, conference, or lecture turn out to be deadly boring, you can have a wonderful time with our planes. All our models are folded from one paper sheet.
 They are exclusively folded without using any adhesive. Several models are equipped with a folded landing gear enabling a safe landing on the intended location provided that you have aimed well. Other models are able to fly loops or cover long distances. Let them start from a vista point in the mountains and see where they touch the ground. **alignment = justify**

leading = 140% ↓

leftindent = 75 → 1. Long Distance Glider **rightindent = 60**

leftindent = 105 → With this paper rocket you can send all your messages even when sitting in a hall or in the cinema pretty near the back.

2. Giant Wing **minlinecount = 2**
 An unbelievable sailplane! It is amazingly robust and can even do

図 9.12
テキストフロー
の組版

複数行のテキストフローは、1つの長方形（はめ込み枠という）内にも複数の長方形内にも配置することができます。このはめ込み枠は 1 ページ上にあっても複数ページ上にあってもかまいません。テキストフローをページ上に配置するには以下の手順を踏む必要があります：

- ▶ メソッド `PDF_add_textflow()` が、テキストを一部分ずつ、その対応する組版オプションとともに受け入れて（ここでオプションは別個の文字列として与えることもできますし、テキスト内にインラインで与えることもできます）、そして1つのテキストフローオブジェクトを作成してハンドルを返します。またはメソッド `PDF_create_textflow()` が、組版制御のためのインラインオプションを含むことのできるテキストの全体を、一度の呼び出しで分析します。これらの関数ではページ上にテキストは配置されません。
- ▶ メソッド `PDF_fit_textflow()` が、このテキストフローの全部ないし一部を、与えられたはめ込み枠内に配置します。すべてのテキストを配置するには、この段階を繰り返す必要があります。その際、このメソッドを呼び出すたびに、新しいはめ込み枠を与えます。このはめ込み枠は、同じページにあっても、別のページにあってもかまいません。
- ▶ メソッド `PDF_delete_textflow()` が、テキストフローを文書内に配置した後に、このテキストフローオブジェクトを削除します。

テキストフローを作成するための関数 `PDF_add/create_textflow()` は、組版処理を制御するためのさまざまなオプションに対応しています。このオプションはメソッドのオプションリストで与えることもできますし、あるいはテキスト内にインラインオプションとして入れ込むこともできます。`PDF_info_textflow()` を使うと、組版の結果や、その他テキストフローに関するたくさんの詳細をクエリーすることができます。以下、テキストフローの配置について、いくつかのよくある応用例を例にして説明します。テキストフローオプションの完全な一覧は *PDFlib API リファレンス* にあります。

`PDF_add/create_textflow()` で対応しているオプションの中には `PDF_fit_textline()` と同様のものがたくさんあります。ですから 229 ページ「9.1 テキスト行を配置・はめ込む」の例をあわせてよく把握しておくといよいでしょう。以下の項では、複数行テキストに関するあるオプションだけを解説します。

クックブック テキスト出力の諸側面に関するコードサンプルが *PDFlib クックブック* の `text_output` カテゴリにあります。

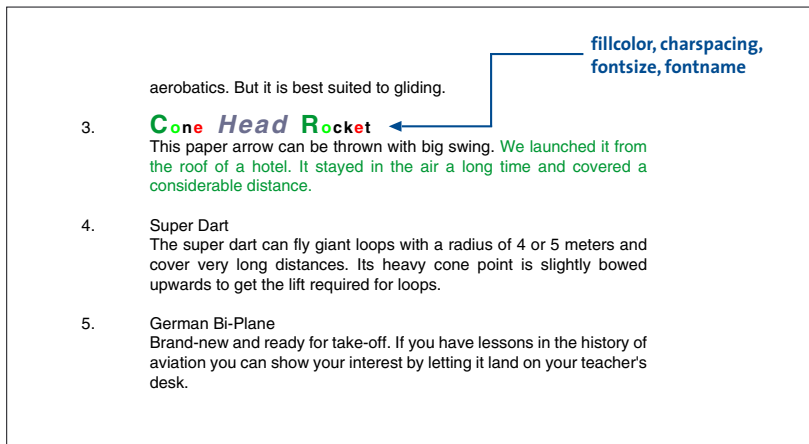


図 9.13
テキストフロー
の組版

9.2.1 テキストフローをはめ込み枠に配置

テキスト枠に対するはめ込み枠は、テキストが配置される領域です。それは `PDF_fit_textflow()` の `llx · lly · urx · ury` 引数で指定される長方形領域として定義されます。

テキストを1つのはめ込み枠に配置 簡単な例から始めましょう。以下のコード断片は、`PDF_add_textflow()` への呼び出しを2回使って、ボールドテキストの部分と標準テキストの部分をつなげます。フォントと文字サイズを明示的に指定しています。1回目の `PDF_add_textflow()` への呼び出しでは、`-1` を与えれば、テキストフローハンドルが返ってきますので、その後もしまた `PDF_add_textflow()` を呼び出すときがあるならそれが使えます。 `text1 · text2` には、印字したい実際のテキストが入っているものとします。

`PDF_fit_textflow()` を使って、できあがったテキストフローをページ上のはめ込み枠に、デフォルトの組版オプションを用いて配置します。

```
/* テキストをボールドフォントで追加 */
tf = p.add_textflow(-1, text1, "fontname=Helvetica-Bold fontsize=9");
if (tf == -1)
    throw new Exception("エラー : " + p.get_errmsg());

/* テキストを標準フォントで追加 */
tf = p.add_textflow(tf, text2, "fontname=Helvetica fontsize=9");
if (tf == -1)
    throw new Exception("エラー : " + p.get_errmsg());

/* Place all text */
result = p.fit_textflow(tf, left_x, left_y, right_x, right_y, "");
if (!result.equals("_stop"))
    { /* ... */ }

p.delete_textflow(tf);
```

テキストを複数ページ上の2つのはめ込み枠に配置 `PDF_fit_textflow()` で配置したテキストがはめ込み枠内に収まりきらなかったときは、出力は中断されてメソッドは文字列 `_boxfull` を返します。PDFlib はすでに配置したテキストの量を記憶していて、このメソッドが再び呼ばれた時には残りのテキストを引き続き配置します。この他に、新規ページを作成する必要もあるかもしれません。次のコード断片は、1つないし複数のページ上の、ページあたり2つのはめ込み枠に、1つのテキストフローを、テキストをすべて配置しおわるまで配置する方法を演示しています (図 9.14 参照)。

クックブック 完全なコードサンプルがクックブックの `textflow/starter_textflow` トピックにあります。

```
/* テキスト全部が配置されるまで回る。配置するべきテキストがまだあるなら新規ページを作成。
 * 全ページに2段組を作成。
 */
do
{
    String optlist = "verticalalign=justify linespreadlimit=120%";

    p.begin_page_ext(0, 0, "width=a4.width height=a4.height");

    /* 1段目に流し込み */
    result = p.fit_textflow(tf, llx1, lly1, urx1, ury1, optlist);

    /* まだテキストがあるなら2段目に流し込み */
```



```

if (!result.equals("_stop"))
    result = p.fit_textflow(tf, llx2, lly2, urx2, ury2, optlist);

p.end_page_ext("");

/* 「_boxfull」ならまだテキストがあるので続ける必要がある。
 * 「_nextpage」は「新規段組を開始」と解釈
 */
} while (result.equals("_boxfull") || result.equals("_nextpage"));

/* エラーかどうかをチェック */
if (!result.equals("_stop"))
{
    /* 枠がとんでも小さくてテキストが全然入らないときは「_boxempty」が起こる。
    */
    if (result.equals( "_boxempty"))
        throw new Exception("エラー : " + p.get_errmsg());
    else
    {
        /* それ以外の戻り値は「return」オプションによるユーザー終了。
        * これを扱うにはそのためのコードが必要。
        */
    }
}
p.delete_textflow(tf);

```

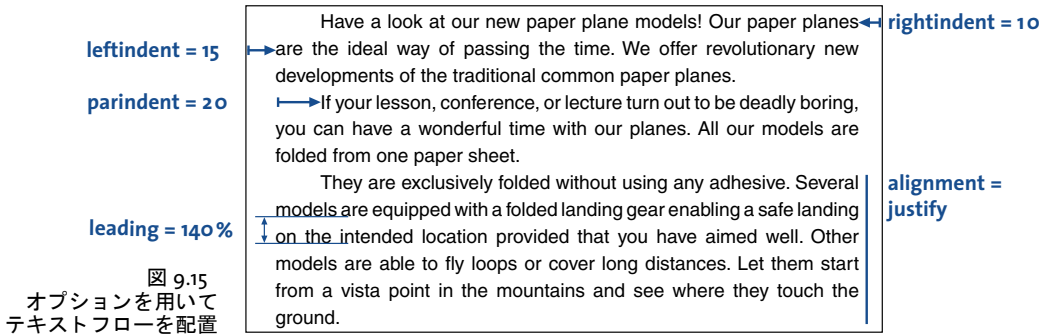
9.2.2 段落の組版のオプション

上の例では段落に、デフォルトの設定を用いました。デフォルトは左揃え、行送り 100% (文字サイズそのまま)、などとなっています。

段落の組版を調整したいなら、**PDF_add_textflow()** にオプションを追加できます。たとえばテキストを左余白から 15 単位、右余白から 10 単位、インデントしたいとします。各段落の先頭行はさらに 20 単位インデントしましょう。テキストを両端揃えとし、行送りは 140% に上げましょう。そして文字サイズを 8 ポイントに下げます。これを実現するには、**PDF_add_textflow()** のオプションリストを以下のように拡張します (図 9.15 参照) :



図 9.14 テキストフローを 2 つのはめ込み枠に配置



```
String optlist =
    "leftindent=15 rightindent=10 parindent=20 alignment=justify " +
    "leading=140% fontname=Helvetica fontsize=8";
```

9.2.3 インラインオプションリストとマクロ

図 9.15 のテキストはまだ完璧ではありません。見出し「Have a look at our new paper plane models!」を独立した行にして、フォントももっと大きくして、そして中央揃えにしましょう。これを実現するにはいくつかの方法があります。

インラインオプションリスト ここまでは組版オプションは、メソッドに直接与えるオプションリスト内で指定してきました。今回もこれと同じやり方を続けるならば、テキストを分割して二度の呼び出しに分ける必要があります。一度目で見出しを、二度目で残りのテキストを配置するわけです。ですが、場合によっては、たとえば書式変更箇所が多いときなどは、この方法は面倒かもしれません。

オプションの取り扱いを楽にするために、テキストフローに対してインラインオプションを与えることもできます。すなわち、テキスト内容と組版オプションを必要に応じて混在させるのです。`PDF_create_textflow()` と `PDF_add_textflow()` で `inlineoptions` オプションを用いると、テキストと、そのテキストの中に直接入れ込まれたインラインオプションが解釈されます。これは単純にオプションをテキスト内に入れ込むということです。インラインオプションリストはテキスト本体の一部として与えられます。デフォルトでは、インラインオプションはキャラクター「`<`」と「`>`」ではさみます。それでは次のように、見出しと残りの段落に対するオプションをテキスト本体の中に入れ込んでみましょう。

注 以下作成例ではすべて、インラインオプションリストに色をつけて示します。改段落キャラクターを矢印で視覚化します。

```
<leftindent=15 rightindent=10 alignment=center fontname=Helvetica fontsize=12>
encoding=winansi>Have a look at our new paper plane models! ←
<alignment=justify fontname=Helvetica leading=140% fontsize=8>
Our paper planes are the ideal way of passing the time. We offer
revolutionary new developments of the traditional common paper planes. ←
<parindent=20>If your lesson, conference, or lecture
turn out to be deadly boring, you can have a wonderful time
with our planes. All our models are folded from one paper sheet. ←
They are exclusively folded without using any adhesive. Several
models are equipped with a folded landing gear enabling a safe
landing on the intended location provided that you have aimed well.
```

Other models are able to fly loops or cover long distances. Let them start from a vista point in the mountains and see where they touch the ground.

デフォルトでは `PDF_add_textflow()` はインラインオプションリストを探します。そこでオプションリストをはさむキャラクターは、`begoptlistchar`・`endoptlistchar` オプションで定義しなおすこともできます。`begoptlistchar` オプションにキーワード `none` を与えるとオプションリストの検出は完全に無効になります。これは、テキストがインラインオプションリストをまったく含まない場合に、「`<`」・「`>`」を確実に通常のキャラクターとして処理させたいときに有用です。

`PDF_add_textflow()` は、`inlineoptions` オプションを与えた場合にのみインラインオプションリストを探します。

記号キャラクターとインラインオプションリスト 記号キャラクターはテキストフローにおいて、インラインオプションリストと組み合わせも使うことができます。インラインオプションリストを開始するキャラクターに対するコード(デフォルトでは:「`<`」U+003C)は、`encoding=builtin` によるフォントに対するテキスト内では記号コードとして解釈されません。これと同じコードの記号グリフを選択するには、テキストフォントに対して利用可能な回避策がそのまま使えます。すなわち、開始キャラクターを `begoptlistchar` オプションで再定義するか、あるいは `textlen` オプションを用いて記号グリフの数を指定します。なお、文字参照 (`<` など) を回避策として用いることはできません。また、`PDF_add_textflow()` で `inlineoptions` オプションを用いている場合には `textlen` オプションは利用できないことにも留意してください。

マクロ 上記のテキストはいくつかの種類の段落でできています。すなわち見出しと本文であり、本文にはさらにインデントのあるものとないものがあります。こうした各種の段落をそれぞれの形に組版していくわけですが、しかしテキストフローがもっと長ければ同じ指定を何度もしなければなりません。段落替えのたびに、対応するインラインオプション群を書かなくて済むようにするには、インラインオプション群をマクロにまとめれば、テキスト内からそのマクロを名前でも参照することができます。図 9.16 に示すような 3 つのマクロを定義しましょう。`H1` は見出し用、`Body` は本文段落用、`Body_indented` はインデントする段落用です。マクロを利用するには、キャラクター `&` をマクロ名の前につけたものをオプションリストに書きます。以下のコード断片は、上で用いたインラインオプション群に従って 3 つのマクロを定義し、それをテキスト内で利用します：

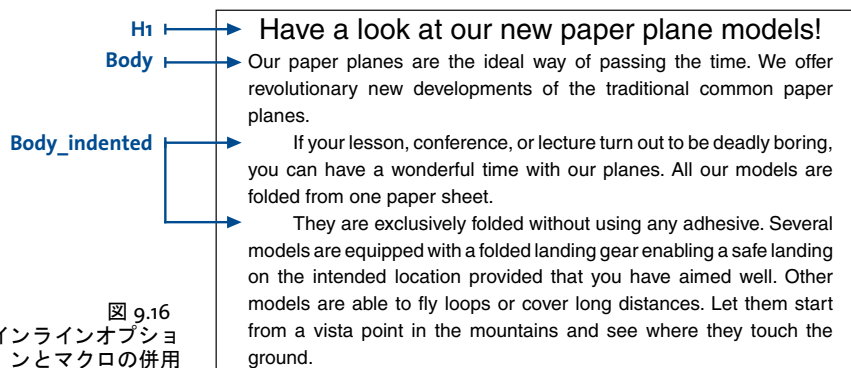


図 9.16
インラインオプションとマクロの併用

```

<macro {
H1 {leftindent=15 rightindent=10 alignment=center
fontname=Helvetica fontsize=12}

Body {leftindent=15 rightindent=10 alignment=justify leading=140%
fontname=Helvetica fontsize=8}

Body_indented {parindent=20 leftindent=15 rightindent=10 alignment=justify
leading=140% fontname=Helvetica fontsize=8}
}>
<&H1>Have a look at our new paper plane models! ←
<&Body>Our paper planes are the ideal way of passing the time. We offer
revolutionary new developments of the traditional common paper planes. ←
<&Body_indented>If your lesson, conference, or lecture
turn out to be deadly boring, you can have a wonderful time
with our planes. All our models are folded from one paper sheet. ←
They are exclusively folded without using any adhesive. Several
models are equipped with a folded landing gear enabling a safe
landing on the intended location provided that you have aimed well.
Other models are able to fly loops or cover long distances. Let them
start from a vista point in the mountains and see
where they touch the ground.

```

オプションの明示的設定 注意しなければならないのは、マクロの中で設定しなかったオプションではすべて前の値が保持されるということです。オプションが勝手に「継承」されておかしな副作用を起こさないようにしたければ、そのマクロが必要とする設定をすべて明示的に指定する必要があります。そうすればそのマクロが他のオプションリストとの順序や組み合わせにかかわらずいつも確実に同じように動作するようにすることができます。

あるいは、この動作を逆に利用して、ある特定の設定群についてはあえて明示的に与えずに、それぞれの利用箇所における設定が保持されるようにするという方式もありえます。たとえば、フォント名は指定するけれども *fontsize* オプションは与えないというマクロを作ってもよいのです。すると、文字サイズをつねに前のテキストと同じにすることができます。

インラインオプションか、メソッドの引数としてオプションを渡すか テキストフローを利用する際には、テキストがプログラムコード内にリテラルに書かれているか、それともどこか外部のソースから来るかというのは重大な違いです。また、組版指示がテキストとは別にあるか、それともテキスト内にあるかというのも重大です。たいていのアプリケーションではテキストはデータベースのような何らかの外部ソースから来るでしょう。現実的には2通りの状況が考えられます：

- ▶ テキスト内容が外部ソースから来て、組版オプションはプログラム内にある場合：実行時に、外部ソースから来る短いテキスト群をプログラム内で合成し、それを組版オプション群と（メソッド呼び出しの所で）組み合わせます。
- ▶ テキスト内容も組版オプションも外部ソースから来る場合：大量のテキストが組版オプション群を含んだ状態で外部ソースから来ます。組版はテキスト内のインラインオプション群によって与えられます。インラインオプションは単純なオプションとして書かれていることもあれば、マクロとして書かれていることもありえます。マクロに関しては、マクロ定義とマクロ呼び出しとを区別して考える必要があります。そうすると、面白い中間的形態が作り出せません：テキスト内容は外部ソースから来て組版のためのマクロ呼び出しもその中に含んでいるのですが、マクロ定義は別に用意しておいて実行時にはじめて与えるようにするのです。この方式の利点は、外部のテキスト

ITEM	DESCRIPTION	QUANTITY	PRICE	AMOUNT
1	Super Kite	2	20.00	40.00
2	Turbo Flyer	5	40.00	200.00
3	Giga Trash	1	180.00	180.00
				TOTAL 420.00

図 9.17
テキストを
表として配置

に手を加えずに組版を簡単に変更できることです。たとえばグリーティングカードを作る際など、さまざまなスタイルをマクロで定義しておけば、カードの雰囲気やデザインを簡単に変更したり、テクニカルにしたり、その他さまざまに変えることができるでしょう。

9.2.4 タブ位置

次の例ではタブキャラクターを用いて左寄せ・右寄せの列のある簡単な表を配置します。表は以下のような複数行のテキストを持ち、個々のエントリが互いにタブキャラクター（矢印で示す）で区切られています：

```
ITEM → DESCRIPTION → QUANTITY → PRICE → AMOUNT ←
1 → Super Kite → 2 → 20.00 → 40.00 ←
2 → Turbo Flyer → 5 → 40.00 → 200.00 ←
3 → Giga Trash → 1 → 180.00 → 180.00 ←
→ → → → TOTAL 420.00
```

この簡単な表を配置するには、`PDF_add/create_textflow()` で以下のオプションリストを用います。`ruler` オプションがタブ位置を定義しており、`tabalignment` がタブ位置の整列方向を指定しており、`hortabmethod` オプションがタブ位置の処理方式を指定しています（出力を図 9.17 に示します）：

```
String optlist =
    "ruler={30    150  250  350} " +
    "tabalignment={left right right right} " +
    "hortabmethod=ruler leading=120% fontname=Helvetica fontsize=9";
```

クックブック 完全なコードサンプルがクックブックの `textflow/tabstops_in_text` トピックにあります。

注 複雑な表を作成するには PDFlib の表機能を推奨します（259 ページ「9.3 表の組版」参照）。

9.2.5 番号付きリストと段落間隔

インラインオプション `leftindent` を用いて番号付きリストを組む方法を以下に示します（図 9.18 参照）：

```
1.<leftindent 10>Long Distance Glider: With this paper rocket you can send all
your messages even when sitting in a hall or in the cinema pretty near the back. ←
<leftindent 0>2.<leftindent 10>Giant Wing: An unbelievable sailplane! It is amazingly
robust and can even do aerobatics. But it is best suited to gliding. ←
<leftindent 0>3.<leftindent 10>Cone Head Rocket: This paper arrow can be thrown with big
```

`leftindent = &indent` \rightarrow
`parindent = -&indent`

図 9.19
マクロによる
番号付きリスト

-
1. Long Distance Glider: With this paper rocket you can send all your messages even when sitting in a hall or in the cinema pretty near the back.
 2. Giant Wing: An unbelievable sailplane! It is amazingly robust and can even do aerobatics. But it is best suited to gliding.
 3. Cone Head Rocket: This paper arrow can be thrown with big swing. We launched it from the roof of a hotel. It stayed in the air a long time and covered a considerable distance.

swing. We launched it from the roof of a hotel. It stayed in the air a long time and covered a considerable distance.

クックブック ビュレットリストと番号付きリストの完全なコードサンプルがクックブックの `textflow/bulleted_list`・`textflow/numbered_list` トピックにあります。

インデント値を設定したり取り消したりしなければならないのが面倒です。しかも各段落ごとに行わなければならないのですからなおさらです。もっとエレガントな解決法としては `list` というマクロを定義します。あわせて便宜のためマクロ `indent` を定義し、定数として使用します。以下のようなマクロの定義になります：

```
<macro {  
  indent {25}}
```

```
list {parindent=-&indent leftindent=&indent hortabsize=&indent  
  hortabmethod=ruler ruler={&indent}}  
>>
```

```
<&list>1. → Long Distance Glider: With this paper rocket you can send all your messages  
even when sitting in a hall or in the cinema pretty near the back. ←  
2. → Giant Wing: An unbelievable sailplane! It is amazingly robust and can even do  
aerobatics. But it is best suited to gliding. ←  
3. → Cone Head Rocket: This paper arrow can be thrown with big swing. We launched  
it from the roof of a hotel. It stayed in the air a long time and covered a  
considerable distance.
```

`leftindent` オプションで左余白からの間隔を指定しています。`parindent` オプションには、`leftindent` を負値にしたものを設定し、各段落の先頭行のインデントを打ち消しています。オプション `hortabsize`・`hortabmethod`・`ruler` では、`leftindent` に対応したタブ位置を指定しています。これによって、番号の後のテキストは、`leftindent` で指定した分だけインデントされるようになります。図 9.19 に `parindent`・`leftindent` オプションの作用を示します。

-
1. Long Distance Glider: With this paper rocket you can send all your messages even when sitting in a hall or in the cinema pretty near the back.
 2. Giant Wing: An unbelievable sailplane! It is amazingly robust and can even do aerobatics. But it is best suited to gliding.
 3. Cone Head Rocket: This paper arrow can be thrown with big swing. We launched it from the roof of a hotel. It stayed in the air a long time and covered a considerable distance.

図 9.18
番号付きリスト

2つの段落の間隔を設定 多くの場合、となりあった段落の間隔は、段落の中の行間よりも、広くとりたいものです。これを実現するには、空の行を挿入して (*nextline* オプションで作成できます)、その空行に適切な行送り値を設定します。この値は、直前の段落の最終行のベースラインと、空行のベースラインとの間隔です。以下の例は、2つの段落の間に 80% のアキを作ります (ここで 100% は、もっとも最近に設定された文字サイズの値に等しい) :

1. → Long Distance Glider: With this paper rocket you can send all your messages even when sitting in a hall or in the cinema pretty near the back.

`<nextline leading=80%><nextparagraph leading=100%>`2. → Giant Wing: An unbelievable sailplane! It is amazingly robust and can even do aerobatics. But it is best suited to gliding.

クックブック 完全なコードサンプルがクックブックの `textflow/distance_between_paragraphs` トピックにあります。

9.2.6 制御キャラクターとキャラクターマッピング

テキストフロー内で制御キャラクター テキストフローの中ではさまざまなキャラクターが特別な扱いを受けます。PDFlib はシンボリックキャラクター名に対応しており、これはその対応する文字コードのかわりとして *charmapping* オプション (テキストを処理する前にその中のキャラクターを置換できるオプション。後述) 内で用いることができます。表 9.1 に、テキストフロー関数群が評価するすべての制御キャラクターを、それぞれのシンボリック名と意味説明とともに挙げてあります。1 つのオプションリストの中で同じオプションを複数回使うことはできませんが、複数のオプションリストを連続して与えることは可能です。たとえば以下の並びは空行を作成します :

```
<nextline><nextline>
```

キャラクターかキャラクター列をマッピング / 除去 *charmapping* オプションを使うと、テキスト内のキャラクターを、別のキャラクターへマップしたり、除去したりすることができます。まずは簡単な場合から始めることにして、テキスト内のすべてのタブをスペースへマップしてみましょう。これを行うには *charmapping* オプションを以下のようにします :

```
charmapping={hortab space}
```

このコマンドではシンボリックキャラクター名 *hortab · space* を用いています。複数のマッピングを一度に行うには、以下のコマンドのようにすれば、すべてのタブと改行キャラクター列をスペースに置換することができます :

```
charmapping={hortab space CRLF space LF space CR space}
```

以下のコマンドはすべてのソフトハイフンを削除します :

```
charmapping={shy {shy 0}}
```

タブ 1 つにつきスペース 4 つに置換します :

```
charmapping={hortab {space 4}}
```

任意長の連続 linefeed キャラクター列を 1 個の linefeed キャラクターに縮めます :

表 9.1 テキストフロー内の制御キャラクターとその意味

Unicode キャラクター	実体名	等価なテキストフローオプション	テキストフロー内における意味
U+0020	SP, space	space	単語揃え・改行
U+00A0	NBSP, nbsp	(なし)	(no-break space) 改行しない空白文字
U+202F	NNBSP, nnbsp	(なし)	(narrow no-break space) 改行しない、かつ組版オプションに従ってその幅を変えることのない固定幅空白文字
U+0009	HT, hortab	(なし)	水平タブ : ruler・tabalignchar・tabalignment オプションに従って処理されます
U+002D	HY, hyphen	(なし)	単語のハイフネーションのための区切り文字
U+00AD	SHY, shy	(なし)	(soft hyphen) ハイフネーション機会。改行箇所でのみ表示される
U+000B U+2028	VT, verttab LS, linesep	nextline	(next line) 次の行へ移る
U+000A U+000D U+000D・U+000A U+0085 U+2029	LF, linefeed CR, return CRLF NEL, newline PS, parasep	next-paragraph	(next paragraph) nextline と同効果。それに加え、parindent オプションが次の行に影響します
U+000C	FF, formfeed	return	PDF_fit_textflow() 関数が中断して文字列 <code>_nextpage</code> を返します。

```
charmapping={linefeed {linefeed -1}}
```

CRLF キャラクター列をそれぞれ 1 個のスペースに置換します :

```
charmapping={CRLF {space -1}}
```

この最後の例についてももう少し詳しく見てみましょう。たとえばどこからテキストを受け取った時、何か他のソフトウェアのせいで行の途中で改行がブツブツ入っていたとしたら、そのままでは適切に組版ができません。適切にはめ込み枠内での組版が行えるようにするには、これらの改行をスペースに置換したいところです。これを行うため、任意長の連続改行を 1 個のスペースに置換しましょう。はじめのテキストは以下のようなものだとします :

```
To fold the famous rocket looper proceed as follows: ← ←
Take a sheet of paper. Fold it ←
lengthwise in the middle. ←
Then, fold down the upper corners. Fold the ←
long sides inwards ←
that the points A and B meet on the central fold.
```

以下のコード断片は、無駄な改行キャラクターを置換したうえでできたテキストを組版する方法を演示しています :

To fold the famous rocket looper proceed as follows:

Take a sheet of paper. Fold it lengthwise in the middle. Then, fold down the upper corners. Fold the long sides inwards that the points A and B meet on the central fold.

図 9.20

上：無駄な改行のあるテキスト

To fold the famous rocket looper proceed as follows: Take a sheet of paper. Fold it lengthwise in the middle. Then, fold down the upper corners. Fold the long sides inwards that the points A and B meet on the central fold.

下：charmapping オプションで改行を置換したもの

```
/* オプションリストを組み立て */
String optlist = "fontname=Helvetica fontsize=9 alignment=justify "
                "charmapping {CRLF {space -1}}"
/* テキストフローをはめ込み枠に配置 */
textflow = p.add_textflow(-1, text, optlist);
if (textflow == -1)
    throw new Exception("エラー：" + p.get_errmsg());

result = p.fit_textflow(textflow, left_x, left_y, right_x, right_y, "");
if (!result.equals("_stop"))
    { /* ... */ }

p.delete_textflow(textflow);
```

図 9.20 に、処置前のテキストのテキストフロー出力と、*charmapping* オプションによる改善後の出力とを示します。

9.2.7 ハイフネーション

PDFlib は自動的にテキストのハイフネーションを行う能力は持ちませんが、テキスト内でソフトハイフンキャラクターによって明示的にハイフネーション機会が示されている場合にはそこで単語を分割することができます。ソフトハイフンキャラクターは Unicode では位置 *U+00AD* にあります。あるいは文字実体を用いることもできます：*­*

グリフ *U+00AD* がそのフォント内で得られる場合にはそれが、そうでない場合には *U+002D* がハイフンキャラクターとして使われます。ソフトハイフンで示されたハイフネーション機会でなくても、単語は強制的にハイフネーションされることがあります。これは単語間隔伸縮や長体など、他の調整手段がうまくいかなかった極端な場合に起こります。

テキスト両端揃えでハイフンキャラクターがある場合とない場合 以下の例では、以下のテキストを両端揃えで印字させます。テキストにはソフトハイフンキャラクターを適宜入れてあります（ここではダッシュで視覚化しています）：

Our paper planes are the ideal way of pas - sing the time. We offer revolu - tionary brand new dev - elop - ments of the tradi - tional common paper planes. If your lesson, confe - rence, or lecture turn out to be deadly boring, you can have a wonder - ful time with our planes. All our models are folded from one paper sheet. They are exclu - sively folded without using any adhe - sive. Several models are equip - ped with a folded landing gear enab - ling a safe landing on the intended loca - tion provided that you

Our paper planes are the ideal way of passing the time. We offer revolutionary brand new developments of the traditional common paper planes. If your lesson, conference, or lecture turn out to be deadly boring, you can have a wonderful time with our planes. All our models are folded from one paper sheet. They are exclusively folded without using any adhesive. Several models are equipped with a folded landing gear enabling a safe landing on the intended location provided that you have aimed well. Other models are able to fly loops or cover long distances. Let them start from a vista point in the mountains and see where they touch the ground.

図 9.21
テキスト両端揃えでソフトハイフンあり。
デフォルト設定と広いはめ込み枠を使用。

Our paper planes are the ideal way of passing the time. We offer revolutionary brand new developments of the traditional common paper planes. If your lesson, conference, or lecture turn out to be deadly boring, you can have a wonderful time with our planes. All our models are folded from one paper sheet. They are exclusively folded without using any adhesive. Several models are equipped with a folded landing gear enabling a safe landing on the intended location provided that you have aimed well. Other models are able to fly loops or cover long distances. Let them start from a vista point in the mountains and see where they touch the ground.

図 9.22
テキスト両端揃えでソフトハイフンなし。
デフォルト設定と広いはめ込み枠を使用。

have aimed well. Other models are able to fly loops or cover long distances. Let them start from a vista point in the mountains and see where they touch the ground.

図 9.21 に、テキスト両端揃えのデフォルト設定で生成されたテキスト出力を示します。完璧な見ええとなつています。なぜなら条件が最適だからです：はめ込み枠が充分広くて、しかも、明示的なハイフネーション機会をソフトハイフンキャラクターで指定しているからです。図 9.22 を見ると、明示的ソフトハイフンがない場合でも出力はおおむね良好です。オプションリストはどちらの場合も以下ようになります：

```
fontname=Helvetica fontsize=9 encoding=winansi alignment=justify
```

9.2.8 ウィド一行・オーファン行

1つの段落の先頭の1行（ないし複数行）だけが段またはページの下端に現れるとき、これをオーファンといいます。同様に、1つの段落の末尾の1行（ないし複数行）が次の段またはページの前頭に現れるとき、これをウィド一行といいます。高品質な文字組版において、孤立したオーファン行またはウィド一行は望ましくないと考えられています。

オーファン制御 テキストフローオプション *minlinecount* は、はめ込み枠の末尾段落の最少行数を指定します。行がそれよりも少ない場合には、それらは次のはめ込み枠の中に配置されます。値 *minlinecount=2* を用いると、はめ込み枠の末尾の段落の1行のオーファン行を避けることができます。

ウィド一行制御 PDFlib は、将来のテキストフロー配置について何も知りませんので、ウィド一行を直接制御することはできません。しかし、クライアントコード内に以下の仕組みでウィド一行制御を実装することは可能です：

- ▶ テキストフローの先頭部分を、先頭はめ込み枠内へブラインドモード（オプション *blind=true*）で、すなわち実際の出力を生成せずにはめ込みます。
- ▶ テキストフローの次の部分を、2番目のはめ込み枠内へブラインドモードではめ込みます。*PDF_info_textflow()* をキーワード *firstparalinecount* とともに呼び出すことによって、この2番目のはめ込み枠の先頭段落の行数をクエリーします。もしこの結果が1ならば、1行ウィド一行を見つけたこととなります。
- ▶ そうであれば、先頭はめ込み枠へ *rewind* オプションを用いて戻ることができますので、アルゴリズムで、ウィド一行を避けるためにはめ込みオプション群を調整する必要があります。

あります。たとえばこれは、先頭はめ込み枠内の行数を *maxlines* オプションを用いて減らすことによって実現できます。

クックブック 完全なコードサンプルがクックブックの *textflow/widows_and_orphans* トピックにあります。

9.2.9 標準改行アルゴリズムの制御

PDFlib は洗練された改行アルゴリズムを実装しています。改行アルゴリズムを制御するテキストフローオプションを表 9.2 に挙げます

改行規則 1つの単語、ないし両端をスペースキャラクターで挟まれた一連のテキストが、1つの行に収まりきらない場合、次の行へ送る必要が出てきます。このような状況で改行アルゴリズムは、どのキャラクターの後で改行が可能かを決定します。

たとえば、 $-12+235/8^45$ のような数式は絶対に途中で改行されませんが、一方、文字列 *PDF-345+LIBRARY* はマイナスの所で次行に送られる可能性があります。テキストがソフトハイフンキャラクターを含んでいればそのうちのいずれかの後で改行される可能性もあります。

括弧と引用符については、開く所と閉じる所とで規則が異なります：括弧や引用符が開く所では改行機会は一切与えられません。引用符については、どれが開いてどれが閉じているのかを判定する手段として、引用符のペアを数えていく仕組みになっています。

表 9.2 改行アルゴリズムを制御するためのオプション

オプション	説明
adjust-method	(キーワード) <i>minspacing</i> ・ <i>maxspacing</i> オプションで指定された制限内で単語間を詰めたり広げたりしてもテキストが1行に収まりきらない時に行の調整に用いる方式。デフォルト: <i>auto</i> 。 auto 以下の方式が順に適用されます: <i>shrink</i> ・ <i>spread</i> ・ <i>nofit</i> ・ <i>split</i> 。 clip <i>nofit</i> (後述) と同じ。ただし、はめ込み枠の右端 (<i>rightindent</i> オプションを考慮) からはみ出した部分を切り落とし。 nofit 最後の単語を次行へ送る。ただし、残される (短い) 行が、 <i>nofitlimit</i> オプションで指定されたパーセント値よりも短くならない場合に限る。均等配置の段落でも若干がたついて見える場合あり。 shrink 単語が行内に収まらないとき、収まるまでテキストを圧縮。ただし <i>shrinklimit</i> の制限に従い、それで収まりきらなければ <i>nofit</i> 方式を適用。 split 最後の単語を次行へ送らず、強制的にハイフネーションする。テキストフォントの場合はハイフンキャラクターを挿入し、記号フォントの場合はしない。 spread 最後の単語を次行へ送り、残された (短い) 行を均等配置するよう単語内の字間を広げる。ただし <i>spreadlimit</i> の制限に従い、それで均等配置できなければ <i>nofit</i> 方式を適用。
advanced-linebreak	(論理値) 複雑用字系で必要な高度な改行アルゴリズムを有効にします。これはタイ語など、単語境界を示すのに空白キャラクターを使わない用字系で改行を行うために必要です。オプション <i>locale</i> ・ <i>script</i> に従います。デフォルト: <i>false</i>
avoidbreak	(論理値) <i>true</i> なら、 <i>avoidbreak</i> が <i>false</i> に再設定されるまで一切改行しない。デフォルト: <i>false</i> 。

表 9.2 改行アルゴリズムを制御するためのオプション

オプション	説明
charclass	<p>(ペアのリスト。ここでペアの 1 番目の要素はキーワードであり、2 番目の要素は Unichar または Unichar のリスト) 指定された Unichar が、指定されたキーワードによって分類されることにより、そのキャラクター (群) の改行時動作を決定します:</p> <p>letter 文字 (a B など) と同様に動作</p> <p>punct 句読点文字 (+ / ; など) と同様に動作</p> <p>open 開きかっこ ([など) と同様に動作</p> <p>close 閉じかっこ (] など) と同様に動作</p> <p>default すべてのキャラクター分類を PDFlib 内蔵のデフォルトにリセット</p> <p>例: <code>charclass={ close » open « letter {/ : =} punct & }</code></p>
hyphenchar	<p>(Unichar またはキーワード) 改行箇所ソフトハイフンに置き換わるべきキャラクターの Unicode 値。デフォルト: U+00AD (SOFT HYPHEN)、ただしそれがフォント内になれば U+002D (HYPHEN-MINUS)。</p>
locale	<p>(キーワード) <code>advancedlinebreak= true</code> の場合に、ローカライズされた改行方式のために用いられるロケール。キーワードは 1 個ないし複数の構成要素から成っており、その中でオプションなコンポーネントは下線キャラクター「_」で区切られます (その文法は NLS/POSIX ロケール ID とは若干異なっています):</p> <ul style="list-style-type: none"> ▶ 必須の、ISO 639-2 に従った 2 文字か 3 文字の小文字の言語コード。例: en (英語)・de (ドイツ語)・ja (日本語)。これは <code>language</code> オプションとは異なります。 ▶ オプションな、ISO 15924 に従った 4 文字の用字系コード。例: Hira (ひらがな)・Hebr (ヘブライ)・Thai (タイ)。 ▶ オプションな、ISO 3166 に従った 2 文字の大文字の国コード。例: DE (ドイツ)・CH (スイス)・GB (イギリス) <p>キーワード <code>_none</code> は、ロケール固有の処理が一切行われなことを指定します。</p> <p>ロケールを指定することは、タイ語などいくつかの用字系では高度な改行のために必須です。デフォルト: <code>_none</code></p> <p>例: <code>Thai · de_DE · en_US · en_GB</code></p>
maxspacing minspacing	<p>(float またはパーセント値) 単語間の最大間隔・最小間隔 (ユーザー座標で表すか、スペースキャラクターの幅に対するパーセント値で指定)。単語間隔の算出時はこの値を限度とする (ただし <code>wordspacing</code> オプションが加算される)。デフォルト: <code>minspacing=50%</code>、<code>maxspacing=500%</code>。</p>
nofitlimit	<p>(float またはパーセント値) <code>nofit</code> 方式における行の長さの下限 (ユーザー座標で表すか、収めるはめ込み枠の幅に対するパーセント値で指定)。デフォルト: 75%。</p>
shrinklimit	<p>(パーセント値) <code>shrink</code> 方式におけるテキスト圧縮の下限。縮小率の算出時はこの値を限度とする。ただし、<code>horizscaling</code> オプションの値を掛け算される。デフォルト: 85%。</p>
spreadlimit	<p>(float またはパーセント値) <code>spread</code> 方式における 2 文字間の間隔の上限 (ユーザー座標で表すか、文字サイズに対するパーセント値で指定)。計算される文字間は <code>charspacing</code> オプションの値に加算される。デフォルト: 0。</p>

インラインオプションリストは通常は改行機会を生み出しません。それは単語内でのオプション変更を可能にするためです。ただし、オプションリストがスペースキャラクターで挟まれている場合はオプションリストの開始位置に改行機会があります。もしそのオプションリストで改行が起きてしかも `alignment=justify` の場合、オプションリストの前にあるスペース群は破棄されます。オプションリストの後のスペース群は保持され、次回先頭に表れます。

改行を防止 `charclass` オプションを使うと、テキストフローが特定のキャラクターの後に改行されるのを防止することができます。以下のオプションはキャラクター / の直後の改行を防止します：

```
charclass={letter /}
```

以下のオプションは、「20° C」のようなテキストの中の度キャラクター「°」と「C」の間の改行を防止します。度キャラクターが開き括弧のように扱われるので行末で孤立しません：

```
charclass={open °}
```

ひとつながりのテキストが複数行に泣き別れてしまうのを防ぐには、それを *avoidbreak* ～ *noavoidbreak* でくくるという方法があります。

クックブック 完全なコードサンプルがクックブックの `textflow/avoid_linebreaking` トピックにあります。

日中韓テキストの組版 テキストフローエンジンは、日中韓テキストを扱えるように作られているので、Unicode 標準の通り、日中韓キャラクターを表意文字として適切に取り扱います。その結果、日中韓テキストはけっしてハイフネーションされません。組版の品質を高めるため、日中韓テキストでテキストフローを使うときは、以下の組版オプションを推奨します。こうすると、欧文テキストが混在していてもそこでハイフネーションが行われなくなり、また、均等に間隔をあけたテキスト出力が作成されます：

```
hyphenchar=none  
alignment=justify  
shrinklimit=100%  
spreadlimit=100%
```

縦書きはテキストフローでは対応していません。

狭いはめ込み枠でテキスト両端揃え はめ込み枠が狭ければ狭いほど、両端揃えのテキストを制御するためのオプションが重要になっていきます。図 9.23 は、狭いはめ込み枠でテキストがさまざまな方式で両端揃えされた出力結果を例示しています。図 9.23 のオプション設定は基本的におおむね良好ですが、ただ、*maxspacing* がやや広すぎる単語間隔を与えているのが気になります。とはいえ、狭いはめ込み枠に対してはこれはこのままにしておくことを推奨します。でないと、*split* 方式による見苦しい強制ハイフネーションの発生頻度が高まるでしょう。

はめ込み枠が狭すぎるために不適切な箇所が強制ハイフネーションされてしまう場合は、対処を考慮して、ソフトハイフンを入れるなり、テキスト両端揃えを制御するオプションを変えるなりする必要があるでしょう。

テキスト両端揃えで `shrinklimit` オプション 見た目にもっとも受け入れやすい解決策は *shrinklimit* オプションを小さくすることでしょう。このオプションは、*shrink* 方式でかかる長体の割合の下限を指定するものです。図 9.24a は、テキストに *shrinklimit=50%* まで長体をかけることで強制ハイフネーションを防いでいる様子を示しています。

テキスト両端揃えで `spreadlimit` オプション 字間を拡げることで改行を制御するのも一つの方法です。これは *spread* 方式で行われ、*spreadlimit* オプションで制御されます。し

Our paper planes are the ideal way of passing the time. We offer revolutionary brand new developments of the traditional common paper planes. If your lesson, conference, or lecture turn out to be deady boring, you can have a wonderful time with our planes. All

図 9.23 狭いはめ込み枠でテキスト両端揃え。デフォルト設定を用いています

単語間ツメ (minspacing オプション)

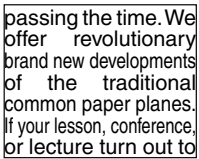
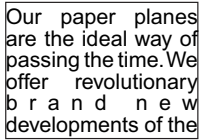
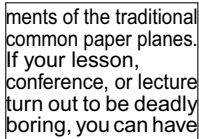
行に長体 (shrink 方式・shrinklimit オプション)

強制ハイフネーション (split 方式)

単語間アケ (spread 方式・maxspacing オプション)

しかしこの方式は美しくないのでめったに使われ不会いでしょう。図 9.24b は、`spreadlimit=5` を使って、字間の最大を非常に広く 5 単位とした例です。

図 9.24 狭いはめ込み枠内の両端揃えテキストのためのオプション

生成される出力	PDF <code>fit_textline()</code> に与えるオプションリスト
a) 	<code>alignment=justify shrinklimit=50%</code>
b) 	<code>alignment=justify spreadlimit=5</code>
c) 	<code>alignment=justify nofitlimit=50</code>

テキスト両端揃えで nofitlimit オプション `nofitlimit` オプションは、`nofit` 方式が適用されたときの行の最小の幅を制御するものです。はめ込み枠が非常に狭い場合は、これをデフォルト値 75% から下げたほうが強制ハイフネーションよりはましです。図 9.24c は、行の最小幅 50% を指定した場合の出力結果を示しています。

9.2.10 高度な用字系固有の改行

PDFlib は、標準の改行アルゴリズムに加えて、追加の改行アルゴリズムを実装しています。この高度な改行アルゴリズムは、いくつかの用字系では必須であり、また、必須でないその他の用字系 / ロケールの組み合わせのなかにも、これにより改行動作が改善されるものがあります。これは `advancedlinebreak` オプションで有効にすることができます。改行はテキストの言語に依存しますので、高度な改行アルゴリズムは `script` オプション (表 7.3 参照) と `locale` オプション (PDFlib API リファレンス参照) に従います。高度な改行は、以下の状況において正しい改行機会を決定します：

- ▶ タイ語など、テキスト内の空白キャラクターの存在に改行が依拠しない用字系に対して。以下のテキストフローオプションは、タイ語に対して高度な改行を有効にします：

```
<advancedlinebreak script=thai locale=tha>
```

- ▶ 仏文テキスト内で引用符として用いられる «» ギユメキャラクターなど、ある特定の句読点キャラクターの特別な扱いを必要とする用字系/ロケールの組み合わせにおいて。下記のテキストフローオプションは、仏文テキストに対して高度な改行を有効にします。その結果、単語を囲うギユメキャラクターが行末で単語と泣き別れしなくなります：

```
<advancedlinebreak script=latn locale=fr>
```

locale テキストフローオプションは *language* テキストオプション (表 7.4 参照) と違うことに留意してください：*locale* オプションは同じ 3 文字の言語識別子で始まることができますが、1 個ないし 2 個の追加部分を任意に含むこともできます。ただし、これらが PDFlib で必要になることはまれです。

9.2.11 テキストをパス・画像に回り込ませる

回り込み機能を利用すると、任意の形状にテキストを入れたり、テキストをパスに回り込ませることができます。範囲枠、明示的な長方形 / 多角形 / 円 / 曲線、パスオブジェクトのいずれかを用いて、テキストフローに対する回り込み領域を指定することができます。画像がクリッピングパスを内蔵している場合には、テキストをその画像のクリッピングパスに自動的に回り込ませることも可能です。

範囲枠を持つ画像にテキストを回り込ませる 最初の作成例として、テキストフローの中に画像を配置して、テキストをその画像全体のまわりに回り込ませてみましょう。まず画像を読み込み、枠内の希望の位置に配置します。この画像を後で名前でも参照するために、オプションリスト `matchbox={name=img margin=-5}` で、画像をはめ込む際に *img* という範囲枠を定義し、5 単位の余白を指定しましょう：

```
result = p.fit_image(image, 50, 35,
    "boxsize={80 46} fitmethod=meet position=center matchbox={name=img margin=-5}");
```

テキストフローを追加します。そしてそれを、以下のように、画像のはめ込み枠 *img* を回り込むべき領域として *wrap* オプションを使って配置しましょう (図 9.25 参照)：

```
result = p.fit_textflow(textflow, left_x, left_y, right_x, right_y,
    "wrap={usematchboxes={{img}}}");
```

図 9.25
範囲枠を持つ画像にテキストを回り込ませる

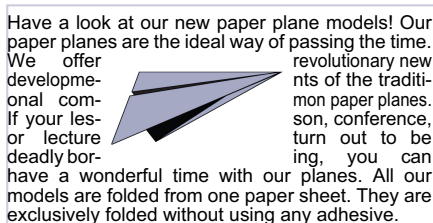
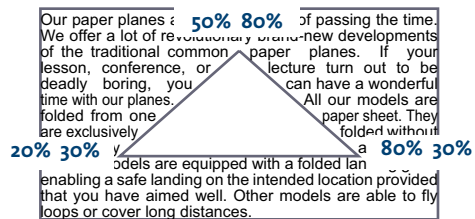


図 9.26
三角形の輪郭にテキストを回り込ませる



テキストを配置する前に、同じ範囲枠名を使ってさらにほかの画像をはめ込んでいくこともできます。その場合、テキストはすべての画像を回り込みます。

クックブック 完全なコードサンプルがクックブックの `textflow/wrap_text_around_images` トピックにあります。

任意のパスにテキストを回り込ませる パスオブジェクトを作成して (71 ページ「3.2.3 直接パス」参照)、それを回り込み形状として用いることもできます。下記のコード断片は、単純な形状 (1 個の円) のパスオブジェクトを構築し、それを `PDF_fit_textflow()` の `wrap` オプションに与えます。パスを配置する参照点は、はめ込み枠の幅と高さに対するパーセント値として表現されています：

```
path = p.add_path_point( -1,  0, 100, "move", "");
path = p.add_path_point(path, 200, 100, "control", "");
path = p.add_path_point(path,  0, 100, "circular", "");

result = p.fit_textflow(tf, llx1, lly1, urx1, ury1,
    "wrap={paths={" +
        "{path=" + path + " refpoint={100% 50%} }" +
    "}}");

p.delete_path(path);
```

inversefill オプションを用いると、テキストをパスの外側に回りこませるのではなく、パスの内側に回りこませることが可能です (すなわち、パスはテキストフロー内に穴を作るのではなく、テキストの入れ物として機能します)：

```
result = p.fit_textflow(tf, llx1, lly1, urx1, ury1,
    "wrap={inversefill paths={" +
        "{path=" + path + " refpoint={100% 50%} }" +
    "}}");
```

画像のクリッピングパスにテキストを回り込ませる TIFF・JPEG 画像は、クリッピングパスを内蔵することができます。このパスは、画像処理アプリケーションで作成されている必要があり、PDFlib はこれを評価します。デフォルトクリッピングパスが画像内で見つければそれが使われますが、画像内の他の任意のクリッピングパスを `PDF_load_image()` の `clippingpathname` オプションで指定することもできます。画像がクリッピングパスとともに読み込まれていれば、そのパスを抽出して先述のように `PDF_fit_textflow()` の `wrap` オプションに与えることができます。取り込んだ画像のクリッピングパスを拡大するために `scale` オプションも与えます：

```
image = p.load_image("auto", "image.tif", "clippingpathname={path 1}");

/* 画像のクリッピングパスからパスオブジェクトを作成 */
path = (int) p.info_image(image, "clippingpath", "");
if (path == -1)
    throw new Exception("エラー：クリッピングパスが見つかりません!");

result = p.fit_textflow(tf, llx1, lly1, urx1, ury1,
    "wrap={paths={{path=" + path + " refpoint={50% 50%} scale=2}}});

p.delete_path(path);
```


画像を配置してテキストをそれに回り込ませる 前の項では画像のクリッピングパスだけを使いました（画像自体ではなく）が、今度は画像をはめ込み枠の中に配置して、テキストをそれに回り込ませてみましょう。これを実現するには、今度も画像を `clippingpathname` オプションをつけて読み込んで、それを `PDF_fit_image()` でページ上に配置する必要があります。テキストを回り込ませるのに適切なパスオブジェクトを作成するために、`PDF_fit_image()` と同じオプションリストをつけて `PDF_info_image()` を呼び出しましょう。こうすれば、クリッピングパスが抽出される際と、画像が実際に配置される際に、必ず同一の変換（拡大縮小など）が適用されます。最後に、参照点 (`PDF_fit_image()` の `x/y` 引数) を `wrap` オプションの `paths` サブオプションの `refpoint` サブオプションに与える必要があります：

```
image = p.load_image("auto", "image.tif", "clippingpathname={path 1}");

/* 画像を何らかのはめ込みオプション群でページ上に配置 */
String imageoptlist = "scale=2";
p.fit_image(image, x, y, imageoptlist);

/* 同じオプションリストを用いて、画像からパスオブジェクトを作成 */
path = (int) p.info_image(image, "clippingpath", imageoptlist);
if (path == -1)
    throw new Exception("エラー：クリッピングパスが見つかりません!");

result = p.fit_textflow(tf, llx1, lly1, urx1, ury1,
    "wrap={paths={{path=" + path + " refpoint={" + x + " " + y + " }}}});

p.delete_path(path);
```

`PDF_fit_textflow()` を複数回呼び出して同一の `wrap` オプションを与えることもできます。これは多段組など、配置された画像が複数のテキストフローをはめ込み枠に重なっているときに有用です。

非長方形形状にテキストを回り込ませる パスオブジェクトを回りこみ輪郭として作成するのではなく、パス要素群を直接 `Textflow` オプション群で指定することもできます。

テキストは、範囲枠で指定される長方形を回り込ませるだけでなく、任意のグラフィック要素を回り込み輪郭として定義することもできます。たとえば以下のオプションリストは、三角形のまわりにテキストを回り込ませます（図 9.26 参照）：

```
wrap={ polygons={ {50% 80% 20% 30% 80% 30% 50% 80%} } }
```

なお、`showborder=true` オプションを使って輪郭を図示してあります。`wrap` オプションは複数の輪郭を持つこともできます。以下のオプションリストは、2つの三角形のまわりにテキストを回り込ませます：

```
wrap={ polygons={ {50% 80% 20% 30% 80% 30% 50% 80%}
    {20% 90% 10% 70% 30% 70% 20% 90%} } }
```

パーセント値（はめ込み枠内における相対座標）のかわりに、ページ上の絶対座標を使うこともできます。

注 横でも縦でもない向きの線分を持つ輪郭を使うときは、`fixedleading=true` に設定することを推奨します。

図 9.27
ひし形へテキスト
を流し込む

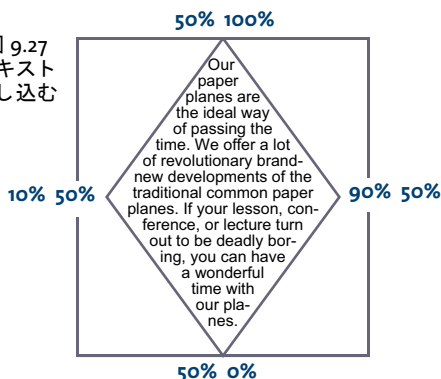
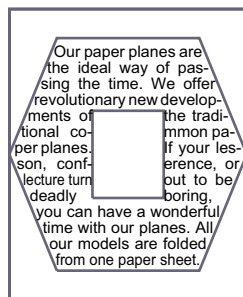


図 9.28
重なり合う輪郭へ流し込む



クックブック 完全なコードサンプルがクックブックの `textflow/wrap_text_around_polygons` トピックにあります。

非長方形の輪郭へ流し込む 回り込み機能を使えば、任意の輪郭の領域の中にテキストフローを配置することもできます。これを実現するには、`wrap` オプションで `addfitbox` または `inversefill` サブオプションを使います。テキストは指定された輪郭のまわりに回り込むのではなく、1 個ないし複数の輪郭の中にテキストが配置されます。以下のオプションリストを使えば、ひし形の中へテキストを流し込むことができます。ここで座標は、はめ込み枠の長方形に対するパーセント値として与えています (図 9.27 参照) :

```
wrap={ addfitbox polygons={ {50% 100% 10% 50% 50% 0% 90% 50% 50% 100%} } }
```

なお、ここでも `showborder=true` オプションを使って輪郭を図示してあります。もし `addfitbox` オプションをつけなければ、ひし形は空のまま、そのまわりにテキストが回り込むこととなります。

重なり合う輪郭へ流し込む 次の例として、重なり合う 2 つの多角形から成る輪郭へ流し込みを行なってみましょう。たとえば六角形の中に四角形が入った輪郭です。`addfitbox` オプションを使えば、はめ込み枠自体は流し込みの範囲から除外され、その後のリストの中の多角形は、重なり合っている領域を除いて流し込みが行われます (図 9.28 参照) :

```
wrap={ addfitbox polygons=
  { {20% 10% 80% 10% 100% 50% 80% 90% 20% 90% 0% 50% 20% 10%}
    {35% 35% 65% 35% 65% 65% 35% 65% 35% 35%} } }
```

もし `addfitbox` オプションをつけなければ、これと反対の効果を得ます : さっき流し込まれた領域は空のままとなり、さっき空だった領域へテキストが流し込まれることとなります。

クックブック 完全なコードサンプルがクックブックの `textflow/fill_polygons_with_text` トピックにあります。

9.3 表の組版

表組版機能を使うと、複雑な表を自動組版することができます。表のセルには、一行か複数行のテキストか、画像か、SVG グラフィックか、PDF ページを入れることができます。表は 1 個のはめ込み枠に収まらなくてもよく、複数のページにわたることが可能です。

クックブック 表の諸側面に関するコードサンプルがPDFlibクックブックのtableカテゴリーにあります。

表の一般的特徴 表組版機能の説明は、以下の概念と用語にもとづきます(図 9.29 参照) :

- ▶ **表**は、長方形の輪郭を持つ仮想のオブジェクトです。横方向の**表行**と縦方向の**列**でできています。
- ▶ **単純セル**は、表内の長方形の領域であり、表行と列の交差として定義されます。**連結セル**は、複数の列か複数の表行、ないし両方にわたっています。**セル**という用語を用いるときは、単純セルと連結セルの両方を指すものとします。
- ▶ 表は、1 個のはめ込み枠に収まりきることもありますし、複数のはめ込み枠が必要になることもあります。1 個のはめ込み枠に配置された表行群は、**表インスタンス**を構成します。`PDF_fit_table()` は、1 回呼び出されるごとに、1 個のはめ込み枠に 1 つの表インスタンスを配置します (269 ページ「9.3.5 表インスタンス」参照)。
- ▶ **ヘッダー**と**フッター**は、表の最初か最後にある 1 個ないし複数の表行のかたまりであり、すべての表インスタンスの上端か下端に繰り返し現れます。ヘッダーにもフッターにも属さない表行は**本体表行**と呼びます。
- ▶ オプションな**キャプション** (図 9.29 では示されていません) は、表の説明を配置するために使える追加の要素です。これは表の任意の辺に配置することができます。

The diagram shows a table with three main sections, each containing a table of material and benefit, and an image of a paper plane. Annotations point to various parts of the table structure:

- ヘッ**: Points to the top header row.
- 単純セル**: Points to a single cell in the first table.
- セルにテキストフロー**: Points to the text in a cell.
- フッ**: Points to the bottom footer row.
- 3列連結セル**: Points to a cell that spans three columns.
- セルに画像とテキスト行**: Points to a cell containing an image and text.
- 3行連結セル**: Points to a cell that spans three rows.

Our Paper Plane Models		
1 Giant Wing		
Material	Offset print paper 220g/sqm	Amazingly robust!
Benefit	It is amazingly robust and can even do aerobatics. But it is best suited to gliding.	
2 Long Distance Glider		
Material	Drawing paper 180g/sqm	3行連結セル
Benefit	With this paper rocket you can send all your messages even when sitting in the cinema pretty near the back.	
3 Cone Head Rocket		
Material	Kent paper 200g/sqm	With big swing!
Benefit	This paper arrow can be thrown with big swing. It stays in the air a long time.	

図 9.29
表の例

例として、図 9.29 の表を作成するすべての側面を説明していきます。表組版オプションの完全な説明については *PDFlib API リファレンス* を参照してください。表の作成はまず、各表セルの内容と視覚的のプロパティを `PDF_add_table_cell()` で定義していくことから始まります。それから、`PDF_fit_table()` を 1 回ないし複数回呼び出して、その表を配置します。

表を配置する際には、そのはめ込み枠の大きさと、その表行や列の罫線と塗り分けを指定することもできます。セルごとの塗り分けなどの細かい指定には、範囲枠機能を使ってください（詳しくは 275 ページ「9.4 範囲枠」参照）。

この節では、表セルの定義と表のはめ込みに必要な、もっとも重要なオプションのみを解説します。いずれの例においても、そのための `PDF_add_table_cell()` と `PDF_fit_table()` への呼び出しだけを示しますが、必要なフォントはすでに読み込まれているものとします。

注 表の処理は、カレント図形ステータスからは独立です。表セルの定義は文書スコープでもできますが、実際に表を配置するのはページかパターン/テンプレート/グリフスコープで行う必要があります。

クックブック 完全なコードサンプルがクックブックの `table/starter_table` トピックにあります。

表に関する組版上の問題を分析 セルの数と、表組版オプションによっては、PDFlib の表組版機能の結果が、自分の期待とは一致しないことがあります。ほとんどの場合において、これは適切なオプションによって矯正できます。しかし、問題の起きているセル群や、セルのグループを、誤ったオプションを用いて特定するのは、難しい場合もあります。表に関する組版上の問題のデバッグを実施可能にするため、PDFlib は、`PDF_fit_table()` に以下のオプションを提供しています：

- ▶ オプション `showcells` は、各セル内枠の境界を視覚化します。このメソッドがページスコープで呼ばれ、かつ PDF/A モードが有効でない場合には、各表セルの中央に、そのセル内容に関する詳細を持った注釈が配置されます。
- ▶ オプション `debugshow` が `true` の場合には、高すぎる、あるいは幅が広すぎる、あるいはセルが小さくなりすぎる表に関するエラーがすべて抑制され、ログ記録されます。結果として作成される表インスタンスは、その表は破損していますが、デバッグの助けとして生成されます。
- ▶ オプション `showgrid` が `true` の場合には、すべての列と表行の縦・横境界が描線されます。すなわち、基礎をなす表グリッドが視覚化されます。

9.3.1 単純な表を配置

表の概念をさらに詳しく説明する前に、単純な表作成の例を示します。表には 6 つのセルがあり、3 表行・2 列に配されています。4 つのセルにはテキスト行があり、1 つのセルには複数行のテキストフローがあります。セルの内容はすべて、余白を 1 単位として横方向に左寄せ、縦方向に中央揃えになっています。

この表を作成するために、まずはテキスト行セルに対するオプションリストを作るために、その `fittextline` サブオプションリストで、必要なオプション `font`・`fontsize` と位置 `{left center}` を定義しましょう。さらに、1 単位のセル余白を定義しましょう。そして、テキスト行セルを 1 つずつそれぞれの列・表行に追加し、その際に中身のテキストも、`PDF_add_table_cell()` への呼び出しに直接与えます。

次に、テキストフローを作成し、そのテキストフローのハンドルを使ってテキストフロー表セルに対するオプションリストを構築した後、そのセルを表に追加しましょう。

最後に、`PDF_fit_table()` を使って表を配置し、その際に、表の外枠とセルの各辺を黒い罫線で視覚化しましょう。列の幅は一切与えませんでしたので、与えたテキスト行と余白から自動的に計算されます。

クックブック 完全なコードサンプルがクックブックの `table/vertical_text_alignment` トピックにあります。

以下のコード断片は、この単純な表を作成する方法を示します。結果を図 9.30a に示します。

```
/* 複数行のテキストフローで表セルに入れるテキスト */
String tf_text = "It is amazingly robust and can even do aerobatics. " +
    "But it is best suited to gliding.";

/* 1列目と2列目の列幅を定義 */
int c1 = 80, c2 = 120;

/* 表インスタンスの左下隅と右上隅を定義 (はめ込み枠) */
double llx=100, lly=500, urx=300, ury=600;

/* エラー時には抜け出す */
p.set_option("errorpolicy=exception");

/* フォントを読み込む */
font = p.load_font("Helvetica", "unicode", "");

/* 1列目に配置するテキスト行セルに用いるオプションリストを定義 */
optlist = "fittextline={position={left center} font=" + font + " fontsize=8} margin==4" +
    " colwidth=" + c1;

/* 列1表行1にテキスト行セルを追加 */
tbl = p.add_table_cell(tbl, 1, 1, "Our Paper Planes", optlist);

/* 列1表行2にテキスト行セルを追加 */
tbl = p.add_table_cell(tbl, 1, 2, "Material", optlist);

/* 列1表行3にテキスト行セルを追加 */
tbl = p.add_table_cell(tbl, 1, 3, "Benefit", optlist);

/* 2列目に配置するテキスト行に用いるオプションリストを定義 */
optlist = "fittextline={position={left center} font=" + font + " fontsize=8} " +
    " colwidth=" + c2 + " margin=4";

/* 列2表行2にテキスト行セルを追加 */
tbl = p.add_table_cell(tbl, 2, 2, "Offset print paper 220g/sqm", optlist);

/* テキストフローを追加 */
optlist = "font=" + font + " fontsize=8 leading=110%";
tf = p.add_textflow(-1, tf_text, optlist);

/* 上で取得したハンドルを使ってテキストフローセルに用いるオプションリストを定義 */
optlist = "textflow=" + tf + " margin=4 colwidth=" + c2;

/* 列2表行3にテキストフロー表セルを追加 */
tbl = p.add_table_cell(tbl, 2, 3, "", optlist);

p.begin_page_ext(0, 0, "width=200 height=100");

/* 表をはめ込むためのオプションリストを表枠とセル罫線つきで定義 */
optlist = "stroke={{line=frame linewidth=0.8} {line=other linewidth=0.3}}";

/* 表インスタンスを配置 */
result = p.fit_table(tbl, llx, lly, urx, ury, optlist);
```

```

/* 結果をチェック。「_stop」はすべてOKを意味します */
if (!result.equals("_stop")) {
    if (result.equals( "_error"))
        throw new Exception("エラー : " + p.get_errmsg());
    else {
        /* それ以外の戻り値はすべて専用のコードで扱う必要があります */
    }
}
p.end_page_ext("");

/* これは、表内で使われたテキストフローハンドル群も一緒に削除します */
p.delete_table(tbl, "");

```

セルの内容の縦位置を調整 表セルの縦方向中央にさまざまな種類の内容を配置すると、それらは、その辺からのまちなちの距離に位置付けられます。図 9.30a において、4つのテキスト行セルは以下のオプションリストで配置されています：

```

optlist = "fittextline={position={left center} font=" + font +
          " fontsize=8} colwidth=80 margin=4";

```

テキストフローセルは特殊なオプションを一切使わずに追加されています。テキスト行を縦方向中央に配置したために、*Benefit* の行がテキストフローの分だけ下へずれます。

図 9.30 テキスト行とテキストフローを表セル内で整列させる

生成される出力		
a)	Our Paper Planes	
	Material	Offset print paper 220g/sqm
	Benefit	It is amazingly robust and can even do aerobatics. But it is best suited to gliding.
b)	Our Paper Planes	
	Material	Offset print paper 220g/sqm
	Benefit	It is amazingly robust and can even do aerobatics. But it is best suited to gliding.

