PDFlib GmbH

PLOP・PLOP DS

Version 5.0

PDF の線形化・最適化・保護・デジタル署名

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、*groups.yahoo.com/neo/groups/pdflib/info* 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*
商用 *PDFlib* ライセンス保持者向けサポート：*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*（*eay@cryptsoft.com*）
OpenSSL Toolkit 内で使用するために *the OpenSSL Project* によって開発されたソフトウェア（*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*
Libcurl multiprotocol file transfer library、*Copyright © 1996-2014, Daniel Stenberg*（*daniel@haxx.se*）

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

0 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、以下のステップを推奨します。

- 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「PLOP の諸機能」（15 ページ）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 3.1 節「PLOP・PLOP DS コマンドラインオプション」（33 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 4 章「PLOP・PLOP DS ライブラリの言語バインディング」（39 ページ）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）、あるいはインストレーションディレクトリから（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。PLOP または PLOP DS の未ライセンス版は、業務目的に使用してはならず、その製品を評価するためにのみ使用することができます。その製品を業務環境に実装するには、有効なライセンスが必要です。

有効なライセンスキーを適用しないと、PLOP は、***unlicensed*** というテキストを出力文書のメタデータに入れ込んで、文書の先頭に追加の表紙ページを挿入します。試験を実行するために、以下の条件の一方ないし両方が真ならば表紙は生成されません：

- 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）。
- 署名を行うために使用するデジタル 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 DS パッケージに入っています。

場合によっては、入力が PDF/A・PDF/UA・PDF/VT・PDF/X のいずれかの規格に準拠していても、この表紙の挿入によって、PDF 出力がそれに準拠しなくなることがあります。この非準拠はこの表紙ページのみに特有であり、有効なライセンスキーを適用した後にはこの問題は起こりません。

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 1 MB 未満）。

p.open_document( ) から取得された各文書ハンドルに対しては、評価モードでは、*p.create_document( )* 呼び出しへのただ 1 回の呼び出しのみが許されます。

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に表紙ページが生成されず例外が発生します。

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利用できます。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。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）*license* パラメータを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 ..."
```

- C++・Java・.NET/C#・Python・Ruby の場合：

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

- C の場合：

```
PLOP_set_option(p, "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 ...")
```

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製品のライセンシング情報
PDFlib license file 1.0
PLOP        5.0             ...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、個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」（10 ページ）参照）。
- *licensefile* パラメータを *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*--plopopt*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 --plopopt "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--plopopt "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
```

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\PLOP5\license
HKLM\SOFTWARE\PDFlib\PLOP5\5.0\license
```

MSI インストーラはライセンスキーをこれらのエントリの末尾へ書き込みます。

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」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・OS X 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/5.0/resource/cmap
<rootpath>/PDFlib/PLOP/5.0/resource/codelist
<rootpath>/PDFlib/PLOP/5.0/resource/glyphlst
<rootpath>/PDFlib/PLOP/5.0/resource/fonts
<rootpath>/PDFlib/PLOP/5.0/resource/icc
<rootpath>/PDFlib/PLOP/5.0
<rootpath>/PDFlib/PLOP
<rootpath>/PDFlib
```

