

# PLOP・PLOP DS

Version 4.1

PDF の線形化・最適化・  
保護・デジタル署名



Copyright © 1997-2011 PDFlib GmbH. All rights reserved.

PDFlib GmbH  
Franziska-Bilek-Weg 9, 80339 München, Germany  
www.pdflib.com  
電話 +49・89・452 33 84-0  
FAX +49・89・452 33 84-99

疑問がおありの際は、PDFlib メーリングリストと、[tech.groups.yahoo.com/group/pdflib](http://tech.groups.yahoo.com/group/pdflib) にあるアーカイブを  
チェックしてください。

ライセンスご希望の際の連絡先 : [jp.sales@pdflib.com](mailto:jp.sales@pdflib.com)  
商用 PDFlib ライセンス保持者向けサポート : [jp.support@pdflib.com](mailto: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。

Adobe・Acrobat・PostScript・XMP は Adobe Systems Inc. の商標です。AIX・IBM・OS/390・WebSphere・iSeries・zSeries は International Business Machines Corporation の商標です。ActiveX・Microsoft・OpenType・Windows は Microsoft Corporation の商標です。Apple・Macintosh・TrueType は Apple Computer, Inc. の商標です。Unicode・Unicode ロゴは Unicode, Inc. の商標です。Unix は The Open Group の商標です。Java・Solaris は Sun Microsystems, Inc. の商標です。HKS は the HKS brand association: Hostmann-Steinberg, K+E Printing Inks, Schmincke の登録商標です。他の企業の製品とサービス名は他の商標やサービスマークである場合があります。

PDFlib PLOP 及び PLOP DS は以下のサードパーティソフトウェアの改変された部分を含んでいます：  
Zlib 圧縮ライブラリ、Copyright © 1995-2002 Jean-loup Gailly and Mark Adler  
Eric Young の書いた Cryptographic ソフトウェア、Copyright © 1995-1998 Eric Young ([ey@cryptsoft.com](mailto:ey@cryptsoft.com))  
Cryptographic ソフトウェア、Copyright © 1998-2002 The OpenSSL Project ([www.openssl.org](http://www.openssl.org))  
Expat XML パーサ、Copyright © 1998, 1999, 2000 Thai Open Source Software Center Ltd  
ICU International Components for Unicode、Copyright © 1995-2009 International Business Machines Corporation and others

PDFlib PLOP 及び PLOP DS は RSA Security, Inc. の MD5 メッセージダイジェストアルゴリズムを含んでいます。



# 目次

## ○ 初めての PLOP・PLOP DS 5

- 0.1 ソフトウェアをインストール 5
- 0.2 PLOP/PLOP DS ライセンスキーを適用 6

## 1 PLOP と PLOP DS の諸機能 11

- 1.1 概観 11
- 1.2 説明書と各サンプルへのロードマップ 13
- 1.3 暗号化・復号・権限 14
- 1.4 Web 最適化（線形化）PDF 15
- 1.5 最適化（軽量化）16
- 1.6 破損 PDF のための修復モード 17
- 1.7 pCOS で文書情報を取得 18
- 1.8 文書情報項目を挿入・取得 19
- 1.9 XMP メタデータを挿入・抽出 20
- 1.10 PLOP DS で電子署名 22
- 1.11 PLOP の処理の詳細 24

## 2 PLOP・PLOP DS コマンドラインツール 27

- 2.1 PLOP・PLOP DS コマンドラインオプション 27
- 2.2 PLOP・PLOP DS コマンドラインの作成例 32

## 3 PLOP・PLOP DS ライブラリの言語バインディング 33

- 3.1 C バインディング 33
- 3.2 C++ バインディング 36
- 3.3 COM バインディング 39
- 3.4 Java バインディング 40
- 3.5 .NET バインディング 42
- 3.6 Perl バインディング 43
- 3.7 PHP バインディング 44
- 3.8 Python バインディング 46
- 3.9 RPG バインディング 47

## 4 PDF セキュリティ 49

4.1 さまざまな PDF セキュリティ機能 49

4.2 PLOP の PDF セキュリティ機能 53

4.3 コマンドラインで PDF 文書を保護 56

## 5 PLOP DS による電子署名 59

5.1 電子署名の基本概念 59

5.2 デジタル ID の取得と管理 60

5.3 PLOP DS で PDF 文書に署名 63

5.4 PLOP DS の署名の暗号の詳細 66

5.5 Acrobat で電子署名を検証 67

## 6 pCOS インタフェース 71

## 7 PLOP・PLOP DS ライブラリ API リファレンス 73

7.1 オプションリスト 73

7.2 一般関数 75

7.3 文書入力・出力関数 78

7.4 例外処理 87

7.5 オプション処理 89

7.6 pCOS 関数 91

7.7 Unicode 変換関数 94

## A PDFlib を PLOP または PLOP DS と組み合わせる 97

## B PLOP ライブラリクイックリファレンス 98

## C 変更履歴 100

## 索引 101

# ○ 初めての PLOP・PLOP DS

## 0.1 ソフトウェアをインストール

PLOP と PLOP DS は、Windows システム用は統合インストーラパッケージとして、その他すべての対応オペレーティングシステム用は統合圧縮アーカイブとして頒布されています。インストーラとアーカイブの中には、PLOP/PLOP DS コマンドラインツールと PLOP/PLOP DS ライブラリが入っており、説明書と作成例も同梱されています。パッケージをインストールまたは解凍した後は、以下の手順を推奨します。

- ▶ PLOP と PLOP DS のさまざまな機能については、その概略紹介が 1 章にあります。
- ▶ PLOP/PLOP DS コマンドラインツールの利用者は、その実行形式をただちに使うことができます。利用できるオプションは 2.1 節「PLOP・PLOP DS コマンドラインオプション」(27 ページ) で説明されているほか、PLOP コマンドラインツールにオプションを何も付けずに実行したときにも表示されます。
- ▶ PLOP/PLOP DS ライブラリ / コンポーネントの利用者は、選んだ環境に応じて 3 章の中のいずれかの節を読み、インストールされている作成例に目を通すべきです。Windows では、PLOP と PLOP DS のプログラミング作成例は、スタートメニューから (COM・.NET の場合)、あるいはインストールディレクトリから (それ以外の言語バインディングの場合) 呼び出すことができます。

PLOP または PLOP DS の商用ライセンスを入手した場合は、次のページに従って、自分のライセンスキーを適用する必要があります。

**評価版の制限** PLOP/PLOP DS のコマンドラインツールとライブラリは、商用ライセンスがなくても、完全動作する評価版として使用することができます。有効なライセンスキーを適用しないと、PLOP は *unlicensed* というテキストを文書のメタデータに入れ込み、文書の先頭に追加表紙ページを挿入します。

入力が PDF/X または PDF/A のいずれかに準拠していても、場合によっては、この表紙ページが挿入されることによって、生成される PDF 出力が、PDF/X または PDF/A に準拠しないものになることがあります。このとき、非準拠となるのはこの表紙ページだけであり、有効なライセンスキーを適用すれば以後はこの問題は発生しません。この点を試すには、この追加表紙ページは、以下の条件の一方ないし両方が真ならば挿入されません。

- ▶ 暗号化を、決まったパスワード文字列 *demo* または *DEMO* で行うとき (*userpassword・masterpassword* オプション)。
- ▶ 電子署名の適用に用いるデジタル ID のサブジェクト名 (**共通名・CN** とも呼ばれます) が、*demo* または *DEMO* を含むとき。試験に適したデジタル ID が PLOP パッケージに入っています。

pCOS の諸機能は、評価モードでは、小さな文書に限定されます (10 ページ未満かつ 1MB 未満)。

PLOP または PLOP DS の未ライセンス版は、実用目的に使用してはならず、製品の評価目的でのみ使用が許されます。このソフトウェアを実用目的に使用するには、有効なライセンスが必要です。

## 0.2 PLOP/PLOP DS ライセンスキーを適用

PLOP/PLOP DS を実用目的に使用するには、有効なライセンスが必要です。ライセンスを購入したら、追加表紙ページが出ないように、また任意のパスワードが使えるようにするために、自分のライセンスキーを適用する必要があります。ライセンスキーの適用にはいくつかの方法があります。以下に示す方法のいずれかを選んでください。

`PLOP_set_option()` の `frontpage` オプションを `false` にすると、有効なライセンスキーが見つからなかったときは、表紙ページは生成されずに例外が発生します。

**注記** PLOP/PLOP DS ライセンスキーはプラットフォーム依存であり、その購入対象のプラットフォームでのみ使用できます。PLOP DS ライセンスキーでは PLOP の全機能が有効になりますが、PLOP ライセンスキーでは、PLOP DS でのみ利用できる署名機能は有効になりません。

**Windows インストーラ** Windows ユーザーは、提供されているインストーラを使って PLOP/PLOP DS をインストールする際にライセンスキーを入力することができます。Windows ではこの方法を推奨します。レジストリへの書き込みアクセスを持たない場合や、インストーラを使えない場合は、以下に示す代替方式を参照してください。

**API 呼び出しで実行時にライセンスキーを適用** 動作時にライセンスキーを設定する行を、自分のスクリプトまたはプログラムに追加します。`license` パラメタを、PLOP オブジェクトをインスタンス化した直後に（すなわち、`PLOP_new()` または同等の呼び出しの後に）設定する必要があります。具体的な文法は、使うプログラミング言語によります：

▶ COM/VBScript の場合：

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

▶ .NET/C# の場合：

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

▶ C・Python の場合：

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

▶ C++・Java の場合：

```
p.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：

```
d licenseopt      s          20
c                  eval      licenseopt=%ucs2('license=... あなたのライセンスキー ...')
c                  callp     PLOP_set_option(PLOP:licenseopt:0)
```

**ライセンスファイルを使用** 実行時呼び出しによってライセンスキーを与えるのではなく、テキストファイル内に以下の形式に従ってライセンスキーを書き込むこともできます（PLOP ディストリビューションに含まれているライセンスファイルテンプレート `licensekeys.txt` を使えます）。「#」キャラクタで始まる行はコメントを内容としますので無視されます。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ます：

```
# PDFlib GmbH製品のライセンス情報
PDFlib license file 1.0
PLOP      4.1      ...あなたのライセンスキー ...
```

ライセンスファイルには、複数の PDFlib GmbH 製品のライセンスキーを、別々の行ごとに含めることもできます。また、複数のプラットフォーム用のライセンスキーを含めて、1 個のライセンスファイルを複数のプラットフォームで使いまわすことも可能です。ライセンスファイルは以下の方法で設定できます：

- ▶ **licensekeys.txt** という名前のファイルが、すべてのデフォルト位置内で検索されます（「デフォルトファイル検索パス」（8 ページ）参照）。
- ▶ **licensefile** パラメタを **set\_option()** API 関数で設定することもできます：

```
p.set_option("licensefile=/path/to/licensekeys.txt");
```

- ▶ PLOP コマンドラインツールの **--plopt** オプションを用いて、**licensefile** オプションをライセンスファイルの名前とともに与えます：

```
plop --plopt "licensefile /path/to/your/licensekeys.txt" ...
```

パス名に空白キャラクタが含まれる場合には、パスを中括弧で囲う必要があります：

```
plop--plopt "licensefile {/path/to/your/license file.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、システムコントロールパネルを用いて「システム」→「詳細設定」→「環境変数」を選択します。Unix では、下記のようなコマンドを適用します：

```
export PDFLIBLICENSEFILE=/path/to/licensekeys.txt
```

- ▶ i5/iSeries システムでは、ライセンスファイルは ASCII でエンコード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は下記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 製品について動作します）：

```
ADDENVVAR ENVVAR(PDFLIBLICENSEFILE) VALUE(/PLOP/4.1/licensefile.txt) LEVEL(*SYS)
```

**ライセンスキーをレジストリに** Windows では、ライセンスファイルの名前を下記レジストリキーに書きこむこともできます：

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

あるいは、ライセンスキーを直接下記レジストリキーのいずれかに書きこむことも可能です：

```
HKLM\SOFTWARE\PDFlib\PLOP4\license
HKLM\SOFTWARE\PDFlib\PLOP4\4.1\license
```

MSI インストーラは、インストール時に与えられたライセンスキーを、これらの項目の末尾に書き込みます。

**注記** 64 ビット Windows システム上で手作業でレジストリを操作する際には注意が必要です。通常、64 ビット PLOP バイナリは Windows レジストリの 64 ビットビューとともに動作するのに対して、64 ビットシステム上で走る 32 ビット PDFlib バイナリはレジストリの 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キーを手作業で追加する必要がある場合には、必ず、**regedit** ツールの 32 ビットバージョンを使用してください。

これは「スタート」→「ファイル名を指定して実行...」ダイアログから下記のように呼び出すことができます：

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

**デフォルトファイル検索パス** Unix・Linux・Mac OS X・i5/iSeries システムでは、ファイルに対してパス・ディレクトリ名を指定していなくても、いくつかのディレクトリがデフォルトで検索されます。以下のディレクトリが検索されます：

```
<rootpath>/PDFlib/PLOP/4.1/resource/cmap  
<rootpath>/PDFlib/PLOP/4.1/resource/codelist  
<rootpath>/PDFlib/PLOP/4.1/resource/glyphlst  
<rootpath>/PDFlib/PLOP/4.1/resource/fonts  
<rootpath>/PDFlib/PLOP/4.1/resource/icc  
<rootpath>/PDFlib/PLOP/4.1  
<rootpath>/PDFlib/PLOP  
<rootpath>/PDFlib
```

Unix・Linux・Mac OS X では、<rootpath> はまず /usr/local へ置き換えられ、ついで HOME ディレクトリへ置き換えられます。i5/iSeries では <rootpath> は空です。

**ライセンス・リソースファイルのデフォルトファイル名** デフォルトでは、デフォルト検索パスディレクトリ内で下記のファイル名が検索されます：

licensekeys.txt (ライセンスファイル)

この機能を利用すれば、環境変数や実行時オプションを設定せずにライセンスファイルを扱うことが可能になります。

**i5/iSeries・zSeries におけるマルチシステムライセンスファイル** i5/iSeries と zSeries 用のライセンスキーはシステム固有なので、マルチシステム間で共有することはできません。リソース共有を実現して、マルチシステムで共有できる単一のライセンスファイルを扱いたいときは、次のライセンスファイル形式を用いれば、複数のシステム固有のキーを単一のファイルに保持することができます。

```
PDFlib license file 2.0  
# Licensing information for PDFlib GmbH products  
PLOP      4.1      ...あなたのライセンスキー...    ...マシン1に対するシリアル番号...  
PLOP      4.1      ...あなたのライセンスキー...    ...マシン2に対するシリアル番号...
```

1 行目のバージョン番号を変えることに留意してください。複数のライセンスキーを列挙し、それぞれ末尾に 8 桁の 16 進シリアル番号 (i5/iSeries の場合) または 4 桁の 16 進 CPU ID (zSeries の場合) を記述します。

**さまざまなライセンシングオプション** 1 台ないし複数のサーバ上で PLOP を使用したり、PLOP をあなた自身の製品とともに再頒布したりするための、さまざまなライセンシングオプションが利用可能です。また当社では、サポート契約・ソースコード契約も提供しています。商用 PDFlib ライセンスの取得にご関心がある場合や、ご質問がある場合は、ご連絡ください：

PDFlib GmbH, Licensing Department  
Franziska-Bilek-Weg 9, 80339 München, Germany

*www.pdflib.com*

電話 +49 • 89 • 452 33 84-0

FAX +49 • 89 • 452 33 84-99

ライセンスに関するお問い合わせ：*sales@pdflib.com*

PDFlib ライセンス保持者向けサポート：*support@pdflib.com*



# 1 PLOP と PLOP DS の諸機能

## 1.1 概観

PLOP には 2 つの種類があります。基本版の PLOP と、拡張版の PLOP DS です。

**PLOP の諸機能** PLOP は、以下の種類の PDF 処理機能を持っています。

- ▶ 保護：PDF 文書を、ユーザーパスワードまたはマスターパスワード（ないし両方）で暗号化します。PDF の暗号化を、利用者がその文書のマスターパスワードを知っていれば除去します。権限設定（印刷を許可しない、テキスト抽出を許可しない等）を、利用者がその文書のマスターパスワードを知っていれば追加・除去します。
- ▶ PDF 文書を線形化して、PDF ファイルを Web サーバから取得する際のビューア体験を改善します（後述）。
- ▶ 冗長なオブジェクトを減らすことによって、PDF 文書のサイズを最適化します。
- ▶ 破損している PDF 文書を修復します。
- ▶ 内蔵の pCOS インタフェースを使って、文書のセキュリティ状態（ユーザーパスワードまたはマスターパスワードで暗号化されている）・権限設定や文書のメタデータ等、さまざまな特性に関する情報を取得します。
- ▶ 定義済みやカスタムの文書情報項目を挿入・取得します。
- ▶ XMP メタデータを挿入・取得します。

**PLOP DS の諸機能** PLOP DS は、PLOP の全機能に加えて、PDF 文書に電子署名を適用する機能を提供します。この署名は Adobe Acrobat か Adobe Reader で検証することができます。

署名は、認証形式 PKCS#12 または PFX のデジタル ID から作成することができます。Windows では、Windows 認証ストアの中のデジタル ID を使うこともできます。Windows 等のプラットフォームでは、PKCS#11 に対応した暗号トークンを用いることも可能です（スマートカード・USB スティック等）。

**利点** PDFlib PLOP と PLOP DS は以下の利点を提供します。

- ▶ PLOP・PLOP DS の操作はすべて PDF/X・PDF/A 対応です。すなわち、もし入力がこのいずれかの規格に準拠しているときは、出力は可能な限りその同じ規格に準拠したものになることが保証されています。もしこれが可能でないときは（PDF/A 入力に対し暗号化を要求した場合等）、その操作を拒否させることもできますし、あるいは規格識別を除去させることもできます。
- ▶ PLOP はスタンドアロンのツールであり、サードパーティのソフトウェアを一切必要とせず、PDF の読み取り・暗号化・署名・書き出しが可能です。
- ▶ PLOP は技術的にも法的にも、サーバ上に投入することができ、完全にスレッドセーフであり、メモリリークについても検査されています。PLOP は高負荷のサーバ利用を想定して製作されており、Web サーバ環境や大量バッチ処理等での活用が可能です。
- ▶ PLOP には多くのプラットフォーム向けのものがあり、いくつかのプログラミング環境用のものがあります。
- ▶ 柔軟性を高めるため、PLOP はコマンドラインツールとしても、さまざまな開発言語用のプログラミングライブラリ（コンポーネント）としても利用可能です。

**PLOP/PLOP DS のコマンドラインツール、それともライブラリ** PLOP/PLOP DS には、さまざまな開発言語用のプログラミングライブラリ（コンポーネント）と、バッチ処理のためのコマンドラインツールがあります。どちらも機能の中身は同じですが、適した応用の分野が異なります。ライブラリとコマンドラインツールのどちらを選ぶか、以下にそのガイドラインを挙げます。

- ▶ PLOP/PLOP DS コマンドラインツールは、PDF 文書のバッチ処理に適しています。プログラミングは必要がなく、強力なコマンドラインオプション群が提供されており、それを使って複雑なワークフローに組み込むことができます。PLOP/PLOP DS コマンドラインツールは、ライブラリの使用に対応していない環境からも呼び出すことができます。
- ▶ PLOP/PLOP DS プログラミングライブラリは、Active Server Pages (ASP)・Visual Basic・Java（サーブレットを含む）・PHP・RPG・プレーン C・C++ アプリケーション開発といった、広く利用されているさまざまな開発環境に、きれいに組み込むことができます。

PLOP/PLOP DS ライセンスは、コマンドラインツールとライブラリの両方をカバーしています。

## 1.2 説明書と各サンプルへのロードマップ

**PLOP の諸言語バイディング用ミニサンプル群** PLOP ディストリビューションには、対応するすべての言語バイディング用のシンプルなプログラミング作成例が多数含まれています。これらは以下のように、PLOP ライブラリによる基本的な各種プログラミングタスクのデモンストレーションとなっています。

- ▶ サンプル *encrypt* では、暗号化されていない PDF 文書をユーザーパスワードとマスターパスワードで暗号化しています。
- ▶ サンプル *decrypt* では、暗号化されている PDF 文書とそのマスターパスワードを用いて復号しています。
- ▶ サンプル *noprint* では、アクセス権限 *noprint* と *nocopy* を設定し、ファイルをマスターパスワードで暗号化しています。
- ▶ サンプル *dumper* では pCOS インタフェースを用いて、ファイルの各種一般プロパティや、暗号化状態に関する情報のほか、文書情報・XMP メタデータを収集しています。
- ▶ サンプル *insertxmp* では、ファイルから XMP メタデータを読み取って、その XMP を PDF 文書に挿入しています。試験用のサンプル XMP ファイルを添付しています。
- ▶ サンプル *linearize* では、既存の PDF 文書に線形化を施し、また、文書情報項目を変更しています。

最適化処理はデフォルトで有効になっているため（電子署名を適用する際を除く）、すべてのサンプルで最適化が暗黙のうちにデモンストレーションされます。

以下のミニサンプルは、PLOP DS で使用するためのものです。

- ▶ サンプル *sign* では、既存の PDF 文書に電子署名を施す方法を示しています。
- ▶ *hellosign* では、PDFlib で文書を動的に作成し、それを PLOP に受け渡して（メモリ上で）、PLOP でそれに電子署名を施す方法を示しています。この作成例の動作には PDFlib 製品が必要ですが、それは PLOP パッケージには含まれていないのでご注意ください。ただし、無料評価版を当社ウェブサイトから入手することができます。

**注記** Windows Vista・Windows 7 では、これらのミニサンプルはデフォルトで「Program Files」ディレクトリにインストールされます。Windows Vista で新たに導入されたセキュリティ方式により、これらのサンプルによって作成された PDF 出力ファイルは、「互換性ファイル」の下でしか見えません。この問題を避けるため、作成礼をユーザーディレクトリにコピーすることを推奨します。

**PLOP コマンドラインツールの呼び出しサンプル** PLOP コマンドラインツールにはさまざまなオプションが用意されており、その解説は「PLOP・PLOP DS コマンドラインオプション」（27 ページ）にあります。1 章のこの後の各項や、2.2 節「PLOP・PLOP DS コマンドラインの作成例」（32 ページ）やその他の章に、PLOP コマンドラインツールの呼び出しサンプルが含まれています。

**pCOS クックブック** pCOS クックブックは、PLOP に内蔵されている pCOS インタフェースのためのコード集です。次の URL にあります。

[www.pdfli.com/pcos-cookbook](http://www.pdfli.com/pcos-cookbook)

pCOS インタフェースの詳細は、PLOP パッケージに含まれている pCOS パスリファレンスで解説しています。

## 1.3 暗号化・復号・権限

注記 PDF 文書の暗号化・復号および権限制限については、詳しくは 4 章で説明しています。本節では概観と、手始めのいくつかの例を示すとどめます。

**権限設定を取得** PLOP ライブラリに内蔵されている pCOS プログラミングインタフェースを使えば、PDF 文書のさまざまな権限設定を取得することができます。必要な関数呼び出しと引数は、すべての PLOP パッケージに入っているミニサンプル *dumper* で見ることができます。PLOP コマンドラインツールでこれに対応するオプションは `--info` です (1.7 節「pCOS で文書情報を取得」(18 ページ) にある例を参照)。

**PLOP で文書を暗号化** `PLOP_create_file()` で `userpassword` オプションか `masterpassword` オプション (ないし両方) を指定すれば、文書を暗号化することができます。ただしユーザーパスワードには必ずマスターパスワードが必要ですが、その逆は真ではありません。PDF 文書の暗号化については、そのサンプルコードを、すべての PLOP パッケージに入っているミニサンプル *encrypt* で見ることができます。PLOP コマンドラインツールでこれらと等価なオプションは `--user` と `--master` です。

例：ユーザーパスワード `demo` とマスターパスワード `DEMO` でファイルを暗号化：

```
plop --user demo --master DEMO --outfile encrypted.pdf input.pdf
plop -u demo -m DEMO -o encrypted.pdf input.pdf
```

**PLOP で権限制限を指定** `PLOP_create_file()` の `permissions` オプションにはさまざまなキーワードを設定することができます (表 4.3 (54 ページ) 参照)、これによって権限制限を指定することができます。PDF 文書の権限設定の指定については、そのサンプルコードを、すべての PLOP パッケージに入っているミニサンプル *noprint* で見ることができます。PLOP コマンドラインツールでこれと等価なオプションは `--permissions` です。ただし権限設定には必ずマスターパスワードが必要です。

例：文書をマスターパスワード `DEMO` で暗号化し、文書の印刷と内容のコピーを不許可にする：

```
plop --master DEMO --permissions "noprint nocopy" --outfile encrypted.pdf input.pdf
plop -m DEMO --permissions "noprint nocopy" -o encrypted.pdf input.pdf
```

**PLOP で文書を復号** `PLOP_create_file()` で `password` オプションに適切なユーザーパスワードかマスターパスワードを指定すれば、文書を復号することができます。PDF 文書の復号については、その完全なサンプルコードを、すべての PLOP パッケージに入っているミニサンプル *decrypt* で見ることができます。PLOP コマンドラインツールでこれと等価なオプションは `--password` です。

例：1 個のファイルを、マスターパスワード `DEMO` で復号。入力文書にアクセス制限が適用されていても、それらはすべて除去されます (出力は復号されるので)：

```
plop --password DEMO --outfile decrypted.pdf encrypted.pdf
plop -p DEMO -o decrypted.pdf encrypted.pdf
```

暗号化や復号については、4.3 節「コマンドラインで PDF 文書を保護」(56 ページ) にも作成例があります。

## 1.4 Web 最適化（線形化）PDF

PLOP では、PDF 文書に、線形化という処理を施すことができます。そこから生まれる特性は、「Web 表示用に最適化」と Acrobat では呼ばれています。線形化は、PDF ファイルの中のさまざまなオブジェクトを認識して、情報を付加し、それによって表示を高速化するものです。

線形化されていない PDF は、クライアントへまるごと転送する必要がありますが、線形化された PDF であれば、Web サーバはバイトサービングという処理を用いて、それを 1 ページずつ転送することが可能になります。これによって Acrobat（ブラウザのプラグインとして動作している）は、PDF 文書内の個々の部分を別々に取得することができるようになります。その結果としてユーザーは、文書全体がサーバからダウンロードされおわるまで待たなくても、その文書の最初の 1 ページの閲覧を開始することができます。このことはユーザー体験の向上をもたらします。

ただし、Web サーバが PDF データをストリーム転送する先はブラウザであって、PLOP ではありません。逆に PLOP は、バイトサービング可能な PDF ファイルを作り出すのです。PDF のバイトサービングを活用するためには、以下のすべての要請が満たされる必要があります。

- ▶ PDF 文書が線形化されている必要があります。これは PLOP で実現できます。線形化は、暗号化または復号と同時に、一度で適用することができます。Acrobat では、ファイルが線形化されているかを調べるには、その文書のプロパティを見ます（「Web 表示用に最適化：はい」）。
- ▶ Web サーバがバイトサービングに対応している必要があります。その基本となっているバイトレンジプロトコルは HTTP 1.1 の一部ですので、現在のすべての Web サーバに実装されています。
- ▶ ユーザーが Acrobat をブラウザのプラグインとして使っていて、かつ Acrobat でページごとのダウンロードを有効にしている必要があります（Acrobat 8/9/X：「編集」→「環境設定」→「一般...」→「インターネット」→「Web 表示用に最適化を許可」）。なお、これはデフォルトでは有効になっています。

PDF ファイルが大きければ大きいほど（ページ数で計るにせよ MB で計るにせよ）、それを Web で送受信するとき、線形化の恩恵をより多く受けることになります。

線形化と暗号化 / 復号とは、組み合わせて適用することが可能です。ただし、保護されたファイルを線形化するためには、適切なマスターパスワードを与える必要があります（表 4.2 参照）。

**注記** PDF 文書を線形化すると、線形化情報が付け加えられるため、一般にそのファイルサイズはわずかに増大します。この増大分は、適用された最適化技法によって、帳消しになることもならないこともあります（1.5 節「最適化（軽量化）」（16 ページ）参照）。

**PLOP で PDF 文書を線形化** `PLOP_create_file()` で `linearize` オプションを指定すれば、線形化処理を有効にすることができます。PDF 文書の線形化については、そのサンプルコードを、すべての PLOP パッケージに入っているミニサンプル `linearize` で見ることができます。

PLOP コマンドラインツールでこれと等価なオプションは `--webopt` です。例：ディレクトリ内のすべての PDF 文書を線形化し（これらはどれもパスワードが不要と前提）、できたファイルをターゲットディレクトリ `output` へコピー。詳細度レベル 2 は、すべての入力・出力ファイルについて、その処理時に名前を印字します：

```
plop --verbose 2 --webopt --targetdir output *.pdf
plop -v 2 -w -t output *.pdf
```

## 1.5 最適化（軽量化）

PDF 文書の処理過程において、PLOP は、他のさまざまな操作に加えて、以下のようなファイル最適化を施すこともできます。

- ▶ PLOP は、同一データの重複出現を検出して、1つを残して全部削除します。これは主にフォントや画像が対象となりますが、それ以外の種類のデータについても適用されることがあります（ICC プロファイル等や、あるいはページでさえも、その内容がまるごと同一であれば）。埋め込まれているフォントや画像は、もし他のフォントや画像の中身がまったく同じデータであれば、削除されます。削除したデータへの参照はすべて、そのフォントや画像を残した箇所への参照に置き換えられます。たとえば、複数の PDF 文書を集めて一つの文書にした場合、もしそれらに同じフォントが埋め込まれていたならば、できあがった PDF の中には余分なフォントデータが入っています。PLOP はその冗長なフォントデータを削除して、そのフォントのデータを1つだけ残します。
- ▶ 使われていないオブジェクトは、**ガベージコレクション**として知られる処理によって、PDF ファイルから削除されます。場合によっては（Acrobat の「名前を付けて保存 ...」／「別名で保存 ...」コマンドでなく「保存」コマンドが使われていると）Acrobat は、変更情報をファイルに追加して、文書の以前の状態を残したままにしています。PLOP は、文書の古いバージョンにまつわるオブジェクトをすべて削除します。
- ▶ 出力は、コンパクトな文法を用いて書き出されます。たとえば、不必要なホワイトスペースは除去され、ある種の非効率な構造（間接整数オブジェクト）はもっと効率的な同等内容に置き換えられ、16進文字列（インデックスカラースペースのためのカラーパレット等）はもっとコンパクトなバイナリ表現に置き換えられます。