図 9.30b に示したように、セルの辺からセルの中身までの縦間隔は、それがテキストフローであるかテキスト行であるかにかかわらず、すべて同じにしたいものです。これを実現するために、まずテキスト行のためのオプションリストを用意しましょう。表行の高さを固定値 14 ポイント、テキスト行の位置を左上で余白 4 ポイントとして定義しましょう。

さきにごったオプション *fontsize=8* は、文字の高さを正確には表しておらず、上下にいくらかあきができています。でも、大文字の高さはフォントの *capheight* 値で正確に表されます。ですので、*fontsize={capheight=6}* を用いれば、文字サイズが結果的にほぼ 8 ポイントになり、また (*margin=4* とあわせて) 高さの合計が 14 ポイントとなって *rowheight* オプションと対応します。ですので全体としては、テキスト行セルに対する *PDF_add_table_cell()* のオプションリストは次のようにしましょう：

```
/* テキスト行セルに用いるオプションリスト */
optlist = "fittextline={position={left top} font=" + font +
          " fontsize={capheight=6}} rowheight=14 colwidth=80 margin=4";
```

テキストフローの追加にあたっては、上記のテキスト行同様、`fontsize={capheight=6}` を用いれば、文字サイズが結果的にほぼ 8 ポイントになり、また (`margin=4` とあわせて) 高さの合計が 14 ポイントとなります：

```
/* テキストフローの追加に用いるオプションリスト */
optlist = "font=" + font + " fontsize={capheight=6} leading=110%";
```

さらに、テキスト *Benefit* のベースラインは、テキストフローの 1 行目に整列させたいものです。と同時に、テキスト *Benefit* のセル上端からの間隔は、テキスト *Material* と同じになるべきです。上端に余白を生じさせないために、テキストフローセルの追加にあたっては `fittextflow={firstlinedist=capheight}` を用いましょう。そしてテキスト行と同じく、余白 4 ポイントを追加しましょう。

```
/* テキストフローセルの追加に用いるオプションリスト */
optlist = "textflow=" + tf + " fittextflow={firstlinedist=capheight} "
          "colwidth=120 margin=4";
```

クックブック 完全なコードサンプルがクックブックの `table/vertical_text_alignment` トピックにあります。

9.3.2 表セルのさまざまな内容

`PDF_add_table_cell()` で表にセルを追加するときは、さまざまな種類のセル内容を指定することができます。表セルは、同時に複数の種類の内容を含むこともできます。罫線・塗りの追加も可能なほか、範囲枠を使って表セル内に追加の内容を配置することもできます。

たとえば紙飛行機の表には、図 9.31 に示す要素があります。

テキスト行	
テキスト行	
テキスト行	テキスト行
テキスト行	テキストフロー

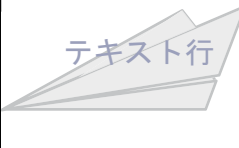


図 9.31
表セルのさまざまな内容

テキスト行による一行テキスト テキストは、`PDF_add_table_cell()` の `text` 引数で与えます。`fittextline` オプションリストで、`PDF_fit_textline()` のすべての組版オプションを与えることができます。デフォルトのはめ込み方式は `fitmethod=nofit` です。テキストがセルに収まりきらないときは、セルが大きくなります。これを避けるには、`fitmethod=auto` を使えば、`shrinklimit` オプションの範囲内でテキストが縮まります。表行の高さが指定されなかった場合は、組版機能はテキストサイズの 2 倍を表セルの高さとして (より正確にいうと : `boxheight` の 2 倍。これは、別途指定されない限りデフォルト値 `{capheight none}` を持ちます)。テキストが回転させてあるときの表行の幅についても同じです。

テキストに色をつけるには `fittextline` オプションリストで `fillcolor` オプションを使用してください。

テキストフローによる複数行テキスト テキストフローは、表関数の外で用意しておいて、`PDF_add_table_cell()` を呼び出す前に `PDF_create_textflow()` か `PDF_add_textflow()` で作成しておく必要があります。そのテキストフローのハンドルは、`textflow` オプションで与えます。`fittextflow` オプションで、`PDF_fit_textflow()` の組版オプションのすべてを与えることができます。

デフォルトのはめ込み方式は `fitmethod=clip` です。すなわち、まず、テキストがセルに収まりきるかどうかが試されます。セルの大きさが充分でないときは、その高さが増やされます。それでもテキストが収まりきらない場合は、末尾が切り落とされます。これを避けるには、`fitmethod=auto` を使えば、`minfontsize` オプションの範囲内でテキストが縮まります。

セルが狭すぎる場合には、1つの単語を好ましくない箇所まで分割させるよう、テキストフローに強制することもできます。`avoidwordsplitting` オプションが `true` の場合は、単語がもう分割されなくなるまでセル幅が広がります。

画像とテンプレート 画像は、`PDF_add_table_cell()` を呼び出す前に `PDF_load_image()` で読み込んでおく必要があります。テンプレートは、`PDF_begin_template_ext()` で作成する必要があります。その画像またはテンプレートのハンドルは、`image` オプションで与えます。`fitimage` オプションで、`PDF_fit_image()` の組版オプションのすべてを与えることができます。デフォルトのはめ込み方式は `fitmethod=meet` です。すなわち画像 / テンプレートが、縦横比を変えないまま、セル内に収まりきるよう配置されます。セルの大きさが、画像 / テンプレートの大きさにしたがって変わることはありません。

ベクトルグラフィック グラフィックは、`PDF_add_table_cell()` を呼び出す前に `PDF_load_graphics()` で読み込んでおく必要があります。そのグラフィックのハンドルは、`graphics` オプションで与えます。`fitgraphics` オプションで、`PDF_fit_graphics()` の組版オプションのすべてを与えることができます。デフォルトのはめ込み方式は `fitmethod=meet` です。すなわちグラフィックが、縦横比を変えないまま、セル内に収まりきるよう配置されます。セルの大きさが、グラフィックの大きさにしたがって変わることはありません。

取り込み PDF 文書のページ PDI ページは、`PDF_add_table_cell()` を呼び出す前に `PDF_open_pdi_page()` で開いておく必要があります。その PDI ページのハンドルは、`pdipage` オプションで与えます。`fitpdipage` オプションで、`PDF_fit_pdi_page()` の組版オプションのすべてを与えることができます。デフォルトのはめ込み方式は `fitmethod=meet` です。すなわち PDI ページが、縦横比を変えないまま、セル内に収まりきるよう配置されます。セルの大きさが、PDI ページの大きさにしたがって変わることはありません。

パスオブジェクト パスオブジェクトは、`PDF_add_table_cell()` を呼び出す前に `PDF_add_path_point()` で作成されている必要があります。そのパスハンドルは `path` オプションで与えられます。`fitpath` オプションでは、`PDF_draw_path()` のすべての組版オプションを指定可能です。パスの外接枠が表セル内に配置されます。セル内枠の左下隅が、パスを配置するための参照点として用いられます。

注釈 表セル内の注釈は、`PDF_create_annotation()` の `type` 引数（ただしこのメソッドを呼び出す必要はありません）に対応する `PDF_add_table_cell()` の `annotationtype` オプシ

ンで作成することができます。`fitannotation` オプションでは、`PDF_create_annotation()` のすべてのオプションを指定可能です。セル枠が注釈長方形として用いられます。

フォームフィールド 表セル内のフォームフィールドは、`PDF_create_field()` の `name・type` 引数 (ただしこのメソッドを呼び出す必要はありません) に対応する `PDF_add_table_cell()` の `fieldname・fieldtype` オプションで作成することができます。`fitfield` オプションでは、`PDF_create_field()` のすべてのオプションを指定可能です。セル枠がフィールド長方形として用いられます。

セル内枠内でセル内容を位置付け デフォルトでは、セル内容はセル枠に合わせて位置付けられます。`PDF_add_table_cell()` で `margin` オプションを使えば、セルの端との間に間隔を指定することができます。その結果できる長方形を、**セル内枠** と呼びます。余白が1つでも定義されていれば、セル内容はセル内枠に合わせて配置されます (図 9.32 参照)。余白が1つも定義されていないときは、セル内枠はセル枠と同じです。

これとあわせて、セルの内容は、内容依存のはめ込みオプションで与えたオプションにも従うことがあります。266 ページ「9.3.4 さまざまな種類の内容を持った表」で説明します。

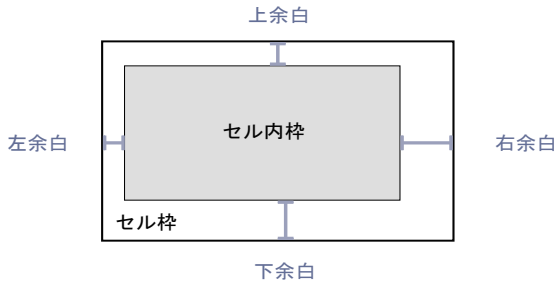


図 9.32
内容をセル内枠にはめ込み

9.3.3 表と列の幅

セルを表に追加する際には、そのセルがまたがる列か表行、または両方の数を、`colspan・rowspan` オプションで定義します。デフォルトではセルの列は1つ、表行も1つです。表の列と表行の総数は、セルを追加するごとに、それぞれの値だけ自動的に加算されます。図 9.33 に、3列・4表行の表の例を示します。

行1	3つの 列に わたる セル		
行2	2つの 列に わたる セル		3つの
行3	単純セル	単純セル 行に
行4	単純セル	単純セル わたるセル
	列1	列2	列3

図 9.33
単純セルと、複数の表行や列を連結したセル

さらに、`colwidth` オプションを使って、セルがまたがる最初の列の幅を明示的に与えることもできます。なお、与えた `colwidth` が効力を及ぼすのは最初の列に対してだけであり、`colspan` グループ内のすべての列に対してではないことに留意してください。各セルごと

に、その最初の列の幅を決めて与えると、それらの幅の値はすべて、表全体の幅に自動的に加算されていきます。図 9.34 に例を示します。

1 1 colspan=3 colwidth=50			3 2 rowspan=3 colwidth=90
1 2 colspan=2 colwidth=50			
1 3 colspan=1 colwidth=50	2 3 colspan=1 colwidth=100		
1 4 colspan=1 colwidth=50	2 4 colspan=1 colwidth=100		

50 100 90

————— 表の全幅 240 —————

図 9.34
列幅を足し合わせると表全体の幅に

または適当であれば、列幅をパーセント値で指定することもできます。その場合この値は、表のはめ込み枠の幅に対する割合になります。パーセント値による指定を行う場合は、すべての列に対して行う必要があります。でなければ一切してはいけません。

PDF_add_table_cell() の *colscalegroup* オプションを使って、いくつかの列を列伸縮グループとしてまとめてある場合、それらの幅は、グループ内でもっとも幅の広い列と同じになります (図 9.35 参照)。

列拡大縮小グループ

	Max. Load	Range	Weight	Speed
Giant Wing	12g	18m	14g	8m/s
Long Distance Glider	5g	30m	11.2g	5m/s
Cone Head Rocket	7g	7m	12.4g	6m/s

図 9.35
1 番目の表行の右 4 セルは、同じ列伸縮グループに属しているの、同じ幅になります。

絶対座標が使われている場合 (パーセント値でなく)、列幅を定義されていないセルがあるときは、その未決定の幅は以下のようにして算定されます: まず、テキスト行を含む各セルについて、列幅かテキストの幅 (回転されているテキストの場合はテキストの高さ) にもとづいて、実際の幅が算出されます。それから、残りの表幅が、まだ決定していない列幅に均等に分配されます。

9.3.4 さまざまな種類の内容を持った表

以下のいくつかの項では、図 9.36 に示すような、さまざまな種類の内容を持った表の例を、一歩ずつ作成していきましょう。

クックブック 完全なコードサンプルがクックブックの `table/mixed_table_contents` トピックにあります。

前準備として、2 つのフォントを読み込む必要があります。表のはめ込み枠の大きさを、その左下隅と右上隅の座標によって定義し、3 つの表列の幅を指定しましょう。そして、新規ページを A4 寸法で開始しましょう:

```
double llx = 100, lly = 500, urx = 360, ury = 600; // 表の座標
```

```
int c1 = 50, c2 = 120, c3 = 90; // 3つの表列の幅
```

```
boldfont = p.load_font("Helvetica-Bold", "unicode", "");
normalfont = p.load_font("Helvetica", "unicode", "");

p.begin_page_ext(0, 0, "width=a4.width height=a4.height");
```

手順 1：最初のセルを追加 まずは、表の最初のセルから始めましょう。このセルは 1 行目の 1 列目に配置し、3 つの列にわたらせませす。1 列目の幅は 50 ポイントです。テキスト行は縦横の中央に置き、すべての端で余白を 4 ポイントとります。以下のコード断片は、最初のセルを追加する方法を示します：

```
optlist = "fittextline={font=" + boldfont + " fontsize=12 position=center} " +
"fillcolor=red margin=4 colspan=3 colwidth=" + c1;
```

```
tbl = p.add_table_cell(tbl, 1, 1, "Our Paper Plane Models", optlist);
```

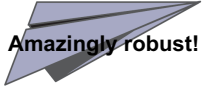
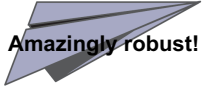
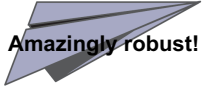
手順 2：2 列にまたがるセルを追加 次の手順として、テキスト行「1 *Giant Wing*」を持つセルを追加しましょう。これは 2 行目の 1 列目に配置し、2 つの列にわたらせませす。1 列目の幅は 50 ポイントです。行の高さは 14 ポイントです。テキスト行は左上に位置付け、すべての端で余白を 4 ポイントとります。262 ページ「セルの内容の縦位置を調整」で述べたのと同様に、`fontsize={capheight=6}` を用いて、テキストの縦揃えを統一しましょう。

この見出し「*Giant Wing*」のセルは、行全体でなく 3 列中 2 列しか連結しないので、行ベースの塗り分けオプションでは色がつけれられません。かわりに範囲枠機能を使って、セルが覆う長方形を灰色の背景色で塗りましょう。範囲枠機能について詳しくは 275 ページ「9.4 範囲枠」で説明しています。以下のコード断片は、見出し「*Giant Wing*」のセルを追加する方法を示します：

```
optlist = "fittextline={position={left top} font=" + boldfont +
" fontsize={capheight=6}} rowheight=14 colwidth=" + c1 +
" margin=4 colspan=2 matchbox={fillcolor={gray .92}}";
```

```
tbl = p.add_table_cell(tbl, 1, 2, "1 Giant Wing", optlist);
```

図 9.36 さまざまな内容の表セルを一步步追加

生成される表	生成手順										
<table border="1"> <tr> <td colspan="2" style="text-align: center;">Our Paper Plane Models</td> <td rowspan="3" style="text-align: center; vertical-align: middle;">  </td> </tr> <tr> <td colspan="2">1 Giant Wing</td> </tr> <tr> <td>Material</td> <td>Offset print paper 220g/sqm</td> </tr> <tr> <td>Benefit</td> <td>It is amazingly robust and can even do aerobatics. But it is best suited to gliding.</td> <td></td> </tr> </table>	Our Paper Plane Models			1 Giant Wing		Material	Offset print paper 220g/sqm	Benefit	It is amazingly robust and can even do aerobatics. But it is best suited to gliding.		<p>手順 1：3 列にまたがるセルを追加</p> <p>手順 2：2 列にまたがるセルを追加</p> <p>手順 3：さらにテキスト行セル 3 つ追加</p> <p>手順 4：テキストフローセル追加</p> <p>手順 5：テキスト行付き画像セル追加</p> <p>手順 6：表をはめ込む</p>
Our Paper Plane Models											
1 Giant Wing											
Material	Offset print paper 220g/sqm										
Benefit	It is amazingly robust and can even do aerobatics. But it is best suited to gliding.										

手順 3：さらに 3 つのテキスト行セルを追加 以下のコード断片は、「*Material*」・「*Benefit*」・「*Offset print paper*」…のセルを追加します。「*Offset print paper*」…のセルは 2 列目で始まるので、同時に 120 ポイントの列幅を定義します。セルの内容は左上に位置付け、すべての端で余白を 4 ポイントとります。

```
optlist = "fittextline={position={left top} font=" + normalfont +
" fontsize={capheight=6}} rowheight=14 colwidth=" + c1 + " margin=4";
```

```
tbl = p.add_table_cell(tbl, 1, 3, "Material", optlist);
tbl = p.add_table_cell(tbl, 1, 4, "Benefit", optlist);

optlist = "fittextline={position={left top} font=" + normalfont +
          " fontsize={capheight=6}} rowheight=14 colwidth=" + c2 + " margin=4";

tbl = p.add_table_cell(tbl, 2, 3, "Offset print paper 220g/sqm", optlist);
```

手順4：テキストフローセルを追加 以下のコード断片は、「*It is amazingly*」… のテキストフローセルを追加します。テキストフローの入った表セルを追加するには、まずテキストフローを作成しましょう。上記のテキスト行同様、`fontsize={capheight=6}` を用いれば、文字サイズが結果的にほぼ 8 ポイントになり、また (`margin=4` とあわせて) 高さの合計が 14 ポイントとなります。

```
tftext = "It is amazingly robust and can even do aerobatics. " +
         "But it is best suited to gliding.";

optlist = "font=" + normalfont + " fontsize={capheight=6} leading=110%";

tf = p.add_textflow(-1, tftext, optlist);
```

取得したテキストフローハンドルは、表セルを追加する時に使います。テキストフローの 1 行目は、テキスト行「*Benefit*」のベースラインと揃っているべきです。と同時に、テキスト「*Benefit*」は、そのセル上端からの間隔がテキスト「*Material*」と同じになるべきです。テキストフローを追加する際は、上に余白が生じないように、`fittextflow={firstlinedist=capheight}` を用います。そしてテキスト行と同じく、余白を 4 ポイント加えます：

```
optlist = "textflow=" + tf + " fittextflow={firstlinedist=capheight} " +
          " colwidth=" + c2 + " margin=4";

tbl = p.add_table_cell(tbl, 2, 4, "", optlist);
```

手順5：テキスト行の入った画像セルを追加 5 番目の手順として、Giant Wing 紙飛行機の画像とテキスト行「*Amazingly robust!*」の入ったセルを追加しましょう。このセルは 2 行目の 3 列目で始まり、3 つの行にまたがります。列幅は 90 ポイントです。セルの余白は 4 ポイントに設定します。1 つ目の例としては TIFF 画像をセル内に配置してみましょう：

```
image = p.load_image("auto", "kraxi_logo.tif", "");

optlist = "fittextline={font=" + boldfont + " fontsize=8} image=" + image +
          " colwidth=" + c3 + " rowspan=3 margin=4";

tbl = p.add_table_cell(tbl, 3, 2, "Amazingly robust!", optlist);
```

あるいは、画像は PDF ページとして取り込むこともできます。PDI ページを閉じるのは必ず、`PDF_fit_table()` を呼び出した後にしてください：

```
int doc = p.open_pdi("kraxi_logo.pdf", "", 0);

page = p.open_pdi_page(doc, pageno, "");

optlist = "fittextline={font=" + boldfont + " fontsize=9} pdipage=" + page +
```

```
" colwidth=" + c3 + " rowspan=3 margin=4";
```

```
tbl = p.add_table_cell(tbl, 3, 2, "Amazingly robust!", optlist);
```

手順 6 : 表をはめ込む 最後の手順として、表を `PDF_fit_table()` で配置しましょう。`header=1` を用いると、1 行目が表のヘッダーになります。`fill` オプションと `area=header · fillcolor={rgb 0.8 0.8 0.87}` サブオプションは、与えた色でヘッダー行を塗るよう指定しています。`stroke` オプションと `line=frame linewidth=0.8` サブオプションを用いて、表の外枠の線幅を 0.8 として定義しましょう。`line=other linewidth=0.3` を用いると、すべてのセルの罫が線幅 0.3 として指定されます：

```
optlist = "header=1 fill={{area=header fillcolor={rgb 0.8 0.8 0.87}}} " +
"stroke={{line=frame linewidth=0.8} {line=other linewidth=0.3}}";
```

```
result = p.fit_table(tbl, llx, lly, urx, ury, optlist);
```

```
if (result.equals("_error"))
    throw new Exception("エラー : " + p.get_errmsg());
```

```
p.end_page_ext("");
```

9.3.5 表インスタンス

1 つのはめ込み枠に配置された表行群は、表インスタンスを構成します。表全体を表現するには、複数の表インスタンスが必要なこともあります。`PDF_fit_table()` は、1 回呼び出されるごとに、1 つのはめ込み枠に 1 つの表インスタンスを配置します。これらのはめ込み枠は、同じページに多段組レイアウトなどで配置しておくことも、または複数のページに配置しておくこともできます。

図 9.37 の表は、3 つのページにわたっています。各ページに 1 つずつあるはめ込み枠に、各表インスタンスが 1 つずつ配置されます。`PDF_fit_table()` を呼び出すたびに、最初の行はヘッダーとして定義され、最後の行はフッターとして定義されます。

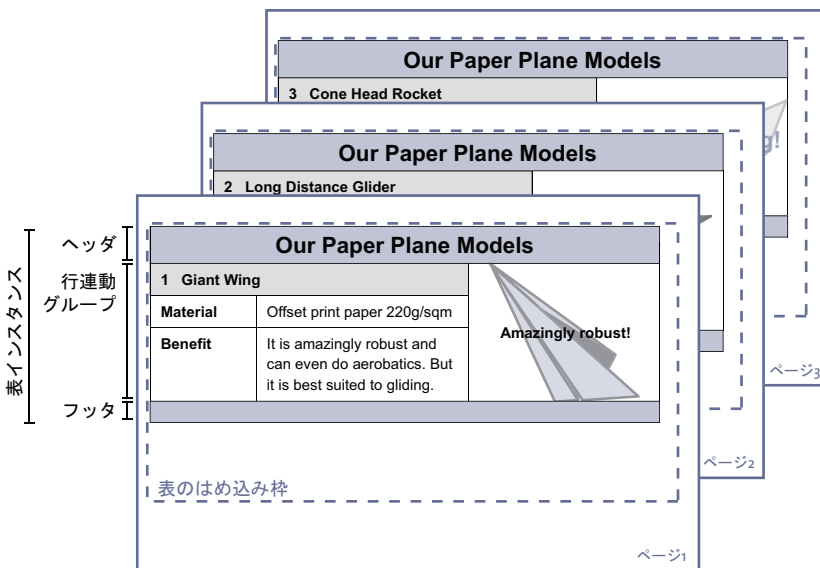


図 9.37 表は複数の表インスタンスに分解され、各はめ込み枠に 1 つずつ配置されます。

以下のコード断片は、表を配置しきるまで表インスタンスをはめ込みつづけるための、一般的なループを示します。配置するべき表インスタンスがある限り、そのつど新規ページを作成します。

```
do {
    /* 新規ページを作成 */
    p.begin_page_ext(0, 0, "width=a4.width height=a4.height");

    /* 最初の行をヘッダーとして使い、すべての表セルに線をひく */
    optlist = "header=1 stroke={{line=other}}";

    /* 表インスタンスを配置 */
    result = p.fit_table(tbl, llx, lly, urx, ury, optlist);
    if (result.equals("_error"))
        throw new Exception("エラー : " + p.get_errmsg());

    p.end_page_ext("");
} while (result.equals("_boxfull"));

/* 結果をチェック。「_stop」はすべてOKを意味します */
if (!result.equals("_stop")) {
    if (result.equals(" _error"))
        throw new Exception("エラー : " + p.get_errmsg());
    else {
        /* これ以外の戻り値はすべて「return」オプションによるユーザー終了。
        * これを扱うには専用のコードが必要です。*/
        throw new Exception ("テキストフロー内でユーザーリターンを検出しました");
    }
}
/* 表内で使ったテキストフローハンドルも削除されます */
p.delete_table(tbl, "");
```

ヘッダー・フッター `PDF_fit_table()` で `header`・`footer` オプションを使えば、表の最初か最後の行の数を定義して、それが各表インスタンスの上端か下端に配置されるようにすることができます。`fill` オプションで `area=header` か `area=footer` を使うと、ヘッダー・フッターを別の色で塗ることができます。ヘッダー行群は表定義の最初の n 行から成り、フッター行群は最後の m 行から成っています。

ヘッダーとフッターは、`PDF_fit_table()` で表インスタンスごとに指定します。結果として、表インスタンスごとに異なるものにもなりえます：ヘッダー/フッターをつけた表インスタンスと省いた表インスタンスを混在させる、といったことも可能です。それによりたとえば、最後の表インスタンスで特別な行を指定する、といったことも可能になります。

表行の連動 いくつかの表行を必ず同じ表インスタンスに入れさせたいときは、`rowjoingroup` オプションを使って、それらを同じ表行連動グループに割り当てることができます。表行連動グループは、連続する複数の表行を持ちます。このグループの表行は、複数の表インスタンスに別れさせられることがなくなります。

セルで複数表行を連結しても、それらの表行は自動的に連動グループにはなりません。

セルを分割 セルが行を連結している場合、後のほうの行がはめ込み枠に収まらないときは、そのセルは分割されます。画像・PDI ページ・SVG グラフィック・テキスト行セルの

場合は、セル内容は次の表インスタンスでも繰り返されます。テキストフローセルの場合は、セル内容は後の表行のセルに続きます。

図 9.38 に、テキストフローセルが分割されてテキストフローが次の表行に続いている様子を示します。図 9.39 に、画像セルが次の表インスタンスの最初の行で繰り返される様子を示します。

表インスタンス1	1 Giant Wing		Our paper planes are the ideal way of passing the time. We offer revolutionary
	Material	Offset print paper 220g/sqm	
表インスタンス2	Benefit	It is amazingly robust and can even do aerobatics. But it is best suited to gliding.	new developments of the traditional common paper planes.

図 9.38
セルの分割

表行を分割 表のはめ込み枠に最後の本体行が収まりきらないとき、それは普通は分割されません。この動作は `PDF_fit_table()` の `minrowheight` オプションで制御され、デフォルト値は 100% です。このデフォルト設定では、表行は分割されず、まるごと次の表インスタンスへ配置されます。

`minrowheight` 値を減らせば、最後の本体行が分割され、表行の内容のうち指定した割合が 1 つ目の部分に、残りが次の部分に配置されます。

図 9.39 に、テキストフロー「*It's amazingly robust*」… が分割され、次の表インスタンスの最初の本体行へテキストフローが続く様子を示します。複数行にわたる画像セルが分割され、画像は繰り返されます。「Benefit」セルも繰り返されます。

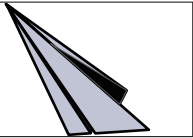

表インスタンス1	1 Giant Wing		
	Material	Offset print paper 220g/sqm	
	Benefit	It is amazingly robust and can even do aerobatics. But	
表インスタンス2	Benefit	it is best suited to gliding.	

図 9.39
表行の分割

9.3.6 表組版のアルゴリズム

この項では、表組版機能が表を配置する際に行うステップを詳説します。以下、横書きテキストの場合について述べます。しかし、「表行高さ」と「列幅」という言葉を互いに入れ替えれば、縦書きや回転テキストにもあてはまります。

`PDF_fit_table()` への最初の呼び出しの際には、オプション `colwidth`・`rowheight`・`fittextline`・`fittextflow` がすべてのセルについて吟味され、表全体の幅と高さが、列幅、表行高さ、テキスト行・テキストフロー内容、先頭はめ込み枠の幅に基づいて算出されます。はめ込み枠の高さは無限と見なされます。先頭表インスタンス（すなわち先頭はめ込み枠内の表先頭部分の配置）は `PDF_fit_table()` の `fitmethod` オプションに従って算出されます。

テキスト行を持つ表セルの高さと幅を算出 表組版機能はまず、テキスト行を持つ表セルのうち、`colwidth` または `rowheight` のない表列または表行にわたるものの寸法を決定します。これを実現するために、`fittextline` オプションに従ってテキスト行の、ひいては表

セルの幅を算出します。テキストサイズの2倍を表セルの高さと見なします（より正確にいうと：枠高さの2倍。これは、別途指定されない限りデフォルト値 `{capheight none}` を持ちます）。縦書きテキストについては、もっとも幅の広いキャラクターの幅がセル幅として用いられます。西向きまたは東向きのテキストについては、テキスト高さの2倍がセル幅として用いられます。

その後、この表セルの残余の幅と高さが、その表セルがわたる表列または表行のうち `colwidth` または `rowheight` が指定されていないものすべてに按分されます。

仮の表寸法を算出 次のステップとして組版機能は、表の仮の幅と高さを、それぞれ列幅・表行高さすべての合計として算出します。パーセント値で指定されている列幅と表行高さは、先頭はめ込み枠の幅と高さに基づいて絶対値へ変換されます。`colwidth` または `rowheight` を持たない列または表行がまだある場合には、仮の表寸法が先頭はめ込み枠に等しくなるまで、残りの余白が均等に按分されます。

`rowheightdefault` オプションを用いて、はめ込み枠の高さを完全に満たすか（キーワード `auto` と `distribute`）、それとも広さを節約するか（キーワード `minimum`）を指定することもできます。`rowheight` オプションで明示的に表行の高さを指定すると `rowheightdefault` はオーバーライドされます。

小さすぎるセルを拡大 ここで組版機能はすべてのセル内枠を決定します（図9.32参照）。余白の合計がセルの幅または高さより小さいときは、そのセル枠は、そのセルに属するすべての列と表行を均等に拡大することによって適切に拡大されます。

テキスト行の横方向をはめ込む 組版機能は、テキスト行を持つすべてのセルの幅を上げて、テキスト行が文字サイズを下げなくてもセルにはめ込めるようにします。これが可能でないときは、テキスト行は自動的に `fitmethod=auto` で配置されます。これによって、テキスト行がセル内枠からはみ出さないことが保証されます。`fittextline` オプションで `fitmethod=auto` に設定すると、セル幅が広がるのを防ぐことができます。

`colscalegroup` オプションを用いると、同一の列伸縮グループに属するすべての列が必ず等しい幅に伸縮されるように、すなわちこれらの幅が統一されて、グループ内で最も広い幅に合わせられるようにすることができます（図9.35参照）。

強制ハイフネーションを避ける 算出された表幅がはめ込み枠より小さいときは、組版機能はテキストフローセルの幅を上げて、テキストが強制ハイフネーションなしにはめ込めるようにします。これはオプション `avoidwordsplitting=false` で回避することもできます。このようなセルの幅は、表幅がはめ込み枠の幅に等しくなるまで上げられます。

`PDF_info_table()` の `horboxgap` キーを用いて、表幅とはめ込み枠幅の差をクエリーできます。

テキストの縦方向をはめ込む 組版機能は、すべてのテキスト行・テキストフローセルの高さを上げて、テキスト行またはテキストフローが文字サイズを下げずにセル内枠にはめ込めるよう試みます。ただし、テキスト行またはテキストフローに対してサブオプション `fitmethod=auto` が設定されている場合、またはテキストフローが `continuetextflow` オプションで他のセルに続いている場合には、セル高さは拡がりません。

このセル高さを上げる処理は、テキスト行またはテキストフローを内容として持つセルに対してのみ適用され、それ以外の種類のセル内容、すなわち画像・グラフィック・PDI ページ・パスオブジェクト・注釈・フィールドに対しては適用されません。

`rowscalegroup` オプションを用いると、同一の表行伸縮グループの中のすべての表行が必ず等しい高さに伸縮されるようにすることができます。

表を次のはめ込み枠へ続ける 算出された表全体の高さがはめ込み枠よりも大きい（すなわち、すべての表セルをはめ込み枠に収めることができない）ときは、組版機能は、そのはめ込み枠に収まらない初めての表行に出会う前に、そのはめ込み枠内に表行を配置することを止めます。

1つのセルが複数の表行にわたり、かつ、そのすべてがはめ込み枠に収まるわけではないときには、このセルは分割されます。セルが画像・PDI ページ・SVG グラフィック・パスオブジェクト・注釈・フォームフィールドを内容として持つときは、`repeatcontent=false` が指定されていない限り、そのセル内容は次のはめ込み枠内で繰り返されます。しかしテキストフローは、セルがわたる後続の表行に続きます（図 9.38 参照）。

rowjoingroup オプションを用いると、1つの連動グループの中のすべての表行が1つのはめ込み枠内に現れるようにすることができます。ヘッダーまたはフッターに属するすべての表行と1つの本体表行は、自動的に表行連動グループを形成します。ですので組版機能は、はめ込み枠に収まらない初めての表行に出会う前に表行を配置することを止めます（図 9.37 参照）。

return オプションを用いると、対象表行を配置した後にその表インスタンス内に絶対もう表行が配置されないようにすることができます。

表行を分割 表行は、非常に高いとき、または1個の本体表行しかないときには、分割されることがあります。末尾の本体表行がはめ込み枠に完全に収まらないときは、それは次のはめ込み枠へ移動されます。この動作は `PDF_fit_table()` の `minrowheight` オプションで制御することができます。このオプションのデフォルト値は100%です。この `minrowheight` 値を小さくすると、末尾本体表行の内容のうち指定した割合がカレントはめ込み枠に配置され、その表行の残りは次のはめ込み枠に配置されます（図 9.39 参照）。

`PDF_info_table()` の `rowsplit` キーワードを用いると、表行が分割されているかどうかをチェックすることができます。

表をはめ込み枠の幅に縮める 表列幅を合計した表幅は、はめ込み枠よりも大きくなる場合があります。たとえばテキスト行の横方向をはめ込んだ後などです。この場合には、表幅がはめ込み枠の幅に等しくなるまで、すべての列が均等に縮められます。この縮小処理は `horshrinklimit` オプションによって制限されます。一切の横縮小を避けるには `horshrinklimit=100%` を用います。

`PDF_info_table()` の `horshrink` キーワードを用いると横縮小率をクエリーできます。

`horshrinklimit` の閾値を超えたとき、すなわち、表を横に過度に縮めなければならない場合には、以下のエラーメッセージが現れます：

```
Table width $1 exceeds fitbox width $2 (required shrinking factor $3 is smaller than 'horshrinklimit')
```

ここで \$1 は算出された表幅を示し、\$2 は `PDF_fit_table()` で与えられたはめ込み枠の幅、\$3 は算出された縮小率です。このエラーが出た場合には、はめ込み枠の幅にもっと大きい値を与える必要があります。

表をはめ込み枠の高さに縮める ヘッダー・フッター行に少なくとも1つの表本体行または表行連動グループを加えた表の高さが、はめ込み枠の高さより大きくなることもあります。この場合には、表の高さがはめ込み枠の高さと等しくなるまですべての表行が均等に低くされます。この縮小過程は `vertshrinklimit` オプションによって制限されます。一切の縦縮小を避けるには `vertshrinklimit=100%` を用います。

`PDF_info_table()` の `vertshrink` キーワードを用いると縦縮小率をクエリーできます。

vertshrinking の閾値を超えたとき、すなわち、表を縦に過度に縮めなければならない場合には、以下のエラーメッセージが現れます：

```
Table height $1 exceeds fitbox height $2 (required shrinking factor $3 is smaller than 'vertshrinklimit')
```

ここで \$1 は算出された表高さを示し、\$2 は *PDF_fit_table()* で与えられたはめ込み枠の高さ、\$3 は算出された縮小率です。このエラーが出た場合には、はめ込み枠の高さにもっと大きい値を与える必要があります。

後続するはめ込み枠の幅が異なる場合 後続するはめ込み枠が最初のはめ込み枠より広い場合、表セルはそのはめ込み枠の幅を完全に満たすように広がるわけではありません。結果として、最初のはめ込み枠よりも広いはめ込み枠群を持つ表インスタンスはすべて、デフォルトでは等しい幅を持ちます。はめ込み枠を完全に満たすには、オプション *fitmethod=auto* を与える必要があります。*fitmethod=meet* を用いると、セル高さがそのアスペクト比を温存するよう適切に拡大されます。

最初のはめ込み枠よりも後続のはめ込み枠のほうが狭い場合には、表セルの幅は、狭いはめ込み枠に収まるよう縮まります。表幅を改めて計算しなおすには、*PDF_fit_table()* を *rewind=1* をつけて呼び出します。

9.4 範囲枠

範囲枠を使うと、PDFlib がページ上に何らかの内容を配置したときに算出した座標を利用することができます。範囲枠を定義するには、そのための専用の API メソッドがあるわけではなく、実内容を配置する `PDF_fit_textline()` や `PDF_fit_image()` などのメソッドで `matchbox` オプションを指定します。範囲枠はさまざまな目的に使えます：

- ▶ 範囲枠は装飾できます。例：色を塗る、枠線で囲む。
- ▶ 範囲枠を使って、1個ないし複数の注釈を `PDF_create_annotation()` で自動的に作成できます。
- ▶ 範囲枠はテキスト行の高さを定義して、それが `PDF_fit_textline()` で枠にはめ込まれるようにしたり、テキストフロー内のテキスト範囲の高さを定義して、それが装飾されるようにしたりします (`boxheight` オプション)。
- ▶ 範囲枠は画像の切り抜き領域を定義します。
- ▶ 範囲枠の座標やその他のプロパティは、`PDF_info_matchbox()` でクエリーして、他の作業に利用することができます。例：画像を挿入。

PDFlib はそれぞれの要素について、ページ上におけるその要素の位置（関連するすべてのオプションで指定された）を記述する外接枠に対応する長方形として、範囲枠を算出します。テキストフローと表セルの場合は、改行や表行分割によって、1つの範囲枠が複数の長方形から成ることもあります。

範囲枠の長方形（群）は、配置する要素を描く前に描かれます。そのため、範囲枠の罫や塗りの効力は実内容では打ち消されることがありますが、その逆はありません。特に、範囲枠の中で、画像で覆われた領域と重なる部分は、画像に隠れます。画像が `fitmethod=slice` または `fitmethod=clip` で配置されているときは、画像のはめ込み枠の外の範囲枠罫も切り落とされます。この現象を避けるには、範囲枠の長方形を、`PDF_fit_image()` を呼び出した後に、`PDF_rect()` など基本的な描画関数を使って描くという方法もあります。範囲枠の長方形の座標は、`PDF_info_matchbox()` を使って取得できますが、ただしこれはその範囲枠が `PDF_fit_image()` への呼び出しで名前を与えられていた場合に限りです。

以下のいくつかの項で、範囲枠の利用例をいくつか示します。範囲枠のオプションリストに対応している関数については、くわしくは [PDFlib API リファレンス](#) を参照してください。

注 範囲枠は `blind` モードにおいては、すなわち `blind` オプションを用いて組版する際には、サポートされていません。

9.4.1 テキスト行を装飾

テキスト行の範囲枠から話を始めましょう。`PDF_fit_textline()` においては範囲枠は、与えられたテキストのテキスト枠と同じになります。デフォルトでは、テキスト枠の幅はテキストの幅に等しく、高さは、与えられた文字サイズのキャップハイトに等しくなります。範囲枠の大きさを図示するために、以下のコード断片は、範囲枠を青の背景色で塗ります（図 9.40a 参照）。

```
String optlist =  
    "font=" + normalfont + " fontsize=8 position={left top} " +  
    "matchbox={fillcolor={rgb 0.8 0.8 0.8} boxheight={capheight none}}";
```

```
p.fit_textline("Giant Wing Paper Plane", 2, 20, optlist);
```

`boxheight` オプションは、`boxheight={capheight none}` がデフォルト設定なので、省略してもかまいません。`boxheight` オプションで枠の高さをもっと増やしてアセンダーまで覆うようにすれば、より美しくなります (図 9.40b 参照)。

枠の高さを増やして文字サイズに一致させたいときは、`boxheight={fontsize descender}` が使えます (図 9.40c 参照)。

次のステップとしては、範囲枠の左・右・下へ変位を加えて広げ、枠のすべての端をテキストと均等な間隔にしてみましょう。さらに枠線幅を指定して、範囲枠のまわりに長方形を描きましょう (図 9.40d 参照)。

クックブック 完全なコードサンプルがクックブックの `textflow/text_on_color` トピックにあります。

図 9.40 ささまざまなサブオプションの範囲枠を使ってテキスト行を装飾

生成される出力	PDF_fit_textline() の matchbox オプションのサブオプション
a) <code>Giant Wing Paper Plane</code>	<code>boxheight={capheight none}</code>
b) <code>Giant Wing Paper Plane</code>	<code>boxheight={ascender descender}</code>
c) <code>Giant Wing Paper Plane</code>	<code>boxheight={fontsize descender}</code>
d) <code>Giant Wing Paper Plane</code>	<code>boxheight={fontsize descender} borderwidth=0.3 offsetleft=-2 offsetright=2 offsetbottom=-2</code>

9.4.2 テキストフロー内で範囲枠を用いる

テキストフローの一部を装飾 この項では、テキストフロー内の一部のテキストを装飾しましょう。「*very dangerous*」という言葉にマーカーペンのように目立たせましょう。これを実現するには、言葉を `matchbox` インラインオプションと `matchbox=end` インラインオプションで囲みます (図 9.41 参照)：

図 9.41 `matchbox` インラインオプションを含んだテキストフロー

生成される出力	PDF_create_textflow() に対するテキストとインラインオプション
It is <code>very dangerous</code> to fly the Giant Wing in a thunderstorm.	It is <matchbox={fillcolor=red boxheight={ascender descender}}>very dangerous <matchbox=end> to fly the Giant Wing in a thunderstorm.

テキストフローの範囲枠に Web リンクを追加 今度は、テキストフローの一部に Web リンクを追加しましょう。1 番目の手順として、リンクをつけたい部分のテキストを示す `kraxi` という範囲枠を含んだテキストフローを作成しましょう。2 番目に、URL を開くアクションを作成しましょう。3 番目に、枠が不可視の `Link` 型の注釈を作成しましょう。そのオプションリストの中で、範囲枠 `kraxi` を参照してリンクの長方形として使いましょう (範囲枠を指定すると、`PDF_create_annotation()` の長方形の座標は無視されます)。

クックブック 完全なコードサンプルがクックブックの `textflow/weblink_in_text` トピックにあります。

```
/* 範囲枠「kraxi」を含んだテキストフローを作成してはめ込み */
String tftext =
    "For more information about the Giant Wing Paper Plane see the Web site of " +
```

```

        "<underline=true matchbox={name=kraxi boxheight={fontsize descender}}>" +
        "Kraxi Systems, Inc.<matchbox=end underline=false>";

String optlist = "font=" + normalfont + " fontsize=8 leading=110%";
tflow = p.create_textflow(tftext, optlist);
if (tflow == -1)
    throw new Exception("エラー : " + p.get_errmsg());

result = p.fit_textflow(tflow, 0, 0, 50, 70, "fitmethod=auto");
if (!result.equals("_stop"))
    { /* ... */ }

/* URIアクションを作成 */
optlist = "url={http://www.kraxi.com}";
act = p.create_action("URI", optlist);

/* 範囲枠「kraxi」上にLink注釈を作成 */
optlist = "action={activate " + act + "}" linewidth=0 usematchbox={kraxi}";
p.create_annotation(0, 0, 0, 0, "Link", optlist);

```

テキスト *Kraxi Systems, Inc.* が複数の行にわたる場合には、1回 `PDF.create_annotation()` を呼び出すだけで、適切な数のアクティブな長方形がリンク注釈に対して自動的に作成されます (図 9.42 参照)。

For information about
Giant Wing Paper
Planes see the Web
site of Kraxi Systems,
Inc.



<http://www.kraxi.com>

図 9.42
テキストフローの一部分に
Web リンクを追加

9.4.3 範囲枠と画像

画像にリンクを追加 画像で覆われた領域に Web リンクを追加するには、画像の範囲枠が使えます。コードは先述の 276 ページ「テキストフローの範囲枠に Web リンクを追加」と同じです。ただし、テキストフローを配置するのではなく、画像を以下のオプションリストを使ってはめ込みます：

```

String optlist = "boxsize={130 130} fitmethod=meet matchbox={name=kraxi}";
p.fit_image(image, 10, 10, optlist);

```


クックブック 完全なコードサンプルがクックブックの `interactive/link_annotations` トピックにあります。

画像にふちをつける この例では、画像の範囲枠を使って、画像のまわりにふちをつけましょう。画像に `fitmethod=meet` を使って、縦横比を保ちつつまるごと、与えられた枠に収めましょう。`borderwidth` サブオプションを使った `matchbox` オプションを使って、画像のまわりに太い枠を描きましょう。`strokecolor` サブオプションで枠の色を決め、`linecap·linejoin` サブオプションを使って角を丸めます。

クックブック 完全なコードサンプルがクックブックの `images/frame_around_image` トピックにあります。

範囲枠はつねに画像より前の時点で描かれるので、一部が画像で隠されてしまいます。これを避けるために、枠の幅の 50 パーセントの *offset* サブオプション群を使って、画像が覆う領域よりもふちを大きくしましょう。あるいは、枠をその分太くするという方法もあります。図 9.43 に、ふちをつけるために *PDF_fit_image()* で使うオプションリストを示します。

図 9.43 画像の範囲枠を使って画像にふちをつける

生成される出力	<i>PDF_fit_image()</i> に対するオプションリスト
	<pre> boxsize={60 60} position={center} fitmethod=meet matchbox={name=kraxi borderwidth=4 offsetleft=-2 offsetright=2 offsetbottom=-2 offsettop=2 linecap=round linejoin=round strokecolor={rgb 0.0 0.3 0.3}} </pre>

テキストを画像に揃える 以下のコード断片は、縦方向のテキストを、画像の右余白に合わせる方法を示します。画像ははめ込み方式 *meet* を用いて、与えられた枠に縦横比を保ってはめ込まれています。はめ込み枠の具体的な座標は *PDF_info_matchbox()* で取得され、縦方向のテキストははめ込み枠の右下隅 (*x2*, *y2*) に合わせて配置されています。範囲枠の辺は描線されています (図 9.44 参照)。

クックブック 完全なコードサンプルがクックブックの `images/align_text_at_image` トピックにあります。

```

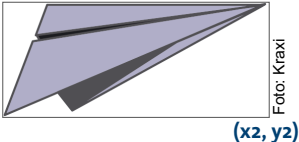
/* このオプションリストを使って画像を読み込んではめ込む */
String optlist = "boxsize={300 200} position={center} fitmethod=meet " +
    "matchbox={name=giantwing borderwidth=3 strokecolor={rgb 0.85 0.83 0.85}}";

/* 画像を読み込んではめ込む */

/* 範囲枠の右下 (第二) 隅の座標を取得 */
if ((int) p.info_matchbox("giantwing", 1, "exists") == 1)
{
    x2 = p.info_matchbox("giantwing", 1, "x2");
    y2 = p.info_matchbox("giantwing", 1, "y2");
}
/* その隅から2だけ間隔をあけてテキスト行を開始 */
p.fit_textline("Foto: Kraxi", x2+2, y2+2, "font=" + font + " fontsize=8 orientate=west");

```

図 9.44 画像の範囲枠の座標を使ってテキスト行をはめ込む

生成される出力	生成手順
	<p>手順 1 : 画像を範囲枠とともにはめ込み</p> <p>手順 2 : 範囲枠情報を得て座標 (<i>x2</i>, <i>y2</i>) を取得</p> <p>手順 3 : 取得した座標 (<i>x2</i>, <i>y2</i>) から <i>orientate=west</i> オプションでテキスト行を開始</p>

10 インタラクティブ機能

クックブック インタラクティブ要素を作成するコードサンプルがPDFlib クックブックの interactive カテゴリにあります。

10.1 リンク・しおり・注釈

この項では、しおり・フォームフィールド・注釈といったさまざまなインタラクティブ要素の作成方法を説明します。以下のインタラクティブ要素を持つ文書を作成してみましょう：

- ▶ 右上には、テキスト `www.kraxi.com` の所に、`www.kraxi.com` への非表示の Web リンクがあります。この領域をクリックすると、その対応する Web ページが表示されます。
- ▶ 灰色のフォームフィールドが、種類はテキストで、Web リンクの下に作ってあります。JavaScript を使ってここには今日の日付が自動的に記入されます。
- ▶ 赤い押しピンは添付を持った注釈です。クリックするとファイル添付が開きます。
- ▶ 左下にはフォームフィールドがあり、種類はボタンで、プリンターのアイコンを表示しています。このボタンをクリックすると Acrobat のメニュー項目「ファイル」→「印刷」が実行されます。
- ▶ ナビゲーションパネルにはしおり「Our Paper Planes Catalog」があります。このしおりをクリックすると、別の PDF 文書のページが表示されます。

以下、こうしたインタラクティブ要素を作成する方法を説明します。

Web リンク まずは、Web サイト `www.kraxi.com` へのリンクを作りましょう。これは 3 つの段階で達成できます。まず、Web リンクをつけたいテキストを配置します。`matchbox` オプションで `name=kraxi` を指定して、テキストのはめ込み枠を後で参照できるようにしておきます。

次に、URI 型 (Acrobat では：「Web ページを開く」) のアクションを作成しましょう。するとアクションハンドルが得られます。このアクションハンドルは後で 1 個ないし複数のインタラクティブ要素に割り当てることができます。

最後に、Link 型の注釈を用いてリンクを実際に作成しましょう。リンクに対する `action` オプションでは、イベント名 `activate` を指定してアクションをトリガーさせ、加えて上記で作成した `act` ハンドルをアクション内容として与えます。デフォルトではリンクは細い黒枠線がついて表示されます。最初のうちはこのほうが位置付けを正確にできて便利ですが、ここでは `linewidth=0` で枠線を非表示にしてあります。

```
normalfont = p.load_font("Helvetica", "unicode", "");
p.begin_page_ext(pagewidth, pageheight, "topdown");

/* テキスト行「Kraxi Systems, Inc.」を配置。範囲枠を使用 */
String optlist =
    "font=" + normalfont + " fontsize=8 position={left top} " +
    "matchbox={name=kraxi} fillcolor={rgb 0 0 1} underline";
p.fit_textline("Kraxi Systems, Inc.", 2, 20, optlist);

/* URIアクションを作成 */
optlist = "url={http://www.kraxi.com}";
int act = p.create_action("URI", optlist);
```

```

/* 範囲枠「kraxi」上にLink注釈を作成 */
optlist = "action={activate " + act + " } linewidth=0 usematchbox={kraxi}";
/* 長方形座標の0は範囲枠の座標に置き換えられる */
p.create_annotation(0, 0, 0, 0, "Link", optlist);

p.end_page_ext("");

```

画像やテキストフローの一部への Web リンクの作成例については 275 ページ「9.4 範囲枠」を参照してください。

クックブック 完全なコードサンプルがクックブックの `interactive/link_annotations` トピックにあります。

別のファイルへ移動するしおり 今度は、別のPDFファイルへ移動するしおり「Our Paper Planes Catalog」を作成しましょう。ファイル名は `paper_planes_catalog.pdf` であるものとします。まず、`GoToR` 型のアクションを作成します。このアクションに対するオプションリストで、移動先文書の名前を `filename` オプションで定義します。また `destination` オプションで、拡大表示させたいページ内の一部分を指定します。具体的には、表示されるのは文書の 2 ページ目で (`page=2`)、位置・倍率指定表示で (`type=fixed`)、ページの中頃が表示され (`left=50 top=200`)、表示倍率 200% (`zoom=2`) となります：

```

String optlist =
    "filename=paper_planes_catalog.pdf " +
    "destination={page=2 type=fixed left=50 top=200 zoom=2}";

goto_action = p.create_action("GoToR", optlist);

```

次の段階として、実際にしおりを作成します。このしおりに対する `action` オプションで、アクションのトリガーとして `activate` イベントを指定し、起こしたいアクションとして上で作成した `goto_action` ハンドルを指定します。`fontstyle=bold` オプションで太字のテキストを指定し、`textcolor=blue` でしおりを青くします。しおりのテキスト「Our Paper Planes Catalog」はメソッドの引数として与えます：

```

String optlist=
    "action={activate " + goto_action + " } fontstyle=bold textcolor=blue";
catalog_bookmark = p.create_bookmark("Our Paper Planes Catalog", optlist);

```

このしおりをクリックすると、文書内のページの指定部分が表示されます。

クックブック 完全なコードサンプルがクックブックの `interactive/nested_bookmarks` トピックにあります。

ファイル添付を持つ注釈 次に、ファイル添付を作成しましょう。まず `FileAttachment` 型の注釈を作成しましょう。`filename` オプションで添付の名前を指定し、`mimetype=image/png` オプションでその種類を指定します。このしおりは押しピンとして表示され (`iconname=pushpin`)、色は赤で (`annotcolor=red`)、説明を持ちます (`contents={Get the Kraxi Paper Plane!}`)。印刷されません (`display=noprint`)：

```

String optlist =
    "filename=kraxi_logo.png mimetype=image/png iconname=pushpin " +
    "annotcolor=red contents={Get the Kraxi Paper Plane!} display=noprint";
p.create_annotation(left_x, left_y, right_x, right_y, "FileAttachment", optlist);

```


iconname で定義したアイコンの大きさは変化しません。アイコンはその標準寸法のまま、指定した長方形の左上隅に表示されます。

10.2 フォームフィールドと JavaScript

印刷のためのボタンフォームフィールド 以下の例は、文書の印刷に使えるボタンフォームフィールドを作成します。最初のバージョンではボタンにラベルをつけておきます。その後でラベルをやめてプリンターのアイコンを使いましょう。まず *Named* 型 (Acrobat では:「メニュー項目を実行」) のアクションを作成しましょう。また、ラベルのフォントも指定しておく必要があります:

```
print_action = p.create_action("Named", "menuname=Print");
button_font = p.load_font("Helvetica-Bold", "unicode", "");
```

このボタンフォームフィールドに対する *action* オプションで、アクション実行のトリガーとして *up* イベント (Acrobat では:「マウスボタンを放す」) を指定し、アクションそのものとして上で作成した *print_action* ハンドルを指定します。 *backgroundcolor=yellow* オプションで背景を黄色に指定し、 *bordercolor=black* で枠線を黒に指定します。オプション *caption=Print* でボタンにテキスト「Print」をつけ、 *tooltip={Print the document}* でユーザーのための追加説明を作成します。 *font* オプションで、上で作成した *button_font* ハンドルを用いてフォントを指定します。デフォルトでは、ラベルの寸法はボタンの領域にちょうど収まるよう自動調整されます。そして、実際にボタンフォームフィールドを作成する際に、適当な座標と、名前 *print_button*、 *pushbutton* 型、適切なオプション群を指定します:

```
String optlist =
    "action {up " + print_action + "} backgroundcolor=yellow " +
    "bordercolor=black caption=Print tooltip={Print the document} font=" +
    button_font;
```

```
p.create_field(left_x, left_y, right_x, right_y, "print_button", "pushbutton", optlist);
```

それでは、この最初のバージョンのボタンを改良して、テキスト *Print* をやめて小さいプリンターアイコンに替えてみましょう。これを達成するには、その対応する画像ファイル *print_icon.jpg* をテンプレートとしてページ作成前に読み込みます。 *icon* オプションを用いてテンプレートハンドル *print_icon* をボタンフィールドに割り当てつつ、上記のコードと同様にフォームフィールドを作成します:

```
print_icon = p.load_image("auto", "print_icon.jpg", "template");
if (print_icon == -1)
{
    /* エラー処理 */
    return;
}
p.begin_page_ext(pagewidth, pageheight, "");
...
String optlist = "action={up " + print_action + "} icon=" + print_icon +
    " tooltip={Print the document} font=" + button_font;

p.create_field(left_x, left_y, right_x, right_y, "print_button", "pushbutton", optlist);
```

クックブック 完全なコードサンプルがクックブックの *form_fields/form_pushbutton* トピックにあります。

単純なテキストフィールド 今度は、テキストフィールドをページ右上隅付近に作成します。ユーザーは今日の日付をこのフィールドに入力することができます。フォントハン

ドルを取得し、**textfield** 型のフォームフィールドを作成して名前を **date**、背景を灰色とします：

```
textfield_font = p.load_font("Helvetica-Bold", "unicode", "");
String optlist = "backgroundcolor={gray 0.8} font=" + textfield_font;
p.create_field(left_x, left_y, right_x, right_y, "date", "textfield", optlist);
```

デフォルトでは文字サイズは **auto** であり、この場合フィールドの高さがそのまま初期の文字サイズとなります。入力がフィールドの終わりまで達すると文字サイズは小さくなって、テキストがつねにフィールドに収まるよう自動調整されます。

クックブック 完全なコードサンプルがクックブックの `form_fields/form_textfield_height` トピックにあります。

JavaScript を持つテキストフィールド 上で作成したテキストフォームフィールドを改良して、ページを開くと自動的に今日の日付が記入されるようにしましょう。まず、**JavaScript** 型 (Acrobat では：「**JavaScript を実行**」) のアクションを作成しましょう。このアクションのオプションリストの中の **script** オプションで JavaScript スニペットを定義します。このスニペットは、今日の日付を **date** テキストフィールドに月日年形式で表示します：

```
String optlist =
    "script={var d = util.printd('mmm dd yyyy', new Date()); " +
    "var date = this.getField('date'); date.value = d;}"
```

```
show_date = p.create_action("JavaScript", optlist);
```

第二段階として、ページを作成しましょう。オプションリストで **action** オプションを与え、その中で、上で作成した **show_date** アクションをトリガーイベント **open** (Acrobat では：「**ページを開く**」) に対して設定します：

```
String optlist = "action={open " + show_date + "}";
p.begin_page_ext(pagewidth, pageheight, optlist);
```

最後に、上と同様にテキストフィールドを作成しましょう。ページを開くたび、ここに自動的に今日の日付が記入されます：

```
textfield_font = p.load_font("Helvetica-Bold", "winansi", "");
String optlist = "backgroundcolor={gray 0.8} font=" + textfield_font;
p.create_field(left_x, left_y, right_x, right_y, "date", "textfield", optlist);
```

クックブック 完全なコードサンプルがクックブックの `form_fields/form_textfield_fill_with_js` トピックにあります。

テキストフィールドのためのフォーマットオプション Acrobat では、テキストフィールドに対してさまざまなオプションを指定して内容をフォーマットすることが可能です。たとえば通貨・日付・パーセントなどです。これは、カスタムの JavaScript コードを Acrobat が使用することによって実装されています。こういったフォーマット機能は PDF リファレンスには記載されていない物ですから、PDFlib は直接にはこれに対応していません。しかしながら、PDFlib ユーザーの便宜を図るため、以下、フォーマットオプションを実現するためのさまざまな簡単な JavaScript コード断片を **PDF_create_field()** の **action** オプションで与える方法について説明します。

テキストフィールドがフォーマットされるようにするには、JavaScript スニペットをそのフィールドの *keystroke・format* のアクションとして設定します。その JavaScript コードから何らかの内部 Acrobat 関数を呼び出し、このメソッドの引数群でフォーマットの詳細を制御します。

以下のサンプルでは、*keystroke・format* の2つのアクションを作成し、これらを1つのフォームフィールドに対して設定して、フィールド内容のフォーマットが小数点以下の桁数2・通貨記号 EUR となるようにしています：

```
keystroke_action = p.create_action("JavaScript",
    "script={AFNumber_Keystroke(2, 0, 3, 0, \"EUR \", true); }");

format_action = p.create_action("JavaScript",
    "script={AFNumber_Format(2, 0, 0, 0, \"EUR \", true); }");

String optlist = "font=" + font + "action={keystroke " + keystroke_action +
    " format=" + format_action + "}";
p.create_field(50, 500, 250, 600, "price", "textfield", optlist);
```

クックブック 完全なコードサンプルがクックブックの `form_fields/form_textfield_input_format` トピックにあります。

フォームフィールド入力を検証 以下のサンプルは、JavaScript をフォームフィールドに検証アクションとして紐付けて、テキストフィールドへのユーザー入力が、求められる形式 `mm/dd/yyyy` に合致しているかどうかをチェックします：

```
optlist =
"script={" +
    "// JavaScript code for date mask format MM/DD/YYYY\n" +
    "var re = /^[0-9]{2}\\/[0-9]{2}\\/[0-9]{4}$/\n" +
    "if (event.value !=\"\") {\n" +
    "  if (re.test(event.value) == false) {\n" +
    "    app.alert ({\n" +
    "      cTitle: \"Incorrect Format\", \n" +
    "      cMsg: \"Please enter date using mm/dd/yyyy format\"\n" +
    "    });\n" +
    "  }\n" +
    "}\n" +
    "}\n" +
    "};";
validate_action = p.create_action("JavaScript", optlist);
textfield_font = p.load_font("Helvetica", "unicode", "");
optlist = "action={validate=" + validate_action + "} " +
    "backgroundcolor={gray 0.8} font=" + textfield_font;
p.create_field(11x, 11y, 11x, 11y, "startdate", "textfield", optlist);
```

10.3 スクリーン注釈と表現アクションを用いたマルチメディア

一般 マルチメディアを、PDF 文書内でスクリーン注釈と表現アクションを使って再生できます。このメディアのデータは、PDF 文書内に埋め込むこともできますし、別の外部ファイルから取って来ることもできますし、リモートのサーバーから引っ張ることもできます。このメディア（表現と呼びます）は、ページ上で注釈によって定義された固定の位置でも、別のフローティングウィンドウ内でも、フルスクリーンモードでも再生させることができます。メディアとともにコントローラーを表示することもできますので、そうすればユーザーがメディアを再生・一時停止・再開・停止できるようになります。あるいはこれらのコントロールをPDFのフォームフィールドを用いて実装することも可能です。

マルチメディア要素を持ったPDFを再生するには、Adobe Acrobat か、マルチメディアアサポートをそなえたサードパーティーのPDFビューアが必要です。Webブラウザに内蔵されているPDFビューアの多くは目下マルチメディアの表現に対応していません。

PDFlibを用いて音声か動画を作成する手順はたいてい以下のとおりです：

- ▶ `PDF_load_asset()` で `type=Rendition` を用いて動画ファイルか音声ファイルかURLを読み込む。
- ▶ `PDF_create_action()` で `type=Rendition` を用いて表現アクションを作成。
- ▶ 動画が再生されるページ上の領域を用意するスクリーン注釈を作成。表現アクションは注釈のさまざまなトリガーイベントに紐付けることができます。たとえばページが開かれた時や注釈がクリックされた時などです。あるいはアクションをしおりかフォームフィールドに紐付けてメディアの再生を制御できるようにすることも可能です。

以下のコード断片は、表現とアクションと注釈を用いた標準的な手順を実装しています：

```
// その動画を持たせた表現オブジェクトを作成
rend_video = p.load_asset("Rendition", videofile,
    "mimetype=video/mp4 name=rendition_video controller");
if (rend_video == -1)
    throw new Exception("エラー：" + p.get_errmsg());

// その動画を再生するための表現アクションを作成
act = p.create_action("Rendition",
    "target=annot_video operation=playfrombeginning rendition=" + rend_video);

// その動画が再生される場所であるスクリーン注釈を作成
p.create_annotation(25, 300, 575, 610, "Screen",
    "title=annot_video name=annot_video action={activate=" + act + "}");
```

あるいはこの標準的な手順ではなく、注釈を使用せず、JavaScript を用いて表現を再生することも可能です（286 ページ「JavaScript でマルチメディア」を参照）。

クックブック 完全なコードサンプルがクックブックの `multimedia/screen_annotations` トピックにあります。

セキュリティを考慮 マルチメディアの再生は頻繁に攻撃のターゲットになるので、Acrobat DC はさまざまなセキュリティチェックを実装しています。マルチメディアコンテンツの再生を成功させるには、以下のいずれかの方式を用いることによって、PDF 文書が必ず信頼されるようにする必要があります：

- ▶ この文書を開く際に、Acrobat は、上端に「セキュリティ上の問題が発生する可能性を防ぐため、いくつかの機能が無効になっています。この文書を信頼する場合は、これらの機能を使用できます。」と書かれた黄色いバーを表示します。「オプション」メニューで「常にこの文書を信頼する」をクリック。
- ▶ 「編集」→「環境設定」→「一般...」→「セキュリティ (拡張)」へ行き、特権の場所のリストにこのファイルかフォルダーを追加。
- ▶ この文書に証明書の署名を行い、「編集」→「環境設定」→「一般...」→「セキュリティ (拡張)」→「有効な証明書を持つ文書を自動的に信頼する」を有効にする。

MIME タイプ `PDF_load_asset()` の `mimetype` オプションは必須です。適切な MIME タイプ (コンテンツタイプ) を与えることは、メディアが必ず正しく再生されるようにするために不可欠です。対応する MIME タイプのリストは、PDF ビューア・バージョンと、オペレーティングシステムが提供している利用可能なメディア再生能力に依存します。Acrobat で以下の JavaScript を使って対応している MIME タイプのリストを知ることができます：

```
// すべてのプレーヤーとメディアタイプをクエリー。JavaScriptコンソールで使用する
var mp = app.media.getPlayers("");
for (var i = 0; i < mp.length; i++) {
    console.println("\n" + mp[i].name + "が対応しているMIMEタイプ群:");
    console.println(mp[i].mimeTypes);
}
```

よく使われる音声・動画形式に対する MIME タイプを表 10.1 に挙げます。`PDF_create_action()` の `rendition` オプションは表現のリストを受け付けます。その場合、PDF ビューアは、最初の対応している形式の表現を採用します。この機能を活用すると、1つの動画を複数の形式で与えることによって、さまざまなターゲット環境でなるべくそれが再生できるようにすることが可能です。

表 10.1 よく使われる MIME タイプ

カテゴリー	よく使われる MIME タイプ
音声	audio/wav · audio/midi · audio/mp3 · audio/mp4
動画	video/mpeg · video/msvideo · video/avi · video/mp4 · video/quicktime

JavaScript でマルチメディア Acrobat 用の JavaScript API は、マルチメディアコンテンツを PDF 内で扱う広汎な機会を提供しています。たとえば、`PDF_create_action()` の `operation` オプションを用いて動画を起動するのではなく、`operation` オプションのかわりにシンプルな JavaScript 命令を与えることが可能です：

```
act = p.create_action("Rendition",
    "target=my_annot script={app.media.openPlayer();} rendition=" + rend_video);
```

表現がその文書の「表現名称ツリー」の中に格納されていれば (Acrobat ではそのような表現を「JavaScript からアクセス可能」と称します)、JavaScript を使ってその表現を操作することも可能です。これを実現するには `PDF_load_asset()` の `nametree` オプションを用います。以後その表現は注釈を一切必要とせずに JavaScript アクションを用いて再生できるようになります。たとえばしおりのクリックをトリガーにしたりすることが可能です：

```
// 表現を名前で取得し、それをフローティングウィンドウ内で特定のプロパティ群を用いて再生
// これが動作するには名称ツリー内の表現に対してだけ
```

```
var rendition = this.media.getRendition("rendition_sound");

if (rendition == null)
    app.alert("表現が見つかりませんでした");
else
{
    var floating = {
        width: 400,
        height: 300
    };
    var player = app.media.openPlayer({
        rendition: rendition,
        settings: {
            windowType: app.media.windowType.floating,
            floating: floating
        }
    });
}
```

以下のコードは、上記の JavaScript コードが文字列変数 *js* に格納されている前提で、それを仮想 (PVF) ファイルへ読み込んで、JavaScript アセットを作成し、そして JavaScript アクションを作成して、それをしおりに紐付けています：

```
// 音声ファイルを持つ隠れた表現を作成してそれを名称ツリー内に格納することによって
// それをJavaScriptからアクセス可能にする
int rend_sound = p.load_asset("Rendition", soundfile,
    "mimetype=audio/midi style=hidden nametree name=rendition_sound");
if (rend_sound == -1)
    throw new Exception("エラー： " + p.get_errmsg());

// JavaScript文字列を仮想 (PVF) ファイルへ読み込む
p.create_pvf("/pvf/javascript/script1.js", js.getBytes(), "");

// PVFファイルからJavaScriptアセットを作成
int js_asset = p.load_asset("JavaScript", "/pvf/javascript/script1.js", "");
if (js_asset == -1)
    throw new Exception("エラー： " + p.get_errmsg());

// アセットからJavaScriptアクションを作成
int act_js = p.create_action("JavaScript", "javascript=" + js_asset);

// この音声表現を再生するためのしおり
p.create_bookmark("JavaScriptを用いて音声を作成 (しおりをクリック)",
    "action={activate=" + act_js + "}");
```

1つのしおりに2つのアクションを紐付ければ、音声表現と動画表現を同時に再生することも可能であり、しかも対応するスクリーン注釈は必要ありません。注釈のない表現は、ページ上に領域を必要としない音声を再生するためにとりわけ有用です。

クックブック A full code sample for sound and video renditions without annotations can be found in the Cookbook topic [multimedia/javascript_for_renditions](#). The topic [interactive/renditions](#) in the pCOS Cookbook demonstrates how to enumerate all renditions in the name tree.