Unix・Linux・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。

ライセン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
```

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。

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*

0.3 説明書とサンプル群への案内

PLOP 用各種ミニサンプル　PLOP ディストリビューションは、すべての対応言語バインディングのためのシンプルなプログラミング例を含んでいます。これらは基本的な PLOP ライブラリプログラミング作業を演示しています：

- *encrypt* サンプルは、暗号化されていない PDF 文書を、ユーザー・マスターパスワードを用いて暗号化します。
- *dumper* サンプルは、pCOS インタフェースを用いて、文書の一般特性群、暗号化に関する情報、署名ステータスに加え、文書情報と XMP メタデータを収集します。
- *insertxmp* サンプルは、ファイルから XMP メタデータを読み取り、その XMP を PDF 文書内に挿入します。試験用のサンプル XMP ファイル群が与えられています。

PLOP DS 用各種ミニサンプル　以下のミニサンプル群が PLOP DS で使えます：

- *sign* サンプルは、既存の PDF 文書に電子署名を行う方法を示します。
- *multisign* サンプルは、複数の PDF 文書に電子署名を行う方法を示し、PKCS#11 トークンのためのセッション処理を演示します。
- *hellosign* は、PDFlib を用いてメモリ内に動的に文書を生成して、それを PLOP DS へ渡し、PLOP DS でそれに電子署名を行う方法を示します。この例は、PLOP パッケージには含まれていない PDFlib 製品を必要とすることに留意してください。ただし、PDFlib の無料評価版を当社 Web サイトから入手できます。

この署名サンプル群は、パッケージ内にも含まれているデモ電子 ID 群を用いるように作られています。この電子 ID ファイル群（*demorsa2048.p12* 等）に対するパスワードは *demo* です。

PLOP コマンドラインツールへの各種サンプル呼び出し　PLOP コマンドラインツールではさまざまなオプションを使えます。それらは以下の章で解説されており、それらは PLOP コマンドラインツールのサンプル呼び出しもそこに含まれています：

- 1 章「PLOP の諸機能」（15 ページ）
- 2 章「PLOP DS の諸機能（電子署名）」（27 ページ）
- 3.1 節「PLOP・PLOP DS コマンドラインオプション」（33 ページ）・
  3.2 節「PLOP・PLOP DS コマンドラインの作成例」（37 ページ）。

pCOS クックブック　pCOS クックブックは、PLOP・PLOP DS に内蔵されている pCOS インタフェースのためのコード断片の集合です。以下の URL にあります：
www.pdflib.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されています。

0.4 PLOP と PLOP DS の概要

PLOP には 2 種類があります : PLOP 基本製品と、電子署名に対応した拡張版 PLOP DS です。

PLOP の諸機能　PLOP では以下のような PDF 処理ができます :

- 保護 : PDF 文書を、ユーザーまたはマスターパスワード（あるいは両方）を用いて暗号化します。PDF 暗号化を、その文書のマスターパスワードを知っている場合に除去します。権限設定群（印刷やテキスト抽出の不許可等）を、その文書のマスターパスワードを知っている場合に追加または除去します。
- PDF 文書を線形化することによって、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向上させます（後述）。
- PDF 文書のサイズを最適化するために、冗長なオブジェクトを削減します。
- 破損した PDF 文書を修復します。
- 内蔵の pCOS インタフェースを用いて、文書のセキュリティ状態（ユーザーまたはマスターパスワードを用いて暗号化されている）、権限設定群、文書メタデータ等多数の特性に関する情報をクエリします。
- 定義済みまたはカスタム文書情報項目群を挿入・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 のすべての機能に加え、PDF 文書に電子署名を行う機能を提供します。この署名は、タイムスタンピング・長期検証・PAdES 署名に対応しています。2.1 節「PLOP DS の署名機能」（27 ページ）で、PLOP DS の電子署名機能のまとめがあります。

さまざまな利点　PDFlib PLOP・PLOP DS は以下の利点を提供します :

- すべての PLOP・PLOP DS 操作は、PDF/A・PDF/UA・PDF/VT・PDF/X 規格に対応しています : 入力がこれらのいずれかの規格に準拠していれば、出力は、可能であればその同じ規格に準拠することが保証されています。これが可能でない場合には（PDF/A 入力に対して暗号化が要求された等）、その操作は拒絶されるか、あるいは規格識別が除去されます。
- PLOP は、PDF を読み取り、暗号化、署名、書き込みするために一切のサードパーティソフトウェアを必要としないスタンドアローンツールです。
- PLOP は、技術的にも法的にもサーバ上に実装することが可能で、完全にスレッドセーフであり、メモリリークに対する検査済みです。PLOP は、ヘビーなサーバ用途のために構築されており、Web サーバ環境において、または大容量バッチ処理等のために使用できます。
- PLOP は、多数のプラットフォーム上で、いくつかのプログラミング環境で利用可能です。
- さらなる柔軟性のために、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のため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 コマンドラインツールかライブラリか　PLOP/PLOP DS は、さまざまな開発言語のためのプログラミングライブラリ（コンポーネント）としても、バッチ操作のためのコマンドラインツールとしても利用可能です。どちらも同じ機能集合を提供しますが、それぞれ異なる実装タスクに適しています。ライブラリとコマンドラインツールのどちらを使うかについて、いくつかのガイドラインを示します :

- コマンドライン PLOP/PLOP DS ツールは、PDF 文書をバッチ処理するのに適しています。プログラミングを一切必要とせずに、それでいて強力なコマンドラインオプション群を提供しており、それらを用いて複雑なワークフローへそれを統合することが可能です。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呼び出すこともできます。
- PLOP/PLOP DS　プログラミングライブラリは、.NET・Java（サーブレットを含む）・PHP・プレーン C・C++ アプリケーション開発等、広く使われているさまざまな開発環境に良く統合します。

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 PLOP の諸機能

注記 PLOP DS の電子署名のための機能群については 2 章「PLOP DS の諸機能（電子署名）」（27 ページ）で解説しています。

1.1 暗号化・復号・権限

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 5 章「PDF 暗号化・復号」（57 ページ）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します。

権限設定を取得　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（1.5 節「pCOS で文書情報を取得」（19 ページ）にある例を参照）。

PLOP で文書を暗号化　*PLOP_create_document()* 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 で権限制限を指定　*PLOP_create_document()* 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（表 5.3（62 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。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 で文書を復号　*PLOP_create_document()* 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。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
```