PLOP では、情報の喪失につながるような最適化の仕方（フォントの埋め込みをやめたり、画像をダウンサンプルしたり等）は一切行いません。入力とまったく同じ品質で文書を表示したり印刷したりするために必要な情報がすべて、出力内へ引き継がれます。

**PLOP で PDF 文書を最適化** PLOP において最適化はデフォルトで有効になっているので、有効にするためにオプションを与える必要は一切ありません。ただし、極限的なパフォーマンス要請に応える目的で、この最適化処理を無効にするには、`PLOP_create_file()` で `optimize=none` オプションを指定します。PLOP コマンドラインツールでこれと等価なオプションは `--fast` です。

例：PLOP コマンドラインツールで文書を最適化：

```
plop --outfile optimized.pdf input.pdf
plop -o optimized.pdf input.pdf
```

## 1.6 破損 PDF のための修復モード

PLOP では、破損を受けている PDF のための修復モードを実装しており、ある種の破損文書をも処理することが可能になっています。しかし稀には、PLOP が修復できずに拒否される破損 PDF 文書もあります。

**PLOP で PDF 文書を修復** 修復モードは、破損を受けている入力に PLOP が出会ったときに自動的に有効になります。しかし、`PLOP_open_document()` の `repair=force` オプションを使って、文書を開く際に何も問題が起こらなかった場合にも修復モードを強制することもできます。PLOP コマンドラインツールでこれと等価なオプションは `--inputopt repair=force` です。`repair=none` を指定して修復モードを無効にすることもできます。

例：PLOP コマンドラインツールで文書の再構築を強制：

```
plop --inputopt repair=force --outfile repaired.pdf damaged.pdf
plop --inputopt repair=force -o repaired.pdf damaged.pdf
```

**無効な XMP メタデータ** PLOP は XMP メタデータ内のある種の問題を修復します。しかし問題によっては修復できないものもあります。たとえば XML メタデータが XML パーシングエラーを引き起こした場合にはつねにその XMP は使用不能とされます。PLOP では、無効な XMP に出会った場合の処理動作を制御するための `xmppolicy` オプションを提供しています。詳しくは「無効な XMP メタデータの扱い」(21 ページ)を参照してください。

## 1.7 pCOS で文書情報を取得

pCOS インタフェースについては詳しくは pCOS パスリファレンスで解説しています。この項では、概要と、いくつかの導入的な作成例を紹介します。

PLOP ライブラリに内蔵されている pCOS プログラミングインタフェースを使えば、PDF 文書のさまざまな特性を取得することができます。pCOS による文書情報の取得については、そのサンプルコードを、すべての PLOP パッケージに入っているミニサンプル *dumper* で見るすることができます。PLOP コマンドラインツールでこれと等価なオプションは `--info` です。

例：PDF 文書に関するセキュリティ等の情報を表示：

```
plop --info *.pdf
plop -i *.pdf
```

このプログラム呼び出しの出力結果は以下のようになります：

```
File name: PLOP-manual.pdf
PDF version: 1.6
Encryption: No encryption
  Master pw: false
  User pw: false
  nocopy: false (copying is allowed)
  nomodify: false (adding form fields and other changes is allowed)
  noannotations: false (adding or changing comments or form fields is allowed)
  noassemble: false (insert/delete/rotate pages, creating bookmarks is allowed)
  noforms: false (filling form fields is allowed)
  noaccessible: false (extracting text or graphics for accessibility is allowed)
  nohighresprint: false (high-resolution printing is allowed)
plainmetadata: true (metadata is not encrypted)
  Linearized: true
  PDF/X status: none
  PDF/A status: none
  Tagged PDF: false
  Signatures: 0
Reader-enabled: false

No. of pages: 90
No. of fonts: 8
  embedded Type 1 CFF font TheSans-Plain
  embedded Type 1 CFF font TheSansExtraBold-Plain
  ...他のフォント群...

CreationDate: 'D:20100616003116Z'
Subject: 'PDFlib PLOP: PDF Linearization, Optimization, Protection'
Author: 'PDFlib GmbH'
Creator: 'FrameMaker 7.0'
Producer: 'Acrobat Distiller 8.1.0 (Windows)'
ModDate: 'D:20070616021141Z'
  Title: 'PDFlib PLOP and PLOP DS Manual'

XMP meta data: is present
```

## 1.8 文書情報項目を挿入・取得

PDF では、文書に関する一般情報を持つ文書メタデータとして、2つの種類を利用することができます：文書情報項目と XMP メタデータです。

文書情報項目とは、キーに文字列を関連づけたものであり、構造化されていない何らかの情報を保持します。定義済みの情報キーである **Subject**・**Title**・**Author**・**Keywords** が広く利用されていますが、他にも特定の目的のために任意のカスタムキーを定義することができます。文書情報項目は、古くてシンプルな種類の PDF メタデータであるということが出来ます。

PLOP を使えば、新しい文書情報項目を追加したり、既存の情報項目の値を書き換えたりすることができます。定義済みの項目もカスタムの項目も設定可能です。入力文書の中に XMP 文書メタデータがあった場合は、メタデータの整合性を保つために、すべての定義済み情報項目が自動的に XMP メタデータへ同期されます。

**PLOP で文書情報項目を挿入** `PLOP_create_file()` で `docinfo` オプションを指定すれば、文書情報項目を設定することができます。文書情報項目の設定については、そのサンプルコードを、すべての PLOP パッケージに入っているミニサンプル `linearize` で見ることができます（このサンプルでは線形化のほかに、文書情報をいかに設定するかも例示しています）。

例：定義済み文書情報項目「**Subject**」と、カスタム情報項目「**Department**」を指定。なお、「**Product Manual**」を中カッコで囲ってスペースキャラクタを保護しています：

```
docinfo={Department Techdoc Subject {Product Manual}}
```

このオプションは PLOP コマンドラインツールに、以下のように `--outputopt` オプションで与えることもできます：

```
plop --outputopt "docinfo={Department Techdoc Subject {Product Manual}}" ←  
  --outfile output.pdf input.pdf  
plop --outputopt "docinfo={Department Techdoc Subject {Product Manual}}" ←  
  -o output.pdf input.pdf
```

**PLOP で文書情報項目を追加** PLOP ライブラリに内蔵されている pCOS プログラミングインタフェースを使えば、PDF 文書から文書情報項目（キーと値）を取得することもできます。必要な関数呼び出しと引数は、すべての PLOP パッケージに入っているミニサンプル `dumper` で見ることができます。

PLOP コマンドラインツールでこれに対応するオプションは `--info` です（1.7 節「pCOS で文書情報を取得」（18 ページ）にある例を参照）。

## 1.9 XMP メタデータを挿入・抽出

XMP (*Extensible Metadata Platform*<sup>1</sup>) は、さまざまな定義済みプロパティを持った XML フレームワークの一種です。しかしその名前が暗示するように、XMP は、個々の要請を満たす目的で、カスタムの拡張スキーマを用いて拡張することもできるようになっています。XMP は文書情報項目よりもはるかに強力であり、またたとえば PDF/A 標準規格では必須とされています。多くの業界団体が、XMP に基づいた規格を、デジタルイメージングやプリプレスデータ交換等、さまざまな垂直アプリケーションのために策定しています。

XMPに関するより詳しい情報や、他の情報源へのリンクが [www.pdflib.com/knowledge-base/xmp-metadata/](http://www.pdflib.com/knowledge-base/xmp-metadata/) にあります。

PLOP を使えば、PDF 文書に XMP メタデータを挿入したり、PDF から XMP を抽出したりすることができます。挿入された XMP の検証も行われるので、生成される出力は必ず有効であることが保証されています。入力文書が PDF/A-1 標準規格に準拠している場合、ユーザーが与える XMP は、PDF/A で定められている XMP の諸規則に準拠していなければなりません。こうした規則 (XMP 拡張スキーマの検証を含む) についても PLOP は検査を行いますので、PDF/A-1 入力にユーザーから与えられた XMP を加えた結果が必ず準拠 PDF/A 出力になることが保証されています。

PLOP による XMP の挿入は、以下の状況や、その他多くの状況で利用することができます (カッコ内は、PLOP ディストリビューションに含まれているサンプル XMP ファイルの名前です)。

- ▶ XMP メタデータを PDF/A-1 文書に追加。PDF/A-1 規格で定義されている XMP 拡張スキーマにも対応しています (*machine\_pdfa1.xmp*)。
- ▶ デジタル化されたレガシ文書のスキャン過程を記述した XMP メタデータを追加 (*engineering.xmp*)。
- ▶ Ghent Workgroup (GWG) Ad Ticket スキームに従った XMP メタデータを追加 (*gwg\_ad\_ticket.xmp*)。詳しくは [www.gwg.org/jobtickets.phtml](http://www.gwg.org/jobtickets.phtml) を参照してください。
- ▶ 会社独自の XMP メタデータを追加 (*acme.xmp*)。

**PLOP で XMP メタデータを挿入** メタデータを挿入するためには、有効な XMP メタデータを UTF-8 形式で持つファイルを作成する必要があります。*PLOP\_create\_file()* で *metadata* オプションを指定すれば、XMP を挿入することができます。このオプションには、いくつかのサブオプションも用意されています。PDF 文書への XMP の挿入については、そのサンプルコードを、すべての PLOP パッケージに入っているミニサンプル *insertxmp* で見ることができます。

例: *gwg\_ad\_ticket.xmp* というファイルから XMP メタデータを挿入して、XMP 2004 標準規格にてらしてその XMP を検証させる:

```
plop --outputopt "metadata={filename=gwg_ad_ticket.xmp validate=xmp2004}" ←  
--outfile output.pdf input.pdf  
plop --outputopt "metadata={filename=gwg_ad_ticket.xmp validate=xmp2004}" ←  
-o output.pdf input.pdf
```

**PLOP で XMP メタデータを抽出** PLOP ライブラリに内蔵されている pCOS プログラミングインタフェースを使えば、PDF 文書から XMP メタデータを抽出することもできます。必要な関数呼び出しと引数は、すべての PLOP パッケージに入っているミニサンプル *dumper* で見ることができます。ただし、このサンプル *dumper* の中のサンプルコードは、

1. [www.adobe.com/products/xmp](http://www.adobe.com/products/xmp) を参照

実際に XMP メタデータを印字しているのではなく、単に文書内で見つかった XMP のサイズを報告しているだけです。

PLOP コマンドラインツールを使って XMP メタデータを抽出することはできません。当社では強力な pCOS コマンドラインツールを提供しており、これを使えば PDF 内の情報を抽出することが可能です。

**無効な XMP メタデータの扱い** PDF 文書はときに、XML レベルで無効な、あるいは XMP/RDF レベルで無効な XMP メタデータを含んでいることがあります。PLOP はデフォルトではそのような文書を拒絶し処理を停止します。このような入力文書についてより細かい制御を行いたい場合は、`PLOP_open_document()` に対して `xmppolicy` オプションを用いれば以下の場合を区別することができます：

- ▶ `xmppolicy=rejectinvalid`: デフォルトでは、無効な XMP があれば PLOP は PDF 出力を生成しません。
- ▶ `xmppolicy=ignoreinvalid`: 無効な XMP を無視し、デバッグ支援のために生成出力 XMP 内に XMP パーシングエラーメッセージのテキストを含めます。このオプションでは PDF/A または PDF/X-3/4/5 出力は一切生成されないことに留意してください。
- ▶ `xmppolicy=remove` : これは、望ましくないメタデータを削除するために有用です。

たとえば、無効な XMP メタデータによって文書群のバッチ処理が中断されるのを防ぐには、入力文書内の無効な XMP が引き起こす問題を無視することができます：

```
plop --inputopt "xmppolicy=ignoreinvalid" --outfile output.pdf input.pdf
plop --inputopt "xmppolicy=ignoreinvalid" -o output.pdf input.pdf
```

## 1.10 PLOP DS で電子署名

注記 PDF 文書に電子的に署名できる機能は、PLOP DS でのみ利用できるものであり、基本版の PLOP にはありません。

注記 PDF 文書に対する電子文書については、詳しくは 5 章で説明しています。本節では概観と、手始めのいくつかの例を示すにとどめます。

**署名の諸特性を取得** PLOP ライブラリに内蔵されている pCOS プログラミングインタフェースを使えば、PDF 文書の署名の諸設定を取得することができます。必要な関数呼び出しと引数は、pCOS クックブックの *interactive\_elements/signatures* トピックにあります。PLOP コマンドラインツールでこれに対応するオプションは `--info` です (1.7 節「pCOS で文書情報を取得」(18 ページ)を参照)。

**PLOP DS で文書に署名** 署名を適用するにはデジタル ID が必要であり、これはファイルとして、または Windows 証明書ストアの中で、あるいは暗号トークン上で (スマートカード・USB スティック等) 得ることができます。前者はデジタル ID にアクセスするためにパスワードが必要ですが、Windows 証明書ストアは通常は Windows ログインで保護されているのでパスワードは不要です。暗号トークンは多くの場合 PIN で保護されています。

電子署名を適用するには、`PLOP_create_file()` で `sign` オプションを指定すればよく、これにはいくつかのサブオプションも指定できます。PDF 文書への署名については、そのサンプルコードを、すべての PLOP パッケージに入っているミニサンプル `sign` で見ることができます。PLOP コマンドラインツールでこれと等価なオプションは `--signopt` です。ミニサンプル `hellosign` では、PDFlib で PDF 文書を動的に作成したのち、PLOP DS で署名を施す方法を示しています。

例：ファイル `demo2048.p12` 中のデジタル ID を使って、PDF 文書に不可視の署名を作成。デジタル ID に対するパスワードは、ファイル `pw.txt` に入っています：

```
plop --signopt "digitalid={filename=demo2048.p12} passwordfile=pw.txt" ←
  --outfile signed.pdf input.pdf
plop -S "digitalid={filename=demo2048.p12} passwordfile=pw.txt" -o signed.pdf input.pdf
```

(Windows のみ) Windows 証明書ストアの中の証明書 (デフォルトのストア「`My`」の中の) を使って、PDF 文書に不可視の署名を作成。ここでは、デジタル ID はパスワードを与える必要がないように、あなたの Windows ログインによって保護されていることを前提としています：

```
plop --signopt "engine=miscapi digitalid={certstore={store=My subject={DEMO PLOP User 2048}}}" ←
  --outfile signed.pdf input.pdf
plop -S "engine=miscapi digitalid={certstore={store=My subject={DEMO PLOP User 2048}}}" ←
  -o signed.pdf input.pdf
```

(Windows 等、PKCS#11 に対応したプラットフォームのみ) 暗号トークンからのデジタル ID を用いて、PDF 文書に不可視の署名を作成します。トークンに対する PKCS#11 インタフェースはライブラリ `cryptoki.dll` に実装されており、このライブラリはスマートカードのサプライヤーによって提供されている必要があります。デジタル ID のパスワードは `pw.txt` に入っています。

```
plop --signopt "engine=pkcs#11 digitalid={filename=cryptoki.dll} passwordfile=pw.txt" ←
  --outfile signed.pdf input.pdf
```

```
plop -S "engine=pkcs#11 digitalid={filename=cryptoki.dll} passwordfile=pw.txt" ←  
-o signed.pdf input.pdf
```

5.3 章にはもっと署名に関する作成例があります。

## 1.11 PLOP の処理の詳細

**受け入れ可能な入力文書** PLOP は、以下の種類の PDF を受け入れ、処理、生成します：

- ▶ PDF 1.4 (Acrobat 5) およびそれより古いすべてのバージョン
- ▶ PDF 1.5 (Acrobat 6)
- ▶ PDF 1.6 (Acrobat 7)
- ▶ PDF 1.7 (Acrobat 8)。技術的に ISO 32000-1 と同等
- ▶ PDF 1.7 Adobe 拡張レベル 3 (Acrobat 9)
- ▶ PDF 1.7 Adobe 拡張レベル 8 (Acrobat X)
- ▶ PDF 2.0。ISO 32000-2 で仕様化されています

行いたい操作によっては、暗号化文書に対してはパスワードが必要になります。PLOP は、さまざまな種類の破損 PDF 文書の修復を試みます。入力文書と要求操作との、ある種の組み合わせには制約が課されますので、以下にそれを示します。

**PDF のバージョン** 生成される出力文書の PDF バージョン番号は、入力文書の PDF バージョン番号よりも低くなることは決してありませんが、以下に示すように、強制的に高い番号へ上げさせられることはあります。PDF 出力のバージョンは、`PLOP_create_file()` の、または PLOP コマンドラインツールの `--outputopt` オプションの、`compatibility` オプションで指定することができます。このオプションが指定されていないときは、PLOP は入力文書の PDF バージョンを使いますが、それは以下の規則に従って変更されます：

- ▶ 暗号化、すなわち `userpassword・masterpassword・permissions` オプションのうちいずれかが指定されているときは、バージョンは PDF 1.4 へ押し上げられます。
- ▶ 文書に電子的に署名すると (`sign` オプション)、バージョンは PDF 1.3 へ押し上げられません。
- ▶ XMP メタデータを挿入すると (`metadata` オプション)、バージョンは PDF 1.4 へ押し上げられます。
- ▶ `permissions` オプションで `plainmetadata` キーワードが指定されていると、バージョンは PDF 1.5 へ押し上げられます。

**PDF/A の実装根拠** 以下の規格と文書が、PLOP における PDF/A 実装の根拠を成しています：

- ▶ PDF/A 規格 (ISO 19005-1:2005)
- ▶ 技術補遺 1 (ISO 19005-1:2005/Cor 1:2007)
- ▶ 技術補遺 2 (ISO 19005-1:2005/Cor.2:2010)
- ▶ PDF/A 技術センターが発行したすべての関連 TechNote

**入力 PDF の特定の特性を放棄** PDF 文書の特性のうちいくつかは、PLOP の操作と衝突する可能性があります。たとえば、PDF/A 文書では暗号化を使うことは許されません。PDF/A 入力に暗号化をかけるよう指示されたら、PLOP はどのようにするべきでしょうか。デフォルトでは PLOP は、例外を発生させてその操作を拒絶します。しかし、`PLOP_create_file()` で、または PLOP コマンドラインツールの `--outputopt` オプションで、`sacrifice` というオプションを使えば、行わせたい操作に対して、入力特性よりも高い優先順位を与えることができます。上記の例でいえば、暗号化を許すために、PDF/A 準拠項目は文書から除去されます。

入力文書の特性と、行わせたい操作との組み合わせは、いく通りかあります。そのいずれの組み合わせにおいても、*sacrifice* オプションを使えば、文書のある特定の特性を放棄することによって操作が許されます（詳しくは表 7.4（81 ページ）参照）：

- ▶ PDF/A:PLOP は電子署名を、PDF/A 準拠なやり方で適用します。PDF/A-1a または PDF/A-1b 標準規格に準拠している入力文書は、署名をした後も PDF/A 準拠であることが保証されています。しかし、暗号化を施すこと、すなわち *userpassword-masterpassword-permissions* オプションのいずれかを指定することは、PDF/A では暗号化が一切禁止されているため、PDF/A 文書に対しては許されません。しかし *sacrifice={pdfa}* オプションを指定すれば、PDF/A 準拠を放棄することができます。
- ▶ PDF/X:PDF/X-1a/3/4/5 では、暗号化や、ページ上に可視の署名フィールドを置くことは許されていません。こうした状況では PLOP は例外を発生させますが、*sacrifice={pdfx}* オプションを指定すれば、PDF/X 準拠を放棄することができます。
- ▶ 入力文書の中の既存の署名は引き継がれません（証明書の署名も）。既存の署名を壊さないようにするため、PLOP は、すでに 1 つないし複数の署名を持っている文書に署名をすることを拒否します。*PLOP\_create\_file()* に対して、または PLOP コマンドラインツールの *--outputopt* オプションに対して、*sacrifice={signature}* というオプションを指定すれば、既存の署名を放棄することができます。
- ▶ PLOP は、Appearance を持たないフォームフィールド（PDFlib 6 または 7 で作成されたフォームフィールド等）を持った文書には署名を適用できないので、その種の入力に対しては例外を発生させます。その理由は、Acrobat はフォームフィールドに対して欠けているアピアランスストリームを再構成する必要があり、するとただちに署名は無効になってしまうためです。この場合、*PLOP\_create\_file()* に対して、または PLOP コマンドラインツールの *--outputopt* オプションの中で、*sacrifice={fields}* というオプションを指定すれば、すべての既存フォームフィールドを放棄することができます。
- ▶ 暗号化されていない文書の中に、暗号化されたファイル添付が入っているとき、そのパスワードが得られないと、処理はデフォルトでは停止します。この場合、*PLOP\_create\_file()* に対して、または PLOP コマンドラインツールの *--outputopt* オプションの中で、*sacrifice={encryptedattachments}* というオプションを指定すれば、すべての暗号化されたファイル添付を放棄することができます。このオプションを指定すると、パスワードが得られない暗号化されたファイル添付はすべて除去されます。

**入力文書から無条件に失われる特性** 以下の入力文書の特性は、PLOP のどの操作を施しても失われます：

- ▶ 入力文書が線形化されているとき、その線形化はデフォルトでは失われます。出力を線形化するには、*PLOP\_create\_file()* に *linearize* オプションを、または PLOP コマンドラインツールに *--linearize* オプションを与えます。
- ▶ Reader 有効化された文書：Reader 有効化されている PDF 文書を PLOP で処理すると、Reader 有効化されていない出力が生成されます。Reader 有効化された PDF を作れるのは Adobe ソフトウェアだけですので、これをどうにかする方法はありません。

**必要な一時ディスク容量** PLOP は入力 PDF 文書を読み込んで、出力 PDF を書き出します。出力文書は、おおよそ入力文書と同じディスク容量を必要とします（PLOP の最適化処理によって冗長な情報が削除されなければ）。多くの場合、これより多くのディスク容量が必要になることはありません。しかし PLOP/PLOP DS は、線形化か電子署名が有効にされているときには、その操作のために追加の一時ディスク容量を必要とします。

一時ファイルはデフォルトではカレントディレクトリに作成されますが、これは *PLOP\_create\_file()* の *tempdirname* オプションで変えることもできます。一時データのディ

スク容量は、おおよそ入力ファイルのサイズに等しくなります。線形化とインコア PDF 生成（すなわち出力ファイル名を与えない）をともに行うときは、PLOP は、おおよそ入力サイズの 2 倍の一時ディスク容量を必要とします。

**大容量 PDF 文書** 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。PLOP 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：

- ▶ 2 GB ファイルサイズ制限:PDF/A-1 などの規格では、ファイルサイズを 2 GB までに制限しています。一文書がこの制限よりも大きくなる場合には、PLOP は PDF/A-1・PDF/X-4・PDF/X-5 出力を生成しているときには例外を発生させます。それ以外の場合であれば 2 GB を超える文書を作成できます。
- ▶ 10 GB ファイルサイズ制限:PDF 文書内の相互参照テーブルは、10 進 10 桁すなわち  $10^{10}-1$  バイトまでに制限されています。これはおおよそ 9.3 GB にあたります。PLOP はこの制限を超える文書を作成することはできません。
- ▶ 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-1・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 個までに制限しています。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、PLOP は PDF/A-1・PDF/X-4・PDF/X-5 出力を生成しているときには例外を発生させます。それ以外の場合であればもっとオブジェクトの多い文書も作成できます。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 個程度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。

# 2 PLOP・PLOP DS コマンドラインツール

## 2.1 PLOP・PLOP DS コマンドラインオプション

PLOP と PLOP DS に一体化したコマンドラインツールを使うと、一切プログラミングを行う必要なしに、1 個ないし複数の PDF 文書に対して、暗号化・復号・最適化・修復・署名を行うことができます。さらに、これを使って PDF 文書の状態を取得することも可能です。PLOP のプログラムを、豊富なコマンドラインオプションで制御することができます。これは 1 個ないし複数の入力 PDF ファイルに対して、次のように呼び出されます（角カッコ内の項目はオプションです）：

```
plop --help
plop [ <一般のオプション群> ] --info [ --outfile <ファイル名> ] <ファイル名> ...
plop [ <一般のオプション群> ] <変換オプション群> --outfile <ファイル名> <ファイル名>
plop [ <一般のオプション群> ] <変換オプション群> --targetdir <パス名> <ファイル名>...
```

PLOP コマンドラインツールは、PLOP ライブラリに乗っかる形で作られています。デフォルトでは PLOP は、破損していることがわかった入力文書については修復を行い、また、出力のファイルサイズが最小になるよう最適化を行います。ライブラリのオプションを、7章のオプション一覧に従って、`--inputopt`・`---outputopt`・`---ploptopt` オプションを使って与えることができます。表 2.1 にすべての PLOP コマンドラインオプションを挙げます。

表 2.1 PLOP コマンドラインオプション一覧

オプション	引数	機能
--		オプション群のリストを終了します。これは、ファイル名が - キャラクターで始まる場合に有用です。
@filename'		オプション群を記述したレスポンスファイルを指定します。文法の詳細は、「レスポンスファイル」(30 ページ)を参照してください。レスポンスファイルは、-- オプションの前、かつ最初の filename の前でのみ認識され、また、他のオプションの引数を置き換えるために用いることはできません。

表 2.1 PLOP コマンドラインオプション一覧

オプション	引数	機能
<code>--compatibility, -c</code>	<バージョン>	<p>生成される PDF 出力文書の PDF バージョンを設定します：</p> <p><b>1.4</b> PDF 1.4。Acrobat 5 以上を必要とします。  <b>1.5</b> PDF 1.5。Acrobat 6 以上を必要とします。  <b>1.6</b> PDF 1.6。Acrobat 7 以上を必要とします。  <b>1.7</b> PDF 1.7。ISO 32000-1 で仕様化されており、Acrobat 8 以上を必要とします。  <b>1.7ext3</b> PDF 1.7 拡張レベル 3。Acrobat 9 以上を必要とします。  <b>1.7ext8</b> PDF 1.7 拡張レベル 8。Acrobat X を必要とします。  <b>2.0</b> PDF 2.0。ISO 32000-2 で仕様化されています。</p> <p>出力が暗号化されている場合には、これを用いて適切な暗号化アルゴリズムが選ばれます。この選ばれた PDF バージョンの対応する最も強い暗号化アルゴリズムが用いられます (AES 暗号化を強制するには 1.6 を指定します)。ここで選んだ PDF バージョンは、他のオプションによって、表 7.4 (81 ページ) で詳述する規則に従って、自動的に押し上げられることがあります：</p> <p>デフォルト：入力文書の PDF バージョン、またはそれより上の処理規則が義務づけるバージョン。</p>
<code>--fast, -f</code>		最適化ステップを無効にして、処理を高速化します。
<code>--help, -?</code> (またはオプションなし)		利用できるオプションをまとめたヘルプを表示します。
<code>--info, -i</code>		入力ファイルに対する状態情報を表示します。PDF 出力は生成されません。
<code>--inmemory</code>		入力ファイル (複数可) をメモリに読み込んで、その上で処理します。これはシステムによっては劇的なパフォーマンス向上をもたらします。
<code>--inputopt</code>	<オプションリスト>	<code>PLOP_open_document()</code> 用のオプションリストを指定できます (表 7.3 (78 ページ) 参照)
<code>--master<sup>2,3</sup>, -m</code>	<パスワード>	出力のマスターパスワード。オプションなしはパスワードなしを意味します。
<code>--noreplace, -n</code>		出力ファイルがすでに存在するときは、上書きされずに例外が発生します。デフォルト：出力ファイルがすでに存在していても上書きされません。
<code>--outfile, -o</code>	<ファイル名>	( <code>--info</code> の場合を除いて、ちょうど 1 個の入力文書が必須です。 <code>--outfile</code> と <code>--targetdir</code> のどちらか 1 つを与える必要があります) 出力ファイル名。入力と出力のファイル名は異なる必要があります。
<code>--outputopt</code>	<オプションリスト>	<code>PLOP_create_file()</code> 用のオプションリストを指定できます (表 7.4 (81 ページ) 参照)
<code>--password<sup>2</sup>, -p</code>	<パスワード>	入力文書 (複数可) のためのユーザーパスワードまたはマスターパスワード。このパスワードがすべての入力文書に対して使われます。必要なパスワードが入力文書によって異なるときは、それぞれ別個のプログラム呼び出しで処理する必要があります。