クックブック 注釈のない音声・動画表現の完全なコードサンプルがクックブックの `multimedia/`
`javascript_for_renditions` トピックにあります。pCOS クックブックの `interactive/`
`renditions` トピックは、名称ツリー内のすべての表現を列挙する方法を示しています。

10.4 地理空間 PDF

クックブック 完全なコードサンプルがクックブックの `geospatial/starter_geospatial` トピックにあります。

10.4.1 地理空間 PDF を Acrobat で利用

PDF 1.7ext3 では、地理空間参照情報（世界座標）を PDF ページ内容に追加することができます。Acrobat では、地理空間参照付き PDF 文書でいくつかのことができます。Acrobat・Acrobat Reader DC では、「ツール」→「ものさし」を有効にする必要があります：

- ▶ マウサーソルの下の地図上の位置の座標を表示：「地図位置ツール」。マウサーソルの下の地図上の位置の座標を、右クリックして「座標をクリップボードにコピー」を選択することでコピーできます。
- ▶ 地図上の位置を検索：「地図位置ツール」→右クリックして「位置を検索」を選択→求める座標を入力。
- ▶ 地図上の位置をマーク：「地図位置ツール」→右クリックして「位置をマーク」を選択。
- ▶ 地図上の距離・周辺・面積を測定：「ものさしツール」。

Acrobat Reader では上記のうち最初の 2 つの機能だけが利用可能です。地理空間測定のためのさまざまな設定を、「編集」→「環境設定」(→「一般...」)→「ものさし (地図情報)」で変更できます。たとえば座標読み上げのための望ましい座標系などです。

PDFlib の地理空間機能は以下の関数とオプションで実装されています：

- ▶ 1 つのページに対して、1 つないし複数の地理参照付き領域を、`PDF_begin/end_page_ext()` の `viewports` オプション (とサブオプション `georeference`) を用いて割り当てることができます。ビューポートを使うと、ページ上の別々の領域で別々の地理空間参照 (`georeference` オプションで指定) を用いることが可能になります。これはたとえば同一ページ上に複数の地図があるときなどに有用です。この方式は、地理空間 PDF への対応を有するすべての PDF ビューアーで動作します。

クックブック 完全なコードサンプルがクックブックの `geospatial/starter_geospatial` トピックにあります。

- ▶ `PDF_load_image()` の `georeference` オプションを使うと、画像に地球ベースの座標を割り当てることができます。

クックブック 完全なコードサンプルがクックブックの `geospatial/georeferenced_image` トピックにあります。

- ▶ `PDF_open_pdi_page()`・`PDF_load_graphics()`・`PDF_begin_template_ext()` の `georeference` オプションを用いて地球ベースの座標をフォーム XObject に割り当てることもできます。しかしこの方式は、Acrobat DC を含むいずれの既知のビューアーにおいても対応していないため、推奨しません。

10.4.2 地理座標系と投影座標系

地理座標系は地球を地理座標で、すなわち緯度と経度を度単位で表して記述します。投影座標系は、地理座標系の上に指定することができ、地理座標系における点から二次元 (投影) 座標系への変換を記述します。ここから算出される座標を Northing・Easting 値といい、投影座標系では角度はもはや不要です。地理座標系が GPS などの全球的応用分野で

用いられているのに対して、投影はそれよりもある程度局地的な地図製作などの応用分野で必要となります。

歴史的・数学的理由により、世界じゅうでさまざまな座標系が用いられています。地理座標系・投影座標系とも、EPSG・WKT という 2 種類の普及した方式で記述することができます。WKT・EPSG とも Acrobat で使用できますが、Acrobat はすべての可能な EPSG コードを実装しているわけではありません。特に、地理座標系のための EPSG コード群には Acrobat は対応していないようです。その場合には WKT の使用を推奨します。以下の Web サイトで、特定の EPSG コードに対応する WKT を示しています：

www.spatialreference.org/ref/epsg

10.4.3 座標系の例

地理座標系の例 WGS84 (世界測地系) 地理座標系は、GPS をはじめとする多くの応用分野 (*OpenStreetMap* など) の基礎となっています。これは以下のように *georeference* オプションの *worldsystem* サブオプションで表現できます：

```
worldsystem={type=geographic wkt={
GEOGCS["WGS 84",
    DATUM["WGS_1984", SPHEROID["WGS 84", 6378137, 298.257223563]],
    PRIMEM["Greenwich", 0],
    UNIT["degree", 0.01745329251994328]]
}}
```

ETRS (欧州地球基準系) 地理座標系は WGS84 とほとんど等価です。これは以下のように指定できます：

```
worldsystem={type=geographic wkt={
GEOGCS["ETRS_1989",
    DATUM["ETRS_1989", SPHEROID["GRS_1980", 6378137.0, 298.257222101]],
    PRIMEM["Greenwich", 0.0],
    UNIT["Degree", 0.0174532925199433]]
}}
```

注 WGS84・ETRS 系に対する EPSG コードはここでは示していません。なぜなら Acrobat は地理座標系に対する EPSG コードに対応しておらず、投影座標系にしか対応していないようだからです (後述)。

投影座標系の例 投影は、その背景にある地理座標系に基づいています。以下の例では、GPS 座標での利用に適した投影座標系を指定します。

中欧では、ETRS89 UTM zone 32 N という系が適用されます。これは広く利用されている UTM (国際メルカトル投影) を用いており、以下のように *georeference* オプションの *worldsystem* サブオプションで表現できます：

```
worldsystem={type=projected wkt={
PROJCS["ETRS_1989_UTM_Zone_32N",
    GEOGCS["GCS_ETRS_1989",
        DATUM["D_ETRS_1989", SPHEROID["GRS_1980", 6378137.0, 298.257222101],
            TOWGS84[0, 0, 0, 0, 0, 0, 0]],
        PRIMEM["Greenwich", 0.0],
        UNIT["Degree", 0.0174532925199433]],
    PROJECTION["Transverse_Mercator"],
    PARAMETER["False_Easting", 500000.0],
```

```
PARAMETER["False_Northing", 0.0],
PARAMETER["Central_Meridian", 9.0],
PARAMETER["Scale_Factor", 0.9996],
PARAMETER["Latitude_Of_Origin", 0.0],
UNIT["Meter", 1.0]]
}}
```

この座標系に対応する EPSG コードは 25832 です。WKT のかわりに、上記の系はその EPSG コードを通じて以下のように指定することもできます：

```
worldsystem={type=projected epsg=25832}
```

10.4.4 Acrobat における地理空間 PDF の制約

地理空間 PDF を Acrobat DC で扱うなかで、私たちは以下の難点に遭遇しています：

- ▶ EPSG コードは地理座標系に対してはまったく動作せず、投影系に対してのみ動作するようです。
回避策：EPSG コードでなく、その対応する WKT を使うこと。
- ▶ 地理空間データをフォーム XObject に紐付けても動作しません。この理由から、`PDF_open_pdi_page()`・`PDF_begin_template_ext()`・`PDF_load_graphics()`に対して `georeference` オプションを指定することは推奨しません。PDF Reference に従えばこれは動作するべきなのですが。
ベクトルベースの地図を作成するための回避策：地理空間データをページに紐付けることはできます。すなわち、`PDF_begin_page_ext()` の `viewports` オプションを用います。
- ▶ 重なり合った地図：同一ページ上に複数の画像ベースの地図を貼り付けることができます。複数の地図が重なり合っているとき、重なり合っている領域内の点の座標を表示させると、Acrobat は、最後に貼り付けられた地図の座標を用います（これがすなわち見えている地図でもありますのでこれは理にかなっています）。しかし、双方の画像ハンドルが同一の（すなわち `PDF_load_image()` への 1 回の呼び出しで取得された）場合には、Acrobat はもはやさまざまな画像の領域を考慮しなくなります：1 番目の画像の座標が誤って 2 番目の画像の領域へ拡張され、誤った座標表示となってしまいます。
回避策：同一ページ上に同じ画像をベースにした地図を複数枚貼り付けたいときは、その画像を複数回開くこと。
- ▶ 面積ものさしツールは地理座標系に対しては正しく動作せず、投影系に対してのみ正しく動作します。



11 タグ付き PDF と PDF/UA

11.1 タグ付き PDF の基礎

タグ付き PDF は、ISO 標準 PDF/UA・PDF/A-1a・PDF/A-2a・PDF/A-3a、米国のセクション 508、ドイツの BITV、その他多くの法規における必須事項です。タグ付き PDF は、以下の利点を提供する文書構造情報で PDF を向上させます：

- ▶ アクセシビリティ：タグ付き PDF は、たとえば Acrobat 内蔵の読み上げ機能や、より高度なスクリーンリーダーソフトウェアなどを通じて、障害を持つユーザーのためにアクセシブルです。
- ▶ 他の文書形式への信頼性高い書き出し・変換：タグ付き PDF を、RTF・XML・HTML といった他の形式へ変換すると、より正確な出力が得られます。

PDF/UA は、タグ付き PDF を、文書タグに関する要件を仕様化することで改良したものです。アクセシブルな PDF 文書を作成したいなら、PDF/UA のための追加規則群に従うことを推奨します：詳しくは 335 ページ「11.4 PDF/UA によるユニバーサルアクセシビリティ」を参照してください。

すべての PDF/UA の要件には従えない場合には（たとえば PDF/UA に準拠していない既存の PDF を元に文書を組み立てなければならないなどの理由で）、PDF/UA モードを無効にして、できる限り多くの PDF/UA 規則に従うことを推奨します。

クックブック タグ付き PDF を生成するためのコードサンプルが、PDFlib クックブックの pdfua カテゴリにあります。クックブック内のタグ付き PDF サンプルはすべて PDF/UA を作成します。

11.1.1 論理構造ツリー（構造ヒエラルキー）を作成

タグ付き PDF は、クライアントがその文書の内部構造に関する情報を提供し、かつ PDF 出力を生成する際に特定の規則に従った場合のみ作成できます。タグ付き PDF を作成するには、*tagged* 文書オプションを *true* に設定する必要があり、また、*lang* オプションが推奨されます（PDF/UA-1 では必須です）：

```
if (p.begin_document("tagged.pdf", "tagged=true lang=en") == -1) {
    throw new Exception("エラー：" + p.get_errmsg());
}
```

タグ付き PDF 文書内の論理構造は、要素のヒエラルキーによって記述されます。これを構造ヒエラルキー、または論理構造ツリーといいます。ルートレベルから始まって、この構造ヒエラルキーは任意の数のレベルから成ります。各レベルにおいては、1 個の要素は、以下の種類のアイテムを 0 個以上含むことができます：

- ▶ 他の構造要素。たとえば *Document*（文書）要素は、複数の *Art*（アーティクル）要素を含むことができ、各 *Art* 要素はさらに、複数の *P*（段落）要素を含むことができます。
- ▶ 内容アイテム。すなわち、ページ上のテキスト列とグラフィックや、取り込まれた画像から作成された XObject や、注釈・フォームフィールドへの OBJR 参照。こうしたアイテムは、構造要素に関連づけられた内容を表現します。

Acrobat における構造ヒエラルキー タグ名と構造ヒエラルキーは、Acrobat DC で以下のように表示できます：

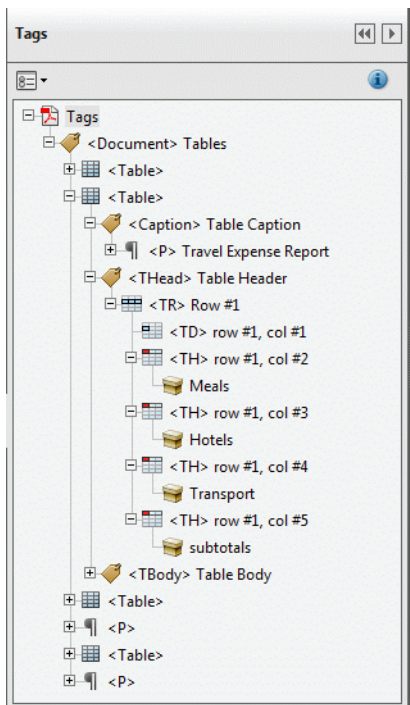


図 11.1 Acrobat のタグパネルが
文書の論理構造ツリーを表示

- ▶ 「表示」→「表示切り替え」→「ナビゲーションパネル」→「タグ」を選択 (図 11.1 参照)

タグネ스팅関数 構造要素は、`PDF_begin_item()` メソッドを用いて明示的に作成できます。この関数は必ず、対応する `PDF_end_item()` への呼び出しとペアにする必要があります。たとえば、以下のコード断片は、見出し 1 個と段落 1 個を内容とするセクション 1 個から成るヒエラルキーを作成します。階層関係をインデントで視覚化しています：

```

/* 種別「Sect」（セクション）の構造要素を作成 */
id_sect = p.begin_item("Sect", "Title={来し方行く末}");

    /* 種別「H1」（見出し）の構造要素を作成 */
    id_h1 = p.begin_item("H1", "Title={企業沿革}");
        p.fit_textline(...);
    p.end_item(id_h1);

    /* 種別「P」（段落）の構造要素を作成 */
    id_p = p.begin_item("P", "");
        p.fit_textline(...);
    p.end_item(id_p);

/* 「Sect」を閉じる */
p.end_item(id_sect);

```

デフォルトでは、構造要素は、カレントでアクティブなアイテムの子として、他の子アイテムがすでに存在すればそれらすべての後に、挿入されます。要素が論理的順序で作成さ

れるならば、これによって正しいツリー構造が得られます。もっと高度な技法については 327 ページ「11.2.5 内容を順序にとらわれず作成」を参照してください。

構造要素は、オプションを通じて与えられる 1 個ないし複数の属性を保持することができます。たとえば、内容要素の自然言語を指定するための *lang* や、画像に対する代替テキストのための *Alt* です。利用可能なオプションの集合は、構造種別によって異なります。

短縮タグ付け 多くの場合において、構造要素の内容は、PDFlib はめ込みメソッドを 1 回呼び出すことで作成することができますので、結果として典型的な並びは *PDF_begin_item()/PDF_fit_*/PDF_end_item()* となります。この並びは、短縮タグ付けを用いて削減することが可能です。ページ内容を作成するための多くのメソッドが対応している *tag* オプションを用いることによって、内容を配置することと、タグ付け情報を与えることとを、ただ 1 回の関数呼び出しにまとめることができます。上記のコード断片は、テキスト・画像配置関数に *tag* オプションを与えることによって単純化できます：

```
/* 種別「Sect」（セクション）の構造要素を作成 */
id_sect = p.begin_item("Sect", "Title={来し方行く末}");

/* 種別「H1」（見出し）の構造要素を作成 */
p.fit_textline(..., "tag={tagname=H1 Title={企業沿革}}");

/* 種別「P」（段落）の構造要素を作成 */
p.fit_textline(..., "tag={tagname=P}");

/* 「Sect」を閉じる */
p.end_item(id_sect);
```

タグネ스팅メソッドは *PDF_begin_item()* にも対応していますので、両方の方式を合わせることも可能です。これは、タグの下にタグをネストする必要があり、そしてネストされたタグがさらに複数の内容アイテムを内容とする場面において役立ちます。たとえば、*Caption* 要素の中に *P* 要素があり、その中にさらに複数のテキストアイテムがあるようにしたければ、以下のように生成できます。

```
id_caption = p.begin_item("Caption", "tag={tagname=P}");
    p.fit_textline(...);
    p.fit_textline(...);
p.end_item(id_caption);
```

tag オプションのオプションリストの中のサブオプションとして 2 番目の *tag* オプションをネストすることもできます。実際、*tag* オプションは任意のレベルまでネストさせることが可能です。ネストされた構造要素の内容が内容アイテム 1 個だけである場合には、上記のコード断片は、テキストをページ上に配置するための呼び出しの中で短縮タグ付けを直接適用することにより、さらに単純化することが可能です。

```
p.fit_textline(..., "tag={tagname=Caption tag={tagname=P}}");
```

短縮タグ付けは、生成された構造要素のハンドルを露出しませんので、*PDF_activate_item()* で、または *tag* オプションの *parent* サブオプションで使うことはできません。これらは構造要素 ID を必要とするからです。*tag* オプションの動作は詳しくは以下のとおりです：

- ▶ 生成された内容のために新規の構造要素が作成され、そして呼び出しから返る前に閉じられます。以下の状況はこの規則から除外されます：
 - ▶ `PDF_begin_document()` で `tag` オプションを用いて作成された構造要素は、`PDF_end_document()` で閉じられます。
 - ▶ `PDF_begin_item()` の `tag` オプションを用いて複数のタグが与えられたときは、これらすべてのタグは、対応する `PDF_end_item()` への呼び出しで閉じられます。
 - ▶ `PDF_fit_table():tagname=Table` または `PDF_fit_table()` で `Table` にロールマップされているタグ名は、PDFlib に対して、必要な表タグ群を作成するよう指示します (317 ページ「11.2.1 自動表タグ付け」参照)。表セルに対して自動的に生成された `TH・TD` タグは、`PDF_add_table_cell()` の `tag` オプションでさらに修飾することもできます。
 - ▶ `PDF_fit_textflow()` : 完全なテキストフローインスタンスは新規の構造要素を形成します。短縮タグ付けは、後述のテキストフローオプションリストの中のタグよりも優先されます。
 - ▶ 生成される要素は、カレントでアクティブなアイテムの、または `parent` オプションで与えられているアイテムの子です。
 - ▶ グループ化要素は、`PDF_begin_item()` でのみ作成することができ、`PDF_begin_document()` 以外の短縮タグ付けを用いて作成することはできません。
 - ▶ `PDF_begin_document()` と `PDF_add_table_cell()` の場合を除いて、短縮タグ付けはページスコープ内でのみ使用できます。

テキストフローオプションリスト内でタグ `PDF_begin/end_item()` を用いて明示的にタグ付けを行うのではなく、多くの関数の `tag` オプションを用いて短縮タグ付けを行うのでもなく、タグ付けの情報をテキストフローの一部として与えることも可能です。構造要素名と属性を `PDF_create_textflow()`・`PDF_add_textflow()` でオプションとして与えることができます。タグ付けの情報をテキストフローオプションリスト内に直接含めることができます：

```
<tagbegin={tagname=H1 Title={Company History}}>Introduction<tagend>\n
<tagbegin={tagname=P}>Kraxi Systems was founded in ... <tagend>\n
```

`tagbegin` オプションは短縮タグ付けと同じオプションを受け付け、一方 `tagend` オプションは最も最近に開かれた構造要素を閉じます。これらのタグ付けオプションは他のテキストフロー制御オプションと組み合わせることもできます。

タグを持ったテキストフローはすべてのインスタンスを出力ページに配置する必要があります。すなわち `PDF_fit_textflow()` を、それが `_stop` を返すまで呼び出しつづける必要があります。テキストフローのすべてのインスタンスが配置されないまれな場合、すなわちテキストの一部が失われる場合においては、最後にこのテキストフローのために `PDF_fit_textflow()` を呼び出す時に `tagstop` オプションを与える必要があります。

テキストフローを出力に配置する際には、テキストフローのインスタンスは、テキストの量とはめ込み枠の寸法によって1個ないし複数生成されます。PDFlib は、テキストフローのすべてのインスタンスが必ず1つの共通の構造要素の子になるようにし、また、テキストフローのインスタンスの末尾で中断された要素が次のインスタンスで必ず自動的にまた有効化されるようにします。テキストフローのインスタンスの配置は、同一ページ上であっても、異なるページ上であってもかまいません。

テキストフローオプションリスト内のタグは以下の場合には無視されます：

- ▶ タグ付けがサポートされていないコンテキストにおいて。たとえばテンプレート上や、タグ付けなしの出力の場合。

▶ `PDF_fit_textflow()` に `tag` オプションを与えている場合。

テキストフローマクロ (242 ページ「9.2.3 インラインオプションリストとマクロ」を参照) を使って、タグ付けのための便利なショートカットを定義することもできます。例:

```
macro {
  H1 {tagbegin={tagname=H1 Title = {Company History}} }
  /H1 {tagend}

  P {tagbegin={tagname=P} }
  /P {tagend}
}
```

マクロはそれ以後、インラインオプションリスト内で、マクロを開始する `&` を使って使用することができます。

```
<&H1>Introduction<&/H1>\n
<&P>Kraxi Systems was founded in ... <&/P>\n
```

なお、オプションリストはつねに各オプションに対して値をそれぞれ1つずつしか伝達しないことに注意してください。1つのオプションを複数回使用した場合、同じオプションリストの中で最後に書いたそのオプションが、それより前に書いたもの全部をオーバーライドします。たとえば、以下の構成は構造要素を2つ開始しません。なぜなら2個目の `tagbegin` が1個目をオーバーライドするからで、`LI` アイテムだけが作成されて `L` アイテムは無視されます:

```
<tagbegin={tagname=L} tagbegin={tagname=LI}> // 間違い: 1つのオプションリスト!
```

構造要素を2つ作成するには、2つの `tagbegin` オプションを別々のインラインオプションリストの中で与えるか、別々の `PDF_add_textflow()` への呼び出しで与える必要があります。例:

```
<tagbegin={tagname=L}><tagbegin={tagname=LI}> // 正しい: 2つのオプションリスト
```

2つのネストした構造要素で同じ内容を囲う場合には、1つのオプションリストの中で `tagbegin` オプションと一緒にもう1つ、ネストした `tag` オプションを与えることができます。たとえば、`Caption` 要素は内容アイテムを子に持たないので、`P` など実際の内容を持つ他の要素を子にする必要があります。この構成はインラインテキストフローオプションを用いて以下のとおり表現できます:

```
<tagbegin={tagname=Caption tag={tagname=P}}>Figure 9.1<tagend>
```

この `tagend` オプションは、対応する `Caption` 要素だけでなく、ネストした `P` 要素をも閉じます。

クックブック テキストフローオプションリスト内でタグを与えてタグ付きPDFを作成するコードサンプルが、PDFlib クックブックの `pdfua` カテゴリー内の `inline_tags_in_textflow_pdfua1` トピックにあります。

11.1.2 標準構造種別

PDF 1.7 と PDF 2.0 のタグセット PDF は、幅広い文書分類に対応するよう設計されたさまざまな標準構造種別に対応しています。PDFlib は、表 11.1 に従って PDF 1.x と PDF 2.0

のすべての標準要素種別に対応しています。この表の中で示している説明は、適切な種別を選ぶ助けとなることを意図しています。選択した PDF 出力互換性によって、利用可能な構造種別のグループ、いわゆるタグセットが異なります：

- ▶ PDF 1.7 タグセットは、**compatibility=1.7** とそれ以下の場合にはつねに利用可能です。**compatibility=2.0** の場合には、このタグセットは、**pdftagset=1.7** であれば利用可能です。これは、表 11.1 で **PDF 2.0** と標識していないすべての種別を含んでいます。
- ▶ PDF 2.0 タグセットは、**compatibility=2.0** の場合のデフォルトです。これは、PDF 1.7 タグセットの種別の多くに加え、表 11.1 で **PDF 2.0** と標識している種別を含んでいます。

構造要素はすべて、標準構造種別の集合から採るか、(カスタム要素の場合には) 標準構造種別のいずれか 1 つへロールマップしているかのいずれかとする必要があります (308 ページ「11.1.5 カスタム構造種別・ロールマップ・カスタムタグセット」を参照)。標準構造種別は以下を含むいくつかのカテゴリーに分類できます：

- ▶ グループ化レベル構造種別は、他の要素をグループ化するコンテナであり、ページ内容をまったく子に持ちません。
- ▶ ブロックレベル構造種別は、ページの内容の総体的なレイアウトを記述します。
- ▶ インラインレベル構造種別は、ブロックレベル要素内の内容のレイアウトを記述します。
- ▶ 特殊種別は、リスト、表、日本語のルビ／割注などのために使用されます。
- ▶ 擬似構造種別は、いかなる構造要素をも作成することなく、内容を特定の特性でマークアップするために用いられます。

複数のカテゴリー子に持ってで利用できる構造種別もあります。

究極的にはページ内容は、内容アイテム、すなわちテキスト・画像・ベクトルグラフィック・インタラクティブ要素によって記述されます。構造種別どうしの関係と、内容を子に持てる要素どうしの関係は、タグネ스팅規則によって規定されます。例外もありますが、標準構造種別に対するネ스팅規則をまとめるとおおむね以下のとおりです：

- ▶ 文書レベルとグループ化レベルの要素は、他の文書レベル・グループ化・ブロックレベルの要素のみを子に持つことができ、インラインレベル要素か内容アイテムを子に持つことは一切できません。
- ▶ 内容アイテムは、ブロックレベルとインラインレベルの要素の子と、リストと表の適切なサブ要素の子にのみなることができます。
- ▶ ブロックレベル要素は、インラインレベル要素と内容アイテムのみを子に持てます。
- ▶ インラインレベル要素は、他のインラインレベル要素と内容アイテムのみを子に持てます。

PDF 1.7 と PDF 2.0 の場合の構造要素ネ스팅規則について詳しくは以下の項で解説します。アプリケーションはこれら規則に従う必要があります。PDF/UA-1 には他にも規則が課されます (335 ページ「11.4 PDF/UA によるユニバーサルアクセシビリティ」を参照)。

構造要素に対するネ스팅規則チェックは、**PDF_begin_document()** の **checktags** オプションを用いて無効化か緩和することもできます。しかしこれは、無効な構造ヒエラルキーを生み出す可能性がありますので推奨しません。このオプションは、レガシアプリケーションのための移行の助けとして意図されているものです。表 11.3 内のいくつかの規則を「厳格規則」として標識しています。オプション **checktags=relaxed** は、この厳格規則群以外のすべてのチェックを強制します。

表 11.1 タグ付き PDF 内の標準種別（タグ）と、PDFlib によって追加された擬似種別

構造種別	説明
文書レベル構造種別（文書レベル・グループ化・ブロックレベルの要素を子に持てるが、インラインレベル要素が内容アイテムを子に持てない）	
<i>Document</i>	自己完結している論理的文書。PDF 2.0 では構造ツリーのルート要素として必須。ネストした Document 要素群を子に持つことも可能。
<i>Document-Fragment</i>	(PDF 2.0) 論理的文書断片。多くの場合、もとの文書全体の一部を抜き出して作成される。ネストした DocumentFragment 要素群を子に持つことも可能。
グループ化レベル構造種別（ページ内容を子に持てない）	
<i>Art</i>	(アーティクル。compatibility=2.0 の場合には pdftagset=1.7 が必須) 文書内のアーティクル。
<i>Aside</i>	(PDF 2.0) 親構造要素内で他の内容から区別されている内容。たとえばサイドバー内のテキストや補足コメント。
<i>BlockQuote</i>	(ブロック引用。compatibility=2.0 の場合には pdftagset=1.7 が必須) 周囲のテキストの作成者以外の誰かに帰する 1 個ないし複数の段落から成るテキストの部分
<i>Div</i>	(区画) 文書の、意味上の意図のない分割。セマンティクスに基づかないグループ化に対して使用することもできます。たとえばブロックレベル要素のシーケンスに lang 属性を紐付けたり、他の標準構造要素がどれもふさわしくないカスタムグループ化構造要素をロールマップしたりする場合はこれにあたります。
<i>Index</i>	(インデックス。compatibility=2.0 の場合には pdftagset=1.7 が必須) 文書の主本文内における指定されたテキストの出現を指し示す Reference 要素を伴った識別テキストを内容とする項目の列
<i>NonStruct</i>	(非構造要素) 固有の構造特性を何も持たない要素をグループ化。これはグループ化のみを目的とするものです。
<i>Part</i>	ヒエラルキーを考慮せずに構造要素群をグループ化したものを内包 (セマンティクス界の Div)
<i>Private</i>	(プライベート要素。compatibility=2.0 の場合には pdftagset=1.7 が必須) アプリケーションに属するプライベート内容を内容とするグループ化要素。この種別の構造上の意義は仕様化されていません。この Private 要素も、その子孫要素のいずれも、他の文書形式へは解釈も書き出しもされません。
<i>Sect</i>	(セクション) ヒエラルキーを考慮して構造要素群をグループ化したものを内包
<i>TOC</i>	(目次。compatibility=2.0 の場合には pdftagset=1.7 が必須) 目次項目群 (構造種別 TOCI) および / または他のネストされた目次エントリー群 (TOC) から成るリスト。
<i>TOCI</i>	(目次項目。compatibility=2.0 の場合には pdftagset=1.7 が必須) 目次のメンバー。これは適切な Link 要素を内容とすべきです。
ブロックレベル構造種別（インラインレベル要素のみを子に持てます）	
<i>FNNote</i>	(PDF 2.0。グループ化レベルかブロックレベルかインラインレベル) 脚注・後注。これを囲っており参照している内容の一部としては通常読まれないもの。
<i>H</i>	(見出し。PDF/UA-1 では structuretype=strong の場合にのみ可) 見出しと通常呼ばれる内容の低レベル区画。H 要素は階層的にネスティングすることを意図されています。H1 などを用いた弱い構造化のほうが望ましいです。
<i>H1~H6</i>	見出しと通常呼ばれる内容の低レベル区画。これらの種別は、アプリケーションがセクション群を階層的にネストできないために見出しのレベルをそのネスティングのレベルから判定できない場合に用いるべきです。Lb1 構造種別を用いて見出しの中のセクション番号を囲うこともできます。

表 11.1 タグ付き PDF 内の標準種別（タグ）と、PDFlib によって追加された擬似種別

構造種別	説明
<i>H7</i> ・ <i>H8</i> ・...	(PDF 2.0 かロールマッピングを持つ PDF/UA-1) もっと深いネスティングのためのさらなる見出しレベル。PDF/UA-1 では P へのロールマッピングを推奨します。
<i>P</i>	(段落) 内容の低レベル区画。多くの場合に段落に用いられますが、任意の内容の低レベル区画を囲えます。
<i>Title</i>	(PDF 2.0。グループ化レベルかブロックレベル) 文書か内容の高レベル区画のタイトルと通常呼ばれる内容を囲います。これを囲っている内容から脚注か後注への参照があるべきです。
インラインレベル構造種別	
<i>BibEntry</i>	(書誌項目。compatibility=2.0 の場合には pdftagset=1.7 が必須) 何らかの引用内容の外部ソースを特定する参照
<i>Code</i>	(compatibility=2.0 の場合には pdftagset=1.7 が必須) コンピューターのプログラムのテキストの断片
<i>Em</i>	(PDF 2.0) 強調の目的のために内容を囲います。強調のレベルを上げるためにネストした Em 要素を使用することもできます。
<i>Lbl</i>	同じ親要素の中の他の内容から区別するための内容。たとえばリスト項目に対するビュレット・番号や、キャプションの中の番号や、見出しの中の付番。
<i>Note</i>	(compatibility=2.0 の場合には pdftagset=1.7 が必須) 脚注や後注のように、文書の本体の中から参照される説明テキストのアイテム。この要素は Lbl を子に持てます。
<i>Sub</i>	(PDF 2.0) ブロックレベル要素の中の下位区画として通常知覚される内容を囲います。たとえば詩の中の節や、ソースコードの行や、住所の中の行。
<i>Quote</i>	(引用。compatibility=2.0 の場合には pdftagset=1.7 が必須) 周囲のテキストの作成者以外の誰かに帰するテキストのインライン部分。この引用されたテキストは、単一の段落内にインラインで含まれているべきです。
<i>Reference</i>	(compatibility=2.0 の場合には pdftagset=1.7 が必須) 文書内の他の所にある内容への参照。これはローカルなリンクのために使用されるべきです。
<i>Span</i>	(表の表行または列の連結とは関係ありません) 特定の固有の特性を何ら持たない一般的な内容のインライン部分。lang など特定の属性を持ったテキストの範囲を区切るために使うこともできます。
<i>Strong</i>	(PDF 2.0) 強い重要性・重大性・緊急性の目的のために内容を囲います。
インタラクティブ要素のための構造種別（グループ化レベルかブロックレベルかインラインレベル）（320ページ「11.2.2 インタラクティブ要素にタグ付け」参照）	
<i>Annot</i>	(PDF 1.5) 内容を1つないし複数の注釈に紐付けます。構造ツリー内に1つないし複数の注釈を含めるために使用することもできます。Link か Form のほうが適切でない限りはこの要素を注釈に対して使用する必要があります。
<i>Form</i>	内容を、対応するフォームフィールドと紐付けます。構造ツリーにフォームフィールドを含めるためにも使えます。Lbl 要素を用いてフォームフィールドのラベルをマークアップすることもできます。
<i>Link</i>	内容をリンク注釈と紐付けます。構造ツリーにリンク注釈を含めるためにも使えます。この要素をリンク注釈に対して使用する必要があります。
和文ルビ・割注のための要素（インラインレベル）	
<i>Ruby</i>	(PDF 1.5) 比較的小さなテキストサイズで書かれ、それが参照する親文字テキストに隣接して配置されたサイドノート。この Ruby 要素は、ルビ組立構造全体を囲むラッパーとしての働きをします。RB/RT ペア 1 個か、RP・RT・RP の並び 1 個を子に持つ必要があります。

表 11.1 タグ付き PDF 内の標準種別（タグ）と、PDFlib によって追加された擬似種別

構造種別	説明
RB	（ルビ親文字テキスト。PDF 1.5）ルビ注釈が施される対象となるフルサイズのテキスト。
RT	（ルビ注釈テキスト。PDF 1.5）ルビ親文字テキストに隣接して配置されるべき、比較的小さなサイズのテキスト。
RP	（ルビ約物）ルビ注釈テキストを囲う約物。これは、ルビ注釈がルビスタイルで適切に組版できずに、通常のコメントとして組版されるか割注として組版される場合にのみ用いられます。
Warichu	（割注。PDF 1.5）より小さなテキストサイズで、それを含むテキスト行の高さの中に、より小さな 2 行として組版され、それが参照するベーステキストに後続して（インラインに）配置される、コメントまたは注釈。WP・WT・WP の並び 1 個を子に持つ必要があります。
WT	（割注テキスト。PDF 1.5）割注コメントの、2 行として組版され、それを囲う WP 要素の間に配置される、より小さなサイズのテキスト。
WP	（割注約物。PDF 1.5）WT 要素の中の内容を囲う約物
リスト構造種別	
L	（リスト。ブロックレベルかインラインレベル）意味的に互いに関連している項目の並び
LI	（リスト項目）リストの個々のメンバーのための内容
LBody	（リスト本体）リスト項目の内容本体
表構造種別（表は自動的にタグ付けすることも可能です。317ページ「11.2.1 自動表タグ付け」参照）	
Table	（ブロックレベルかインラインレベル）長方形セル群の 2 次元論理構造。複雑な下位構造を含むこともあります
TR	（表行）表ヘッダーセル（TH）群か表データセル（TD）群の表行
TH	（表ヘッダーセル）表の 1 個ないし複数の表行、または列、または表行と列を記述する表ヘッダーセル。
TD	（表データセル）表の内容の一部であるデータを内容とする表セル
THHead	（表ヘッダー行グループ：PDF 1.5）表のヘッダーを成す TR 要素のグループ
TBody	（表本体行グループ：PDF 1.5）表の本体部を成す TR 要素のグループ
TFoot	（表フッター行グループ：PDF 1.5）表のフッターを成す TR 要素のグループ
キャプション構造種別（グループ化レベルかブロックレベル）	
Caption	（PDF 2.0。PDF 1.5 では表とリストに対してのみ可）表・リスト・図など要素を説明する短いテキスト
図・式構造種別（グループ化レベルかブロックレベルかインラインレベル）	
Figure	グラフィカルな内容
Formula	数式・化学式など式。この種別は、内容要素全体を式として識別します。
ページ装飾構造種別（グループ化レベルかブロックレベルかインラインレベル）	
Artifact	（PDF 2.0）ページ装飾。実際のページ内容に対してこう呼ばれます（Section 12.1.6, »Artifacts«, page 299 を参照）。ページ装飾は構造ツリーの要素としては PDF 2.0 のみ使用できます。ページ装飾はマーク付き内容としてはどのバージョンの PDF でも使用できます。構造ツリー内ではページ装飾は、他の構造要素をコンテキストとして必要とする本文外の内容のために使用できます。たとえば行番号です。この種別は注釈をページ装飾としてタグ付けするにも必要です。

表 11.1 タグ付き PDF 内の標準種別（タグ）と、PDFlib によって追加された擬似種別

構造種別	説明
擬似構造種別	
Artifact	直接マーク付き内容としてのページ装飾。実のページ内容に対してこう呼ばれます（309 ページ「11.1.6 ページ装飾」を参照）。
ASpan	（アクセシビリティースパン。PDF には Span として書き込まれますが、インラインレベル構造種別 Span とは区別する必要があります）構造要素に属しない、あるいは構造要素の一部分であるような内容に対して、アクセシビリティ特性群を紐付けます。この ASpan 擬似要素は、Alt・ActualText・Lang・E といったアクセシビリティ属性を持った Span として書き込まれます。ASpan は、いかなる構造要素とも紐付きません。
ReversedChars	右書き用字系の中の、反転したキャラクター群によるテキストを指定。
Clip	マークされた切り抜き列を指定。これは、クリッピングパスまたはテキスト表現モード 7 のテキストのみを内容とする列であり、目に見えるグラフィックまたは PDF_save()/PDF_restore() を一切含みません。

通常要素と直接要素の違い たいていの構造要素は、対応するテキストかグラフィックをマークアップし、かつ文書構造ツリー内へ対応するエントリーを追加することによって、PDF へ出力されます。このような要素は通常要素と呼ばれ、Acrobat のタグペーンの中の構造ツリーの中に表示されます。

これに対し、マーク付き内容のみから成り、構造ツリー内にエントリーを持たない要素というものもあります。このような要素は Acrobat のタグペーンに表示されません。このような直接要素の利点は、PDF 出力内で必要なスペースが少ない点です。構造種別のなかには、**direct** オプションを用いてそのステータスを変えることができます（デフォルトは **true**）。通常の構造要素と直接要素との比較を表 11.2 に示します。PDF 2.0 では直接要素には他にもネスティング規則が課されるので、表ではそれも示しています（いずれの場合にも通常の PDF 2.0 のネスティング規則が子要素に課せられます）。

デフォルトで直接的な要素が別の直接でない要素を子に持っている場合にはオプション **direct=false** が必須です。たとえば **Reference** が **Link** を子に持っている場合がこれにあたります。

表 11.2 通常構造要素と直接構造要素

	通常構造要素 (direct=false)	直接構造要素 (direct=true)
対象要素	その他すべての構造種別	Code・BibEntry・Em・FENote・Note・Quote・Reference・Span・Strong・Sub BibEntry・TOCI・Note の子として Lbl 疑似アイテム Artifact・ASpan・ReversedChars・Clip
構造ツリーの一部である	○	×
ページ境界をまたぐことが可能	○	×

表 11.2 通常構造要素と直接構造要素

	通常構造要素 (direct=false)	直接構造要素 (direct=true)
他の要素で割り込むことが可能	○	×
PDF activate_item() で有効にすることが可能	○	×
子要素に対するネスティング規則	通常・直接要素を内容とすることが可能	その親が直接要素を内容とすることが可能な場合に限り、直接要素のみを内容とすることが可能

空の構造要素 一般的には、子要素も内容アイテムも子に持たない構造要素は避けるべきです。しかし、以下の例のように、この規則にはいくつかの例外があって、ここでは空の構造要素が、何らかの他の構造の中にあつて、意味的な役割を伝えます：

- ▶ 空の *TD* 要素は空の表セルに対して必要です。PDFlib の自動表組版機能はこの規則に従います (317 ページ「11.2.1 自動表タグ付け」参照)。
- ▶ 空の *LI* 要素をリスト構造の中で。
- ▶ 空の *Span* 要素で、空白キャラクターに対する *ActualText* を提供。
- ▶ 空の *Div* 要素で、メタデータプロパティまたは属性を提供。
- ▶ 空の *Document* 要素で、何も内容がない 1 ページ文書を表す。
- ▶ 空の *NonStruct* 要素で、任意のタグセットを含める。

プログラミングスコープとページ境界 多くの構造要素は、ページスコープ内でのみ作成できます。グループ化要素は、複数のページにわたる場合があります、文書スコープ内でも作成できます。疑似アイテムと直接アイテムは、ページを終了または一時停止する前に閉じる必要があります。テンプレートスコープでのタグ付け操作はサポートされていません。

11.1.3 PDF 1.7 の構造要素ネスティング規則

PDF 1.x では、PDF が文書全体を内容としている場合には *Document* 要素を構造ツリーのルートとして使用するべきです。PDF が文書の断片を内容としている場合には *Part* か *Sect* をルート要素として使用するべきです。ルート要素は *PDF_begin_document()* の *tag* オプションを用いて与えることもできて便利です。構造要素ヒエラルキーに対する PDF 1.7 のタグネスティング規則を表 11.3 に挙げます。

11.1.4 PDF 2.0 の構造要素ネスティング規則

PDF 2.0 のタグネスティング規則 PDF 2.0 は、親子の組み合わせのうちどれが許されどれが禁止かについて厳しい掟を課しています。PDF 2.0 のタグネスティング規則を表 11.4 に挙げます。「内容アイテム」列は、要素がページ内容 (テキスト・画像・ベクトル画像

表 11.3 PDF_begin_item() と、さまざまな関数の tag/begin_tag オプションに対する PDF 1.7 のタグネスティング規則

アイテム	規則
内容アイテム	以下の構造種別は、ページ内容、すなわちテキストか画像かベクトルグラフィックを子に持てます： H・H1・H2・… P Lbl・LBody TH・TD Span・Quote・Note・Reference・BibEntry・Code Link・Annot Figure・Formula RB・RT・RP・WT・WP Artifact・ASpan・ReversedChars・Clip
グループ化要素	グループ化要素は、内容アイテムか ASpan か ILSE を子に持つてはいけません。すなわち、内容が作成できる前に、BLSE が作成される必要があります： Document・Part・Art・Sect・Div・BlockQuote・Caption・TOC・TOCI・Index・NonStruct・Private 以下の要素に対しては、グループ化要素の子である場合、オプション Placement=Block を推奨します：Figure・Formula・Form・Link・Annot
ブロックレベル要素	以下のブロックレベル要素は、内容アイテムを子に持つてはいけません。すなわち、内容が作成できる前に、然るべきグループ化要素か BLSE が作成される必要があります： L・LI・Table・TR・THead・TFoot・TBody 厳格規則：P 要素はグループ化要素を子に持てません。
擬似・インライン要素	擬似要素（すなわち Artifact・ASpan・ReversedChars・Clip）と、以下の ILSE は、direct=true の場合には子孫を一切持つてはいけません： Code・BibEntry・Note・Quote・Reference・Span ただしこれらの要素は、direct=false の場合には子を持つことができます。 擬似要素と、以下の ILSE の中には、direct=true の場合にはページ装飾を作成できません：Span・Quote・Note・Reference・BibEntry・Code
表要素	Table 要素は、1 個ないし複数の TR 要素を、あるいは 1 個のオプションな THead に続けてその後の 1 個ないし複数の TBody 要素と 1 個のオプションな TFoot とを、子に持てます。TBody は Table の唯一の子であるべきではありません。これに加えて Table 要素は、1 個の Caption 要素を、その最初または最後の子として持つこともできます。 TH・TD 要素は、TR・TH・TD・THead・TBody・TFoot を子に持てません。 THead・TBody・TFoot 要素は、TR 要素のみを内容とすることができ、かつ Table のみを親とすることができます。 TR は、Table・THead・TBody・TFoot のみを親とすることができます。
リスト要素	L 要素は、オプションに 1 個の Caption 要素を、また 1 個ないし複数の LI 要素を子に持てます。 LI 要素は、1 個ないし複数の Lbl または LBody 要素を、あるいは両方を子に持てます。LI は L のみを親とすることができます。LBody は LI のみを親とすることができます。
目次	TOC 要素は、1 個のオプションな Caption 要素を最初の子として、また 1 個ないし複数の TOCI および TOC 要素を（組み合わせでも）、子に持てます。 TOCI 要素は、Lbl・Reference・NonStruct・P・TOC 要素のみを内容とすることができます。 TOCI は TOC のみを親とすることができます。
ラベル要素	Lbl は、Annot・LI・Link・TOCI・BibEntry・Note のみを親とすることができます。

表 11.3 PDF_begin_item() と、さまざまな関数の tag/begintag オプションに対する PDF 1.7 のタグネ스팅規則

アイテム	規則
インタラクティブ要素：リンク・フォームフィールド・注釈	以下の構造種別は、PDF_create_field()・PDF_create_annotation() の tag オプションに対しては、あるいは Link・Annot・Form に対する親としては許容されません：表要素、ILSE、ルビ・割注要素、擬似要素。 Annot 要素はネストできません。 Link 要素はネストできません。 Form 要素は、PDF_create_field() によって自動的に作成される OBJR 要素以外の要素を子に持つてはいけません。
和文のルビ・割注	Ruby は RB・RT・RP を子に持てますが、それ以外の構造種別を子に持つてはいけません。 Warichu は WT・WP を子に持てますが、それ以外の構造種別を子に持つてはいけません。
PDF 2.0 の整合性規則	まれにしか使われない PDF 1.7 の親子タグ組み合わせに対して、PDF 2.0 のネ스팅規則との整合性のために、以下のネ스팅規則が導入されています： 非互換の変更（ネ스팅規則が厳しくなったもの）： ▶ RB・RT・RP は Ruby の子にのみなれる ▶ WT・WP は Warichu の子にのみなれる ▶ Document は Caption を子に持てない ▶ Caption は Document か Caption を子に持てない ▶ H・H1 などは Document か Part か Div を子に持てない ▶ TH・TD は Document か Caption を子に持てない ▶ direct=true によるインラインレベル要素は L か Table か Document を子に持てない

と、インタラクティブ要素のための (OBJR) を子に持つことが許されているかどうかを表しています。

表 11.4 PDF_begin_item() と、さまざまな関数の tag/begintag オプションに対する PDF 2.0 のタグネ스팅規則。特記なき限り、子要素は親要素の中に何回でも出現できます。

親要素	許される子要素	内容アイテム
構造ツリーのルート	構造ツリーのルートは、Document 要素を 1 つだけ子に持つ必要があります。また、1 個の NonStruct 要素を子に持つこともできます。ルートは、PDF_begin_document() の tag オプションを用いて与えることもできて便利です。	—
Document・DocumentFragment	Document・DocumentFragment・Part・Div・Sect・Aside・NonStruct・P・H1・…・H ¹ ・Title・Link・Annot・Form・L・Table・Figure・Formula・Artifact	—
Part	この Part 要素の Part でない最も近い祖先要素の子として許されているすべての構造種別。	—
Div	この Div 要素の Div でない最も近い祖先要素の子として許されているすべての構造種別。	—
Sect	DocumentFragment・Part・Div・Sect・Aside・NonStruct・P・H1・…・H ¹ ・Title・Lb1・Link・Annot・Form・FENote・L・Table・Caption・Figure・Formula・Artifact	—
Aside	Document・DocumentFragment・Part・Div・Sect・NonStruct・P・H1・…・H ¹ ・Lb1・Link・Annot・Form・FENote・L・Table・Caption ¹ ・Figure・Formula・Artifact	○

表 11.4 PDF_begin_item() と、さまざまな関数の tag/begintag オプションに対する PDF 2.0 のタグネ스팅規則。特記なき限り、子要素は親要素の中に何回でも出現できます。

親要素	許される子要素	内容アイテム
NonStruct	この NonStruct 要素の NonStruct でない最も近い祖先要素の子として許されているすべての構造種別。	—
Title	Part · Div · Aside · NonStruct · P · Lbl · Em · Strong · Span · Link · Annot · Form · Ruby · Warichu · L · Table · Caption ¹ · Figure · Formula · Artifact この要素がグループ化要素として使用されている場合には、以下の要素も子として許されますが、内容アイテムと相互排他です：FENote	○ ×
Sub	NonStruct · Lbl · Em · Strong · Span · Link · Annot · Form · Ruby · Warichu · L · Figure · Formula · Artifact	○
P	NonStruct · Sub · Lbl · Em · Strong · Span · Link · Annot · Form · Ruby · Warichu · L · Figure · Formula · Artifact	○
H1 · ... · H	Sect ¹ · NonStruct · Sub · Lbl · Em · Strong · Span · Link · Annot · Form · Ruby · Warichu · Figure · Formula · Artifact	○
Lbl	NonStruct · Sub · Em · Strong · Span · Link · Annot · Form · Ruby · Warichu · Figure · Formula · Artifact	○
Em · Strong · Span	NonStruct · Sub · Lbl · Em · Strong · Span · Link · Annot · Form · Ruby · Warichu · Figure · Formula · Artifact	○
Link	Div · Sect · NonStruct · Sub · Lbl · Em · Strong · Span · Annot · Ruby · Warichu · Figure · Formula · Artifact この要素がグループ化要素として使用されている場合には、以下の要素も子として許されますが、内容アイテムと相互排他です： DocumentFragment · Part · Aside · P · Hn · H · Title · Form · FENote · L · Table · Caption	○ ×
Annot	Div · Sect · NonStruct · Sub · Lbl · Em · Strong · Span · Link · Annot · Ruby · Warichu · Figure · Formula · Artifact この要素がグループ化要素として使用されている場合には、以下の要素も子として許されますが、内容アイテムと相互排他です： DocumentFragment · Part · Aside · P · Hn · H · Title · Form · FENote · L · Table · Caption	○ ×
Form	Div · NonStruct · Lbl · Caption ¹ · Artifact この要素がグループ化要素として使用されている場合には、以下の要素も子として許されますが、内容アイテムと相互排他です： Part · L · Table · Figure · Formula	○ ×
Ruby	NonStruct · RB · RT · RP	○
Warichu	NonStruct · WT · WP	○
RB · RT · RP · WT · WP	NonStruct · Sub · Em · Strong · Span · Link · Annot · Form · Artifact	○
FENote	Part · Div · Sect · Aside · NonStruct · P · Sub · Lbl · Em · Strong · Span · Link · Annot · Form · Ruby · Warichu · L · Table · Figure · Formula · Artifact この要素がグループ化要素として使用されている場合には、以下の要素も子として許されますが、内容アイテムと相互排他です： DocumentFragment · Caption	○ ×

表 11.4 PDF_begin_item() と、さまざまな関数の tag/begintag オプションに対する PDF 2.0 のタグネ스팅規則。特記なき限り、子要素は親要素の中に何回でも出現できます。

親要素	許される子要素	内容アイテム
<i>L</i>	NonStruct · L · LI · Caption ¹ · Artifact	—
<i>LI</i>	Div · NonStruct · Lbl · FENote · LBody · Artifact	○
<i>LBody</i>	Part · Div · Sect · Aside · NonStruct · P · H1 · … · H ¹ · Sub · Em · Strong · Span · Link · Annot · Form · Ruby · Warichu · FENote · L · Table · Caption ¹ · Figure · Formula · Artifact	○
<i>Table</i>	NonStruct · TR · THead ¹ · TBody · TFoot ¹ · Caption ¹ · Artifact	—
<i>TR</i>	NonStruct · TH · TD · Artifact	—
<i>TH · TD</i>	Div · Sect · NonStruct · P · H1 · … · H ¹ · Lbl · Em · Strong · Span · Link · Annot · Form · Ruby · Warichu · FENote · L · Table · Figure · Formula · Artifact	○
<i>THead · TBody · TFoot</i>	NonStruct · TR · Artifact	—
<i>Caption</i>	Part · Div · Sect · Aside · NonStruct · P · H1 · … · H ¹ · Sub · Lbl · Em · Strong · Span · Link · Annot · Form · Ruby · Warichu · FENote · L · Table · Figure · Formula · Artifact この要素がグループ化要素として使用されている場合には、以下の要素も子として許されませんが、内容アイテムと相互排他です：DocumentFragment	○ ×
<i>Figure</i>	ブロック要素として使用される場合：Part · Div · Sect · Aside · NonStruct · P · H1 · … · H ¹ · Lbl · Em · Strong · Span · Link · Annot · Form · Ruby · Warichu · FENote · L · Table · Caption ¹ · Figure · Formula · Artifact	○
<i>Formula</i>	Part · Div · Aside · NonStruct · P · H1 · … · H ¹ · Sub · Lbl · Em · Strong · Span · Link · Annot · Form · Ruby · Warichu · FENote · L · Table · Caption ¹ · Figure · Formula · Artifact この要素がグループ化要素として使用されている場合には、以下の要素も子として許されませんが、内容アイテムと相互排他です：Sub	○ ×
<i>Artifact</i>	(すべての構造種別。H · Caption は 1 回だけ出現できます。)	○

1. この要素は親要素の中に 1 回だけ出現できます。

PDF 2.0 のさらなるタグネ스팅規則 2 つの構造種別の組み合わせを制限するこの主要なタグネ스팅規則のほかにも、PDF 2.0 は、表 11.5 の通り、シンプルな親子関係だけでない制限を課しています。PDF 2.0 の必須事項は PDFlib によって強制されますが、推奨事項は強制されません。

表 11.5 PDF_begin_item() と、さまざまな関数の tag/begintag オプションに対する PDF 2.0 のさらなるタグネ스팅規則

構造種別	要請
以下のPDF 2.0の必須規則はPDFlibによって強制されます：	
<i>Document · DocumentFragment</i>	それぞれの Document か DocumentFragment 構造要素の中では、見出し構造要素はすべてが Hn かすべてが H のどちらかである必要があります。
<i>Annot</i>	1 つの Annot 構造要素の中の注釈はすべて同じ注釈種別である必要があります。Annot をリンク注釈かフォームフィールドに対して使用してはいけません。

表 11.5 PDF_begin_item() と、さまざまな関数の tag/begin tag オプションに対する PDF 2.0 のさらなるタグネスティング規則

構造種別	要請
Form	フォームフィールドは Form か Artifact 要素の中に入れる必要があります。
Link	1 つの Link 要素はリンク注釈を 1 つだけ子に持つことができ、複数を持つことはできません。
Ruby	1 つの Ruby 要素は、1 つの RB 要素を子に持つ必要があり、その後には 1 つの RT 要素か並び RP・RT・RP を持つ必要があります。
Warichu	1 つの Warichu 要素は並び WP・WT・WP を子に持つ必要があります。
Caption	Caption 要素はその親要素の中の最初か最後の要素である必要があります。キャプションの数は 1 個を超えることはできません。
H、それとも H1・H2・...	1 つの Document か DocumentFragment 要素とともに使用できるのは、番号なしの見出し要素 H (強い構造化) か番号付きの見出し要素 H1・H2 など (弱い構造化) のどちらかであり、両者を混在させることはできません。
以下の PDF 2.0 の推奨規則は PDFlib によって強制されません：	
H	H 要素はつねにその親要素の中の最初の構造要素であるべきです。
Title	Title 要素はその親であるグループ化要素の中で 1 回だけ出現するべきで、かつそのグループ化要素の中の内容の先頭か先頭付近に出現するべきです。
FENote	テキストが脚注か後注であると見なされるためには、それを囲っている内容から脚注か後注への参照があるべきです。そのような参照を実現するには、Link 要素を使用してそのリンク注釈の中の構造移動先を通じて参照するか、あるいは構造要素内で ref オプションを使用します。
Sub	Sub 要素が使用される場合には、その Sub 要素の親であるブロックレベル要素の中の他の内容もすべて Sub 構造要素群で囲われているべきです。

11.1.5 カスタム構造種別・ロールマップ・カスタムタグセット

表 11.1 に挙げた標準構造種別に加えて、カスタム構造種別名も使用することが可能です。カスタム構造種別は通常、構造種別名をローカライズしたり (たとえばドイツ語 *Abbildung* が *Figure* にマップ)、アプリケーション独自の種別名で作業したり (たとえば *Normal* が *P* にマップ) するために用いられます。カスタム構造種別を含んだ文書の再利用を可能にするためには、そのカスタム名を、標準構造種別の集合の中の、それと正確に同等、ないしおおむね同等のものへマップする必要があります。また、標準構造種別を、その意味付けを変更するために他の標準種別へ再マップすることも可能です。カスタム要素種別は、インライン要素・擬似要素へはマップできません。標準種別への要素マッピングは *rolemap* 文書オプションで実現できます。たとえば

```
p.begin_document("tagged.pdf",
    "tagged=true lang=en rolemap={ {Heading H1} {Subhead H2} {Paragraph P} }");
```

カスタム構造種別は、それがロールマップされた、対応する標準構造種別のネスティング規則に従う必要があります。

Acrobat におけるロールマップ ロールマップは、Acrobat DC で以下のように表示・編集できます：

- ▶ 「表示」 → 「表示切り替え」 → 「ナビゲーションパネル」 → 「タグ」を選択し、「タグ」パネルの上端にあるメニューボタンをクリックして、ドロップダウンリストから「ロールマップを編集」を選択

PDF 2.0 でタグセット PDF 2.0 は、2 つの定義済みのタグセットをサポートしており、*pdftagset=1.7* か *pdftagset=2.0* かで選択できます。これらの定義済みのタグセットのほか、カスタムのタグセットを定義することも可能です。これらは統一資源識別子 (URI) の形式の名前空間によって識別されます。カスタムのタグセットは *tagsets* 文書オプション内で定義する必要があります。各タグセットに対して、対応する名前空間 URI とロールマップを与える必要があります。例：

```
tagsets={
  {namespace={http://www.pdflib.com/tagsets/foo} rolemap={{MyTag P} ...} }
  {namespace={http://www.pdflib.com/tagsets/bar} rolemap={...} }
}
```

このやり方でカスタムタグセットが宣言されている前提で、対応するタグを、*PDF_begin_item()* などタグ付け操作に与えることができます：

```
tag={tagname=MyTag namespace={http://www.pdflib.com/tagsets/foo}}
```

11.1.6 ページ装飾

実質的な内容とページ装飾 ページのさまざまな内容は、以下のカテゴリーのいずれかにあてはまります：

- ▶ 実質的な内容。文書作成者によって、その文書の意味を伝えるために作成されています。その文書の論理構造ツリーは、実際の内容を成すオブジェクト群を記述しているほか、注釈も含む場合があります。
- ▶ 実質的なページ内容に貢献しておらず、ページネーションまたはレイアウト目的のために作成されているグラフィックまたはテキストオブジェクトを、ページ装飾といいます。ページ装飾は、構造ツリー内に含まれておらず、スクリーンリーダーによって読み上げられません。

ページ装飾を標識することは、アクセシビリティを向上させるために強く推奨され、PDF/UA-1 では必須です。典型的なページ装飾は、反復されるヘッダー・フッター、ページ番号、背景画像、その他各ページ上で反復されるアイテムです。

PDF 1.7 ではページ装飾は直接要素としてのみ表現できます。すなわちそれらは構造ツリーに含まれません。PDF 2.0 ではページ装飾は構造ツリーの一部になることもできます (オプション *direct=false*)。PDF 1.7 の直接ページ装飾を異なり、PDF 2.0 の構造ツリー内のページ装飾はコンテキスト (構造ツリー内の他のエントリー群) を持っています。これは行番号などに対して意味を持ちます。構造ツリー内のページ装飾は、注釈にページ装飾としてタグ付けする場合にも必要です。

Acrobat におけるページ装飾 ページ装飾は、Acrobat DC で以下の方式のいずれかでチェックできます：

- ▶ 「ツール」 → 「アクセシビリティ」 → 「読み上げ順序」を選択すると、ページ上の内容要素を表示または編集できます。ページ装飾は、Acrobat の「読み上げ順序」ツール内では「背景」と呼ばれています。これは構造要素とは異なり、このツールをアクティブにした際にふちとタグ名を用いて視覚化されません。

- ▶ ページ装飾を識別するには、「表示」→「表示切り替え」→「ナビゲーションパネル」→「コンテンツ」を選択します。「コンテンツ」パネルには、すべてのページ内容が、それぞれの構造種別名または「ページ装飾」のいずれか適切なほうとともに一覧表示されます。ページ装飾は読み上げられませんので、ページ上に、対応する番号付きブロックはありません。ただし、この一覧内のページ装飾をクリックすると、その対応する、ページ上の内容要素がハイライトされます。
- ▶ ページ装飾を検索：「表示」→「表示切り替え」→「ナビゲーションパネル」→「タグ」を選択し、「タグ」パネルの上端にあるメニューボタンをクリックして、「検索 ...」の後、ドロップダウンリストから「ページ装飾」を選択します。ページ装飾は文書構造の一部ではありませんので、タグナビゲーションパネルにも順序パネルにも、対応するエントリーはありません。
- ▶ 「表示」→「読み上げ ...」を有効にすると、Acrobat がページ上の構造要素を読み上げることができるようになります。ページ装飾は読み上げられません。

内容をページ装飾として指定 ページ装飾は、PDFlib で、`PDF_begin_item()` 内で *Artifact* タグ名で指定することができます：

```
id = p.begin_item("Artifact", "");
```

あるいは、ページ装飾は、短縮タグ付けで、すなわちさまざまな関数の *tag* オプションで指定することもできます：

```
p.fit_textline(text, x, y, "tag={tagname=Artifact}");
```

ページ装飾は、BLSE がカレントでアクティブでないときのみ作成することを推奨します。しかし、アプリケーションによってはこれはつねに可能とは限りませんので、ページ装飾が作成される際には、PDFlib はカレントでアクティブな要素を自動的に中断し、そのページ装飾の後でそれを再びアクティブ化します。なお、タグネ스팅規則 (303 ページ「11.1.3 PDF 1.7 の構造要素ネ스팅規則」参照) は、ILSE 内にページ装飾を許容しません。

ページ装飾を分類 非実質的なページ内容は、*Artifact* 擬似タグで識別し、*artifacts* オプションの以下のキーワードのうちのいずれか 1 つに従って分類する必要があります：

- ▶ **Pagination** (ページネーション) ページ装飾：ランニングヘッドやページ番号といったページ機能群。ページネーションページ装飾は、*artifacts* オプションと、キーワード *Header*・*Footer*・*Watermark* のうちのいずれか 1 つでさらに分類できます。
- ▶ **Layout** (レイアウト) ページ装飾：罫線や表シェーディングといったタイポグラフィまたはデザイン要素。
- ▶ **Page** (ページ) ページ装飾：補助、たとえばトンボやカラーバーなど。
- ▶ **Background** (背景) ページ装飾：ページの幅または高さいっぱい、あるいは構造要素の寸法いっぱい伸びる画像または色付き領域。

ページネーションページ装飾と背景ページ装飾では、どのページ辺ないしページ辺群 (*Top/Bottom/Left/Right*) にそのページ装飾が付着しているかを指定する *Attached* オプションが使えます。

以下の例は、下位種別 *Header* を持つページネーションページ装飾を作成します：

```
id = p.begin_item("Artifact",
    "artifacts=Page Pagination artifactssubtype=Header Attached={Top Left}");
```

自動ページ装飾タグ付け すべてのページ内容は、構造要素かページ装飾のいずれかとしてタグ付けされているべきですので、PDFlib は自動的に、ある種のグラフィックな要素群にタグ付けを行います。以下の装飾要素は、タグ付き PDF モードでは自動的に、*artifacctype=Layout* を持つ *Artifact* としてタグ付けされます：

- ▶ ブロック装飾：*PDF_fill_*block()* によって作成されるすべての装飾要素、すなわち *backgroundcolor*・*bordercolor* プロパティに従って作成される描線と塗り。
- ▶ 範囲枠装飾：範囲枠オプション *fillcolor*・*shading*・*strokecolor* に従って作成される範囲枠長方形。
- ▶ 表装飾：自動表タグ付けが有効の場合には (317 ページ「11.2.1 自動表タグ付け」参照)、表の罫線と網掛け、すなわちオプション *fill*・*stroke*・*showborder*・*showgrid* に従った描線と塗り。2 個目以降の表インスタンスにおいては、*header*・*footer*・*caption* オプションによって生成された表行もすべてページ装飾として標識されます。
- ▶ テキスト行ページ装飾：*leader*・*shadow*・*showborder*
- ▶ テキストフローページ装飾：*leader*・*shadow*・*showborder*・*showtabs*
- ▶ テキスト装飾：*underline*・*overline*・*strikeout*

すべてのタグ付き PDF 機能同様、自動ページ装飾タグ付けはページスコープ内でのみ動作します。

11.1.7 テキスト処理

言語指定 タグ付き PDF では、テキストの自然言語は明示的に指定されるべきです：これによって、スクリーンリーダーがその文書を読み上げる際に適切な言語へ切り替わられるようになります。この自然言語は、さまざまなレベルで指定することができます：

- ▶ *PDF_begin_document()* の *lang* オプションを、主要言語、すなわちその文書の全体としての自然言語を指定するために設定する必要があります。この指定は、ページ内容のみならず、しおりや注釈といったインタラクティブ要素をもカバーします。この言語指定は、*PDF_end_document()* で変更することもできます。取り込んだ PDF 文書に基づいて言語を選びたい場合などにこれは有用でしょう。
- ▶ この文書言語は、個別の構造要素について、*PDF_begin_item()* の *lang* オプションか、さまざまな関数の *tag* オプションを用いてオーバーライドすることもできます。
- ▶ PDF 2.0 では、注釈の自然言語を指定するために *PDF_create_annotation()* にも *lang* オプションを与えることが可能です。

テキストとして符号化されながら、しかし自然言語の一部ではない内容、たとえばプログラミングコード、音符、書体サンプル、数式などは、空の言語コードを用いるべきです。例：*lang={}*