暗号化や復号については、5.3 節「コマンドラインで PDF 文書を保護」（64 ページ）にも作成例があります。

1.2 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」）。
- 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 PDF ビューア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 X/XI：「**編集**」→「**環境設定**」→「**インターネット**」→「***Web* 表示用に最適化を許可**」）。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 5.2 参照）。

小さなファイルを線形化　線形化は、大きな PDF 文書の Web ベース表示の向上を目指すものですので、1 ページ文書に対してはあまり意味がありません（可能ですが）。しかし、Acrobat のバグによって、小さな線形化された文書は常に線形化文書として処理されるわけではありません。たとえば、Acrobat X/XI は 4KB より小さなすべての文書を非線形化文書と見なします。

PLOP で PDF 文書を線形化　*PLOP_create_document()* 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。

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
```

1.3 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**」コマンドを使われていると）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。

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。

こんにち、冗長オブジェクトの問題のある PDF 文書はごく一部のみとなっていることから、この最適化処理はデフォルトでは無効となっています。

PLOP で PDF 文書を最適化　最適化処理を有効にするには、*PLOP_create_document()* で *optimize=all* オプションを指定、あるいは、PLOP コマンドラインツールで *--outputopt* オプションを指定します。

例：PLOP コマンドラインツールで文書を最適化：

```
plop --outputopt optimize=all --outfile optimized.pdf input.pdf
```

PLOP を用いて XMP メタデータを除去

アプリケーションによっては、PDF 出力に、あらゆる状況で必要となるわけではない大量の XMP メタデータを付けて生成するものがあります。極端な場合には、PDF ファイル全体のサイズのほとんどを XMP メタデータが占めていることすらあります。こうした場合には、望まない XMP 文書メタデータを、PLOP を用いて以下のように除去できます：

```
plop --inputopt xmppolicy=remove --outfile output.pdf input.pdf
```

これによって、細かなメタデータを除去するかわりに PDF ファイルサイズを大幅に削減できる可能性があります。

1.4 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
```

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」（22 ページ）を参照してください。

1.5 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
```

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: 1.7
         Encryption: No encryption
          Master pw: false
            User pw: false
             nocopy: false (copying is allowed)
           nomodify: false (adding form fields and other changes is allowed)
           noannots: 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)
      nohiresprint: false (high-resolution printing is allowed)
     plainmetadata: true  (metadata is not encrypted)
        Linearized: true
       PDF/X status: none
       PDF/A status: none
      PDF/UA status: none
      PDF/VT status: none
        Tagged PDF: false
        Signatures: 0
    Reader-enabled: false

       No. of pages: 140
       No. of fonts: 10
           embedded TrueType font PDFlibLogo-Regular
           embedded Type 1 CFF font ThesisAntiqua-Bold
           embedded Type 1 CFF font TheSans-Italic
           ...
           embedded Type 1 CFF font ThesisAntiqua-Normal
           embedded Type 1 CFF font TheSansMonoCondensed-Plain

             Author: 'PDFlib GmbH'
       CreationDate: 'D:20141111172554'
            Creator: 'FrameMaker 11.0.2'
            ModDate: 'D:20141111172554+02'00''
           Producer: 'Acrobat Distiller 11.0 (Windows)'
            Subject: 'PDFlib PLOP and PLOP DS: PDF Linearization, Optimization,
Protection, Digital Signature'
              Title: 'PDFlib PLOP and PLOP DS Manual'

      XMP meta data: is present
     Encr. attachm.: no
```