表 2.1 PLOP コマンドラインオプション一覧

オプション	引数	機能
<code>--permissions<sup>2,3</sup></code>	<権限群>	( <code>--master</code> が必須です) 出力文書に対するアクセス権限リスト。キーワード <code>noprint</code> ・ <code>nomodify</code> ・ <code>nocopy</code> ・ <code>noannots</code> ・ <code>noassemble</code> ・ <code>noforms</code> ・ <code>noaccessible</code> ・ <code>nohiresprint</code> ・ <code>plainmetadata</code> を任意の数含みます (表 4.3 (54 ページ) 参照)。この他に、次のキーワードも使えます (デフォルト: 権限制限なし): <b>keep</b> 入力文書の権限設定を引き継ぎます。入力文書から引き継いだ権限設定に変更を加えるために、この設定にキーワードを追加して修正条項とすることもできます。例: <code>keep noprint</code>
<code>--plopt</code>	<オプションリスト>	<code>PLOP_set_option()</code> 用のオプションリストを指定できます (表 7.7 (89 ページ) 参照)。これを使って、 <code>license</code> または <code>licensefile</code> オプションを渡すことが可能です。
<code>--resize, -R</code>	<ブロックサイズ>	(MVS のみ) 出力ファイルのレコードサイズ。デフォルト: 0 (非ブロック)
<code>--tempfilename, -T</code>	<ファイル名>	(MVS のみ) PLOP の内部処理に必要な一時ファイルのフルファイル名。空にすると、PLOP が一意な一時ファイル名を生成します。PLOP が完了した時にこの一時ファイルを削除するのはユーザー側の役割です。デフォルト: 空
<code>--tempdirname</code>	<ディレクトリ名>	PLOP の内部処理に必要な一時ファイルの作成されるディレクトリの名前。空にすると、PLOP は一時ファイルをカレントディレクトリに生成します。デフォルト: 空
<code>--searchpath, -s<sup>1</sup></code>	<パス>	(複数回指定可) ファイルの検索されるディレクトリの名前。このパスはマイナスキャラクタ「-」で始めてはいけません (その必要があるときは頭に ./ を付けます)。デフォルト: カレントディレクトリ
<code>--signopt, -S</code>	<オプションリスト>	(PLOP DS でのみ利用可能) 文書に電子的に署名するための、 <code>PLOP_create_file()</code> の <code>sign</code> オプション用のオプションリストを指定できます (表 7.6 (84 ページ) 参照)。
<code>--targetdir, -t</code>	<ディレクトリ名>	( <code>--outfile</code> と <code>--targetdir</code> のどちらか 1 つを与える必要があります) 出力ディレクトリ名。このディレクトリはすでに存在していなければなりません。
<code>--user, -u<sup>2,3</sup></code>	<パスワード>	出力のユーザーパスワード。オプションなしはパスワードなしを意味します。
<code>--verbose, -v</code>	0, 1, 2, 3	詳細度レベル (デフォルト: 1): 0 出力なし 1 エラーのみ 2 エラーとファイル名 3 詳細レポート
<code>--webopt, -w</code>		PDF 出力を Web 配信のために線形化します。線形化は他のさまざまな処理オプションと組み合わせることもできますし、単独で使うこともできます。デフォルト: 線形化しない

1. このオプションは複数回与えることもできます。
2. このオプションはすべての入力ファイルに対して用いられます。
3. このオプションを指定すると出力は暗号化されます。これらのうちのいずれか 1 つでも与えると、PLOP は出力を暗号化します。

**PLOP コマンドラインを組み立てる** PLOP コマンドラインを組み立てる際には、以下の規則を守る必要があります：

- ▶ 入力ファイルは、*searchpath* として指定されたすべてのディレクトリ内で検索されません。
- ▶ オプションによっては短縮形も利用でき、長いオプションと混ぜ書きも可能です。
- ▶ 長いオプションは省略もできますが、ただしその省略形は一意でなくてはなりません (例 `---plopt` のかわりに `--plop`)。
- ▶ 1 個のオプションを複数回書くと、最後のものだけが有効とされます。ただし、表 2.1 で複数回与えることもできると注記しているオプションについてはその限りではありません。
- ▶ 入力ファイルの暗号化状態によっては、処理のためにはユーザーパスワードかマスターパスワードが必要になります。これは `--password` オプションで与える必要があります。PLOP はこのパスワードが、要請されたアクションに対して十分なものを調べ (表 4.2 参照)、もしそうでないときは例外を発生させます。

PLOP は、まだどのファイルをも処理しない前に、コマンドライン全体を調べます。コマンドライン上のどの位置のオプションであろうと、その中にオプション文法誤りが見つかったときには、どのファイルも一切処理されません。いずれかのファイルを処理できないときは (必要なパスワードがない等の原因で)、エラーメッセージが作成されて、PLOP は残りのファイルの処理を続けます。

**ファイル名** ブランクキャラクタを含むファイル名は、PLOP のようなコマンドラインツールで用いる際には、ある特殊な取り扱いが必要です。ブランクキャラクタを含むファイル名を処理するためには、ファイル名全体をダブルクォートキャラクタで囲う必要があります。ワイルドカードは標準的な流儀に従って使用できます。たとえば `*.pdf` は、所与のディレクトリ内において、ファイル名接尾辞 `.pdf` を持ったすべてのファイルを表します。なお、システムによっては大文字と小文字は区別され、システムによってはされません (すなわち、`*.pdf` は `*.PDF` と別扱いになる場合があります)。また Windows システムではワイルドカードは、ブランクキャラクタを含むファイル名に対しては働かないことに注意してください。

Windows では、すべてのファイル名オプションは Unicode 文字列を受け付けます。たとえば Explorer からファイルをコマンドプロンプトウィンドウへドラッグした場合にはそうなります。

**レスポンスファイル** オプションは、コマンドラインで直接与える方法のほかに、レスポンスファイルで与える方法もあります。レスポンスファイルの内容は、コマンドラインの中で、`@filename` オプションが見つかった位置に挿入されます。

レスポンスファイルは、オプション群と引数群を記述したシンプルテキストファイルです。以下の文法規則に従う必要があります。

- ▶ オプションの複数の値は、空白系文字、すなわちスペース・ラインフィード・リターン・タブのいずれかで区切る必要があります。
- ▶ 空白系文字を含む値は、ダブルクォーテーションマーク「"」で囲う必要があります。
- ▶ 値の最初と最後のダブルクォーテーションマークは切り捨てられます。
- ▶ ダブルクォーテーションマークをリテラルに用いるためには、バックスラッシュでマスクして「\」とする必要があります。
- ▶ バックスラッシュキャラクタをリテラルに用いるためには、もう 1 個のバックスラッシュでマスクして「\\」とする必要があります。

レスポンスファイルは入れ子にすることもできます。すなわち、レスポンスファイルの中で **@filename** を用いて別のライセンスファイルを参照することも可能です。

レスポンスファイルは、ファイル名・パスワード引数に対して、Unicode 文字列を含むことが可能です。レスポンスファイルは UTF-8・EBCDIC-UTF-8・UTF-16 のいずれかの形式で符号化することができ、対応する BOM で始まっている必要があります。BOM が見つからないときは、レスポンスファイルの内容は、zSeries では EBCDIC として、それ以外の i5/iSeries を含むすべてのシステムでは ISO 8859-1 (Latin-1) として解釈されます。

**終了コード** PLOP コマンドラインツールは終了コードを返しますので、それを使えば、指示した操作が成功裏に実行されたかどうかを調べることができます：

- ▶ 終了コード 0：すべてのコマンドラインオプションと入力ファイルが、成功裏に、かつ完全に処理された。
- ▶ 終了コード 1：1 個ないし複数のファイル処理エラーが起きたが、処理は継続された。
- ▶ 終了コード 2：コマンドラインオプション内に何らかのエラーが見つかった。処理はその特定の悪いオプションの位置で停止し、どの文書も一切処理されていない。

## 2.2 PLOP・PLOP DS コマンドラインの作成例

以下の作成例は、PLOP コマンドラインオプションのいくつかの有用な組み合わせを示しています。すべてのサンプルは2つの形式で示しており、1番目ではすべてのオプションの長い形式を用いていますが、2番目では同等の短いオプション形式を用いています。この他にも、以下の節で作成例が得られます：

- ▶ 1章（さまざまな節）
- ▶ 4.3節「コマンドラインでPDF文書を保護」（56ページ）
- ▶ 5.3節「PLOP DSでPDF文書に署名」（63ページ）

カレントディレクトリ内のすべてのPDFファイルに関するセキュリティ等の情報を表示：

```
plop --info *.pdf
plop -i *.pdf
```

ディレクトリ内のすべてのPDF文書を線形化し（これらはどれもパスワードが不要と前提）、できたファイルをターゲットディレクトリ **output** へコピー。最適化はデフォルトで有効になっている（電子署名が同時に作成される場合を除く）ので、ファイルを線形化すると、同時にそのサイズは最適化されます。詳細度レベル2は、すべての入力・出力ファイルについて、その処理時に名前を印字します：

```
plop --verbose 2 --webopt --targetdir output *.pdf
plop -v 2 -w -t output *.pdf
```

カレントディレクトリ内のすべてのファイルを、同一のユーザーパスワード **demo** とマスターパスワード **DEMO** で暗号化し、できたファイルをターゲットディレクトリ **output** に置きます：

```
plop --targetdir output --user demo --master DEMO *.pdf
plop -t output -u demo -m DEMO *.pdf
```

ファイル **demo2048.p12** 中のデジタルIDを使って、PDF文書に不可視の署名を作成。デジタルIDに対するパスワードは、ファイル **pw.txt** に入っています：

```
plop --signopt "digitalid={filename=demo2048.p12} passwordfile=pw.txt" ←
    --outfile signed.pdf input.pdf
plop -S "digitalid={filename=demo2048.p12} passwordfile=pw.txt" -o signed.pdf input.pdf
```

# 3 PLOP・PLOP DS ライブラリの言語バインディング

この章では、PLOP/PLOP DS ライブラリの、各言語独特の諸側面を説明します。

## 3.1 C バインディング

PLOP は、C にいくつかの C++ モジュールを加えて記述されています。C バインディングを使用するには、静的または共有ライブラリ (DLL/SO) を使用することができ、中央 PLOP インクルードファイル *ploplib.h* を自分のクライアント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、*ploplibd.h* を用いて PLOP DLL を実行時に動的に読み込むこともできます (詳しくは次項を参照)。

**注記** PLOP の C バインディングを使用するアプリケーションは、C++ コンパイラでリンクを行う必要があります。このライブラリは C++ で実装されている部分をいくつか含んでいるからです。C リンカを使用すると、未解決の外部実体が生じる可能性があります。ただし、必要な C++ 対応ライブラリ群に対してアプリケーションが明示的にリンクされればこの限りではありません。

**PLOP を実行時に読み込まれる DLL として使用** 多くのクライアントでは PLOP を、静的結合ライブラリとして、またはリンク時に結合される動的ライブラリとして使用しますが、DLL を実行時に読み込んで、すべての API 関数へのポインタを動的に取得することも可能です。これは特に、必要時にのみ DLL を読み込むのに有用です。PLOP では、この動的使用を実現するための特殊な機構を用意しています。これは以下の規則に従って利用できます：

- ▶ *ploplib.h* でなく *ploplibd.h* をインクルードします。
- ▶ *PLOP\_new()*・*PLOP\_delete()* でなく *PLOP\_new\_dl()*・*PLOP\_delete\_dl()* を使用します。
- ▶ *PLOP\_TRY()*・*PLOP\_CATCH()* でなく *PLOP\_TRY\_DL()*・*PLOP\_CATCH\_DL()* を使用します。
- ▶ 他のすべての PLOP 呼び出しに対して関数ポインタを使用します。
- ▶ 追加モジュール *ploplibd.c* をコンパイルし、できたオブジェクトファイルに対して自分のアプリケーションをリンクします。

この動的読み込み機構は *encryptdl.c* サンプルで演示されています。

**注記** DLL を実行時に読み込めるのは、選ばれたプラットフォーム上のみです。

**例外処理** PLOP API では、ライブラリが発生させる例外に対処する機構を提供しています。これは、C 言語にはネイティブな例外処理がないことを補うためです。*PLOP\_TRY()*・*PLOP\_CATCH()* マクロを使用することによって、例外が発生したときにエラー処理とクリーンアップのための専用のコード群が呼び出されるようにクライアントコードを作ることができます。これらのマクロは 2 つのコードセクションを作ります：例外を発生させる可能性のあるコードを持った *try* 節と、例外に対処するコードを持った *catch* 節です。*try* ブロック内で呼び出された API 関数のいずれかが例外を発生させたときは、プログラムの実行は *catch* ブロックの先頭ステートメントへただちに引き継がれます。PLOP クライアントコード内で以下の規則を守る必要があります：

- ▶ *PLOP\_TRY()* と *PLOP\_CATCH()* は必ず対にする必要があります。

- ▶ `PLOP_new()` が例外を発生させることは一切ありません。try ブロックは有効な PLOP オブジェクトハンドルでのみ開始できますので、`PLOP_new()` への呼び出しはあらゆる try ブロックの外で行う必要があります。
- ▶ `PLOP_delete()` が例外を発生させることは一切ありませんので、try ブロックの外で呼び出しても安全です。catch 節の中で呼び出すこともできます。
- ▶ try ブロックと catch ブロックの両方で用いられる変数については特に注意が必要です。コンパイラは 1 個のブロックから別のブロックへの制御の遷移について知りませんので、この場合には不適切なコードが生成される可能性があります (レジスタ変数最適化など)。  
幸い、この種の問題を避けるための簡単な規則があります: try ブロックと catch ブロックの両方で用いられる変数は **volatile** 宣言する必要があります。 **volatile** キーワードを使用することで、コンパイラに対して、危険な最適化をこの変数に対して適用しないよう伝達することができます。
- ▶ try ブロックを去る場合には (return ステートメントなどによって、すなわち対応する `PLOP_CATCH()` への呼び出しをバイパスして)、例外機構に知らせるために、return ステートメントの前に `PLOP_EXIT_TRY()` を呼び出す必要があります。
- ▶ すべての PLOP 言語バインディングの場合と同様、例外が発生したときには文書処理は停止する必要があります。

以下のコードはこれらの規則を、クライアントコード内で PLOP 例外を扱う典型的なイデオムとともに演示しています (完全なサンプルが PLOP パッケージ内にあります) :

```
if ((plop = PLOP_new()) == (PLOP *) 0)
{
    printf("out of memory\n");
    return(2);
}
PLOP_TRY(plop)
{
    /* API関数群を直接または間接に呼び出すステートメント群 */
}
PLOP_CATCH(plop)
{
    printf("Error %d in %s() on page %d: %s\n",
        PLOP_get_errnum(plop), PLOP_get_apiname(plop),
        pageno, PLOP_get_errmsg(plop));
}
PLOP_delete(plop);
```

**名前文字列に対する Unicode の扱い** C 言語は Unicode にネイティブに対応していません。API 関数の文字列引数のなかには、**名前文字列**として宣言されているものもあります。これらは、**length** 引数の存在と、文字列の先頭の BOM の存在に従って扱われます。C では、**length** 引数が 0 でないときは、その文字列は UTF-16 として解釈されます。**length** 引数が 0 のときは、その文字列は、UTF-8 BOM で始まっていれば UTF-8 として、EBCDIC UTF-8 BOM で始まっていれば EBCDIC UTF-8 として、BOM が見つからなければ **host** エンコーディングとして (すべての EBCDIC ベースのプラットフォームでは **ebcdic** として) 解釈されます。

**オプションリストに対する Unicode の扱い** オプションリスト内の文字列には特に注意が必要です。UTF-16 形式の Unicode 文字列として表現することができず、バイト列としてのみ表現できるからです。この理由から、Unicode オプションに対しては UTF-8 が用い

られています。オプションの先頭に BOM を探すことによって、PLOP はそれをどのように解釈するかを決定します。この BOM を用いて文字列の形式が決定されます。より厳密には、文字列オプションの解釈は以下のように働きます：

- ▶ オプションが UTF-8 BOM (`\xEF\xBB\xBF`) で始まっていれば、それは UTF-8 として解釈されます。
- ▶ オプションが EBCDIC UTF-8 BOM (`\x57\x8B\xAB`) で始まっていれば、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、その文字列は *winansi* として (EBCDIC ベースのプラットフォームでは *ebcdic* として) 扱われます。

注記 `PLOP_convert_to_unicode()` ユーティリティ関数を使うと、UTF-16 文字列から UTF-8 文字列を生成することができます。これは Unicode 値を持つオプションリストを作成するのに有用です。

## 3.2 C++ バインディング

注記 C++ で書かれた .NET アプリケーションについては、C++ バインディングを通じてではなく、PLOP .NET DLL を直接利用することを推奨します（ただしクロスプラットフォームアプリケーションの場合には C++ バインディングを利用するべきです）。PLOP ディストリビューションに、この組み合わせを演示する .NET CLI とともに使用するための C++ サンプルコードがあります。

*ploplib.h* C ヘッダファイルに加えて、C++ 用のオブジェクト指向ラップが PLOP クライアントのために提供されています。これは *plop.hpp* ヘッダファイルを必要としており、このヘッダファイルは *ploplib.h* をインクルードしています。*plop.hpp* はテンプレートベースの実装となっていますので、対応する *plop.cpp* モジュールは不要です。C++ オブジェクトラップを利用することで、すべての PLOP 関数名に PLOP\_ 接頭辞が付いた API 関数による関数的アプローチを、よりオブジェクト指向のアプローチへ置き換えることができます。

**PLOP を実行時に読み込まれる DLL として使用** C 言語バインディングと同様、C++ バインディングでも、PLOP を自分のアプリケーションに実行時に動的に結合させることができます（「PLOP を実行時に読み込まれる DLL として使用」（33 ページ）を参照）。動的読み込みは、*plop.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下記のようにして有効にすることができます：

```
#define PLOPCPP_DL 1
```

これに加え、追加モジュール *ploplibdl.c* をコンパイルし、できたオブジェクトファイルに対して自分のアプリケーションをリンクする必要があります。動的読み込みの詳細は PLOP オブジェクト内に隠されていますので、それは C++ API に影響を与えません：動的読み込みが有効にあってなくても、すべてのメソッド呼び出しは同じに見えます。

注記 DLL を実行時に読み込めるのは、選ばれたプラットフォーム上のみです。

**C++ の文字列処理** PLOP 4.1 では、新しい Unicode 対応の C++ バインディングを導入しています。新しいテンプレートベースのアプローチで、文字列処理に関して以下の使用パターンが可能です：

- ▶ C++ 標準ライブラリ型 *std::wstring* の文字列が基本文字列型として用いられます。これは、UTF-16 または UTF-32 で符号化された Unicode キャラクターを持つことができます。これは PLOP 4.1 のデフォルト動作であり、カスタムデータ型（次項参照）が *wstring* に対して大きな利点を持たない限り、新しいアプリケーションに対する推奨アプローチです。
- ▶ 文字列処理のためのカスタム（ユーザー定義）データ型を、そのカスタムデータ型が *basic\_string* クラステンプレートのインスタンス化であり、かつユーザーが与える変換メソッドによって Unicode との相互変換が可能である限り、用いることができます。
- ▶ プレーン C++ 文字列を、PLOP 4.0 までのバージョンに対して開発された既存の C++ アプリケーションとの互換性のために用いることができます。この互換方式は、既存アプリケーションのためだけに用意されています（ソースコード互換性について下記注記を参照）。

新しいインタフェースは、PLOP メソッドとやりとりされるすべての文字列がネイティブ *wstring* であると見なします。*wchar\_t* データ型のサイズによって、*wstring* は UTF-16 で（2 バイトキャラクタ群）、または UTF-32 で（4 バイトキャラクタ群）符号化された Unicode

文字列を内容として持つと見なされます。ソースコード内のリテラル文字列は、ワイド文字であることを示すために先頭に `L` を付ける必要があります。リテラル内で Unicode キャラクタは `\u・\U` 文法で作成できます。この文法は標準 ISO C++ に含まれているのですが、コンパイラによってはこれに対応していないものがあります。その場合にはリテラル Unicode キャラクタは 16 進キャラクタで作成する必要があります。

**アプリケーションを新しい C++ バインディングに合わせて変更** PLOP 4.0 までのバージョンに対して開発された既存の C++ アプリケーションは、以下のようにして PLOP 4.1 に合わせて変更することができます：

- ▶ TET C++ クラスは `pdflib` 名前空間内に入りましたので、クラス名を修飾する必要があります。いちいち `pdflib::PLOP` を書くことを避けるため、クライアントアプリケーションは PLOP メソッドを使う前に下記を追加する必要があります：

```
using namespace pdflib;
```

- ▶ アプリケーションの文字列処理を `wstring` へ切り替えます。これは外部情報源からのデータについても当てはまります。しかし、ソースコード内の文字列リテラル（オプションリストも）は、`L` 接頭辞を頭に付ける必要があります。たとえば

```
const wstring docoptlist = L"password=foo";
```

- ▶ PLOP のエラーメッセージと例外文字列 (`PLOP::Exception` クラス内の `get_errmsg()` メソッド) を処理するには、適切な `wstring` 対応メソッド (`wcerr` 等) を用いる必要があります。
- ▶ PLOP C++ バインディングで `plop.cpp` モジュールは必要なくなりました。PLOP ディストリビューションはこのモジュールのダミー実装を含んでいますが、これは PLOP アプリケーションのビルド処理からは除くべきです。

**レガシアプリケーションとの完全ソースコード互換性** 新しい C++ バインディングはアプリケーションレベルのソースコード互換性を志向して設計されていますが、クライアントアプリケーションは再コンパイルする必要があります。レガシアプリケーションに対して完全なソースコード互換性を実現するには以下の方法が用意されています：

- ▶ `plop.hpp` をインクルードする前に `wstring` ベースのインタフェースを下記のように無効化します：

```
#define PLOPCPP_PLOP_WSTRING 0
```

- ▶ `plop.hpp` をインクルードする前に `pdflib` 名前空間を下記のように無効化します：

```
#define PLOPCPP_USE_PDFLIB_NAMESPACE 0
```

**C++ のエラー処理** PLOP API 関数は、エラー発生時には C++ 例外を発生させます。これら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する必要があります。さらなるエラー情報を提供するために、PLOP クラスはパブリックな `PLOP::Exception` クラスを提供しており、このクラスは、詳細なエラーメッセージ、例外番号、例外を発生させた PLOP API 関数の名前を取得するためのメソッドを公開しています。

PLOP ルーチンが発生させたネイティブな C++ 例外は期待どおりに動作します。以下のコードは、PLOP が発生させた例外をキャッチします：

```
try {  
    ...さまざまなPLOP命令...  
} catch (PLOP::Exception &ex) {
```

```
wcerr << L"Error " << ex.get_errnum()  
<< L" in " << ex.get_apiname()  
<< L"(): " << ex.get_errmsg() << endl;  
}
```

## 3.3 COM バインディング

**PLOP の COM 版をインストール** PLOP/PLOP DS を、提供されている Windows インストーラでインストールします。このインストーラは適切なレジストリ項目を作成し、PLOP コンポーネントを Windows に登録して、任意の COM 互換のプログラムからそれを使えるようにします。

**COM での例外処理** PLOP/PLOP DS コンポーネントは、COM の標準的な例外動作を実装しており、説明メッセージとともに COM 例外を発生させます。PLOP の利用者は、標準的なプログラミング手段を用いてこの例外をとらえ、それに対処することができます。

**PLOP の COM 版を .NET で使用** PLOP の COM 版は .NET で、PLOP.NET (3.5 節「.NET バインディング」(42 ページ) 参照) のかわりに使うことも可能です。まず、*tlbimp.exe* ユーティリティを使って、PLOP の COM 版から .NET のアセンブリを作成する必要があります：

```
tlbimp plop_com.dll /namespace:plop_com /out:Interop.plop_com.dll
```

このアセンブリを、自分の .NET アプリケーションの中で使うことができます。Visual Studio .NET の中で *plop\_com.dll* への参照を追加すると、アセンブリが自動的に作成されます。

以下の抜粋コードでは、PLOP の COM 版を VB.NET で使う方法を示しています：

```
Imports plop_com
...
Dim p As plop_com.IPDF
...
p = New PLOP()
...
buf = p.get_buffer()
```

以下の抜粋コードでは、PLOP の COM 版を C# で使う方法を示しています：

```
using plop_com;
...
static plop_com.IPDF p;
...
p = New PLOP();
...
buf = (byte[])p.get_buffer();
```

この後のコードは、PLOP の .NET 版と同様に書くことができます。C# では、*get\_buffer()* の戻り値をキャストする必要があることに注意してください。なぜなら、ここで COM オブジェクトから返される VARIANT データ型からは、自動変換がないためです。

## 3.4 Java バインディング

**PLOP の Java 版をインストール** PLOP/PLOP DSはネイティブなCライブラリとして実装されており、JavaにはJNI (Java Native Interface) を通じてアタッチします。当然、Javaアプリケーションを開発するには、JNIへの対応を含んだJDKが必要です。PLOPバインディングが動作するためには、PLOPのJavaラップライブラリとPLOPのJavaパッケージが、Java VMから見えている必要があります。

**PLOP の Java パッケージ** Javaの開発者にとって整合性のあるルックアンドフィールを保つため、PLOPは次のパッケージ名を持ったJavaパッケージとして構成されています：

```
com.pdflib.plop
```

このパッケージは *plop.jar* ファイルの中であり、*plop* という1個のクラスを含んでいます。PLOPをさまざまなJava開発環境で使用するためにあたっての最新情報が、*readme.txt* ファイル内にあるかもしれません。

このパッケージを自分のアプリケーションに与えるには、自分の **CLASSPATH** 環境変数に *plop.jar* を追加するか、またはJavaコンパイラ・ランタイムへの呼び出しに **-classpath plop.jar** オプションを追加するか、ないしはそれと同等の手順をJava IDE内で踏む必要があります。Java VMの設定として、**java.library.path** プロパティにディレクトリの名前を設定すれば、そのディレクトリでネイティブライブラリが検索されるようにすることができます。たとえば

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

このプロパティの値は次のようにして知ることができます：

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

このほかに、以下のようなプラットフォーム独自の手順を踏む必要があります：

- ▶ Unix：ライブラリ *libplop\_java.so* を、共有ライブラリのためのデフォルトの場所のうちのいずれか1つに、または適切に設定されたディレクトリに置く必要があります。
- ▶ Mac OS X：ライブラリ *libplop\_java.jnilib* を、共有ライブラリのためのデフォルトの場所のうちのいずれか1つに、または適切に設定されたディレクトリに置く必要があります。
- ▶ Windows：ライブラリ *plop\_java.dll* を、Windowsのシステムディレクトリに、またはPATH環境変数に示されているディレクトリに置く必要があります。

**PLOP サブレットと Java アプリケーションサーバ** PLOP/PLOP DSは、サーバサイドのJavaアプリケーションに、とりわけサブレットに完全に適合しています。特定のサブレットエンジンでPLOPを使用する際には、以下の設定上のきまりを守る必要があります：

- ▶ サブレットエンジンがネイティブライブラリを探すディレクトリは、ベンダによって異なります。よくある候補としては、システムディレクトリや、背後のJava VMに特有のディレクトリ、サブレットエンジンのローカルディレクトリが挙げられます。自分のサブレットエンジンのベンダから提供されている説明書を見てください。
- ▶ サブレットをロードするのが特別なクラスローダで、制限されていたり、専用のクラスパスを使用していたりすることはよくあります。サブレットエンジンによって

は、特別なエンジンクラスパスを定義して、PLOP パッケージが確実に見つかるようにする必要があります。

PLOP ディストリビューションには、PLOP をサブレット内で使用している例が入っています。

**Java での例外処理** PLOP/PLOP DS のメソッドはすべて、エラー時には *PLOPException* 型の例外を発生させます。PLOP の利用者は、標準的な Java 言語の機能を使ってその例外をキャッチし、それに対処することができます。

```
try {
    plop plop;
    /* ... さまざまなPLOP命令 ... */
} catch (PLOPException e) {
    System.err.println("暗号化: PLOP例外が発生しました:");
    System.err.println(e.get_apiname() + ": " + e.getMessage());
} finally {
    /* PLOPオブジェクトを削除 */
    if (plop != null) plop.delete();
}
```

## 3.5 .NET バインディング

注記 PLOP を .NET Framework とともに使用するためのさまざまな種類とオプションに関する詳しい情報が、PDFlib-in-.NET-HowTo.pdf 文書にあります。この文書は、ディストリビューションパッケージにあるほか、PDFlib Web サイトにもあります。

PLOP の .NET 版は、.NET に関連するすべての概念に対応しています。技術的用語でいうならば、PLOP.NET 版は、.NET フレームワークの制御下で走る C++ クラスです (非マネージの PLOP コアライブラリにマネージャラップを付けたもの)。厳密名を持つ静的ライブラリとしてパッケージされています。PLOP アセンブリ (*PLOP\_dotnet.dll*) には、ライブラリ本体に加えてメタ情報が含まれています。

**PLOP の .NET 版をインストール** PLOP を、提供されている Windows MSI インストーラでインストールします。PLOP.NET MSI インストーラは、PLOP アセンブリと追加データファイル群・説明書・サンプルを、マシン上に対話的にインストールします。このインストーラは PLOP の登録も行なって、Visual Studio .NET の「参照の追加」ダイアログボックスの .NET タブで簡単に参照できるようにします。

**.NET でのエラー処理** PLOP.NET は .NET の例外に対応しており、実行時の問題が発生したときには、詳細なエラーメッセージのついた例外を発生させます。このような例外をキャッチして、それに対して適切に対処するのは、クライアント側の役割です。それをしないと、.NET フレームワークがその例外をキャッチして、通常はアプリケーションを中絶させます。

例外関連の情報を伝達するために、PLOP ではそれ自身の例外クラス *PLOP\_dotnet.PLOPException* を定義しており、メンバ *get\_errnum*・*get\_errmsg*・*get\_apiname* を持たせています。

**PLOP を C++・CLI とともに使用** C++ で書かれた .NET アプリケーション (共通言語基盤 = CLI に基づく) は、PLOP C++ バインディングを使用せずに直接 PLOP.NET DLL を利用することができます。そのソースコードは下記のように PLOP を参照する必要があります:

```
using namespace PLOP_dotnet;
```

## 3.6 Perl バインディング

Perl 用 PLOP ラップは、1 個の C ラップと 2 個の Perl パッケージモジュールから成ります。このモジュールの 1 個は各 PLOP API 関数と同等のものを Perl で提供するもので、もう 1 個は PLOP オブジェクトのためのものです。C モジュールは、Perl インタプリタが実行時に読み込む共有ライブラリを、パッケージファイルからいくらかの助けを借りてビルドするために用いられます。Perl スクリプトは共有ライブラリモジュールを、*use* ステートメントを通じて参照します。