ハイパーテキスト文字列は、言語を指定するための Unicode 言語識別子を含むこともできます。しかし現在の PDF ビューア群はこれに対応していません。Unicode 言語識別子は、以下の並びから成っています：

- ▶ Unicode エスケープキャラクター U+001B (2 バイト)
- ▶ ISO 639 言語コード 1 個 (2 ASCII バイト)。例：*en*・*ja*・*de*
- ▶ オプションに、ISO 3166 国コード 1 個 (2 ASCII バイト)。例：*US*・*JP*
- ▶ Unicode エスケープキャラクター U+001B (2 バイト)

16 進表記での例：


```
001B656E5553001B      (=enUS)
001B7A68001B          (=zh)
001B6465001B          (=de)
```

Unicode 対応言語では、この 2 個の ASCII キャラクターはまとめて 1 個の Unicode 値とする必要があります：

```
\u001B\u6465\u001B      (=de)
```

C 言語では、この 2 個の ASCII キャラクターはまとめて 1 個の Unicode 値とする必要があります。 `charref=true` の場合、この並びは以下のように表現できます：

```
&#x001B;&#x6465;&#x001B;    (=de)
```

単語間を空白キャラクターで区切る 単語は空白キャラクター (U+0020) で区切るべきです。 `autospace` オプションを用いると、テキスト出力関数群のうちのいずれかへのそれぞれの呼び出しの後に、自動的に空白キャラクターを生成させることができます。

ハイフネーション ハイフネーション、すなわち 1 つの単語を行末で 2 つの部分に分割することは、ハードハイフン (U+002D) ではなく、ソフトハイフンキャラクター (U+00AD) を用いて表現される必要があります。このソフトハイフンキャラクターが用いられていれば、Acrobat は、テキストを検索する際に、ハイフネーションされた単語を正しく再結合 (デハイフネーション) することができます。テキストがソフトハイフン U+00AD を含んでいるときには、テキストエンジンは、U+00AD に対するグリフがそのフォント内で得られる場合にはそれを、そうでないなら U+002D を用います。そのフォントが U+00AD と U+002D に対して別々のグリフを持っている場合には、ソフトハイフン U+00AD をテキスト内で使用すれば、ハイフネーションのためのタグ付き PDF の要件を満たすには充分です。このシンプルなハイフネーション方式にフォントが対応しているかどうかをチェックするには以下のコード断片を使用します：

```
gid1 = (int) p.info_font(font, "glyphid", "unicode=0x00AD"); /* ソフトハイフン */
gid2 = (int) p.info_font(font, "glyphid", "unicode=0x002D"); /* ハイフン-マイナス */

if (gid1 != -1 && gid2 != -1 && gid1 != gid2)
    System.err.println("フォントは別個のソフトハイフングリフを持っています");
else
    System.err.println("フォントは別個のソフトハイフングリフを持っていません");
```

そのフォントが U+00AD に対する別個のグリフを持っていない (あるいは別のハイフネーションキャラクターが用いられている) 場合には、そのハイフンを、U+00AD を内容とする *ActualText* 属性を持った *Span* か *ASpan* でタグ付けする必要があります (ただし PDF 1.4 ではこれは可能ではありません)。テキストフローを用いて複数行テキストが作成される際には、この *ActualText* は、指定された *hyphenchar* に自動的に割り当てられます (かつ *autospace* は抑止されます)。

テキスト行に対して必要な *ActualText* を付けることは以下のように達成できます：

- ▶ `PDF_fit_textline()` でオプション `tagtrailinghyphen` を指定すれば、然るべき *ActualText* が付き、かつ *autospace* が抑制されます。このオプションのデフォルトは U+00AD ですので、そのテキストがハイフンとして U+00AD を用いている場合には、正しいタグ付けが自動的に行われます。

- ▶ フォントがU+00ADに対する別個のグリフを持っていない場合には、然るべきフォールバックフォントからのソフトハイフンを用いてこれを修正することもできます。以下のフォント読み込みオプションを用います：

```
fallbackfonts={{fontname=AuxiliaryFont forcechars=x00AD}}
```

- ▶ 然るべき *ActualText* を手動で割り当てるには以下のようにします。ただし、これが必要なのは、そのフォントが U+00AD と U+002D に対して 2 個の別々のグリフを持っていない場合のみであることに留意してください（このコードは PDF 1.5 以上でのみ動作します。PDF 1.4 の場合にはオプション *direct=false* を追加してください）：

```
p.set_option("charref=true");  
p.fit_textline("-", x, y, "tag={tagname=ASpan ActualText=&#x00AD;}");
```

11.1.8 代替記述・置換テキスト・略語拡張

タグ付き PDF は、追加情報なしでは容易に読めない画像・テキストのアクセシビリティを向上させる機能群を提供しています。

代替記述 (Alt) テキストへ自然に翻訳されないアイテムに、代替記述を *Alt* オプションを通じて割り当てることができます。たとえば画像・数式や、*contents* オプションを持たない注釈などです。

この代替記述は、スクリーンリーダーが読める 1 個のまるごとの単語ないし句から成るべきです。この記述の末尾は、スクリーンリーダーがそれを後続テキストと合体させてしまわないよう、ピリオドか空白キャラクターかいずれか適切なほうとするべきです。「この画像の内容は ...」といった先頭句を代替記述に入れることは避けることを推奨します。この *Alt* 値は、その構造要素とそのすべての子の記述を提供します。*ASpan* 擬似要素を用いると、構造要素の一部分に *Alt* を割り当てることができます。

たとえば、企業ロゴの画像を、*Alt* オプションを通じて記述してもよいでしょう：

```
p.fit_image(image, x, y, "tag={tagname=Figure Alt={Kraxi企業ロゴ}}")
```

置換テキスト (ActualText) テキストに類似のアイテムであっても、それが何らかの非標準的な方式で表現されている場合には、それに等価な代替テキストを *ActualText* タグ付けオプションを通じて割り当てることができます。たとえばスワッシュキャラクターやドロップキャップを持った図や、ピクセル群を用いて単語を表現している画像などです。一方で、スキャンされたページに対する OCR テキストは、*ActualText* ではなく不可視テキスト（すなわち *textrendering=3*）として与えるべきです。

この代替テキストは、人がその内容を見た時に見えるものと同等の 1 個ないし複数のキャラクターを内容とするべきです。この *ActualText* 値は、その構造要素とそのすべての子に対する代替としての役割を果たします。*ASpan* 擬似要素を用いると、構造要素の一部分に *ActualText* を割り当てることができます。

たとえば、対応する Unicode 値が全く得られない記号 *flower* グリフに対して、適切な *ActualText* を割り当てることで、このグリフが実際にはビュレットキャラクター U+2022 として用いられていることを明らかにすることができます：

```
p.fit_textline("&.flower;", x, y, "tag={tagname=ASpan ActualText={&#x2022;} } ...");
```

代替テキストと置換テキストに対するネ스팅規則 *Alt*・*ActualText* 属性を用いる際には、以下の規則に従う必要があります：

- ▶ **Alt** と **ActualText** は、対象構造要素配下の下位ヒエラルキー全体を覆いますので、その構造要素の構造ヒエラルキー内のいずれかの祖先がすでに **Alt** 属性か **ActualText** 属性を含んでいる場合には、どちらの属性も許容されません。
- ▶ **Alt** か **ActualText** 属性を持つ要素は、内容アイテムか、1 個ないし複数の **Link** 要素を子に持つ必要があります。この属性が **Link** 要素に適用される場合には、それは、内容アイテムか、**PDF_create_annotation()** によって作成された 1 個ないし複数の **OBIR** 要素を子に持つ必要があります (320 ページ「リンクとその他の注釈種別」参照)。そうでないと、その属性がどのページ上で読み上げられるべきなのかを決定することが不可能になります。この規則は当該要素自体に適用されます: 内容アイテムを持つ子要素を持っていても充分ではありません。

Acrobat における代替・置換テキスト 構造要素の **Alt**・**ActualText** 属性は、Acrobat DC で以下のように表示できます:

- ▶ 「表示」→「表示切り替え」→「ナビゲーションパネル」→「タグ」を選択し、ヒエラルキー内の構造要素を右クリックして、「プロパティ...」を選択すると、オブジェクトプロパティダイアログが表示されます。「タグ」タブに、「実際のテキスト」・「代替テキスト」属性が表示されています。

略語拡張 (E) 略語と頭字語には、拡張テキストを、**tag** オプションの **E** サブオプションで割り当てることができます。この拡張テキストは、スクリーンリーダーが読める 1 個のまるごとの単語ないし句から成るべきです。略語が拡張テキストを何も持たない場合でも、**E** 属性を、テキストから読み上げへの変換処理を支援するために与えることができます。たとえば、用語 **IBM** は、拡張テキスト **IBM** (空白キャラクターをはさんで) をそれに割り当てられて持つことができます。

以下のコード断片では、拡張テキスト **Mister** が略語 **Mr.** に割り当てられています:

```
p.fit_textline("Mr.", x, y, "tag={tagname=ASpan E={Mister} } ...");
```

11.1.9 印刷ストリーム順序と論理読み取り順序

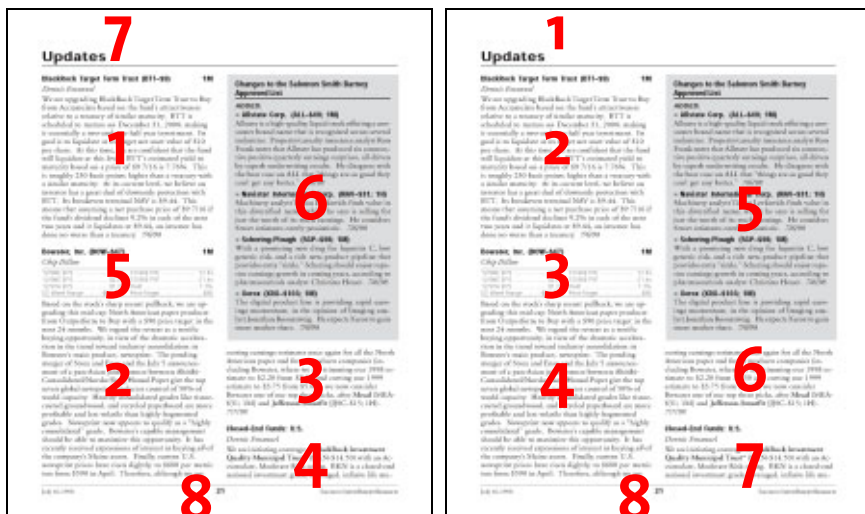
PDF には、内容の順序付けについて、根本から異なる 2 種類の概念があります。図 11.2 に、2 段組のメインテキストに、表 1 個がはさまり、灰色背景上のサマリー 1 個が挿入された、さらにヘッダーとフッターもあるサンプルページを図示します。

PDFlib API メソッド呼び出しの順序は、ページ内容ストリーム内の PDF テキスト・描画演算子の順序 (Acrobat では「生の印刷ストリーム順序」といいます) を決定付けます。ページ内容は、制御アプリケーションに都合のいい任意の方式で作成されるものであるので、これはただの技術的な順序付けであり、意味付け上の重要性は必ずしもありません。この印刷ストリーム順序は Acrobat の「順序」・「コンテンツ」パネルで表示されます。

論理読み取り順序は、人がテキストを読む順序です。これは、スクリーンリーダーと Acrobat の読み上げ機能によって用いられる順序付けを決定します。この論理読み取り順序は、論理構造ツリーによって決定されます。タグ付き PDF では、すべての内容が意味付け上正しい順序でタグ付けされていることが必須であり、すなわち、その構造ヒエラルキーはそのページ内容を人が読む順に含んでいる必要があります。正しいタグ付けにより、スクリーンリーダーが内容を論理順に表現できるようになります。ページ装飾は構造ツリーの一部分ではありませんので、論理読み取り順序からは除外されています。

Acrobat における読み取り順序と印刷ストリーム順序 論理読み取り順序は、Acrobat DC で以下の方式でチェックできます:

図 11.2
論理読み取り順序
(左)と印刷ストリー
ム順序(右)



- ▶ 「表示」→「表示切り替え」→「ナビゲーションパネル」→「タグ」を選択し、要素の順序を上から下へチェックします。この順序付けは、望む読み取り順序を正確に反映しているべきです。
- ▶ 「表示」→「読み上げ...」を有効にすると、Acrobat にページ内容を、その文書内で指定されている読み取り順序で読み上げさせることができます。

「順序」・「コンテンツ」パネルでは、ページ内容を印刷ストリーム順序で一覧表示しています。

内容を論理読み取り順序で作成 多くの状況においてうまくいく自然な方式は、1 個の構造要素のすべての構成部分をシーケンシャルに生成し、ついで論理シーケンス内の次の要素へ移っていくことです。専門的にいうと、構造ツリーは単一の深さ優先トラバーサルによって作成され、ページ内容を作成する PDFlib 関数は、内容が読まれる順に呼び出されます。

PDFlib では、内容を任意の順序で作成しながらもなお構造ヒエラルキーを論理順に作成するためのいくつかの方法が可能です。これらの方式は 327 ページ「11.2.5 内容を順序にとらわれず作成」で説明します。

11.1.10 Adobe Acrobat におけるタグ付き PDF の不具合

この項では、私達がタグ付き PDF 出力を Adobe Acrobat で試験していくなかで得た知見を述べます。表 11.6 に、Acrobat におけるバグと動作不安定を、以下の機能ごとにまとめて挙げます：

- ▶ Acrobat のアクセシビリティチェッカー: Acrobat のアクセシビリティチェッカーを利用すると、スクリーンリーダーのような支援技術に対するタグ付き PDF 文書の適合性を調べることができます。
- ▶ Acrobat の読み上げ機能: タグ付き PDF は、Acrobat の読み上げ機能を向上させます。
- ▶ Acrobat のタグパネル内の「検索...」機能を使ってページ装飾とマークなし内容を検索することもできます。

表 11.6 タグ付き PDF 関連の Acrobat DC の不具合

説明・推奨・回避策

Acrobatのアクセシビリティチェッカー

Alt 属性が Figure タグについて無視される。

アクセシビリティチェッカーが、チェックマーク記号を持った種類チェックボックスのフォームフィールドについて「Character encoding - failed」と警告を発するが、読み上げ機能はそのフィールド内容を完璧に読み上げる。

Acrobatの読み上げ機能

タグ付きのページが PDI で配置されており、かつページ装飾のみを子に持っている場合にも、読み上げ機能はその配置されているページの内容を読み上げる。

その他のAcrobatの機能

タグパネル内の「検索 ...」機能が、ページ装飾を誤ってマークなし内容として報告する。例：表装飾。対照的に、これらのアイテムはコンテンツパネル内では「コンテナ<ページ装飾>」として正しく表示される。

11.2 タグ付き PDF 作成の高度なトピック

11.2.1 自動表タグ付け

`PDF_fit_table()` は、表内容のために `PDF_add_table_cell()` に与えられた情報に基づいて、生成される表のために適切なタグを自動的に作成することができます。`PDF_fit_table()` の tag オプションで `tagname=Table` を用いると、表 11.7 に説明するとおりの自動表タグ付けが行われます。

表 11.7 `PDF_fit_table()` の tag オプションと自動タグ付け

<code>PDF_fit_table()</code> の tag オプション	結果
<code>tagname=Table</code>	推奨アプローチ：自動表タグ付けが行われます。 <code>tagname=Table</code> のかわりに、Table ヘッローラマップされたカスタムタグを用いることもできます。
<code>tagname=Artifact</code>	キャプションを含め、表内容全体がページ装飾としてマークされます。BBox 属性が自動的に追加されます。
他の任意の <code>tagname</code>	Table 構造は一切作成されず、そのセル内容は tag オプションで指定された要素の子として追加されます。擬似アイテムとインラインアイテムは <code>PDF_fit_table()</code> に対しては許容されません。
tag オプションが与えられていない	自動タグ付けなし。tag オプションが個別の <code>PDF_add_table_cell()</code> への呼び出しに対して与えられた場合には (fit* オプションのうちのいずれかの対するサブオプションとして)、そのセル内容に対する、対応する構造要素が作成されますが、表構造は一切作成されません。

注 自動表タグ付けは、PDFlib の表エンジンが用いられる場合にのみ威力を発揮します。手作業で作成された表に正しくタグを付けることは可能ではありますが、この処理は、クライアントアプリケーション側で表構造に関する詳しい知識を必要とします。表行 / 列構造のみならず、ヘッダーセルと表行 / 列スパンに関する関連情報が必須です。空セルにもタグを付ける必要があります。

Acrobat で表タグを視覚化 表要素の構造は、Acrobat DC で以下のように視覚化できません：

- ▶ 「表示」 → 「アクセシビリティ」 → 「読み上げ順序」を選択。表要素がハイライトされ、表の左上隅近くの小さな番号で標識されます。表サマリーが存在する場合には、それはこの小さな番号の後に表示されます。
- ▶ 表の左上隅にある番号または構造種別名を選択し、「読み上げ順序」ダイアログ内の「テーブルエディター」をクリック。表構造が縦横の線で視覚化されます。「テーブルエディターオプション」ダイアログ内で、Acrobat に、表セルの種別に従って TH/TD アイコンを表示させることができます (図 11.3 参照)。
- ▶ 表セルを右クリックして、「テーブルセルのプロパティ ...」を選択すると、そのセル種別 (ヘッダーセルなら TH、データセルなら TD)、`scope` 属性、`rowspan`・`colspan` 属性、`header/ID` 値をチェックできます。

なお、Acrobat においてテーブルエディターで作業するにあたっては以下の制約があります：

Travel Expense Report				
	Meals	Hotels	Transport	subtotals
Madrid				
13-Aug-2012	37.74	112.00	45.00	
14-Aug-2012	27.28	112.00	45.00	
subtotals	65.02	224.02	90.00	379.02
Paris				
15-Aug-2012	96.25	109.00	36.00	
16-Aug-2012	35.00	109.00	36.00	
subtotals	131.25	218.00	72.00	421.25
Totals	196.27	442.00	162.00	827.27

図 11.3
タグ付き PDF 表のための
Acrobat のテーブルエディター
がヘッダー (TH)・データ
(TD) セルを表示

- ▶ 表セルを視覚化する線が、誤った位置に表示されることがあります。
- ▶ 表が、Table へロールマップされたカスタム構造要素 (標準要素 Table ではなく) を用いていると、Acrobat はテーブルエディターをアクティブにしません。
- ▶ テーブルエディターは、表内に *Caption* 要素が存在していても表示しません。
- ▶ 表セルが縦書きテキスト (たとえば *orientate=east* または *west* によるテキスト行) を含んでいると、このセルとその右隣セルはテーブルエディターで表示されませんが、ただしそれらは論理構造ツリー内には存在し、それらの内容はページ上で見えます。

自動作成される表タグと属性 自動表タグ付けは、ページスコープ内でのみ、かつ、*PDF_fit_table()* で *tag={tablename=Table}* を用いた場合にのみ働きます。動作の詳細は以下のとおりです：

- ▶ それぞれの論理表に対して *Table* 要素が1つずつ作成されます。1個の表が2個以上のインスタンスに分割されている場合、*Table* 要素は1つだけ作成されます。*PDF_fit_table()* の *tag* オプションに *Summary* サブオプションが与えられている場合には、*Table* 要素に *Summary* 属性が追加されます。
- ▶ *PDF_fit_table()* で *caption* オプションが指定された場合には、最初の表インスタンスの中に *Caption* 要素が作成されます。グループ化要素として、*Caption* はいかなる内容アイテムをも子に持つことを許しませんので、この *caption* オプションの *tag* サブオプションを与えることによって、*Caption* の子として構造要素を指定する必要があります。この要素がこのキャプションの実際の内容を保持できます。各 *Table* 要素に対して許される *Caption* 要素は1つだけですので、2個目以降の表インスタンスの中で生成されたキャプションは *Artifact* として標識されます。
- ▶ それぞれの表行に対して1個の *TR* 要素が作成されます。最初の表インスタンスの中では、*PDF_fit_table()* の *header* オプション内で指定されている表行は *THHead* でラップされ、*footer* オプション内で指定されている表行は *TFoot* でラップされます。それ以外のすべての表行は、ヘッダーかフッターが存在する場合には *TBody* でラップされます。ヘッダー表行グループとフッター表行グループの出現は1回だけである必要がありますので、2個目以降の表インスタンスの中では、ヘッダー表行とフッター表行は *Artifact* として標識されます。

- ▶ 表セルはそれぞれ、`PDF_add_table_cell()`の`tag`オプションの`tagname`サブオプションに従って、`TH` (表ヘッダー) か `TD` (表データ) のいずれかの要素でラップされます。このオプションが指定されていない場合には、そのセル種別は以下のように選択されます：
 - ▶ セルに対する `Scope` 属性は `TH` を強制します (たとえ `tagname=TD` が指定されていても)。
 - ▶ カレントセルの `id` を指定した `Headers` サブオプションを持つ `tag` オプションを別のセルが含んでいる場合には、カレントセルは強制的に `TH` になります (たとえ `tagname=TD` が指定されていても)。
 - ▶ そのセルが、`PDF_fit_table()` の `header` オプションで指定されているとおりの表ヘッダーの一部分である表行に含まれている場合には、それは `TH` でラップされ、また、`Scope=Column` が追加されます。
- ▶ `PDF_add_table_cell()` が呼び出されていない表セルそれぞれについて、空の `TD` 要素が作成されます。
- ▶ `TH`・`TD` 要素は、`PDF_add_table_cell()` の `rowspan`・`colspan` オプションに従って、適切な `RowSpan`・`ColSpan` 属性を得ます。`tag` オプションの `RowSpan`・`ColSpan` サブオプションは使えません。
- ▶ `Table`・`TH`・`TD` 要素には、適切な `Width`・`Height` 属性が割り当てられます：`Table` 要素には `BBox` 属性も割り当てられます。
- ▶ 他の表セル属性は、`PDF_add_table_cell()`の`tag`オプションに対してサブオプションとして与えることが可能です。以下のオプションは許容されません：`RowSpan`・`ColSpan`・`Height`・`index`・`parent`・`Width`。
- ▶ 表行とセルは、`PDF_add_table_cell()`への呼び出しの順序にかかわらず、左上セル (すなわち列1・行1) から右下セルへジグザグ順に出力されます。
- ▶ 装飾的な各種表要素は、自動的に `artifacctype=Layout` の `Artifact` としてタグ付けされます。すなわち、表セル・表行・列・表全体の罫線とシェーディング (塗り / 描線)、範囲枠の塗りと罫線、`showborder` の罫線、`debugshow`・`showcells`・`showgrid` を通じて制御される視覚化支援です。
- ▶ 同じ表についての `PDF_fit_table()`への2回目以降の呼び出しにおいては、自動タグ付けがすべての関連するタグを処理しますので、`tag` オプションは無視されます。

クックブック 自動表タグ付けのためのコードサンプルが、PDFlib クックブックの `pdfua` カテゴリー内の `table_pdfua1`・`invoice_pdfua1` トピックにあります。

表セルにタグと属性を追加 短縮タグ付けを、表キャプションか表セルか、または表セルの内容に適用することができます。`PDF_add_table_cell()` に `tag` オプションを与えると、以下の状況において有用です：

- ▶ 表セルが、ヘッダー行に含まれていないか、`Scope` 属性を持たない場合、`tagname=TH` を用いてそれを強制的にヘッダーセルにすることもできます (`TD` データセルではなく)。
- ▶ リンクに対して必要な正しいタグ構造を作成：詳しくは 322 ページ「表セル内でリンクにタグ付け」を参照してください。

`PDF_add_table_cell()` の `tag` オプションのサブオプションに対しては、以下の制約が課されます：

- ▶ 以下のオプションは使えません：`ColSpan`・`Height`・`index`・`parent`・`RowSpan`・`Width`。

- ▶ **tagname** オプションは、表セルの種別を指定するために、値 **TH** か **TD** のいずれかのみを持つことができます。しかし、**tag** オプションをネストすることによって子孫タグを指定することもできます。

表セルの内容にタグと属性を追加 `PDF_add_table_cell()` の以下のオプション（または `caption` オプションの対応するサブオプション）にサブオプションとして **tag** を与えることもできます：

`fitannotation`・`fitfield`・`fitgraphics`・`fitimage`・`fitpath`・`fitpdipage`・`fittextflow`・`fittextline`

これは以下の状況において有用です：

- ▶ 表セルの内容に対して下位構造を指定。この **tag** オプションは、その表セルの **TH** か **TD** 要素の子要素を作成します。自動表タグ付けが有効の場合には、**tag** オプションに対して、**tagname** について以下の値は許容されません（言い換えれば、表のネストには対応していません）：

`Table`・`TR`・`TH`・`TD`・`thead`・`TBody`・`TFoot`

- ▶ 以下に挙げる表の属性は、自動的に作成されることができませんが、アクセシビリティ目的のために必要な場合があります。これらはユーザーによって与えられる必要があります：

- ▶ 表内の1個ないし複数の **TH** セルを参照する **TD** セル（すなわちこれは、`PDF_fit_table()` の **header** オプションの一部ではないヘッダーセルを参照します）に対しては、**Headers** オプションを与える必要があります。

- ▶ いずれかの **Headers** オプション内で参照されている **TH** セルに対しては、**Id** オプションと **Scope=Row** オプションを与える必要があります（**TH** 列ヘッダーセルに対しては **Scope=Column** は自動的に作成されます）。

- ▶ キャプションは任意の内容を子とすることができ、その内容自体もタグ付けされることがあります。たとえば、以下のオプションリストは、ネストされた **P** 要素の中のテキスト行1個を内容とする **Caption** 要素を作成します：

```
caption={ fittextline={tag={tagname=P title={出張旅費報告書}} ... } ... }
```

11.2.2 インタラクティブ要素にタグ付け

リンク・注釈・フォームフィールドも、アクセシブルにする必要があります。その対応するインタラクティブ要素を、構造ツリー内で、構造ツリー内の正しい位置に表現する必要があります。

クックブック タグ付きのリンクを作成するためのコードサンプルが `starter_pdfua1` サンプルにあります。`image_with_link_pdfua1` トピックは、背景画像付きのリンクを生成します。`table_of_contents_pdfua1` トピックは、目次とともに、**TOC**・**TOCI** 構造要素と、動作するリンクを生成します。

リンクとその他の注釈種別 注釈は、アクセシビリティのために以下のアイテムを要請します（図 11.4 参照）：

- ▶ **Link** 要素（リンク注釈に対して）または **Annot** 要素（その他すべての注釈種別に対して）は、次の2つのアイテムのためのコンテナとして働きます。**Alt** または **ActualText** オプションを与えることによって代替記述か置換テキストを与えることもできます。**Link**

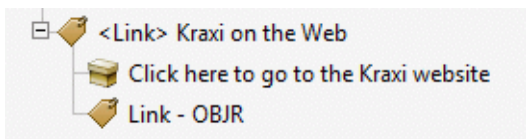


図 11.4
構造ツリー内における
アクセシブルなリンクの表現

要素の *Alt* 属性はそのリンクの目的を記述すべきです。リンクのターゲットがカレント文書内に位置している場合には (*GoTo* アクション)、その *Link* 要素はさらに *Reference* 要素の内容となるべきです (オプション *direct=false* を用いて)。

- ▶ インタラクティブ要素を表現するテキストを、このコンテナ要素の内側に作成する必要があります。テキストが全く必要ない場合には、この要素はスキップすることができます。たとえばテキスト注釈の場合などです。注釈がラスター画像かベクトルグラフィックによって表現されている場合には、これは Artifact としてタグ付けされるべきです。この場合、このリンクの代替テキストは、そのグラフィックとリンクの両方を記述することになります。さまざまな内容作成関数の *matchbox* オプションを用いることによって、次の要素のための形状情報を用意することを推奨します。*matchbox* オプションを与える時点におけるカレントのアクティブな要素は構造アイテムであるべきです。すなわちオプション *direct=true* は許されません。
- ▶ 注釈は、*PDF_create_annotation()* を用いて作成する必要があります。注釈に加えて、このメソッドは、その注釈のための、種別 *OBJR* (オブジェクト参照) の、対応する構造要素を生成します。リンク注釈に対しては、*PDF_create_annotation()* の *contents* オプションを与えるべきです (これは PDF/UA-1 では必須です)。その他の注釈種別については、*PDF_create_annotation()* の *contents* オプション、または *ActualText* タグ付けオプションを用いて作成する必要があります。 *usematchbox* オプションを用いると、2 番目のステップで作成された視覚内容の形状を簡便に与えることができます。

この 2 番目と 3 番目のアイテムは、どちらを先に作成してもかまいません。上記の注釈の要件は、*PDF_add_table_cell()* のオプション *fitannotation* を用いて表セル内に作成される注釈に対してもあてはまります。

明示的にリンクにタグ付け 以下のコード断片は、この必須の 3 つのアイテムを持ったインタラクティブリンクを作成します (生成されるタグ構造を図 11.4 に示します) :

```
/* 親Link要素を作成 */
id_link = p.begin_item("Link", "Title={Kraxi on the Web} Alt={Kraxi on the Web}");

/* このリンクを表現する可視内容を作成 */
p.fit_textline("Click here to go to the Kraxi website", x, y,
    "matchbox={name={kraxi}} fontsize=14 font=" + font);

/* URIアクションを作成 */
action = p.create_action("URI", "url={http://www.kraxi.com}");

/* 名前付き範囲枠「kraxi」上にLink注釈を作成 */
p.create_annotation(0, 0, 0, 0, "Link",
    "action={activate=" + action + "}"
    "usematchbox={kraxi} contents={Kraxi Inc. Webサイトへのリンク}");

p.end_item(id_link);
```

テキストフローオプションリスト内のタグを用いてリンクにタグ付け テキストフローオプションリスト内で範囲枠機能とタグを用いることによって、テキストフロー内でリンクに正しくタグ付けを行うことが可能です。276 ページ「テキストフローの範囲枠に Web リンクを追加」で示した一般的なイディオムを使い、以下のコード断片を用いてリンク注釈に対して然るべきタグを追加できます。テキストフローは必要な数の範囲枠長方形を自動的に生成しますので、ページ上に 1 つの名前付き範囲枠のすべてのインスタンスを生成してタグ付けするには `PDF_create_annotation()` を 1 回呼び出せば充分です。

```
final String text =
"Visit our <tag={tagname=Link} matchbox={name=kraxi}>Web site<tagend matchbox=end>";
...
tflow = p.create_textflow(text, optlist);
result = p.fit_textflow(tflow, 0, 0, 50, 70, "fitmethod=auto");
...
/* URIアクションを作成 */
act = p.create_action("URI", "url={http://www.kraxi.com}");

/* 名前付き範囲枠 (群)「kraxi」上にリンク注釈を作成 */
p.create_annotation(0, 0, 0, 0, "Link",
    "action={activate " + act + "}" +
    "usematchbox={kraxi} contents={Link to Kraxi Inc. Web site}");
```

1 つの範囲枠が複数の長方形から成ることもあります。たとえばハイフネーションによる場合や、`matchbox` オプションの複数のインスタンスが同じ名前を与えた場合です。この場合には PDFlib はデフォルトでは複数の有効領域 (その範囲枠の各長方形ごとに 1 つずつ) から成るただ 1 つのリンク注釈を生成します。あるいは `PDF_create_annotation()` のオプション `singlelink=false` を使用することによって複数のリンク注釈を生成することも可能です。その場合にはそれぞれの注釈はただ 1 つの長方形から成ることになります。ただしこれは PDF 2.0 では許されません。

クックブック テキストフロー内にタグ付きのリンクを作成するためのコードサンプルが、PDFlib クックブックの `pdfua` カテゴリの中の `textflow_pdfua1` トピックにあります。

表セル内でリンクにタグ付け 表セル内のリンクは、上述のタグ構造を必要とします。ただしこれは、構造要素 `TH/TD`・`Link`・`OBJR` と、さらにおそらく内容を、正しくネストする必要がありますので、やや複雑かもしれません。これを達成するには、`PDF_add_table_cell()` に `tag` オプションを与えてそのネスト機能を利用する必要があります。以下のオプションリストは、表セルにテキスト 1 行とリンク注釈 1 個を入れます。それを囲う `TD` 要素は外側の `tag` オプションで与えられ (`TD` は表エンジンによって自動的に作成されますので、外側の `tagname` サブオプションは省略できます)、そして `Link` 要素は内側の `tag` オプションで与えられます。最後に、必要な `OBJR` 要素は、`PDF_create_annotation()` と同等の働きをする `fitannotation` オプションによって自動的に作成されます：

```
fittextline={font=... fontsize=25 fillcolor=blue}
annotationtype=Link fitannotation={contents={Kraxi home page} action={activate ...}}
tag={tagname=TD tag={tagname=Link}}
```

フォームフィールドにタグ付け フォームフィールドはアクセシビリティのために以下のアイテムを必要とします (図 11.5 参照)：

- ▶ `Part` 構造要素を用いて、関連するフォームフィールド群のグループをグループ化することも可能です。

- ▶ **Div** 構造要素を、以下に挙げる個々のフォームフィールドの要素群のコンテナにすることを推奨します。
- ▶ フィールドの目的を記述したテキストを **P** 構造要素で囲い、それを **Caption** 構造要素の子とする。
- ▶ チェックボックスとラジオボタンについては、そのチェックボックス/ラジオボタンとその対応するキャプションを囲うもう1つの **Div** 要素を作成することを推奨します。**Caption** 構造要素は論理的にたいていフィールドより前にあるので、それは構造ヒエラルキー内で **Form** 要素より前に来ます。ページ上での順序が逆であってもそうなります。
- ▶ フォームフィールドは **PDF_create_field()** を用いて作成する必要があります。それは **P** 要素の子である **Form** 要素の子であるべきです。このメソッドは、フィールドだけでなく、この **Form** 要素の中に種別 **OBJR** (オブジェクト参照) の構造要素をも生成します。**Alt** か **ActualText** オプションを与えることによって、代替記述か置換テキストを与えることも可能です。**PDF_create_field()** の **tooltip** オプションを与えることにより、フィールドのアクセシビリティを向上させるべきです (これは PDF/UA-1 では必須です)。ラジオボタンを作成する際には、**Form** 要素は、ラジオボタングループのための **PDF_create_fieldgroup()** の中では必要なく、個々のボタンを作成するための **PDF_create_field()** の中でのみ必要です。

以下のコード断片は、テキストフィールドに対して推奨される構造を、図 11.5 に示すとおり生成します。**PDF_fit_textline()** の **tag** オプションを使って、短縮タグ付けをネストして用いて、**Caption** 構造要素と **P** 構造要素を生成しています。**PDF_create_field()** の中で短縮タグ付けを使って、**P** 構造要素と **Form** 構造要素をネストさせて生成しています：

```
id_Div = p.begin_item("Div", "");

labeltext = "Enter name:";
p.fit_textline(labeltext, x1, y1, "tag={tagname=Caption tag={tagname=P}} ...");

p.create_field(x2, y2, x3, y3, "name", "textfield",
    "tag={tagname=P tag={tagname=Form}} tooltip={" + labeltext + "} " +
    "bordercolor={gray 0} font=" + font);

p.end_item(id_Div);
```

上記のフォームフィールドの要件は、**PDF_add_table_cell()** のオプション **fitfield** を用いて表セル内に生成されるフォームフィールドにも課されます。

クックブック すべての種類のタグ付きフォームフィールドを作成するコードサンプルが、PDFlib クックブックの pdfua カテゴリーの form_fields_pdfua1 トピックにあります。

タグ付きしおり しおりには、通常のページ指向の移動先に加えて、構造要素を割り当てることもできます。このようなしおりを**タグ付きしおり**といい、Acrobat はこれのために追加の機能を提供しています。Acrobat でタグ付きしおりを右クリックすると、機能「ページを削除」と「ページを抽出」が利用可能となっており、これらはその構造化要素を含んでいるページないしページ群に対して動作します。タグ付きしおりは、しおりと構造要素との間に接続を作成します。この接続を作成できる方法は2つあります：

- ▶ **PDF_create_bookmark()** でしおりを作成し、そのハンドルを、**PDF_begin_item()** の **bookmark** オプションに、またはさまざまな関数の **tag** オプションに与える：

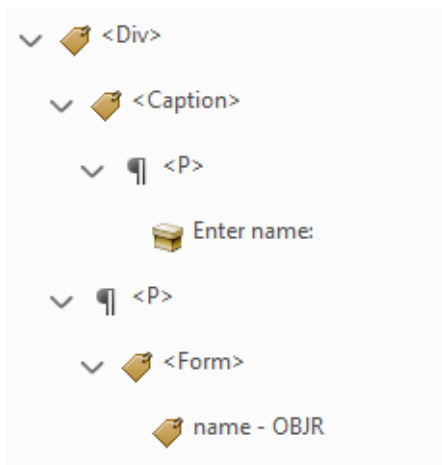


図 11.5
構造ツリー内におけるフォームフィールドの表現

```
bm = p.create_bookmark("第1章", "");
id = p.begin_item("H1", "Title={第1章} bookmark=" + bm);
p.fit_textline(text, x, y, "");
p.end_item(id);
```

この方式は短縮タグ付けでも用いることができます：

```
bm = p.create_bookmark("第1章", "");
p.fit_textline(text, x, y,
  "tag={tagname=H1 Title={第1章} bookmark=" + bm + "}");
```

この方式の難点は、しおりテキストが、その構造要素とその内容が作成される前には得られていなければならないという点です。これは、参照される構造要素のほうに構造ツリー内で上にあった場合には困るでしょう。

- ▶ **PDF_begin_item()** で構造アイテムを作成し、そのハンドルを **PDF_create_bookmark()** の **item** オプションに与えます。アイテムのハンドルのかわりに、キーワード **current** を、**PDF_create_bookmark()** が呼び出される時点におけるカレントの構造要素を指し示すショートカットとして与えることができます：

```
id = p.begin_item("H1", "Title={第1章} ");
bm = p.create_bookmark("第1章", "item=current");
p.fit_textline(text, x, y, "");
p.end_item(id);
```

この方式には、しおりテキストが、その構造要素とその内容が作成される時点で得られればよいという利点があります。ただし、これは短縮タグ付けでは使えません。

タグ付きしおりは、オープン構造要素を参照することのみ可能であり、擬似アイテムまたはインラインアイテムを参照することはできません。しおりの移動先が構造要素と一致するようにするのはクライアントコード側の役割です（そうしないと、Acrobat でしおりをクリックしたらその要素へジャンプせず、その文書内の別の場所へジャンプしてしまいます）。関連付けられる構造要素が複数のページにわたっている場合には、そのしおりはこの範囲内の最初のページを指し示すべきです。

11.2.3 インタラクティブ要素にタグ付けする PDF 2.0 の機能

インタラクティブ要素をページ装飾として PDF 1.7 では、ページ装飾はマーク付き内容アイテムとしてのみ表現できるので、インタラクティブ要素をページ装飾として分類することはできません。PDF 2.0 では、ページ装飾を構造ツリー内に含めることができ、それを用いてインタラクティブ要素をページ装飾としてタグ付けすることが可能です。以下のコード断片 (*compatibility=2.0* が必須) は、リンク注釈をページ装飾として作成します：

```
p.create_annotation(100, 100, 300, 200, "Link",
    "tag={tagname=Artifact direct=false}");
```

構造移動先 PDF 1.7 では移動先は、文書内の 1 つのページの上の視覚的な位置を指し示します。構造移動先は、PDF 2.0 の機能であり、文書ヒエラルキー内のいずれかの構造要素を指し示すものですので、物理的な位置から独立です。この技法は 2 つの手順を必要とします：

- ▶ **elementid** タグ付けサブオプションで構造要素に一意な文字列を割り当てます。これはリンクやしおりの移動先として働きます：

```
p.fit_textline("Introduction",
    50, 700, "tag={tagname=H1 elementid={Intro}} fontsize=24");
```

- ▶ **GoTo** アクションで構造要素を指名して移動先として使用します：

```
action = p.create_action("GoTo", "destination={type=fitwindow structdest={Intro}}");
```

この要素 ID はターゲット要素に割り当てるのは使用前でも使用後でもかまいません。構造移動先は **GoTo** アクションに限らず、しおりの注釈など、通常の移動先を使えるすべての場合に使用できます。構造移動先を使って名前付き移動先を定義することも可能です。

11.2.4 リストにタグ付け

リストは、関連する項目群をまとめるために使われます。これは以下の構造要素によって表現されます：

- ▶ 以下のすべての構造要素を内容とする **L** 要素 1 個。**ListNumbering** オプションを用いると、**Lbl** 要素内で用いられる付番系列を指定することができます。この **ListNumbering** オプションは、**Lbl** 要素がない場合でも、スクリーンリーダーのために有用でしょう。
- ▶ オptional な **Caption** 要素 1 個。**Caption** はグループ化要素ですので、内容アイテムを内容とすることはできず、他の構造要素のみを内容とすることができます (**P** など)。
- ▶ 以下を内容とする、1 個ないし複数のリスト項目 (**LI**)：
 - ▶ オptional な、ピュレットや番号などを持ったラベル (**Lbl**) 1 個。
 - ▶ そのリスト項目の実際の内容を持った **LBody** 要素 1 個。**LBody** は、内容アイテムか、ネストされたリストも含めて他の構造要素のいずれかを内容とすることができます。

明示的にリストにタグ付け 以下のコード断片は、1 個のキャプションと 3 個の項目を持ったリストを作成します。それぞれのリスト項目の頭には、ラベルとしてタグ付けされるピュレットキャラクター U+2022 が付きます (生成されるタグ構造を図 11.6 に示します)：

```
id_list = p.begin_item("L", "ListNumbering=Disc");
```

```
/* Caption要素とP要素の両方を一回で生成 */
```

```
p.fit_textline("The following kinds of fruit are available:",
```

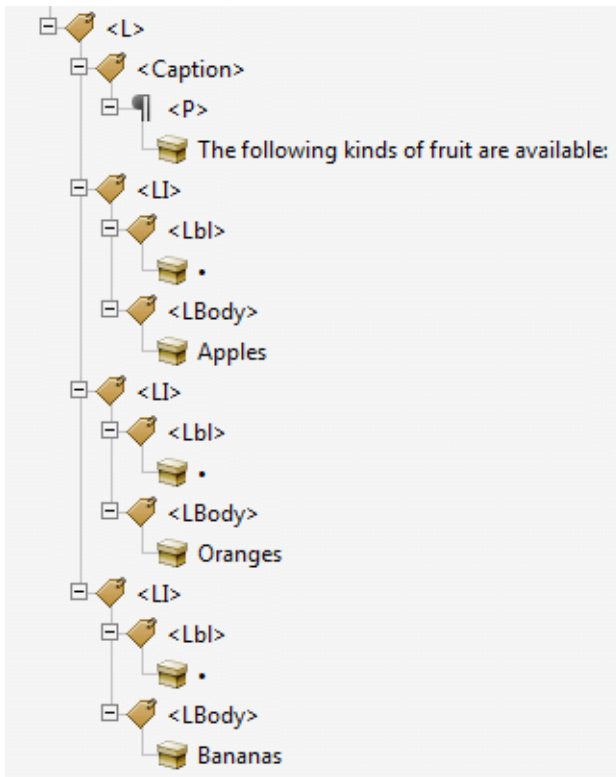


図 11.6
アクセシブルなリストの
構造ツリー内における表現

```

    x1, y, "tag={tagname=Caption tag={tagname=P}}");
    y -= leading;

    id_listitem = p.begin_item("LI", "");
    p.fit_textline("&#x2022;", x1, y, "tag={tagname=Lbl}");
    p.fit_textline("Apples", x2, y, "tag={tagname=LBody}"); y -= leading;
    p.end_item(id_listitem);

    id_listitem = p.begin_item("LI", "");
    p.fit_textline("&#x2022;", x1, y, "tag={tagname=Lbl}");
    p.fit_textline("Oranges", x2, y, "tag={tagname=LBody}"); y -= leading;
    p.end_item(id_listitem);

    id_listitem = p.begin_item("LI", "");
    p.fit_textline("&#x2022;", x1, y, "tag={tagname=Lbl}");
    p.fit_textline("Bananas", x2, y, "tag={tagname=LBody}"); y -= leading;
    p.end_item(id_listitem);

    p.end_item(id_list);

```

テキストフローオプションリスト内のタグを用いてリストにタグ付け 図 11.6 の構造は、あるいは、テキストフローオプションリスト内の以下のタグを用いて生成することも可能です（見やすいようインデントしています）：

```

<tagbegin={tagname=L ListNumbering=None}>
  <tagbegin={tagname=Caption tag={tagname=P}}>

```

```

    The following kinds of fruit are available:\n
<tagend>
<tagbegin={tagname=LI}>
  <tagbegin={tagname=Lbl}>&#x2022;<tagend>
  <tagbegin={tagname=LBody}>Apples\n<tagend>
<tagend>
<tagbegin={tagname=LI}>
  <tagbegin={tagname=Lbl}>&#x2022;<tagend>
  <tagbegin={tagname=LBody}>Oranges\n<tagend>
<tagend>
<tagbegin={tagname=LI}>
  <tagbegin={tagname=Lbl}>&#x2022;<tagend>
  <tagbegin={tagname=LBody}>Bananas\n<tagend>
<tagend>
<tagend>

```

クックブック タグ付き PDF リストを作成するためのコードサンプルが、PDFlib クックブックの pdfua カテゴリ内のトピック list_pdfua1 にあります。

11.2.5 内容を順序にとらわれず作成

314 ページ「11.1.9 印刷ストリーム順序と論理読み取り順序」で述べたように、構造ヒエラルキー内の要素群を作成する際に論理読み取り順序のとおりにすることは不可欠です。アプリケーションが、論理読み取り順序とは異なる順序でページ内容を（たとえば、段組の相互関係にかかわらずつねに上から下へ）処理する場合には、いくつかの PDFlib 機能を用いて、構造ヒエラルキー内で正しい順序付けを保つことができます：

- ▶ 子要素群を順序にとらわれず作成するために *index* オプションを用いる。これは、新規構造要素がその親の中へ挿入される位置を変更します。
- ▶ 構造要素群を順序にとらわれず作成するために *parent* オプションを用いる。これは、新規構造要素が挿入される親要素を変更します。
- ▶ 構造ヒエラルキー内で前後へジャンプするために *PDF_activate_item()* メソッドを用いる。これを用いると、構造ヒエラルキー内のいずれかの要素に対して、構造要素または内容アイテムをさらに追加することができます。

これらの方式について詳しくは後述します。

クックブック 要素を順序にとらわれずタグ付けするためのコードサンプルが、PDFlib クックブックの pdfua カテゴリ内のトピック tag_out_of_order_pdfua1*parallel_columns_pdfua1 内にあります。

子要素群を順序にとらわれず作成 構造要素内の子要素群を順序にとらわれず作成するためには、構造ツリー内の場所を指定するために、*PDF_begin_item()* の *index* オプションか、あるいはさまざまな関数の *tag* オプションの *index* サブオプションを用いることができます。以下のコード断片は、テキストフラグメント群を逆順に出力しながら、構造ツリー内における順序を正しくするために、それぞれの新規テキストフラグメントを親要素の新たな最初の子 (*index=0*) として挿入します。それぞれの新規要素が親の新たな最初の子として挿入されていますから、結果として論理順は作成順の反対になります：

```

p.fit_textline("三", x, y, "tag={tagname=P index=0}");
y += leading;
p.fit_textline("二", x, y, "tag={tagname=P index=0}");
y += leading;
p.fit_textline("一", x, y, "tag={tagname=P index=0}");

```


カレントでアクティブなタグの、その親要素内における番号をクエリーするために、`PDF_get_option()` とその `activeitemindex` か `activeitemkidcount` キーワードを用いることができますので、後で構造ツリー内のこの場所へ戻ってくるのが可能です。以下のコード断片は、新規要素を、保管していた番号の位置にある要素の後に挿入します：

```
nextindex = p.get_option("activeitemindex", "") + 1;
```

...同一レベル上の要素群をさらに作成...

```
p.fit_textline(text, x, y, "tag={tagname=P index=" + nextindex + "}");
```

構造要素群を順序にとらわれず作成 子要素群を、カレント位置ではなく、構造ツリー内のどこか他の場所に作成するためには、`parent` オプションを用います。これは、まだ閉じられていない構造要素を指し示す必要があります。短縮タグ付けで作成された要素は、同一のメソッド呼び出しの中で作成されて閉じられてしまいますので、これは `parent` オプションのターゲットとしては使えません。カレントでアクティブなタグのIDをクエリーするために、`PDF_get_option()` とその `activeitemid` キーワードを用いることができますので、後で構造ツリー内のこの場所へ戻ってくるのが可能です：

```
parent_id = p.get_option("activeitemid", "");
```

```
...
```

```
p.fit_textline(text, x, y, "tag={tagname=P parent=" + parent_id + "}");
```

さらに柔軟性を得る方法として、`parent` オプションと `index` オプションを組み合わせることもできます。`PDF_suspend/resume_page()` を用いると、ページに割り込んで、別のページで作業を続け、その後その一時停止したページに戻ってさらに内容を追加することができます。

複雑なレイアウトのためにアイテムをアクティブ化 複雑な非線形のページレイアウトのための構造情報の作成を支援するために、PDFlib はアイテムのアクティブ化という機能を提供しています。これを呼び出すと、開発者が複数の構造分枝の進捗を同時進行させ、かつそれぞれの分枝が 1 個ないし複数のページにわたる可能性があるような状況において、以前に作成された構造要素をアクティブ化することができます。この技法によって恩恵を受ける典型的な状況は以下のとおりです：

- ▶ 1つのページ上に複数の段組
- ▶ メインテキストに割り込む挿入部、たとえばまとめやその他非線形のテキストアイテム
- ▶ 段組どうしの間に配置される表・イラスト

`PDF_activate_item()` メソッドを用いると、構造ツリーの別々の分枝どうしの間を行ったり来たりすることが可能になります。「論理順」アプローチでは、クライアントアプリケーションはページ内容を論理順に構築する必要があり、これはたとえそれを視覚順に作成するほうが簡単であっても変えることはできません。これと対照的に、アイテムのアクティブ化を用いると、内容を視覚順で作成することが可能になります（あるいはアプリケーションにとって便利ないかなる順序でも）。この技法は、内容が複数ページにわたる場合にも適用できます。

Acrobat での諸問題を回避するため、`PDF_activate_item()` を呼び出した直後には、内容アイテムを追加するべきではなく、他の構造要素だけを追加するべきです。

カレントでアクティブな構造要素をクエリー `parent·index` オプションや `PDF_activate_item()` を使うには、カレントでアクティブな構造要素とその子に関する情報が必要になります。このステータス情報は、アプリケーション側で把握しておいてもよいですが、PDFlib からクエリーすることも可能です。メソッド `PDF_get_option()` でキーワード `activeitemid·activeitemindex·activeitemkidcount·activeitemname·activeitemstandardname` を用いると、カレント要素の ID・番号・子要素数・名前・標準名（それがロールマップされている場合）が得られます。

11.3 タグ付き PDF ページを PDI で取り込む

クックブック タグ付き PDF 文書からページを取り込むためのコードサンプルが、PDFlib クックブックの pdfua カテゴリ内のトピック `clone_pdfua・merge_and_stamp_pdfua1` にあります。

タグ付き PDF モードでは、タグ付き PDF 文書からのページは、その構造要素タグ群と一緒に取り込まれます。タグ付き PDF モードで `usetags=true` を用いて取り込まれたタグ付き PDF 文書のページのことを「タグ付きページ」と呼ぶことにしましょう。このステータスをクエリーするには、`PDF_info_pdi_page()` の `tagged` キーワードを用います。タグ付きページの取り込みは、以下の説明のように動作します。

タグ付き PDF 文書を開く `PDF_open_pdi_document()` が、取り込まれる文書がカレントの PDF/A-1a/2a/3a または PDF/UA モードと互換かどうかをチェックしたうえで、取り込まれる文書の構造ツリーとロールマップを読み取ります。

`usetags` オプションが `false` の場合には、その文書の構造情報は無視され、その文書からは何のタグも取り込まれません。なお、`PDF_open_pdi_document()` がオブジェクトスコープ内で、すなわち `PDF_begin_document()` の前に呼び出された場合には、`usetags` のデフォルトは `false` です。

注 属性クラスとクラスマップは取り込まれません。

入力文書言語を転写 もし PDF 文書の大半ないしすべてのページが、PDI を用いて取り込まれたものである場合には、その入力文書内にその文書言語エンタリーが存在しているならば、それを転写することを推奨します。この文書言語を転写するには、pCOS と以下のコード断片を用います：

```
if (p.pcos_get_string(indoc, "type:/Root/Lang").equals("string"))
{
    inputlang = p.pcos_get_string(indoc, "/Root/Lang");
    optlist += " lang=" + inputlang;
}
```

```
p.begin_document(filename, optlist);
```

タグ付きページを開く `PDF_open_pdi_page()` が、取り込まれたページの内容を構成する構造要素群を選択し、そのページ上に存在するタグをフィルターします。取り込まれたページ上の、取り込まれた構造下位ツリーの最上位を形成する 1 個ないし複数の構造要素が選択されます。`usetags` オプションが `false` の場合には、そのページの構造情報は完全に無視されます。

取り込まれた文書のロールマップ内のエンタリー群は、もしそのページ上で、その対応する要素が用いられているならば、出力文書のロールマップへコピーされます。衝突しあうロールマップエンタリー群（すなわち、あるカスタムタグがすでに、生成文書のロールマップ内で、あるいはそれ以前に取り込まれた文書内で、別の標準タグへマップされている）は無視されます。ただし PDF/UA モードでは、衝突しあうロールマップエンタリーを持つページは拒絶され、すなわち `PDF_open_pdi_page()` への呼び出しは失敗します。

タグなし PDF 内容を取り込む タグなし文書のページと、`usetags=false` を用いて開かれたページも、処理できますが、それらは自動的にページ装飾として標識されます。

無効なタグ構造を持つ文書を取り込む PDFlib は、303 ページ「11.1.3 PDF 1.7 の構造要素ネスティング規則」で説明した通りの、ISO 32000-1 で課されているタグネスティング規則に関する厳格なチェックを行います。これらのチェックは、取り込まれる文書に対しても、そして取り込まれるページから作成されるタグ構造に対しても適用できます。取り込まれたページ内のネスティング規則は、デフォルトではチェックされません。しかし、これらのチェックは、`PDF_open_pdi_document()` の `checktags` オプションで有効にすることも可能です。`checktags=strict` の場合には、`PDF_open_pdi_page()` ですべてのタグネスティング規則が強制されます。もしも取り込まれたページの構造ヒエラルキーが、構造要素に対するネスティング規則に違反しているときは、`PDF_open_pdi_page()` への呼び出しは失敗し、`PDF_get_errmsg()` は以下のようなエラーを報告します：

```
Structure type 'Document' cannot contain content items,  
but only other structure elements
```

多数の既存の現実世界のタグ付き PDF 文書がこのタグネスティング規則に違反していますので、取り込んだ文書内のこれらの問題への対処として、以下の方式のいずれかを用いることができます：

- ▶ 取り込まれた構造ヒエラルキーのてっぺんに追加のタグを挿入する（たとえば `PDF_fit_pdi_page()` の `tags` オプションで）ことは、取り込まれたページがその文書ルートの直下に内容アイテムを子に持っているありがちな問題を直すために有用です。
- ▶ 他の諸問題は、追加のタグを挿入しても解決できません。たとえば表やリストのための構造要素が不完全なときなどです。可能であれば入力文書を直すことを考慮すべきです。これが実行可能な解決策でない場合には、`PDF_open_pdi_document()` で `checktags=none` と設定することによって、非標準のタグ構造を持つタグ付き PDF ページを取り込むこともできます。
- ▶ もしも取り込まれた構造自体は正しくて、ただ出力文書の生成された新しいタグ群と衝突しているという場合には、その新しいタグ群を適切に調整することを試みるべきです。もしこれが現実的でないならば、`PDF_begin_document()` で `checktags=none` と設定することによって、生成されたタグ付け構造内の衝突を無視することもできます。これは PDF/UA-1 モードでは許容されません。

いずれの形の `checktags=none` によって処理された非標準入力も、非標準の PDF 出力を生成するおそれがありますので、この設定は推奨しません。

取り込まれたページ内のタグをクエリーおよびチェック 状況によっては、取り込まれた構造要素群を、生成される新たな構造ヒエラルキー内に正しく統合させることが難しい場合もあります。この状況を支援するために、ページを開くことが成功した後に `PDF_info_pdi_page()` を用いると、取り込まれたタグ付き PDF ページのいくつかの特性をクエリーすることができます：

- ▶ キーワード `fittingpossible` は、そのページがカレントコンテキスト内に配置できるかどうかを報告します。その `tag` オプションを与えると、そのページを追加の最上位レベルタグとともに配置できるかどうかをチェックできます。この `tag` オプションの `tagname` サブオプションだけが評価されます：他のサブオプションは与えるべきではありません。このキーワードを用いることは、取り込まれるページ内の構造情報に関してよくわかっておらず、取り込まれた構造要素群が不適合であるせいで `PDF_fit_pdi_page()` で例外が発生することを避けたい状況において推奨されます。もしもページがこの `fittingpossible` テストで拒絶された場合には、その `tag` オプションを通じて追加のタグを挿入することを試みることもできます。

- ▶ キーワード `topleveltagcount` は、最上位レベル構造要素の数を報告します。なぜなら最上位レベルタグが複数ある場合もあるからです。なお `topleveltagcount` は、そのページ内容が何の構造要素によっても覆われていない稀な場合において、0 となることもあります。このようなページは、タグ付きでないページのように動作します。他の内容アイテムと同様、これはグループ化要素の子として配置することはできず、そのページのてっぺんに追加のタグが必要です。
- ▶ キーワード `topleveltag` は、その取り込まれたページの最上位レベル構造要素（群）を報告します。これは、取り込まれたページ構造にかぶせて追加の構造要素を挿入することができるか、あるいは挿入する必要があるかどうかを決定するために有用でしょう。
- ▶ キーワード `lang` は、すべての最上位レベルの取り込まれた構造要素（群）の `lang` 属性を報告します。これは、より高い構造要素内で `lang` 属性が必要かどうかを決める助けとなります。

タグ付きページを配置 `PDF_fit_pdi_page()` が、取り込まれたページを新たなページ上に配置し、その構造ヒエラルキーを、生成文書の構造ツリー内に統合します。その際には、カレントでアクティブな要素を、取り込まれた構造ツリーの親として用います。追加の構造要素を、`PDF_fit_pdi_page()` の `tag` オプションで作成することもできます。これは、取り込まれた構造ヒエラルキーのための新たな親としての役割を果たします。

取り込まれたヒエラルキー内に `Alt` または `ActualText` 属性が存在する場合には、生成される文書構造内のもっと上にある既存の属性との衝突が見出されたときは、この `Alt` または `ActualText` 属性は除去されます（313 ページ「代替テキストと置換テキストに対するネスティング規則」参照）。

そのページが、`PDF_open_pdi_document()` と `PDF_open_pdi_page()` で `usetags=true` を用いて開かれている場合には、それは出力文書内に 1 回しか配置できません。なぜなら、その取り込まれた構造は、出力文書の構造ヒエラルキーの一意な位置に統合される必要があるからです。ページが複数の位置において内容と構造に寄与する場合には、それを複数回開く必要があります（すなわち、別々のページハンドルが配置されます）。

未知の文書構造を持つページを取り込む際には、以下の戦略を推奨します：

- ▶ `PDF_info_pdi_page()` の `topleveltagcount` キーワードがタグカウント 0 を報告したならば、そのページは `usetags=false` を用いて取り込まれているか、空であるか、ページ装飾のみを内容としているかのどちらかです。アプリケーションによっては、このようなページは実質的な内容に全く寄与しないのでスキップすると決めることもできるでしょう。
- ▶ `PDF_info_pdi_page()` の `fittingpossible` キーワードがこのページについて 1 を返したならば、これは `PDF_fit_pdi_page()` で安全に配置できます。このテストは、そのページが `usetags=false` を用いて読み込まれているか、ページ装飾として配置される場合には不要です。
- ▶ あるいは、取り込まれたページの構造のてっぺんに追加のタグを挿入することもできます。このタグを何にするかの選択は、アプリケーションによって異なり、とりわけ、取り込まれたページが構造ツリー内に挿入される場所にある構造要素の種別に依存します。
- ▶ `PDF_info_pdi_page()` の `fittingpossible` キーワードが、追加のタグを用いてもなおそのページを拒絶するならば、アプリケーションは別のタグを試すか、あるいはそのページを諦めるかのどちらかでしょう。

以下のコード断片は、上述の戦略を実装したものです。ページが直接配置できない場合に、追加の *P* 要素を挿入しています：

```
fittingpossible = true;
additionaltag = "";

toopleveltagcount = (int) p.info_pdi_page(page, "toopleveltagcount", "");

if (toopleveltagcount == 0)
{
    /* このページは構造要素を何も含んでいません、
     * すなわち、これは空であるか、ページ装飾のみを内容としています。
     * 「Artifact」を追加しましょう。
     */
    additionaltag = "tag={tagname=Artifact} ";
}
else
/*
* ページを追加のタグなしで配置してみます。
* これがうまくいかないならば、別のタグを挿入しましょう。
*/
if (p.info_pdi_page(page, "fittingpossible", "") == 0)
{
    additionaltag = "tag={tagname=P} ";

    if (p.info_pdi_page(page, "fittingpossible", additionaltag) == 0)
    {
        fittingpossible = false;
    }
}

if (fittingpossible)
{
    p.fit_pdi_page(page, 0, 0, "adjustpage " + additionaltag);
}
else
{
    System.err.println("ページをスキップします: " + p.get_errmsg());
}
```

さまざまな用途 タグ付き PDF モードでページを取り込むことの用途は、以下のように分類することができます：これらの用途に対するオプションと要件を表 11.8 にまとめます：

- ▶ タグを温存：タグ付き PDF 文書からページを取り込んで、そのタグ群を出力へコピーします。そのページを構成する構造要素群は、新たな構造ヒエラルキーの一部となります。オプションに、取り込まれたヒエラルキーのてっぺんに追加のタグを配置することもできます。
- ▶ タグを無視：タグ付きまたはタグなし PDF 文書からページを取り込んで、そのページ全体に 1 個の新たな構造要素でタグ付けします。既存の構造要素群は破棄され、取り込まれたページの内容は単一の要素を構成し、内部構造を一切持たなくなります。
- ▶ ページ装飾として配置：タグ付きまたはタグなし PDF 文書からページを取り込みます。たとえば背景グラフィックに使う場合などです。ページは自動的にページ装飾としてタグ付けされ、既存の構造要素群は破棄されます。

表 11.8 タグ付き PDF モードでタグ付き・タグなし PDF ページを取り込み

用途	オプション	要件とコメント
タグを温存	<code>PDF.open_pdi_document()</code> と <code>PDF.open_pdi_page()</code> で <code>usetags=true</code>	<ul style="list-style-type: none"> ▶ 親¹ は、インライン要素が擬似要素であってはいけません ▶ 取り込まれたタグ付きページを配置できるのは1回だけ ▶ 取り込まれたページの最上位レベル要素（群）はネスティング規則を守る必要があります
タグを無視してカレント要素内にページを配置	<code>PDF.open_pdi_document()</code> か <code>PDF.open_pdi_page()</code> で <code>usetags=false</code> 、そして <code>PDF.fit_pdi_page()</code> で <code>tagname=Span direct=true</code>	<ul style="list-style-type: none"> ▶ 取り込まれたページの内容は、カレントでアクティブなアイテムの一部になります ▶ 非グループ化親要素のみ許容されます
ページ装飾として配置	<code>PDF.open_pdi_document()</code> か <code>PDF.open_pdi_page()</code> で <code>usetags=false</code> 、そして <code>PDF.fit_pdi_page()</code> で <code>tagname=Artifact</code>	<ul style="list-style-type: none"> ▶ カレントで有効な要素がページ装飾を子として許している必要があります ▶ PDF/UA モードでは内容は自動的にページ装飾としてタグ付けされます ▶ ページを複数回配置できます

1. `PDF.begin_item()` で、または `PDF.fit_pdi_page()` の `tag` オプションで指定された、カレントでアクティブなタグ

11.4 PDF/UA によるユニバーサルアクセシビリティ

11.4.1 PDF/UA-1 規格

注 PDF/UA 規格に関する一般情報が PDFlib ウェブサイトにあります。

注 PDF/UA に初めてふれる方には、PDF 協会が発行した文書「PDF/UA in a Nutshell」を推奨します (www.pdfa.org/pdfua-in-a-nutshell/ 参照)。

PDF/UA-1 は、PDF 文書のユニバーサルアクセシビリティを向上させるために、タグ付き PDF の要素群や、その他の文書の諸側面の詳細を仕様化したものです。PDF/UA-1 は、WCAG 2.0 (Web コンテンツアクセシビリティガイドライン群¹) を PDF 文書に適用したものと捉えることができます。

PDF/UA-1 は、PDF 1.7 と ISO 32000-1 で定義されているタグ付き PDF に基づいています。これは PDF ファイル形式に何らかの新たな機能を加えたのではなく、主に、いくつかのアクセシビリティとタグ付けの側面が PDF 1.7 ではオプションであるのを必須としたものです。またこれは、構造要素のさまざまな種別どうしの関係を明確化していません。

ISO 14289-1 で定義された PDF/UA-1 PDF/UA-1 は、PDF 文書のアクセシビリティを向上させるために以下の手段を用いています：

- ▶ 文書構造に関する特定の要件を強制。例：タグ付け規則など。
- ▶ 特定の補足情報を必須化。例：メタデータ、グラフィックのための代替テキスト。
- ▶ アクセシビリティの目的を阻害する特定の PDF 要素を防止。

PDFlib の PDF/UA-1 の実装は、以下の文書に基づいています：

- ▶ PDF/UA-1 規格 (ISO 14289-1:2014)

注 PDF 協会が発行している文書「Tagged PDF Best Practice Guide: Syntax」を推奨します (www.pdfa.org/resource/tagged-pdf-best-practice-guide-syntax/ にあります)

PDF/UA-1・PDF/A 両立文書を作成 PDF/UA 文書でありながら、同時にアーカイブングのための PDF/A にも準拠している文書を作成すると、有用なこともあるでしょう (346 ページ「12.2 PDF/A によるアーカイブング」を参照)。実際、PDF/A-1a/2a/3a を作成したい場合には、PDF/UA の要件を遵守することによって、生成される文書のアクセシビリティを向上させることを推奨します。PDF/A・PDF/UA-1 両立文書を作成するには、`PDF_begin_document()` の `pdfa・pdfua` オプションに対して適切な値を与えます。例：

```
ret = p.begin_document("combo.pdf", "pdfa=PDF/A-2a pdfua=PDF/UA-1 lang=en");
```

このような二重用途の文書を作成する際には、両方の規格に準拠する両立ファイルは、両方の規格によって課せられる要件に従う必要がありますので、留意してください。その PDF 互換レベルは、関与する規格群の PDF 互換性の最小値です：取り込まれる PDF 文書が、PDF/UA 規格と PDF/A 規格の両方を遵守している必要があります。