1.6 文書情報項目を挿入・読み取り

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できます。

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_document()* 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。

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 で文書情報項目を読み取り　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）を読み取ることもできます。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（1.5 節「pCOS で文書情報を取得」（19 ページ）にある例を参照）。

PDF/A における文書情報項目　PDF/A 規格では文書情報項目に対して特別な取り扱いが義務付けられていることに留意してください：

- PDF/A-1：標準文書情報項目　***Title・Author・Subject・Keywords・Creator・Producer・CreationDate・ModDate*** は文書 XMP メタデータと同期されている必要があります。PLOP はこの同期を自動的に提供します。
- PDF/A-2/3：文書情報項目は存在してもよいですが、PDF/A 準拠リーダーによって無視されなければなりません。それらが存在する場合には、文書 XMP と同期しているべきであり、これは PDF/A-1 の場合と同じく PLOP によって自動的に行われます。

1.7 XMP メタデータを挿入・抽出・除去

XMP（*Extensible Metadata Platform*）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 PDF/A 等さまざまな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/* 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 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 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 文書に追加。PDF/A 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/download/job-tickets/* 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_document()* 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 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、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。

PLOP で XMP メタデータを除去　場合によっては、XMP メタデータを除去したい場合もあるでしょう。たとえば、それがもはや実際の文書に合致していない場合等です。これは PLOP で以下のように実現できます：

```
plop --inputopt xmppolicy=remove --outfile output.pdf input.pdf
```