**PLOP の Perl 版をインストール** Perl 拡張機構は共有ライブラリを実行時に、DynaLoader モジュールを通じて読み込みます。Perl 実行形式が、共有ライブラリに対応した形でコンパイルされている必要があります (多くの Perl 設定ではそのようになっています)。

PLOP バインディングが動作するためには、Perl インタプリタは PLOP Perl ラップとモジュール *plop\_pl.pm*・*PDFlib/PLOP.pm* を利用可能である必要があります。以下に説明するプラットフォーム固有の方式のほかに、Perl の *@INC* モジュール検索パスに、*-I* コマンドラインオプションを用いてディレクトリを追加することも可能です：

```
perl -I/path/to/plop encrypt.pl
```

**Unix** Perl は、*plop\_pl.so* (Mac OS X では *plop\_pl.bundle*)・*plop\_pl.pm*・*PDFlib/PLOP.pm* を、カレントディレクトリ内で、あるいは下記 Perl コマンドで印字されるディレクトリ内で検索します：

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

Perl はサブディレクトリ *auto/plop\_pl* も検索します。上記コマンドの典型的出力は下記ようになります：

```
/usr/lib/perl5/site_perl/5.10/i686-linux
```

**Windows** PLOP は、Perl 5 の Windows に対する ActiveState ポートにも対応しています。これは ActivePerl とも呼ばれます。DLL *plop\_pl.dll* とモジュール *plop\_pl.pm*・*PDFlib/PLOP.pm* が、カレントディレクトリ内で、あるいは下記 Perl コマンドで印字されるディレクトリ内で検索されます：

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

上記コマンドの典型的出力は下記ようになります：

```
C:\Program Files\Perl5.10\site\lib
```

**Perl での例外処理** PLOP の例外が発生すると、Perl の例外が発生します。これは以下のように、*eval* シーケンスを用いて捕捉・対処できます：

```
eval {  
    ...さまざまなPLOP命令...  
};  
die "例外をキャッチしました: $@" if $@;
```

## 3.7 PHP バインディング

**PLOP の PHP 版をインストール** PLOP/PLOP DS は、PHP へ動的にアタッチできる C ライブラリとして実装されています。PLOP は PHP のいくつかのバージョンに対応しています。アンパックした PLOP アーカイブの中から、自分が使う PHP のバージョンに合わせて、適切な PLOP ライブラリを選ぶ必要があります。

PLOP を PHP で使う際のさまざまな種別やオプションに関する詳しい情報は、PHP 用のロード可能な PLOP モジュールを使用すべきかどうかという問いも含めて、PDFlib の Web サイトにある *PDFlib-in-PHP-HowTo* 文書に掲載しています。これは主に PDFlib を PHP で使う際のことを述べていますが、その説明は PLOP を PHP で使う場合についても同様に当てはまります。

PHP を設定して、外部の PLOP ライブラリについてわからせる必要があります。以下の 2 つの選択肢があります：

- ▶ *php.ini* に以下のいずれかの行を追加：

```
extension=plop_php.so          ; Unix・Mac OS X 用
extension=plop_php.dll         ; Windows 用
```

PHP はこのライブラリを、Unix の場合は *php.ini* 内の *extension\_dir* 変数で指定されているディレクトリで検索し、Windows の場合はそのほかに標準のシステムディレクトリ群でも検索します。どのバージョンの PLOP の PHP バインディングをインストールしてあるかは、下記の 1 行の PHP スクリプトで調べることができます：

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

これは、自分の現在の PHP 設定に関する長い情報ページを表示します。このページで、*plop* と題されたセクションを調べます。もしこのセクションに下記

```
PDFlib PLOP (PDF Linearization, Optimization, Protection) => enabled
```

(この後に PLOP のバージョン番号)があれば、PLOP の PHP 版を正しくインストールできています。

- ▶ 自分のスクリプトの先頭に、以下のいずれかの行を書いて、PLOP を動作時にロードする：

```
dl("plop_php.so");           # Unix・Mac OS X 用
dl("plop_php.dll");          # Windows 用
```

**PHP でのファイル名処理** PDF や画像・フォント等のディスクファイルに対する無修飾のファイル名（パス要素のない）と相対ファイル名は、PHP の Unix 版と Windows 版とは、扱われ方が異なります：

- ▶ Unix 諸システムの PHP の場合、パス要素を持たないファイルは、スクリプトが置かれているディレクトリで検索されます。
- ▶ Windows の PHP の場合、パス要素を持たないファイルは、PHP DLL が置かれているディレクトリでのみ検索されます。

**PHP 5 での例外処理** PHP 5 では、構造化された例外処理に対応しているので、PLOP の例外は PHP の例外として伝達されます。標準的な *try/catch* 技法を使って PLOP 例外を取り扱えます：

```
try {
```

...さまざまなPLOP命令...

```
} catch (PLOPException $e) {  
    print "PLOP例外が発生しました:\n";  
    print "[" . $e->get_errnum() . "]" . $e->get_apiname() . ": "  
        $e->get_errmsg() . "\n";  
}  
catch (Exception $e) {  
    print $e;  
}
```

## 3.8 Python バインディング

**PLOP の Python 版をインストール** Python の拡張機構は、実行時に共有ライブラリを読み込むことによって動作します。PLOP バインディングが動作するためには、Python インタプリタが PLOP Python ラッパを利用可能である必要があります。このラッパは、PYTHONPATH 環境変数内に挙げられているディレクトリ群の中で検索されます。Python ラッパの名前はプラットフォームによって異なります：

- ▶ Unix・Mac OS X : *plop\_py.so*
- ▶ Windows : *plop\_py.pyd*

**Python のエラー処理** Python バインディングは、PLOP エラーをネイティブな Python 例外へ翻訳する特殊なエラーハンドラをインストールします。この Python 例外は、通常の try/catch 技法で扱えます：

```
try:
    ...さまざまなPLOP命令...
except PLOPException:
    print 'PLOP例外をキャッチしました!'
```

## 3.9 RPG バインディング

PLOP/PLOP DS では、PLOP の関数を埋め込んだ ILE-RPG の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を定義した */copy* モジュールを提供しています。

**Unicode 文字列の扱い** PLOP の関数はすべて、引数として可変長の Unicode 文字列をとりますので、**%UCS2** ビルトイン関数を用い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。PLOP 関数が返す文字列はすべて可変長の Unicode 文字列です。**%CHAR** ビルトイン関数を用いて、これらの Unicode 文字列をシングルバイト文字列に変換してください。

**注記** **%CHAR** 関数と **%UCS2** 関数は、文字列を Unicode から、または Unicode へ変換する際に、カレントジョブの CCSID を用います。作成例は、CCSID 37 (US EBCDIC) に基づいています。オプションリスト内の特別なキャラクタのうちのいくつかは ([[]] 等)、別のコードページの下でこれらの作成例を動作させると、正しく変換されない可能性があります。

文字列はすべて可変長文字列として渡されますので、さまざまな関数で、文字列長を明示的に示す *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)。

**PLOP の RPG プログラムをコンパイル・バインドする** PLOP 関数を RPG から使用するには、コンパイルされた PLOPLIB サービスプログラムが必要です。PLOP 定義をコンパイル時にインクルードするためには、次のように、自分の ILE-RPG プログラムの D スペック内でその名前を指定する必要があります。

```
d/copy QRPGLSRC,PLOPLIB
```

PLOP ソースファイルライブラリが自分のライブラリリスト上にないときは、次のように、そのライブラリも指定する必要があります。

```
d/copy ploplib/QRPGLSRC,PLOPLIB
```

自分の ILE-RPG プログラムのコンパイルを開始する前には、PLOP に同梱している PLOPLIB サービスプログラムの入ったバインディングディレクトリを作成しておく必要があります。下記の例では、ライブラリ PLOPLIB 内に PLOPLIB というバインディングディレクトリを作成したいものとします。

```
CRTBNDDIR BNDDIR(PLOPLIB/PLOPLIB) TEXT('PLOPLIB Binding Directory')
```

バインディングディレクトリを作成した後は、PLOPLIB サービスプログラムを自分のバインディングディレクトリに追加する必要があります。下記の例では、さきに作成したバインディングディレクトリにライブラリ PLOPLIB 内のサービスプログラム PLOPLIB を追加したいものとします。

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

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

```
CRTBNDRPG PGM(PLOPLIB/ENCRYPT) SRCFILE(PLOPLIB/QRPGLSRC) SRCMBR(*PGM) DFTACTGRP(*NO) BNDDIR(PLOPLIB/PLOPLIB)
```

**RPG でのエラー処理** ILE-RPG で書かれた PLOP クライアントは、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。例外を見張るもう 1 つの方法は、ILE-RPG 内の *\*PSSR* グローバルエラー処理サブルーチンを用いることです。例外が発生した際には、ジョブログは、エラー番号、失敗した関数、例外の理由を示します。PLOP 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=PLOP_new
*
c    monitor
*
c    eval      doc=PLOP_open_document(p:%ucs2('/tmp/my.pdf'):inputoptlist)
:
:
*    Error Handling
c    on-error
*    Do something with this error
*    don't forget to free the PLOP object
c    callp     PLOP_delete(p)
c    endmon
```

# 4 PDF セキュリティ

## 4.1 さまざまな PDF セキュリティ機能

PDF 文書はパスワードセキュリティで保護することができます。これ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る）を与えないとファイルを開いて閲覧できないようにする。
- ▶ マスターパスワード（所有者パスワードまたは権限パスワードとも呼ばれる）を与えないと、セキュリティ設定、すなわち諸権限・ユーザーパスワード・マスターパスワードを一切変更できないようにする。ユーザーパスワードとマスターパスワードを持つファイルは、そのどちらかのパスワードを与えれば開いて閲覧できます。
- ▶ 権限設定は、その PDF 文書に対する特定の動作（印刷やテキスト抽出等）を制限する。
- ▶ 添付パスワードを指定すると、文書自体の内容本体は暗号化せず、ファイル添付のみを暗号化することができます。

これらの保護機能のうち 1 つでも用いている PDF 文書は暗号化されます。文書のセキュリティ設定を Acrobat で表示または変更するには、それぞれ「ファイル」→「文書のプロ

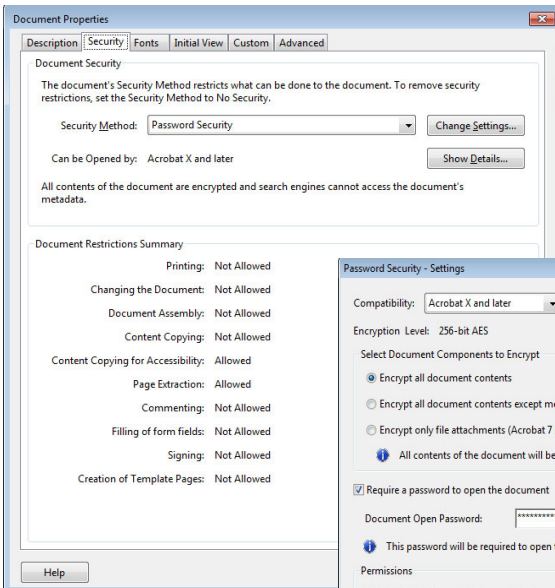
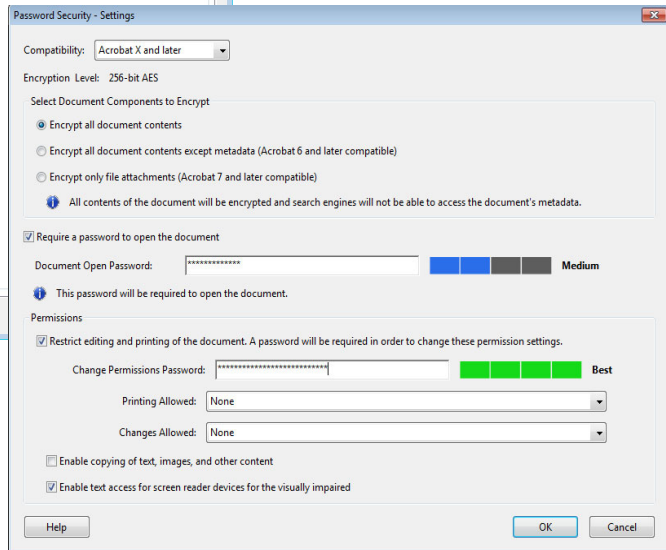


図 4.1 Acrobat で標準セキュリティ設定を表示（左）・設定（下）



「パティ...」→「セキュリティ」→「詳細を表示...」か「設定を変更...」をクリックします。  
 図 4.1 に Acrobat のセキュリティ設定ダイアログを示します。

**暗号化アルゴリズムとキー長** PDF の暗号化は、以下の暗号化アルゴリズムを利用しています：

- ▶ RC4。対称ストリーム暗号です（すなわち、同じアルゴリズムを用いて暗号化と復号ができます）。RC4 はプロプライエタリなアルゴリズムです。
- ▶ AES（高度暗号化標準）。規格 FIPS-197 で仕様化されています。AES はさまざまな応用で利用されている最新のブロック暗号です。

実際の暗号化キーは扱いにくいバイナリ列なので、それはもっとユーザーフレンドリーなプレーンキャラクタから成るパスワードから導出されます。PDF と Acrobat の発展の過程のなかで、PDF 暗号化方式は改良を重ねられ、より強力なアルゴリズム、より長い暗号化キー、より洗練されたパスワードを用いるようになってきています。表 4.1 に、すべての PDF バージョンについて、暗号化キーとパスワードの特徴を示します。

表 4.1 PDF の各バージョンにおける暗号化アルゴリズム・キー長・パスワード

PDF・Acrobat バージョン、 pCOS アルゴリズム番号	暗号化アルゴリズムとキー長	最大パスワード長とパスワードエンコーディング
PDF 1.1 ~ 1.3 (Acrobat 2 ~ 4)、 アルゴリズム 1	RC4 40 ビット（弱い。使用するべきではありません）	32 キャラクタ（Latin-1）
PDF 1.4 (Acrobat 5)、 アルゴリズム 2	RC4 128 ビット	32 キャラクタ（Latin-1）
PDF 1.5 (Acrobat 6)、 アルゴリズム 3	PDF 1.4 と同じ、ただし暗号化方式の異なる 応用	32 キャラクタ（Latin-1）
PDF 1.6 (Acrobat 7)・ PDF 1.7 = ISO 32000-1 (Acrobat 8)、 アルゴリズム 4	AES-128	32 キャラクタ（Latin-1）
PDF 1.7ext3 (Acrobat 9)、 アルゴリズム 9	AES-256 で、パスワードの扱いに脆弱性があるもの	127 UTF-8 バイト （Unicode）
PDF 1.7ext8 (Acrobat X)・ PDF 2.0 = ISO 32000-2、 アルゴリズム 11	AES-256 で、パスワードの扱いが改良されたもの	127 UTF-8 バイト （Unicode）

PDF の暗号化では、ユーザーまたはマスターパスワードを直接使って文書内容を暗号化するのではなく、パスワードと権限設定などを含む他の諸パラメータとから暗号化キーを算出しています。実際に文書を暗号化するのに用いられる暗号化キーの長さは、パスワードの長さからは独立です（表 4.1 参照）。

**パスワード** PDF の暗号化は内部的に、PDF バージョンによって 40・120・250 ビットのいずれかの暗号化キーで動作します。ユーザーが与えたパスワードからバイナリ暗号化キーが導出されます。パスワードには長さやエンコーディングの制約があります：

- ▶ PDF 1.7 (ISO 32000-1) までは、パスワードは最大長 32 キャラクタに限られ、Latin-1 エンコーディング内のキャラクタのみを含むことができます。
- ▶ PDF 1.7ext3 では Unicode キャラクタを導入し、最大長を、パスワードの UTF-8 表現で 127 バイトに押し上げました。UTF-8 ではキャラクタを可変長 1 ~ 4 バイトに符号化しますので、パスワード内に許される Unicode キャラクタの数は、非 ASCII キャラクタを含む場合には 127 より少なくなります。たとえば、日本語キャラクタは UTF-8 表

現では通常 3 バイトを必要としますので、パスワード内で最大 42 個の日本語キャラクタまでが使えることになります。

あいまいさを避けるために、Unicode パスワードは SASLprep という処理 (RFC 3454 の Stringprep に基づき RFC 4013 で仕様化されています) によって正規化されます。この処理では、非テキストキャラクタを除去し、ある種のキャラクタクラスを正規化します (たとえば非 ASCII 空白キャラクタは ASCII 空白キャラクタ U+0020 へマップされます)。パスワードは Unicode 正規形 KC へ正規化され、パスワード内に右書きキャラクタと左書きキャラクタが混在していた場合に起こりうるあいまいさを回避するために特殊な双方向処理が施されます。

PDF 暗号化の強度は、暗号化キーの長さによってのみ決まるのではなく、パスワードの長さや質によっても左右されます。名前や単語そのままなどをパスワードに使うべきではないということは広く知られています。容易に推測できたり、いわゆる辞書アタックによってシステムティックにあたられるからです。さまざまな調査によれば、かなりの数のパスワードは配偶者やペットの名前、ユーザーの誕生日、子供のニックネームなどを用いており、そのため容易に推測可能になっています。

**権限設定** PDF では、文書の操作に関するさまざまな制限を符号化することができ、これらは個別に承認または拒否することができます (ただし設定によっては互いに依存しあうものもあります) :

- ▶ **印刷**:印刷が許可されていないときは、Acrobat の印刷ボタンは無効になります。Acrobat は、高解像度印刷と低解像度印刷の区別に対応しています。低解像度印刷では、個人的な利用にしか適さないような、そのページのビットマップ画像が生成されますが、高品位印刷と再 PDF 化はできません。ビットマップ印刷では出力品質が低くなるだけでなく、印刷処理がかなり遅くなることにも留意してください。
- ▶ **編集一般**: これを無効にされると、文書への変更は一切禁止されます。内容の抽出と印刷は許されます。
- ▶ **内容のコピーと抽出**: これを無効にされると、文書の内容を選択してクリップボードへそれをコピーして内容を再利用することが禁止されます。アクセシビリティインタフェースも無効になります。このような文書を Acrobat で検索する必要があるときは、Acrobat の環境設定で「承認済みプラグインのみ」を選択する必要があります。
- ▶ **コメントとフォームフィールドの作成**: これを無効にされると、コメントとフォームフィールドの追加・変更・削除が禁止されます。フォームフィールドへの記入は許されます。
- ▶ **フォームフィールドへの記入または署名**: これを有効にされると、ユーザーはフォームに署名や記入はできますが、フォームフィールドを作成することはできません。
- ▶ **内容アクセシビリティを有効にする**: その文書の内容をアクセシビリティソフトウェア (読み上げソフトウェア等) が利用することを許します。この設定は PDF 2.0 では非推奨として宣言されています: アクセシビリティ目的での内容抽出は**内容のコピーと抽出**設定に基づきます。
- ▶ **文書の取りまとめ**: これを無効にされると、ページの挿入・削除・回転およびしおり・サムネールの作成が禁止されます。

印刷の禁止など、何らかのアクセス制限を設定すると、Acrobat のそれに対応するメニュー項目が無効になります。しかし、これはサードパーティの PDF ビューアなどのソフトウェアでもそうなるとは限りません。文書内のアクセス権限が実際に効力を持つかどうかは、PDF ツールの開発者にかかっているのです。実際、いくつかの PDF ツールは権限設定を全然無視することで知られています: 商用の PDF クラッキングツールを使えば、いかなるアクセス制限も無効化することができます。これは暗号化のクラッキングとは関係あり

ません：パスワードのない PDF ファイルを、画面では見られても印刷はできないようにすることは、単に不可能なのです。このことは ISO 32000-1 に下記のように記されていません：

「ひとたび文書が成功裡に開かれ復号されれば、準拠リーダは技術的にその文書の内容全体にアクセス可能となる。暗号化辞書内で指定されている文書権限設定群を強制できる性質のものは PDF 暗号化の中に何もない。」

**暗号化された文書構成要素** デフォルトでは、PDF 暗号化はつねに 1 個の文書のすべての構成要素をカバーします。しかし、場合によっては、文書内のいくつかの構成要素は暗号化せず、それ以外だけを暗号化したいときもあります：

- ▶ PDF 1.5 (Acrobat 6) では、プレーンテキストメタデータという機能が導入されました。この機能を使うと、暗号化された文書に、暗号化されていないメタデータを入れることができます。これによって、検索エンジンが文書のメタデータを、暗号化された文書からでも取り出せるようにすることができます。
- ▶ PDF 1.6 (Acrobat 7) からは、保護されていない文書の中のファイル添付であっても、暗号化することが可能です。これによって、暗号化されていない文書を、秘密の添付のためのコンテナとして利用することができます。

**セキュリティ推奨項目** 以下のことは、できる暗号化が弱くてクラックされる可能性がありますので、避けるべきです：

- ▶ 1～6 キャラクタから成るパスワードは避けるべきです。可能なすべてのパスワードを試す攻撃（パスワードに対するブルートフォースアタック）に対して弱いからです。
- ▶ パスワードは単なる単語に似てはいけません。可能な単語をすべて試す攻撃（辞書アタック）に対して弱いからです。パスワードには非アルファベットキャラクタを含ませるべきです。自分の配偶者やペットの名前、誕生日、その他簡単に推測できる項目を使ってはいけません。
- ▶ PDF 1.6 (Acrobat 4) までの 40 ビット RC4 アルゴリズムは避けるべきです。可能なすべてのキーを試す攻撃（暗号化キーにたい売るブルートフォースアタック）に対して弱いからです。
- ▶ 最新の AES アルゴリズムのほうが、古い RC4 アルゴリズムよりも望ましいです。
- ▶ PDF 1.7ext3 (Acrobat 9) に従った AES-256 は避けるべきです。パスワードチェックアルゴリズムに脆弱性を含んでいることから、パスワードに対するブルートフォースアタックが容易なためです。このため、Acrobat X と PLOP 4.1 では、新しい文書を保護するために Acrobat 9 の暗号化は決して使用しません（既存の文書を復号するためにのみ使用します）。

まとめると、PDF 1.7ext8/PDF 2.0 に従った AES-256 か PDF 1.6/1.7 に従った AES-128 を使用するべきです。どちらを使うかは Acrobat X が利用可能かどうかによって決まります。パスワードは 6 キャラクタよりも長くするべきであり、非アルファベットキャラクタを含ませるべきです。

**Web 上の PDF を保護** PDF が Web で提供される場合には、ユーザーは必ずその文書のローカルコピーを自分のブラウザで作ることができます。PDF 文書がユーザーにローカルコピーをとられないようにする方法はありません。

## 4.2 PLOP の PDF セキュリティ機能

PLOP は、Acrobat の標準のセキュリティ機能を、PDF ファイルに適用したり、PDF ファイルから除去したりします。PLOP では、ユーザーパスワードとマスターパスワードを適用することができ、また、アクセス権限を設定して、Acrobat で文書を印刷できないようにしたり、テキストを抽出できないようにしたり、文書を変更できないようにしたりすることができます。文書を復号するには、適切なマスターパスワードが必要です。

**暗号化アルゴリズムとキー長** 文書の保護に用いられる暗号化アルゴリズムとキー長は、生成文書の PDF バージョンに依存します。生成文書の PDF バージョンは、入力文書の PDF バージョンと `PLOP_create_file()` の `compatibility` オプションに依存します。暗号化アルゴリズムは以下のように選ばれます：

- ▶ PDF バージョン 1.3 以下は、保護オプション `userpassword・masterpassword・permissions` のいずれかが適用されていれば、PDF 1.4 へ押し上げられます。RC4 40 ビットは決して用いられません。
- ▶ PDF 1.4・1.5：128 ビットキーによる RC4 暗号化の各種類が用いられます。
- ▶ PDF 1.6・PDF 1.7・PDF 1.7ext3：AES-128 が用いられます。なお、PDF 1.7ext3 (Acrobat 9) に従った AES-256 は、既知の脆弱性がありますので決して用いられません。
- ▶ PDF 1.7ext8・PDF 2.0：Acrobat X に従った AES-256 が用いられます。

40 ビットの暗号化キーが安全でないことは広く知られていますので、PLOP ではつねに 128 ビットキーを用い、40 ビットキーを適用する暗号化は一切行いません。ただし、40 ビット暗号化された文書は入力としては受け付けられます。

**さまざまな PLOP の操作に必要なパスワード** PDF 文書の権限設定に反映された作成者の意図に厳密に従うためには、暗号化文書に対してあらゆる操作を許すわけにはいきません。PLOP は以下のルールに従って動作します：

- ▶ 暗号化の状態を pCOS 擬似オブジェクト `encrypt/algorithm` で取得することは、パスワードがどうであれ、つねに可能です。
- ▶ 文書のプロパティを pCOS インタフェースで取得できるかどうかは、pCOS モードによって決まります。たとえば、XMP 文書メタデータ・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ければ（あるいはユーザーパスワードのみが与えられている場合）、マスターパスワードなしで取得できます。pCOS パスリファレンスで詳しく述べています。
- ▶ ユーザーパスワード・マスターパスワード・権限設定を、変更・除去するには、マスターパスワードが必要です。
- ▶ 暗号化された文書に対して、線形化・最適化・修復・署名を行うには（1.4 節「Web 最適化（線形化）PDF」（15 ページ）参照）、マスターパスワードが必要です。

表 4.2 に、すべての操作について何が必要かをまとめました。

表 4.2 暗号された文書に対するさまざまな操作のために必要なパスワード

知っているパスワード	暗号化の状態を取得 (pCOS 擬似オブジェクト「encrypt」)	文書情報・XMP メタデータ・しおり・注釈内容を pCOS で取得	パスワード・権限を変更	線形化・最適化・修復・署名
なし	可能	ユーザーパスワードが設定されていないときのみ	不可	不可
ユーザー	可能	可能	不可	不可

表 4.2 暗号された文書に対するさまざまな操作のために必要なパスワード

知っているパスワード	暗号化の状態を取得 (pCOS 擬似オブジェクト「encrypt」)	文書情報・XMP メタデータ・しおり・注釈内容を pCOS で取得	パスワード・権限を変更	線形化・最適化・修復・署名
マスター	可能	可能	可能	可能

**PLOP でパスワードを設定** PLOP ライブラリ API と PLOP コマンドラインオプションでは、元の PDF 文書を入力文書と呼び、暗号化または復号された生成物を出力文書と呼ぶことにします (どちらも同じファイル名の場合もありますが)。入力文書が保護されている場合、PLOP は表 4.2 に従って、行いたい操作によってユーザーパスワードかマスターパスワードのいずれかを必要とします。入力文書を成功裏に開くことができたならば (保護されていない文書だった場合、または正しいパスワードを与えたことによって)、出力文書にはユーザーパスワード・マスターパスワード・権限設定を任意の組み合わせで適用できます。ただし PLOP は、クライアントが出力文書のために与えるパスワードについて、以下のように作用します：

- ▶ ユーザーパスワードか権限設定が与えられているのに、マスターパスワードが与えられていない場合は、通常の利用者がセキュリティ設定を簡単に変更ことができ、したがって保護を破れてしまいます。ですので PLOP 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一の場合、ユーザーとファイルの所有者との区別はもはや不可能となり、したがってやはり有効な保護は破れてしまいます。PLOP はこの状況をエラーと見なします。
- ▶ AES-256 では Unicode パスワードが許されます。それ以外のすべての暗号化アルゴリズムでは、Latin-1 文字セットに限られたパスワードを必要とします。古い暗号化アルゴリズムの場合に、与えられたパスワードが Latin-1 文字セット外のキャラクタを含んでいると、例外が発生します。
- ▶ パスワードは、AES-256 では 127 UTF-8 バイトまでに、古い暗号化アルゴリズムでは 32 キャラクタまでに切り落とされます。

**PLOP で権限を設定** PLOP は、表 4.3 に示す任意の権限設定を、取得・設定・削除することができます。特記なき限り、すべての動作はデフォルトでは許されます。アクセス制限を指定すると、Acrobat のそれに対応する機能が無効になります。アクセス制限は、ユーザーパスワードを設定しなくても適用できますが、マスターパスワードは必要です。表 4.3 に、使える権限キーワードを列挙します。

表 4.3 PLOP\_create\_file() の permissions オプションに対するアクセス制限キーワード一覧

キーワード	説明
<i>noprint</i>	Acrobat でそのファイルが印刷できなくなります。
<i>nomodify</i>	Acrobat 上でユーザーがフォームフィールドを追加することも、その他いかなる変更を加えることもできなくなります。
<i>nocopy</i>	Acrobat でテキストやグラフィックをコピー・抽出できなくなり、アクセシビリティも無効になります。
<i>noannots</i>	Acrobat で注釈・フォームフィールドを追加・変更できなくなります。
<i>noforms<sup>1</sup></i>	( <i>noannots</i> と暗黙に見なされます) Acrobat でフォームフィールドへの記入ができなくなります。 <i>noannots</i> が指定されていなくても同様です。
<i>noaccessible<sup>1</sup></i>	(PDF 2.0 では非推奨) Acrobat でテキストやグラフィックをアクセシビリティ目的で抽出できなくなります。

表 4.3 PLOP\_create\_file() の permissions オプションに対するアクセス制限キーワード一覧

キーワード	説明
<b>noassemble<sup>1</sup></b>	(nomodify と暗黙に見なされます) Acrobat でページの挿入・削除・回転およびしおり・サムネールの作成ができなくなります。nomodify が指定されていなくても同様です。
<b>nohiresprint<sup>1</sup></b>	Acrobat で高解像度印刷ができなくなります。noprnt が指定されていなければ、印刷は「画像として印刷」機能に制限され、ページの低解像度版が印刷されます。
<b>plain-metadata<sup>2</sup></b>	暗号化された文書でも、文書のメタデータを暗号化しないままにします。

1. PDF 出力のバージョン番号を PDF 1.4 (Acrobat 5 以上が必要) へ押し上げます
2. PDF 出力のバージョン番号を PDF 1.5 (Acrobat 6 以上が必要) へ押し上げます

**PLOP でできないこと** 暗号化文書に対しては、技術的に可能であっても、文書の作成者の意図を冒すので PLOP ではあえて対応していない操作がいくつかあるということを認識することは重要です：

- ▶ PLOP はクラッカーツールではありません。これを利用して、保護された文書に対し、その (適切なユーザーまたはマスター) パスワードを知らずにアクセスを得ることはできません。
- ▶ PLOP では、マスターパスワードなくして権限設定を変えることを許していません。
- ▶ PLOP では、マスターパスワードなくして、ユーザーパスワード・マスターパスワードを変更することを許していません。
- ▶ PLOP では、ユーザーパスワードかマスターパスワードなくして、暗号化された文書の文書情報フィールドを読みません。
- ▶ PLOP は PDF のパスワードセキュリティに対応していますが、証明書ベースの暗号化には対応しておらず、またいかなるサードパーティの PDF 用の暗号化やデジタル著作権管理システム (Adobe Digital Editions や FileOpen 等) にも対応していません。
- ▶ PLOP はデジタル著作権管理 (DRM) システムではありません。文書を個別のコンピュータ・ユーザ・CPU に紐付けることはできません。

## 4.3 コマンドラインで PDF 文書を保護

文書を暗号化するには、`PLOP_create_file()` で `userpassword` オプションか `masterpassword` オプション（両方でも可）指定します。ただし、ユーザーパスワードは必ずマスターパスワードを必要としますが、逆は真ではありません。PLOP ライブラリによる PDF 文書の保護および保護の除去については、その完全なサンプルコードを、すべての PLOP パッケージに入っている `encrypt・decrypt` プログラミングサンプルで見ることができます。PLOP コマンドラインツールでこれと等価なオプションは `--user` と `--master` です。

権限設定は、`PLOP_create_file()` で `permissions` オプションを用いて指定できます。コマンドラインツールでこれと等価なオプションは `--permissions` です。

**注記** Windows では、コマンドライン上のパスワードは、Latin-1 文字セット外の Unicode キャラクタを含むことも可能です。

**暗号化の例** 以下のサンプルコマンドライン呼び出しでは、コマンドラインを長いものと短縮形の両方で示しています。

ファイルをユーザーパスワード `demo` とマスターパスワード `DEMO` で暗号化：