タグ付き PDF 出力のためには、PDF/A-1a を避けて、もっと新しい PDF/A-2a か PDF/A-3a 規格のほうで作業することを推奨します。なぜなら、PDF/UA-1 と PDF/A-1a の間にはいくつか衝突があるからです：

1. www.w3.org/TR/WCAG20/ 参照

- ▶ PDF/UA-1 は、注釈が存在する場合には、ページオプション `taborder=structure` を必須とします。しかし、この `taborder` オプションは PDF 1.5 が必須で、PDF/A-1a では使用できません。結果として、注釈は PDF/A-1a・PDF/UA-1 両立文書内では使えません。
- ▶ PDF/UA-1 では `TH` 要素に対して `scope` オプションが推奨されていますが、PDF 1.5 でのみ利用可能であり、PDF/A-1a の場合には使用できません。
- ▶ `TH`・`TFoot` 要素には PDF 1.5 が必要なので、PDF/A-1a の場合にはそれを使用して表内のヘッダー表行かフッター表行をグループ化することはできません。

クックブック PDFlib クックブック内の `invoice_pdfua1` サンプルが、PDF/UA-1・PDF/A-2a 両立文書を作成する方法を演示しています。

PDFlib は、技術的な PDF/UA の要件への違反を検出した場合には、例外を発生させます。この場合には PDF 出力は何も作成されません。表 11.9 に、PDF/UA 準拠出力を作成するための一般的要件を挙げます。

表 11.9 PDF/UA 準拠のための一般的要件

項目	PDF/UA-1 準拠のための PDFlib の要件
一般の文書要件	<p><code>PDF.begin/end_document()</code> : <code>pdfua</code> オプションを PDF/UA-1 に設定する必要があります。これはタグ付き PDF を必須とします (tagged オプションは自動的に設定されます)。</p> <p>PDF 1.7ext3 以上を要する操作 (リッチメディア注釈・ポートフォリオなど) を避ける必要があります。lang オプションが必須です。</p> <p>オプション <code>viewerpreferences</code> : サブオプション <code>displaydoctitle</code> に対して <code>true</code> のみが許容されます。</p> <p>オプション <code>permissions</code> : キーワード <code>noaccessible</code> が許容されません。</p>
タグ付け	<p>すべてのタグ付け規則 (303 ページ「11.1.3 PDF 1.7 の構造要素ネ스팅規則」参照) に従う必要があります。checktags 文書オプションを <code>none</code> に設定してはいけません。</p>
フォント	<p>フォントオプション <code>embedding</code> を <code>true</code> にする必要があります。オプション <code>unicodemap=false</code> が許容されません。埋め込みは標準 Type 1 フォントについても必須です。</p>
テキスト出力と PUA Unicode キャラクター	<p>PUA Unicode キャラクター (ロゴや記号など) が、適切な置換テキストを持つ必要があります。それを囲う内容アイテムに対する <code>PDF.begin_item()</code> の <code>ActualText</code> オプションで指定するか、その対応する出力メソッドの同等の tag オプションで指定します (353 ページ「PUA キャラクター」参照)。</p>
不可視テキスト	<p>埋め込み必須は、不可視テキスト (主として OCR の生成結果のために有用) に対してのみ用いられているフォントについてだけはあてはまりません。不可視とは <code>textrendering=3</code> です。これは、<code>optimizeinvisible</code> オプションを用いて制御することもできます。</p> <p>不可視テキストは、レンダリングされる等価物を一切持たない場合には、Artifact としてマークする必要があります。</p>
レイヤー	<p>レイヤーを使えますが、ただし <code>PDF.define_layer()</code> のいくつかのオプションを避ける必要があります (PDFlib API リファレンス参照)。</p>
外部内容	<p><code>PDF.begin_template_ext()</code>・<code>PDF.load_graphics()</code>・<code>PDF.open_pdi_page()</code> : <code>reference</code> オプションを避ける必要があります。</p>
PDF取り込み	<p><code>PDF.open_pdi_document()</code> : 取り込まれる文書が PDF/UA に準拠している必要があります。340 ページ「11.4.4 PDF/UA 文書のページを PDI で取り込む」を参照してください。</p>
メタデータ	<p><code>PDF.set_info()</code> で <code>key=Title</code> を、または <code>PDF.begin/end_document()</code> で <code>metadata</code> に与える XMP 内で <code>dc:title</code> を、空でない値とともに与える必要があります。</p>

11.4.2 タグ付けの要件

PDF/UA は、タグ付き PDF に基づいていますので、293 ページ「11.1 タグ付き PDF の基礎」で説明したタグ付き PDF の要件に従う必要があります。ただし、PDF/UA-1 はこれに加えて、タグ付けの要件を多数課することによって、アクセシビリティの向上を図っています。

意味付けの要件 ユーザーは、文書ヒエラルキーを作成し、以下に挙げる意味付け規則に従う必要があります。適切な構造要素を選ぶことは、PDF/UA 規格準拠の決定的な構成要素です。これらの側面に責任を負うのはアプリケーション側であることを理解することが重要です。なぜなら PDFlib はこれらをチェックできないからです：

- ▶ タグ付けは、その文書構造に対して適切な構造要素を用いる必要があります：それが見出しであるなら、それを見出しとしてタグ付けする必要があります。それが表であるなら、それを表としてタグ付けする必要があります。
- ▶ その文書にとって意味をなさない内容を、その文書ヒエラルキー内に含めてはいけません。それは *Artifact* としてタグ付けする必要があります。
- ▶ 構造要素群を、論理読み取り順序に配列する必要があります。これは、そのタグ群を読み順に作成することによって最も容易に達成できます。ただし、複雑なレイアウトに対しては、*PDF_activate_item()* を用いてこれを達成することもできます (327 ページ「11.2.5 内容を順序にとらわれず作成」参照)。
- ▶ 内容の色や形式やレイアウトのせいで、意図する情報がそのままではアクセシブルでない内容に、適切にタグ付けする必要があります。
- ▶ グラフィック内で表現されているテキストは、もしそれがテキストを自然言語で含まない場合には (フォントサンプルやスクリプトサンプルなど)、説明を持った *Alt* オプションを必須とします。
- ▶ 画像のキャプションを *Caption* タグでマークする必要があります。
- ▶ 内容がリストとして読まれることを意図している場合には、リスト要素 (*L*) を作成する必要があります。
- ▶ ヘッダーとフッターを *Artifact* として、*artifacttype=Pageination* と *artifactssubtype=Header* か *Footer* とともにタグ付けする必要があります。
- ▶ 論理的に一体であるグラフィカル要素群のグループに対しては、ただ1つの *Figure* を作成する必要があります。
- ▶ 脚注・後注・注ラベルと、その文書内の場所への参照に、適切に *Note* か *Reference* としてタグ付けする必要があります。

注 PDF 協会が発行した文書「Tagged PDF Best Practice Guide: Syntax」は、PDF/UA-1 のためのタグ付き PDF 構造要素の正しい使い方に関するガイダンスを提供しています。

タグごとの要件 表 11.10 に、PDF/UA-1 準拠を達成するための、タグごとに対する要件を挙げます。ここに挙げた標準種別へマップされたカスタム構造種別についても、これらの規則に従う必要があります。たとえば、*rolemap* オプション内でカスタムタグ *Illustration* が *Figure* へマップされているならば、それも *Figure* に対する条件下に置かれます。

表 11.10 PDF/UA-1 準拠のためのタグごとの要件

構造種別	PDF/UA-1 準拠のための PDFlib の要件
標準構造種別	<i>PDF_begin_document()</i> : <i>rolemap</i> オプションが標準構造種別をマップしてはいけません。
<i>Figure</i>	オプション <i>Alt</i> か <i>ActualText</i> のうちの 1 つを与える必要があります。

表 11.10 PDF/UA-1 準拠のためのタグごとの要件

構造種別	PDF/UA-1 準拠のための PDFlib の要件
<i>Formula</i>	オプション Alt を与える必要があります。
<i>Table</i>	表要素は、論理的な表に対しては作成する必要がありますが、レイアウト目的のために作成される表に対しては作成してはいけません。PDFlib によって組版される表は自動的にタグ付けできます。317 ページ「11.2.1 自動表タグ付け」を参照してください。
<i>TH</i>	表はヘッダーを含むべきです。TH 要素に対してはオプション Scope が推奨されます（これは、PDF_fit_table() の header オプションが用いられている場合には、自動表タグ付けによって常に生成されます）。
<i>L</i>	構造種別 L（リスト）は ListNumbering オプションを必須とします。その子 LI（リスト項目）群がいずれも Lbl（ラベル）要素を含まない場合には、ListNumbering を None にする必要があります。一方、もしも ListNumbering=None でありながら、可視なリストラベルがある場合には、それらをページ装飾としてマークする必要があります。
<i>Note</i>	Note としてタグ付けされる脚注と後注に対しては id オプションが必須です（id はどこでも参照されていませんので、これは何の得ももたらしません）。

見出し すべての見出しに対して、適切な見出しタグを用いる必要があります。PDF 文書内における見出しの階層ネストについては 2 通りのアプローチがあります：

- ▶ 強く構造化された文書：グループ化要素群が、必要だけ深くネストすることによって、アーティクル・セクション・下位セクションなどを持った内容の組織を反映します。各レベルにおいて、そのグループ化要素の子群は、1 個の見出し *H* と、そのレベルの内容のための 1 個ないし複数の段落 *P* と、ネストされた下位セクション群のためのさらなるグループ化要素群から成るべきです。強い構造は通常、XML 文書において用いられます。
- ▶ 弱く構造化された文書：その文書の構造ヒエラルキーが比較的フラットであり、ほんの 1～2 個のレベルのグループ化要素群を持ち、すべての見出し・段落やその他 BLSE 群をそれらの直接の子とします。その内容の組織が、その論理構造として反映されず、特定の見出しレベル *H1*・*H2*・*H3* などによって表現できます。見出しタグは子孫を一切持てません。弱い文書構造は通常、HTML 内において用いられます。

PDF/UA-1 モードにおいては、この区別を明示するために、*structuretype* 文書オプションを用いる必要があります。PDFlib は、文書構造の種別に応じて、見出し要素の使用に関する以下の規則を強制します。

- ▶ すべての文書：
 - ▶ すべての見出しタグ内で *Title* オプションを用いることによって、文書のセクションを表示する必要があります（例：「第 1 章」）。
 - ▶ 見出し要素 *H*・*H1*・*H2* などは子孫を一切持ってはいけません。
- ▶ 弱く構造化された文書、すなわち *structuretype=weak*（デフォルト）：
 - ▶ 見出しの連鎖は、*H1* から始まり、数値レベルを一切スキップしてはいけません。たとえば、*H1H3* という連鎖は許容されません。
 - ▶ 6 個を超える見出しレベルが必要な場合には、追加の見出しレベル *H7*・*H8* などを使うこともできます。これらは標準構造種別ではありませんので、*rolemap* オプション内にエントリが必要で、推奨されるマッピングは *P* です。
 - ▶ 番号なしの見出し要素 *H* を使ってはいけません。
- ▶ 強く構造化された文書、すなわち *structuretype=strong*（非推奨）：

- ▶ 見出しのために *H* を用いる必要がありますが、ただしその構造ヒエラルキーの各ノード内に複数の *H* タグがあってははいけません。
- ▶ 番号付きの見出し要素 *H1*・*H2* などを使ってはいけません。

11.4.3 内容種別ごとの追加の要件

表 11.11 に、さまざまな内容種別とインタラクティブ要素に関連した PDF/UA-1 の要件と推奨事項を挙げます。

表 11.11 内容種別とインタラクティブ要素ごとに対する PDF/UA-1 の要件と推奨事項

内容種別	PDF/UA-1 準拠のための PDFlib の要件
テキスト	ページ上のすべてのテキストの自然言語を宣言する必要があります。テキスト列内の自然言語の変化も宣言する必要があります。この自然元とを宣言するには、 <code>PDF_begin_document()</code> の <code>lang</code> オプションやその他の手段を用います。詳しくは 311 ページ「言語指定」を参照してください。要素の言語属性はそのすべての子孫へ継承されます。
ベクトルグラフィックとラスター画像	ラスター画像とベクトルグラフィックに、Artifact か Figure か Formula としてタグ付ける必要があります。これは、 <code>PDF_rect()</code> などのような低レベルパス構築関数と、 <code>PDF_draw_path()</code> を用いたパスオブジェクトと、 <code>PDF_fit_image()</code> や同様の関数にあてはまります。ベクトルグラフィックからラスター画像を内容として持つ SVG グラフィックも、この要件の対象となります。
取り込まれた PDF ページ	<code>PDF_fit_pdi_page()</code> を用いて配置された、グラフィックを含んだ PDF ページに、Artifact か Figure としてタグ付ける必要があります。
注釈	<p><code>PDF_begin/end_page_ext()</code> : オプション <code>taborder</code> に対して値 <code>structure</code> のみが許容されます。</p> <p><code>PDF_create_annotation()</code> で <code>type=Link</code> とする場合 : その注釈は Link 構造要素内に含まれている必要があります。</p> <ul style="list-style-type: none"> ▶ <code>PDF_create_action()</code> の <code>ismap</code> オプションが、リンク注釈内のアクションに対しては許容されません。 <p><code>PDF_create_annotation()</code> で種別を <code>Link</code>・<code>Popup</code> 以外とする場合 :</p> <ul style="list-style-type: none"> ▶ その注釈は Annot 構造要素内に含まれている必要があります。 ▶ 可視注釈¹のためには、サブオプション <code>ActualText</code> を持った <code>contents</code> オプションか <code>tag</code> オプションが必須です。 <p><code>PDF_fit_pdi_page()</code> : ページが <code>usetags=false</code> を用いて開かれている場合には、<code>useannots</code>・<code>usefields</code> オプションで <code>none</code> 以外の値は許されません。</p>
フォーム フィールド	<p>ページ上にフォームフィールドが存在する場合 : <code>PDF_begin/end_page_ext()</code> : オプション <code>taborder</code> に対して値 <code>structure</code> のみが許容されます。</p> <p><code>PDF_create_field()</code> と、<code>PDF_add_table_cell()</code> のオプション <code>fieldname</code>・<code>fieldtype</code> : <code>PDF_create_field()</code> か <code>tag</code> オプションを用いて Form タグを作成する必要があります。<code>PDF_create_field()</code> と <code>PDF_create_fieldgroup()</code> のオプションが必須です。</p>
ページラベル	<code>PDF_begin/end_document()</code> で <code>labels</code> オプションを用いて、および <code>PDF_begin/end_page_ext()</code> で <code>label</code> オプションを用いて作成されるページラベルは、意味付的に適切であるべきです。
しおり	<code>PDF_create_bookmark()</code> を用いてしおりを生成することが推奨されます。このしおり群は、正しい読み取り順序と、その内容のネストを反映するべきです。
添付	<code>PDF_load_asset()</code> : オプション <code>description</code> が推奨されます。添付はそれ自体でアクセシブルであるべきです。

1. 注釈が可視と見なされるのは、その長方形の少なくとも一部分がそのページの `CropBox` 内にあり、かつ `PDF_create_annotation()` の `display` オプションが `hidden`・`noview` 以外の場合です。

11.4.4 PDF/UA 文書のページを PDI で取り込む

既存の PDF 文書からのページを PDF/UA 準拠出力文書へ取り込もうとする際には、追加の諸規則が適用されます (PDF 取り込みについて詳しくは 211 ページ「8.3 PDF ページを PDI で取り込む」を参照してください)。既存の PDF 文書からタグ付きページを取り込むためには、その取り込まれる文書とページが、以下のクライテリアに従ってカレント文書に互換である必要があります：

- ▶ **PDF_open_pdi_document()** : PDF/UA-1 文書を取り込むことができ、かつ **usetags** オプションを **true** にする必要があります。タグなし文書、非 PDF/UA 文書、**PDF_open_pdi_document()** で **usetags=false** を用いて読み込まれたタグ付き文書も配置できます。ただし、そのようなページは自動的にページ装飾として標識されます。
- ▶ **PDF_open_pdi_page()** : 取り込まれる文書のロールマップが、**PDF_begin_document()** の **rolemap** オプションによって与えられたマッピングと互換である必要があります。これはすなわち、カスタム構造種別が、**rolemap** オプションと、取り込まれる文書のロールマップ (またはそれ以前に取り込まれた文書のロールマップ) とで、別々の標準種別へマップされてはいけないということです。
- ▶ **PDF_open_pdi_page()** : 取り込まれるページの見出し構造が、生成される文書の構造種別と互換である必要があります。すなわち、**structuretype=weak** の場合には **H1**・**H2** などのみ (**H**は不可) がそのページ上で使われている必要があります：**structuretype=strong** の場合には **H** のみ (**H1**・**H2** は不可) が、その取り込まれるページ上で使われている必要があります：番号付きと番号なしの見出しを両方持ったページは拒絶されます。

PDFlib 内で PDF/UA-1 準拠が構成されており、かつ取り込まれた文書が上記の要件を遵守しているならば、生成される文書も PDF/UA-1 に準拠していることが保証されます。

注 PDFlib は、PDF 入力文書の PDF/UA 準拠に関する検証は行わず、任意の入力 PDF 文書を PDF/UA へ変換することもできません。

12 PDF のバージョンと規格

注 PDF/UA 規格については 335 ページ「11.4 PDF/UA によるユニバーサルアクセシビリティ」で扱っています。

12.1 PDF のバージョン

12.1.1 PDF のバージョンの概観

PDFlib は以下の PDF バージョンのうちいずれか 1 つに従った出力を生成します：

- ▶ PDF 1.4 (Acrobat 5、2001 年リリース)。主に PDF/A-1 を生成するために有用
- ▶ PDF 1.5 (Acrobat 6、2003 年リリース)
- ▶ PDF 1.6 (Acrobat 7、2005 年リリース)
- ▶ PDF 1.7 (Acrobat 8、2006 年リリース)。技術的には ISO 32 000-1:2008 と同等
- ▶ PDF 1.7 Adobe 拡張レベル 3 (Acrobat 9、2008 年リリース)
- ▶ PDF 1.7 Adobe 拡張レベル 8 (Acrobat X/XI/DC、2010/2012/2015 ~ 2021 年リリース)
- ▶ ISO 32000-2 に従った PDF 2.0:2017。年月入り改訂版 ISO 32000-2:2020 を含みます

PDF 出力のバージョンは、`PDF_begin_document()` の `compatibility` オプションで制御することができます。どの PDF 互換モードでも、それよりも高いレベルのための PDFlib 機能を利用することはできません (表 12.1 参照)。そのような機能を利用しようとすると例外が発生します。

PDI で取り込む文書の PDF バージョン どの互換モードにおいても、PDI で取り込めるのはそれ以下の PDF バージョンの PDF 文書だけです。それより新しい PDF バージョンの PDF を取り込む必要がある場合は、それに合った `compatibility` オプションを設定する必要があります (213 ページ「8.3.3 文書・ページ関連のチェック」参照)。ただしこの上位 PDF バージョン取り込み不可ルールの例外として、PDF 1.7 拡張レベル 3 (Acrobat 9)・PDF 1.7 拡張レベル 8 (Acrobat X/XI/DC) に従った文書は PDF 1.7・PDF 2.0 文書へも取り込むことが可能です。

12.1.2 PDF 1.x の機能

表 12.1 特定の PDF 互換モードを要する PDFlib 機能

機能	PDFlib API メソッド・オプション
PDF 1.7 拡張レベル 8 (Acrobat X/XI/DC) 以上を要する機能	
AES-256 を用いて PDF を暗号化	<code>PDF_begin_document()</code> : <code>masterpassword</code> か <code>userpassword</code> オプションを与えた場合には、256 ビットを用いた AES 暗号化と、これより前の PDF バージョンよりも強力な暗号化アルゴリズムが使用されます。
PDF 1.7 拡張レベル 3 (Acrobat 9) 以上を要する機能	
マルチメディア	<code>PDF_load_asset()</code> で <code>type=3D/Sound/Video</code> を用いる <code>PDF_create_annotation()</code> : オプション <code>type=RichMedia</code> <code>PDF_create_action()</code> : オプション <code>type=RichMediaExecute</code>
地理空間 PDF	<code>PDF_begin_document()</code> : オプション <code>viewports</code> <code>PDF_load_image()</code> : オプション <code>georeference</code>
フォルダーのある PDF ポートフォリオ	<code>PDF_add_portfolio_folder()</code>
PRC 形式の 3D モデルの埋め込み	<code>PDF_load_3ddata()</code> : オプション <code>type=PRC</code>
PDF 1.7 = ISO 32000-1 (Acrobat 8) 以上を要する機能	
PDF ポートフォリオ	<code>PDF_begin_document()</code> : オプション <code>portfolio</code> <code>PDF_add_portfolio_file()</code>
添付に Unicode ファイル名	<code>PDF_begin/end_document()</code> : オプション <code>attachments</code> 、サブオプション <code>filename</code>
PDF 1.6 (Acrobat 7) 以上を要する機能	
AES-128 を用いて PDF を暗号化	<code>PDF_begin_document()</code> : <code>masterpassword</code> か <code>userpassword</code> オプション
NChannel カラー	<code>PDF_create_devicen()</code> : オプション <code>subtype=nchannel</code>
ユーザー単位	<code>PDF_begin/end_document()</code> : オプション <code>userunit</code>
印刷の拡張	<code>PDF_begin/end_document()</code> : <code>viewerpreferences</code> オプションに対するサブオプション <code>printscaling</code>
文書を開くモード	<code>PDF_begin/end_document()</code> : オプション <code>openmode=attachments</code>
ファイル添付のみを暗号化	<code>PDF_begin/end_document()</code> : オプション <code>attachmentpassword</code>
添付の説明	<code>PDF_begin/end_document()</code> : オプション <code>attachments</code> に対するサブオプション <code>description</code>
U3D 形式の 3D モデルの埋め込み	<code>PDF_load_3ddata()</code> ・ <code>PDF_create_3dview()</code> <code>PDF_create_annotation()</code> : オプション <code>type=3D</code> <code>PDF_create_action()</code> : オプション <code>type=GoTo3DView</code>
PDF 1.5 (Acrobat 6) 以上を要する機能	
マルチメディア	<code>PDF_load_asset()</code> で <code>type=Rendition</code> を用いる <code>PDF_create_annotation()</code> : オプション <code>type=Screen</code> <code>PDF_create_action()</code> : オプション <code>type=Rendition</code>

表 12.1 特定の PDF 互換モードを要する PDFlib 機能

機能	PDFlib API メソッド・オプション
インキ 8 色を超える DeviceN 色空間	PDF_create_devicen(): オプション names
さまざまなフィールドオプション	PDF_create_field()・PDF_create_fieldgroup()
ページレイアウト	PDF_begin/end_document(): オプション pagelayout=twopageleft/right
さまざまな注釈オプション	PDF_create_annotation()
拡張権限設定	PDF_begin_document() で permissions=plainmetadata、表 3.3 参照
タグ付き PDF	PDF_begin_item() に対するさまざまなオプション。 PDF_begin/end_page_ext(): オプション taborder
レイヤー	PDF_define_layer()・PDF_begin_layer()・PDF_end_layer()・PDF_layer_dependency()
JPEG 2000 画像	PDF_load_image() で imagetype=jpeg2000
圧縮オブジェクトストリーム	compatibility=1.5 以上の場合には、自動的に圧縮オブジェクトストリームが生成されます。ただし、PDF_begin_document() で objectstreams=none と設定されている場合を除きます。

12.1.3 PDF 2.0 の機能

PDF 2.0 は ISO 32000-2:2017 で仕様定義されており、年月入り改訂版 ISO 32000-2:2020 で改定されています。PDF 2.0 は以下のグループの機能を導入しています：

- ▶ Acrobat 9 の諸機能。PDFlib では `compatibility=pdf1.7ext3` 文書オプションで対応しています。たとえば地理参照付き PDF・ヒエラルキー型ポートフォリオ・AES-256 暗号化など。詳しい一覧については表 12.1 を参照してください。
- ▶ Acrobat X の諸機能。PDFlib では `compatibility=pdf1.7ext8` 文書オプションで対応しています。とりわけ、より強力な暗号化アルゴリズムを用いた AES-256 暗号化。
- ▶ PDF/X・PDF/A-3・PDF/VT 規格のいずれかで導入された諸機能、すなわち出力intent・連携ファイル・文書部分ヒエラルキーなど。
- ▶ 表 12.2 に挙げる追加機能。

PDF 2.0 についてさらに詳しい説明が PDFlib のウェブサイトにあります。

12.1.4 PDF の廃止済の機能

PDF 2.0 で廃止済の機能 PDF 2.0 では、表 12.3 に挙げる、それより前のバージョン群において利用可能であった機能のいくつかが廃止になっています。これらの機能が使われている場合には、PDFlib は PDF 1.x モードでは警告を、PDF 2.0 モードではエラーを発生します。PDF 1.x モードでもこれらの機能を避けることを推奨します。

ISO 32000 外の機能 表 12.4 に挙げる機能は、ISO 32000-1 の一部であったことはありません。これらは Acrobat でのみ動作し、サードパーティーのビューア群では動作しません。これらの機能のいずれかが使用されている場合には、PDFlib は PDF 1.x モードでは警告を、PDF 2.0 モードではエラーを発生します。これらの機能は標準の PDF の一部ではないので、PDF 1.x モードでも避けることを推奨します。

表 12.2 PDF.2.0 (ISO 32000-2) か特定の PDF/A・PDF/VT・PDF/X いずれかの規格を要する機能

機能	PDFlib API メソッド・オプション
タグ付き PDF	<p>以下の構造要素を、<code>PDF_begin_item()</code> と、さまざまなメソッドの tag オプションに与えるには、PDF 2.0 を要します：DocumentFragment・Aside・H7 など・Title・FENote・Sub・Em・Strong・Artifact（構造ヒエラルキーの要素として）</p> <p>構造移動先：<code>PDF_add_nameddest()</code> に対する移動先オプション群と、<code>PDF_create_action()</code>・<code>PDF_create_annotation()</code>・<code>PDF_create_bookmark()</code>・<code>PDF_begin/end_document()</code> で <code>structdest</code> オプションを用いた場合の移動先オプション</p> <p><code>PDF_create_action()</code> で <code>remotestructdest</code> オプションを用いた場合</p> <p>PDF 2.0 で構造要素に対する新たなネ스팅規則が導入されています。PDF 2.0 でオプションが追加されている構造エレメント属性があります。例：<code>artifacttype</code>・<code>artifactssubtype</code>・<code>ListNumbering</code></p> <p>構造要素群のための名前空間：<code>PDF_end_document()</code> の <code>pdftagset</code>・<code>tagset</code> オプションと、<code>PDF_begin_item()</code> の <code>namespace</code> オプション</p>
出カインテント	<p>文書レベル出カインテント ICC プロファイルが PDF 2.0・PDF/A・PDF/X のいずれかを要する：<code>PDF_load_iccprofile()</code> で <code>usage=outputintent</code> を用いた場合</p> <p>ページレベル出カインテント ICC プロファイルが PDF 2.0 を要する：<code>PDF_begin_page_ext()</code> で <code>outputintents</code> オプションを用いた場合</p>
グラフィックステート	<p><code>PDF_create_gstate()</code> と、さまざまなメソッドの <code>gstate</code> オプション：<code>useblackptcomp</code>・<code>halftoneorigin</code> オプション</p>
タブ順序	<p><code>PDF_begin/end_page_ext()</code>：<code>taborder</code> オプションに対する <code>annotations</code>・<code>widgets</code> キーワード</p>
文書部分ヒエラルキー	<p><code>PDF_begin/end_dpart()</code>：文書部分ヒエラルキーは PDF 2.0 か PDF/VT を要します。</p>
ファイル添付の関係	<p><code>PDF_load_asset()</code> で <code>type=Attachment</code> を用いた場合と <code>PDF_add_portfolio_file()</code> に対する <code>relationship</code> オプション、および <code>PDF_begin/end_document()</code> の <code>attachments</code> オプションと <code>PDF_create_annotation()</code> の <code>attachment</code> オプションに対する <code>relationship</code> サブオプション：関係指定は PDF 2.0 か PDF/A-3 を要します。</p>
連携ファイル	<p><code>PDF_end_document()</code>・<code>PDF_begin/end_page_ext()</code>・<code>PDF_begin/end_dpart()</code>・<code>PDF_begin_template_ext()</code>・<code>PDF_load_image()</code>・<code>PDF_open_pdi_page()</code>・<code>PDF_load_graphics()</code> に対する <code>associatedfiles</code> オプション：PDF 2.0 か PDF/A-3 を要します。</p> <p><code>PDF_create_annotation()</code>・<code>PDF_begin_item()</code> に対する <code>associatedfiles</code> オプションと、さまざまなメソッドの tag オプションは、PDF 2.0 を要します</p> <p><code>PDF_end_document()</code> に対する <code>structureassociatedfiles</code> オプションは PDF 2.0 を要します。</p>
注釈	<p><code>PDF_create_annotation()</code>：<code>blendmode</code>・<code>lang</code>・<code>opacityfill</code> オプション</p>

表 12.3 PDF 2.0 では廃止だが PDF 1.x モードでは今も利用可能な PDF 機能

PDF 2.0 で廃止済の PDFlib の機能	注
PDF_begin_document() : permissions で noaccessible キーワード	文書のアクセシビリティを制限することは PDF 2.0 では廃止済。
PDF_begin/end_document() : viewerpreferences、printarea・ printclip・viewarea・viewclip サブオプション	ブリプレスのビューア環境設定は PDF 2.0 では廃止済。
PDF_begin_page_ext()・PDF_end_ page_ext() : separationinfo	分版辞書は PDF 2.0 では廃止済。
PDF_create_annotation() : type=Movie	type=RichMedia を使用。
PDF_create_action() : type=Movie	type=RichMediaExecute を使用。
PDF_create_gstate() : blendmode はただ1つのキーワードのみを受け 付け、複数の値のリストを受け 付けません	ブレンドモード配列は PDF 2.0 では廃止済。

表 12.4 ISO 32000 外だが PDF 1.x モードでは今も利用可能な PDF 機能

ISO 32000 外の機能	PDFlib API メソッド・オプション
バーコードフィールド	PDF_create_field()・PDF_create_fieldgroup() : barcode
検索インデックス	PDF_begin/end_document() : search
編集可能な透かし	PDF_begin_template_ext() : watermark

12.2 PDF/A によるアーカイビング

12.2.1 各種の PDF/A 規格

注 PDF/A 規格に関する一般情報が PDF ウェブサイトにあります。

ISO 19005 規格シリーズで定められた各種の PDF/A 形式は、長期間にわたって安全にアーカイブできる、あるいは企業や政府の環境において信頼性を持ったデータ交換に利用できる、首尾一貫、かつ堅牢な PDF の部分集合を提供します。

PDF 協会における PDF/A PDFlib GmbH は PDF 協会 (the PDF Association) の創立メンバーです。PDF 協会は、そのさまざまな活動の 1 つとして、PDF/A 技術ワーキンググループ (TWG) をホストしています。この業界組織の目的は「オープン標準ベースの PDF 技術を利用した電子文書実装を、教育・専門技術・経験共有を通じて、全世界のステークホルダーのために促進する」です。詳しくは PDF 協会のウェブサイト www.pdfa.org を訪れてください。



ISO 19005-1 に従った PDF/A-1a:2005・PDF/A-1b:2005 PDF/A-1 は PDF 1.4 をベースに、色・フォント・注釈などの要素の使用にさまざまな制約を課しています。PDF/A-1 には 2 つの種類があります：

- ▶ ISO 19005-1 レベル B 準拠 (PDF/A-1b) は、文書の体裁が長期にわたって保持されることを保証します。簡単にいえば PDF/A-1b は、文書を将来いつの日か処理するときそれが今と同じに見えることを保証するものです。
- ▶ ISO 19005-1 レベル A 準拠 (PDF/A-1a) は、レベル B をベースに、タグ付き PDF から知られた諸特性を追加します：これは、文書の論理構造と自然な読み上げ順序を保持するために、構造情報と、信頼におけるテキスト意味付けを必須とします。PDF/A-1a は、その文書が将来において処理される際にそれが同じに見えることを保証するのみならず、そのコンテンツを信頼性を持って解釈でき、身体障害を持つユーザーに対してもアクセシブルであることを保証するものです。

PDFlib 内の PDF/A-1 対応は以下の文書に基づいています：

- ▶ PDF/A-1 規格 (ISO 19005-1:2005)
- ▶ 技術正誤表 1 (ISO 19005-1:2005/Cor 1:2007)。
- ▶ 技術正誤表 2 (ISO 19005-1:2005/Cor.2:2011)。
- ▶ PDF 協会が発行した TechNote 0010 「Clarifications of ISO 19005」。

PDF/A-1 と (準拠レベルを添えずに) 言うときは、PDF/A-1a と PDF/A-1b の両方の準拠レベルを意味します。

ISO 19005-2 に従った PDF/A-2a・PDF/A-2b・PDF/A-2u 各種の PDF/A-2 規格は、ISO 32000-1 (すなわち PDF 1.7) に基づいています。このことは、これらが PDF/A-1 よりも多くの機能をサポートしていることを意味します。PDF/A-1 と異なり、より新しい PDF/A-2 規格は、透過・レイヤー・JPEG 2000 圧縮・PDF/A ファイル添付・PDF パッケージやその他の PDF 諸機能を許しています。PDF/A-2 は以下の種類を定義しています：

- ▶ ISO 19005-2 レベル B 準拠 (PDF/A-2b)。文書の視覚的体裁を確保します。

- ▶ ISO 19005-2 レベル A 準拠 (PDF/A-2a)。信頼における Unicode テキスト意味付けと、構造情報を持つタグ付き PDF を追加しています。タグは、PDF/A-2a 文書が完全にアクセシブルであることを確実にします。
- ▶ ISO 19005-2 レベル U 準拠 (PDF/A-2u)。PDF/A-2a と PDF/A-2b の中間に位置付けられます。なぜならこれは、信頼における Unicode テキスト意味付けについては必須としますが、構造情報については必須としないからです。PDF/A-2u は、そのページ群が忠実に再現できることを、また、そのテキストが抽出・検索できることを保証します。

PDFlib 内の PDF/A-2 対応は以下の文書に基づいています：

- ▶ PDF/A-2 規格 (ISO 19005-2:2011)
- ▶ PDF 協会が発行した TechNote 0010 「Clarifications of ISO 19005」。

PDF/A-2 と (準拠レベルを添えずに) 言ったときは、PDF/A-2a・PDF/A-2b・PDF/A-2u の 3 種類すべての準拠レベルを意味します。

ISO 19005-3 で定義された PDF/A-3a・PDF/A-3b・PDF/A-3u PDF/A-3 は、以下の違いを除いて PDF/A-2 と同様です：

- ▶ PDF/A-2 では、PDF/A-1 か PDF/A-2 に準拠したファイル添付しか許容しませんが、PDF/A-3 では任意のファイル種別を添付として許容します。
- ▶ 添付されたファイルは、文書全体か、ページか、あるいはその文書のその他の要素と関連付けられます。ファイル添付と、文書のそれに対応する部分との間の関係が、ソース・代替・補足データなど、明示的に指定される必要があります。

PDFlib 内の PDF/A-3 対応は以下の文書に基づいています：

- ▶ PDF/A-3 規格 (ISO 19005-3:2012)
- ▶ PDF 協会が発行した TechNote 0010 「Clarifications of ISO 19005」。

PDF/A-3 と (準拠レベルを添えずに) 言ったときは、PDF/A-3a・PDF/A-3b・PDF/A-3u の 3 種類すべての準拠レベルを意味します。

電子インボイスのための ZUGFeRD 標準は、PDF/A-3 に基づいた重要な応用です。これは、そのインボイスの、機械が読める XML 版を、PDF/A-3 に準拠した、人が読める文書の中へ埋め込みます。ZUGFeRD に関するさらに詳しい情報が PDFlib ウェブサイトにあります。

12.2.2 一般的要件

クックブック PDF/A を生成するためのコードサンプルが PDFlib クックブックの pdfa カテゴリーにあります。

PDFlib クライアントプログラムが、この節で記す規則に従うなら、有効な PDF/A 出力が保証されます。PDFlib は、PDF/A 規則への違反を検出したときには例外を発生させますので、アプリケーション側でそれを処理する必要があります。この場合には PDF 出力は作成されません。表 12.5 に、PDF/A 準拠出力を作成するための一般的要件を挙げます。

PDF/A・PDF/UA-1 両立文書を作成 PDF/A 文書が同時に PDF/UA-1 に準拠することも可能です。実際、PDF/A-1a/2a/3a を作成したい場合には、生成文書のアクセシビリティを向上させるために、PDF/UA の要件に従うことを推奨します。詳細と制約については、335 ページ「PDF/UA-1・PDF/A 両立文書を作成」を参照してください。

PDF/A・PDF/X 両立文書を作成 PDF/A 文書が同時に PDF/X-3 か PDF/X-4 に準拠することも可能です。しかし PDF/X-4p か PDF/X-5n とは無理です。このような両立ファイル

表 12.5 PDF/A 準拠レベル A・B・U のための一般的要件

項目	PDF/A 準拠 (すべての準拠レベル) のための PDFlib の要件
PDF/A準拠レベルとPDF互換性	<p>PDF_begin_document(): pdfa オプションを、必要な PDF/A 準拠レベルに設定する必要があります。例: pdfa=PDF/A-2b</p> <p>PDF/A-1: PDF 1.5 以上を要する操作 (レイヤーなど) を避ける必要があります。</p> <p>PDF/A-2/3: PDF 1.7ext3 以上を要する操作 (PDF ポートフォリオなど) を避ける必要があります。</p>
フォント	<p>フォントオプション embedding を true にする必要があります。オプション unicodemap=false は許容されません。</p> <p>標準 Type 1 フォントについても埋め込みが必須です。この埋め込み必須は、不可視テキスト (主として OCR の生成結果のために有用) に対してのみ用いられているフォントに対してだけはあてはまりません。これは、optimizeinvisible オプションを用いて制御することもできます。</p>
ページ寸法	<p>(PDF/A-2/3) PDF_begin/end_page_ext(): PDF/A には、厳密なページ寸法制限はありません。しかし、ページ寸法 (幅と高さ、およびすべての枠エンタリー) を、PDF/A-1 では範囲 3 ~ 14400 ポイント (508 cm)、PDF/A-2/3 では範囲 3 ~ 14400 ユーザー単位に収めることが推奨されます。</p>
レイヤー	<p>PDF/A-1: PDF_define_layer() と PDF_set_layer_dependency() を避ける必要があります。</p> <p>PDF/A-2/3: レイヤーを使うことはできますが、PDF_define_layer() のいくつかのオプションを避ける必要があります。</p>
セキュリティ	<p>PDF_begin_document(): userpassword・masterpassword・attachmentpassword・permissions オプションを避ける必要があります。</p>
外部コンテンツ	<p>PDF_begin_template_ext()・PDF_load_graphics()・PDF_open_pdi_page(): reference オプションを避ける必要があります。</p> <p>PDF_load_asset(): external オプションを避ける必要があります。</p>
ファイルサイズ	<p>生成される PDF 文書のファイルサイズが 2 GB を超えてはならず、かつ、PDF オブジェクトの数が 8.388.607 未満でなければなりません。これらの制約について詳しくは 65 ページ「3.1.6 PDF 文書の最大サイズとその他の制限」を参照してください。</p>
PDF取り込み	<p>PDF_open_pdi_document() が制約されます。ただし infomode=true の場合を除きます。354 ページ「12.2.7 PDF/A 文書を PDI で取り込み」を参照してください。</p>

を作成するには、**PDF**_begin_document() の pdfa・pdfx オプションに対して適切な値を与えます。例:

```
ret = p.begin_document("combo.pdf", "pdfa=PDF/A-2b pdfx=PDF/X-4");
```

12.2.3 色と画像の要件

PDF/A は、忠実な色再現を保証するために、デバイス独立な色指定を必須としています。色空間は以下のソースから来る可能性があります:

- ▶ **PDF**_load_image() と **PDF**_fill_imageblock () を用いて直接的に、および **PDF**_load_graphics() を通じて間接的に読み込まれた画像
- ▶ **PDF**_set_graphics_option() か **PDF**_setcolor() を用いた明示的な色指定
- ▶ オプションリストを通じた色指定。たとえばテキストフロー内などにおいて。
- ▶ 透過グループのためにブレンドする色空間: **PDF**_begin/end_page_ext()・**PDF**_begin_template_ext()・**PDF**_load_graphics(): オプション transparencygroup でサブオプション colorspace

- ▶ スポットカラーか DeviceN カラーの代替色空間
- ▶ 注釈・フォームフィールドは枠・背景・内容に色を指定することができます

表 12.6 に、上に挙げた操作すべてにおいて従う必要がある、色処理のための PDF/A 要件を挙げます。

表 12.6 PDF/A 準拠レベル A・B・U のための色と画像の要件

項目	PDF/A (すべての準拠レベル) のための PDFlib の要件
出力条件 (出カインテント)	もしもデバイス依存色空間 Gray・RGB・CMYK のうちのいずれかがその文書内で用いられており、かつ然るべきデフォルト色空間がそのページに対して存在していないならば、 <code>PDF_begin_document()</code> の直後に、 <code>PDF_load_iccprofile()</code> で <code>usage=outputintent</code> として呼び出すか、 <code>PDF_process_pdi()</code> で <code>action=copyoutputintent</code> として呼び出す必要があります。
グレースケールカラー	グレースケールカラーを使うのは、グレースケールか RGB か CMYK の出カインテントが存在する場合か、 <code>defaultgray</code> オプションが設定されている場合だけです。
RGBカラー	ページ内容に RGB カラーを使うのは、RGB 出カインテントが存在する場合か、 <code>defaultrgb</code> オプションが設定されている場合だけです。フォームフィールドと注釈では RGB カラーはつねに使用できます。
CMYKカラー	CMYK カラーを使うのは、CMYK 出カインテントが存在する場合か、 <code>defaultcmyk</code> オプションが設定されている場合だけです。
Separation (スポット)・DeviceNカラー	<ul style="list-style-type: none"> ▶ その代替色は上述の諸規則に従う必要があります。 ▶ PDF/A-2/3 : <code>PDF_create_devicen()</code> の前に、その DeviceN 色空間内のすべてのカスタムスポットカラーについて <code>PDF_makespotcolor()</code> を呼び出す必要があります。
透過とオーバープリント	<p>PDF/A-1 : 透過を避ける必要があります。このことは以下の API 機能に影響を与えます :</p> <ul style="list-style-type: none"> ▶ <code>PDF_load_image()</code> : <code>masked</code> オプションを避ける必要があります。ただし、そのマスクが 1 ビット画像を指し示している場合を除きます。 ▶ <code>PDF_load_image()</code> : 内在透過 (アルファチャンネル) を持つ画像が許容されません。これを読み込むには <code>ignoremask</code> オプションを用いる必要があります。 ▶ <code>PDF_load_graphics()</code> : 透過要素を含んだ SVG グラフィックを避ける必要があります。 ▶ <code>PDF_create_gstate()</code> : <code>opacityfill</code>・<code>opacity</code> オプションを避ける必要があります。ただし、それが値 1 を持つ場合を除きます。<code>blendmode</code> を用いる場合には、それが Normal である必要があります。<code>softmask</code> を用いる場合には、それが none である必要があります。 ▶ <code>PDF_create_annotation()</code> : <code>opacity</code> オプションを避ける必要があります。 <p>PDF/A-2/3 : 透過は許容されますが、<code>PDF_create_gstate()</code> では以下の規則に従う必要があります : もしもカレント色空間が ICC ベース CMYK カラーであり、かつ <code>overprintfill</code> か <code>overprintstroke</code> が true の場合には、<code>overprintmode=1</code> は許容されません。</p>
透過グループ	<p><code>PDF_begin/end_page_ext()</code>・<code>PDF_begin_template_ext()</code>・<code>PDF_open_pdi_page()</code>・<code>PDF_load_graphics()</code> : オプション <code>transparencygroup</code> が以下のとおり制限されます :</p> <p>PDF/A-1 : オプション <code>transparencygroup</code> は許容されません。</p> <p>PDF/A-2/3 : <code>transparencygroup</code> オプションのサブオプション <code>colorspace</code> が、上でグレースケール・RGB・CMYK カラーについて述べた要件を満たす必要があります。<code>PDF_open_pdi_page()</code> と <code>PDF_load_graphics()</code> に対しては <code>transparencygroup=auto</code> が強制されます。</p> <p>PDF/A-2/3 : <code>transparencygroup</code> オプションのサブオプション <code>colorspace</code> が、上でグレースケール・RGB・CMYK カラーについて述べた要件を満たす必要があります。<code>PDF_begin/end_page_ext()</code> に対してオプション <code>transparencygroup=none</code> とサブオプション <code>colorspace=none</code> は許容されません。</p>

表 12.6 PDF/A 準拠レベル A・B・U のための色と画像の要件

項目	PDF/A (すべての準拠レベル) のための PDFlib の要件
画像とテンプレート	<p>PDF.load_image(): interpolate=true オプションを避ける必要があります。</p> <p>PDF/A-2/3 : JPEG 2000 画像は特定の条件を満たす必要がありますので、詳しくは 189 ページ「JPEG 2000 画像」を参照してください。</p>

出力インテント 出力条件は、意図する出力先デバイスを定義します。これは、一貫性のある色表現のために重要です。PDF/X では常に出力インテントが必須ですが、PDF/A ではこれと異なり、出力インテント ICC プロファイルの使用はオプションです。出力インテントは、RGB などデバイス依存カラーがその文書内で使われている場合にのみ必須です。ICC ベースカラーなどデバイス独立カラーのみがその文書内で使われている場合には出力インテントは必要ありません。PDF/X は出力インテントとしてプリンター ICC プロファイルにのみ対応していますが、PDF/A ではモニタープロファイルも許容されます。これにより、広く使われている sRGB プロファイルを出力インテントとして使うこともできるようになっています。出力インテントを指定するには、以下のように ICC プロファイルを用います：

```
icc = p.load_iccprofile("sRGB", "usage=outputintent");
```

ICC プロファイルを読み込むのではなく、出力インテントを、取り込んだ PDF/A 文書からコピーすることもできます (355 ページ「取り込んだ文書から PDF/A 出力インテントをコピー」参照)。生成される出力文書の出力インテントを設定するのはちょうど 1 回だけにする必要があります。これを設定するのは **PDF_begin_document()** の直後にするべきです。

PDF/A を作成するためのカラー戦略 表 12.7 に挙げるカラー戦略の概略が、さまざまな PDF/A アプリケーションを計画するうえで役立つかもしれません。多くの状況でうまくいく最も簡単な手は、**sRGB** 出力インテント ICC プロファイルを使うことです。なぜならこれはグレースケールと RGB カラーに対応しているからです。さらに、**sRGB** は PDFlib に内部的に知られていますので、外部プロファイルデータや構成を要しません。

黒いテキストを、出力インテントプロファイルを必要とせず作成したい場合には、CIELab 色空間を使えます。その **Lab** カラー値 (**a, a, a**) は、純粋な黒をデバイス独立な形で指定しており、かつ出力インテントプロファイルなしで PDF/A に準拠しています (これと異なり DeviceGray では出力インテントプロファイルが必須)。PDFlib は各ページの先頭でカレントカラーを黒に初期化します。ICC 出力インテントが指定されているかどうかに応じて、PDFlib は DeviceGray 色空間か Lab 色空間のいずれかを黒に対して用います。以下の呼び出しを用いれば **Lab** 黒色を手動で設定できます：

```
p.set_graphics_option("fillcolor={lab 0 0 0}");
```

表 12.7 準拠レベル A・B・U のための PDF/A のカラー戦略

出力インテント	その文書内で使える色空間					
	CIELab	ICC ベース	Separation・DeviceN	グレースケール ¹	RGB ^{1,2}	CMYK ¹
なし	○	○	○	—	—	—
グレースケール/ICC プロファイル	○	○	○	○	—	—

表 12.7 準拠レベル A・B・U のための PDF/A のカラー戦略

出カインテント	その文書内で使える色空間					
	CIELab	ICC ベース	Separation・DeviceN	グレースケール ¹	RGB ^{1,2}	CMYK ¹
RGB ICCプロファイル	○	○	○	○	○	—
CMYK ICCプロファイル	○	○	○	○	—	○

1. ICC プロファイルを持たないデバイス色空間か、ページ・パターン・テンプレートのためのデフォルト色空間
2. RGB カラーはフォームフィールドと注釈に対してはつねに許されます。

表 12.7 に挙げた色空間のみならず、カスタムスポットカラーを、その対応する代替色空間に従って使うこともできます。PDFlib は CIELab を、内蔵の HKS・PANTONE スポットカラーに対する代替色空間として用いていますので、これらは常に PDF/A 規格とともに使えます。カスタムスポットカラーに対しては、その代替色空間を、それが出力インテントと互換になるように選び取る必要があります。DeviceN 色空間内で使用されるすべてのカスタムスポットカラーについて `PDF_makespotcolor()` を呼び出す必要があります。

12.2.4 インタラクティブ機能に対する要件

表 12.8 に、PDF/A 準拠出力を生成する際に制約されるすべての操作を示します。禁じられたいずれかの関数を PDF/A モード内で呼び出すと例外が発生します。

表 12.8 すべての PDF/A 準拠レベルのためのインタラクティブ機能に対する要件

項目	PDF/A (すべての準拠レベル) のための PDFlib の要件
注釈	<p>PDF/A-1 : <code>PDF_create_annotation()</code> に以下の制約が課されます :</p> <ul style="list-style-type: none"> ▶ <code>PDF_create_annotation()</code> : <code>type=FileAttachment・Highlight</code> は許されません。以下の種別は、RGB 出カインテントを指定している場合のみ許されます : <code>Freetext・Text・Stamp・Underline・Squiggly・Strikeout・Polygon・Polyline・Line・Square・Circle・Ink</code>。 ▶ テキスト注釈に対しては <code>zoom・rotate</code> オプションを <code>true</code> に設定する必要があります。 ▶ <code>richtext</code> オプションは許されません。 ▶ <code>annotcolor・fillcolor・interiorcolor</code> オプション : グレースケールカラーは、出力インテント付き (種類は何でもよい) でのみ許されます。 ▶ <code>opacity</code> オプションを用いてはいけません。 <p>PDF/A-2/3 : <code>PDF_create_annotation()</code> に以下の制約が課されます :</p> <ul style="list-style-type: none"> ▶ <code>PDF_create_annotation()</code> : <code>type=3D・Screen</code> を用いることは許されません。 ▶ <code>annotcolor・fillcolor・interiorcolor</code> オプション : グレースケールカラーは、出力インテント付き (種類は何でもよい) でのみ許され、CMYK カラーは、CMYK 出力インテント付きでのみ許されます ▶ テキスト注釈の場合には <code>zoom・rotate</code> オプションを <code>true</code> に設定してはいけません。 ▶ <code>richtext</code> オプションは許されません。

表 12.8 すべての PDF/A 準拠レベルのためのインタラクティブ機能に対する要件

項目	PDF/A (すべての準拠レベル) のための PDFlib の要件
添付	<p>PDF/A-1 : <code>PDF.begin/end_document()</code> : attachments オプションを避ける必要があります。</p> <p>PDF/A-2 : <code>PDF.begin/end_document()</code> : attachments オプションの参照先が PDF/A-1 文書か PDF/A-2 文書である必要があります。</p> <p>PDF/A-3 : <code>associatedfiles</code> オプションを用いて任意のファイル種別を添付できますが、attachments オプションを避ける必要があります。以下の条件に従う必要があります :</p> <ul style="list-style-type: none"> ▶ 添付は、その文書のさまざまな部分と関連付けることができます。そのためには、<code>PDF.end_document()</code> ・ <code>PDF.begin/end_page_ext()</code> ・ <code>PDF.begin/end_dpart()</code> ・ <code>PDF.begin_template_ext()</code> ・ <code>PDF.create_annotation()</code> ・ <code>PDF.load_image()</code> ・ <code>PDF.open_pdi_page()</code> ・ <code>PDF.load_graphics()</code> の <code>associatedfiles</code> オプションを用います。それぞれの添付を、その文書のちょうど 1 つの部分に関連付ける必要があります。すなわち、<code>PDF.load_asset()</code> を用いて作成されたそれぞれのアセットハンドルを、ちょうど 1 つの <code>associatedfiles</code> オプションに与える必要があります。 ▶ <code>mimetype</code> ・ <code>relationship</code> サブオプションが必須です。 ▶ <code>description</code> サブオプションが推奨されます。 ▶ <code>external=true</code> サブオプションを避ける必要があります。
アクションと JavaScript	<p><code>PDF.create_action()</code> : <code>type=Hide</code> ・ <code>Launch</code> ・ <code>Movie</code> ・ <code>ResetForm</code> ・ <code>ImportData</code> ・ <code>JavaScript</code> を持つアクションを避ける必要があります。 <code>type=name</code> に対しては、<code>NextPage</code> ・ <code>PrevPage</code> ・ <code>FirstPage</code> ・ <code>LastPage</code> のみが許容されます。</p> <p><code>PDF.begin/end_document()</code> : オプション <code>action</code> は、トリガーイベント <code>open</code> とともにのみ使用できます。</p> <p><code>PDF.begin/end_document()</code> ・ <code>PDF.begin/end_page_ext()</code> : オプション <code>action</code> を避ける必要があります。</p>
フォームフィールド	<p><code>PDF.create_field/fieldgroup()</code> は以下の制約に縛られます :</p> <ul style="list-style-type: none"> ▶ 使用されているすべてのフォントが埋め込まれている必要があります。 ▶ オプション <code>backgroundcolor</code> ・ <code>bordercolor</code> ・ <code>fillcolor</code> ・ <code>strokecolor</code> : RGB カラーはつねに許され、グレースケールカラーは出カインテント (どの種別でも) 付きの場合にのみ許され、CMYK カラーは CMYK 出カインテント付きの場合にのみ許されます (表 12.6 も参照)。 ▶ <code>action</code> オプションは許されません。

12.2.5 レベル U 準拠のための追加の PDF/A の要件

PDF/A-2u と PDF/A-3u のための、規格の要件の多くは、PDFlib によって自動的に満たされます。レベル U 準拠の文書を生成する際には、表 12.9 に挙げる操作のみが制限されます。言い換えれば、アプリケーションがすでに PDF/A-2b か PDF/A-3b を作成しており、かつ表 12.9 の制約に従っていれば、生成された文書はそれぞれ、PDF/A-2u か PDF/A-3u としても宣言できます。

表 12.9 PDF/A 準拠レベル U のための追加の制約

項目	PDF/A-2u/3u 準拠のための PDFlib メソッド ・ オプションの要件
フォント	フォントオプション <code>unicodemap=false</code> を避ける必要があります。

12.2.6 レベル A 準拠のための追加の PDF/A の要件

PDF/A-1a ・ PDF/A-2a ・ PDF/A-3a を作成する際には、293 ページ「11.1 タグ付き PDF の基礎」に従ったすべてのタグ付き PDF の要件を満たす必要があります。表 12.10 に、レベ

ル A に従った出力を生成するための必須操作と推奨操作を挙げます。一般的なタグ付き PDF の諸規則のほかにも、PDF/UA の要件にも従うことによって、生成文書のアクセシビリティを向上させることを強く推奨します。詳しくは 335 ページ「11.4 PDF/UA によるユニバーサルアクセシビリティ」を参照してください。

正しい構造情報を作成することはユーザー側の役割です。文書のテキスト全体を 1 個の構造エレメントに入れたら、技術的には正しい PDF/A ですが、忠実な意味付け再生というゴールに違反しています。

表 12.10 PDF/A 準拠レベル A のための追加の要件

項目	PDF/A-1a/2a/3a 準拠のための PDFlib の要件
フォント	フォントオプション <code>unicodemap=false</code> を避ける必要があります。
タグ付き PDF	タグ付き PDF のためのすべての要件を満たす必要があります (293 ページ「11.1 タグ付き PDF の基礎」参照)。文書の構造ヒエラルキーは、その文書の論理構造を、できるだけ正確に反映するべきです。
単語境界	単語間を空白キャラクター (U+0020) で区切る必要があります。 <code>autospace</code> オプションを用いるとこの作業を単純化できます。
テキスト出力と PUA Unicode キャラクター	PDF/A-2a/3a : PUA Unicode キャラクター (ロゴや記号など) が、適切な置換テキストを持つ必要があります。それを囲うコンテンツアイテムに対する <code>PDF.begin_item()</code> の <code>ActualText</code> オプションで指定するか、その対応する出力メソッドの同等の <code>tag</code> オプションで指定します (詳しくは後述)。
注釈	<code>PDF.create_annotation()</code> : テキストを全く表示しない注釈に対しては <code>contents</code> オプションが推奨されます。

PUA キャラクター PDF/A-2a と PDF/A-3a は、私用領域、略して PUA、すなわち主として範囲 U+E000 ~ U+F8FF (詳しくは 109 ページ「BMP と PUA」を参照) の中にある Unicode 値を持ったキャラクターに対して、追加の要件を含んでいます。PUA キャラクターは通常、装飾的・記号グリフ、または企業ロゴなどカスタムグリフです。PDF/A-2a/3a は、PUA キャラクターが、そのキャラクターのテキスト表現を内容とする *ActualText* 属性を伴うことを必須とします。この *ActualText* は、個別の PUA キャラクターに対して割り当てることもできますし、ある 1 個の PUA キャラクターを包含するもっと長いキャラクター列に対して割り当てることもできます。この *ActualText* を、*Span* インラインレベルエレメントとともに与えることを推奨します。

`PDF.info_font()` を使って、ある特定のコードのある指定したフォントに対する Unicode 値をチェックできます (152 ページ「6.5.2 フォント依存のエンコーディング・Unicode・グリフ名クエリー」参照) :

```
uv = (int) p.info_font(font, "unicode", "code=" + c);
```

この生成される Unicode 値 *uv* が PUA に帰する場合には、これは *ActualText* 属性を必要とします。以下のコード断片は、PDFlib 企業ロゴのグラフィカル表現を包含した *PDFlibLogo* というフォントを想定しています。このロゴをページ上に配置する際には、テキスト「*PDFlib Logo*」を内容とする然るべき *ActualText* サブオプションを持った *Span* エレメントが `tag` オプションで与えられます :

```
p.fit_textline(text, 50, 700,
    "fontname=PDFlibLogo fontsize=24 " +
    "tag={tagname=Span ActualText={PDFlib Logo}}");
```

そのグリフに関して何ら情報を持たない場合、ゆえに然るべき *ActualText* がたやすくは得られない場合には、そのフォント内におけるそのグリフの名前を使うという手もあります。これを知るには以下のようにします：

```
gn_idx = (int) p.info_font(font, "glyphname", "code=" + c);
glyphname = p.get_option(gn_idx, "");
```

このグリフ名を *ActualText* で使う際、定常的な句と組み合わせるのも手でしょう。たとえば *Wingdings* フォント内のコード `ox1A` は、コンピューターキーボードの絵を内容とし、グリフ名 `keyboard` を持ちます。このグリフは U+F037 へマップします。すなわち PUA 値です。実テキストとして「*keyboard* の記号」を用いれば、この記号に対して意味を成すでしょう。ただし、プログラマ的に構築される *ActualText* は、当座しのぎの解決策と考えるべきです。人の選ぶテキストは常に、機械生成の *ActualText* よりも望ましいものです。

12.2.7 PDF/A 文書を PDI で取り込み

PDF/A 準拠の出力文書に既存の PDF 文書を取り込もうとするときは、追加の規則が適用されます (PDF 取り込みについて詳しくは 211 ページ「8.3 PDF ページを PDI で取り込む」を参照)。あらゆる取り込み文書は、表 12.11 に従ったカレントの PDF/A モードと互換な PDF/A 準拠レベルに準拠している必要があります。

注 PDFlib は、入力 PDF 文書の PDF/A 準拠に関する検証は行わず、任意の入力 PDF 文書を PDF/A へ変換することもできません。

ある特定の PDF/A 準拠レベルが PDFlib で構成されていて、かつ、取り込んだ文書がそれと互換なレベルを厳守しているならば、生成される出力は、選ばれた PDF/A 準拠レベルに従っていることが保証されます。カレントの PDF/A レベルと非互換の文書は `PDF_open_pdi_document()` で拒絶されます。

表 12.11 さまざまな PDF/A 出力レベルに対する互換 PDF/A 入力レベル

PDF/A 出力レベル	取り込まれる文書の PDF/A レベル				
	PDF/A-1a:2005	PDF/A-1b:2005	PDF/A-2a・ PDF/A-3a	PDF/A-2b・ PDF/A-3b	PDF/A-2u・ PDF/A-3u
PDF/A-1a:2005	許容	—	—	—	—
PDF/A-1b:2005	許容	許容	—	—	—
PDF/A-2a・PDF/A-3a	許容	—	許容	—	—
PDF/A-2b・PDF/A-3b	許容	許容	許容	許容	許容
PDF/A-2u・PDF/A-3u	許容	—	許容	—	許容

クックブック 完全なコードサンプルがクックブックの `pdfa/clone_pdfa` トピックにあります。

1 個ないし複数の PDF/A 文書を取り込むときは、そのすべてが表 12.12 に従った互換な出力条件で作成されている必要があります。すべての取り込み文書の出力インテントは同一か互換である必要があり、この条件を満たすよう手配するのはユーザー側の役割です。PDFlib は、いくつか特定の項目を修正することはできますが、PDF/A の検証に利用されるようにも、また、取り込んだ文書に PDF/A 準拠を強制するようにも作られていません。たとえば取り込んだ PDF のページに足りないフォントがあっても、PDFlib はその埋め込みはしません。

表 12.12 PDF/A 文書（すべての準拠レベル）を取り込む際の出カインテントの互換性

生成する文書の出カインテント	取り込む文書の出カインテント			
	なし	グレースケール	RGB	CMYK
なし	有	—	—	—
グレースケールの ICC プロファイル	有	有 ¹	—	—
RGB の ICC プロファイル	有	—	有 ¹	—
CMYK の ICC プロファイル	有	—	—	有 ¹

1. 取り込む文書の出カインテントと、生成する文書の出カインテントが、同一である必要があります。

取り込んだページを連結する際に、できあがる PDF 出力文書が入力文書（群）と同じ PDF/A 準拠レベル・出力条件に準拠するようにしたければ、取り込んだ PDF の PDF/A 状況を以下のようにクエリーすることができます：

```
pdfalevel = p.pcos_get_string(doc, "pdfa");
```

このステートメントは、取り込んだ文書が PDF/A レベルに準拠していればその PDF/A 準拠レベルを示す文字列を取得し、そうでなければ *none* を返します。この返された文字列を使えば、*PDF_begin_document()* で *pdfa* オプションを使って、出力文書の PDF/A 準拠レベルを適切に設定することができます。

取り込んだ文書から PDF/A 出カインテントをコピー PDF/A 準拠レベルをクエリーする以外の方法として、PDF/A 出カインテントは、取り込んだ文書からコピーすることもできます。PDF/A 文書には必ずしも出カインテントがあるとはかぎらないので、それをコピーしようとする前に、まず pCOS を使って、出カインテントが存在するかどうかをチェックする必要があります。

クックブック 完全なコードサンプルがクックブックの *pdfa/clone_pdfa* トピックにあります。

これは、*PDF_load_iccprofile()* を使って出カインテントを設定する方法のかわりに使うことができ、取り込んだ文書の出カインテントを、生成する出力文書へコピーします。出カインテントのコピーは、取り込んだ PDF/A と PDF/X の文書で動作します。

12.2.8 PDF/A のための XMP 文書メタデータ

PDF/A は、PDF 文書にメタデータを埋め込むために、XMP 形式に強く依存しています。PDF/A では、2 つの種類の XMP 文書レベルメタデータに対応しています：1 つは、定義済みスキーマというよく知られたメタデータスキーマの集合です。これは、XMP 仕様の基礎をなすバージョンから採られたものです。もう 1 つはカスタム拡張スキーマです。PDFlib は、XMP の中の必須の PDF/A 準拠エンタリー群を自動的に作成するほか、いくつかのよく使われるエンタリー（*CreationDate* など）も自動的に作成します。

文書レベル XMP XMP 文書メタデータは、*PDF_begin_document()* か *PDF_end_document()* または両方の *metadata* オプションで与えることができます。PDF/A モードでは、PDFlib は、ユーザーが与えた XMP 文書メタデータが PDF/A の要請に準拠しているかどうかを検証します。取り込み PDF 文書の中の XMP メタデータは、pCOS パス */Root/Metadata* を用いて入力 PDF から抽出することができます。

クックブック 完全なコードサンプルがクックブックのinterchange/import_xmp_from_pdfトピックにあります。

コンポーネントレベル XMP 文書全体に対してだけでなく、XMP メタデータは、PDF 文書内のそれ以外のページや画像といった構成要素に対しても与えることができます。コンポーネントレベルのメタデータに対しては、PDF/A-1 の要請はありませんが、PDF/A-2 と PDF/A-3 では、コンポーネントレベル XMP 内のカスタムプロパティ群をも文書レベル XMP と同様の拡張スキーマ記述によって記述することを義務付けています。

コンポーネントレベル XMP メタデータは、*PDF_begin/end_page_ext()* や *PDF_load_image()* などの関数の *metadata* オプションで与えることができます。

定義済み XMP スキーマ PDF/A 内の文書メタデータに対する XMP の使用は、以下の仕様に基いています：

- ▶ PDF/A-1 : XMP 2004 仕様
- ▶ PDF/A-2 ・ PDF/A-3 : XMP 2005

それぞれの XMP 仕様に記述されたスキーマを定義済みスキーマといい、それらの名前空間 URI とその望ましい名前空間接頭辞とともに表 12.13 に挙げています。定義済みスキーマのプロパティのみを PDF/A では使えます。ただし、拡張スキーマ記述が存在する場合は例外です（後述）。PDF/A-1 のための定義済み XMP 2004 スキーマ群内のプロパティの完全な一覧が、PDF 連合の PDF/A 技術センターからの TechNote 0008 内にあります。PDF/A-2/3 は、XMP 2005 からの定義済みスキーマ群を追加していますが、この追加スキーマは画像と動的メディアに関連したものですので、文書メタデータのために有用とはなりにくいでしょう。

表 12.13 PDF/A-1 のための定義済み XMP スキーマ（詳しくは XMP 2004 と XMP 2005 を参照）

スキーマの名称と説明	名前空間 URI	望ましい名前空間接頭辞
PDF/A-1 ・ PDF/A-2 ・ PDF/A-3 で使うための XMP 2004 スキーマ群		
Adobe PDF スキーマ	http://ns.adobe.com/pdf/1.3/	pdf
Dublin Core スキーマ	http://purl.org/dc/elements/1.1/	dc
EXIF 独自プロパティ群用 EXIF スキーマ	http://ns.adobe.com/exif/1.0/	exif
TIFF プロパティ群用 EXIF スキーマ	http://ns.adobe.com/tiff/1.0/	tiff
Photoshop スキーマ	http://ns.adobe.com/photoshop/1.0/	photoshop
XMP 基本ジョブチケットスキーマ	http://ns.adobe.com/xap/1.0/bj	xmpBJ
XMP 基本スキーマ	http://ns.adobe.com/xap/1.0/	xmp
XMP メディア管理スキーマ	http://ns.adobe.com/xap/1.0/mm/	xmpMM
XMP ページドテキストスキーマ	http://ns.adobe.com/xap/1.0/t/pg/	xmpTPg
XMP 権利管理スキーマ	http://ns.adobe.com/xap/1.0/rights/	xmpRights
PDF/A-2 ・ PDF/A-3 で使うための追加 XMP 2005 スキーマ群		
カメラ Raw スキーマ	http://ns.adobe.com/camera-rawsettings/1.0/	crs
追加 EXIF プロパティ群用 EXIF スキーマ	http://ns.adobe.com/exif/1.0/aux/	aux
XMP 動的メディアスキーマ	http://ns.adobe.com/xmp/1.0/DynamicMedia/	xmpDM

XMP 拡張スキーマ記述 自分が必要とするメタデータが、定義済みスキーマに含まれていないときは、XMP 拡張スキーマを定義することもできます。PDF/A では、カスタムスキーマを文書に埋め込む際に用いなければならない拡張方式を記述しています。表 12.14 に、1 個ないし複数の拡張スキーマとそのプロパティ群を記述するために用いなければならないスキーマをまとめ、あわせてその名前空間 URI と、必要な名前空間接頭辞を示します。名前空間接頭辞の、必要な、という点に注意してください（定義済みスキーマで示した名前空間接頭辞とは異なり、単にこうつけるのが望ましいというだけではなく、それぞれこの通りにつける必要があります）。

コンポーネントレベル XMP（ページレベルなど）に対してカスタム XMP プロパティが用いられる場合には、それに対応する拡張スキーマ記述をそのカスタム XMP プロパティ群とともに同一メソッド（`PDF_begin_page_ext()` など）内で与えることもできます。あるいは、コンポーネントレベル XMP に対する拡張スキーマ記述を文書レベル XMP とともに `PDF_begin_document()` で与えることも可能です。

XMP 拡張スキーマ記述の構築に関するさらなる詳細と例が、PDF/A 技術センターからの TechNote 009 にあります。

クックブック 完全なコードサンプルと XMP 作成例がクックブックの `pdfa/xmp_extension_schema_pdfa2b.pdfa/pdfa_extension_schema_with_type_pdfa2b` トピックにあります。

表 12.14 PDF/A 拡張スキーマコンテナスキーマと補助スキーマ

スキーマの名称と説明	名前空間 URI ¹	必要な名前空間接頭辞
PDF/A 拡張スキーマコンテナスキーマ：あらゆる拡張スキーマ記述を埋め込むためのコンテナ	<code>http://www.aiim.org/pdfa/ns/extension/</code>	<code>pdfaExtension</code>
PDF/A スキーマ値種別：1 個の拡張スキーマが任意個数のプロパティを持つのを記述	<code>http://www.aiim.org/pdfa/ns/schema#</code>	<code>pdfaSchema</code>
PDF/A プロパティ値種別：1 個のプロパティを記述	<code>http://www.aiim.org/pdfa/ns/property#</code>	<code>pdfaProperty</code>
PDF/A ValueType 値種別：拡張スキーマプロパティで用いるカスタム値種別を記述。XMP 2004 の種別一覧にない種別を使いたいときにのみ必要となります。	<code>http://www.aiim.org/pdfa/ns/type#</code>	<code>pdfaType</code>
PDF/A フィールド種別スキーマ：種別が構造化されている場合に、その中のフィールドを記述	<code>http://www.aiim.org/pdfa/ns/field#</code>	<code>pdfaField</code>

1. 名前空間 URI は、ISO 19005-1 では誤って列挙されており、技術正誤表 1 で修正されました。

12.3 PDF/X による印刷出力

12.3.1 PDF/X 規格ファミリー

注 PDF/X に初めてふれる方には、PDF 協会が発行した文書「PDF/X in a Nutshell」を導入として推奨します（www.pdfa.org/pdfx-in-a-nutshell/ 参照）。

PDF/X 形式群は、ISO 15930 で記述され、商業印刷に適したデータの受け渡しに利用できる一貫した堅牢な PDF の部分集合を定義しています。PDFlib は、以下に説明する種類の PDF/X に準拠した出力を生成し入力进行处理することができます。表 12.15 に、各種 PDF/X の主な違いを挙げます。

表 12.15 各種 PDF/X の比較

	PDF/X-3	PDF/X-4	PDF/X-4p	PDF/X-5n
PDF バージョン	PDF 1.4	PDF 1.6	PDF 1.6	PDF 1.6
CMYK・スポットカラー	可	可	可	可
デバイス独立カラー (ICCBased・Lab)	可	可	可	可
透過・レイヤー	—	可	可	可
外部参照出力インテント ICC プロファイル	—	—	可	可
n 色出力インテント ICC ロファイル	—	—	—	可

ISO 15930-4 で定義された PDF/X-1a:2003 この規格は、PDF 1.4 に基づいており、そのいくつかの機能（透過など）が禁止されています。PDF/X-1a は、CMYK・スポットカラーにのみ対応しており、RGB または ICC ベースの色には対応していません。ですので PDF/X-1a は廃止済です。

ISO 15930-6 で定義された PDF/X-3:2003 この規格は、PDF 1.4 に基づいており、グレースケール・CMYK・スポットカラーだけでなくデバイス独立色に基づくワークフローに対応しています。出力デバイスとしては単色・RGB・CMYK のいずれかを使うことができます。いくつかの PDF 1.4 機能、とりわけ透過が禁止されています。PDF バージョンが古いので、また、透過が除外されているので、PDF/X-3 は廃止済です。

ISO 15930-7 で定義された PDF/X-4 この規格は、PDF 1.6 に基づいています。PDF/X-4 では、透過とレイヤーは許されますが、それ以外のいくつかの PDF 1.6 の機能は依然禁止されています。その変種である PDF/X-4p では、出力インテント ICC プロファイルを、容量を抑えるために PDF 文書の外に置くことが許されます。PDFlib は ISO 15930-7:2010 を実装しています。

ISO 15930-8 で定義された PDF/X-5 この規格は「部分的交換」のためのものです。部分的交換を行うには、ファイルの作り手と受け手の間で事前の協議が必要です。PDF/X-5n は、PDF 1.6 に基づいており、n 色の印刷特性に対する外部出力インテント ICC プロファイルに対応しています。このプロファイルは **xCLR** プロファイルとも呼ばれます。

PDFlib は、ISO 15930-8:2010 を、正誤表 1 も含めて実装しています。

12.3.2 一般的要件

クックブック PDF/X を生成するためのコードサンプルが PDFlib クックブックの pdfx カテゴリーにあります。

PDFlib クライアントプログラムが、この項に記す諸規則に従えば、有効な PDF/X 出力が保証されます。PDFlib は、PDF/X 規則への違反を検出すると例外を発生させ、PDF 出力は何も生成されません。表 12.16 に、PDF/X 準拠出力を生成するための一般的要件を挙げます。

表 12.16 PDF/X 準拠のための一般的要件

項目	PDF/X 互換のための PDFlib の要件
PDF/X準拠レベルとPDF互換性	PDF .begin_document(): pdfx オプションを、必要な PDF/X 準拠レベルに設定する必要があります。例: pdfx=PDF/X-4。 PDF/X-3: PDF 1.5 以上を必要とする操作を避ける必要があります。 PDF/X-4/5n: PDF 1.7 以上を必要とする操作を避ける必要があります。
フォント	フォントオプション embedding が true である必要があります。埋め込みは PDF コアフォントについても必須です。この埋め込みの要件への唯一の例外は、不可視テキストのためにだけ使用されているフォントです（主として OCR の出力のための有用）。これは optimizeinvisible オプションを用いて制御できます（PDF/X-3 では許されません）。
ページ寸法	PDF .begin/end_page_ext(): ページ枠群を cropbox・bleedbox・trimbox・artbox オプションを通じて設定可能です。これらが以下の要件を満たす必要があります： ▶ TrimBox か ArtBox を設定する必要がありますが、ただしこれらの枠エントリーを両方とも設定してはいけません。TrimBox も ArtBox も見つからないときは、PDFlib は CropBox（もしあれば）を TrimBox として採り、CropBox も見つからないときは MediaBox を採ります。 ▶ BleedBox が、存在するときは、ArtBox と TrimBox を完全に包含している必要があります。 ▶ CropBox が、存在するときは、ArtBox と TrimBox を完全に包含している必要があります。
レイヤー	PDF/X-3: レイヤーを避ける必要があります。 PDF/X-4/5n: レイヤーを使えますが、ただし PDF .define_layer() のいくつかのオプションを避ける必要があります。
文書情報フィールドとXMPメタデータ	Creator・Title 情報フィールドを、空でない値に設定する必要があります。 PDF .set_info() を用いるか、(PDF/X-4/5n では) PDF .begin/end_document() の metadata オプションで xmp:CreatorTool・dc:title XMP プロパティを用います。 PDF .set_info() で Trapped 情報フィールドに、または PDF .begin/end_document() の metadata オプションでその対応する XMP プロパティ pdf:Trapped には、値 True か False 以外を避ける必要があります。
セキュリティー	PDF .begin_document(): userpassword・masterpassword・permissions オプションを避ける必要があります。
ファイルサイズ	PDF/X-4/5n: 生成される PDF 文書のファイルサイズが 2 GB を超えてはならず、かつ、PDF オブジェクトの数が 8,388,607 未満でなければなりません。これらの制限に関して詳しくは 65 ページ「3.1.6 PDF 文書の最大サイズとその他の制限」を参照してください。
PDF取り込み	PDF .open_pdi_document() が制約されます。ただし infomode=true の場合を除きます。365 ページ「12.3.6 PDF/X 文書を PDI で取り込む」を参照してください。

12.3.3 出力インテントと色の要件

PDF/X は、忠実な色再現を保証するために、デバイス独立な色指定を必須としています。色空間は以下のソースから来る可能性があります：

- ▶ `PDF_load_image()` と `PDF_fill_imageblock()` を用いて直接的に読み込まれた画像と、`PDF_load_graphics()` を通じて間接的に読み込まれた画像
- ▶ `PDF_set_graphics_option()` か `PDF_setcolor()` を用いた明示的な色指定
- ▶ オプションリストを通じた色指定。たとえばテキストフロー内において。
- ▶ スポットまたは DeviceN カラーの代替色空間
- ▶ 透過グループのためのブレンドする色空間：`PDF_begin/end_page_ext()`・`PDF_begin_template_ext()`・`PDF_load_graphics()`：オプション `transparencygroup` でサブオプション `colorspace`

上記の操作は、以下に詳しく示すように、PDF/X 出力を生成する際、さまざまな制約に従う必要があります。

出力インテント 出力インテントは、意図される出力先デバイスか印刷条件を定義します。これは、プルーフィング（ある印刷機器を他の機器でシミュレートする）と、デバイス独自カラーの特性化のために有用であり、透過計算のためのブレンド色空間としても有用です。PDF/X の出力インテントは通常、グレースケール・RGB・CMYK いずれかの ICC プリンタープロファイルによって記述されます。ただし PDF/X-5n は n 色 ICC プロファイルに対応しています。詳細は PDF/X の種類によって異なります：

- ▶ PDF/X-3/4: 出力インテントを、出力先デバイスか印刷条件のための ICC プロファイルを埋め込むことによって指定できます。
- ▶ PDF/X-4p・PDF/X-5n: 出力インテントのための外部 ICC プロファイルを参照することによって（規格の名前の中の *p* は、外部プロファイルが参照されることを意味します）。チェックサムによって保護された、ICC ファイルへの強力な参照が作成されます。参照 ICC プロファイルはその PDF 出力内に必ずしも埋め込まれるわけではないのですが、それでも PDF 作成時にこれが利用可能である必要があります。`urls` オプションを、1 個ないし複数の、その ICC プロファイルを発見できる場所の有効な URL とともに与える必要があります：

```
if (p.load_iccprofile("CGATS TR 001",
    "usage=outputintent urls={http://www.color.org}") == -1)
{
    /* エラー */
}
```

- ▶ PDF/X-5n: その PDF 文書の受信者がその出力インテント ICC プロファイルを利用できるようにするために、ICC プロファイルを PDF 文書内に添付することもできます。そのためには `embedprofile` オプションを用います。これは、ICC プロファイルを PDF 文書に、ファイル添付のように添付します。このようにプロファイルを埋め込む場合には、`urls` オプションは必要ありません。

対応しているすべての PDF/X 種別における出力インテント処理の比較を表 12.17 に示します。出力インテントに対する ICC プロファイルは、意図する印刷プロセスと用紙に基づいて、プリントプロバイダーによって指定されるべきです。印刷工程がわからない場合、またはプリントプロバイダーが出力インテントを何も指定していない場合には、然るべき ICC プロファイルを選ぶための推奨事項が www.pdflib.com にあります。

表 12.17 さまざまな PDF/X 種別に対する出力インテント処理

	PDF/X-3	PDF/X-4	PDF/X-4p	PDF/X-5n
出力インテントの色空間	グレースケールか RGB か CMYK	グレースケールか RGB か CMYK の ICC プロファイル	RGB か CMYK	n 色 ICC プロファイル (xCLR)
出力インテント ICC プロファイルの埋め込み	常に埋め込まれる	常に埋め込まれる	外部参照、ただし ICC プロファイルを添付することも可能	
その他の制約	—	CMYK 出力インテントプロファイルを他の目的に使用することは不可	外部参照、ただし ICC プロファイルを添付することも可能	出力インテントプロファイルを他の目的に使用することは不可

PDF/X-3・PDF/X-4/4p に対する色の要件 PDF/X-3/4 は、ICC プロファイルを用いたデバイス独立なワークフローをサポートしています。表 12.18 に、これらの規格のための要件を挙げます。PDF/X-5n は、別のワークフローを対象にしていますので、その要件は次の項で別途挙げます。

PDF/X-4 には特殊な規則が適用されます：CMYK 出力インテントプロファイル（すなわち *usage=outputintent* を用いて読み込まれた）を、同一文書内の ICC ベース色空間（すなわち *usage=iccbased* を用いて読み込まれた）のために用いることはできません。この要件は、PDF/X 規格によって必須とされているものであり、CMYK プロファイルに対してのみ適用され、グレースケール・RGB プロファイルには適用されません。この衝突が生じた場合には、単に ICC プロファイルを省略してかわりにプレーンな CMYK カラーを使用するのがよいでしょう。なぜなら出力インテント CMYK プロファイルはいずれにせよ適用されるからです。不要なエラーメッセージを避けるため、PDFlib は、画像か SVG グラフィックに埋め込まれている CMYK ICC プロファイルが PDF/X 出力インテントと等しい場合にはそれを無視します。

この制約は、取り込まれる文書にも適用されます：取り込まれるページが、生成される文書の出力インテントと同じ CMYK ICC プロファイルを用いている場合には、それは *PDF_open_pdi_page()* によって拒絶されます。

表 12.18 PDF/X-3・PDF/X-4/4p 準拠のための色の要件

項目	PDF/X-3・PDF/X-4/4p 互換のための PDFlib の要件
出力条件 (出力インテント)	<i>PDF_begin_document()</i> の直後に、 <i>PDF_load_iccprofile()</i> を <i>usage=outputintent</i> として、または <i>PDF_process_pdi()</i> を <i>action=copyoutputintent</i> として呼び出す必要があります。グレースケールか RGB か CMYK の出力インテントを与える必要があります。
グレースケールカラー	グレースケールカラーを使うのは、グレースケールか CMYK の出力インテントが存在するか、 <i>defaultgray</i> オプションが設定されているときだけです。
RGBカラー	RGB カラーを使うのは、RGB 出力インテントが存在するか、 <i>defaultrgb</i> オプションが設定されているときだけです。
CMYKカラー	CMYK カラーを使うのは、CMYK 出力インテントが存在するか、 <i>defaultcmky</i> オプションが設定されているときだけです。

表 12.18 PDF/X-3・PDF/X-4/4p 準拠のための色の要件

項目	PDF/X-3・PDF/X-4/4p 互換のための PDFlib の要件
Separation (スポット)・DeviceNカラー	<ul style="list-style-type: none"> ▶ PDF/X-3/4 : 代替色空間は上記の諸規則に従う必要があります。 ▶ PDF/X-4 : <code>PDF_create_devicen()</code> の前に、その DeviceN 色空間内のすべてのカスタムスポットカラーについて <code>PDF_makespotcolor()</code> を呼び出す必要があります。 ▶ PDF/X-4 : <code>PDF_create_devicen()</code> の <code>process</code> オプションの <code>colorspace</code> サブオプションは、PDF/X 出力インテントと合致している必要があります。

PDF/X-5n に対する出力インテントと色の要件 PDF/X-5n に対する出力インテント ICC プロファイルは、n 色印刷デバイスを記述している必要があります。このようなプロファイルは色空間指定 `xCLR` を内容として持ちます。ここで `x` は、2 から F までの 16 進数で、すなわち `2CLR` ~ `FCLR` となり、2 ~ 15 個のインキを表します。`xCLR` プロファイルは、プレーンな PDF では対応されておらず、PDF/X-5n の出力インテントとしてのみ使用可能です。PDF はこのようなプロファイルに直接対応していませんので、このようなプロファイルは、出力インテントとして埋め込むことはできず、外部ファイルとして参照する必要があります。ただしこれは外部オブジェクトとして、ファイル添付にして埋め込むことは可能ですので、そうすると入手可能性を向上させることができます。

`xCLR` プロファイルは、名前付きインキ群の名前とカラー値を持ったインキテーブルを内容として持ちます。PDF/X-5n 文書内で許される色空間と関連して、`xCLR` 出力インテントプロファイルがインキ `Black` を内容として持っているか、または場合によってはすべての CMYK インキ `Cyan・Magenta・Yellow・Black` を内容として持っているかは重要です。たとえば、CMYK・オレンジ・グリーン・バイオレットインキ (CMYKOGV) を用いた印刷処理のための `7CLR` プロファイルは、両方の要件を満たします。

表 12.19 に、PDF/X-5n に対する要件を挙げます。

表 12.19 PDF/X-5n 準拠のための色の要件

項目	PDF/X-5n 互換のための PDFlib の要件
出力条件 (出力インテント)	<code>PDF_begin_document()</code> の直後に、 <code>PDF_load_iccprofile()</code> を <code>usage=outputintnt</code> として、または <code>PDF_process_pdi()</code> を <code>action=copyoutputintnt</code> として呼び出す必要があります。n 色 (<code>xCLR</code>) プロファイルを与える必要があります。
グレースケールカラー	グレースケールカラーを使えるのは、出力インテントがインキ <code>Black</code> を内容として持っているか、 <code>defaultgray</code> オプションが設定されているときだけです。
RGBカラー	RGB カラーを使えるのは、 <code>defaultrgb</code> オプションが設定されているときだけです。
CMYKカラー	CMYK カラーを使えるのは、出力インテントがインキ <code>Cyan・Magenta・Yellow・Black</code> をすべて内容として持っているか、 <code>defaultcmk</code> オプションが設定されているときだけです。
Separation (スポット)・DeviceNカラー	<ul style="list-style-type: none"> ▶ 代替色空間は上記の諸規則に従う必要があります。 ▶ <code>PDF_create_devicen()</code> の前に、その DeviceN 色空間内のすべてのカスタムスポットカラーについて <code>PDF_makespotcolor()</code> を呼び出す必要があります。出力インテント内のインキリストにあるスポットカラーはこの要件から除外されます。 ▶ <code>PDF_create_devicen()</code> のオプション <code>subtype=nchannel</code> と <code>process</code> は許容されません。

然るべき PDF/X 出力インテントを選ぶ PDF/X 出力インテントの選択は通常、印刷出力をとりしきる印刷業者との話し合いで決まります。出力インテントの選択に関する情報が

印刷所側から出てこないときは、然るべき ICC 出力インテントプロファイルをインターネットで見つけることができます。

PDF/X-3/4/5 を作成するためのカラー戦略 色処理については、以下の 2 種類の戦略が可能であり、場合によっては組み合わせることもできます：

- ▶ **デバイス独立カラー**：出力インテント ICC プロファイルの種類にかかわらず、デバイス独立色空間を、すなわち ICC ベースか CIE Lab を使えます。Lab カラー値 (0, 0, 0) は、純粋な黒をデバイス独立な形で指定します。Lab 黒色を手作業で設定するには、以下の呼び出しを用います：

```
p.set_graphics_option("fillcolor={lab 0 0 0}");
```

- ▶ **デバイス依存カラー**：デバイス独自のグレースケールか RGB か CMYK カラーを使えます。グレースケールカラーはどんな種類の出力インテントに対しても使えますが、RGB・CMYK カラーは整合する出力インテントとしか使えません。

スポット・DeviceN カラーを、その対応する代替色空間に従って使うこともできます。PDFlib は CIE Lab を、内蔵の HKS・Pantone スポットカラーに対する代替色空間として用いていますので、これらは常に PDF/X-3/4/5 で使えます。カスタムスポットカラーに対しては、その代替色空間を、それが出力インテントと互換になるように選び取る必要があります。DeviceN 色空間内で使用されているすべてのカスタムスポットカラーについて `PDF_makespotcolor()` を呼び出す必要があります。

表 12.20 内のカラー戦略の要約は、PDF/X のアプリケーションを計画するために有用でしょう。

表 12.20 PDF/X-3/4/5 のカラー戦略

出力インテント ICC プロファイル	その文書で使える色空間					
	CIE Lab	ICC ベース	Separation・ DeviceN	グレイ スケール ¹	RGB ¹	CMYK ¹
グレースケール	○	○	○	○	—	—
RGB ²	○	○	○	○	○	—
CMYK	○	○	○	○	—	○
PDF/X-5nのための xCLR	○	○	○	○	—	—

1. ICC プロファイルを、またはそのページのためのデフォルト色空間を持たないデバイス色空間

2. PDF/X 規格はプリンタープロファイルを必須とします。sRGB などモニタープロファイルを使うことはできません。RGB プリンタープロファイルは非常に稀です。

12.3.4 画像と透過の要件

ラスター画像は、363 ページ「12.3.5 インタラクティブ機能のための要件」で説明した色関連の諸要件に従う必要があります。に、PDF/X 準拠のための画像関連の追加の要件を挙げます。

12.3.5 インタラクティブ機能のための要件

表 12.22 に、PDF/X 準拠出力を生成する際に制限されるすべての操作を挙げます。PDF/X モードでこの禁じられた関数のうちのいずれかと呼び出すと例外が発生します。

表 12.21 PDF/X 準拠のための画像要件

項目	PDF/X 準拠のための PDFlib の要件
画像	<ul style="list-style-type: none"> ▶ PDF/X-3 : JBIG2 画像は避ける必要があります。 ▶ PDF/X-4/5 : JPEG 2000 画像は特定の諸要件を満たす必要があります。詳しくは 189 ページ「JPEG 2000 画像」参照。
透過	<p>PDF/X-3 : 透過は避ける必要があります。これは以下の API 機能に影響します :</p> <ul style="list-style-type: none"> ▶ <code>PDF_load_image()</code> : masked オプションは、そのマスクが 1 ビット深度の画像を参照しているのではない限り、避ける必要があります。 ▶ <code>PDF_load_image()</code> : 内在透過 (アルファチャンネル) を持った画像は許容されません。それを読み込むには <code>ignoremask</code> オプションを用いる必要があります。 ▶ <code>PDF_load_graphics()</code> : 透過要素を内容として持つ SVG グラフィックは避ける必要があります。 ▶ <code>PDF_create_gstate()</code> : <code>opacityfill</code>・<code>opacitystroke</code> オプションは、それが値 1 を持つのではない限り、避ける必要があります。<code>blendmode</code> が使用される場合には、それは Normal である必要があります。<code>softmask</code> が使用される場合には、それは none である必要があります。 <p>PDF/X-4/5n : 透過画像・グラフィックを使用できます。</p>
透過グループ	<p><code>PDF_begin/end_page_ext()</code>・<code>PDF_begin_template_ext()</code>・<code>PDF_open_pdi_page()</code>・<code>PDF_load_graphics()</code> : オプション <code>transparencygroup</code> に以下の制約が課せられます :</p> <ul style="list-style-type: none"> ▶ PDF/X-3 : オプション <code>transparencygroup</code> は許容されません。 ▶ PDF/X-4 : <code>transparencygroup</code> オプションのサブオプション <code>colorspace</code> は、表 12.18 のグレースケール・RGB・CMYK カラーに対する要件を満たす必要があります。<code>PDF_open_pdi_page()</code> と <code>PDF_load_graphics()</code> に対しては <code>transparencygroup=auto</code> が強制されます。 ▶ PDF/X-5n : 透過オブジェクトを内容として持つすべてのページに対してオプション <code>transparencygroup</code> が必要です。<code>transparencygroup</code> オプションのサブオプション <code>colorspace</code> は、表 12.19 に示されているグレースケール・RGB・CMYK カラーに対する要件を満たす必要があります。<code>PDF_begin/end_page_ext()</code> に対しては、オプション <code>transparencygroup=none</code> とサブオプション <code>colorspace=none</code> は許容されません。

表 12.22 インタラクティブ機能のための PDF/X の要件

項目	PDF/X 互換のための PDFlib メソッド・オプションの要件
注釈とフォームフィールド	<p><code>PDF_create_annotation()</code>・<code>PDF_create_field()</code> : <code>BleedBox</code> (<code>BleedBox</code> が存在しない場合には <code>TrimBox/ArtBox</code>) 内では注釈を避ける必要があります。以下の種別の場合には、注釈の長方形を、少なくとも線幅の半分は <code>BleedBox</code> の外に置く必要があります : <code>Square</code>・<code>Circle</code>・<code>Ink</code>・<code>PolyLine</code>・<code>Polygon</code>。<code>PDF_create_annotation()</code> の <code>calloutline</code>・<code>cloudy</code>・<code>createrichtext</code> オプションは許されません。</p> <p>PDF/X-3 : <code>PDF_create_annotation()</code> で <code>type=Caret</code>・<code>FileAttachment</code>・<code>Highlight</code> を用いることは許されません。</p> <p>PDF/X-4/5 : <code>PDF_create_annotation()</code> で <code>type=Caret</code>・<code>Highlight</code>・<code>Screen</code> を用いることは許されません。</p> <p><code>PDF_fit_pdi_page()</code> で <code>useannots/usefields</code> オプションを用いた場合 : しおりとフォームフィールドは、それが生成されるページの <code>BleedBox</code> (それが生成されるページの <code>BleedBox</code> がないなら <code>TrimBox/ArtBox</code>) の中に位置する場合にのみ取り込まれます。この場合にはオプション <code>clipannotations=false</code> が必須です。</p>
ファイル添付	<p>PDF/X-3 : <code>PDF_begin/end_document()</code> : オプション <code>attachments</code> を避ける必要があります。</p>

表 12.22 インタラクティブ機能のための PDF/X の要件

項目	PDF/X 互換のための PDFlib メソッド・オプションの要件
アクションと JavaScript	<code>PDF_create_action()</code> : すべてのアクションを、JavaScript も含めて避ける必要があります。
ビューア環境設定/表示・印刷領域	<code>PDF_begin/end_document()</code> : <code>viewerpreferences</code> オプションに対して <code>viewarea</code> ・ <code>viewclip</code> ・ <code>printarea</code> ・ <code>printclip</code> サブオプションを用いる場合には、 <code>media</code> か <code>bleed</code> 以外の値は許容されません。

12.3.6 PDF/X 文書を PDI で取り込む

既存の PDF 文書のページを PDF-X 出力文書へ取り込もうとする際には、追加の規則が適用されます（詳しくは 211 ページ「8.3 PDF ページを PDI で取り込む」参照）。取り込まれる文書はすべて、表 12.23 に従ってカレント PDF/X モードに互換な PDF/X レベルに準拠している必要があります。許容されるすべての PDF/X-4/5 出力との組み合わせに対して、以下の追加の規則に服する必要があります：取り込まれるページが、生成される文書の出力インテントと同じ CMYK ICC プロファイルを用いている場合には、それは PDF/X-4/5 規格に違反するので、`PDF_open_pdi_page()` によって拒絶されます。

ある特定の PDF/X 準拠レベルが PDFlib で構成されていて、かつ、取り込まれた文書もその互換レベルを遵守しているならば、生成出力はその選択された PDF/X 準拠レベルに準拠していることが保証されます。カレント PDF/X レベルに互換でない文書は `PDF_open_pdi_document()` で拒絶されます。

表 12.23 さまざまな PDF/X 出力レベルに対する互換 PDF/X 入力レベル

PDF/X 出力レベル	取り込まれる文書の PDF/X レベル			
	PDF/X-1a・PDF/X-3	PDF/X-4	PDF/X-4p	PDF/X-5n
PDF/X-3	許容			
PDF/X-4	許容	許容	許容	
PDF/X-4p	許容	許容	許容 ¹	
PDF/X-5n				許容 ²

- `PDF_process_pdi()` で `action=copyoutputintenc` を指定すると、外部出力インテント ICC プロファイルへの参照がコピーされます。
- 取り込まれる文書が、同一の出力インテント（チェックサムが等しい）を使用している必要があります。

複数の PDF/X 文書を取り込む場合は、それらはすべて同一の出力条件に対して作成されている必要があります。

PDFlib は、いくつか特定の項目を修正することはできますが、PDF/X の検証を行ったり、取り込み文書に PDF/X 互換を強制したりするようにはもともと作られていません。たとえば PDFlib は、取り込んだ PDF ページに欠けているフォントを埋め込んだりはしませんし、取り込んだページに対して色補正なども一切行いません。

ページを取り込んだ結果としてできる PDF 出力文書が、入力文書（1つ複数）と同じ PDF/X 準拠レベルと出力条件に準拠するようにしたい場合は、次のように、取り込まれた PDF の PDF/X ステータスをクエリーすることができます：

```
pdfxlevel = p.pcos_get_string(doc, "pdfx");
```

このステートメントは、取り込まれた文書が ISO PDF/X レベルに準拠している場合には、PDF/X 準拠レベルを表す文字列を得ます。そうでない場合には *none* を得ます。この返ってきた文字列を用いれば、*PDF_begin_document()* の *pdfx* オプションを用いて出力文書の PDF/X 準拠レベルを適切に設定することができます。

取り込まれた文書から PDF/X 出力インテントをコピー PDF/X 準拠レベルをクエリーするだけでなく、取り込まれた文書から PDF/X 出力インテントをコピーすることもできます。

クックブック 完全なコードサンプルがクックブックの *pdfx/clone_pdfx* トピック内にあります。

これは、*PDF_load_iccprofile()* による出力インテント設定のかわりに利用することができ、取り込み文書の出力インテントを、生成する文書にコピーします。出力インテントをコピーする方法は、取り込まれた PDF/A と PDF/X の文書に対して通用します。

12.4 PDF/VT による可変・トランザクション印刷

12.4.1 PDF/VT 規格

注 PDF/VT 規格に関する一般情報が PDFlib Web サイトにあります。PDF 協会が発行した文書「PDF/VT Application notes」も推奨します (www.pdfa.org/resource/pdfvt-application-notes/参照)。

PDF/VT 規格は ISO 16612-2 として発行されています。これは「さまざまな環境で可変文書印刷 (VDP) を可能にするよう設計された」ものです。PDF/VT 文書は、最終的なコンテンツエレメント群と、関連するメタデータとを内容としますが、変数やテンプレートを含んではいません。PDF/VT は、PDF/X-4 に基づいており、さらに透過とレイヤーを含めて PDF 1.6 の諸機能に対応しています。PDF/X の要件に加えて、PDF/VT 規格は、大量パーソナライズ印刷の要請を満たす補足機能を PDF に追加しています。PDF/VT は、電子印刷ファイルの高パフォーマンスレンダリングを可能にするために、効果的なリソース管理を PDF に追加しています。

PDF/VT-1 は、PDF 1.6 に基づいており、単一ファイル交換のために設計されており、PDF/X-4 に基づいています。1 個の PDF 文書を表示するために必要なリソースのすべてが、単一の PDF/VT-1 ファイル内に含まれています。

12.4.2 PDF/VT の諸概念

この項では、PDF/VT が基づいている技術的諸概念のあらましを提供します。

文書部分ヒエラルキー 文書部分 (DPart) ヒエラルキーは、1 個の PDF/VT ファイル内の文書群ないし文書の部分群の並びと関係を指定します。代表的なシナリオでは、この PDF/VT ファイルは、多くの受領者のための下位文書群を内容としており、そしてそれぞれの文書部分は、個々の受領者のための下位文書を構成するページ群に対応しています。受領者にページを割り当てるだけでなく、この文書部分ヒエラルキーは、もっと複雑な構造を反映することもできます。たとえば、受領者をその住所内の郵便番号に従ってグループ化したり、郵便番号を州に従ってまとめたり、州を国に従ってまとめたりすることができます。この種の文書組織化は、その文書内のすべてのページを包含する 1 個の樹状構造を作成します。このツリーの要素群を DPart ノードといい、そこではそれぞれの内部ノードは他の DPart ノード群を内容とし、それぞれの葉ノードは 1 つの受領者のための 1 個ないし複数のページを指定します。

PDF/VT ファイル内の文書部分ヒエラルキーを用いて、ページにアクセスすることもできます。これは、ページ番号やページラベルによるアクセスといった他の方式のかわりに使えます。この DPart ヒエラルキーは、PDF/VT ファイル内では必須です。そのオプションなレコードレベル値は、個別の受領者に対するレコードに対応する DPart ヒエラルキー内のレベルを選択します。これはスコープヒントのために意味を持ちます (後述)。

文書部分メタデータ 文書ツリー内のルートノードから葉群まで下る文書部分ヒエラルキー内の各ノードは、文書部分メタデータ (DPM) を含むことができます。これを使うと、1 つの受領者の文書とその部分群に関する情報を伝えることができます。とりわけ、印刷のために意味を持つ諸特性 (ある文書部分の印刷部数など) や、受領者に関する情報 (その対応する郵便番号など) を DPM 内に符号化することが可能です。

JDF (または他の) 印刷メタデータは、PDF/VT では必須ではありませんが、JDF 対応ワークフローにおいては実用的な価値を加えます。

反復するグラフィカル内容に対する最適化 印刷要素が複数のページ上で再利用されることはよくあります。たとえば、1つの送付文書のすべてのページ上に現れる企業ロゴや製品画像です。反復するグラフィカル内容の処理を最適化することは、印刷ファイルのファイルサイズと処理速度を向上させるための重要な戦略です。PDFは、ファイルサイズを最適化するための手段として、つねにXObjectをサポートしてきました。これは、1個の印刷要素のために必要なデータをそのファイル内に1回だけ入れ込んで、複数のページ（または同一ページ上の複数のインスタンス）からこのデータへの参照を許すものです。XObjectは、ラスター画像か、任意のテキスト・ベクトルグラフィックコンテンツを内容とすることができます。PDFにおけるXObjectは、1個の文書の全体のサイズを最適化することに主眼を置いています。PDFは現在までのところ、反復するページコンテンツのレンダリングを最適化するための手段を一切含んでいません。たとえばある特定のページ上のある画像が、後で同一文書内の別のページ上に再び現れるということや、あるいは次の印刷ジョブ内に現れるかもしれないということや、消費ソフトウェアに対して告げることのできるものが、PDF内には何もないのです。PDF/VTは、PDFにおけるXObjectの既存の概念を拡張して、印刷パフォーマンスを最適化するための以下の手段を追加します：

- ▶ **一意識別**：XObjectに対して、すべての文書を通じて一意である識別子（*GTS_XID* という）を割り当てることができます。この識別子はキャッシングの実装のために利用可能です。なぜならこの実装は、同等なXObjectを識別する必要があるからです。簡単な言葉で言えば、ジョブ1のためにすでに処理されている、かつジョブ2内で再利用されることが発見されたグラフィックについては、再度リップする必要がなく、そのレンダリング結果をキャッシュから採ることができます。
- ▶ **スコープヒントと環境コンテキスト**：XObjectが、そのグラフィカル内容が再利用されるページ群か文書群の範囲に関する情報（*GTS_Scope* という）を含むことができます。これによってXObjectは、キャッシュ内のそのレンダリング結果の有用継続期間に関する情報を伴うことができます：このコンテンツはカレント受領者のためにだけ再利用されるのか、それとも同一ファイルないしファイルストリーム内のどこか別の所で再利用されるのか、あるいは全く再利用されないのか。環境コンテキスト（*GTS_Env* という）が与えられると、XObjectはグローバル用途を、すなわち、それが複数のPDF/VTインスタンス内で再利用されるということを指定することができます。環境コンテキスト文字列について制限は全くありません。たとえば、顧客名やジョブ名を用いてその環境を識別できます。
- ▶ **カプセル化ヒント**：RIP内のXObjectキャッシングアルゴリズムは、XObjectの、その呼び出しコンテキストと、その同一ページ（または他のページ群。たとえば複数のページを同一シート上に重ね合わせる場合）上の既存の印刷要素群との、相互作用を考慮に入れる必要があります。たとえばもしも、あるXObjectが色や線幅を指定しておらず、それが参照される時点において有効な色と線幅に基づいてその体裁が変わるならば、レンダリング結果をキャッシュしても、この変わる体裁のせいで効果的ではありません。もしもそのXObjectが透過要素を含んでおり、その既存の背景をそのXObjectとブレンドしなければならない状態の場合にも、同様の状況が起こります。RIP内におけるXObjectのキャッシングを支援するために、PDF/VTはカプセル化XObjectという概念を導入しており、それがそうであることを標識できるようにしています（*GTS_Encapsulated* キーを用いて）。カプセル化XObjectは、RIP内におけるキャッシングを支援する特定の諸規則を満たす必要があります。

これらのエントリーはすべてオプションです：PDF/VTは、反復するグラフィカル内容の最適化をいずれも必須としていませんが、それらを利用すれば、PDF/VT対応RIPを用いたときに目ざましい印刷パフォーマンス向上が実現します。

12.4.3 PDF/VT-1 を生成するための規則の要約

クックブック PDF/VT-1 を生成するためのコードサンプルが、PDFlib クックブックの pdfvt カテゴリ内にあります。

PDFlib を使って PDF/VT-1 を作成することは、以下の手段で実現できます：

- ▶ PDF/VT は PDF/X に基づいていますので、基礎をなす PDF/X 準拠レベルのためのすべての要件を満たす必要があります。pdfvt・pdfx 文書オプションで、然るべき値を用いる必要があります。
- ▶ 文書部分群を指定する必要があります。DPart API を用います。その文書部分ヒエラルキー内の各ノードが、オプションに DPM メタデータを伴うこともできます。その DPart ツリー内のレコードレベルを指定することもできます。
- ▶ 反復するグラフィカル内容のためのスコープヒントを与えるべきです。
- ▶ その文書が透過を含む場合には、カプセル化 XObject のための諸条件を満たすために追加の努力を払うべきです。
- ▶ 既存の PDF 文書からページを取り込む際には、追加の諸規則が適用されます (374 ページ「12.4.7 PDF/X・PDF/VT 文書を PDI で取り込む」参照)。

表 12.24 に、359 ページ「12.3.2 一般的要件」で挙げた、その対応する PDF/X の要件に加えて、PDF/VT 準拠出力を生成するための必須操作とオプションな操作を要約します。特定の諸側面については、以下に続く各項で詳しく説明します。

表 12.24 PDF/VT 準拠のための必須操作とオプションな操作

項目	PDF/VT 準拠のための PDFlib の要件
準拠レベル	<p>PDF_begin_document() の pdfvt オプションを、必要な PDF/VT 準拠レベルに設定する必要があります。例：pdfvt=PDF/VT-1</p> <p>PDF/VT-1：オプション pdfx=PDF/X-4 が自動的に設定されます。pdfx オプションに対して他の値を与えることはエラーです。</p>
文書部分ヒエラルキー	<p>文書部分ヒエラルキーを指定する必要があります。これは以下の操作を必要とします：</p> <ul style="list-style-type: none">▶ PDF_begin_document()：nodenamelist オプションが、DPart ツリーのすべてのレベルのための名前を指定する必要があります。recordlevel オプションを用いて、受領者レコード群に対応する DPart ツリーのレベルを指定することもできます。▶ PDF_begin_dpart()・PDF_end_dpart()：文書部分ヒエラルキーを構築する必要があります。▶ 文書部分群が、オプションに文書部分メタデータ (DPM) を伴うこともできます。DPM を生成するには、POCA インターフェイスを用いて DPM 辞書を構築します。
反復するグラフィカル内容のためのスコープヒント	<p>PDF_load_image()・PDF_open_pdi_page()・PDF_begin_template_ext()、および PDF_load_graphics() の templateoptions オプション：pdfvt オプションに scope サブオプションを、値 unknown・singleuse・record・file・stream・global のうちの 1 つとともに与えることによって、反復する画像・テンプレート・取り込み PDF ページのための用途ヒントを与える必要があります。</p> <p>scope=stream と scope=global に対しては、サブオプション environment が PDF/VT 環境コンテキストを指定する必要があります。これはすなわち、PDF/VT 準拠のリーダーが、関連する XObject 群を管理するための管理インターフェイスを提供するために使える識別子です。たとえば顧客名やジョブ名を用いてその環境を識別できます。</p>

表 12.24 PDF/VT 準拠のための必須操作とオプションな操作

項目	PDF/VT 準拠のための PDFlib の要件
カプセル化XObject	<p>PDF.begin_document(): その文書が透過要素を一切含んでいない場合には、オプション <code>usestransparency=false</code> を与えることによって、透過グループを持たないすべてのフォーム XObject を PDFlib がカプセル化できるようにする必要があります。</p> <p>PDF.load_image(): 画像は、XObject がカプセル化されていると標識されることを可能にするために、<code>renderingintent</code> か <code>mask</code> オプションを用いるべきです。(373 ページ「12.4.6 カプセル化 XObject」を参照)。</p>

PDF/VT・PDF/A 両立文書を作成 PDF/VT 印刷文書でありながら、同時にアーカイビングのための PDF/A にも準拠している文書を作成すると、有用なこともあるでしょう (PDF/A に関する詳しい情報は 346 ページ「12.2 PDF/A によるアーカイビング」を参照してください)。このような二重用途の文書を作成する際には以下に留意してください:

- ▶ 以下の文書オプションを用いれば、PDF/VT・PDF/A 両立文書を作成できます:

```
ret = p.begin_document("combo.pdf", "pdfx=PDF/X-4 pdfvt=PDF/VT-1 pdfa=PDF/A-2b");
```

- ▶ このマニュアル内で PDF/VT と PDF/A に対して述べた制約に従う必要があり、かつ、両方の規格で許容されている機能だけを使えます。
- ▶ 取り込まれる PDF 文書が、受け付け可能であるためには、PDF/X 規格と PDF/A 規格の両方を遵守している必要があります。

12.4.4 文書部分ヒエラルキーと文書部分メタデータ (DPM)

クックブック 文書部分ヒエラルキーを持った PDF/VT を作成するためのコードサンプルが、PDFlib クックブックの `pdfvt` カテゴリにあります。

文書部分ヒエラルキーを作成 文書内のすべてのページを、文書部分ヒエラルキー内に組織化する必要があります。インボイスなどシンプルな場合においては、このツリー構造は、*root* レベルと *recipient* レベルの 2 つのレベルから成ります。この文書内のページ群は、受領者レコード群の線形配列を構成しています。ここで各レコードは 1 個ないし複数のページを内容としています。この文書部分ヒエラルキーの構造を、文書オプションリストで指定する必要があります。例:

```
if (p.begin_document(outfile,
    "pdfvt=PDF/VT-1 nodenamelist={root recipient} recordlevel=1") == -1)
```

もっと複雑な文書は、もっと深い文書ヒエラルキーを必要とするかもしれません。たとえば、カスタマイズされた小冊子が、前付・本文・後付部から成る場合です。これらは *docpart* レベル内で保持されます:

```
if (p.begin_document(outfile,
    "pdfvt=PDF/VT-1 nodenamelist={root recipient docpart} recordlevel=1") == -1)
```

この *recordlevel* オプションは、その受領者レコードレベルを発見できる場所の、*nodenamelist* 内でのゼロベースの番号を指定します。これは *scope=record* オプションに対して意味を持ちます。

ヒエラルキーの構造を定義した後は、ページ群を文書ヒエラルキーノード群内でグループ化する必要があります。これを達成するには `PDF_begin/end_dpart()` を用います:

```

/* DPartヒエラルキーのルートノードを作成 */
p.begin_dpart("");
    p.begin_dpart("");          /* 受領者レベルに新規ノードを作成 */
        p.begin_page_ext(...); /* 1個ないし複数のページを作成 */
        ...
        p.end_page_ext(...);
    p.end_dpart("");          /* 受領者ノードを閉じる */

    p.begin_dpart("");          /* 受領者レベルに次のノードを作成 */
        p.begin_page_ext(...); /* 1個ないし複数のページを作成 */
        ...
        p.end_page_ext(...);
    p.end_dpart("");          /* 受領者ノードを閉じる */
/* ルートノードを閉じる */
p.end_dpart("");

```

この `PDF_begin/end_dpart()` への呼び出し群が、`nodenamelist` オプションに従ったツリー構造を作成する必要があります。すなわち、その最大ネストレベルが、この `nodenamelist` 配列内のエントリーの数に対応する必要があります。

`PDF_begin_document()` の `groups` オプションと、`PDF_begin/end_page_ext()` の `group · pagenumber` オプションは、PDF/VT モードでは許容されません。なぜなら、これらは DPart 構造を阻害するからです。

文書部分メタデータ (DPM) を作成 文書部分ヒエラルキー内の各ノードに対して、1つの葉ノードに対応するページ群に、またはこのノード配下の下位ツリー全体に適用されるメタデータ情報を指定できます。PDF/VT は特定の種類のメタデータを何ら規定していませんが、これは CIP4 機構によって発行された JDF 規格とともに用いられることを意図しています。

この CIP4 メタデータ形式は、代表的な PDF データ型を活用しており、そのメタデータは、PDF/VT 規格で仕様化されている形で PDF 出力へ書き出されます。

PDFlib ユーザーは、POCA (PDF Object Creation API = PDF オブジェクト作成 API) インターフェイスを使って、PDF 辞書・配列やその他のデータ型から成る任意のデータ構造を作成することもできます。DPM メタデータに対する最上位レベル辞書を、`PDF_begin/end_dpart()` の `dpm` オプションへ渡すことができます：

```

dpm          = p.poca_new("type=dict usage=dpm");
cip4_root    = p.poca_new("type=dict usage=dpm");
cip4_metadata = p.poca_new("type=dict usage=dpm");

p.poca_insert(dpm,          "type=dict key=CIP4_Root value=" + cip4_root);
p.poca_insert(cip4_root,   "type=dict key=CIP4_Metadata value=" + cip4_metadata);
p.poca_insert(cip4_metadata, "type=string key=CIP4_Conformance value=base");
p.poca_insert(cip4_metadata, "type=string key=CIP4_Creator value=starter_pdfvt1");
p.poca_insert(cip4_metadata, "type=string key=CIP4_JobID value={Kraxi Systems invoice}");

/* DPartヒエラルキー内にノードを作成してDPMメタデータを追加 */
p.begin_dpart("dpm=" + dpm);

p.poca_delete(dpm, "recursive=true");

```

クックブック PDFlib pCOS クックブックに、PDF/VT 文書から DPM を取得して、その対応する XML 表現を作成するためのコードサンプルがあります。

12.4.5 反復するグラフィカル内容のためのスコープヒント

スコープヒント `pdfvt` オプションの `scope` サブオプションを、然るべき値とともに、`PDF_load_image()`・`PDF_load_graphics()`・`PDF_open_pdi_page()`・`PDF_begin_template_ext()` に与えるべきです。これを達成するには、カレント PDF 文書内で、または（なるべく）複数文書にわたって、画像・ページ・テンプレートが再利用される場所と回数をクライアントアプリケーションが知る必要があります。この `scope` オプションは必須ではありませんが、これは強く推奨されます。なぜならこれは、RIP のための重要な最適化情報を提供するからです。そのような情報が得られない場合には、間違っているおそれのある値を与えるよりも、この `scope` オプションを省略するか、値 `unknown` とともに指定するべきです。

スコープ `singleuse`・`record`・`file` に対しては、PDFlib は、そのスコープ値がその XObject の実際の使用と整合しているかどうかをチェックして、もしもそのスコープが大きすぎるときは、ログファイル内にたとえば以下のような警告を出力します：

```
XObject with handle 9 was assigned PDF/VT scope 'record', but was used only once
(use 'scope=singleuse' instead)
```

このような警告を得たときには、もっと然るべきスコープ値を、その対応する画像かグラフィックか取り込み PDF ページかテンプレートに割り当てることによって、RIP 内の不必要なキャッシングを回避し、ひいては印刷パフォーマンスを向上させることができるかどうかをチェックするべきです。

ある XObject への参照が少なすぎる場合（その `scope` オプションと比較して）には、クライアントコードに改善の余地がありうることを示唆する警告が発生するだけでありますが、ある XObject への参照が多すぎる場合にはエラーになります。なぜならこれは PDF/VT 規格に違反するからです。より具体的には、以下の状況で例外が発生します：

- ▶ オプション `scope=singleuse` が与えられており、かつ、その XObject がその文書内で複数回使用されている。
- ▶ オプション `scope=record` が与えられており、かつ、その XObject が複数の受領者レコード内で使用されている。

`scope=stream` と `scope=global` に対しては、`environment` サブオプションを与える必要があります。これは、キャッシュされたオブジェクトを複数の文書にわたって識別するために用いられる、然るべき文字列を内容とするべきです。ワークフローに応じて、顧客参照やジョブ参照をこのオプションのために使うとよいでしょう。たとえば、複数の印刷ジョブにわたって頻繁に柄笑える企業ロゴを、この方法で識別するとよいでしょう。

一意な ID PDFlib は、すべての取り込まれた PDF ページ・画像・グラフィックに対して、一意な ID を自動的に割り当てます（グラフィックに対しては、`templateoptions` オプションを与える必要があります）。等価な画像・ページ・グラフィック群に等しい ID 値が割り当てられることにより、RIP 内での効果的なキャッシングを可能にしています。テンプレートに対しては、一意な ID を、`pdfvt` オプションの `xid` サブオプションを通じてユーザーが与えるべきです。テンプレートに対する識別子は、PDF/VT に従って等価な PDF フォーム XObject を作成するテンプレート定義群（すなわち、同一の視覚的出力を持つテンプレート）に対して等しくするべきです。等価でないテンプレートは、異なる識別子を持つか、識別子を全く持たないようにする必要があります。

アプリケーションが、PDFlib によって作成された一意な ID 文字列を必要とする場合、これを取得するには、画像とテンプレートに対しては `PDF_info_image()` の、グラフィックに対しては `PDF_info_graphics()` の、取り込まれた PDF ページに対しては `PDF_info_pdi_page()` の、`xid` オプションを用います。

12.4.6 カプセル化XObject

XObject をカプセル化すると、リッピングのパフォーマンスが大いに向上します。ただし XObject をカプセル化できるのは、完全に不透明な文書内か、(透過が使われている文書内では) 特定の要件が満たされた場合のみです。

文書内の透過 その文書内で透過が使われるかどうかを PDFlib はあらかじめ知りませんので、ヒントとして `PDF_begin_document()` に `usestransparency` オプションを与えることもできます。クライアントがこのオプションを用いて、透過が一切使われていないと宣明した場合には、すべての XObject をカプセル化できます。その文書内で透過が使われている場合には、その XObject 群がカプセル化されるようにするには、クライアントは特定の諸規則に従う必要があります。`usestransparency=false` が与えられたにもかかわらず、その文書が透過要素を含んでいた場合には、PDFlib は例外を発生させます。

以下の条件のうちの 1 つないし複数が真であれば、その生成文書内で透過が使われていると見なされます：

- ▶ `PDF_load_image()`か`PDF_fill_imageblock()`が、以下のいずれかのオプションとともに呼び出されている：
 - ▶ その画像がアルファチャンネルを含んでおり、かつ `ignoremask` オプションが `false`。
 - ▶ ピクセルあたりビット数が複数である画像のハンドルを持った `masked` オプション。
- ▶ `PDF_load_graphics()` を用いて取り込まれたグラフィックが、何らかの透過要素を含んでいる。
- ▶ `PDF_create_gstate()` が、以下のいずれかのオプションとともに呼び出されている：
 - ▶ オプションリストを持った (すなわち、キーワード `none` を持っているのではない) `softmask` オプション。
 - ▶ 値 1 以外を持ったオプション `opacityfill` か `opacitystroke` のいずれか 1 つ。
 - ▶ 値 `None` ・ `Normal` 以外を持った `blendmode` オプション。
- ▶ `PDF_open_pdi_page()`か`PDF_fill_pdfblock()`を用いて取り込まれたPDFページが、何らかの透過要素を含んでいる。取り込まれた PDF ページ内の透過を識別するには、pCOS インターフェイスと `usespagetransparency` 擬似オブジェクトを用います (詳しくは pCOS パスリファレンスを参照してください)。

XObject をカプセル化 PDFlib は、多くの XObject を以下の規則に従ってカプセル化します：

- ▶ `PDF_load_image()`か`PDF_fill_imageblock()`を用いて作成される画像 XObject は、以下の条件のいずれかが真であれば、カプセル化されます：
 - ▶ `PDF_load_image()`の `renderingintent` オプションが値 `Auto` 以外とともに与えられている。
 - ▶ `PDF_load_image()`の `mask` オプションが値 `true` とともに与えられている。
- ▶ `PDF_begin_template_ext()` を用いて作成されるフォーム XObject は、もしも文書オプション `usestransparency=false` が与えられているか、`transparencygroup` オプションがサブオプション `colorspace` と `isolated=true` とともに与えられている場合には、カプセル化されます。
- ▶ `PDF_load_graphics()`か`PDF_load_image()`で `templateoptions` オプションを付けて生成されたフォーム XObject はつねにカプセル化されます。文書オプション `usestransparency=true` が与えられている場合には、然るべき色空間を持った透過グループがそのフォーム XObject に紐付けられます。そうでない場合にはグループは一切生成されません。

- ▶ `PDF_open_pdi_page()` を用いて生成されたフォーム XObject は、その取り込み PDF ページがレイヤーを一切含んでいないか、または `PDF_open_pdi_document()` が `uselayers=false` とともに呼び出されている場合には、カプセル化されます。文書オプション `usestransparency=true` が与えられている場合には、然るべき色空間を持った透過グループがそのフォーム XObject に紐付けられます。そうでない場合にはグループは一切生成されません。取り込むページが、すでにカプセル化されている XObject を含んでいる場合には、そのページを取り込む際に、このプロパティは変更されずに温存されます。

XObject をカプセル化できないときは、警告がログファイルへ書き出されます。

カプセル化 XObject に対するスコープ `PDF_load_image()`・`PDF_fill_imageblock()`・`PDF_load_graphics()`・`PDF_begin_template_ext()` を用いて作成するすべてのカプセル化 XObject に対して、`pdfvt` オプションの `scope` サブオプションを与えることを推奨します。この `scope` サブオプションは、非カプセル化 XObject に対して与えることも許容されていますので、もしも画像かグラフィックが取り込まれるページかテンプレートの継続期間がわかっているならば、その生成される XObject のカプセル化ステータスにかかわらず、すべての関連する API 呼び出しにこれを与えることもできます。

12.4.7 PDF/X・PDF/VT 文書を PDI で取り込む

既存の PDF 文書からのページを PDF/VT 準拠の出力文書内へ取り込もうとする際には、特別な諸規則が適用されます (PDI について詳しくは 211 ページ「8.3 PDF ページを PDI で取り込む」を参照してください)。すべての取り込まれる文書は、PDF/X-3/4/4p か PDF/VT-1 に準拠している必要があります。もしも PDF/VT 準拠が PDFlib 内で構成されており、かつ取り込まれる文書がその互換 PDF/X レベルを遵守しているならば、その生成出力が、その選択された PDF/VT 準拠レベルに準拠することが保証されます。取り込まれる文書は、受け付け可能な PDF/X レベルのうちの 1 つを遵守していない場合には、拒絶されます。

PDF/VT 取り込みでの DPart と DPM の制約 PDF/VT 文書を取り込む際には、以下の制約に留意してください：文書部分ヒエラルキー (DPart) と文書部分メタデータ (DPM) は取り込まれません。それらを、pCOS を用いてクエリーして、`PDF_begin/end_dpart()` と POCA 関数群を用いて再構築することが可能です。

13 PPS と PDFlib Block Plugin

PDFlib Personalization Server (PPS) は、可変データを処理する、テンプレートを用いる PDF ワークフローをサポートするものです。ブロックという概念を用い、外部の情報源から取得した、任意の量の、1 行ないし複数行のテキスト、または画像・PDF ページ・ベクトルグラフィックを、取り込んだページに入れ込むことができます。これを利用して、PDF 文書のカスタマイズを要するアプリケーションを容易に実装が可能です。たとえば：

- ▶ メールを連結
- ▶ ダイレクトメールの宛名印刷
- ▶ 納品書・明細などを発行
- ▶ 名刺の項目内容を各人ごとに変更

ブロックは、PDFlib Block Plugin を使って対話的に作成・編集もできますし、フォームフィールド変換プラグインを使って既存の PDF フォームフィールドを PDFlib ブロックに変換もできます。PPS を使ってブロックに流し込みができます。Block Plugin は内蔵版 PPS を含んでいますので、PPS でブロックに流し込んだ結果を Acrobat でプレビューできます。

注 ブロック処理を利用するには PDFlib Personalization Server (PPS) が必要です。PPS はすべての PDFlib 商用パッケージに含まれていますが、PPS に対するライセンスキーをご購入いただく必要があります。PDFlib や PDFlib+PDI のライセンスキーだけではご利用になれません。PDF テンプレートに対話的にブロックを作成するには Adobe Acrobat 用 PDFlib Block Plugin が必要となります。

クックブック 可変データとブロックに関するコードサンプルが PDFlib クックブックの blocks カテゴリにあります。

13.1 PDFlib Block Plugin をインストール

Block Plugin が動作する Acrobat のバージョンは以下のとおりです (Adobe Reader では動作しません)：

- ▶ Windows : Acrobat 8/9/X/XI
- ▶ Windows : Acrobat Standard・Pro 2017/2020
- ▶ Windows : Acrobat Pro DC (32 ビット・64 ビット)
- ▶ macOS : Acrobat Standard・Pro 2017/2020 (Intel)
- ▶ macOS : Acrobat DC (Intel・M1 用ユニバーサルバイナリー)

Windows 版 Acrobat DC には 32 ビット版と 64 ビット版がありますので、インストーラーが 2 種類あります。インストールされている Acrobat のバージョンに合致しているインストーラーを使うことが重要です。

Windows で PDFlib Block Plugin をインストール PDFlib Block Plugin と PDF フォームフィールド変換プラグインを Acrobat にインストールするには、プラグインのファイルを Acrobat のプラグインフォルダーのサブディレクトリーに入れる必要があります。プラグインのインストーラーはこれを自動的に実行しますが、手作業も可能です。プラグインのファイル名は *Block.api* と *AcroFormConversion.api* です。

64 ビット Windows 上の 32 ビット Acrobat の場合はプラグインフォルダーはたいてい以下のとおりです：

C:\Program Files (x86)\Adobe\Acrobat DC\Acrobat\plug_ins\PDFlib Block Plugin

64 ビット Acrobat の場合はプラグインフォルダーはたいてい以下のとおりです：

C:\Program Files\Adobe\Acrobat DC\Acrobat\plug_ins\PDFlib Block Plugin

macOS で PDFlib Block Plugin をインストール このプラグインをすべてのユーザーのためにインストールするには以下の操作を行います：

- ▶ ディスクイメージをダブルクリックすることによってマウント。プラグインファイル群の入ったフォルダーが現れます。
- ▶ このプラグインフォルダーを、システムのライブラリーフォルダー内の以下のパスへ複製 (*Plug-ins* フォルダーがまだない場合には作成)：

/Library/Application Support/Adobe/Acrobat/XXX/Plug-ins

あるいは以下の操作を行うと、このプラグインを単独のユーザーのためだけにインストールできます：

- ▶ デスクトップをクリックすることによって Finder 内に確実にいるようにしてから、**Option** キーを押しながら「移動」→「ライブラリー」を選択することによって、そのユーザーのライブラリーフォルダーを開きます。
- ▶ プラグインフォルダーを、そのユーザーのライブラリーフォルダーの中の以下のパスへ複製 (*Plug-ins* フォルダーがまだない場合には作成)：

/Users/<ユーザー名>/Library/Application Support/Adobe/Acrobat/XXX/Plug-ins

多言語インターフェイス PDFlib Block Plugin は、ユーザーインターフェイス内で複数言語に対応しています。Block Plugin は、Acrobat のアプリケーション言語に従ってインターフェイス言語を自動的に選択します。目下、日本語・英語・ドイツ語のインターフェイスが利用可能です。Acrobat がこれ以外の言語モードで動作している場合には、Block Plugin は英語インターフェイスを使用します。

Windows 版 Acrobat DC のためのサンドボックス保護 Acrobat DC 2020 では、サンドボックス保護という新たなセキュリティーモデルが導入されました。これを有効にするには「環境設定」→「セキュリティー (拡張)」→「起動時に保護モードを有効にする (プレビュー)」・「保護されたビュー」を選択します。これが有効になっていると、さまざまな操作が制限され、文書ウィンドウの上端の黄色い帯にセキュリティーメッセージが表示されます。サンドボックス保護について詳しくは：

helpx.adobe.com/acrobat/using/whats-new/2020-august.html

www.adobe.com/devnet-docs/acrobatetk/tools/AppSec/sandboxprotections.html

サンドボックスが有効になっていると、PDFlib Block Plugin のプレビュー機能に影響があります。保護ビューはデフォルトでは、Acrobat の *AppData* ディレクトリーと、temp ディレクトリーなどいくつかのディレクトリーへのアクセスは許しますが、任意のユーザーディレクトリーへのアクセスは許しません。Block Pluginが読み書きできるディレクトリーは、保護ビューのデフォルトディレクトリーリストに含まれているか、以下の場所にあるポリシーファイルの中で構成 (ホワイトリスト) されているものに限られます (Acrobat の 32 ビット版の場合と 64 ビット版の場合)：

C:\Program Files (x86)\Adobe\Acrobat DC\Acrobat\PDFlibBlockCustomPolicies.txt

C:\Program Files\Adobe\Acrobat DC\Acrobat\PDFlibBlockCustomPolicies.txt

デフォルトではこのポリシーファイルは、以下のディレクトリーへのアクセスを許していますが、管理者はこれにディレクトリー名を追加することもできます：

```
; Protected Path Section
FILES_ALLOW_ANY = C:\Users\\*.*
FILES_ALLOW_ANY = C:\Users\Public\*.*
```

もし保護モードか保護ビューが有効にされていて、かつ、使用されているディレクトリーがホワイトリストされていない場合には、Block Plugin の機能のうち、プレビューや、ブロックの取り込み／書き出しなどは、失敗する可能性があります。

トラブルシューティング PDFlib Block Plugin が動作しないように見られる場合には、以下を確認してください：

- ▶ 「編集」 → 「環境設定」 (→ 「一般 ...」) → 「一般」で「承認されたプラグインのみを使用」チェックボックスがオフになっていることを確認してください。Acrobat がこの承認済みモードで動作していると、プラグインは読み込まれません。
- ▶ Adobe Designer または Adobe Experience Manager によって作成された PDF フォームは、Block Plugin の適切な動作を妨げることがあります。他の Acrobat のプラグインの動作についても同様です。なぜならこうしたフォームは、Acrobat の内部セキュリティーモデルと衝突するからです。ですので、Designer の静的な PDF フォームは利用せずに、動的な PDF フォームだけを Block Plugin への入力として用いることを推奨します。

13.2 ブロックの概念の概要

13.2.1 文書デザインとプログラムコードの分離

PDFlib のデータブロックを利用すると、取り込んだページに、可変のテキスト・画像・PDF ページ・ベクトルグラフィックを簡単に配置できます。単純な PDF ページと違って、データブロックを含むページは、後でサーバーサイドで行われるべき処理についての情報を内部に持っています。PDFlib ブロックの概念は、以下の 2 種類の作業を完全に分離するものです：

- ▶ デザイナーはページレイアウトを作成し、可変なページ構成要素の位置を指定するとともに、その文字サイズ・色・画像拡大縮小といった関連特性も指定します。レイアウトを PDF 文書として作成し、その後デザイナーが、Acrobat 用 PDFlib Block Plugin を使って、可変データブロックとそのそれぞれのプロパティを指定します。
- ▶ プログラマーは、取り込まれる PDF ページの PDFlib ブロックに含まれる情報をデータベースのフィールドなど動的な情報と紐づけるコードを書きます。プログラマーは、ブロックの詳細（内容が名前なのか ZIP コードなのかとか、ページ上の具体的な位置や書式）については何も知る必要がないので、レイアウトがどう変更されても影響されずにいられます。ブロックに関連する詳細についてはすべて、ファイル内のブロックプロパティに基づいて PPS の側で処理されます。

言い換えれば、プログラマーが書くコードが「データ非依存」になります。すなわち汎用であり、ブロックのいかなる特性にも依存しないのです。たとえば手紙の宛先を印字するブロックがあったとして、かりにデザイナーがそれをページ上で別の所へ動かそうが、文字サイズを変えようが、汎用であるブロック処理コードに変更を加える必要はありません。デザイナーがこのブロックにラストネームでなくファーストネームを印字したいと思ったら、ブロックのプロパティを Acrobat プラグインで変更するだけで正しい出力が生成されます。

中間ステップとして、ブロックへの流し込みは Acrobat でプレビューできますので、開発とテストのサイクルを高速化できます。ブロックのプレビューには、ブロックの定義で指定しているデフォルトデータ（文字列や画像ファイル名）が用いられます。

13.2.2 ブロックのプロパティ

ブロックの動作はブロックのプロパティで制御できます。Block Plugin を使ってブロックにプロパティを割り当てます。

定義済みブロックプロパティ ブロックはページ上の長方形として定義され、名前や種類、その他自由なプロパティを割り当てることができます。これらプロパティは後で PPS によって処理されます。名前は、ブロックを識別する任意の文字列であり、たとえば *firstname*・*lastname*・*zipcode* のように名づけることができます。PPS では、さまざまな種類のブロックを使うことができます：