```
plop --user demo --master DEMO --outfile encrypted.pdf input.pdf
plop -u demo -m DEMO -o encrypted.pdf input.pdf
```

カレントディレクトリ内のすべてのファイルを、同一のユーザーパスワード `demo` とマスターパスワード `DEMO` で暗号化し、できたファイルをターゲットディレクトリ `output` へ置く：

```
plop --targetdir output --user demo --master DEMO *.pdf
plop -t output -u demo -m DEMO *.pdf
```

スペースキャラクタを含むパスワードは、次の例のように中カッコで（オプションリスト文法に従うために）、さらにストレートな引用符で（シェル文法に従うために）くくる必要があります。文書をマスターパスワード `two words` で暗号化：

```
plop --master "{two words}" --outfile encrypted.pdf input.pdf
plop -m "{two words}" -o encrypted.pdf input.pdf
```

**復号の例** 1 個のファイルをマスターパスワード `DEMO` で復号。入力文書にアクセス制限がかけられていたとしても、それらはすべて除去されます（なぜなら出力は暗号化されていないので）：

```
plop --password DEMO --outfile decrypted.pdf encrypted.pdf
plop -p DEMO -o decrypted.pdf encrypted.pdf
```

**より強い暗号化方式で再暗号化** PLOP を利用すると、短いキーや弱いパスワードで暗号化されている文書に、もっと強い暗号化を施すこともできます。古いパスワードと新しいパスワードを与える必要があります。PDF 1.6 出力互換を選択すると、強い AES 暗号化が有効になります。次の例では、入力文書はマスターパスワード `old` で暗号化されているときに、出力をマスターパスワード `DEMO` で AES 暗号化する場合を想定しています。新しいパスワードは古いパスワードと同じでもかまいません。もちろん、実際にはこの例のような短いパスワードではなく、強いパスワードだけを用いるべきです（「セキュリティ推奨項目」（52 ページ）参照）：

```
plop --compatibility 1.6 --password old --master DEMO --outputfile strong.pdf weak.pdf
plop -c 1.6 -p old -m DEMO -o strong.pdf weak.pdf
```

**権限設定** マスターパスワード *DEMO* と、権限設定 *noprint・nocopy・noannots* を、ディレクトリ内のすべてのファイルに適用して、できたファイルをターゲットディレクトリ *output* に置く。入力文書で使われている暗号化が何であるかにかかわらず、AES 暗号化が用いられます (PDF バージョン 1.6 により強制されるので)。詳細度レベル 2 では、すべての入力・出力ファイルの名前が、処理されるにつれて印字されます：

```
plop --verbose 2 --compatibility 1.6 --master DEMO ←
    --permissions "noprint nocopy noannots" --targetdir output *.pdf
plop -v 2 -c 1.6 -m DEMO --permissions "noprint nocopy noannots" -t output *.pdf
```

すべての権限設定をファイルから除去し、その結果を別の出力ファイルへ、同じマスターパスワードでコピー。これには入力文書に対するマスターパスワードが必要です：

```
plop --password DEMO --master DEMO --outfile unrestricted.pdf protected.pdf
plop -p DEMO -m DEMO -o unrestricted.pdf protected.pdf
```

文書を再暗号化し (たとえば、弱い暗号化を強い AES 暗号化に換えたり、弱いパスワードをもっと良いものに換えたり)、権限設定は入力文書のものを複製。結果を別の出力ファイルへコピー。これには入力文書に対するマスターパスワードが必要です：

```
plop --password DEMO --master LONGPASSWORD --permissions keep ←
    --outfile unrestricted.pdf protected.pdf
plop -p DEMO -m LONGPASSWORD --permissions keep -o unrestricted.pdf protected.pdf
```



# 5 PLOP DS による電子署名

注記 PDF 文書に電子的に署名する機能は、PDFlib PLOP DS でのみ利用可能であり、基本版の PLOP にはありません。

## 5.1 電子署名の基本概念

電子署名について詳しく説明することはこのマニュアルの範囲外です。しかし、PLOP DS で PDF 文書に電子的に署名する際に役割を果たす最も重要な概念をいくつか挙げていきます。

電子署名は、公開鍵暗号（非対称暗号化ともいう）に基づいています。そこでは、文書に署名する人だけが知っている秘密鍵と、署名を検証するための誰もが知っている公開鍵が用いられます。

公開鍵は一般に、いわゆる証明書ファイルに入った状態で頒布されます。この証明書ファイルの中には、署名者の公開鍵のほか、その人の名前や具体的な連絡先が入っています。証明書の偽証を防ぐため、この情報パッケージはさらに、人やその他の主体（企業・サーバ等）に対して証明書を発行している、信頼された第三者によって署名されます。このような信頼された第三者を、認証局（CA）またはトラストセンタ（TC）といいます。認証局自身の証明書をルート証明書といいます。これは通常、認証局のウェブサイトで公開されて、誰でもダウンロードできるようになっています。これは、認証局によって発行された証明書をそのフィンガープリントによって検証するために必要です（後述）。

証明書は一般に、X.509 形式で格納されます。証明書とそれに対応する秘密鍵の入ったパッケージをデジタル ID といいます。これを証明書と区別することは重要です。証明書は誰にでも自由に配布してかまいませんが、デジタル ID はしっかり隠さなければいけません。デジタル ID の中の秘密鍵にアクセスする（電子署名を行うために）には通常、パスワードまたはパスフレーズが必要です。デジタル ID の広く利用されている格納形式は PKCS#12 と PFX です。ただし証明書とデジタル ID は、いつもきちんと区別されているとは限りません。「証明書を使って文書に署名」という言い方をしている人がいますが、実際はそれは「デジタル ID を使って署名」という意味でしょう。

証明書には必ず、それに関連づけられたハッシュ値（フィンガープリントともいう）があり、その証明書が真正かをダブルチェックするために利用できます。認証局の証明書を手作業で検証するには、そのフィンガープリントを適当なソフトウェアを使って読み取り、それを他の何らかの（信頼できる）手段で得たその認証局のフィンガープリントと照らし合わせる必要があります。証明書は一定の期間のみ有効です。その有効期限が切れればただちに証明書は無効になりますし、あるいは、認証局によって明示的に停止された場合も無効となります。証明書が停止される必要が生じるのは、証明書の所有者が関連の機関を去ったときや、秘密鍵が漏れたかもしれないときです。

公開鍵基盤（PKI）は、証明書を配布したりその有効性を検査したりするためのあらゆる関連タスクを網羅したソフトウェア環境です。証明書の検査は、オンライン検査の場合もあれば（オンライン証明書状態プロトコル＝OCSP というプロトコルを使用）、停止リストを用いる場合もあります。認証局も PKI も得られない場合は、自己署名証明書を使うことも可能です。その場合は、証明書のフィンガープリントを直接、信頼できる転送手段でやりとりする必要があるため、一般には小規模の利用者グループでしか現実的ではありません。

## 5.2 デジタル ID の取得と管理

**電子署名作成のための各種暗号化エンジン** PLOP DS はさまざまな暗号化エンジンに対応しています。暗号化エンジンとは、電子署名の生成に必要なさまざまな暗号機能を実装したソフトウェア部品です。暗号化エンジンの選択は、デジタル ID の形式と保管場所、および他のソフトウェアやオペレーティングシステムとの統合に影響を与えます。PLOP DS は、以下の暗号化エンジンに対応しています。

- ▶ **builtin** エンジン：すべてのプラットフォームで利用可能です。これは、必要な暗号関数群を PLOP DS のカーネル内に実装したもので、外部に一切依存しません。このエンジンはデフォルトで有効ですが、`PLOP_create_file()` の `sign` オプションでサブオプション `engine=builtin` を設定して明示的に選択することもできます。
- ▶ **mscapi** エンジン：Windows オペレーティングシステムに内蔵の Microsoft Cryptographic API を指します (Windows でのみ利用可能)。これを利用すると、PLOP DS は、Windows が提供する暗号インフラストラクチャや、CAPI ドライバでアタッチされるサードパーティのソフトウェア・ハードウェアと共同することができます。`mscapi` エンジンを選択するには、`PLOP_create_file()` の `sign` オプションでサブオプション `engine=mscapi` を設定します。
- ▶ **pkcs#11** エンジン：暗号トークンへの統一されたアクセスを提供する PKCS#11 というソフトウェアインタフェースを指します。ここでトークンとは、スマートカード・USB スティックその他の暗号デバイスを指します。トークンは多くの場合、ソフトウェア認証よりも高いセキュリティを実現し、多くは PIN で保護されています。このエンジンはすべてのプラットフォームで利用できるわけではありません。`pkcs#11` エンジンを選択するには、`PLOP_create_file()` の `sign` オプションでサブオプション `engine=pkcs#11` を設定します。
- ▶ PLOP DS ユーザーは、**外部暗号化エンジン (CE)** をフックアップすることもできます。これを利用すると、証明書や署名の生成に関してカスタムな要請を実装し、既存の暗号化ハードウェアまたはソフトウェアモジュールをフックアップすることができます。CE インタフェースの解説とそのバイナリはお求めに応じて提供します。標準の PLOP DS 内のバイナリでは CE インタフェースには対応していません。

**デジタル ID の対応形式** PLOP DS では、PDF 文書に署名するためにデジタル ID が必要です。デジタル ID の中には、署名者のデジタル証明書のほかに、それに対応する秘密鍵が入っていますので、通常、パスワードやそれに類する手段で守られています。PLOP DS は以下の種類のデジタル ID に対応しています：

- ▶ すべてのプラットフォームで、`engine=builtin` と設定した場合：PKCS#12 形式 (通常 `.p12`) か PFX 形式 (通常 `.pfx`) のデジタル ID ファイル
- ▶ Windows 上で、`engine=mscapi` と設定した場合：Windows 証明書ストア内のデジタル ID
- ▶ PKCS#11 対応のすべてのプラットフォームで、`engine=pkcs#11` と設定した場合：コンピュータに接続されているスマートカードその他の暗号トークン (デバイス) 上に格納されているデジタル ID

**デジタル ID の入手元** デジタル ID を入手できる元はさまざまあります。多くの ID は、電子メールへの署名を想定されています。こうした電子メール ID を PLOP DS で使って PDF 文書を署名することも可能です。デジタル ID をどこから得るかは、必要な ID の個数や (たとえば従業員ごとに 1 個ずつ、あるいは会社の ID を 1 個だけ)、求める制御の自由度によって選択できます：

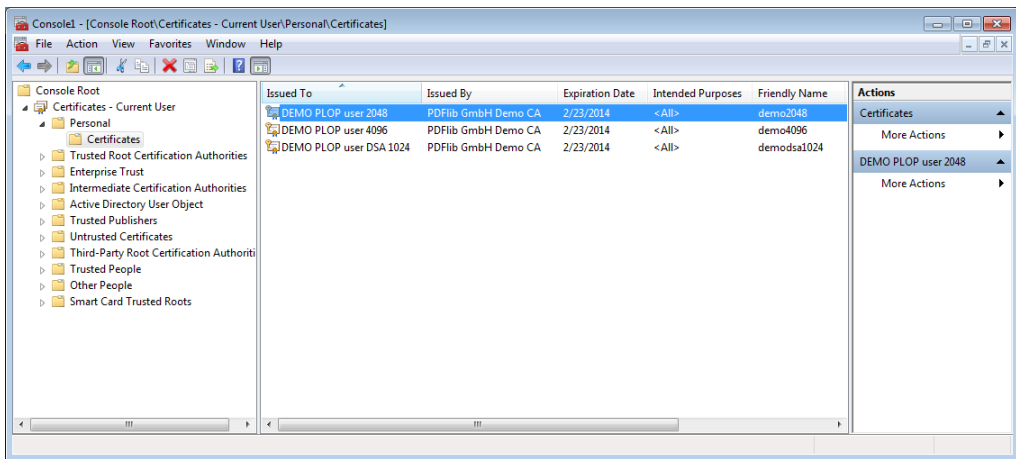


図 5.1 Windows 証明書ストアを管理コンソール (MMC) で管理

- ▶ IDを無償または有償で発行しているパブリック認証局のいずれかからデジタルIDを得る。
- ▶ 自分のプライベート認証局を構築して、デジタルIDを自分で作成できるようにする。認証局を構築できるソフトウェアパッケージはさまざまあります。たとえば無償の OpenSSL ソフトウェアや ([www.openssl.org](http://www.openssl.org) 参照)、Java の一部である *keytool* アプリケーションや、Microsoft Windows Server オペレーティングシステムの一部である証明書サービスなどが挙げられます。
- ▶ 自己署名証明書からデジタルIDを作成する。Acrobat で自己署名証明書を作成するには次のようにします：
  - Acrobat X : 「ツール」 → 「保護」 → 「その他の保護」 → 「セキュリティ設定」 → 「デジタル ID」 → 「IDを追加」 → 「今すぐデジタル IDを新規作成」
  - Acrobat 9 : 「アドバンスト」 → 「セキュリティ設定」 → 「デジタル ID」 → 「IDを追加」 → 「今すぐデジタル IDを新規作成」
  - Acrobat 8 : 「アドバンスト」 → 「セキュリティ設定」 → 「デジタル ID」 → 「IDを追加」 → 「Acrobat で使用する Self-Sign デジタル IDを作成」
 その次のステップで、ターゲットとして PKCS#12 のディスクファイルか、Windows 証明書ストアを指定できます。どちらの方式も PLOP DS で使えます。

**Windows 証明書ストアを管理** Windows オペレーティングシステムでは、任意の数の証明書を持つことができ、それらはいくつかの証明書ストアにまとめられます (物理的にはレジストリに格納されます)。PFX 形式か PKCS#12 形式の新規証明書をインストールするには、証明書ファイルを単にダブルクリックして、証明書のインポートウィザードに従っていけば完了します。PLOP DS パッケージに入っているデモ証明書で、パスワード *demo* で試すことができます。

Windows で証明書を表示したりまとめたりするには、Microsoft 管理コンソール (MMC) を使って次のようにします：

- ▶ 「スタート」 → 「ファイル名を指定して実行 ...」をクリックし、「mmc」と入力して、OK をクリックします。すると、管理コンソールが起動します。
- ▶ 「ファイル」メニューで「スナップインの追加と削除 ...」をクリックします。

- ▶ 「利用できるスタンドアロン スナップイン」で「証明書」を選択し、「追加」をクリックします。
- ▶ その次のダイアログで「ユーザー アカウント」を選択し、「完了」をクリックします。あるいはもし、「サービス アカウント」か「コンピュータ アカウント」に自分の証明書を入れてあるなら、それを用います。
- ▶ 「OK」をクリックします。

これで、インストールされている証明書を閲覧できるようになりました。自分自身の証明書は「個人」カテゴリにあり、これは PLOP DS において次のオプションリストで指定できます (--*signopt* コマンドラインオプションに与えるか、*PLOP\_create\_file()* の *sign* オプションに与える) :

```
engine=mscapi digitalid={certstore={store=My subject={Demo PLOP User 2048}}}
```

証明書の詳細を見るには、MMC で証明書をダブルクリックします。証明書を PFX 形式へ書き出すには、一覧の中の証明書を右クリックして、「すべてのタスク」→「エクスポート ...」をクリックします。すると、証明書のエクスポートウィザードが起動します。

管理コンソールを利用して、証明書を取り込むすることもできます。証明書ストア(「個人」等)を右クリックして、「すべてのタスク」→「インポート ...」を選択します。.

Windows では、以下のキー長が使えます :

- ▶ RSA アルゴリズム : Microsoft Enhanced Cryptographic Provider で 384 ビットから 16384 ビット、Microsoft Base Cryptographic Provider で 384 ビットから 512 ビット。
- ▶ DSA アルゴリズム : 512 ビットから 1024 ビット (Acrobat が 4096 ビットまでの DSA を使えるのとは異なります)

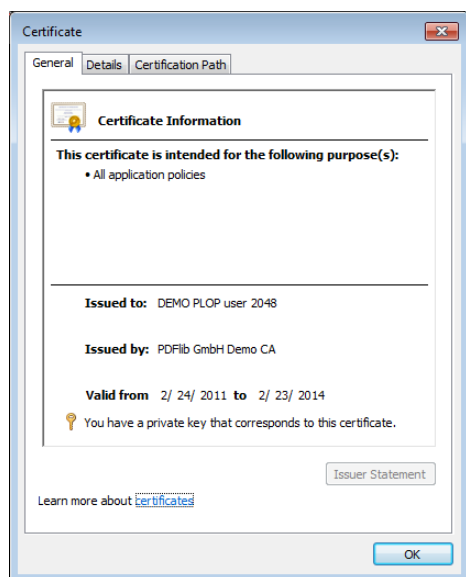


図 5.2  
Windows の証明書プロパティ

## 5.3 PLOP DS で PDF 文書に署名

署名は PDF ではフォームフィールドとして実装されています。PDF の署名はつねに文書全体に紐づいており（特定のページにではなく）、次の 2 つの種類があります：

- ▶ 不可視署名：ページ上でスペースをまったくとりません。Acrobat で「署名」タブを表示させると（Acrobat X：「表示」→「表示切り替え」→「ナビゲーションパネル」→「署名 ...」。Acrobat 8/9：「表示」→「ナビゲーションパネル」→「署名 ...」）見ることができます。
- ▶ 可視署名：文書のいずれかのページのどこかに置かれた矩形のフォームフィールドを用いています。ページ番号・フィールド名・フィールド座標を指定できます。

どちらの種類も署名でも、ほかにもさまざまなプロパティを指定することができます。たとえば位置や署名の理由、連絡先情報等です。

PLOP DS を使って電子署名を行うには、デジタル ID が必要です（5.2 節「デジタル ID の取得と管理」（60 ページ）参照）。デジタル ID またはトークンを利用するには、そのパスワードが必要です。Windows 証明書ストアの中の個人の（アカウントごとの）デジタル ID を利用する場合、ID は Windows ログインで保護されています。

**PLOP DS で署名を行う** 以下の例で、PLOP DS のコマンドラインツールとライブラリを使って PDF 文書に電子的に署名を行う方法を示します。--*signopt* に与えているオプションリストは、PLOP DS の API 関数 *PLOP\_create\_file()* (*sign* オプション) に与えれば、自分自身のプログラムの中で署名を作成することができます。対応しているすべての言語バイインディングに対する完全なプログラミング作成例が、PLOP DS パッケージに入っています。これらの作成例では、デジタル ID ファイル *demo2048.p12* と *demo2048.pfx* のパスワードが *demo* であり、また、Windows 証明書ストアの中のデジタル ID が架空のユーザー名 *DEMO PLOP user 2048* のものであると前提しています。サンプルデジタル ID ファイルがディストリビューションパッケージに入っています。

PDF 文書に不可視署名を作成。ファイル *demo2048.p12* のデジタル ID を使用。デジタル ID のパスワードはファイル *pw.txt* の中にある場合：

```
plop --signopt "digitalid={filename=demo2048.p12} passwordfile=pw.txt" ←  
--outfile signed.pdf input.pdf  
plop -S "digitalid={filename=demo2048.p12} passwordfile=pw.txt" -o signed.pdf input.pdf
```

可視署名フィールドを、ページ 1 の左下部分に作成。パスワード *demo* は直接与えています。これはマルチユーザーシステムでは推奨しません。なぜならパスワードを含むコマンドラインが他のユーザーに見えるかもしれないからです（Unix システムなら *ps* コマンド等で）：

```
plop --signopt "appearance={fieldname=Signature1 rect={10 10 200 100} } ←  
digitalid={filename=demo2048.p12} password={demo}" --outfile signed.pdf input.pdf  
plop -S "appearance={fieldname=Signature1 rect={10 10 200 100} } ←  
digitalid={filename=demo2048.p12} password={demo}" -o signed.pdf input.pdf
```

**注記** 以下の Windows での作成例を動作させるには、ファイル *demo2048.pfx* 内のデジタル ID をダブルクリックして Windows 証明書ストアにインストールする必要があります。

(Windows のみ) PDF 文書に不可視署名を作成。Windows 証明書ストアの証明書を使用（デフォルトストア *My* の）。ここでは、デジタル ID は自分の Windows ログインによって保護されているため、パスワードを与える必要がないと前提しています：

```
plop --signopt "engine=mscapi digitalid={certstore={store=My subject={Demo PLOP User 2048}}}" ←  
    --outfile signed.pdf input.pdf  
plop -S "engine=mscapi digitalid={certstore={store=My subject={Demo PLOP User 2048}}}" ←  
    -o signed.pdf input.pdf
```

(Windows のみ) PDF 文書に不可視署名を作成。ファイル *demo2048.pfx* の証明書を使用 :

```
plop --signopt "engine=mscapi digitalid={filename=demo2048.pfx} passwordfile=pw.txt" ←  
    --outfile signed.pdf input.pdf  
plop --S "engine=mscapi digitalid={filename=demo2048.pfx} passwordfile=pw.txt" ←  
    -o signed.pdf input.pdf
```

不可視署名を作成し、PDF を暗号化するマスターパスワード *SECRET* と、デジタル ID にアクセスするパスワード *demo* で文書を暗号化 :

```
plop --master SECRET --signopt "digitalid={filename=demo2048.p12} password={demo}" ←  
    --outfile signed.pdf input.pdf  
plop --m SECRET --S "digitalid={filename=demo2048.p12} password={demo}" ←  
    -o signed.pdf input.pdf
```

**スマートカードその他の暗号トークンで署名** PLOP DS内のPKCS#11 エンジンを用いると、スマートカードその他の暗号トークン上の証明書を利用することができます。そのためには、トークン独自のプロトコルを実装している DLL または共有ライブラリが必要です。PKCS#11 DLL はトークンのベンダーから、そのドライバキットに含まれた形で提供されている必要があります。それはシステムにインストールされている必要があります、かつ PLOP DS で利用可能になっている必要があります。Windows の場合これはすなわち、DLL は、Windows のシステムディレクトリか、PATH 環境変数に含まれているディレクトリか、あるいはカレントディレクトリにコピーされている必要があることを意味します。なお、PKCS#11 DLL が他の DLL に依存している可能性もあります。そのような場合には、トークンのベンダーが提供している必要な DLL がすべて、PLOP DS で利用可能になっている必要があります。

以下の作成例では、ベンダー独自の PKCS#11 DLL を *cryptoki.dll* とします。実際の DLL の名前はこれとは異なる可能性があります。

PKCS#11 で指定されたトークンの中のデジタル ID を用いて、PDF 文書に不可視署名を作成。トークンの PIN はファイル *pw.txt* に入っているものとします :

```
plop --signopt "engine=pkcs#11 digitalid={filename=cryptoki.dll} passwordfile=pw.txt" ←  
    --outfile signed.pdf input.pdf  
plop -S "engine=pkcs#11 digitalid={filename=cryptoki.dll} passwordfile=pw.txt" ←  
    -o signed.pdf input.pdf
```

PKCS#11 で指定されたトークンの中のデジタル ID を用いて、PDF 文書に不可視署名を作成。このコマンドでは PIN は与えておらず、かわりにトークンの内蔵キーボードでトークンの PIN を入力する必要があります :

```
plop --signopt "engine=pkcs#11 digitalid={filename=cryptoki.dll}" ←  
    --outfile signed.pdf input.pdf  
plop -S "engine=pkcs#11 digitalid={filename=cryptoki.dll}" -o signed.pdf input.pdf
```

ページ 1 の左下部分に可視署名フィールドを作成。トークンの PIN 1234 を直接与えています。これはマルチユーザーシステムでは、コマンドラインがパスワードごと他のユーザーに見えるおそれがあるため、推奨しません：

```
plop --signopt "appearance={fieldname=Signature1 rect={10 10 200 100} } ←  
engine=pkcs#11 digitalid={filename=cryptoki.dll} password={1234}" ←  
--outfile signed.pdf input.pdf  
plop -S "appearance={fieldname=Signature1 rect={10 10 200 100} } ←  
engine=pkcs#11 digitalid={filename=cryptoki.dll} password={1234}" ←  
-o signed.pdf input.pdf
```

**トークン上のデジタル ID を選択** スマートカードなどの暗号トークン 1 個の中に、複数のデジタル ID が入っている場合があります。たとえば、1 個は電子メールの暗号化用、もう 1 個は文書への電子署名用というような場合です。この場合には、`PLOP_create_file()` の `sign` オプションの `keyusage` サブオプションを用いて、PLOP DS 署名のためのターゲット ID を選択することができます。これは、証明書の `KeyUsage` 拡張内に符号化されている用途フラグに基づきデジタル ID を選択するキーワードを受け付けます (`KeyUsage` 拡張の詳細な説明は RFC 3280 を参照してください)。たとえば、スマートカードの中に 2 個の ID があり、`nonrepudiation` フラグを伴う ID を署名に使う必要があるときは、`sign` オプションに対して下記のサブオプションを用いることができます：

```
digitalid={filename=cryptoki.dll keyusage={nonrepudiation=set}}
```

**デジタル ID のロック解除** デジタル ID は、電子署名を作成するための秘密鍵を保持しているため、通常、パスワードないしパスフレーズまたは PIN で保護されています。デジタル ID を PLOP DS で使うためにロック解除するには、適切な認証を与える必要があります。間違ったパスワードを与えた場合、PLOP DS は例外を発生させます。デジタル ID のロック解除の具体的な方法は、以下のように、選択している暗号化エンジンによって異なります。

- ▶ `engine=builtin` のとき：そのパスワードを `sign` オプションの `password` サブオプションに与える必要があります。PLOP DS コマンドラインツールを使う場合は、パスワードは別ファイルに入れて `passwordfile` サブオプションで別途与えることを強く推奨します。パスワードファイルを使わずにパスワードを直接与えた場合、他のユーザーがそれを読める可能性があります。なぜならマルチユーザーシステムではコマンドラインが他のユーザーに見えるかもしれないからです。
- ▶ `engine=mscapi` のとき：Windows 証明書ストアの中のデジタル ID は、自分の証明書の設定によっては、自分の Windows ログインによって保護されているので、パスワードをさらに与える必要はありません。
- ▶ `engine=pkcs#11` のとき：暗号トークンがソフトウェアからのパスワード/PINの送信を許している場合は、`engine=builtin` の場合と同様に `password` オプションを与える必要があります (上述)。トークンが PIN やパスワードの直接入力が必要としている場合は、`password` オプションは省略することができ (あるいは空文字列を与え)、PIN はトークンのキーボードに手入力する必要があります。パスワード/PIN の具体的扱いは、暗号トークンによってさまざまです。

## 5.4 PLOP DS の署名の暗号の詳細

**署名の暗号化アルゴリズムとキー長** 署名生成のための暗号化アルゴリズムとキー長は、デジタル ID によって決定されます（それらは、ID の公開鍵 / 秘密鍵ペアを作成する際に指定されています）。PLOP DS では、以下のアルゴリズムとキー長に対応しています。

- ▶ RSA、キー最大長 4096 ビット
- ▶ DSA、キー最大長 4096 ビット。DSA では SHA-1 メッセージダイジェストのみ使えます。

**メッセージダイジェスト（ハッシュ関数）** 電子署名を生成するために用いられるメッセージダイジェストアルゴリズム（ハッシュ関数）は、いくつかの要因によって決定されます。PLOP DS は、安全な SHA-256 アルゴリズムを可能な限り用いるを試みますが、可能でないときは SHA-1 を用います。

SHA-256 か SHA-1 かの選択は 2 つの要因によって行われます：SHA-256 実装が利用可能かどうかと、生成される PDF 出力の種類です。SHA-256 が利用可能かどうかは、選択している暗号化エンジンによって決まります：

- ▶ *engine=builtin* の場合には、SHA-256 はつねに利用可能です。
- ▶ *engine=pkcs#11* の場合には、SHA-256 が利用可能かどうかは、暗号トークンの能力によって決まります。トークンの文書を参照するか、トークンのベンダーに問い合わせて、そのトークンの暗号機能に関する情報を得てください。
- ▶ *engine=mscapi* の場合には、SHA-256 が利用可能かどうかは Windows のバージョンによって決まります。Microsoft の文書によると、Windows XP SP3 以上では利用可能なはずですが、しかし、XP SP3 で SHA-256 が利用可能でない場合があります。

SHA-256 が利用可能な場合には、生成される PDF 出力が以下の条件を満たすときのみ用いられます：

- ▶ PDF 入力バージョン、要求されている操作、および *compatibility* に基づき決定される PDF 出力バージョンが 1.6 以上。
- ▶ PDF/A-1 出力を生成している。これは上記の PDF 1.6 規則に対する特別な例外であり、PDF/A 文書の安全な電子署名を可能にするものです。

SHA-256 アルゴリズムが入手可能でない場合、または生成される PDF 出力が上に挙げた条件を満たさない場合には、SHA-1 アルゴリズムがかわりに用いられます。MD5 アルゴリズムは十分に安全ではないと考えられますので、PLOP DS が署名にこれを用いることは決してありません。

**PLOP DS の暗号化の制約** PLOP DS は現在のところ、以下の署名関連の機能には対応していません：

- ▶ Certified PDF：これは特殊な署名の一種で、作成者が文書の有効性を保証し、それを他人が後から変更できるものです。
- ▶ 増分アップデートと、1 個の文書に複数の署名。
- ▶ 署名の見た目：可視署名フィールドで表示されるテキストまたは画像を指定することはできません。
- ▶ 電子署名の適用は、線形化と組み合わせることはできません。




## 5.5 Acrobat で電子署名を検証

PLOP DS は、Acrobat の文書化された仕様に従った標準的な PDF 署名を作成します。PLOP DS で作成された電子署名は、検証するためにサードパーティのソフトウェアを一切必要とせず、Acrobat Standard/Professional 8 以上か Acrobat Reader 8 以上で検証できます。標準の Acrobat 署名に対応したサードパーティの PDF 署名検証ソフトウェアでも、PLOP DS で作成された署名を検証することが可能です。

署名が有効なのは、以下の条件がすべて真のときです：

- ▶ 文書が署名された後に変改を受けていない。
- ▶ 署名に用いられている証明書が有効である。証明書は、期限切れ（証明書の中の有効期限によって）の場合や、停止されている場合（オンラインでの試験または停止リストとの照合が必要）には無効です。
- ▶ 証明書が既知の人または組織に属するか、またはよく知られた認証局によって発行されている（後述）。

Acrobat で PDF 署名を検証するには次のようにします。「署名」タブを開いて（「表示」→「ナビゲーションパネル」→「署名 ...」）、署名を右クリックし、そして「署名を検証」を選択します。するとダイアログボックスが現れ、そこに Acrobat は、署名の検証状態アイコンを、その他の情報とともに表示します。Acrobat では、個々の署名と、文書の署名ステータスごとに、さまざまなアイコンを用いています：

- ▶ チェックマーク  は、その署名が有効であることを示します。これはすなわち、その署名者がすでに認証されており、文書が変改を受けていないということです。
- ▶ 赤い×  は、その署名が無効であることを示します。これはすなわち、その署名者の証明書を認証できなかった（期限切れや停止されているなど）か、あるいは文書が変改を受けたということです。
- ▶ 三角形  は、その署名者のアイデンティティを認証できなかったか、あるいは文書が署名された後に変改を受けているために、その文書に問題があることを示します。

Acrobat 8 では署名の状態を、可視署名のフィールド領域にも表示します。しかし、Acrobat 9/X では個々のフィールドには署名状態アイコンは表示されなくなっており、署名パネル（ウィンドウ上端付近）と「署名」タブ（ウィンドウ左側）にしか表示されません。この動作は混在環境では混乱を招きかねませんので、レジストリキー

```
HKEY_CURRENT_USER\Software\Adobe\Adobe Acrobat\10.0\Security\cPubSec\iDisplayValidIcon
```

を値 0（数字のゼロ）に設定すれば、以前の動作（すなわち、検証状態アイコンが各フィールドに表示される）に戻すことができます。これによって、Acrobat 9/X の動作が Acrobat 8 と同様に戻ります。Adobe が提供している Acrobat セキュリティドキュメンテーションには、Acrobat のレジストリ設定について、より詳しく記されています。

**署名者の証明書に対する信頼を確立させる** 文書への署名に用いられた証明書が信頼に足ると認めるべきであると Acrobat が確信できるのは、次のいずれかの場合です：

- ▶ Acrobat に証明書が内蔵されている認証局（CA）が発行した証明書は、つねに信頼に足るとして受け入れられます。ただし、内蔵 CA はソフトウェア証明書を発行しないので、それを PLOP DS で利用することはできません。
- ▶ Windows 証明書ストアに内蔵された CA のいずれか 1 つが発行した証明書。自分の組織外のエンドユーザーに対してはこれが推奨方式です。詳しくは後述します。

- ▶ その証明書がWindows証明書ストアにある。このためにはユーザーが手作業で署名者の証明書を Windows 証明書ストアに追加する必要があるため、一般には組織環境でしか実用的ではありません。
- ▶ Acrobat を自分で手作業で設定して、個々の署名者の証明者を受け入れるように、またはある特定の CA によって署名された証明書をすべて受け入れるようにする。これは組織環境で推奨します。そのために必要な手順は後述します。

**Acrobat が Windows 証明書ストアにアクセスできるようにする** Acrobat が Windows 証明書ストアにアクセスするよう設定されていれば、証明書ストアに入っている CA で作成されたすべての署名を Acrobat で検証できます。これは Windows にすでに内蔵されている CA のいずれか 1 つでも（制御された組織環境の外にいるユーザーに推奨）、カスタム CA でも（自分自身の組織 CA を運営していて、それをすべてのユーザーの証明書ストアに設定できる場合に適している）可能です。管理コンソールでは、これらのよく知られた CA

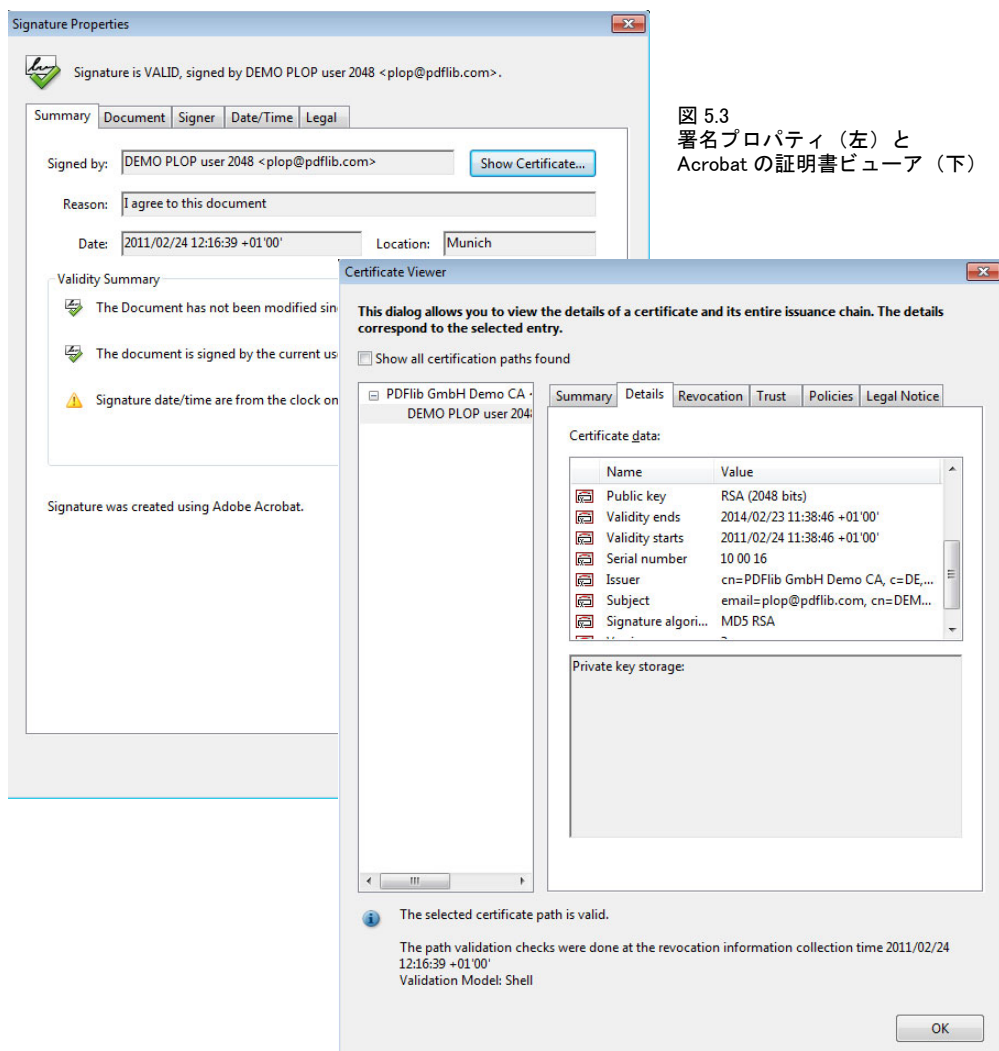


図 5.3  
署名プロパティ（左）と  
Acrobat の証明書ビューア（下）

の証明書は、「信頼されたルート証明機関」ストアか、または「サードパーティルート証明機関」で見つけることができます。

Windows に内蔵の CA を活用するには次の手順を踏みます。Windows に内蔵の商用 CA のいずれか 1 つからデジタル ID を取得します。内蔵 CA の一覧を見るには次のようにします。管理コンソールを起動し、署名スナップインがある状態で（「Windows 証明書ストアを管理」（61 ページ）参照）、「信頼されたルート認証機関」→「証明書」を開きます。すると、何ダースもの商用 CA の一覧が現れます。この一覧から証明書を選んで、その名前をダブルクリックします。「証明書」ダイアログで「詳細設定」を選び、項目「サブジェクト」へスクロールして、それをダブルクリックします。すると、その CA に連絡をとるのに十分な情報（電子メールアドレス等）が表示されるはずです。これらの CA のいずれか 1 つからデジタル ID を取得し、そしてそれを使って PLOP DS で PDF 文書に署名を行います。

カスタム CA を Windows ストアに知らせるには、その証明書を取得し、それをダブルクリックして、そして Windows 証明書ストアへそれを取り込みます。

上記のどちらの場合においても、Acrobat が選択された CA（または Windows 証明書ストア内の任意の CA）によって発行されたすべての証明書を確実に受け入れるようにしておく必要があります。Acrobat 8/9/X では次の手順を踏みます（図 5.4 参照）。「編集」→「環境設定」→ [「一般 ...」] → 「セキュリティ」→ 「詳細環境設定 ...」→ 「Windows 統合」。そして現れるダイアログで、「以下の操作に対しては、Windows 証明書ストアのすべてのルート証明書を信頼します：」と書かれた下の「署名を検証」チェックボックスをオンにします。

**個々の証明書を受け入れる** 個々の証明書を、信頼済みアイデンティティの一覧に追加するには、以下のいずれかの方法を用います（詳しくは Acrobat のヘルプを参照してください）：

- ▶ 証明書を Windows 証明書ストアに追加：証明書ファイルをダブルクリックします。すると、証明書をインストールするためのウィザードが現れます。加えて、Acrobat を Windows 証明書ストアにアクセスするよう設定する必要があります（上述）。

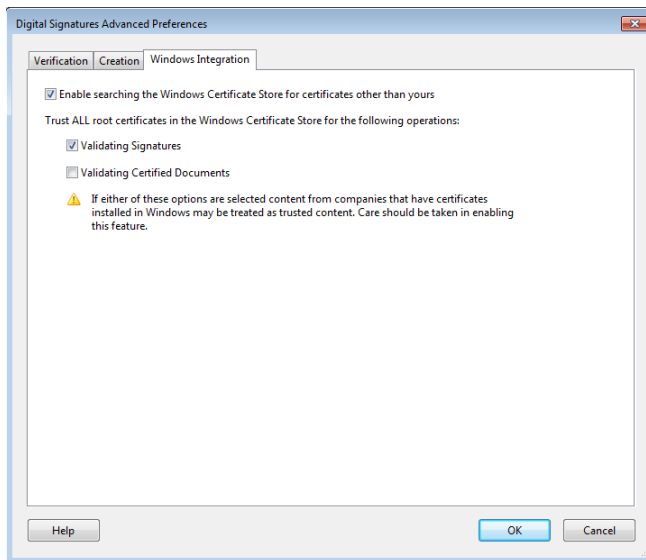


図 5.4  
Acrobat を、Windows 証明書ストアへアクセスするよう設定

- ▶ 証明書をディスクファイルかディレクトリサーバから取り込む：

Acrobat X ではこれは次の手順でできます：

「ツール」→「電子署名」→「その他の電子署名」→「信頼済み証明書 ...」→「連絡先を追加」→「参照 ...」

Acrobat 8/9：「アドバンス」→「信頼済み証明書の管理 ...」→「連絡先を追加」→「参照 ...」→証明書を選択→「信頼を編集 ...」をクリック→「信頼」タブへ移り、「次の対象についてこの証明書を信頼します：」の中の「署名、および信頼されたルート」チェックボックスを選択します。

- ▶ 署名された PDF から証明書をとり込む：Acrobat 9/X ではこれは次の手順でできます。署名された PDF を開き、「署名」タブを開き（Acrobat X：「表示」→「表示切り替え」→「ナビゲーションパネル」→「署名」。Acrobat 8/9：「表示」→「ナビゲーションパネル」→「署名」）、署名を右クリックして「プロパティ」または「署名のプロパティを表示 ...」を選択します。すると、「署名のプロパティ」ダイアログが現れます。「要約」タブで「証明書を表示」をクリックし、「詳細」タブへ移って証明書のフィンガープリントを調べます（MD5 または SHA-1 ダイジェスト。または両方）。もしこのフィンガープリントが、署名者から自分へ信頼するに足る手段で別途もたらされた署名と合致するならば、「信頼」タブへ移り、「信頼済み証明書に追加」をクリックして、最後に「この証明書を信頼済みのルートとして使用」（Acrobat 9/X）または「署名、および信頼されたルート」（Acrobat 8）にチェックを入れます。

**ある CA から発行された証明書をすべて受け入れる** この方式は、多くの別々の個人が皆同じ CA（よくあるのは組織の CA）から発行された証明書を使って署名した文書を扱う場合に推奨します。その CA の証明書を取得して、個々の証明書と同じ上述のやり方で Acrobat へ取り込みます。あるいは、Acrobat から Windows 証明書ストアへのアクセスを許すなら、Windows へ取り込むこともできます。すると Acrobat は、この CA から発行される証明書で作成されるすべての署名を受け入れるようになります。

## 6 pCOS インタフェース

pCOS (*PDFlib Comprehensive Object Syntax*) インタフェースは、PDF 文書のあらゆるセクションから、ページ内容を記述しない任意の、ページ寸法・メタデータ・インタラクティブ要素等の情報を取得するための、シンプルでエレガントな機能群を提供します。pCOS インタフェースの利用例と、pCOS パス文法の説明は、別文書である pCOS パスリファレンスに含まれています。さらなる作成例が pCOS クックブックにあります：

[www.pdfliib.com/pcos-cookbook/](http://www.pdfliib.com/pcos-cookbook/)



# 7 PLOP・PLOP DS ライブラリ API リファレンス

## 7.1 オプションリスト

オプションリストは、PLOP の操作を制御する強力かつ簡単な方式です。多くの API メソッドは、大量の関数引数を必要とするのではなく、オプションリスト（略して *optlist*）に対応しています。これは、任意の数のオプションを含むことのできる文字列です。オプションリストはさまざまなデータ型や、配列のような複合データに対応しています。多くの言語においてオプションリストは、必要なキーワードと値を連結することによって、簡単に組み立てることができます。C プログラマはオプションリストを組み立てるために、*sprintf()* 関数を使いたいところでしょう。1 個のオプションリストは、次の形の対を 1 つないし複数含みます。

名前 値（複数可）

名前と値の間、および複数の名前 / 値対どうしの間は、任意の空白類文字（スペース・タブ・キャリッジリターン・ニューライン）で区切ることができます。値は、複数の値のリストから成る場合もあります。また、名前と値の間は等号「=」で結ぶこともできます：

名前=値

**単純値** 単純値は、以下のデータ型のいずれかを用いることができます：

- ▶ 論理値：*true* または *false*。論理値のオプションで値が省略されたときは、値 *true* と見なされます。略記として、名前 *false* のかわりに *no* 名前を用いることも可能です。
- ▶ 文字列：空白類文字または「=」キャラクタを含む文字列は、`{ }` でかこむ必要があります。空文字列は `{ }` で作れます。キャラクタ `{ }`・`\` は、文字列の中身としたいなら、前に `\` キャラクタを付ける必要があります。
- ▶ テキスト文字列：いくつかのオプションで用いられる特殊な文字列です。文字列型のオプションの多くは ASCII 値しか受け入れることができませんが、テキスト文字列は ASCII 以外に Unicode 値も保持することが可能です。Unicode 対応の言語バイndenディングでは、単に任意の Unicode 値をそうしたオプションに与えることができます。Unicode 非対応の言語バイndenディングでは、文字列を UTF-8 として（i5/iSeries と zSeries では EBCDIC UTF-8 として）解釈するべきなら、ユーザーはテキスト文字列の頭に UTF-8 BOM を付ける必要があります。UTF-8 BOM がないときは、テキスト文字列は *auto* エンコーディングで、すなわち Windows の場合はカレントコードページ、i5/iSeries の場合はカレントジョブのエンコーディング、zSeries の場合は *ebcdic*、Unix・Mac OS X の場合は *iso8859-1* で解釈されます。
- ▶ キーワード：固定されたキーワードの定義済みリストのうちの 1 つ
- ▶ 浮動小数点値・整数：10 進の浮動小数点値または整数。小数点としては点とカンマが使えます。
- ▶ ハンドル：いくつかの内部オブジェクトハンドル、たとえば文書やページのハンドル。実際にはこれらは整数値です。

型によって、またオプションの解釈によっては、さらなる制約が課される場合があります。たとえば、整数や動小数点値は特定の値範囲に制限されるかもしれませんし、ハンド

ルはそのオブジェクトの種別に対して有効でなければならない等です。オプションに対する制約条件は、それぞれの関数の説明に記してあります。単純値のいくつかの例（1行目は空白キャラクタを含む文字列の例です）：

```
password={secret string}  
linearize=true
```

**リスト値** リスト値は複数の値から成り、それらの値は単純値かあるいはまたリスト値かもしれません。リストは{と}でかこまれます。リスト値の例：

```
permissions={ noprint nocopy }
```

**注記** バックスラッシュ\キャラクタは、多くのプログラミング言語において、特殊な取り扱いが必要です。

## 7.2 一般関数

---

**C** *PLOP \*PLOP\_new(void)*

---

新規の PLOP コンテキストを作成します。

**戻り値** 新規コンテキストへのハンドル、または十分なメモリが得られない場合は NULL。コンテキストは、他のすべての API 関数に与える必要があります。

**バインディング** オブジェクト志向言語では、新規 PLOP オブジェクトが作成されたときには自動的に呼び出されるので、得られません。

---

**Java** *void delete()*

**C#** *void Dispose()*

**C** *void PLOP\_delete(PLOP \*plop)*

---

PLOP コンテキストを削除し、その内部リソースをすべて解放します。

**詳細** コンテキスト内のすべての開いている文書は自動的に閉じられます。しかし、文書が必要なくなった時点で *PLOP\_close\_document()* でそれを閉じておくのは良いプログラミング習慣です。

**バインディング** C の場合、この関数は *PLOP\_TRY()/PLOP\_CATCH()* 節の中で呼び出してはいけません。

Java の場合、このメソッドは PLOP のファイナライザメソッドによって呼び出されません。しかし、明示的に *delete()* を呼び出して適切なクリーンアップを行わせることを強く推奨します。例外が起きたときにもこれは然りです。

Perl・PHP・COM の場合、この関数は PLOP オブジェクトが破壊されたときに自動的に呼び出されます。

.NET の場合、非マネージのリソースをクリーンアップするために処理の最後で *Dispose()* を呼び出すべきです。

---

**C++** *void create\_pvf(string filename, const void \*data, size\_t size, string optlist)*

**C# Java** *void create\_pvf(String filename, byte[ ] data, String optlist)*

**Perl PHP** *create\_pvf(string filename, string data, string optlist)*

**VB** *Sub create\_pvf(filename As String, data, optlist As String)*

**C** *void PLOP\_create\_pvf(PLOP \*plop,  
const char \*filename, int len, const void \*data, size\_t size, const char \*optlist)*

---

メモリ上で与えられたデータから、名前付きの仮想の読み取り専用のファイルを作成します。

**filename** (名前文字列) 仮想ファイルの名前。これは任意の文字列であり、以後、他の PLOP 読み出しの中でこの仮想ファイルを参照するために用いることができます。

**len** (C 言語バインディングのみ) **filename** の UTF-16 文字列に対する長さ (バイト単位)。**len=0** の場合、ヌル終端文字列を与える必要があります。

**data** 仮想ファイルにしたいデータへの参照。COM の場合、これは仮想ファイルを構成するデータがあるバイトのバリエーション型です。C・C++ の場合、これはメモリ位置へのポインタです。Java の場合、これはバイト配列です。Perl・PHP の場合、これは文字列です。

**size** (C・C++ のみ) データを含むメモリブロックのデータ長をバイト単位で表したものの。

**optlist** 表 7.1 に従ったオプションリスト。次のオプションが使えます：*copy*。

**詳細** この関数は、繰り返し使用される電子 ID や XMP メタデータのために有用でしょう。仮想ファイルは、入力ファイルを用いるあらゆる API 関数に与えることができます。ここの関数のなかには、データが必要なくなるまで仮想ファイルにロックをかけるものもあります。仮想ファイルは、*PLOP\_delete\_pvf()* で明示的に、または *PLOP\_delete()* で自動的に削除されるまでメモリ上に保持されます。

PLOP オブジェクトはそれぞれ、独自の PVF ファイルの集合を保持します。仮想ファイルは、異なる PLOP オブジェクト間で共有することはできません。別々の PLOP オブジェクトを使用しているマルチスレッドは、PVF の使用を同期する必要はありません。*filename* が既存の仮想ファイルを参照しているときは、例外が発生します。この関数は、*filename* がディスク上の通常のファイルですでに使用されているかどうかはチェックしません。

*copy* オプションを与えていない限り、対になる *PLOP\_delete\_pvf()* への呼び出しが成功するまでは、与えたデータを呼び出し側で変更したり解放（削除）したりしてはいけません。このルールに従わないと、クラッシュする可能性が高いです。

表 7.1 *PLOP\_create\_pvf()* に対するオプション一覧

オプション	説明
-------	----

<i>copy</i>	(論理値) PLOP は、与えられたデータの内部コピーをただちに作ります。この場合、与えたデータをこの呼び出しの直後に呼び出し側で捨ててもかまいません。COM・.NET・Java バインディングの場合、 <i>copy</i> オプションは自動的に <i>true</i> に設定されます（それ以外のバインディングでのデフォルト： <i>false</i> ）。それ以外の言語バインディングでは、 <i>copy</i> オプションを与えなければデータはコピーされません。
-------------	--

---

**C++** *int delete\_pvf(string filename)*

**C# Java** *int delete\_pvf(String filename)*

**Perl PHP** *int delete\_pvf(string filename)*

**VB** *Function delete\_pvf(filename As String) As Long*

**C** *int PLOP\_delete\_pvf(PLOP \*plop, const char \*filename, int len)*

---

名前付きの仮想ファイルを削除し、そのデータ構造を解放します（ただし内容は解放しません）。

**filename** (名前文字列) *PLOP\_create\_pvf()* に与えたのと同じ、仮想ファイルの名前。

**len** (C 言語バインディングのみ) *filename* の UTF-16 文字列に対する長さ（バイト単位）。*len=0* の場合、ヌル終端文字列を与える必要があります。

**戻り値** 対応する仮想ファイルが存在しているがロックされているときは -1（PHP では 0）、それ以外のときは 1。

**詳細** ファイルがロックされていなければ、PLOP はただちに、*filename* に関連付けられていたデータ構造を削除します。*filename* が有効な仮想ファイルを参照していないときは、この

関数は無言のまま何もしません。この関数への呼び出しが成功した後は、*filename* は再利用することもできます。すべての仮想ファイルは *PLOP\_delete()* で自動的に削除されます。

具体的な動作は、対応する *PLOP\_create\_pvf()* を呼び出したときに *copy* オプションを与えていたかどうかで異なります。すなわち、*copy* オプションを与えていた場合は、ファイルの管理データ構造もファイル内容自体（データ）も両方解放されますが、そうでなかった場合は内容は、クライアント側で解放されるものと思われるので解放されません。

---

**C++** *double info\_pvf(string filename, string keyword)*

**C# Java** *double info\_pvf(String filename, String keyword)*

**Perl PHP** *float info\_pvf(string filename, string keyword)*

**VB** *Function info\_pvf(filename As String, keyword As String) As Double*

**C** *double PLOP\_info\_pvf(PDF \*p, const char \*filename, int len, const char \*keyword)*

---

仮想ファイルか PDFlib 仮想ファイルシステム (PVF) の諸特性を取得します。

**filename** (名前文字列) 仮想ファイルの名前。 *keyword=filecount* のときは、*filename* は空にすることができます。

**len** (C 言語バインドイングのみ) *filename* の UTF-16 文字列に対する長さ (バイト単位)。 *len=0* の場合、ヌル終端文字列を与える必要があります。

**keyword** 表 7.2 に従ったキーワード。

表 7.2 *PLOP\_info\_pvf()* に対するキーワード一覧

キーワード	説明
<i>filecount</i>	カレント PLOP オブジェクトのために保持されている PDFlib 仮想ファイルシステムの中のファイルの総数。 <i>filename</i> 引数は無視されます。
<i>exists</i>	そのファイルが PDFlib 仮想ファイルシステム内に存在するなら (かつ削除されていないなら) 1、しないなら 0
<i>size</i>	(存在している仮想ファイルに対してのみ) 指定した仮想ファイルのサイズをバイト単位で。
<i>iscopy</i>	(存在している仮想ファイルに対してのみ) 指定した仮想ファイルが作成された際に <i>copy</i> オプションが与えられたなら 1、そうでないなら 0。
<i>lockcount</i>	(存在している仮想ファイルに対してのみ) 指定した仮想ファイルに対して PLOP 関数群によって内部的に設定されたロックの数。ロックカウントが 0 のときにのみファイルは削除できます。

**詳細** この関数は、仮想ファイルか PDFlib 仮想ファイルシステム (PVF) のさまざまな特性を返します。特性はキーワードで指定されます。

## 7.3 文書入力・出力関数

注記 PLOP では現状、1 個の PLOP オブジェクトの中で同時に複数の文書进行处理することには対応していません。1 個の文書を `PLOP_open_document*()` 関数のうちのいずれかで開いた後に、他の文書を開きたい場合は、まず前者を閉じる必要があります。

---

**C++** `int open_document(string filename, string optlist)`  
**C# Java** `int open_document(String filename, String optlist)`  
**Perl PHP** `int open_document(string filename, string optlist)`  
**VB** `Function open_document(filename As String, optlist As String) As Long`  
**C** `int PLOP_open_document(PLOP *plop, const char *filename, int len, const char *optlist)`

---

PDF 文書（保護されているかもしれない）进行处理するために開きます。

**filename**（名前文字列。ただし Unicode ファイル名に対応しているのは Windows 上のみ）開きたい PDF ファイルのフルパス名。Windows の場合、必要な権限があれば（ASP で動作している場合はないかもしれない）、UNC パスまたは割り当てられたネットワークドライブも使えます。

Unicode 非対応の言語バインディングの場合、`len=0` のファイル名はカレントシステムコードページで解釈されますが、ただし UTF-8 BOM が頭についているときは、UTF-8 または EBCDIC UTF-8 として解釈されます。

**len**（C 言語バインディングのみ）**filename** の UTF-16 文字列に対する長さ（バイト単位）。`len=0` の場合、ヌル終端文字列を与える必要があります。

**optlist** 表 7.3 に従ったオプションリスト（7.1 節「オプションリスト」（73 ページ）参照）。

**戻り値** エラーの場合は `-1`（PHP では `0`）、そうでないなら文書ハンドル。エラーの後は、`PLOP_get_errmsg()` を呼び出して、そのエラーについてより詳しく知ることを推奨します。

**詳細** 文書が暗号化されている場合は、そのユーザーパスワードかマスターパスワードを `password` オプションで与える必要があります。ただし、`requiredmode` オプションが指定されている場合はこの限りではありません。

表 7.3 `PLOP_open_document*()` に対するオプション一覧

オプション	説明
<code>inmemory</code>	（論理値。 <code>PLOP_open_document()</code> のみ） <code>true</code> の場合、PLOP はファイル全体をメモリ内に読み込んで、そこでそれを処理します。これはシステムによっては（特に MVS）非常にパフォーマンス向上につながりますが、かわりにメモリを食います。 <code>false</code> の場合、文書の部分部分が必要に応じて都度都度ディスクから読まれます。デフォルト： <code>false</code>
<code>password</code>	（文字列。暗号化された文書に対しては、 <code>requiredmode</code> がある場合以外は必須）文書のユーザーパスワードかマスターパスワード。表 4.2（53 ページ）で述べたように、文書にどの操作をしたいかによって、その文書のユーザーパスワードが必要か、マスターパスワードが必要か、それともパスワードが必要ないかが決まります。EBCDIC プラットフォームではパスワードは <code>ebcdic</code> エンコーディングか EBCDIC-UTF-8 で与える必要があります。

表 7.3 PLOP\_open\_document\*() に対するオプション一覧

オプション	説明
<b>repair</b>	(キーワード) 破損した PDF 入力文書をどう扱うかを指定します。文書を修復すると通常の処理より時間がかかりますが、ある種の破損 PDF の処理ができるようになる可能性があります。ただし文書によっては、修復できないほど破損していることもありえます (デフォルト: <i>auto</i> ): <b>force</b> 文書に問題があろうとなかろうと、無条件で文書の修復を試みます。 <b>auto</b> PDF を開く際に問題が検出された場合のみ文書を修復します。 <b>none</b> 文書を修復する試みは行われません。PDF 内に問題があった場合は、関数呼び出しは失敗します。
<b>requiredmode</b>	(キーワード) 文書を開く際に受け入れ可能な最低限の pCOS モード ( <i>minimum/restricted/full</i> )。求めた pCOS モードより結果の pCOS モードが低かったときは、呼び出しは失敗します。呼び出しが成功した場合、結果の pCOS モードは少なくともこのオプションで指定したものであることが保証されます。ただし、それより高い可能性もあります。たとえば、暗号化されていない文書に対して <i>requiredmode=minimum</i> を指定した場合、結果は <i>full</i> モードになります。デフォルト: <i>full</i>
<b>xmppolicy</b>	(キーワード) 入力文書内の無効な文書レベル XMP の扱いを制御します。無効な XMP は、標準識別子を見つけることができないことを暗黙に前提しますので、たとえば PDF/A 文書がそれとして扱われません。使えるキーワード (デフォルト: <i>rejectinvalid</i> ): <b>rejectinvalid</b> 無効な XMP の場合には、XML 解析エラーメッセージを含む例外を発生させ、処理を停止させます。 <b>ignoreinvalid</b> ( <i>sacrifice={pdfa1 pdfx}</i> ) を暗黙に前提します) 無効を、XMP が存在しないかのように扱います。出力 XMP は、文書情報項目に基づき生成されます。また、XML 解析エラーメッセージを <i>&lt;pdfx:invalid_source_XMP_exception&gt;</i> 要素内に入れ込みます。 <b>remove</b> 入力 XMP を、有効であってもなくても無条件に無視します。出力 XMP は一から生成されます。これは、望ましくないメタデータを削除するのに有用でしょう。ただしこの場合も、標準識別子 (PDF/A などの) は入力 XMP から読み込まれて出力へ複製されます。

---