- ▶ **テキスト行ブロック**は、1 行のテキストデータを持ちます。このデータは、PPS のテキスト行メソッドで処理されます。
- ▶ **テキストフローブロック**は、1 行ないし複数行のテキストデータを持ちます。複数行のテキストは、PPS のテキストフローフォーマッターによって組版されます。複数のテキストフローブロックを連結して、前のブロックからあふれたテキストを次のブロックに入れることも可能です（399 ページ「テキストフローブロックを連結」参照）。
- ▶ **画像ブロック**は、ラスター画像を持ちます。DTPアプリケーションでTIFFやJPEGのファイルを貼り付けるのと似ています。

- ▶ **PDFブロック**は、他のPDF文書のページから取り込んだ任意のPDFコンテンツを持ちます。これは、DTPアプリケーションでPDFページを貼り付けるのと似ています。
- ▶ **グラフィックブロック**は、ベクトルグラフィックを持ちます。これは、レイアウトアプリケーションでSVGファイルを貼り付けるのに似ています。

ブロックは、その種類によって異なるさまざまな定義済みプロパティを持っています。プロパティは、Block Plugin で作成・変更できます (384 ページ「13.3.2 ブロックプロパティを編集」を参照)。定義済みブロックプロパティの全一覧が 402 ページ「13.7 ブロックのプロパティ」にあります。たとえばテキストブロックならテキストのフォントやサイズを指定できますし、画像ブロックやPDF ブロックなら拡大縮小率や回転を指定できます。PPS はブロックの種類ごとに、それを処理する専用の関数を提供しています (`PDF_fill_textblock()` など)。これら関数は、取り込まれた PDF ページにおいてブロックを名前を検索し、そのプロパティを分析して、クライアントの与えたデータ (1 行テキスト・複数行テキスト・ラスター画像・PDF ページ・ベクトルグラフィックのいずれか) を、新しいページ上に、指定されたブロックプロパティに従って配置します。プログラマーがブロックのプロパティをオーバーライドすることも可能です。そのためにはブロック流し込み関数でそれに対応するオプションを指定します。

デフォルト内容に関するプロパティ ブロックのデフォルト内容を持たせる特殊なブロックプロパティを指定することも可能です。ブロック流し込み関数に変数データが与えられなかったときや、ブロック内容が次の印刷実行時には変わるかもしれないが現時点では固定なときにそのブロックに配置されるテキスト・画像・PDF・グラフィック内容を指定できます。

このデフォルトプロパティはBlock Pluginのプレビュー機能でも用いられます(392ページ「13.5 ブロックを Acrobat でプレビュー」参照)。

カスタムブロックプロパティ 定義済みのブロックプロパティを利用すれば変数データ処理アプリケーションを手軽に実装できますが、これらのプロパティについて PPS は内部的に知っていて自動的に処理できるわけですが、ブロックにもっと高い柔軟性を与えるためにデザイナーがカスタムプロパティを割り当てることも可能です。カスタムプロパティを利用してブロックの概念を拡張することで、より高度な変数データ処理アプリケーションの要請に応えられるでしょう。

カスタムプロパティには何のルールもありません。PPS はカスタムプロパティを何ら処理しないからです。PPS はクライアントがカスタムプロパティを利用できるようにするだけです。クライアントのコードでカスタムプロパティを取得してよしなに処理できます。レイアウトやデータ抽出についてブロックのカスタムプロパティを見て決めるようなアプリケーションが可能です。科学的なアプリケーションでは数値出力の桁数を指定するカスタムプロパティを設けてもよいでしょうし、データベースのフィールド名をカスタムブロックプロパティにしてそこからブロック用データを取得してもよいでしょう。

13.2.3 なぜ PDF のフォームフィールドを使わないの

経験ある Acrobat ユーザーなら疑問に思うでしょう。なぜ我々は新たにブロックという概念を導入したのか、なぜ PDF にすでにあるフォームフィールドのしくみを活用しないのかと。根本的な違いは、PDF のフォームフィールドは対話的に記入されることを主眼として作られているのに対して、PDFlib のブロックは自動的に流し込まれることを目的としている点です。対話的記入と自動流し込みを両方とも要するアプリケーションの場合には、フォームフィールド変換プラグインを用いて PDF フォームと PDFlib ブロックを併用する

ことも可能です (389 ページ「13.4 PDF フォームフィールドを PDFlib ブロックに変換」参照)。

両者は似た点も多いですが、PDFlib ブロックには、PDF フォームフィールドに比べて、表 13.1 に示す利点があります。


表 13.1 PDF フォームフィールドと PDFlib ブロックの比較

機能	PDF フォームフィールド	PDFlib ブロック
設計の趣旨	対話的利用	自動流し込み
文字組版機能 (フォント指定・文字サイズ指定よりも高度な)	—	カーニング・単語間隔・文字間隔・下線 / 上線 / 取り消し線
OpenType レイアウト機能	—	何ダースもの OpenType レイアウト機能 (合字・スワッシュ文字・オールドスタイル数字など)
複雑用字系への対応	制約あり	シェーピング・双方向テキスト (アラビア語・デーヴァナーガリーなど)
フォント制御	フォント埋め込み	フォント埋め込み・サブセット化・エンコーディング
テキスト組版制御	左・中央・右揃え	左・中央・右・両端揃え。各種組版アルゴリズム・制御。インラインオプションを用いてテキストの見映えを制御可能
テキストの途中でフォントなどテキスト属性を変更可能	—	○
書き込んだ内容が PDF のページ記述に取り込まれる	—	○
追加フィールドの内容をユーザーが編集可能	○	×
プロパティの拡張セット	—	○ (カスタムブロックプロパティ)
画像ファイルを流し込める	—	BMP・CCITT・GIF・PNG・JPEG・JBIG2・JPEG 2000・TIFF
ベクトルグラフィックを流し込める	—	SVG
カラー対応	RGB	グレースケール・RGB・CMYK・Lab・スポットカラー (HKS・Pantone スポットカラーが Block Plugin に内蔵)・DeviceN
PDF 各種規格	—	PDF/A・PDF/X・PDF/VT・PDF/UA
流し込みの際にグラフィックやテキストのプロパティをオーバーライド可能	—	○
透過内容	—	○
テキストブロックを連結可能	—	○

13.3 PDFlib Block Plugin でブロックを編集

13.3.1 ブロックを作成

ブロックツールをアクティブにする PDFlib ブロックを作成する Block Plugin は、Acrobat におけるフォームツールに似ています。ブロックツールをアクティブにするとページ上のブロックが全部表示されます。Acrobat の他のツールを選択するとブロックは見えなくなりますが、なくなったわけではありません。ブロックツールをアクティブにするには以下の操作のどちらかを行います：

- ▶ ブロックアイコン  をクリック。これは Acrobat DC で以下の場所にあります：「ツール」→「PDF を編集」をクリック。
- ▶ メニュー項目「PDFlib ブロック」→「PDFlib ブロックツール」。

ブロックを作成・変更 ブロックツールをアクティブにしたら、十字ポインターをドラッグすれば、ページ上の希望の位置に希望の大きさのブロックを作れます。ブロックはつねに長方形で、かつ辺がページの辺と平行になります (*rotate* プロパティを用いるとブロックの内容をページの辺に平行でなくすることができます)。ブロックの長方形をドラッグし終わると、「PDFlib ブロックプロパティ」ダイアログが現れ、ブロックの各種プロパティを編集できます (384 ページ「13.3.2 ブロックプロパティを編集」参照)。ブロックツールはブロックの名前を自動的に生成しますが、この名前はプロパティダイアログで変更可能です。ブロック名はページ内では一意である必要がありますが、別のページで同じ名前を使うのはかまいません。

ダイアログの一番上で、ブロックの種類を「Textline」(テキスト行)・「Textflow」(テキストフロー)・「Image」(画像)・「PDF」・「Graphics」(グラフィック) から選べます。ブロックの種類ごとに色が変わってあります (図 13.1 参照)。タブは、その時選んでいるブロック種別のものでアクティブになります。「PDFlib ブロックプロパティ」ダイアログは、ブロックの種類によって、プロパティをグループやサブグループにまとめて階層的に表示します。

注 PDF にブロックを追加した後や、既存のブロックに変更を加えた後には、Acrobat の「名前を付けて保存 ...」コマンドを使うほうが (「上書き保存」よりも) ファイルサイズが小さくなります。

ブロックを選択 コピー・移動・削除・プロパティ編集などのブロック操作は、選択した 1 個ないし複数のブロックに対して働きます。ブロックツールを用いてブロックを選択するには以下のとおり操作します：

- ▶ ブロック 1 個を選択するには単にそれをシングルクリック。
- ▶ 複数のブロックを選択したい場合は、2 個目以降のブロックを Shift キーを押しながら選択。
- ▶ ページ上のすべてのブロックを選択するには、Ctrl+A (Windows の場合) か Cmd+A (macOS の場合) を押すか、または「編集」→「すべて選択」。

コンテキストメニュー 1 個ないし複数のブロックを選択している時にコンテキストメニューを開くと、ブロックに関するさまざまな機能 (「PDFlib ブロック」メニューから利用できる各種機能と同じ) をさっと実行できます。コンテキストメニューを開くには、選択した 1 個ないし複数のブロックを、Windows ではマウスの右ボタンでクリック、macOS では Ctrl+ クリックします。たとえばブロックを削除するには、ブロックツールで選択し

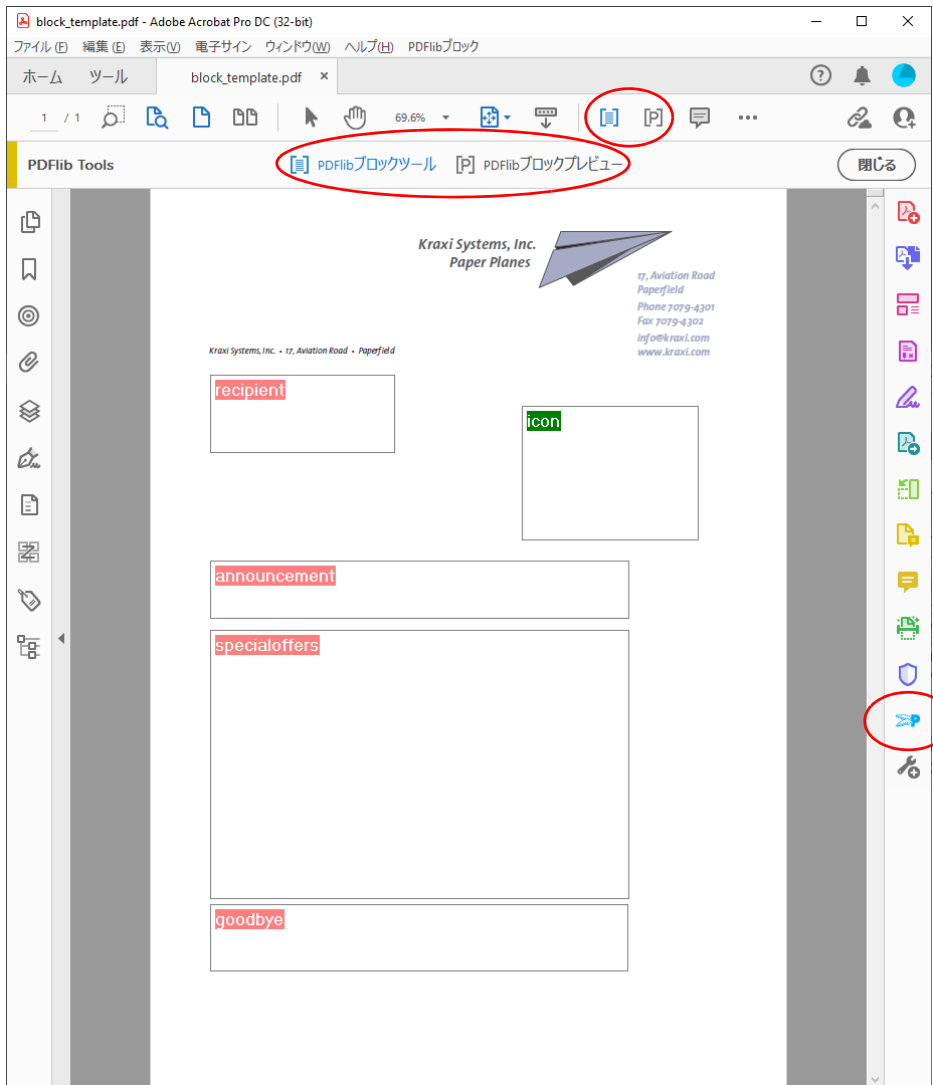


図 13.1
各種ブロック
の視覚化

てから **Delete** キーを押してもいいのですが、コンテキストメニューで「編集」→「削除」という方法もあります。

ページ上でブロックのない所を右クリック（macOS では Ctrl+ クリック）した場合のコンテキストメニューには、ブロックプレビューを作成する項目と、ブロック機能を構成する項目があります。

ブロックのサイズと位置 ブロックツールを使って選択した 1 個ないし複数のブロックを別の位置へ動かすこともできます。Shift キーを押しながらブロックをドラッグすると真横か縦へだけ動きます。ブロックをきっちり揃えるのに便利でしょう。ポインターをブロックの角付近へ持って行くと、ポインターが二重矢印になり、ブロックのサイズを変えることができます。

複数のブロックの位置やサイズを揃えたいときは、複数のブロックを選択して、「PDFlib ブロック」メニューかコンテキストメニューから「整列」・「中央揃え」・「均等配置」・「サ



図 13.2
ブロックプロパティダイアログ

「**移動**」コマンドを選択します。1 個ないし複数のブロックの位置を、矢印キーを使って小刻みに動かすこともできます。

あるいは、ブロックの座標を数値でプロパティダイアログに入力するという方法もあります。この座標系の原点はページの左上隅です。座標の表示は、その時点で Acrobat で選んでいる単位によります：

- ▶ Acrobat DC で表示単位を変更するには、「編集」→「環境設定」(→「一般...」)→「単位とガイド」→「ページと定規の単位」でインチ・センチメートル・パイカ・ポイント・ミリのいずれかを選択。
- ▶ カーソルの座標を表示するには、「表示」→「表示切り替え」→「カーソル座標」。

ただしここで選んだ単位は、**Rect** プロパティにのみ効力を持ち、他の数値プロパティ (**fontsize** など) には効力を持ちません。

グリッドでブロックの位置合わせ Acrobat のグリッド機能を活用すると、ブロックの位置合わせやサイズ変更が正確にできます：

- ▶ グリッドを表示：「表示」→「表示切り替え」→「定規とグリッド」→「グリッド」。

- ▶ グリッドスナップを有効化: 「表示」→「表示切り替え」→「定規とグリッド」→「グリッドにスナップ」。
- ▶ グリッドを変更: 「編集」→「環境設定」(→「一般...」)→「単位とガイド」でグリッドの線の間隔や位置や色を変更できます。

「グリッドにスナップ」を有効にしていると、構成したグリッドにブロックのサイズと位置が揃います。この「グリッドにスナップ」は、新規作成するブロックにも効きますし、既存のブロックをブロックツールで移動・サイズ変更する際にも効きます。

画像やグラフィックを選択することによってブロックを作成 ブロックの長方形を手作業でドラッグせずに、既存のページ内容からブロックのサイズを定義することも可能です。まず、メニュー項目「PDFlib ブロック」→「オブジェクトをクリックでブロックを定義」を有効化します。そのうえでブロックツールを使ってページ上の画像をクリックすると、その画像と同じ位置に同じサイズのブロックができます。それ以外のグラフィックオブジェクトをクリックすることもでき、その場合、ブロックツールはそのグラフィック(たとえばロゴ)全体を選択しようとします。この「オブジェクトクリック」機能の意図は、ブロックを定義する作業を支援することです。できたブロックの位置やサイズは後から自由に変更できます。画像やグラフィックにブロックが固定されてしまうわけではなく位置・サイズ決めの補助として使うだけです。

この「オブジェクトクリック」機能は、ページでどのベクトルグラフィックや画像が論理的要素を形成しているかを認識しようと試みます。ページ内容をクリックすると、それが白い場合と極端に大きい場合でなければ、その外接枠(対象を囲む長方形)が選択されます。その次の段階として、この検知された長方形に他の物の一部が入り込んでいればそれが選択領域に追加され、これが繰り返されます。こうして最終的に得られた領域に基づいてブロックの長方形が生成されるのです。結局のところ「オブジェクトクリック」機能は、個々の線ではなくグラフィック全体を選択しようとするようになります。

フォントプロパティを自動検出 Block Plugin は、テキスト行またはテキストフローブロックの置かれた場所の背景にあるフォントを分析し、対応するブロックプロパティに自動的に書き込むこともできます:

fontname · fontsize · fillcolor · charspacing · horzscaling · wordspacing ·
textrendering · textrise

このフォントプロパティの自動検出は結果が変になることもあるため、背景を無視させたいときには「PDFlib ブロック」→「背景フォント・色の検出」で機能の有効・無効を切り替えることができます。デフォルトではこの機能は無効になっています。

ブロックをロック ブロックがうっかり移動・サイズ変更・削除されぬようロックして保護することもできます。ブロックツールがアクティブな状態でブロックを選択してコンテキストメニューから「ロック」を選択します。ブロックがロックされていると、移動・サイズ変更・削除できず、そのプロパティダイアログを編集することもできません。

13.3.2 ブロックプロパティを編集

新規ブロックを作成した時と、既存ブロックをダブルクリックした時と、ブロックのコンテキストメニューから「プロパティ」を選択した時には、プロパティダイアログが現れ、その選んだブロックのあらゆる設定を編集できます(図 13.2 参照)。プロパティにはブロック種別によってグループがいくつかあり、詳しくは 402 ページ「13.7 ブロックのプロパティ」で説明します。

ダイアログ内の 1 個ないし複数のプロパティを変更した時だけ「適用」ボタンが有効になります。プロパティの変更をブロックに適用した後は、ブロックの名前の中にアスタリスクが表示されて、ブロックが変更されたがまだディスクには保存されていないというを示します。ロックされたブロックでは「適用」ボタンは無効となります。

注 ブロック種別やプロパティの設定によっては表示されないプロパティもあります。たとえば、タブ設定を編集するための「hortabmethod=ruler におけるルーラータブ」プロパティサブグループは、「テキスト組版」→「ルーラータブ」グループの hortabmethod プロパティを ruler に設定しているときのみ表示されます。

注 ブロックプロパティにテキストを入力するとキャラクターの置換が起こることがあります。たとえばストレート引用符がスマート引用符へ置き換えられたりします。この置き換えはオペレーティングシステムが行なっていることですので、無効化するには「システム環境設定」→「キーボード」→「ユーザー辞書」→「スマート引用符とスマートダッシュを使用」を選択します。

プロパティの値を変更するには、入力したい数や文字列をプロパティの入力領域に入力するか (*linewidth* など)、ドロップダウンリストから値を選ぶか (*fitmethod*・*orientate* など)、ダイアログの右側にある「...」ボタンをクリックしてフォントやカラー値やファイル名を選びます (*backgroundcolor*・*defaultimage* など)。*fontname* プロパティの場合は、システムにインストールされているフォントの一覧から選ぶこともできますし、カスタムのフォント名を打ち込むことも可能です。どの方式でフォント名を入力したにせよ、PPS がブロックへ内容を流し込むシステムでそのフォントが利用可能でなければなりません。

変更されたプロパティは、ブロックプロパティダイアログ内で太字で表示されます。ブロックのいずれかのプロパティが変更されているときには、その表示されているブロック名の後に接尾辞 (*) が付されます。プロパティの編集が済んだら、「適用」ボタンをクリックしてブロックを更新します。定義したプロパティは、PDF ファイル内にブロック定義の一部として格納されます。

重なったブロック 重なりあうブロック群は選択しづらいことがあります。その領域をクリックすると必ず最前面のブロックが選ばれてしまうからです。そんなときはコンテキストメニューの「ブロックの選択」項目を用いれば、ブロックのうちの 1 個を名前で見分けることができます。1 個のブロックを選んだ直後にその領域で行う操作は、その選択した 1 個のブロックに対してのみ効力を持ち、他のブロックに対しては効力を持ちません。たとえば *Enter* を押せば、その選択したブロックのプロパティを編集できます。この方法を駆使すれば、ブロックの上に他のブロックが部分的ないし完全にかぶさっていても、簡単にそのブロックのプロパティを編集することができます。

ブロックプロパティの値を繰り返し使用・リセット キー入力やクリックの量をいくらか軽減できるよう、ブロックツールは、直前のブロックのプロパティダイアログで入力されたプロパティ値を記憶しています。これらの値は新規ブロックの作成時に再利用されます。もちろんその値はいつでも別の値でオーバーライドしてかまいません。

プロパティのダイアログで「全てリセット」ボタンを押すと、下記以外のブロックプロパティがデフォルトにリセットされます：

- ▶ *Name*・*Type*・*Rect*・*Description* プロパティ
- ▶ すべてのカスタムプロパティ

注 定義済みプロパティのデフォルト値は、プレビュー生成時のプレースホルダーデータを保持する *defaulttext*・*defaultimage*・*defaultpdf*・*defaultgraphics* プロパティとは別物ですので注意してください (392 ページ「ブロックのデフォルト内容」参照)。

複数のブロックをまとめて編集 複数のブロックのプロパティをまとめて編集すれば大いに時間を節約できます。複数のブロックを選択するには以下のとおり操作します：

- ▶ メニュー項目「PDFlib ブロック」→「PDFlib ブロックツール」を選択することによって、ブロックツールをアクティブにします。
- ▶ 1 個目のブロックをクリックして選択。この最初に選んだブロックがマスターブロックになります。他の 1 個ないし複数のブロックを Shift+ クリックすることによって選択ブロック群に加えます。あるいは「編集」→「すべてを選択」をクリックすると、現在のページの全ブロックを選択できます。
- ▶ これらのいずれかのブロックをダブルクリックすると、ブロックプロパティダイアログが開きます。この時ダブルクリックしたブロックは新たにマスターブロックになります。
- ▶ あるいはいずれかのブロックをクリックしてマスターブロックにしたうえで Enter キーを押すことによってブロックプロパティダイアログを開くこともできます。

プロパティのうち、選択しているブロックすべてに該当する部分集合だけがプロパティダイアログに表示されます。プロパティ値はマスターブロックから採られてダイアログに表示されます。この時、選択しているブロックすべてにプロパティ群を以下のように適用できます：

- ▶ チェックボックス「マスターブロックのすべてのプロパティを適用」をチェックしていないとき：「適用」をクリックすると、選択しているすべてのブロックへ、ダイアログで手変更されたプロパティ（黒でハイライトされている）だけがコピーされます。
- ▶ チェックボックス「マスターブロックのすべてのプロパティを適用」をチェックしているとき：「適用」を押すと、選択しているすべてのブロックへ、マスターブロックのすべてのカレントプロパティと、ダイアログで手変更されたすべてのプロパティがコピーされます。これを利用して、あるブロックのブロックプロパティを他の 1 個ないし複数のブロックへコピーすることもできるわけです。

以下の定義済みプロパティおよびカスタムプロパティは共用できません。すなわち、複数のブロックでまとめて編集できません：

Name · Description · Subtype · Type · Rect · Status

13.3.3 ページ間・文書間でブロックをコピー

Block Plugin は、ブロックを現在のページの中で、または現在の文書の中で、あるいは他の文書へ移動・コピーする手段をいくつか提供しています：

- ▶ ブロックをマウスでドラッグすることによって、移動・コピーするか、あるいは他のページが開いている文書へブロックを貼り付ける
- ▶ ブロックを、通常のコピー/貼り付け操作を用いて、同一文書内の 1 つないし複数のページへ複製
- ▶ ブロックを、新しいファイル（ページが空白の）か既存の文書（既存のページにブロックを適用）へ書き出す
- ▶ 他の文書からブロックを取り込む

ブロックの定義は維持したままページ内容を差し替えたいときは、ブロックを温存しながら背景の 1 つないし複数のページを置換することができます。そのためには、「ツール」→「ページを整理」→「置換」を用います。

ブロックを移動・コピー ブロックの位置を変えたりコピーを作成したりするには、1 個ないし複数のブロックを選択して、Ctrl キー（Windows の場合）か Alt キー（macOS の場

合)を押しながら、新しい位置へドラッグします。キーを押している間は、マウスマウスカーソルが変わります。コピーされたブロックは、名称と位置が自動的に変更される以外は、元のブロックと同じプロパティを持ちます。

また、コピー/貼り付けを使って、ブロックを、同一ページ内の他の場所へ、または同一文書内の他のページへ、あるいはその時に Acrobat で開いている他の文書へコピーすることもできます：

- ▶ ブロックツールをアクティブにして、コピーしたいブロック群を選択。
- ▶ Ctrl+C (Windows の場合) か Cmd+C (macOS の場合) または「編集」→「コピー」を使って、選択したブロックをクリップボードへコピー。
- ▶ コピー先ページへ移動 (必要なら)。
- ▶ ブロックツールがアクティブであることを確認して、Ctrl+V (Windows の場合) か Cmd+V (macOS の場合) または「編集」→「貼り付け」を使って、クリップボードからブロックを現在のページ・文書へ貼り付け。

ブロックを他のページ群へ複製 1 個ないし複数のブロックの複製を、現在の文書の中の任意の数のページ上に一気に作ることもできます：

- ▶ ブロックツールをアクティブにして、複製したいブロック群を選択。
- ▶ 「PDFlib ブロック」メニューかコンテキストメニューから「取り込みと書き出し」→「複製...」を選択。
- ▶ どういうブロックを複製するかを選び(「選択されているブロック」または「このページ上の全ブロック」)、この選んだブロック群を複製したい複製先ページの範囲を選択。

ブロックを書き出し・取り込み ブロックの書き出し/取り込み機能を使うと、ある 1 つのページ上のブロックの定義や、ある文書のすべてのブロックの定義を、複数の PDF ファイル間で共用することが可能です。これは、既存のブロック定義を温存したままページ内容を差し替えるときに便利です。ブロック定義を別ファイルとして書き出すには以下のとおり操作します：

- ▶ ブロックツールをアクティブにして、書き出したいブロック群を選択。
- ▶ 「PDFlib ブロック」メニューかコンテキストメニューから「取り込みと書き出し」→「書き出し...」を選択。ページ範囲と、ブロック定義を持たせたい新規 PDF ファイル名を入力。

ブロック定義を取り込むには「PDFlib ブロック」→「取り込みと書き出し」→「取り込み...」を選択します。ブロックを取り込む際には、取り込んだブロックを文書の全ページに適用するのか、それとも特定のページ範囲にだけ適用するかを選べます。複数のページを選択した場合、ブロック定義は変更されずに各ページへコピーされます。取り込むブロック定義よりも取り込み先範囲のほうがページ数が多い場合には、「テンプレートを繰り返す」チェックボックスを用いることもできます。これをチェックすると、取り込みファイル内のブロックのシーケンスが、現在の文書の中で、文書の終わりに達するまで繰り返されます。

書き出しでブロックを他の文書へコピー ブロックを書き出す際には、そのブロック群を他の文書のページ群へ直接適用することもできます。結果として、ある文書から別の文書へブロック群を転写することができるわけです。そのためには、ブロックの書き出し先として既存の文書を選びます。「既存のブロックを削除」チェックボックスをチェックすると、書き出し先の文書にブロックが存在していてもすべて削除されてから、新たにブロック群がその文書へコピーされます。

13.3.4 Block Plugin のユーザーインターフェイスを XML でカスタマイズ

Block Plugin のユーザーインターフェイスの設定の一部は、Acrobat セッションごとに保管 / 再読み込みされており、XML 構成ファイルで制御可能です。構成ファイルのサンプルとして *factory settings.xml* をディストリビューションに同梱しています。構成が変更されたとき、新しい設定は *user settings.xml* に格納されます。変更された構成は、Acrobat が起動されるたびに読み込まれ、Acrobat が閉じられるときに書き込まれます。この構成ファイルは以下のような場所に格納されています (システムディレクトリーの名前はローカライズされている場合があります。必要に応じて DC を他の Acrobat トラック名へ置き換えてください) :

Windows : C:\Users\<user>\AppData\Local\Adobe\Acrobat\DC\PDFlib\Block Plugin 5
macOS : /Users\<user>/Library/Application Support/Adobe/Acrobat/DC/PDFlib/Block Plugin 5

以下の XML エレメントを用いて構成を手動で変更できます :

- ▶ エレメント */Block_Plugin/MainDialog/CloseOnApply* : ブロックプロパティダイアログの「適用したらダイアログを閉じる」チェックボックスの初期状態を制御。このチェックボックスは、ブロック作成後またはブロックプロパティ変更後にもブロックプロパティダイアログが開いたままになるかどうかを決めるものです。
- ▶ エレメント */Block_Plugin/MainDialog/ApplyAllProps* : ブロックプロパティダイアログの「マスターブロックのすべてのプロパティを適用」チェックボックスの初期状態を制御。このチェックボックスは、選択している複数のブロックへコピーされるのはマスターブロックのすべてのプロパティなのか、それともダイアログで変更したプロパティ群のみかを指定するものです。
- ▶ エレメント */Block_Plugin/FontDialog/ShowBaseFonts* : ブロックプロパティダイアログのフォント一覧 (「書式」プロパティグループの *fontname* プロパティ) に、システムにインストールされているフォント群だけでなくベースフォント 14 種も表示するかどうかを制御。
- ▶ エレメント */Block_Plugin/Command/ControlByClick* : メニュー項目「PDFlib ブロック」→「オブジェクトをクリックでブロックを定義」の初期状態を制御。
- ▶ エレメント */Block_Plugin/Command/DetectFonts* : メニュー項目「PDFlib ブロック」→「背景フォント・色の検出」の初期状態を制御。
- ▶ エレメント */Block_Plugin/Command/KeyAccelerator* : *control* (Windows では Ctrl キーを、macOS では Command キーを示します) ・ *shift* (Shift キーを示します) ・ *control+shift* ・ *none* のいずれかの値をとり、以下のキーボードショートカットに対するアクセラレーターキーを指定 :

A (すべてを選択) ・ C (コピー) ・ I (ブロックプロパティダイアログ) ・ V (貼り付け) ・ X (切り取り)

変更は、Acrobat を次に起動した際に有効になります。キーボードショートカットは実行時には変更できないためです。このエントリーがない場合、アクセラレーターは利用できません。デフォルトは *control* です。

- ▶ エレメント *Configuration/Preferences/PreviewStatusMessage* : 各プレビュー操作後にステータスメッセージダイアログ (「10 個のブロックが処理されました …」など) が表示されるかどうかを制御。

13.4 PDF フォームフィールドを PDFlib ブロックに変換

PDFlib ブロックを手作業で作らず、PDF フォームフィールドをブロックへ自動変換させることも可能です。これが特に役立つのは、複雑な PDF フォームに PPS で自動流し込みできるようにしたい場合や、既存の大量の PDF フォームを自動流し込みできるように変換したい場合などです。1つのページのすべてのフォームフィールドを PDFlib ブロックに変換するには、「PDFlib ブロック」→「フォームフィールドの変換」→「現在のページ」を選択します。文書のすべてのフォームフィールドを変換したければ「全ページ」を選びます。選択したフォームフィールドだけを変換したければ(1個または複数のフォームフィールドを選択するには、Acrobat の「ツール」→「リッチメディア」から「オブジェクトを選択」ツールを選択)、「選択されているフォームフィールド」を選択します。

フォームフィールド変換の詳細 自動フォームフィールド変換においては、「PDFlib ブロック」→「フォームフィールドの変換」→「変換オプション ...」ダイアログで選んでいる種類のフォームフィールドが、テキスト行ブロックかテキストフローブロックに変換されます。デフォルトではすべての種類のフォームフィールドが変換されます。変換されたフィールドの属性は、表 13.3 のとおり、それぞれ対応するブロックプロパティへ変換されます。

同名の複数のフォームフィールド フォームフィールドの場合には同じページに同名のものも複数あっても許されるのですが、ブロックの場合には名称がページ上で一意でなければなりません。ですので、フォームフィールドがブロックに変換される際には、生成されるブロックの名前の末尾に数を付加して一意なブロック名が生成されます (389 ページ「フォームフィールドを対応するブロックに紐付け」も参照)。

なお、Acrobat の内部的な問題により、同名のフォームフィールドが複数あるとフィールドの属性が正しく報告されません。同名の複数のフィールドで属性が異なると、生成されるブロックにはこの属性の違いが反映されません。変換処理はこの場合には警告メッセージを表示し、関連するフォームフィールド群の名前を示します。その場合には、生成されたブロックのプロパティをよくチェックする必要があります。

フォームフィールドを対応するブロックに紐付け 同名のフィールドが複数あった場合 (ラジオボタンなど)、変換先のフォームフィールドの名前は変更されてしまっていますから、どのフォームフィールドがどのブロックになったのかを特定しづらくなります。これは特に、FDF ファイルか XFDF ファイルをソースとしてブロックへ流し込んでその結果をフォーム記入と同じにしたいときに問題となります。

この問題を解決するため、AcroFormConversion プラグインでは、元のフォームフィールドに関する情報を、それに対応するブロックを生成する際に、カスタムプロパティ群として記録します。表 13.2 に、ブロックを正しく特定するために利用できるカスタムプロパティの一覧を示します。プロパティの型はすべて文字列です。

ブロックを対応するフォームフィールドへバインド PDFlib フォームフィールドと生成 PDFlib ブロックとを同期させるために、生成されたブロックを、その対応するフォームフィールドにバインドしておくことができます。言い換えれば、プラグインが内部的にフォームフィールドとブロックとの紐付けを保持するということです。変換処理が再実行される際、バインドされたブロックは、その対応する PDFlib フォームフィールドの属性を反映して更新されます。ブロックがバインドされていると、作業の二度手間が省けて便

表 13.2 ブロックに対応する元のフォームフィールドを同定するためのカスタムプロパティ

カスタムプロパティ	意味
<code>PDFlib:field:name</code>	フォームフィールドの完全修飾名。
<code>PDFlib:field:pagenumber</code>	元の文書でフォームフィールドが存在していたページの番号（文字列で）。
<code>PDFlib:field:type</code>	フォームフィールドの種類。pushbutton・checkbox・radiobutton・listbox・combobox・textfield・signature のうちのいずれか。
<code>PDFlib:field:value</code>	(type=checkbox の場合にのみ可) フォームフィールドの出力値。

表 13.3 PDF フォームフィールドから PDFlib ブロックへの変換

以下の PDF フォームフィールド属性は 以下の PDFlib ブロックプロパティに変換される
全フィールド	
位置	Rect
名前	Name
ツールヒント	Description
一般→一般プロパティ→表示と印刷	Status : 表示 =active 非表示 =ignore 表示 / 印刷しない =ignore 非表示 / 印刷する =active
一版→一般プロパティ→向き	orientate : 0=north、90=west、180=south、270=east
表示方法→テキスト→フォント	fontname
表示方法→テキスト→フォントサイズ	fontsize : 文字サイズ auto は、ブロックの高さの 3 分の 2 の固定文字サイズに変換され、fitmethod は auto に設定されます。複数行のブロック／フィールドにおいては、この組み合わせでは結果として自動的に適切な文字サイズになるので、ブロックの高さの 3 分の 2 という初期値よりも小さくなる場合があります。
表示方法→テキスト→テキストの色	strokecolor・fillcolor
表示方法→境界線と色→境界線の色	bordercolor
表示方法→境界線と色→塗りつぶしの色	backgroundcolor
表示方法→境界線と色→幅	linewidth : 細=1、標準=2、太=3
テキストフィールド	
オプション→整列	position : 左揃え ={left center} 中央 ={center center} 右揃え ={right center}
オプション→デフォルト	defaulttext
オプション→複数行	チェックありならテキストフローブロックを作成 チェックなしならテキスト行ブロックを作成
ラジオボタン・チェックボックス	

表 13.3 PDF フォームフィールドから PDFlib ブロックへの変換

以下の PDF フォームフィールド属性は 以下の PDFlib ブロックプロパティに変換される
「デフォルトでチェック」がオンの場合 :	defaulttext :
オプション→チェックボックススタイル、	チェックマーク =4
オプション→ボタンスタイル	円形 =l 十字形 =8 ひし形 =u 四角形 =n 星形 =H (文字は ZapfDingbats フォントにおける各記号を表します)
リストボックス・コンボボックス	
オプション→選択されている (デフォルト) 項目	defaulttext
ボタン	
オプション→アイコンとラベル→ラベル	defaulttext

利です。フォームが対話的利用のために更新された時に、対応するブロックも自動的に更新されるからです。

ブロックを生成した後に変換元のフォームフィールドを残したくない場合は、「PDFlib ブロック」→「フォームフィールドの変換」→「変換オプション ...」ダイアログの「変換されたフォームフィールドを削除」オプションを選びます。このオプションを選ぶと、変換処理後にフォームフィールドが完全に削除されます。削除されたフィールドに関連づけられていたアクション (JavaScript など) もすべて文書から削除されます。

バッチ変換 フォームフィールドを PDFlib ブロックに変換したい PDF 文書が多数ある場合には、バッチ変換機能を利用して、任意の数の文書を自動処理することも可能です。「PDFlib ブロック」→「フォームフィールドの変換」→「バッチ変換 ...」を選択すれば、バッチ処理ダイアログが現れます :

- ▶ 入力ファイルは、個別に選ぶこともできますし、1 個のフォルダーの中身をすべてまとめて処理させることもできます。
- ▶ 出力ファイルは、入力ファイルと同じフォルダーへ書き出すこともできますし、別のフォルダーへ書き出すこともできます。出力ファイルには、入力ファイルと区別するためにプレフィックスを追加することもできます。
- ▶ 大量の文書を処理する際には、ログファイルを指定することを推奨します。変換後、ログファイルには、処理されたすべてのファイルの一覧と、それぞれの変換結果に関する詳細が書き込まれており、エラーが起きた場合にはエラーメッセージも書き込まれます。

変換処理の間、変換される PDF 文書は Acrobat で表示されますが、変換が完了するまで、Acrobat のメニュー機能やツールは使用できません。

13.5 ブロックを Acrobat でプレビュー

注 PDFlib ディストリビューションの中の block_template.pdf 文書でプレビュー機能を試せます。必要なリソース（フォント・画像など）も PDFlib ディストリビューションに含まれています。

PDFlib ブロックは PPS によって処理されます。PPS を用いることで、ブロックへの流し込み処理について、そのデータソース（データベース内のテキストや、ディスク上の画像ファイルなど）や、生成される文書の書式・対話的性質をカスタマイズすることができます。この処理について詳しくは 397 ページ「13.6 PPS でブロックに流し込み」で説明します。

しかし Block Plugin には内蔵バージョンの PPS が含まれており、これを用いて、プログラミングを一切必要とせずに Acrobat 上で、流し込まれたブロックのプレビューバージョンを生成することができます。このプレビュー機能は、カスタムプログラミングと同等の柔軟性を提供することはできませんが、ブロックへの流し込み結果を手軽に眺めるには適しています。ブロックプレビューを活用すれば、ブロックの位置やサイズを改善したり、ブロックのプロパティ（フォント名・文字サイズなど）をチェックしたりすることができます。プレビューの表示結果に満足するまで、ブロックを変更し、プレビューを生成しなおすことができます。プレビューは、カレントページについても、文書全体についても生成できます。


プレビューはつねに新規 PDF 文書として表示されます。元の文書（ブロックを有している）は、プレビューを生成しても変更を受けません。プレビュー文書は、必要に応じて保存することもできますし、捨てることもできます。

ブロックのデフォルト内容 プラグインの場合には、ブロックのテキスト・画像・ベクトルグラフィック・PDF 内容をサーバーサイドのデータソース（データベースなど）から取得するのは無理ですので、プレビュー機能ではつねにブロックのデフォルト内容が用いられます。具体的には *defaulttext*・*defaultimage*・*defaultpdf*・*defaultgraphics* プロパティで指定した値が用いられます。PPS で使う現実のブロック内容を代表するようなサンプルデータ集合をデフォルトデータとして使うのが普通です。デフォルト内容のないブロックはプレビュー生成の際に無視されます。*Status=ignoredefault* のブロックについても同様です。

新規ブロックでは、デフォルトプロパティは空です。プレビュー機能を使うには、「デフォルト内容」プロパティグループの *defaulttext*・*defaultimage*・*defaultpdf*・*defaultgraphics* プロパティ（ブロックの種類による）に書き込むか、「高度な PPS オプション ...」ダイアログの同名オプションに適切な値を与える必要があります。

注 記号フォントのデフォルトテキストを入力するのは少しトリッキーな場合があります。詳しくは 396 ページ「デフォルトテキストに記号フォントを使用」を参照してください。

ブロックプレビューを生成 ブロックプレビューを生成するには、以下のいずれかの方法を用います：

- ▶ PDFlib ブロックプレビューアイコン  をクリック。これは Acrobat DC で以下の場所にあります：「ツール」→「PDF を編集」をクリック。
- ▶ メニュー項目「PDFlib ブロック」→「プレビュー」→「プレビューの生成」。
- ▶ ブロックツールがアクティブの時は、どのブロックもない所を右クリックすれば、コンテキストメニューに項目「プレビューの生成」と「プレビュー設定 ...」が現れます。

プレビューは、ディスク上の PDF ファイルに基づいて作成されます。Acrobat 上で変更を行っていた場合には、ブロック PDF を「ファイル」→「上書き保存」または「ファイ

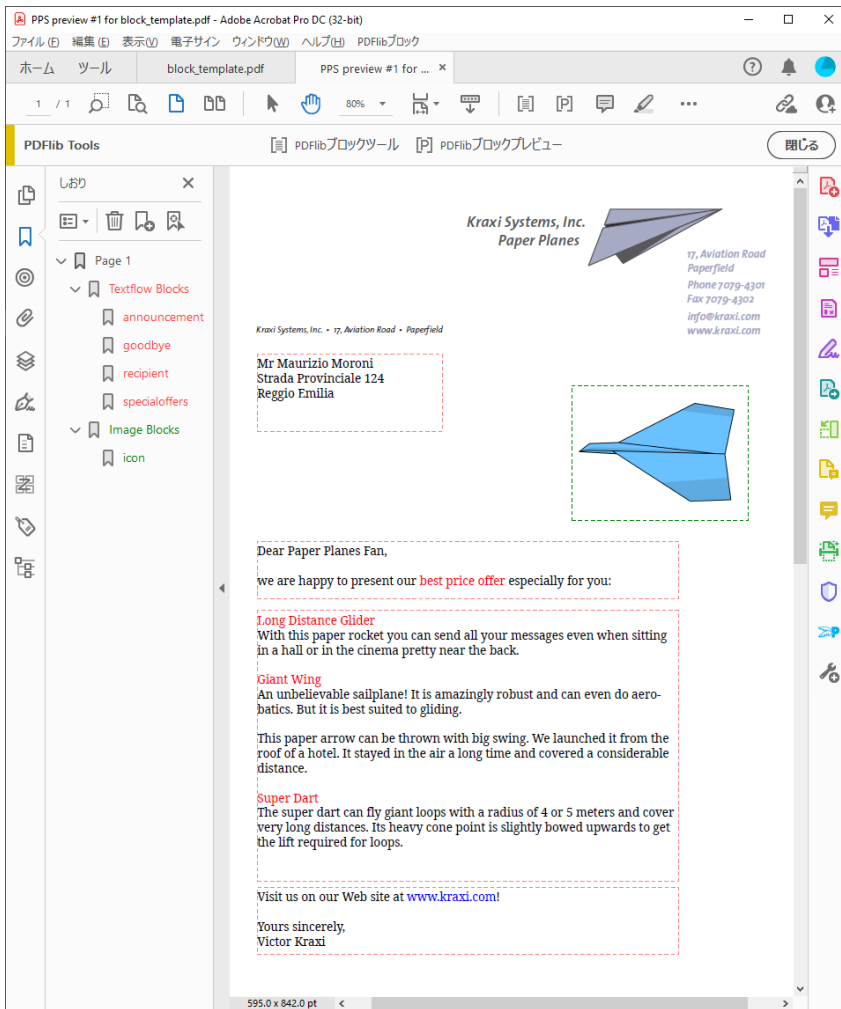


図 13.3
 図 13.1 の文書のプレビュー PDF。ブロック情報レイヤーと注釈を内容として持っています。

ル」→「名前を付けて保存 ...」を用いてディスクに保存してはじめて、その変更はプレビューに反映されます。変更を受けたブロックは、ブロック名の後にアスタリスクが付いていることで識別できます。プレビュー作成前にブロック PDF が自動保存されるようプレビュー機能を構成することも可能です。そうすれば、対話的に行なった変更が確実に、ただちにプレビュー内で反映されます。

プレビューを構成 ブロックプレビューの生成と、その基礎をなす PPS の動作について、いくつかの性質を、「PDFlib ブロック」→「プレビュー」→「プレビュー設定 ...」で構成できます：

- ▶ 現在のページをプレビューするか、それとも文書全体をプレビューするか。
- ▶ 生成されるプレビュー文書の出力ディレクトリー。
- ▶ ブロック PDF をプレビュー作成前に自動保存。
- ▶ ブロック情報レイヤーと注釈を追加。
- ▶ 生成される出力へブロック群をコピー。

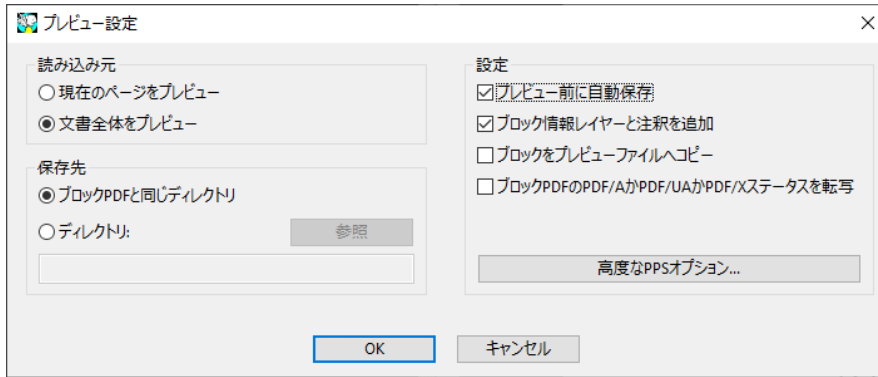


図 13.4 ブロックのプレビューを構成

- ▶ 「**ブロック PDF の PDF/A か PDF/UA か PDF/X ステータスを転写**」: これらの規格では、レイヤーと注釈の使用は制限されていますので、「**ブロック情報レイヤーと注釈を追加**」オプションとこのオプションは同時には使えません。
- ▶ 「**ブロックをプレビューファイルへコピー**」を使うと、流し込みの際、生成されるプレビューへ PDF ブロック群をコピーできます。ブロックへの流し込みが成功したか否かにかかわらず、すべてのブロックがコピーされます。
- ▶ 「**高度な PPS オプション**」ダイアログを使うと、PPS の関数のオプションリストを PPS の API に従って追加指定できます。たとえば、ブロックへの流し込みに使うフォントや画像が位置しているディレクトリを `PDF_set_option()` の `searchpath` オプションを用いて指定できます。この高度なオプションを指定する際には、PPS のコードの開発者と連携して行うことを推奨します。

ブロックの順序 文書が Acrobat の「別名で保存 ...」で保存される際には、そのブロック群は、そのブロック名に従ってアルファベット順に並べ替えられます。ブロック群がプレビューによって処理される順序も、pCOS によって報告される順序も、これと同じです。とはいえ、各種アプリケーションでは通常、ブロックへの流し込みを、その名称に基づいて行いますので（ファイル内での格納順によってではなく）、PDF 文書内での順序が問題となることは通常ありません。

プレビューで提供される情報 生成されるプレビュー文書には、元のページ内容（背景）と、流し込みが行われたブロックのほかに、さまざまな情報も含まれることがあります。この情報は、ブロックや PPS 構成のチェックや改善に役立つものです。デフォルト内容を持つアクティブなブロックそれぞれについて、以下のアイテムが生成されます：

- ▶ **エラーマーカー**：流し込みが成功しなかったブロックは、打ち消し線の付いた長方形として視覚化されていますので、容易に識別できます。エラーマーカーは、ブロックが処理できなかったときには必ず作成されます。
- ▶ **しおり**：ブロックの処理結果はしおりにまとめられます。このしおりは、ページ番号とブロックの種類に従って、かつエラーが起きたときはエラーにも従って、構造化されています。しおりは「表示」→「表示切り替え」→「ナビゲーションパネル」→「しおり」（Acrobat X/XI/DC）で表示できます。しおりは必ず作成されます。
- ▶ **注釈**：処理されたブロックごとに、ブロック内容そのものに加えて、ページ上に注釈が作成されます。この注釈の長方形は、元のブロックの枠を視覚化しています（デフォ

ルト内容と流し込みモードによっては、これはブロック内容の枠とは異なる場合があります)。この注釈の中にはブロックの名前が入っており、ブロックへの流し込みができなかったときにはエラーメッセージも入っています。注釈はデフォルトで生成されますが、プレビューの構成で無効化することもできます。PDF/A-1・PDF/X 規格では注釈の使用は制限されていますので、「**ブロック PDF の PDF/A か PDF/UA か PDF/X ステータスを転写**」オプションを有効にすると、注釈は生成されません。

- ▶ **レイヤー**：ページの内容は、分析とデバッグが容易になるよう、複数のレイヤーに分けて配置されます。ページ背景（すなわち元のページの内容）、ブロックの種類毎、流し込みができなかったエラーブロック、ブロック情報を持った注釈について、それぞれ別々のレイヤーが作成されます。空のレイヤーは生成されません（エラーが何も起きなかった場合など）。レイヤーの一覧を「**表示**」→「**ナビゲーションパネル**」→「**レイヤー**」で表示できます。デフォルトでは、ページのすべてのレイヤーが表示されます。いずれかのレイヤーの内容を見えなくするには、レイヤー名の左にある眼のアイコンをクリックします。レイヤーの生成は、プレビューの構成で無効化することも可能です。PDF/A-1・PDF/X-3 規格ではレイヤーの使用は制限されていますので、「**ブロック PDF の PDF/A か PDF/UA か PDF/X ステータスを転写**」オプションを有効にすると、レイヤーは生成されません。

PDF/A か PDF/UA か PDF/X ステータスを転写 「ブロック PDF の PDF/A か PDF/UA か PDF/X ステータスを転写」構成は、これらの規格に従った PDF 出力を生成する必要があるときに有用です。転写モードは、入力以下いずれかの規格に準拠しているときに有効にできます：

PDF/A-1a:2005, PDF/A-1b:2005
PDF/A-2a, PDF/A-2b, PDF/A-2u
PDF/A-3a, PDF/A-3b, PDF/A-3u

PDF/UA-1

PDF/X-3:2003
PDF/X-4, PDF/X-4p
PDF/X-5n

プレビューが転写モードで作成されるときは、PPS は、ブロック PDF の以下の性質を、生成するプレビューへ複製します：

- ▶ PDF 規格識別。
- ▶ 出力インテント条件。
- ▶ ページ寸法。すべてのページ枠を含みます。
- ▶ タグ付き PDF：文書言語（もしあれば）。
- ▶ XMP 文書メタデータ。

規格準拠の PDF 文書を転写する際には、すべてのブロック流し込み操作が、それぞれの規格に準拠している必要があります。たとえば、出力インテントがなければ、ICC プロファイルのない RGB 画像は使えません。同様に、使用フォントはすべて埋め込む必要があります。すべての要件の一覧を、346 ページ「12.2 PDF/A によるアーカイビング」と、358 ページ「12.3 PDF/X による印刷出力」に示しています。PDF/A または PDF/X 転写モードでのブロック流し込み操作が、選択されている規格に違反するときには（デフォルト画像が RGB 色空間を用いているにもかかわらず文書が適切な出力インテントを含んでいない場合など）、エラーメッセージが表示され、プレビューは生成されません。これによりユーザーは作業フローのかなり早い時点で規格違反の危険に気づけます。

デフォルトテキストに記号フォントを使用 記号フォントを用いているブロックにデフォルトテキストを与える方法は2通りあります：

- ▶ 8ビットレガシーコードを使用する。このコードはたとえばWindowsの文字コード表アプリケーションで表示されます。この8ビットコードを *defaulttext* に与えます。そのためには具体的には、対応する8ビットキャラクターを直接的に入力するか（たとえばWindowsの文字コード表からコピー・ペーストすることによって）、あるいは数値のエスケープシーケンスを用います。この場合には、「テキスト作成」プロパティグループの *charref* プロパティをデフォルト値 *false* のままにしておく必要があり、文字参照は使用できません。たとえば、以下のデフォルトテキストは、*charref=false* であれば、記号フォント Wingdings の「スマイリー」グリフを生成します：

```
J  
\x4A  
\112
```

- ▶ Unicode 値か、フォント内で用いられているグリフ名を使用する。「テキスト作成」プロパティグループの *charref* プロパティを *true* に設定し、記号の文字参照かグリフ名参照を与えます（122 ページ「5.4.2 文字参照」を参照）。たとえば、以下のデフォルトテキストは、*charref=true* であれば、記号フォント Wingdings の「スマイリー」グリフを生成します：

```
&#xF04A;  
&.smileface;
```

どちらのやり方でも、ブロックプロパティダイアログでは、実際の記号グリフではなく、代替表現が表示されることに留意してください。

13.6 PPS でブロックに流し込み

PPS でブロックに流し込みを行うには、まずブロックを含むページを、`PDF_fit_pdi_page()` 関数で出力ページ上に貼り付ける必要があります。ページを貼り付けた後、その上のブロックへ `PDF_fill_*block()` 関数群で流し込みを行うことができます。

簡単な例：可変テキストをテンプレートに追加 PDF テンプレートへの動的テキストの追加は、非常に頻繁に必要となる動作です。以下のコード断片は、入力 PDF 文書（テンプレート、ブロックコンテナ）の中のページを開き、それを出力ページ上に配置し、そして `firstname` というテキストブロックに可変テキストを入れ込んでいます：

```
doc = p.open_pdi_document(filename, "");
if (doc == -1)
    throw new Exception("エラー：" + p.get_errmsg());

page = p.open_pdi_page(doc, pageno, "");
if (page == -1)
    throw new Exception("エラー：" + p.get_errmsg());

p.begin_page_ext(width, height, "");
/* 取り込んだページを貼り付け */
p.fit_pdi_page(page, 0.0, 0.0, "");

/* 貼り付けたページ上のブロック1個へ流し込み */
p.fill_textblock(page, "firstname", "Serge", "encoding=winansi");

p.close_pdi_page(page);
p.end_page_ext("");
p.close_pdi_document(doc);
```

クックブック 完全なコードサンプルがクックブックの `blocks/starter_block` トピックにあります。

ブロックのプロパティをオーバーライド 場合によっては、プログラマーが、ブロックの定義が与えているプロパティ群を一部だけ採用し、その他のプロパティをカスタムの値でオーバーライドしたいことがあります。これはさまざまな場合に有用です：

- ▶ 業務上の要請で特定のオーバーライドが必要と判断される場合に対応。
- ▶ 画像・PDF ページの拡張倍率を、ブロック定義から採らずに、アプリケーションで算出。
- ▶ ブロックの座標をプログラムで変える。生成したい請求書のデータ項目数が一定でない場合など。
- ▶ 別々のスポットカラー名を与えることも可能。プリントサービス業務で、顧客ごとの要請に合わせるため。

プロパティをオーバーライドするには、`PDF_fill_*block()` 関数群のオプションリストにプロパティの名前と値を与えます。例：

```
p.fill_textblock(page, "firstname", "Serge", "fontsize=12");
```

これは、ブロックの内部の `fontsize` プロパティを、与えた値 12 でオーバーライドします。ほとんどすべてのプロパティ名を、オプションとして用いることが可能です。

プロパティのオーバーライドは、それぞれの関数呼び出しにのみ適用されます。ブロック定義内に保持されるわけではありません。

テキストフローブロックを流し込みと同時に移動させる テキストフローブロックはサイズが固定なので、そのテキスト内容が変わるとそぐわない場合があります。テキストがわずかしかないときには、2つのブロックの間にアキが生じてしまうかもしれません。テキストが多すぎる場合には、そのブロックの長方形に収まりきらないこともありえます。そのような場合には、テキストフローの流し込みの結果をクエリーして次のブロックの位置を調整することが可能です：

- ▶ デフォルトの *fitmethod* は *auto* ですので、テキストがブロックの長方形にむりやりはめ込まれます。テキストが多い時にブロックからあふれてもよいことにするには、*fitmethod* を *nofit* に設定する必要があります。これを指定するには、デザイン時にブロックテンプレート内でブロックプロパティで指定するか、*PDF_fill_textblock()* に *fitmethod* オプションを与えます。
- ▶ *PDF_fill_textblock()* にダミーオプション *textflowhandle=-1* (PHPでは *textflowhandle=0*) を与えると、このメソッドは、ブロックの内容のテキストフローハンドルを返します。
- ▶ 返されたテキストフローハンドルを *PDF_info_textflow()* に与えることによって、キーワード *textendy* を用いてテキストの終点をクエリーします。
- ▶ *PDF_pcos_get_number()* と pCOS パス *pages[...]/blocks/<blockname>/Rect[1]* を用いてブロックの下辺の位置をクエリーします。
- ▶ この2つの値の差をとります。このオフセットが正なら、テキストフローがブロックを満たしきらなかつたということです。負なら、テキストフローがブロックからあふれたわけです。いずれの場合も、次のブロックを、このオフセットの分だけ上か下へ動かせばよいのです。これを実現するには、*PDF_fill_textblock()* の *refpoint* オプションで *Rect* プロパティをオーバーライドします。このオプションでは絶対座標が必要ですから、ブロックの縦位置をクエリーする必要があります (前の手順を参照)、そして元の位置とオフセットの和を *refpoint* オプションに与える必要があります。
- ▶ この方式は、任意の数のテキストフローブロックに対しても、各ブロックのオフセットを累積させることによって、適用してゆくことができます。各ブロックの内容に応じて、その次のブロックを、上か下へ、適切な量だけ動かしていきます。

クックブック 完全なコードサンプルが *starter_block* サンプルにあります。

流し込んだブロックの上に取り込んだページを配置 取り込んだページは、どのブロック流し込み関数を使うよりも前に、出力ページに配置しておく必要があります。ということは元ページは通常、ブロック内容よりも下層に配置されることとなります。しかし場合によっては、流し込みが行われたブロックよりも上層に元ページを配置したいこともあるでしょう。これを実現するには、ページをいったん *PDF_fit_pdi_page()* の *blind* オプションを用いて貼り付けることにより、そのページのブロックとその位置を PPS に知らせておき、ブロックへの流し込みが済んだ後にページを再び貼り付けることにより、実際にページ内容を表示させます：

```
/* ブロックを用意するためにページをblindモードで配置してページを見えなくする */  
p.fit_pdi_page(page, 0.0, 0.0, "blind");
```

```
/* ブロックへ流し込み */  
p.fill_textblock(page, "firstname", "Serge", "encoding=winansi");  
/* ... いろいろなブロックへ流し込み ... */
```

```
/* ページを再度配置、今度は見えるように */  
p.fit_pdi_page(page, 0.0, 0.0, "");
```

クックブック 完全なコードサンプルがクックブックの `blocks/block_below_contents` トピックにあります。

ブロックへ流し込む際にコンテナページを無視 取り込んだブロックは、そのブロックの背景のページ内容を一切参照せずに、プレースホルダーとして使ってもよいでしょう。ブロックを持つコンテナページをブラインドモードで、すなわち `PDF_fit_pdi_page()` で `blind` オプションを指定して、1 個ないし複数のページ上に貼り付けたうえで、ブロックへの流し込みを行うという方法です。こうすれば、出力ページ上にコンテナページを貼り付けることなく、ブロックやそのプロパティの利点を活用することができ、また、ブロックを複数のページ上へ（または同一出力ページ上へも）複製することが可能になります。

クックブック 完全なコードサンプルがクックブックの `blocks/duplicate_block` トピックにあります。

テキストフローブロックを連結 テキストフローブロックは、前のブロックからあふれたテキストが次のブロックに入るよう、連結することが可能です。たとえば、長い可変テキストがあって、別のページへ続かせる必要が想定される場合、2 個のブロックを連結しておけば、1 個目のブロックがいっぱいになっても、残りは 2 個目のブロックへ流し込まれます。

PPS は、`PDF_fill_textblock()` とブロックプロパティに与えられたテキストから内部的に 1 つのテキストフローを生成します。連結されていないブロックの場合には、このテキストフローはそのブロックの中に配置され、そのテキストフローハンドルは呼び出しが終わった時点で削除され、あふれたテキストは失われます。

連結されたテキストフローの場合には、最初のブロックへ流し込んだ後に余っているあふれテキストを、その次のブロックへ流し込むことができます。最初のテキストフローの余りがブロック内容として使われ、新たなテキストフローは生成されません。このテキストフローブロックの連結は以下のように動作します：

- ▶ 連結されたテキストブロックチェーンのうちの最初の `PDF_fill_textblock()` を呼び出す際に、`textflowhandle` オプションに値 `-1` (PHP の場合：0) を与える必要があります。`PDF_fill_textblock()` は内部的に生成されたテキストフローハンドルを返しますので、これをアプリケーション側で保持しておく必要があります。
- ▶ 次に `PDF_fill_textblock()` を呼び出す際に、前段で返されたテキストフローハンドルを `textflowhandle` オプションに与えることができます（この時 `text` 引数にテキストを与えても無視されるので空にするべきです）。ブロックへ、テキストフローの余りが流し込まれます。
- ▶ この処理を、さらなるテキストフローブロック群に対して繰り返すことができます。
- ▶ 返されたテキストフローハンドルを `PDF_info_textflow()` に与えればブロック流し込みの結果を知ることができます。終了状況やテキストの終了位置などがわかります。

なお、`fitmethod` プロパティは `clip` に設定する必要があります（`textflowhandle` を与えている場合にはこれがデフォルトです）。テキストフローブロックを連結する基本的なコード断片は以下ようになります：

```
p.fit_pdi_page(page, 0.0, 0.0, "");
tf = -1;

for (i = 0; i < blockcount; i++)
{
    String optlist = "encoding=winansi textflowhandle=" + tf;
    int reason;
    tf = p.fill_textblock(page, blocknames[i], text, optlist);
}
```

```

text = null;

if (tf == -1)
    break;

/* いちばん最近のfit_textflow()呼び出しの結果をチェック */
reason = (int) p.info_textflow(tf, "returnreason");
result = p.get_string(reason, "");

/* テキストが全部配置されたならループを抜ける */
if (result.equals("_stop"))
{
    p.delete_textflow(tf);
    break;
}
}

```

クックブック 完全なコードサンプルがクックブックの blocks/linked_textblocks トピックにあります。

ブロックの流し込み順序 ブロック関数群 *PDF_fill_block()* は、プロパティとブロック内容を、以下の順序で処理します：

- ▶ 背景：*backgroundcolor* プロパティが存在し、*None* 以外の色空間キーワードを持っているときは、指定された色でブロック領域が塗られます。
- ▶ 枠線：*bordercolor* プロパティが存在し、*None* 以外の色空間キーワードを持っているときは、指定された色と線幅でブロックの枠が描線されます。
- ▶ 内容：与えられたブロック内容と、*bordercolor*・*linewidth* 以外のすべてのプロパティが処理されます。
- ▶ テキスト行・テキストフローブロック：テキストもデフォルトテキストも与えられていないときは、何も出力されません。背景色やブロックの枠線もありません。

ネストされたブロック ブロックへ流し込みを行う前には、そのブロックを含むページを出力ページ上にまず貼り付ける必要があります（そうでないと、ページを拡張・回転・平行移動した後のブロックの位置を PPS が知りえないため）。ページをブロックのコンテナとしてのみ使っており、静的内容を新ページへコピーしなくてよい場合には、取り込んだページを、*blind* オプションを用いて貼り付けることができます。

取り込んだページを、どのような方法で出力ページ上に貼り付けても、ブロックへの流し込みは行うことができます：

- ▶ ページを *PDF_fit_pdi_page()* で直接貼り付け。
- ▶ ページをテーブルセル内に *PDF_fit_table()* で間接的に貼り付け。
- ▶ ページを他の PDF ブロックの内容として *PDF_fill_pdfblock()* で貼り付け。

この 3 番目の方法、すなわち PDF ブロックへ、ブロックを含む他のページを流し込むという方法を用いると、ブロックコンテナをネストすることができます。これを活用すると、面白い使い方を簡単に実装できます。たとえば、2 段階のブロック流し込み処理で、組み付けとパーソナライゼーションの両方を実装することができます：

- ▶ 第一層のブロックコンテナページには、いくつかの大きな PDF ブロックを置きます。これらは、印刷する紙の上の主要な領域を表しています。PDF ブロックの配置は、想定している紙の後工程を反映しています（折り・断裁など）。
- ▶ この第一層の PDF ブロックそれぞれへ、第二層のコンテナ PDF ページを流し込みます。この PDF ページには、テキスト・画像・PDF・グラフィックのうちのいずれかのブロッ

クを置いておき、それらへ可変テキストを流し込んでパーソナライゼーションを行います。

この方法でブロックコンテナをネストできます。ブロックのネストは何重でも可能ですが、三重以上のネストが必要になることはまれでしょう。

この第二層のブロックコンテナ（レターのテンプレートページなど）は、それぞれの組み付けページで同じにすることもできますし、別のものにすることもできます。もし同じにした場合には、まずレターテンプレート上のブロック群への流し込みを行ってから、そのレターテンプレート自体を次の第一層ブロック内に貼り付ける必要があります。なぜなら、PPS はつねに、テンプレートページがもっとも最近に配置された位置を用いるからです。

クックブック 完全なコードサンプルがクックブックの `blocks/nested_blocks` トピックにあります。

ブロックの座標 ブロックの長方形の座標は、PDF のデフォルト座標系を参照しています。ブロックを含んだページを PPS で出力ページに配置するときには、`PDF_fit_pdi_page()` に対していくつかの位置付け・拡張オプションを与えることができます。これらのオプションは、そのブロックが処理される際に考慮されます。これを利用すると、1つのテンプレートページを出力ページ上に何度でも配置して、そのたびにそのブロック群ヘデータを流し込むことができます。たとえば1枚の組み付け紙上に、1つの名刺テンプレートを4回配置するといったことが可能です。ブロック関数群は、座標系の変換を正しく行い、すべてのブロックに対して、それがページ上に配置されるたびに、正しくテキストを配置します。クライアントに求められるのはただ、ページを配置して、そしてその配置したページ上のすべてのブロックを処理することだけです。以後はそのページを、出力ページ上の他の場所に配置したうえで、その新しい場所に対してさらにブロック処理操作を行うことができ、これを繰り返していくことが可能です。

Block Plugin におけるブロック座標の表示のされ方は、PDF ファイル内に格納されているものとは異なっています。プラグインでは Acrobat の方式を用いて、座標の原点をページの左上隅に置いています。内部座標（ブロック内に格納されているもの）では PDF の方式を用いて、座標の原点をページの左下隅に置いているためです。プロパティダイアログの座標表示は、Acrobat で指定されている単位にも従います（382 ページ「ブロックのサイズと位置」参照）。