```

C++ int open_document_callback(void *opaque, size_t filesize,
                               size_t (*readproc)(void *opaque, void *buffer, size_t size),
                               int (*seekproc)(void *opaque, long offset), const char *optlist)
C   int PLOP_open_document_callback(PLOP *plop, void *opaque, size_t filesize,
                                   size_t (*readproc)(void *opaque, void *buffer, size_t size),
                                   int (*seekproc)(void *opaque, long offset), const char *optlist)
    
```

---

PDF 文書を (保護されているかもしれない)、ユーザーが与えた関数で開きます。

**opaque** 何らかの不透明なデータ構造へのポインタ。readproc へ渡されます。PLOP はこのポインタやその背後のデータを使いません。

**filesize** 文書の長さをバイト単位で。

**readproc** メモリ位置 *buffer* にある文書の任意の *size* バイトの切れ端を与えることのできななければならないプロシージャ。このプロシージャは、取得したバイト数を返さなければなりません。

**seekproc** 文書内の位置 *offset* ヘシークするためのプロシージャ。このプロシージャはエラーが起きたら -1 を、そうでないなら 0 を返さなければなりません。

**optlist** 表 7.3 に従ったオプションリスト (7.1 節「オプションリスト」(73 ページ) 参照)。

戻り値 エラーの場合は -1 (PHP では 0)、そうでないなら文書ハンドル。エラーの後は、`PLOP_get_errmsg()` を呼び出して、そのエラーについてより詳しく知ることを推奨します。

バインディング C・C++ 言語バインディングでのみ利用可能です。

---

**C++** `int create_file(string filename, string optlist)`

**C# Java** `int create_file(String filename, String optlist)`

**Perl PHP** `int create_file(string filename, string optlist)`

**VB** `Function create_file(filename As String, optlist As String) As Long`

**C** `int PLOP_create_file(PLOP *plop, const char *filename, int len, const char *optlist)`

---

PDF 出力文書 (保護されているかもしれない) を、メモリ内またはディスクファイル上に作成します。

**filename** (名前文字列。ただし Unicode ファイル名に対応しているのは Windows 上のみ) 生成したい出力ファイルの名前。`PLOP_open_document()` に与えた入力ファイル名とは異なっている必要があります。これが空文字列のときは出力はメモリ内に生成され、後で `PLOP_get_buffer()` で取り出せます。MVS システムの場合、空文字列は `linearize` オプションと組み合わせては使えません。

Unicode 非対応の言語バインディングの場合、`len=0` のファイル名はカレントシステムコードページで解釈されますが、ただし UTF-8 BOM が頭についているときは、UTF-8 または EBCDIC UTF-8 として解釈されます。

**len** (C 言語バインディングのみ) **filename** の UTF-16 文字列に対する長さ (バイト単位)。`len=0` の場合、ヌル終端文字列を与える必要があります。

**optlist** 表 7.4 に従ったオプションリスト (7.1 節「オプションリスト」(73 ページ) 参照)。

戻り値 エラーの場合は -1 (PHP では 0)、そうでないなら文書ハンドル。エラーの後は、`PLOP_get_errmsg()` を呼び出して、そのエラーについてより詳しく知ることを推奨します。

**詳細** この関数を呼び出す前には、`PLOP_open_document*()` 関数のいずれかが呼び出されている必要があります。もっとも最近のこれらの関数のいずれかへの呼び出しで開かれた文書が処理されます。ユーザーパスワードとマスターパスワードについて強えられる制約条件については 4.2 節「PLOP の PDF セキュリティ機能」(53 ページ) を参照してください。

`userpassword`・`masterpassword`・`permissions` のいずれかのオプションが与えられているときは、文書は暗号化されます。暗号化アルゴリズムは、入力文書の PDF バージョンと `compatibility` オプションに基づき選ばれます (「暗号化アルゴリズムとキー長」(53 ページ) を参照)。

表 7.4 PLOP\_create\_file() に対するオプション一覧

オプション	説明
<b>compatibility</b>	<p>(キーワード) 生成される PDF 出力文書の PDF バージョンを指定します :</p> <p>1.4 PDF 1.4. Acrobat 5 以上を必要とします。</p> <p>1.5 PDF 1.5. Acrobat 6 以上を必要とします。</p> <p>1.6 PDF 1.6. Acrobat 7 以上を必要とします。</p> <p>1.7 PDF 1.7. ISO 32000-1 で仕様化されており、Acrobat 8 以上を必要とします。</p> <p>1.7ext3 PDF 1.7 拡張レベル 3. Acrobat 9 以上を必要とします。</p> <p>1.7ext8 PDF 1.7 拡張レベル 8. Acrobat X を必要とします。</p> <p>2.0 PDF 2.0. ISO 32000-2 で仕様化されています。</p> <p>出力を暗号化する場合、これを用いて、適切な暗号化アルゴリズムが選択されます。選択された PDF バージョンが対応している可能な限り強い暗号化アルゴリズムが用いられます (AES 暗号化を強いるには 1.6 を用います)。選択した PDF バージョンは、他のオプションによって、以下の規則に従って自動的に押し上げられる場合があります :</p> <ul style="list-style-type: none"> <li>▶ 文書に電子的に署名すると (sign オプション)、バージョンは PDF 1.3 へ押し上げられます。</li> <li>▶ 暗号化、すなわち <i>userpassword</i>・<i>masterpassword</i>・<i>permissions</i> オプションのうちのいずれかを指定すると、バージョンは PDF 1.4 へ押し上げられます。</li> <li>▶ XMP メタデータを挿入すると (<i>metadata</i> オプション)、バージョンは PDF 1.4 へ押し上げられます。</li> <li>▶ <i>permissions</i> オプションで <i>plainmetadata</i> キーワードを指定すると、バージョンは PDF 1.5 へ押し上げられます。</li> </ul> <p>デフォルト : 入力文書の PDF バージョン、または上述の規則によって強制される上位バージョン。</p>
<b>docinfo</b>	<p>(テキスト文字列の対のリスト) 出力文書の文書情報項目群を設定します。文書に文書 XMP メタデータがある場合は、与えた文書情報項目は XMP へもミラーされます。それぞれの対には、項目の名前とその値が入っています。以下の定義済みキーとカスタムキーを与えることができます (デフォルト : 文書情報項目は入力文書からコピーされます) :</p> <p><b>Subject</b> 文書のサブタイトル</p> <p><b>Title</b> 文書のタイトル</p> <p><b>Author</b> 文書の作成者</p> <p><b>Keywords</b> 文書の内容を表すキーワード</p> <p><b>Trapped</b> 文書にトラッピングが適用されているかどうかを示します。許される値は <i>True</i>・<i>False</i>・<i>Unknown</i> です。PDF/X 入力の場合、<i>sacrifice</i> に <i>pdfx</i> があるなら、<i>Unknown</i> のみが許されます。</p> <p><b>Creator</b>・<b>CreationDate</b>・<b>Producer</b>・<b>ModDate</b>・<b>GTS_PDFXVersion</b>・<b>GTS_PDFXConformance</b>・<b>ISO_PDFEVersion</b> 以外の任意の名前</p> <p>ユーザー定義のフィールド名 (スペースキャラクタを含んではいけません)。PLOP は任意の数のフィールドに対応しています。1 個のカスタムフィールド名は 1 度だけ与える必要があります。</p>
<b>flush</b>	<p>(キーワード) 放出方針を設定します。これはメモリ内生成 (すなわち空の filename) に対してのみ有効であり、<i>PLOP_get_buffer()</i> によって返されるデータの量を決定します。<i>linearize</i> オプションが <i>true</i> のときは、放出方針は <i>none</i> にする必要があります (デフォルト : <i>none</i>) :</p> <p><b>none</b> 返されるバッファは、出力文書を構成するすべてのデータを含んでいることが保証されます。</p> <p><b>content</b> <i>PLOP_get_buffer()</i> は、PDF 内容データの比較的大きなかたまり (具体的には 1 個の PDF ストリームオブジェクト) が処理されるたびごとに停止し、返されるバッファには出力文書の部分部分しか入っていません。</p> <p><b>heavy</b> <i>PLOP_get_buffer()</i> は比較的小さな分量ずつを処理し、したがって比較的小いぶんばんに呼び出されます。</p>
<b>linearize</b>	<p>(論理値。sign または <i>linearize</i> とともに使うことはできません) <i>true</i> の場合、出力文書は線形化されます。MVS システムの場合、このオプションはメモリ内生成 (すなわち空の filename) と組み合わせることはできません。デフォルト : <i>false</i></p>

表 7.4 PLOP\_create\_file() に対するオプション一覧

オプション	説明
<b>master-password<sup>1</sup></b>	(文字列) 文書のマスターパスワード。これが空の場合、マスターパスワードは適用されません。EBCDIC プラットフォームの場合、パスワードは <i>ebcdic</i> エンコーディングか EBCDIC-UTF-8 で与える必要があります。デフォルト : 空
<b>metadata</b>	(オプションリスト。linearize と組み合わせることは不可) 文書の XMP メタデータを与えます。PDF/A-1・PDF/X 識別項目は、この与える XMP の中では許されません。このオプションリストには以下のオプションを含むことができます : <b>filename</b> (名前文字列。必須) 妥当な XMP メタデータを UTF-8 形式で含むファイルの名前。 <b>validate</b> (キーワード) XMP メタデータはキーワードによって検証されます : <i>none</i> 検証なし <i>xmp2004</i> XMP 2004 仕様に従って検証 <i>pdfa1</i> <i>xmp2004</i> と同様ですが、それに加えて定義済みプロパティとスキーマの試験と、PDF/A に従って拡張スキーマの検証も行います。 デフォルト : <i>none</i> 。ただし入力が PDF/A-1 に準拠しているかつ <i>sacrifice</i> オプションに <i>pdfa1</i> が含まれていない場合は <i>pdfa1</i> が強制されます。
<b>optimize</b>	(キーワード) 文書を処理する際に適用したい最適化処置 (デフォルト : <i>sign</i> オプションが与えられている場合は <i>none</i> 、そうでないなら <i>all</i> ) : <b>all</b> 実装されているすべての最適化を適用。 <b>none</b> 最適化を一切適用しない。これは若干速度を向上させますが、そのかわりファイルサイズは大きくなります。
<b>permissions</b>	(キーワードリスト。masterpassword が必要) 出力文書のアクセス権限リスト。noprint・nomodify・nocopy・noannots・noassemble・noforms・noaccessible・nohiresprint・plainmetadata キーワードを任意の数含むことができます (表 4.3 (54 ページ) 参照)。デフォルト : 空
<b>recordsize</b>	(整数。MVS のみ) 出力ファイルのレコードサイズ。デフォルト : 0 (非ブロック出力)
<b>sacrifice</b>	(キーワードのリスト) このオプションを用いると、入力 PDF の特性と求められた操作とが衝突した場合の動作を制御することができます。デフォルトでは PLOP は、衝突を検出したときには一切出力を生成せず、例外を発生させます。しかし、処理を許すために、文書の何らかの特質を放棄させることが可能です。表 7.5 に挙げるキーワードに対応しています : これらは、入力トリガと操作トリガが両方とも真でない限り無視されます (デフォルト : 空のリスト、すなわち衝突が起きた場合は例外が発生し、出力は一切生成されない) :
<b>sign</b>	(オプションリスト。linearize と組み合わせて使うことはできません。PLOP DS でのみ利用可能) 生成文書に、表 7.6 に挙げるサブオプションに従って署名します。
<b>tempdirname</b>	(文字列) PLOP の内部処理に必要な一時ファイルが作成されるディレクトリの名前。空の場合、PLOP は一時ファイルをカレントディレクトリに生成します。このオプションは、tempfilename オプションが与えられているときは無視されます。デフォルト : 空
<b>tempfilename</b>	(文字列。MVS のみ) PLOP の内部処理に必要な一時ファイルのフルファイル名。空の場合、PLOP は一意な一時ファイル名を生成します。PLOP_close_document() の後でこの一時ファイルを削除するのはユーザー側の役割です。このオプションを与えた場合、filename 引数は空にしておくべきです。デフォルト : 空
<b>user-password<sup>1</sup></b>	(文字列。masterpassword オプションが必要) 文書のユーザーパスワード。これが空の場合、ユーザーパスワードは適用されません。EBCDIC プラットフォームの場合、パスワードは <i>ebcdic</i> エンコーディングか EBCDIC-UTF-8 で与える必要があります。デフォルト : 空

1. Winansi エンコーディング外のキャラクタは、PDF 1.7 拡張レベル 3 以上を生成する場合にのみ許されます。

表 7.5 PLOP\_create\_file() の sacrifice オプションに対するサブオプション一覧

オプション	説明
<b>encrypted-attachments</b>	(入力トリガ: 文書が暗号化されていないが、暗号化されたファイル添付を 1 個ないし複数含んでいる。操作トリガ: 暗号化されたファイル添付に対する適切なパスワードが password オプションで与えられていない) このキーワードを与えると、パスワードの得られない暗号化されたファイル添付は削除されます。
<b>fields</b>	(入力トリガ: 文書の中に、NeedAppearances=true のフォームフィールドがある。操作トリガ: sign オプション) このキーワードを与えると、既存のフォームフィールドは削除されます (既存の署名フィールドがあるときはそれも)。
<b>pdfa1</b>	(入力トリガ: 文書が PDF/A-1a:2005 か PDF/A-1b:2005 に準拠している。操作トリガ: userpassword・masterpassword・permissions オプションのうちのいずれか) このキーワードを与えると、PDF/A-1 入力を暗号化することができますが、PDF/A-1 準拠項目は削除されます (すなわち、出力は PDF/A-1 であると標識されなくなります)。
<b>pdfx</b>	(入力トリガ: 文書が PDF/X-1a か PDF/X-3/4/5 に準拠している。操作トリガ: ページ上の署名矩形を持つ sign オプション、または userpassword・masterpassword・permissions オプションのうちのいずれか) このキーワードを与えると、暗号化や、可視署名フィールドを BleedBox (BleedBox が存在しないときは TrimBox/ArtBox) 内に追加することが許されますが、PDF/X 準拠項目は削除されます (すなわち、出力は PDF/X であると標識されなくなります)。
<b>signatures</b>	(入力トリガ: 文書の中に 1 個ないし複数の署名がある。操作は任意) このキーワードを与えると、無効な署名の入った出力を作成することを避けるために、既存の署名はクリアされます (すなわち、署名の値は削除されますが、対応するフォームフィールドは削除されません)。

表 7.6 PLOP\_create\_file() の sign オプションに対するサブオプション一覧 (PLOP DS でのみ利用可能)

オプション	説明
<b>appearance</b>	(オプションリスト) 署名を保持するフォームフィールドの視覚表現を指定します :
<b>fieldname</b>	(テキスト文字列:ピリオド「.」キャラクタで終わってはいけません) 署名フィールドの名前。文書の中にこの名前の署名フィールドがある場合は、それが署名のために用いられ (この場合 <i>page</i> ・ <i>rect</i> は無視されます)、そうでないならフィールドが作成されず。この名前のフィールドが存在しているけれども、種別が <i>Signature</i> でなかった場合は、例外が発生します。デフォルト : <i>Signature1</i>
<b>page</b>	(整数) 署名フィールド (可視であろうと不可視であろうと) が作成されるページの番号。最初のページの番号は 1。デフォルト : 1
<b>rect</b>	(矩形) 署名フィールドの左下隅と右上隅の座標を PDF 座標で表したもの (1 単位は 1/72 インチ)。デフォルト : { <i>o o o o</i> }、この値で不可視署名を作成されます
<b>contactinfo</b>	(テキスト文字列) 受取側が署名を検証するために署名者に連絡をとれるようにするために署名者によって与えられる情報 (電話番号等)
<b>digitalid</b>	(オプションリスト。必須) 署名者のデジタル ID (証明書と秘密鍵) を、以下のサブオプションのいずれか 1 つのみで指定します :
<b>filename</b>	(文字列) PKCS#12 形式 ( <i>engine=builtin</i> または CE インタフェースの場合のみ) または PFX 形式のデジタル ID ファイルの名前。PKCS#12・PFX ファイルは、仮想ファイルとして、すなわちメモリデータに <i>PLOP_create_pvf()</i> でファイル名を割り当てて、与えることもできます。 <i>engine=pkcs#11</i> の場合は、このオプションには、スマートカード等暗号トークン用の PKCS#11 DLL/ 共有ライブラリの名前を与えます。
<b>certstore</b>	(オプションリスト。 <i>engine=miscapi</i> の場合のみ) Windows の証明書ストアの中の ID を指し示すためのオプション :
<b>subject</b>	(文字列。必須) 与えた文字列が「サブジェクト」項目の中にある ID を探します。それは通常、デジタル ID の「共通名」(CN) 項目を保持しています。
<b>store</b>	(文字列) 証明書ストアの名前 (よくある名前 : <i>My</i> ・ <i>root</i> ・ <i>trust</i> ・ <i>CA</i> )。デフォルト : <i>My</i>
<b>keyusage</b>	(オプションリスト。 <i>engine=pkcs#11</i> の場合のみ) 複数の ID が (スマートカード等に) 存在しているときにターゲット ID を選ぶための基準。キーワードはそれぞれ、 <i>KeyUsage</i> 認証拡張の中のビット 1 個に対応しています。各キーワードの値はそれぞれ、ID を選ぶときにその拡張ビットが 1 であるべき ( <i>set</i> )、0 であるべき ( <i>clear</i> )、あるいは無視するべき ( <i>ignore</i> ) ということを指定します。PLOP DS は、この指定された基準に合う ID を使用します。合う ID が見つからなかったときは例外が発生します。次のキーワードが使えます : <i>clear</i> ・ <i>ignore</i> ・ <i>set</i> 。どの項目もデフォルトは <i>ignore</i> です。
<b>digitalsignature</b>	(キーワード) <i>digitalsignature</i> キー用途拡張 (すなわちビット 0) の扱いを指定するキーワード <i>clear/ignore/set</i> のうち 1 つ。
<b>nonrepudiation</b>	(キーワード) <i>nonrepudiation</i> キー用途拡張 (すなわちビット 1) の扱いを指定するキーワード <i>clear/ignore/set</i> のうち 1 つ。
<b>engine</b>	(キーワード) 電子署名に使用する暗号化エンジンを指定します (デフォルト : <i>builtin</i> ) :
<b>builtin</b>	内蔵の暗号化エンジンを使用します。デジタル ID はディスクファイル (PFX または PKCS#12) から取ってくる必要があります。
<b>miscapi</b>	(Windows のみ) Microsoft Crypto API を暗号化エンジンとして使用します。デジタル ID は証明書ストアからもディスクファイル (PFX のみ) から取ってこられます。
<b>pkcs#11</b>	(一部プラットフォームのみ) PKCS#11 インタフェースを用いて暗号トークンから証明書を取得します。対応する PKCS#11 DLL/ 共有ライブラリの名前を、 <i>digitalid</i> オプションの <i>filename</i> サブオプションで与えている必要があります。

表 7.6 PLOP\_create\_file() の sign オプションに対するサブオプション一覧 (PLOP DS でのみ利用可能)

オプション	説明
<b>location</b>	(テキスト文字列) 署名の行われた物理的な場所
<b>password</b>	(文字列。空でも可。engine=builtin の場合は、password か passwordfile のどちらか 1 つだけが必要。他のエンジンの場合は代替方式が使える可能性があります) デジタル ID に対するパスワードないしパスフレーズまたは PIN を指定。engine=pkcs#11 の場合は、このオプションに暗号トークンのための PIN を与える必要がありますが、ただし PIN をトークン本体に対話的に入力する必要のある場合はこの限りではありません (キーボード付きスマートカードリーダー等)。EBCDIC プラットフォームの場合、パスワードは ebcdic エンコーディングであると見なされます。
<b>passwordfile</b>	(文字列。engine=builtin の場合は、password か passwordfile のどちらか 1 つだけが必要。他のエンジンの場合は代替方式が使える可能性があります) ファイルの 1 行目 (ラインエンドキャラクタ (複数可) を除く) が、デジタル ID に対するパスワードないしパスフレーズまたは PIN として用いられます。EBCDIC プラットフォームの場合、パスワードファイルの内容は ebcdic エンコーディングであると見なされます。
<b>reason</b>	(テキスト文字列) 文書に署名した理由
<b>subfilter</b>	(キーワード) PDF 署名の種類 (デフォルト : adbe.pkcs7.detached) : <b>adbe.pkcs7.detached</b> PKCS#7 署名済データフィールド内にいかなるデータもカプセル化されません。この方式は動的な文書変更に対応しますので、たとえば日付フィールドに JavaScript コードで記入したりすることが可能です。 <b>adbe.pkcs7.sha1</b> PKCS#7 署名済データフィールド内にデータの SHA-1 ダイジェストがカプセル化されます。この方式は動的な文書変更に対応しません。

---

**C++** `const char *get_buffer(long *size)`

**C# Java** `byte[] get_buffer()`

**Perl PHP** `string get_buffer()`

**VB** `Function get_buffer() As Variant`

**C** `const char *PLOP_get_buffer(PLOP *plop, long *size)`

---

出力文書の内容をメモリから全部または一部取り出します。

**size** C バインディングでのみ必須。返されるバッファの長さが格納されるメモリ位置へのポインタ。

**戻り値** 出力データの入ったバッファ。COM の場合、これは符号なしバイトのバリエーション配列です。JavaScript で COM を使う場合、返されたバリエーション配列の長さを取得することは許されていません（ただし、それ以外の言語で COM を使う場合は可能です）。クライアント側では、他のいかなる PLOP ライブラリ関数を呼ぶよりも前に、このバッファ内容を消費する必要があります。

**詳細** `PLOP_create_file()` に空のファイル名を与えることによってメモリ内生成を要求してあった場合は（そうでないなら出力はファイルへ直接書き出されます）、PDF 出力はこの関数によってのみ取り出すことができます。`PLOP_get_buffer()` は、`PLOP_close_document()` を呼び出すよりも前に呼び出す必要があります。

`PLOP_create_file()` の `flush` オプションがそのデフォルト値 `none` を持っている場合、返されたバッファには、出力文書のすべてのデータが入っていることが保証されています。`flush=content` の場合、`PLOP_get_buffer()` は、PDF 内容データの比較的大きなかたまり（具体的には 1 個の PDF ストリームオブジェクト）が処理されるたびごとに停止し、返されるバッファには出力文書の部分部分しか入っていません。`flush=heavy` の場合、この関数は比較的小さな分量ずつを処理し、したがって比較的ひんぱんに呼び出されます。

`flush=none` でない限り、`PLOP_get_buffer()` は、空のバッファを返すまで繰り返し呼び出す必要があります。クライアント側では、完全な出力文書を生成するためには、返された部分部分を連結していく必要があります。`flush=none` の場合は、`PLOP_get_buffer()` は 1 回呼び出すだけで充分です。

---

**C++** `void close_document(int doc)`

**C# Java** `close_document(int doc)`

**Perl PHP** `close_document(long doc)`

**VB** `Sub close_document(doc As Long)`

**C** `void PLOP_close_document(PLOP *plop, int doc)`

---

入力・出力文書を閉じます。

**doc** `PLOP_open_document*()` で得られた有効な文書ハンドル。

**詳細** この関数は、処理が完了した時に、かつ `PLOP_delete()` を呼び出すより前に、クリーンアップのために呼び出す必要があります。

## 7.4 例外処理

PLOP では、ライブラリの例外を C 言語で取り扱うための追加のメソッドを提供しています。それ以外の PLOP の言語バインディングでは、それぞれの言語のネイティブの例外処理システムを利用しています (*try/catch* 節等)。言語ラップは、生成される例外オブジェクトの中に、例外の番号・説明・API 関数名に関する情報を入れ込みます。Java 言語バインディングの場合、こうした項目は個別に取得することができます。

PLOP 例外が発生した時には、その PLOP オブジェクトについては *PLOP\_delete()* 以外の PLOP 関数は一切呼び出してはいけません。

Java と .NET 用の PLOP 言語バインディングでは別途、*PLOPEXception* オブジェクトを定義しており、これは詳細なエラー情報にアクセスするためのメンバをいくつか提供しています。

---

**C++** *int get\_errnum()*

**C# Java** *int get\_errnum()*

**Perl PHP** *int get\_errnum()*

**VB** *Function get\_errnum() As Long*

**C** *int PLOP\_get\_errnum(PLOP \*plop)*

---

もっとも最近に発生した例外、ないし失敗した関数呼び出しの原因の番号を得ます。

**戻り値** 例外のエラー番号。

**バインディング** .NET の場合、このメソッドは *PLOPEXception* オブジェクトの中の *Errnum* としても利用可能です。

Java の場合、このメソッドは *PLOPEXception* オブジェクトの中の *get\_errnum()* としても利用可能です。

---

**C++** *string get\_errmsg()*

**C# Java** *String get\_errmsg()*

**Perl PHP** *string get\_errmsg()*

**VB** *Function get\_errmsg() As String*

**C** *const char \*PLOP\_get\_errmsg(PLOP \*plop)*

---

もっとも最近に発生した例外、ないし失敗した関数呼び出しの原因の説明テキストを得ます。

**戻り値** エラーを説明する文字列、またはもっとも最近の API 呼び出しが何らエラーを発生させなかった場合は空文字列。

**バインディング** .NET の場合、このメソッドは *PLOPEXception* オブジェクトの中の *Errmsg* としても利用可能です。

Java の場合、このメソッドは *PLOPEXception* オブジェクトの中の *getMessage()* としても利用可能です。

---

**C++** `string get_apiname()`

**C# Java** `String get_apiname()`

**Perl PHP** `string get_apiname()`

**VB** `Function get_apiname() As String`

**C** `const char *PLOP_get_apiname(PLOP *plop)`

---

もっとも最近の例外を発生させた、ないし失敗した API 関数の名前を得ます。

戻り値 PLOP API 関数の名前。

バインディング .NET の場合、このメソッドは `PLOPException` オブジェクトの中の `Apiname` としても利用可能です。

Java の場合、このメソッドは `PLOPException` オブジェクトの中の `get_apiname()` としても利用可能です。

---

**C** `PLOP_TRY(PLOP *plop)`

---

例外処理フレームをセットアップします。かならず `PLOP_CATCH()` と対にする必要があります。

詳細 「例外処理」 (33 ページ) 参照。

---

**C** `PLOP_CATCH(PLOP *plop)`

---

例外をキャッチします。かならず `PLOP_TRY()` と対にする必要があります。

詳細 「例外処理」 (33 ページ) 参照。

---

**C** `PLOP_EXIT_TRY(PLOP *plop)`

---

`PLOP_TRY()` の中から、対応する `PLOP_CATCH()` 節へ入ることなく抜けることを、例外機構に通知します。

詳細 「例外処理」 (33 ページ) 参照。

---

**C** `PLOP_RETHROW(PLOP *plop)`

---

例外を他のハンドラへ投げなおします。

詳細 「例外処理」 (33 ページ) 参照。

## 7.5 オプション処理

**C++** `void set_option(string optlist)`

**C# Java** `void set_option(String optlist)`

**Perl PHP** `set_option(string optlist)`

**VB** `Sub set_option(optlist As String)`

**C** `void PLOP_set_option(PLOP *plop, const char *optlist)`

PLOP のための 1 つないし複数のグローバルオプションを設定します。

**optlist** 表 7.7 に従ってグローバルオプションを指定するオプションリスト。1 つのオプションが複数回与えられた場合、最後に出てきたものがそれより前のすべてを上書きします。1 つのオプション (*searchpath* 等) に対して複数の値を与えたいときは、すべての値を 1 つのリスト引数にしてこのオプションに与えます。

**詳細** 表 7.7 で特記してあるオプションについては、この関数を複数呼び出すことで値を蓄積させることができます。特記していないオプションについては、新しい値が古い値を上書きします。

表 7.7 PLOP\_set\_option() に対するグローバルオプション一覧

オプション	説明
<b>filename-handling</b>	(キーワード。Windows では必須ではありません) ファイル名に対するターゲットエンコーディング。Windows ではこのオプションは与えられたファイル名には適用されますが、生成されるファイルの名前には適用されません (デフォルト: Mac OS X では <i>unicode</i> 、それ以外では <i>honolang</i> ): <b>ascii</b> 7 ビット ASCII <b>basicebcdic</b> コードページ 1047 に従った基本 EBCDIC、ただし Unicode 値 <= U+007E のみ <b>basicebcdic_37</b> コードページ 0037 に従った基本 EBCDIC、ただし Unicode 値 <= U+007E のみ <b>honolang</b> 環境変数 LC_ALL・LC_CTYPE・LANG が <i>utf8</i> ・ <i>UTF-8</i> ・ <i>cpXXXX</i> ・ <i>CPXXXX</i> ・ <i>iso8859-x</i> ・ <i>ISO-8859-x</i> のいずれかを指定していれば、それが解釈されてファイル名に適用されます。 <b>legacy</b> <i>auto</i> エンコーディング (すなわちカレントのシステムエンコーディング) を用いて、ファイル名を解釈し、また <i>honolang</i> パラメタが設定されている場合には LANG 変数を解釈します。 <b>unicode</b> (EBCDIC-) UTF-8 形式の Unicode エンコーディング すべての有効なエンコーディング名 PLOP によって認識される任意の (内部またはユーザー定義) エンコーディング Unicode 非対応バイディングで UTF-8 BOM なしで length=0 で与えられたファイル名は、この <i>filenamehandling</i> オプションに従って解釈されます。