ブロックプロパティでスポットカラー ブロックプロパティで特色（スポットカラー）を使うには、「...」をクリックすれば、すべての HKS・Pantone スポットカラーの一覧を表示できます。これらのカラー名は PPS に内蔵されており、ことさら準備なしに使用できます。カスタムスポットカラーに対しては、Block Plugin で代替色を定義することが可能です。ブロックプロパティで代替色を指定していないときは、PPS アプリケーションで `PDF_makespotcolor()` か 然るべきカラーオプションリストを用いてカスタムスポットカラーをあらかじめ定義しておく必要があります。そうでないとブロックへの流し込みは失敗します。

13.7 ブロックのプロパティ

PPS と Block Plugin には、すべての種類のブロックに適用できる一般プロパティ群と、ブロックの種類「テキスト行」・「テキストフロー」・「画像」・「PDF」・「グラフィック」にそれぞれ特有のプロパティ群が用意されています。

プロパティは、ハンドルとアクションリスト以外、オプションリストと同じデータ型をサポートしています。

ブロックプロパティの名前は一般に、`PDF_fit_textline()` や `PDF_fit_image()` など API 関数のオプションと同じです (たとえば `fitmethod` や `charspacing`)。その場合には動作は、それぞれ対応するオプションの説明に書いてあるものとまったく同じです。

13.7.1 管理プロパティ

管理プロパティ群はすべての種類のブロックに適用されます。必須エントリー群は Block Plugin によって自動生成されます。管理プロパティの一覧を表 13.4 に示します。

表 13.4 管理プロパティ

キーワード	とりうる値・解説
Description	(文字列) ブロックの機能に関する、人が読める説明。エンコーディングは PDFDocEncoding か Unicode (後者の場合は先頭 BOM)。このプロパティは、ユーザーへの情報提供のためだけにあり、PPS からは無視されます。
Locked	(論理値) true にすると、ブロックとそのプロパティを Block Plugin で編集できなくなります。このプロパティは PPS からは無視されます。デフォルト : false
Name	(文字列、必須) ブロックの名称。このブロック名は、ページごとに一意である必要がありますが、文書内では一意でなくてもかまいません。3 種のキャラクター [] / は、ブロック名には使えません。ブロック名は最長 125 文字です。
Subtype	(キーワード、必須) ブロックの種類によって、Text・Image・PDF・Graphics のいずれか。なお、テキスト行ブロックとテキストフローブロックはどちらも Subtype が Text になり、両者は textflow プロパティで区別されます。
textflow	(論理値) 一行処理か複数行処理かを制御。このプロパティは、Block Plugin のユーザーインターフェイスでは表示されず、それぞれテキスト行ブロックとテキストフローブロックへマップされます (デフォルト : false) : false テキスト行ブロック : テキストが一行で組まれ、 <code>PDF_fit_textline()</code> で処理されます。 true テキストフローブロック : テキストが複数行にわたることができ、 <code>PDF_fit_textflow()</code> で処理されます。標準テキストプロパティ群のほかにテキストフロー関連プロパティ群も指定できます (表 13.9 参照)。
Type	(キーワード、必須) つねに Block

13.7.2 長方形プロパティ

長方形プロパティ群はすべての種類のブロックに適用され、ブロックの長方形本体の書式を記述します。必須エントリー群は Block Plugin によって自動生成されます。長方形プロパティの一覧を表 13.5 に示します。

表 13.5 長方形プロパティ

キーワード	とりうる値・解説
background-color	(色) このプロパティを指定して None 以外の色空間キーワードを与えると、長方形が描かれ、与えた色で塗られます。既存のページ内容を覆い隠したいときに有効です。デフォルト : None
bordercolor	(色) このプロパティを指定して None 以外の色空間キーワードを与えると、長方形が描かれ、与えた色で描線されます。デフォルト : None
linewidth	(float. 正の値でなければならない) ブロックの長方形の描画に使用する線の描線幅。bordercolor 設定時のみ有効です。デフォルト : 1
Rect	(長方形、必須) ブロックの座標。座標原点はページ左下隅。ただし Block Plugin では、座標は Acrobat の方式で、すなわちページ左上隅を原点として表されます。座標の単位は、Acrobat ではその時点で選択されている単位で表示されますが、PDF ファイル内部ではつねにポイント単位で格納されています。
Status	(キーワード) PPS とブロック機能がブロックを処理する方式を記述 (デフォルト : active) : active ブロックをそのプロパティに従って完全に処理。 ignore ブロックを無視。 ignoredefault defaulttext/image/pdf/graphics プロパティ・オプションが無視される、すなわち可変内容がないとき (特にプレビューで) ブロックが空のままになる点を除き、active と同じ。これは特に、ブロックがプレビュー生成のためのデフォルト内容を持っているかもしれないけれども、サーバーサイドでブロックへ流し込みが行われる際にはそのブロックのデフォルト内容が用いられないようにしたいときに有効です。また、ブロックをプレビューする際にも、ブロックプロパティからデフォルト内容を削除することなくデフォルト内容を無効化するために用いることができます。 static 可変内容を配置しない。ブロックにデフォルトのテキスト・画像・PDF・グラフィック内容があれば、それが用いられます。

13.7.3 書式プロパティ

書式プロパティ群は組版の詳細を指定します：

- ▶ 透過書式プロパティの一覧を表 13.6 に示します。これらはすべての種類のブロックに適用されます。
- ▶ テキスト書式プロパティの一覧を表 13.7 に示します。これらはテキスト行ブロックとテキストフローブロックに適用されます。

表 13.6 すべてのブロック種別に対する透過書式プロパティ

キーワード	とりうる値・解説
<i>blendmode</i>	(キーワードリスト。PDF/A-1 モードで用いられるときは値 <code>Normal</code> を持つ必要があります) ブレンドモードの名前： <code>None</code> ・ <code>Color</code> ・ <code>ColorDodge</code> ・ <code>ColorBurn</code> ・ <code>Darken</code> ・ <code>Difference</code> ・ <code>Exclusion</code> ・ <code>HardLight</code> ・ <code>Hue</code> ・ <code>Lighten</code> ・ <code>Luminosity</code> ・ <code>Multiply</code> ・ <code>None</code> ・ <code>Normal</code> ・ <code>Overlay</code> ・ <code>Saturation</code> ・ <code>Screen</code> ・ <code>SoftLight</code> 。デフォルト： <code>None</code>
<i>opacityfill</i>	(float。PDF/A モードで用いられるときは値 <code>1</code> を持つ必要があります) 塗り操作の不透明度を範囲 <code>0</code> ~ <code>1</code> で表したものの。値 <code>0</code> は完全透過を意味し、 <code>1</code> は完全不透過を意味します。
<i>opacitystroke</i>	(float。PDF/A モードで用いられるときは値 <code>1</code> を持つ必要があります) 描線操作の不透明度を範囲 <code>0</code> ~ <code>1</code> で表したものの。値 <code>0</code> は完全透過を意味し、 <code>1</code> は完全不透過を意味します。

表 13.7 テキスト行・テキストフローブロックに対するテキスト書式プロパティ






















キーワード	とりうる値・解説		
<i>charspacing</i>	(float またはパーセント値) 文字間隔。パーセント値は <i>fontsize</i> に対する割合。デフォルト : 0		
<i>decoration-above</i>	(論理値) true の場合、underline・strikeout・overline オプションで有効にされたテキスト装飾がテキストの前面に描かれ、そうでなければテキストの背面に描かれます。描画順序を変えると、装飾線の書式に影響を与えます。デフォルト : false		
<i>fillcolor</i>	(色) テキストの塗り色。デフォルト : gray 0 (= 黒)		
<i>fontname</i> ¹	(文字列) フォントの名前。PDF_load_font() が求めるものと同じです。Block Plugin では、システムで利用可能なフォントの一覧が表示されます。ただしこうしたフォント名は、macOS・Windows・Unix システム間で互換とは限りません。fontname の先頭が「@」キャラクターである場合は、そのフォントは縦書きモードで適用されます。ブロックに流し込みを行う際には、PDF_fill_textblock() に、font オプションを与えていない限り、テキストのエンコーディングをオプションとして与える必要があります。		
<i>fontsize</i> ¹	(float) 文字のサイズをポイント単位で指定		
<i>horizscaling</i>	(float またはパーセント値) テキストの横伸縮。デフォルト : 100%		
<i>italicangle</i>	(float) テキストの斜体角度を度単位で指定。デフォルト : 0		
<i>kerning</i>	(論理値) カーニングの動作。デフォルト : false		
<i>overline</i>	(論理値) 上線のモード。デフォルト : false		
<i>shadow</i>	(複合) 影付き効果を生成 (デフォルト : 影なし)。以下のサブプロパティを使えます : <i>fillcolor</i> (色) 影の色。デフォルト : {gray 0.8} <i>offset</i> (float 2 個かパーセント値 2 個のリスト) テキストの参照点からの影のオフセットを、ユーザー座標で、または文字サイズに対するパーセント値で表したものの。デフォルト : {5% -5%}		
<i>strikeout</i>	(論理値) 取り消し線のモード。デフォルト : false		
<i>strokecolor</i>	(色) テキストの描線色。デフォルト : gray 0 (= 黒)		
<i>strokewidth</i>	(float・パーセント値・キーワードのいずれか。textrendering が袋文字に設定されているときのみ意味を持ちます) 袋文字の線幅 (ユーザー座標で、または percentage に対するパーセント値で)。キーワード auto、またはそれと等価な値 0 は、内蔵のデフォルトを用います。デフォルト : auto		
<i>textrendering</i>	(整数) テキスト表現モード。Type 3 フォントでは値 3 のみ意味を持ちます (デフォルト : 0) : <table border="0" style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 50%; vertical-align: top;"> <p>0  テキストを塗る</p> <p>1  テキストを描線 (袋文字)</p> <p>2  テキストを塗って描線</p> <p>3 不可視テキスト</p> </td> <td style="width: 50%; vertical-align: top;"> <p>4  テキストを塗り、クリッピングパスに追加</p> <p>5  テキストを描線し、クリッピングパスに追加</p> <p>6  テキストを塗って描線し、クリッピングパスに追加</p> <p>7  テキストをクリッピングパスに追加 (ブロックでは不可)</p> </td> </tr> </table>	<p>0  テキストを塗る</p> <p>1  テキストを描線 (袋文字)</p> <p>2  テキストを塗って描線</p> <p>3 不可視テキスト</p>	<p>4  テキストを塗り、クリッピングパスに追加</p> <p>5  テキストを描線し、クリッピングパスに追加</p> <p>6  テキストを塗って描線し、クリッピングパスに追加</p> <p>7  テキストをクリッピングパスに追加 (ブロックでは不可)</p>
<p>0  テキストを塗る</p> <p>1  テキストを描線 (袋文字)</p> <p>2  テキストを塗って描線</p> <p>3 不可視テキスト</p>	<p>4  テキストを塗り、クリッピングパスに追加</p> <p>5  テキストを描線し、クリッピングパスに追加</p> <p>6  テキストを塗って描線し、クリッピングパスに追加</p> <p>7  テキストをクリッピングパスに追加 (ブロックでは不可)</p>		
<i>textrise</i>	(float またはパーセント値) テキストの縦方向のオフセット。パーセント値は <i>fontsize</i> に対する割合。デフォルト : 0		
<i>underline</i>	(論理値) 下線のモード。デフォルト : false		

表 13.7 テキスト行・テキストフローブロックに対するテキスト書式プロパティ

キーワード	とりうる値・解説
<i>underline-position</i>	(float・パーセント値・キーワードのいずれか) 下線テキストのベースラインに対する描線の相対位置。パーセント値は <code>font-size</code> に対する割合。デフォルト : <code>auto</code>
<i>underline-width</i>	(float・パーセント値・キーワードのいずれか) 下線テキストの線幅。パーセント値は <code>font-size</code> に対する割合。デフォルト : <code>auto</code>
<i>wordspacing</i>	(float またはパーセント値) 単語間隔。パーセント値は <code>font-size</code> に対する割合。デフォルト : <code>0</code>

1. このプロパティは、テキスト行・テキストフローブロックでは必須です。Block Plugin はこれを自動生成します。

13.7.4 テキスト作成プロパティ

テキスト作成プロパティ群は、テキスト行・テキストフローブロックの前処理工程を指定します。テキスト行・テキストフローブロックに適用されるテキスト作成プロパティの一覧を表 13.8 に示します。

表 13.8 テキスト行・テキストフローブロックに対するテキスト作成プロパティ

キーワード	とりうる値・解説
<i>charref</i>	(論理値) true にすると、数値参照・文字参照・グリフ名参照の置き換えが行われます。デフォルト : グローバルな <i>charref</i> オプション
<i>escape-sequence</i>	(論理値) true にすると、内容文字列・ハイパーテキスト文字列・名前文字列内のエスケープシーケンスの置き換えが行われます。デフォルト : グローバルな <i>escapesequence</i> オプション
<i>features</i>	(キーワードのリスト) <i>script</i> ・ <i>language</i> オプションに従って、OpenType フォントのどのタイプグラフィー機能をテキストに適用するかを指定。フォントにない機能のキーワードは警告なく無視されます。以下のキーワードを使えます : <i>_none</i> フォント内のどの機能も適用しない。例外として、 <i>vert</i> 機能は <i>novert</i> キーワードを用いて明示的に無効化する必要があります。 <名前> 4文字のOpenTypeタグ名を与えるとその機能が有効化されます。よく用いられる機能名は <i>liga</i> ・ <i>ital</i> ・ <i>tnum</i> ・ <i>smcp</i> ・ <i>swsh</i> ・ <i>zero</i> などです。利用できる機能の名前と説明の全一覧を、163ページ「7.4.1 対応しているOpenTypeレイアウト機能」に示しています。 <i>no<名前></i> 機能名の前に接尾辞 <i>no</i> をつけると (<i>noliga</i> など) その機能が無効化されます。デフォルト : 横書きモードでは <i>_none</i> 。縦書きモードでは <i>vert</i> が自動的に適用されます。OpenType 機能対応を利用するには <i>PDF_load_font()</i> の <i>readfeatures</i> オプションが必要です。
<i>language</i>	(キーワード。 <i>script</i> を与えられている場合にのみ意味を持ちます) 指定した言語に従ってテキストを処理。これは <i>features</i> ・ <i>shaping</i> オプションに対して意味を持ちます。すべてのキーワードの一覧を 171 ページ「7.5.2 シェーピング処理を制御」に示しています。例 : <i>ARA</i> (アラビア語)・ <i>JAN</i> (日本語)・ <i>HIN</i> (ヒンディー語)。デフォルト : <i>_none</i> (言語未定義)
<i>script</i>	(キーワード。 <i>shaping=true</i> の場合には必須) 指定した用字系に従ってテキストを処理。これは <i>features</i> ・ <i>shaping</i> ・ <i>advancedlinebreaking</i> オプションに対して意味を持ちます。最もよく使われる用字系のキーワードは以下の通りです : <i>_none</i> (未定義の用字系)・ <i>latn</i> ・ <i>grek</i> ・ <i>cyrl</i> ・ <i>armn</i> ・ <i>hebr</i> ・ <i>arab</i> ・ <i>deva</i> ・ <i>beng</i> ・ <i>guru</i> ・ <i>gujr</i> ・ <i>orya</i> ・ <i>taml</i> ・ <i>thai</i> ・ <i>laoo</i> ・ <i>tibt</i> ・ <i>hang</i> ・ <i>kana</i> ・ <i>han</i> 。キーワード <i>_auto</i> は、テキスト内の大多数のキャラクターが属する用字系を選択しますが、ただし <i>latn</i> ・ <i>_none</i> は無視されます。キーワードの全一覧が 171 ページ「7.5.2 シェーピング処理を制御」にあります。デフォルト : <i>_none</i>
<i>shaping</i>	(論理値) true にすると、テキストが <i>script</i> ・ <i>language</i> オプションに従って整形 (シェーピング) されます。 <i>script</i> オプションの値を <i>_none</i> 以外にする必要があります。かつ、必要なシェーピングテーブルがフォント内で得られる必要があります。デフォルト : <i>false</i>

13.7.5 テキスト組版プロパティ

テキストフローブロックにだけ使用できるテキスト組版プロパティの一覧を表 13.9 に示します。ただし *stamp* プロパティだけはテキスト行ブロックにも使用できます。これらを用いて、テキストフロー処理のための初期オプションリストを構築できます (*PDF_create_textflow()* の *optlist* 引数に対応)。テキストフローで用いるインラインオプションリストは、プラグインでは指定できず、サーバー上で *PDF_fill_textblock()* によるブロックへの流し込みの際に、またはブロックの *defaulttext* プロパティ内で、テキスト内容の一部として与えることができます。

表 13.9 テキスト組版プロパティ (主にテキストフローブロック用)

キーワード	とりうる値・解説
<i>adjust-method</i>	(キーワード) <i>minspacing</i> ・ <i>maxspacing</i> オプションで指定した制限の範囲内で単語間隔を詰めたり拡げたりしてもテキストが行内に収まりきらない時に行の調整に用いる方式 (デフォルト: <i>auto</i>): <i>auto</i> 以下の方式を順に適用: <i>shrink</i> ・ <i>spread</i> ・ <i>nofit</i> ・ <i>split</i> . <i>clip</i> はめ込み枠の右端 (<i>rightindent</i> オプションを考慮) からはみ出した部分を切り落とす点を除いて、 <i>nofit</i> と同じ。 <i>nofit</i> 最後の単語を次行へ送る。ただし、残される (短い) 行が、 <i>nofitlimit</i> オプションで指定されたパーセント値よりも短くならない場合に限りです。均等配置の段落であっても、若干がたつて見えることがあります。 <i>shrink</i> 単語が行内に収まらないとき、 <i>shrinklimit</i> の制限内でテキストを圧縮。それでも収まらなければ <i>nofit</i> 方式を適用。 <i>split</i> 最後の単語を次行へ送らずハイフネーションを強制。テキストフォントならハイフンキャラクターを挿入しますが、記号フォントなら挿入しません。 <i>spread</i> 最後の単語を次行へ送り、残された (短い) 行を均等配置するように単語内の文字間の間隔を <i>spreadlimit</i> の制限内で拡張。それで均等配置できなければ <i>nofit</i> 方式を適用。
<i>advanced-linebreak</i>	(論理値) 複雑用字系で必要とされる高度な改行アルゴリズムを適用します。これはタイ語など、単語間の区切りを空白キャラクターを用いて示さない用字系での改行に必須です。locale・script オプションに従います。デフォルト: <i>false</i>
<i>alignment</i>	(キーワード) 段落内の行の書式を指定。デフォルト: <i>left</i> . <i>left</i> 左揃え (<i>leftindent</i> を始点として)。 <i>center</i> 中央揃え (<i>leftindent</i> から <i>rightindent</i> までの間で)。 <i>right</i> 右揃え (<i>rightindent</i> を終点として)。 <i>justify</i> 両端揃え。
<i>avoid-emptybegin</i>	(論理値) <i>true</i> にすると、はめ込み枠先頭の空行が削除されます。デフォルト: <i>false</i>
<i>fixedleading</i>	(論理値) <i>true</i> にすると、各行内で最初に見つかった行送り値を用います。そうでないなら、行内のすべての行送り値のうちの最大値を用います。デフォルト: <i>false</i>
<i>hortab-method</i>	(キーワード) テキスト内の水平タブの扱い。算出位置がカレントテキスト位置より左のときは、そのタブは無視されます (デフォルト: <i>relative</i>): <i>relative</i> <i>hortabsize</i> で指定された分、位置を進めます。 <i>typewriter</i> <i>hortabsize</i> の次の倍数まで位置を進めます。 <i>ruler</i> <i>ruler</i> オプション内の <i>n</i> 番目のタブ値まで位置を進めます。ここで <i>n</i> は、その行内でそれまでに見つかったタブの数です。n がタブ位置の数を超える分については、 <i>relative</i> 方式を適用します。

表 13.9 テキスト組版プロパティ (主にテキストフローブロック用)

キーワード	とりうる値・解説
hertabsize	(float またはパーセント値) 水平タブの幅 ¹ 。その解釈は <code>hertabmethod</code> オプションに依存します。デフォルト : 7.5%
lastalignment	(キーワード) 段落内の最終行の書式。alignment オプションの全キーワードと以下のキーワードを使えます (デフォルト : auto) : auto alignment オプションの値を使用、ただしそれが justify なら left を使用。
leading	(float またはパーセント値) テキストのベースライン間の間隔を、ユーザー座標で、または文字サイズに対するパーセント値で指定。デフォルト : 100%
locale	(キーワード) <code>advancedlinebreak=true</code> のときに用字系特有の改行方式で用いられるロケール。キーワードは、以下の 1 個ないし複数の構成要素から成り、オプションな構成要素は下線キャラクター「_」で区切られます (その文法は、NLS/POSIX のロケール ID とは若干異なっています) : <ul style="list-style-type: none"> ▶ (必須) ISO 639-2 (www.loc.gov/standards/iso639-2 を参照) に従った、小文字 2 文字または 3 文字の言語コード。例 : en (英語) ・ de (ドイツ語) ・ ja (日本語)。これは language オプションとは異なっています。 ▶ (オプション) ISO 15924 に従った、4 文字の用字系コード。例 : Hira (ひらがな) ・ Hebr (ヘブライ) ・ Arab (アラビア) ・ Thai (タイ)。 ▶ (オプション) ISO 3166 に従った、大文字 2 文字の国コード。例 : DE (ドイツ) ・ CH (スイス) ・ GB (イギリス)。 ロケールを指定することは、高度な改行のために必須ではありません : キーワード <code>_none</code> は、ロケール独自の処理が行われないことを指定します。デフォルト : <code>_none</code> 。 例 : <code>de_DE</code> ・ <code>en_US</code> ・ <code>en_GB</code>
maxspacing minspacing	(float またはパーセント値) 単語間の最大間隔・最小間隔 (ユーザー座標で表すか、空白キャラクターの幅に対するパーセント値で指定)。算出される単語間隔が、与えられた値までに制限されず (ただし、 <code>wordspacing</code> オプションはなお加算されます)。デフォルト : <code>minspacing=50%</code> 、 <code>maxspacing=500%</code>
minlinecount	(整数) はめ込み枠の最終段落の最小行数。これより行が少ないときには次のはめ込み枠内に配置されます。値 2 を用いると、段落の中の 1 行だけがはめ込み枠末尾に配置される (「オーファン」) を避けられます。デフォルト : 1
nofitlimit	(float またはパーセント値) <code>nofit</code> 方式における行の長さの下限 (ユーザー座標で表すか、はめ込み枠の幅に対するパーセント値で指定)。デフォルト : 75%
parindent	(float またはパーセント値) 段落の先頭行の左インデント ¹ 。leftindent にこの値が加算されます。このオプションを行内で指定すると、タブのように動作します。デフォルト : 0
rightindent leftindent	(float またはパーセント値) 全テキスト行の右インデント・左インデント ¹ 。leftindent が行内で指定された場合、決定された位置がカレントテキスト位置より左のときは、このオプションはカレント行については無視されます。デフォルト : 0
ruler²	(float のリスト、またはパーセント値のリスト) <code>hertabmethod=ruler</code> に対する絶対タブ位置のリスト ¹ 。このリストは、非負エントリを昇順で最大 32 個持てます。デフォルト : <code>hertabsize</code> の整数倍
shrinklimit	(パーセント値) <code>shrink</code> 方式におけるテキスト長体の下限。算出される縮小率が、与えられた値までに制限されます。ただし、 <code>horizscaling</code> オプションの値が乗算されます。デフォルト : 85%
spreadlimit	(float またはパーセント値) <code>spread</code> 方式における 2 文字間の間隔の上限 (ユーザー座標で表すか、文字サイズに対するパーセント値で指定)。算出された文字間隔が、 <code>charspacing</code> オプションの値に加算されます。デフォルト : 0

表 13.9 テキスト組版プロパティ（主にテキストフローブロック用）

キーワード	とりうる値・解説
<i>stamp</i>	（キーワード。テキスト行・テキストフローブロック）このオプションを使うと、ブロック長方形内の対角線上にスタンプを作成することができます。スタンプのテキストは可能な限り拡大されて印字されます。スタンプのテキストを枠内に配置する際には、 <code>position・fitmethod・orientate</code> （ <code>north・south</code> のみ）オプションに従います。デフォルト： <code>none</code> 。 <i>llzur</i> 左下隅から右上隅へ向かう対角線上にスタンプが配置されます。 <i>ulzlr</i> 左上隅から右下隅へ向かう対角線上にスタンプが配置されます。 <i>none</i> スタンプは作成されません。
<i>tabalignchar</i>	（整数）タブの小数点揃えの整列位置にしたいキャラクターの Unicode 値。デフォルト：キャラクター「,」（U+002E）
<i>tabalignment</i> ²	（キーワードのリスト）タブ位置の整列方式。このリスト内の各エントリはそれぞれ、 <code>ruler</code> オプション内で、その対応するエントリの整列方式を定義します（デフォルト： <code>left</code> ）： <i>center</i> テキストはタブ位置で中央揃えされます。 <i>decimal</i> 最初に現れる <i>tabalignchar</i> をタブ位置で左揃えされます。 <i>tabalignchar</i> が見つからないときは右揃えが適用されます。 <i>left</i> テキストはタブ位置で左揃えされます。 <i>right</i> テキストはタブ位置で右揃えされます。

1. ユーザー座標で、またははめ込み枠の幅に対するパーセント値で指定します。
2. タブ設定は、ブロックプロパティダイアログの「テキスト組版」グループの「`hortabmethod=ruler` におけるルーラータブ」サブグループで編集することができます。

13.7.6 オブジェクトはめ込みプロパティ

はめ込みプロパティ群は、すべての種類のブロックで利用できますが、ただしいくつかのプロパティは、特定の種類のブロックでのみ利用できます。これらは、ブロック内に内容が配置される方法を制御します：

- ▶ テキスト行・画像・PDF・グラフィックブロックで利用できるはめ込みプロパティの一覧を表 13.10 に示します。
- ▶ テキストフローブロックで利用できるはめ込みプロパティの一覧を表 13.11 に示します（主に縦方向のはめ込みに関するものです）。

オブジェクトはめ込みアルゴリズムは、ブロック長方形をはめ込み枠として用います。*fitmethod=clip* の場合を除き、切り落としは行われません。ブロック内容がブロック長方形からはみ出さないようにしたいときは、*fitmethod=nofit* を避けてください。

表 13.10 テキスト行・画像・PDF・グラフィックブロックに対するはめ込みプロパティ

キーワード	とりうる値・解説
<i>alignchar</i>	(Unichar またはキーワード。テキスト行ブロックにのみ適用) 指定されたキャラクターをテキスト内に見つけたとき、その左下隅を、ブロック長方形の左下隅に整列させます。横書きテキストで orientate=north か south の場合には、position オプションの 1 番目の値で位置を決定します。縦書きテキストで orientate=west か east の場合には、position オプションの 2 番目の値で位置を決定します。指定された位置整列キャラクターがテキスト内にないときは、このオプションは無視されます。値 0 がキーワード none ならば、位置整列キャラクターを無効にします。fitmethod は指定されれば適用しますが、alignchar で強制的に位置付けられるので、はめ込み枠内にテキスト配置することはできません。デフォルト : none
<i>dpi</i>	(float のリスト。画像ブロックにのみ適用) 縦方向・横方向の画像解像度を表す 1 個または 2 個の値。単位は pixels per inch。値 0 の場合、画像内部に解像度が格納されていればそれを用い、なければ 72 dpi とします。fitmethod プロパティが指定されていて、そのキーワードが auto・meet・slice・entire のいずれかならば、本プロパティは無視されます。デフォルト : 0
<i>fitmethod</i>	(キーワード) 与えられた内容がブロック長方形に収まりきらないときの解決方式 : auto・clip・entire・meet・nofit・slice。(デフォルト : meet)。
<i>margin</i>	(float のリスト。テキスト行ブロックにのみ適用) テキスト枠の縦・横方向の長さ差し引きを記述する 1 個または 2 個の値。デフォルト : 0
<i>minfontsize</i>	(float またはパーセント値。テキスト行にのみ適用) fitmethod=auto で shrinklimit を超えたとき、テキストがブロック長方形に収まるよう縮小される際に、許される最小の文字サイズ。下限値を、ユーザー座標で、またはブロックの高さに対するパーセント値で指定します。下限に達したときは、テキストは指定された minfontsize を文字サイズとして作成されます。デフォルト : 0.1%
<i>orientate</i>	(キーワード) 内容を配置する向きを指定します。とりうる値は north・east・south・west。デフォルト : north
<i>position</i>	(float のリスト) 内容の中における参照点の位置を指定する、1 個または 2 個の値。ブロック内でのパーセント値として位置を指定します。テキスト行ブロックのみ : キーワード auto を、リストの 1 番目の値として用いることもできます。これは、テキストの筆記方向が右書きの場合（アラビア語・ヘブライ語など）には right を意味し、そうでない場合（欧文など）には left を意味します。 デフォルト : [0 0]、すなわち左下隅
<i>rotate</i>	(float) 回転角を度単位で表したものの。処理が始まる前にブロックが反時計回りに回転されます。参照点が回転の中心となります。デフォルト : 0

表 13.10 テキスト行・画像・PDF・グラフィックブロックに対するはめ込みプロパティ

キーワード	とりうる値・解説
<i>scale</i>	(float のリスト。画像・PDF・グラフィックブロックのみ) 縦方向・横方向の、求める伸縮倍率を表す 1 個または 2 個の値。fitmethod プロパティが指定されていて、かつそのキーワードが auto・meet・slice・entire のいずれかならば、本プロパティは無視されます。デフォルト: 1
<i>shrinklimit</i>	(float またはパーセント値。テキスト行ブロックにのみ適用) fitmethod=auto でテキストを収める際に適用される縮小倍率の下限。デフォルト: 0.75

表 13.11 テキストフローブロックに対するはめ込みプロパティ

キーワード	とりうる値・解説
<i>firstlinedist</i>	(float・パーセント値・キーワードのいずれか) ブロック長方形上端とテキスト先頭行ベースラインとの間隔を、ユーザー座標で表すか、その文字サイズ (fixedleading=true なら行の先頭の文字サイズ、そうでないなら行内のすべての文字サイズのうちの最大値) に対するパーセント値で表すか、キーワードで表したもの (デフォルト: leading)。 <i>leading</i> 先頭行について決定された行送り値。A のような、読み分け記号付きの文字は普通、はめ込み枠上端に接するでしょう。 <i>ascender</i> 先頭行について決定されたアセンダー値。d や h のような、大きなアセンダーを持つ文字は普通、はめ込み枠上端に接するでしょう。 <i>capheight</i> 先頭行について決定されたキャップハイト値。H のような大文字は普通、はめ込み枠上端に接するでしょう。 <i>xheight</i> 先頭行について決定された x ハイト値。x のような小文字は普通、はめ込み枠上端に接するでしょう。 fixedleading=false なら、先頭行内で見出されたすべての leading・ascender・xheight・capheight 値のうちの最大値が用いられます。
<i>fitmethod</i>	(キーワード) ブロックがテキストフローに対して小さすぎるときの解決方式: <i>auto</i> テキストが収まるまで、fontsize と leading を縮めます。 <i>clip</i> テキストをブロックの端で切り落としします (テキストフローブロックを連結する場合に有用です)。 <i>nofit</i> テキストをブロック下端からはみ出させます (ブロックを移動させる場合に有用です)。 デフォルト: textflowhandle オプションが与えられているなら clip、そうでないなら auto
<i>lastlinedist</i>	(float・パーセント値・キーワードのいずれか。fitmethod=nofit のときは無視されます) テキスト最終行ベースラインとはめ込み枠下端との間隔を、ユーザー座標で表すか、文字サイズ (fixedleading=true なら行の先頭の文字サイズ、そうでないなら行内のすべての文字サイズのうちの最大値) に対するパーセント値で表すか、キーワードで表したもの。デフォルト: 0、すなわちはめ込み枠下端をベースラインとして用い、アセンダーは普通、はめ込み枠の下へはみ出すでしょう。 <i>descender</i> 最終行について決定されたアセンダー値。g や j のような、アセンダーを持つ文字は普通、はめ込み枠下端に接するでしょう。 fixedleading=false なら、最終行内で見出されたすべてのアセンダー値のうちの最大値が用いられます。
<i>linespread-limit</i>	(float またはパーセント値。verticalalign=justify の場合にのみ可) 上下合わせの場合に行送りを増やす際の最大値を、ユーザー座標で表すか、行送りに対するパーセント値で表したもの。デフォルト: 200%
<i>maxlines</i>	(整数またはキーワード) はめ込み枠内の最大行数。あるいはキーワード auto を指定して、できるだけ多くの行をはめ込み枠内に入れさせることもできます。最大行数が入ったとき、PDF_fit_textflow() は文字列 _boxfull を返します。

表 13.11 テキストフローブロックに対するはめ込みプロパティ

キーワード	とりうる値・解説
<i>minfontsize</i>	(float またはパーセント値) 特に <code>fitmethod=auto</code> のとき、はめ込み枠に収まるようテキストが縮小される際に許される最小文字サイズ。下限値を、ユーザー座標で、またははめ込み枠の高さに対するパーセント値で指定します。下限に達してもなおテキストが収まりきらないときは、文字列 <code>_boxfull</code> が返されます。デフォルト : 0.1%
<i>orientate</i>	(キーワード) テキストを配置する向きを指定します。とりうる値は <code>north</code> ・ <code>east</code> ・ <code>south</code> ・ <code>west</code> 。デフォルト : <code>north</code>
<i>rotate</i>	(float) 回転角を度単位で表したもの。はめ込み枠の左下隅を中心として、座標系を回転させます。これによって、枠とテキストが回転されます。テキストが配置された時点で回転はリセットされます。デフォルト : 0
<i>verticalalign</i>	(キーワード) はめ込み枠内のテキストの縦揃え (デフォルト : <code>top</code>) :
<i>top</i>	先頭行から下へ順に組版。テキストがはめ込み枠に満たないときは、テキストの下に余白があきます。
<i>center</i>	はめ込み枠内の縦方向の中央にテキストを配置。テキストがはめ込み枠に満たないときは、テキストの上下に余白があきます。
<i>bottom</i>	最終行から上へ順に組版。テキストがはめ込み枠に満たないときは、テキストの上に余白があきます。
<i>justify</i>	はめ込み枠の上端と下端にテキストを合わせます。それを実現するために、行送りを増やします。ただし、 <code>linespreadlimit</code> で指定された限界までしか増やしません。先頭行の高さは、 <code>firstlinedist=leading</code> の場合のみ増やします。

13.7.7 デフォルト内容のためのプロパティ

デフォルト内容に関するプロパティ群は、内容が特に与えられなかったときのブロックへの流し込みの方法を指定します。これらはとりわけプレビュー機能で有用です。なぜならプレビューではブロックにそのデフォルト内容を流し込むからです。デフォルト内容に関するプロパティの一覧を表 13.12 に示します。

表 13.12 デフォルト内容のためのプロパティ

キーワード	とりうる値・解説
<i>default-graphics</i>	(文字列。グラフィックブロックにのみ適用) グラフィックがクライアントアプリケーションから与えられなかったときに用いられるグラフィックファイルのパス名。 ¹
<i>defaultimage</i>	(文字列。画像ブロックにのみ適用) 置き換え画像がクライアントアプリケーションから与えられなかったときに用いられる画像のパス名。 ¹
<i>defaultpdf</i>	(文字列。PDF ブロックにのみ適用) 置き換え PDF がクライアントアプリケーションから与えられなかったときに用いられる PDF 文書のパス名。 ¹
<i>default-pdfpage</i>	(整数。PDF ブロックにのみ適用) デフォルト PDF 文書内のページのページ番号。デフォルト : 1
<i>defaulttext</i>	(文字列。テキスト行・テキストフローブロックにのみ適用) 可変テキストがクライアントアプリケーションから与えられなかったときに用いられるテキスト ²

1. ファイル名には絶対パスを付けず、SearchPath 機能を利用するよう PPS クライアントアプリケーションを作っておくほうがよいでしょう。そうすればブロック処理を、プラットフォームやファイルシステムの細かい違いから切り離すことができます。
2. テキストは winansi エンコーディングか Unicode で解釈されます。

13.7.8 カスタムプロパティ

カスタムプロパティは、すべての種類のブロックに適用されます。PPS とプレビュー機能からは無視されます。カスタムプロパティの命名規則を表 13.13 に示します。

表 13.13 すべてのブロック種別に対するカスタムブロックプロパティ

キーワード	とりうる値・解説
3 種類のキャラクター [] / を含まないあらゆる 名前	(文字列・名前・float のいずれか、または float のリスト) 各カスタムプロパティの値をどう解釈するかは、全くクライアントアプリケーションの領分です。PPS からは無視されます。

13.8 pCOS でブロック名とプロパティをクエリー

PPSによる自動ブロック処理に加えて、内蔵のpCOS機能を使うと、ブロック名を評価したり、標準・カスタムプロパティをクエリーしたりすることができます。

クックブック 取り込んだPDFの中に含まれているブロックのプロパティをクエリーする完全なコードサンプルがクックブックの `blocks/query_block_properties` トピックにあります。

ブロックの数と名前を知る クライアントコード側では、取り込んだページ上のブロックの名前も数も知らなくてかまいません。なぜならクエリーすることもできるからです。以下のステートメントは、ページ番号 *pagenum* のページ上のブロックの数を返します：

```
blockcount = (int) p.pcos_get_number(doc, "length:pages[" + pagenum + "]/blocks");
```

以下のステートメントは、ページ *pagenum* 上の *blocknum* 番目のブロックの名前を返します（ブロックとページの番号は0から始まります）：

```
blockname = p.pcos_get_string(doc,
    "pages[" + pagenum + "]/blocks[" + blocknum + "]/Name");
```

返されたブロック名はその後、ブロックのプロパティをクエリーしたり、ブロックヘテキスト・画像・PDF・グラフィック内容を流し込んだりするために利用できます。指定されたブロックが存在しないときには例外が発生します。これを避けるには、*length* 接頭辞を用いれば、ブロックの数を知り、ひいては *blocks* 配列の最大添字を知ることができます（配列の添字は0から始まるので、ブロックの数は最大可能添字より1大きいことに留意してください）。

ブロックが存在するかどうかをチェック クライアントアプリケーションコードにさらに柔軟性を加えるために、ブロックに流し込みを行う前に、そのブロックが存在するかどうかをチェックすることもできます。こうしておけば、デザイナーが別のページへブロックを移動させても、そのブロックへ流し込みを行うアプリケーションを破壊せずにすみません。

以下のコードは、*foo* という名前のブロックがページ上に存在するかどうかをチェックします：

```
/* pCOSオブジェクト種別「辞書」はそのブロックが存在することを意味します */
if (pcos_get_string(doc, "type:pages[" + pagenum + "]/blocks/" + "foo").equals("dict"))
{
    /* ブロック「foo」はそのページ上に存在 */
}
```

ブロックを番号か名前で特定 ブロックプロパティを特定するパス文法において、以下の表現は互いに等価です。ここでは、番号6のブロックが、その *Name* プロパティを *foo* に設定されているとします：

```
pages[...]/blocks[6]
pages[...]/blocks/foo
```

ブロックの座標をクエリー 名前 *foo* のブロックの左下隅と右上隅を記述する2個の座標ペア (*llx*, *lly*) および (*urx*, *ury*) は、以下のようにクエリーできます：

```
llx = p.pcos_get_number(doc, "pages[" + pagenum + "]/blocks/foo/rect[0]");
lly = p.pcos_get_number(doc, "pages[" + pagenum + "]/blocks/foo/rect[1]");
```

```
urx = p.pcos_get_number(doc, "pages[" + pagenum + "]/blocks/foo/rect[2]");
ury = p.pcos_get_number(doc, "pages[" + pagenum + "]/blocks/foo/rect[3]");
```

なお、上記の座標はデフォルト座標系で与えられます（左下隅が原点、ただし、そのページの CropBox によって変更されている可能性もあります）。一方 Block Plugin は、ページ左上隅に原点を持つ Acrobat のユーザーインターフェイス座標系に従って座標を表示しません。

pCOS 擬似オブジェクト *rect*（すべて小文字の）を用いてクエリーされる値は、関連するいかなる CropBox/MediaBox・Rotate エントリーをも考慮に入れ、かつ座標の順序を正規化します。これに対し、ネイティブ PDF キー *Rect* を用いてクエリーされる値は、CropBox が存在する場合には新しい座標としてそのまま受け渡すことはできません。

topdown オプションはブロック座標をクエリーする際には考慮されないことに留意してください。

カスタムプロパティをクエリー カスタムプロパティは、以下の例のようにクエリーすることができます。ここでは、ページ *pagenum* 上の *b1* というブロックからプロパティ *zipcode* をクエリーしています：

```
zip = p.pcos_get_string(doc, "pages[" + pagenum + "]/blocks/b1/Custom/zipcode");
```

ブロック内に具体的に何というカスタムプロパティがあるかわからない場合には、実行時にその名前を取得することも可能です。*b1* というブロックの最初のカスタムプロパティの名前を得るには、以下のようにします：

```
propname = p.pcos_get_string(doc, "pages[" + pagenum + "]/blocks/b1/Custom[0].key");
```

番号を 0 でなく 1 つずつ増やしていけば、すべてのカスタムプロパティの名前を得ることができます。*length* 接頭辞を用いれば、カスタムプロパティの数を知ることができます。

存在しないブロックプロパティとデフォルト値 ブロックまたはプロパティが実在するかどうかを知るには、*type* 接頭辞を用います。パスに対する型が 0 か *null* ならば、そのオブジェクトは PDF 文書内に存在していません。なお、定義済みプロパティの場合、これはプロパティのデフォルト値が用いられることを意味します。

カスタムプロパティの名前空間 さまざまなソースからの PDF 文書をやり取りする際に混乱が生じることを避けるため、カスタムプロパティ名をつけるときには必ず、インターネットドメイン名を企業固有の接頭辞として使い、その後にコロン「:」とプロパティ名本体を続けることを推奨します。たとえば、ACME 社であれば以下のようなプロパティ名を使用するのです：

```
acme.com:digits
acme.com:refnumber
```

標準プロパティとカスタムプロパティはブロック内で異なる格納のされ方をしているもので、標準 PPS プロパティ名（402 ページ「13.7 ブロックのプロパティ」で定義されているもの）がカスタムプロパティ名と衝突することは決してありません。

13.9 ブロックをプログラマ的に作成・取り込む

13.9.1 POCA で PDFlib ブロックを作成

PDFlib ブロックは、PPS に内蔵されている POCA インターフェイスでプログラマ的に作成することも可能です。POCA を用いると、ブロックのために必要な PDF データ構造を生成して、*PDF_begin/end_page_ext()* の *blocks* オプションに与えることができます。ブロック定義を作成する際には、419 ページ「13.10 PDFlib ブロックの仕様」の要請に従う必要があります。ブロックプロパティは、402 ページ「13.7 ブロックのプロパティ」に挙げたデータ型に従って作成する必要があります。

クックブック PDFlib ブロックを PPS で作成するのコードサンプルが PDFlib クックブックの *blocks* カテゴリにあります。

PDFlib ブロックの仕様には、1つのブロックの名前が2回記録されているという残念な冗長性があります。すなわちページのメイン *Blocks* 辞書内に1回と、特定のブロック辞書内の *Name* エントリー内にもう1回です。この2個の名前は、そのブロックに PPS で流し込みを行う際や、そのブロックを Block Plugin でプレビューする際に問題が起こることを避けるために、同一でなければなりません。*PDF_begin/end_page_ext()* はそのため、*blocks* オプションで与えられた辞書がこの「同一ブロック名」規則に違反するブロック定義を含んでいる場合には、例外を発生させます。以下のコードサンプルでは、その対応するペアを青色で示しています。

以下のコード断片では、419 ページ「ブロック辞書のキー」で示すブロック定義を POCA 関数群を用いて作成する様子を演示しています：

```
/* ブロック辞書を作成 */
blockdict = p.poca_new("containertype=dict usage=blocks");

/* -----
 * テキストブロックを作成
 * -----
 */
textblock = p.poca_new("containertype=dict usage=blocks type=name key=Type value=Block");

container1 = p.poca_new("containertype=array usage=blocks " +
    "type=integer values={70 640 300 700}");

p.poca_insert(textblock, "type=array key=Rect value=" + container1);
p.poca_insert(textblock, "type=name key=Name value=job_title");
p.poca_insert(textblock, "type=name key=Subtype value=Text");
p.poca_insert(textblock, "type=name key=fitmethod value=auto");
p.poca_insert(textblock, "type=name key=fontname value=Helvetica");
p.poca_insert(textblock, "type=float key=fontsize value=12");

/* このブロックをページのブロック辞書内に挿入 */
p.poca_insert(blockdict, "type=dict key=job_title direct=false value=" + textblock);

/* -----
 * 画像ブロックを作成
 * -----
 */
imageblock = p.poca_new("containertype=dict usage=blocks " +
    "type=name key=Type value=Block");
```

```

container2 = p.poca_new("containertype=array usage=blocks " +
    "type=integer values={70 440 300 600}");

p.poca_insert(imageblock, "type=array key=Rect value=" + container2);
p.poca_insert(imageblock, "type=name key=Name value=logo");
p.poca_insert(imageblock, "type=name key=Subtype value=Image");
p.poca_insert(imageblock, "type=name key=fitmethod value=auto");

/* このブロックをページのブロック辞書内に挿入 */
p.poca_insert(blockdict, "type=dict key=logo direct=false value=" + imageblock);

/* -----
 * PDFブロックを作成
 * -----
 */
pdfblock = p.poca_new("containertype=dict usage=blocks " +
    "type=name key=Type value=Block");

container3 = p.poca_new("containertype=array usage=blocks " +
    "type=integer values={70 240 300 400}");

p.poca_insert(pdfblock, "type=array key=Rect value=" + container3);
p.poca_insert(pdfblock, "type=name key=Name value=pdflogo");
p.poca_insert(pdfblock, "type=name key=Subtype value=PDF");
p.poca_insert(pdfblock, "type=name key=fitmethod value=meet");

/* このブロックをページのブロック辞書内に挿入 */
p.poca_insert(blockdict, "type=dict key=pdflogo direct=false " + "value=" + pdfblock);

/* -----
 * このブロック辞書をカレントページ内に挿入
 * -----
 */
p.end_page_ext("blocks=" + blockdict);

/* クリーンナップ */
p.poca_delete(blockdict, "recursive");

```

13.9.2 PDFlib ブロックを取り込む

入力文書から1個ないし複数のPDFlibブロックを、*PDF_process_pdi()*と *action=copyallblocks* か *action=copyblock* で以下のようにカレント出力ページへコピーすることも可能です：

```

if (p.process_pdi(p, doc, 0, "action=copyallblocks block={pagenumber=1}") != 1)
{
    /* エラー */
}

```

このようにすると、マルチレベルのブロック流し込みワークフローを実装することができます。ただし、各ページ上でブロック名は一意でなければならないことに留意してください。すなわち、同じ名前を持つ複数のブロックを同じページ上へ取り込むことはできません。ブロックをコピー時に名称変更するには *outputblockname* サブオプションを用います。

13.10 PDFlib ブロックの仕様

ブロックの文法は、PDF ページを構成するデータ構造にアプリケーションが独自データを追加格納できるようにする拡張のしくみを定めた PDF Reference に準拠しています。ここでは PDFlib ブロックの文法を説明します。Block Plugin か PDFlib でブロックを作成するユーザーにはこの情報は必要ではありません。

PDFlib ブロックの PDF オブジェクト構造 ページ辞書は *PieceInfo* エントリーを含んでおり、このエントリーは値として別の辞書を持っています。ページ辞書は、ブロック構造の作成または最終更新のタイムスタンプを内容とするキー *LastModified* をも含む必要があります。この辞書はキー *PDFlib* を含んでおり、このキーはアプリケーションデータ辞書を値として持っています。このアプリケーションデータ辞書は、表 13.14 に挙げる 2 個の標準キーを含んでいます。

表 13.14 PDFlib アプリケーションデータ辞書内のエントリー

キー	値
<i>LastModified</i>	(日付文字列、必須) ページ上のブロックが作成された、または最近更新された日時。このエントリーは、POCA インターフェイスでブロックを作成した場合には PDFlib が作成します。
<i>Private</i>	(辞書、必須) ブロックリスト (表 13.15 参照)

ブロックリストとは、ブロック処理に関する一般的な情報に加えて、ページ上のすべてのブロックの一覧をも含んだ辞書です。ブロックリスト辞書の中のキーの一覧を表 13.15 に示します。

表 13.15 ブロックリスト辞書内のエントリー

キー	値
<i>Blocks</i>	(辞書、必須) キーはそれぞれ、ブロック名の名前オブジェクト。その対応する値は、そのブロックに対するブロック辞書です (表 13.17 参照)。ブロック辞書内の Name キーは、この辞書内のブロック名と同じでなければなりません。
<i>BlockProducer</i> ¹	(文字列) ブロックをプログラマ的に作成するのに用いられたソフトウェアの名前。このエントリーは、POCA インターフェイスでブロックを作成した際に PDFlib によって作成されます。
<i>PluginVersion</i> ¹	(文字列) ブロックの作成に使われた Block Plugin のバージョン識別を表す文字列。
<i>pdfmark</i> ¹	(論理値) ブロックリストが pdfmark を用いて生成されている場合は true でなければなりません。
<i>Version</i>	(数、必須) ファイルが準拠するブロック仕様のバージョン番号。本文書では、バージョン 10 のブロック仕様を解説しています。

1. キー *BlockProducer*・*PluginVersion*・*pdfmark* のうちのいずれか 1 つが、かつ 1 つのみが存在する必要があります。

ブロックプロパティのデータ型 プロパティはオプションリストと同じデータ型に対応しています。ただしハンドルと、アクションリストのような特化されたリストには対応していません。これらの型が PDF データ型へどのようにマップされているかを表 13.16 に示します。

ブロック辞書のキー ブロック辞書は、表 13.17 に挙げるキーを含むことができます。

表 13.16 ブロックプロパティに対するデータ型

データ型	PDF での型および注釈
論理値	(論理値)
文字列	(文字列)
キーワード (名前)	(名前) そのプロパティが対応するキーワードのリストにないキーワードを与えるとエラー。
<i>float</i> ・整数	(数値) オプションリストでは点もカンマも小数点として対応しているのに対し、PDF の数値では点を必要とします。
パーセント 値	(要素 2 個の配列) 配列の 1 番目の要素は数、2 番目の要素はパーセントキャラクターを持った文字列。
リスト	(配列)
色	<p>(要素 2 個か 3 個の配列) 配列の 1 番目の要素は色空間を指定し、2 番目の要素はカラー値を指定。色なしを指定するには、各プロパティを省略する必要があります。配列の 1 番目の要素に対しては以下のエンタリーを指定できます：</p> <p><i>/DeviceGray</i> 2 番目の要素はグレー値 1 個。</p> <p><i>/DeviceRGB</i> 2 番目の要素は RGB 値 3 個の配列。</p> <p><i>/DeviceCMYK</i> 2 番目の要素は CMYK 値 4 個の配列。</p> <p><i>[/Separation/ スポットカラー名]</i> 1 番目の要素は、キーワード Separation とスポットカラー名 1 個を持った配列。2 番目の要素は濃度値。 オプションとして 3 番目の要素で、スポットカラーの代替色を指定します。代替色はそれ自身が 1 個の色配列であり、DeviceGray・DeviceRGB・DeviceCMYK・Lab のいずれかの色空間で表されます。代替色を指定しないときは、スポットカラー名は、PPS が内部的に知っている色か、またはアプリケーションで動作時に定義しておいた色でなければなりません。</p> <p><i>[/Lab]</i> 1 番目の要素はキーワード Lab を持った配列。2 番目の要素は Lab 値 3 個の配列。</p>
<i>unichar</i>	(テキスト文字列) UTF-16 BOM U+FEFF で始まる utf16be 形式の Unicode 文字列

表 13.17 ブロック辞書内のエンタリー

プロパティグループ	値
管理プロパティ群	(いくつかのキーは必須) 表 13.4 に従った管理プロパティ群
長方形プロパティ群	(いくつかのキーは必須) 表 13.5 に従った長方形プロパティ群
書式プロパティ群	(いくつかのキーは必須) 表 13.6 に従った、すべての種類のブロックに適用される書式プロパティ群と、表 13.7 に従った、テキスト行・テキストフローブロックに適用されるテキスト書式プロパティ群
テキスト作成プロパティ群	(オプション) 表 13.8 に従った、テキスト行・テキストフローブロックに適用されるテキスト作成プロパティ群
テキスト組版プロパティ群	(オプション) 表 13.9 に従った、テキスト行・テキストフローブロックに適用されるテキスト組版プロパティ群
オブジェクトはめ込みプロパティ群	(オプション) 表 13.10 に従った、テキスト行・画像・PDF・グラフィックブロックに適用されるオブジェクトはめ込みプロパティ群と、表 13.11 に従った、テキストフローブロックに適用されるはめ込みプロパティ群

表 13.17 ブロック辞書内のエントリー

プロパティグループ	値
デフォルト内容に関するプロパティ群	(オプション) 表 13.12 に従った、デフォルト内容に関するプロパティ群
カスタム	(辞書、オプション) 表 13.13 に従った、カスタムプロパティに対するキー / 値ペアを含んだ辞書

A 改訂履歴

日付	更新点
2022年8月28日	▶ PDFlib 10.0.0に関する更新
2021年6月06日	▶ PDFlib 9.3.1に関する更新
2020年8月16日	▶ PDFlib 9.3.0に関する更新
2019年2月01日	▶ PDFlib 9.2.0に関する更新
2018年2月01日	▶ PDFlib 9.1.2に関する更新
2017年7月24日	▶ PDFlib 9.1.1に関する更新
2017年5月09日	▶ PDFlib 9.1.0に関する更新
2016年8月14日	▶ PDFlib 9.0.7に関する更新
2016年1月13日	▶ PDFlib 9.0.6に関する更新
2015年5月24日	▶ PDFlib 9.0.5に関する更新
2014年12月16日	▶ PDFlib 9.0.4に関する更新
2014年5月14日	▶ PDFlib 9.0.3に関する更新
2013年12月17日	▶ PDFlib 9.0.2に関する更新
2013年6月06日	▶ PDFlib 9.0.1に関する更新
2013年3月12日	▶ PDFlib 9.0.0に関する更新
2011年6月09日	▶ PDFlib 8 VT Edition (内部的には8.1.0)に関する更新
2010年12月09日	▶ PDFlib 8.0.2に関するさまざまな更新・修正
2010年9月22日	▶ PDFlib 8.0.1p7に関するさまざまな更新・修正
2010年4月13日	▶ PDFlib 8.0.1に関するさまざまな更新・修正
2009年12月07日	▶ PDFlib 8.0.0に関する更新
2009年3月13日	▶ PDFlib 7.0.4に関するさまざまな更新・修正
2008年2月13日	▶ PDFlib 7.0.3に関するさまざまな更新・修正
2007年8月08日	▶ PDFlib 7.0.2に関するさまざまな更新・修正
2007年2月19日	▶ PDFlib 7.0.1に関するさまざまな更新・修正
2006年10月03日	▶ PDFlib 7.0.0に関する更新と再構成
2006年2月21日	▶ PDFlib 6.0.3に関するさまざまな更新・修正。Rubyの節を追加
2005年8月09日	▶ PDFlib 6.0.2に関するさまざまな更新・修正
2004年11月17日	▶ PDFlib 6.0.1に関する小規模な更新・修正 ▶ 8章に言語別関数プロトタイプ用新書式導入 ▶ 3章にハイパーテキストの例を追加

日付	更新点
2004年6月18日	▶ PDFlib 6 に関する大規模な変更
2004年1月21日	▶ PDFlib 5.0.3 に関する小規模な追加・修正
2003年9月15日	▶ PDFlib 5.0.2 に関する小規模な追加・修正。ブロックの仕様を追加
2003年5月26日	▶ PDFlib 5.0.1 に関する小規模な更新・修正
2003年3月26日	▶ PDFlib 5.0.0 に関する全面的変更と書き直し
2002年6月14日	▶ PDFlib 4.0.3 に関する小規模な変更と .NET バインディングに関する追加
2002年1月26日	▶ PDFlib 4.0.2 に関する小規模な変更と IBM eServer エディションに関する追加
2001年5月17日	▶ PDFlib 4.0.1 に関する小規模な変更
2001年4月1日	▶ PDFlib 4.0.0 の PDI・他機能を解説
2001年2月5日	▶ PDFlib 3.5.0 のテンプレート・CMYK 機能を解説
2000年12月22日	▶ ColdFusion 解説と PDFlib 3.03 に関する追加。COM エディションを別マニュアルに
2000年8月8日	▶ Delphi 解説と PDFlib 3.02 に関する小規模な追加
2000年7月1日	▶ PDFlib 3.01 に関する追加・説明の明瞭化
2000年2月20日	▶ PDFlib 3.0 に関する変更
1999年8月2日	▶ PDFlib 2.01 に関する小規模な変更・追加
1999年6月29日	▶ 言語バインディングごとに節を分離 ▶ PDFlib 2.0 に関する追加
1999年2月1日	▶ PDFlib 1.0 に関する小規模な追加（公開せず）
1998年8月10日	▶ PDFlib 0.7 に関する追加（一つのお客様用のみ）
1998年7月8日	▶ PDFlib 0.6 の PDFlib スクリプティングサポートを初めて記述
1998年2月25日	▶ PDFlib 0.5 に関して説明を若干追加
1997年9月22日	▶ PDFlib 0.4 と本マニュアルを初めて公開

索引

記号

.NET バインディング 39

A

Acrobat のブロック作成用プラグイン 375

ArtBox 71

ascender 157

asciifile オプション 67

autosubsetting オプション 149

B

BleedBox 71

BMP 109, 191

bytes → *hypertextformat*

C

C++ バインディング 35

capheight 157

CCSID 117

CIE L*a*b* カラースペース 90

CMap 118, 119

COLR カラーフォント 128

CropBox 71

currentx・currenty オプション 157

C バインディング 31

D

defaultgray/rgb/cmyk 色空間 88

descender 157

DeviceN カラー 94

dpi 計算 187

E

EBCDIC 66

ebcdicutf8 → *hypertextformat*

errorpolicy オプション 213

EUDC (end-user defined characters) 176

EUDC (エンドユーザー定義キャラクタ) フォント 127

EXIF JPEG 画像 189

G

get_buffer() 64

grid.pdf 69

H

HKS カラー 92

hypertextformat オプション 114

I

IBM System i 66

IBM Z 66

iccprofilegray/rgb/cmyk オプション 87

ISO 14289 (PDF/UA) 335

ISO 15930 (PDF/X) 358

ISO 19005 (PDF/A) 346

ISO 32000-1 (PDF 1.7) 341

ISO 32000-2 (PDF 2.0) 343

J

Javadoc 38

Java バインディング 37

JBIG2 190

JFIF 189

JPEG 189

EXIF 形式の画像 189

JPEG2000 189

L

leading 157

M

masked 195

MediaBox 71

N

NChannel 色空間 95

.NET バインディング 39

n 色 ICC プロファイル 358

O

Objective-C バインディング 42

OBJR 構造エレメント

インタラクティブ要素のための 321, 323

OpenType

カラーフォント 128

コレクション 127

フォント 127

フォントバリエーション 127

レイアウト機能 163
OTC (OpenType Collection) 175
overline オプション 159

P

page 188
Pantone カラー 91
PDF 1.7 341
PDF 1.7 と PDF 2.0 のタグセット 297
PDF 2.0 343
PDF_EXIT_TRY() 33
PDF_get_buffer() 64
PDF/A 346
PDF/UA 335
PDF/X 358
PDFlib
 機能一覧 24
PDFlib Personalization Server 375
pdflib.upr 63
PDFLIBRESOURCEFILE 環境変数 63
PDFlib の機能 24, 28
PDF 取り込みライブラリ (PDI) 211
PDI (PDF 取り込み) 211
pdusebox オプション 215
Perl バインディング 44
permissions 77
Photoshop CMYK 画像 192
PHP バインディング 46
PNG 189
POCA (PDF オブジェクト作成 API)
 ブロックを作成するための 417
 文書部分メタデータ (DPM) のための 371
PPS (PDFlib Personalization Server) 375
Python バインディング 48

R

RAW 画像データ 191
resourcefile オプション 63
RGB カラー 83
RPG バインディング 49
Ruby バインディング 51

S

sbix カラーフォント 128
SearchPath オプション 61
SING フォント 131, 176
sRGB カラー空間 86
strikeout オプション 159
SVG 199
 色 205
SVG カラーフォント 128

T

textformat オプション 114
textrendering オプション 159
textx・texty オプション 157
TIFF 190
topdown オプション 70
TrimBox 71
TrueType フォント 127
TTC (TrueType コレクション) 127
TTC (TrueType Collection) 175
Type 3 (ユーザー定義) フォント 131

U

underline オプション 159
UPR (Unix PostScript Resource) 59
usehypertextencoding オプション 114
usercoordinates オプション 68
utf8/utf16/utf16be/utf16le → hypertextformat
UTF 形式 110

W

WOFF・WOFF2 フォント 130

X

xCLR ICC プロファイル 358
XMP メタデータ 79
XObject 74
x ハイト 157

Z

ZUGFeRD 標準
 電子インボイスのための 347

あ

アセンダー 157
アルファチャンネル
 別画像からの 195
暗号化 76

い

一意識別
 PDF/VT のための XObject の 368
色空間 83
色を変更
 オブジェクトの 104
インコア生成 64
印刷ストリーム順序 314
インチ 68
インライン画像 188

う

ウイドー行 250
上付き 157

え

エスケープシーケンス 121
絵文字修飾シーケンス 161
エラー処理 55
エンコーディング
 カスタム 118
 システムからの取得 117

お

オーバープリント制御 107
オーファン行 250

か

カーニング 157
回転 69
カスタムエンコーディング 118
カスタム構造種別 308
画像
 拡張 187
 データを再利用 187
 ファイル形式 189
画像を拡張 187
カテゴリ
 リソース 60
カプセル化 XObject
 PDF/VT のための 373
カプセル化ヒント
 PDF/VT のための 368
カラーフォント 128
カラーブレンド 98
カレント点 72
環境変数 PDFLIBRESOURCEFILE 63
韓国語 119, 175

き

記号フォント
 グリフを選択 138
輝度マスク 105
機能
 PDFlib の 24, 28
機能一覧
 PDFlib 24
基本多言語面 109
キャップハイト 157
キャラクターとグリフ 109
キャラクターメトリック 157
行送り 157

切り抜き 71

く

グラデーション 98
グラフィック 199
グリフ 109
 記号フォントから選択 138
 グリフ ID (GID) 指定 133
 置換 183
 入手可能性 152
グリフレット 131
クロマキーマスク処理 195

け

権限 76
権限/パスワード 76
言語バインディング→バインディング

こ

コアフォント 143
高度な改行 254
黒点補償 89
コピー
 ページ枠を 226

さ

座標系 68
 下向き 69
 メートル 68
サブセット化 149
サブパス 71

し

シェーディング 98
しおり
 構造を持った 323
システムエンコーディング対応 117
下付き 157
下向き座標 69
出カインテント
 PDF/A のための 349
 PDF/X に対する 361, 362
商用ライセンス 13
私用領域 (PUA) 109, 137
所有者パスワード 76

す

透かし (編集可能な) 236
スケーラブルベクトルグラフィック (SVG)
 199
スコープヒント

PDF/VT のための 372
スタイル名
Windows の 145
ステンシルマスク 196
スポットカラー (分版カラスペース) 91

そ

ソフトマスク 105

た

タイリングパターン 100
ダウンサンプリング 187
タグセット 297, 298
タグセット名前空間 309
タグ付きしおり 323
脱色
オブジェクトを 104
縦書き 175
単位 68

ち

着色
オブジェクトを 104
画像に 197
中国語 119, 175

つ

強く構造化された文書 338

て

ディセッター 157
テキスト位置 157
テキスト進行方向 175
テキストスタイル 157
テキストフロー 238
テキストメトリック 157
デフォルト色空間 87
デフォルト座標 68
添付パスワード 76
テンプレート 74

と

透過
PDF/VT 内の 373
取り込まれた PDF ページ内で検出 373
取り込み PDF ページ内で検出 373

な

アルファチャンネル
内蔵 193

内容文字列 114
名前文字列 114

に

日中韓 (日本語・中国語・韓国語)
Windows コードページ 120
カスタムフォント 175
構成 119
日本語 119, 175

ぬ

塗り 71

ね

ネスト
例外の 32

は

バイトサービング 66
バイト順序マーク (BOM) 110, 115
ハイパーテキスト文字列 114
バインディング 31
パス 71
パスオブジェクト 72
パスワード 76
パターン (タイリング) 100
バックスラッシュ置換 121
パリアブルフォント 127
反転
オブジェクトの色を 104

ひ

評価版 11
標準 14 フォント 143
標準化異体字シーケンス 179
標準構造種別 297
描線 71

ふ

ファイル検索 61
フォーム XObject 74
フォームフィールド : PDFlib ブロックへの変換 389
フォント
OpenType 127
SING 131
TrueType 127
TrueType コレクション 127
Type 3 (ユーザー定義) 131
Type 3 (ユーザー定義) フォント 131
Windows のスタイル名 145

WOFF・WOFF2 130
記号グリフを選択 138
サブセット化 149
標準欧文 143
メトリック 157
ユーザー定義 (Type 3) 131
リソース構成 59
フォント名エイリアス設定 141
不可視テキスト 405
複数画像オブション 197
復号配列
 画像の色に対する 197
複数ページ画像ファイル 188
袋文字 405
プラグイン
 ブロック作成用 375
ブレンドモード 101, 104
ブロック
 POCA で作成 417
 プラグイン 375
ブロックプロパティ 378
文書部分ヒエラルキー 367, 370
文書部分メタデータ (DPM) 367, 371

へ

ページ寸法規格 70
ページ寸法限界
 Acrobat の 71
ページ定義 68
ベクトルグラフィック 199
編集可能なすかし 236

ほ

ホストエンコーディング 117
ホストフォント 144

ま

マスターパスワード 76

み

ミリメートル 68

め

メートル座標 68
メトリック 157
メモリ内に PDF 文書を生成 64

も

文字参照 122

ゆ

ユーザースペース 68
ユーザー定義 (Type 3) フォント 131
ユーザーパスワード 76

よ

用字系固有の改行 254
横書き 175
読み上げ順序 314
弱く構造化された文書 338

り

リソースカテゴリ 60

れ

例外 55
レイヤーと PDI 215
レコードレベル
 PDF/VT のための 367
連携ファイル 80

ろ

ロールマップ
 カスタム構造種別のための 308
ログ記録 57
論理読み取り順序 314

PDFlib GmbH

Franziska-Bilek-Weg 9
80339 München, Germany
www.pdflib.com
電話 +49・89・452 33 84-0

ライセンス発行のお問い合わせ
sales@pdflib.com

サポート
support@pdflib.com (お持ちのライセンス番号をお書きください)