<b>license</b>	(文字列) ライセンスキーを設定します。PLOP_open_document*() への初めての呼び出しよりも前に設定する必要があります。
<b>licensefile</b>	(文字列) ライセンスキー (複数可) の入ったファイルの名前を設定します。ライセンスファイルは、PLOP_open_document*() への初めての呼び出しよりも前に 1 度だけ設定できます。あるいはライセンスファイルの名前は、PLOPLICENSEFILE という環境変数で与えたり、(Windows の場合) レジストリで与えたりすることも可能です。

表 7.7 PLOP\_set\_option() に対するグローバルオプション一覧

オプション	説明
<b>frontpage</b>	(論理値) <i>false</i> の場合、有効なライセンスキーが見つからないときに例外を発生させます。 <i>true</i> の場合、0.1 節「ソフトウェアをインストール」(5 ページ) に従って評価モードで表紙が生成されます。このオプションは、 <i>PLOP_open_document*()</i> への初めての呼び出しよりも前に設定する必要があります。有効なライセンスキーが見つかったときは、このオプションは何の効果も持ちません。デフォルト : <i>true</i>
<b>searchpath<sup>1</sup></b>	(名前文字列のリスト) 読み込みたいファイルの入ったディレクトリの相対パス名か絶対パス名(複数可)。この検索パスは複数回設定することができます。その場合、項目は蓄積されて、設定された順に使用されます。空文字列を指定すると、それまでの検索パス項目がすべて削除されます。Windows の場合、 <i>searchpath</i> はレジストリ項目で設定することも可能です。デフォルト : 空
<b>shutdown-strategy</b>	(整数) すべての PLOP オブジェクトに対して 1 度だけ割り当てられるグローバルリソースの解放方針。グローバルリソースはそれぞれ、それが初めて必要とされた時点で要求によって初期化されます。このオプションは、1 個のプロセス内のすべての PLOP オブジェクトに対して同一の値に設定する必要があります。そうしない場合の動作は未定義です (デフォルト : 0): <ul style="list-style-type: none"> <li>o 何個の PLOP オブジェクトがリソースを使用しているかを参照カウンタが追跡します。最後の PLOP オブジェクトが削除されて参照カウンタがゼロになった時点で、リソースは解放されます。</li> <li>1 リソースはプロセスの最後まで保持されます。これは速度をわずかに向上させる可能性があります。最後の PLOP オブジェクトが削除された後により多くのメモリを必要とします。</li> </ul>

1. オプションの値は複数回の呼び出しによって蓄積させることが可能です。

## 7.6 pCOS 関数

PDF からオブジェクトデータを取得するための完全な pCOS 文法に対応しています。詳しい説明は pCOS パスリファレンスを参照してください。

---

**C++** `double pcos_get_number(int doc, string path)`

**C# Java** `double pcos_get_number(int doc, String path)`

**Perl PHP** `double pcos_get_number(long doc, string path)`

**VB** `Function pcos_get_number(doc as Long, path As String) As Double`

**C** `double PLOP_pcos_get_number(PLOP *plop, int doc, const char *path, ...)`

---

数値型か論理値型の pCOS パスの値を得ます。

**doc** `PLOP_open_document*()` で取得した有効な文書ハンドル。

**path** 数値オブジェクトか論理値オブジェクトに対する完全な pCOS パス。

**追加の引数群** (C 言語バインディングのみ) **key** 引数にプレースホルダがある場合、それに対応する任意の数の追加パラメタを与えることができます (`%s` で文字列、`%d` で整数。`%%` を用いると 1 個のパーセント記号になります)。これらの引数を利用すれば、可変の数値や文字列値を含む複雑なパスをいちいち構成する手間が省けます。プレースホルダの数と型が、与える追加パラメタ群に一致するようにするのは、クライアント側の役割です。

**戻り値** pCOS パスで示されたオブジェクトの数値。論理値の場合、`true` なら 1 が返され、そうでないなら 0 が返されます。

---

**C++** `string pcos_get_string(int doc, string path)`

**C# Java** `String pcos_get_string(int doc, String path)`

**Perl PHP** `string pcos_get_string(long doc, string path)`

**VB** `Function pcos_get_string(doc as Long, path As String) As String`

**C** `const char *PLOP_pcos_get_string(PLOP *plop, int doc, const char *path, ...)`

---

名前・文字列・論理値のいずれかの型の pCOS パスの値を得ます。

**doc** `PLOP_open_document*()` で取得した有効な文書ハンドル。

**path** 名前・文字列・論理値のいずれかのオブジェクトに対する完全な pCOS パス。

**追加の引数群** (C 言語バインディングのみ) **key** 引数にプレースホルダがある場合、それに対応する任意の数の追加パラメタを与えることができます (`%s` で文字列、`%d` で整数。`%%` を用いると 1 個のパーセント記号になります)。これらの引数を利用すれば、可変の数値や文字列値を含む複雑なパスをいちいち構成する手間が省けます。プレースホルダの数と型が、与える追加パラメタ群に一致するようにするのは、クライアント側の役割です。

**戻り値** pCOS パスで示されたオブジェクトの値の文字列。論理値の場合、文字列 `true` か `false` が返されます。

**詳細** pCOS がフルモードで動作していないとき、かつオブジェクトの型が文字列の場合には、この関数は例外を発生させます。例外として、`/Info/*` オブジェクト群（文書情報キー）は制限 pCOS モードでも `nocopy=false` か `plainmetadata=true` なら取得することができ、また、`bookmarks[...]/Title` と `annots[...]/contents` は制限 pCOS モードでも `nocopy=false` なら取得できます。

この関数では、PDF 文書から得られる文字列はテキスト文字列であると前提しています。バイナリデータの入った文字列オブジェクトは、これではなく `PLOP_pcos_get_stream()` で取得するべきで、それならデータは一切変更されません。

**バインディング** C バインディング：返される文字列は、最大 10 項目を持つリングバッファ内に格納されます。10 個を超える文字列がクエリされたときには、バッファは再利用されますので、10 個を超える文字列を同時に利用したい場合には、クライアント側でその文字列を複製しておく必要があります。たとえば、`printf()` 文の引数ではこの関数を最大 10 回まで呼び出すことができます。同時に 10 個を超える文字列が使用されないならば、その戻り文字列は互いに独立であることが保証されているからです。

C・C++ 言語バインディング：文字列は BOM のない UTF-8 形式で返されます。zSeries では、結果は BOM のない EBCDIC-UTF-8 形式で返されます。

C バインディング：返された文字列は、次にこの関数を呼び出すまでのあいだ使用できません。

Java・.NET：結果は Unicode 文字列として提供されます。もうテキストがないときは null オブジェクトが返されます。

Perl・PHP 言語バインディング：結果は UTF-8 文字列として提供されます。もうテキストがないときは null オブジェクトが返されます。

RPG 言語バインディング：結果は EBCDIC-UTF-8 文字列として提供されます。

---

**C++** `const unsigned char *pcos_get_stream(int doc, int *length, string optlist, string path)`

**C# Java** `byte[] pcos_get_stream(int doc, String optlist, String path)`

**Perl PHP** `string pcos_get_stream(long doc, string optlist, string path)`

**VB** `Function pcos_get_stream(doc as Long, optlist As String, path As String)`

**C** `const unsigned char *PLOP_pcos_get_stream(PLOP *plop, int doc, int *length, const char *optlist, const char *path, ...)`

---

`stream`・`fstream`・文字列のいずれかの型の pCOS パスの値を得ます。

`doc` `PLOP_open_document*()` で取得した有効な文書ハンドル。

`length` (C・C++ 言語バインディングのみ) 返されるストリームデータの長さをバイト単位で受け入れる変数へのポインタ。

`optlist` 表 7.8 に従っていくつかの取得オプションを指定するオプションリスト。

`path` ストリームオブジェクトか文字列オブジェクトに対する完全な pCOS パス。

**追加の引数群** (C 言語バインディングのみ) `key` 引数にプレースホルダがある場合、それに対応する任意の数の追加パラメータを与えることができます (`%s` で文字列、`%d` で整数。`%%` を用いると 1 個のパーセント記号になります)。これらの引数を利用すれば、可変の数値や文字列値を含む複雑なパスをいちいち構成する手間が省けます。プレースホル

ダの数と型が、与える追加パラメータ群に一致するようにするのは、クライアント側の役割です。

**戻り値** ストリームか文字列に入っている暗号化されていない状態のデータ。ストリームまたは文字列が空のときは、返されるデータは空 (C・C++ では NULL) になります。

オブジェクトが *stream* 型のときは、すべてのフィルタがストリームの内容から除去されます (すなわち、実際の生データが返されます)。オブジェクトが *fstream* 型か**文字列型**のときは、データは PDF ファイル内で見つかったそのままで返されますが、ただし例外として ASCII85・ASCII-Hex フィルタは除去されます。

**詳細** pCOS がフルモードで動作していないとき、この関数は例外を発生させます。例外として、*/Root/Metadata* オブジェクトは制限 pCOS モードでも *nocopy=false* か *plainmetadata=true* なら取得することができます。*path* が *stream*・*fstream*・**文字列型**のオブジェクトを指していないときにも例外が発生します。

名前と違ってこの関数は、**文字列型**のオブジェクトを取得するためにも使えます。*PLOP\_pcos\_get\_string()* の場合、オブジェクトをテキスト文字列として取り扱いますが、それとは違ってこの関数では、返すデータに一切の変改を加えません。バイナリ文字列データは PDF 内で用いられることは稀で、自動的に検出しようとしても確実ではありません。ですので、文字列オブジェクトをバイナリデータとして取得するかテキストとして取得するか、考えて適切な関数を選ぶのはユーザー側の役割です。

**バインディング** COM : 多くのクライアントプログラムでは、ストリーム内容を保持するためにバリエーション型を用いています。JavaScript で COM を使う場合、返されたバリエーション配列の長さを取得することは許されていません (ただし、それ以外の言語で COM を使う場合は可能です)。

C・C++ 言語バインディング : 返されたデータバッファは、次にこの関数を呼び出すまでのあいだ使用できます。

この関数を利用すると、PDF から埋め込みフォントデータを抽出できます。フォントはそれぞれのフォントベンダのライセンス許諾下にあり、それぞれの知的所有権者の明示的な許諾なしに再利用してはいけませんので、利用者はこのことに留意してください。関連するライセンス許諾を協議するにはお使いのフォントのベンダに連絡してください。

表 7.8 *PLOP\_pcos\_get\_stream()* に対するオプション一覧

オプション	説明
<b>convert</b>	(キーワード。非対応のフィルタで圧縮されているストリームに対しては無視されます) 文字列またはストリームの内容が圧縮されるかどうかを制御 (デフォルト : <i>none</i> ) :
<b>none</b>	内容をバイナリデータとして扱い、一切変換しません。
<b>unicode</b>	内容をテキストデータとして (すなわち <i>PLOP_pcos_get_string()</i> と全く同様に) 扱い、Unicode に規格化します。Unicode 非対応の言語バインディングの場合、これはデータが BOM なしの UTF-8 形式に変換されることを意味します。 このオプションは、PDF 内でめったに使われないデータ型「テキストストリーム」(JavaScript 等のために使われます。ただし JavaScript の大多数はストリームオブジェクトでなく文字列オブジェクト内に格納されます) のために必要です。

## 7.7 Unicode 変換関数

---

**C++** `string convert_to_unicode(string inputformat, string input, string optlist)`

**C# Java** `string convert_to_unicode(string inputformat, byte[] input, string optlist)`

**Perl PHP** `string convert_to_unicode(string inputformat, string input, string optlist)`

**VB** `Function convert_to_unicode(inputformat as String, input, optlist as String) As String`

**C** `const char *PLOP_convert_to_unicode(PLOP *p,  
const char *inputformat, const char *input, int inputlen, int *outputlen, const char *optlist)`

---

任意のエンコーディングの文字列を、さまざまな形式の Unicode 文字列へ変換します。

**inputformat** 入力文字列の解釈を指定する Unicode テキスト形式またはエンコーディング名:

- ▶ Unicode テキスト形式: `utf8`・`ebcdicutf8`・`utf16`・`utf16le`・`utf16be`・`utf32`
- ▶ すべての内部的に知られている 8 ビットエンコーディングと、ホストシステム上で利用可能なエンコーディングと、日中韓エンコーディング `cp932`・`cp936`・`cp949`・`cp950`
- ▶ キーワード `auto` は次の動作を指定します: 入力文字列に UTF-8 または UTF-16 BOM がある場合はそれを用いて適切な形式が決定され、ない場合はカレントのシステムコードページであると見なされます。

**input** Unicode へ変換したい文字列 (COM ではバリエーション)

**inputlen** (C 言語バインディングのみ) 入力文字列の長さをバイト単位で。 `inputlen = 0` の場合には、ヌル終端文字列を与える必要があります。

**outputlen** (C 言語バインディングのみ) 返される文字列の長さ (バイト単位で) が格納されるメモリ位置への C スタイルのポインタ。

**optlist** 入力の解釈と Unicode 変換のためのオプション群を指定したオプションリスト:

- ▶ 表 7.9 に従った入力フィルタオプション群: `charref`・`escapesequence`
- ▶ 表 7.9 に従った Unicode 変換オプション群: `bom`・`errorpolicy`・`inflate`・`outputformat`

**戻り値** 指定された引数とオプションに従って入力文字列から生成された Unicode 文字列。入力文字列が、指定された入力形式に準拠していないとき (無効な UTF-8 文字列など) は、`errorpolicy=return` の場合には空の出力文字列が返され、`errorpolicy=exception` の場合には例外が発生します。

**詳細** この関数は、汎用の Unicode 文字列変換に有用でしょう。これは、適切な Unicode コンバータを提供していない環境で作業をするユーザーの便宜のために提供されています。

**スコープ** 任意

**バインディング** C バインディング: 返される文字列は、最大 10 項目を持つリングバッファ内に格納されます。10 個を超える文字列が変換されたときには、バッファは再利用されますので、10 個を超える文字列を同時に利用したい場合には、クライアント側でその文字列を複製しておく必要があります。たとえば、`printf()` 文の引数ではこの関数を最大 10 回まで呼び出すことができます。同時に 10 個を超える文字列が使用されないならば、その戻り文字列は互いに独立であることが保証されているからです。

表 7.9 PLOP\_convert\_to\_unicode() に対するオプション一覧

オプション	説明
<b>bom</b>	(キーワード) <code>outputformat=utf32</code> の場合には無視されます) バイト順序マーク (BOM) を出力文字列に加えるかどうかの方針。使えるキーワード (デフォルト: <code>none</code> ): <b>add</b> BOM を加えます。 <b>keep</b> 入力文字列に BOM があるなら BOM を加えます。 <b>none</b> BOM を加えません。 <b>optimize</b> <code>outputformat=utf8</code> または <code>ebcdicutf8</code> かつ出力文字列が範囲 U+007F のキャラクタのみ含む場合以外には BOM を加えます。
<b>charref</b>	(論理値) <code>true</code> の場合、数値・文字実体参照とグリフ名参照の置き換えを有効にします。デフォルト: <code>false</code>
<b>errorpolicy</b>	(キーワード) 変換エラーの場合の動作 (デフォルト: <code>exception</code> ): <b>return</b> 文字参照が解決できないときに代替キャラクタが使用されます。変換エラーの場合に空文字列が返されます。 <b>exception</b> 変換エラーの場合に例外が発生します。
<b>escape-sequence</b>	(論理値) <code>true</code> の場合、文字列内のエスケープシーケンスの置き換えを有効にします。デフォルト: <code>false</code>
<b>inflate</b>	(論理値) <code>inputformat=utf8</code> の場合のみ。 <code>outputformat=utf8</code> の場合には無視されます) <code>true</code> の場合、無効な UTF-8 入力文字列が来ても例外が発生せず、指定された出力形式のインフレートされたバイト文字列が生成されます。これはデバッグのために有用でしょう。デフォルト: <code>false</code>
<b>output-format</b>	(キーワード) 生成される文字列の Unicode テキスト形式: <code>utf8</code> ・ <code>ebcdicutf8</code> ・ <code>utf16</code> ・ <code>utf16le</code> ・ <code>utf16be</code> ・ <code>utf32</code> 。空文字列は <code>utf16</code> と同等です。デフォルト: <code>utf16</code> Unicode 対応言語バインディング: 出力形式は <code>utf16</code> を強制されます。 C++ 言語バインディング: 次の出力形式のみ許されます: <code>utf8</code> ・ <code>utf16</code> ・ <code>utf32</code> 。

---

**C++** `string utf16_to_utf8(string utf16string)`

**Perl PHP** `string utf16_to_utf8(string utf16string)`

**C** `const char *PLOP_utf16_to_utf8(PLOP *p, const char *utf16string, int len, int *size)`

---

非推奨。PLOP\_convert\_to\_unicode() を使用してください。

---

**C++** `string utf8_to_utf16(string utf8string, string ordering)`

**Perl PHP** `string utf8_to_utf16(string utf8string, string ordering)`

**C** `const char *PLOP_utf8_to_utf16(PLOP *p, const char *utf8string, const char *ordering, int *size)`

---

非推奨。PLOP\_convert\_to\_unicode() を使用してください。



# A PDFlib を PLOP または PLOP DS と組み合わせる

PDFlib のバージョン番号によっては、PLOP と PDFlib・PDFlib+PDI・PDFlib Personalization Server (PPS) を組み合わせることは意味があるでしょう。表 7.10 に、PDFlib ファミリーにおける暗号化・線形化・最適化 / 修復モード・電子署名の機能の有無をまとめました。自分の PDFlib のバージョンの対応していない機能を必要とするあらゆる場面において、PLOP または PLOP DS を PDFlib に組み合わせることは意味があります。

表 7.10 さまざまな PDFlib バージョンにおける暗号化・線形化・最適化・電子署名への対応

PDFlib バージョン	暗号化	線形化	最適化 / 修復モード	電子署名
PDFlib/PDFlib+PDI/PPS 5	有	—	—	—
PDFlib/PDFlib+PDI/PPS 6	有	有	—	—
PDFlib/PDFlib+PDI/PPS 7・8	有	有	有	—

PLOP は、PDF を動的に生成して後処理するために、PDFlib と容易に相互作用するよう設計されています。この章では、この 2 つの製品を結合する方法を説明します。PDFlib で生成した文書を、PLOP コマンドラインツールを使って後処理することも可能ですが、PLOP ライブラリを使ってそうするほうを推奨します。

**注記** PDFlib 7・8 ではフォームフィールドに対して Appearance ストリームを生成しないので、フォームフィールドを含む PDFlib 生成文書に PLOP を使って署名することは、フォームフィールドを *sacrifice* オプションで削除しない限りできません。

**ファイルベースでの結合** ファイルベース方式は、非常に大きな PDF 文書を扱う場合や、PDFlib/PLOP 結合の総メモリ要求を下げる必要がある場合に推奨します。単に、適切な PDFlib ルーチンで PDF ファイルをディスク上に生成した後、それを *PLOP\_open\_document()* で処理します。

**メモリベースでの結合** メモリベース方式は比較的速いですが、メモリを比較的多く必要とします。非常に大きな文書を扱う場合を除いて、これは Web アプリケーションで動的な PDF 生成や署名を行う場合に推奨します。PDFlib で PDF ファイルをディスク上に生成するのではなく、*PDF\_begin\_document()* に空のファイル名を与えることによってインコアPDF生成を利用し、生成されたPDFデータの入ったバッファの内容を *PDF\_get\_buffer()* で取り出し、*PLOP\_create\_pvf()* で仮想ファイルを作成します。この仮想ファイルに用いたファイル名を、その後 *PLOP\_open\_document()* を用いて PLOP/PLOP DS へ渡します。こうすると物理的なファイルをディスク上に作成する必要がありません。ただし、PDFlib のバッファ内容を複数の部分に分けて取り出すことはできません。なぜなら PLOP/PLOP DS には文書全体を 1 個のバッファで与える必要があるからです。ですので、*PDF\_end\_document()* と *PDF\_delete()* との間で *PDF\_get\_buffer()* を呼び出す必要があります。

すべての PLOP パッケージに入っている *hellosign* プログラミングサンプルでは、PDFlib を使って動的に PDF 文書を生成し、それを PLOP にメモリ内で渡して電子署名を適用する方法を示しています。

# B PLOP ライブラリクイックリファレンス

以下の表は、すべての PLOP API 関数の概観です。頭に (C) がついているのは C プロトタイプを表しており、Java 言語バインディングでは利用できません。

## 一般関数

関数プロトタイプ	ページ
(C) <i>PLOP *PLOP_new(void)</i>	75
<i>void delete()</i>	75
<i>void create_pvf(string filename, const void *data, size_t size, string optlist)</i>	75
<i>int delete_pvf(string filename)</i>	75
<i>double info_pvf(String filename, String keyword)</i>	77

## 文書入力・出力

関数プロトタイプ	ページ
<i>int open_document(String filename, String optlist)</i>	78
(C) <i>int PLOP_open_document_callback(PLOP *plop, void *opaque, size_t filesize, size_t (*readproc)(void *opaque, void *buffer, size_t size), int (*seekproc)(void *opaque, long offset), const char *optlist)</i>	79
<i>int create_file(String filename, String optlist)</i>	80
<i>close_document(int doc)</i>	86
<i>byte[] get_buffer()</i>	86

## エラー処理

関数プロトタイプ	ページ
<i>int get_errnum()</i>	87
<i>String get_errmsg()</i>	87
<i>String get_apiname()</i>	88

## オプション処理

関数プロトタイプ	ページ
<i>void set_option(String optlist)</i>	89

## pCOS 関数

関数プロトタイプ	ページ
<i>double pcos_get_number(int doc, String path)</i>	91
<i>String pcos_get_string(int doc, String path)</i>	91
<i>byte[] pcos_get_stream(int doc, String optlist, String path)</i>	92

## Unicode 変換関数

### 関数プロトタイプ

`string convert_to_unicode(string inputformat, byte[] input, string optlist)`

### ページ

94

# C 変更履歴

このマニュアルの変更履歴

日付	変更
2011年03月04日	▶ PLOP 4.1・PLOP DS 4.1 のためのメジャーオーバーホール
2008年12月05日	▶ PLOP 4.0・PLOP DS 4.0 の XMP・PVF・PKCS#11（スマートカード）対応に関する更新
2007年07月15日	▶ PLOP 3.0 と PLOP DS 3.0 に関する更新
2004年09月27日	▶ PLOP 2.1 に関する更新
2003年12月01日	▶ 新しいメジャーリリース PLOP 2.0 に関する更新
2002年11月23日	▶ Perl 用 PSP バインディングの記述を追加
2002年11月07日	▶ ILE-RPG での PSP の利用に関する節を追加
2002年10月22日	▶ PSP 1.0.1 に関する若干の変更
2002年09月17日	▶ PSP 1.0.0 に関する第一版

# 索引

## B

byteserving 15

## C

C++ と .NET 42

C++ バインディング 36

certified PDF 66

CLI 36

COM バインディング 39

C バインディング 33

## D

DSA による署名 66

## G

Ghent Workgroup (GWG) 20

## J

Java バインディング 40

## K

KeyUsage 証明書拡張 65

## M

Microsoft Cryptographic API (MSCAPI) 60

## N

.NET バインディング 42

noaccessible 54

noannots 54

noassemble 55

nocopy 54

noforms 54

nohiresprint 55

nomodify 54

nonrepudiation keyusage フラグ 65

noprint 54

## P

page-at-a-time ダウンロード 15

pCOS 71

API 関数 91

クックブック 13

PDF/A 24, 25

と XMP メタデータ 20

PDF/X 25

PDFlib と PLOP/PLOP DS 97

PDF バージョン, 生成出力の 24

Perl バインディング 43

PDF 形式 60

PHP バインディング 44

PKCS#11 60, 64

PKCS#12 60

plainmetadata 55

PLOP DS の署名の暗号の詳細 66

PLOP\_CATCH() 88

PLOP\_close\_document() 86

PLOP\_convert\_to\_unicode() 94

PLOP\_create\_file() 80

PLOP\_create\_pvf() 75

PLOP\_delete() 75

PLOP\_delete\_pvf() 76

PLOP\_EXIT\_TRY() 34, 88

PLOP\_get\_apiname() 88

PLOP\_get\_buffer() 86

PLOP\_get\_errmsg() 87

PLOP\_get\_errnum() 87

PLOP\_info\_pvf() 77

PLOP\_new() 75

PLOP\_open\_document() 78

PLOP\_open\_document\_callback() 79

PLOP\_pcos\_get\_number() 91

PLOP\_pcos\_get\_stream() 92

PLOP\_pcos\_get\_string() 91

PLOP\_RETHROW() 88

PLOP\_set\_option() 89

PLOP\_TRY() 88

PLOP・PLOP DS コマンドラインツール

オプション 27

作成例 32

終了コード 31

諸機能 11

PLOP・PLOP DS ライブラリ

API リファレンス 73

クイックリファレンス 98

諸機能 11

Python バインディング 46

## R

Reader 有効化された PDF 25

RPG バインディング 47

RSA による署名 66

## W

Web 最適化 PDF 15

## X

XMP メタデータ 19, 20  
プレーンテキスト 52  
無効な 21

## あ

暗号化アルゴリズム, 電子署名の 66  
暗号化エンジン 60  
暗号化されたファイル添付 25  
暗号化ファイル添付 52  
暗号トークン 64  
暗号の詳細, PLOP DS の署名の 66

## い

一時ディスク容量の必要量 25  
インストール, PLOP/PLOP DS 5

## お

オプションリスト 73

## か

外部暗号化エンジン 60  
ガベージコレクション 16

## き

キー長, 電子署名の 66

## け

権限設定 51  
権限パスワード 49  
検証, 電子署名を Acrobat で 67

## さ

最適化 16  
最適化 PDF 15

## し

修復モード, 破損 PDF のための 17  
終了コード 31  
証明書, Windows でまとめる 61  
商用ライセンス 8  
署名→電子署名  
署名, PDF 文書に 63

所有者パスワード 49

## す

ストリーム最適化 16  
スマートカード 64

## せ

線形化 PDF 15

## そ

増分アップデート 66

## た

大容量 PDF 文書 26

## つ

使われていないオブジェクト 16

## て

デジタル ID 60  
デジタル署名→電子署名  
電子署名 22, 59  
Acrobat で検証 67  
入力文書内の 25  
添付パスワード 49

## は

バイトサービング 15  
パスワード 49, 50  
デジタル ID の 65  
パスワードファイル, デジタル ID の 65  
破損した入力 PDF 17  
ハッシュ関数, 電子署名の 66

## ひ

評価版 5

## ふ

ファイル添付  
暗号化 52  
フォームフィールド, 入力文書の 25  
フォント最適化 16  
文書情報項目 19

## へ

ページごとのダウンロード 15

## ほ

放棄, 入力文書の特性を 24

## ま

マスターパスワード 49

## む

無効な XMP メタデータ 21

## め

メッセージダイジェスト, 電子署名の 66

## ゆ

ユーザーパスワード 49

## ら

ライセンスキー 6

## れ

例外処理 87

    C の 33

レスポンスファイル 30

**PDFlib GmbH**

Franziska-Bilek-Weg 9  
80339 München, Germany  
[www.pdflib.com](http://www.pdflib.com)

電話 +49・89・452 33 84-0  
fax +49・89・452 33 84-99

疑問がおありの際は、PDF メーカーリストと、  
[tech.groups.yahoo.com/group/pdflib](http://tech.groups.yahoo.com/group/pdflib) のアーカイブをチェックしてください

ライセンスに関するお問い合わせ  
[sales@pdflib.com](mailto:sales@pdflib.com)

サポート  
[support@pdflib.com](mailto:support@pdflib.com) (お使いのライセンス番号をお書きください)