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
```

1.8 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 を受け入れます：

- PDF 1.6（Acrobat 7）およびそれより古いすべてのバージョン
- 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・XI）
- PDF 2.0。ISO 32000-2（現在策定中）に従ったものです

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る場合があります。PLOP は、さまざまな種類の破損 PDF 文書の修復を試みます。

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小さ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。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：

- PDF/A-1・PDF/X モードでは、PDF バージョンは変更されずに保たれます。PDF/A-2/3 モードでは、PDF 1.7 が生成されます。
- それ以外の場合には、PDF 出力バージョンは少なくとも PDF 1.6 です。
- 暗号化（オプション *masterpassword*）は、PDF バージョンを、暗号化アルゴリズム 4 の場合には PDF 1.7ext3 へ押し上げ、暗号化アルゴリズム 11 の場合には PDF 1.7ext8 へ押し上げます。
- 署名機能のなかには、PDF バージョンを PDF 1.7ext8 へ押し上げるものがあります（表 6.1 参照）。

規格準拠　PLOP の処理はいくつかの PDF 規格に準拠しています。入力が以下のいずれかの規格に準拠している場合には、PLOP によって生成される出力はそれと同じ規格に準拠することが保証されています：

- PDF/A-1/2/3：すべての種類
- PDF/X-3/4/5・PDF/VT-1/2：すべての種類
- PDF/UA-1

PLOP の操作（とりわけ暗号化）のなかには、特定の規格と互換でないものもあることに留意してください。この場合には、*sacrifice* オプションを用いて優先順位を設定することもできます（下記）。

入力 PDF の特定の特性を放棄　PDF 文書の特性のうちのいくつかは、特定の PLOP のアクション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 文書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、その操作を拒絶して例外を発生させます。しかし、*PLOP_create_document()* で、または PLOP コマンドラインツールの *--outputopt* オプションで、オプション *sacrifice* を使えば、行わせたいアクションに対して、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、行わせたいアクションとの組み合わせは、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 7.5 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-1・PDF/A-2・PDF/A-3 のいずれかの標準規格に準拠している入力文書は、PDF/A 準拠の署名付き出力を生成することが保証されています。しかし暗号化は PDF/A 文書に対しては許されません。この規格では暗号化が一切禁止されているからです。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。署名視覚化のために用いられている PDF ページも PDF/A も準拠する必要があります（6.3.1 節「署名をグラフィックかロゴで視覚化」（79 ページ）参照）。
- 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。署名視覚化は PDF/X モードでは使えません。
- PDF/UA：多くの PLOP 操作は、*permissions=noaccessible* を例外として、自動的に PDF/UA-1 に準拠します。*sacrifice={pdfua}* オプションを用いて PDF/UA 準拠を放棄することもできます。署名視覚化は PDF/UA モードでは使えません。
- PLOP は、視覚表現を持たない署名なしフォームフィールド（PDFlib 7/8/9 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ず、その種の入力に対してはエラーを発します。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_document()* に対して、または PLOP コマンドラインツールの *--outputopt* オプションで、*sacrifice={fields}* というオプションを指定すれば、既存フォームフィールド群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。*PLOP_create_document()* に対して、または PLOP コマンドラインツールの *--outputopt* オプションで、*sacrifice={encryptedattachments}* というオプションを指定すれば、暗号化されたファイル添付群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_document()* 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。なお、線形化は電子署名と組み合わせることはできません。
- 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、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_document()* 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、おおよそ入力のサイズの 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 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・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 文書内の昔ながら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。これはおよそ 9.3 GB にあたります。しかし、圧縮されたオブジェクトストリームを用いれば、この制約を超えることが可能です。圧縮されたオブジェクトストリームはいずれにせよ全体のファイルサイズを削減しますが、*objectstreams* 実装の一部である圧縮された相互参照ストリームはもはや 10 進 10 桁の制約に縛られず、それゆえ 10 GB を超える PDF 文書の作成を許容します。
- オブジェクトの数：一文書内のオブジェクトの数は全般的には PDF によって制限されていませんが、PDF/A・PDF/X-4・PDF/X-5 規格では、一文書内の間接オブジェクトの数を 8,388,607 個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・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。他のモードでは、もっとオブジェクトの多い文書も必ず作成できます。このチェックは、オプション *limitcheck=false* を用いて無効にすることも可能です。

PLOP でできないこと　以下の制約に留意してください：

- PLOP はクラッカーツールではありません。これを用いて、保護された文書に対するアクセスを、適切なユーザーまたはマスターパスワードを知ることなく得ることはできません。なぜならこれは文書作成者の意図に反することだからです。
- 動的 XFA フォームを処理することはできません。なぜならそれは純正 PDF 文書ではなく、薄い PDF レイヤー内にパッケージされた XML フォームだからです。

2 PLOP DS の諸機能（電子署名）

注記 PDF 文書に電子的に署名する機能は PLOP DS でのみ利用可能であり、PLOP 基本製品では利用できません。

PDF 文書に対する電子署名については、詳しくは 6 章「PLOP DS による電子署名」（67 ページ）で網羅します。この章では、出発点として、概要と、最初の例を提供します。

2.1 PLOP DS の署名機能

さまざまな PDF 署名機能

- 既存の PDF 署名フィールド内に署名を作成、もしくは署名を保持する新規のフィールドを生成。この署名は、ページ上の特定の位置において不可視にすることも可視にすることも可能です。
- ロゴや手書き署名のスキャン等の表現を PDF ページとして取り込むことによって電子署名を視覚化。
- 署名を破壊することなくフォーム記入等の文書変更ができるよう許可する PDF 認証（作成者）署名を作成。
- 検証情報を、ISO 32000-1 に従って署名内に直接格納することもできますし、ISO 32000-2 と PAdES パート 4 で仕様化されているように文書セキュリティストア（DSS）内に格納することもできます。
- 署名を、増分的な PDF 更新セクション内に行うことによって既存の署名群と文書構造を温存することもできますし、最適化・暗号化のために文書構造を書き換えることによって行うこともできます。

PDF のさまざまなバージョンと規格 PLOP DS は、あらゆる標準的な PDF のバージョンと規格に対応しています：

- PLOP DS は、Acrobat XI すなわち PDF1.7（ISO 32000-1）拡張レベル 8 までのすべての PDF バージョンを処理します。PLOP DS は、将来の規格 PDF 2.0（ISO 32000-2）に準拠した文書を処理することもできます。
- PLOP DS は、PDF/A-1/2/3（ISO 19005）アーカイビング規格群に対応しています：入力文書が PDF/A に準拠していれば出力文書も準拠が保証されます。PLOP DS は、PDF/A に要求される XMP 拡張スキーマに完全対応しています。PDF/A 準拠の XMP メタデータを PDF 文書に挿入できる点は PLOP の重要な特長です。
- 同様に PLOP DS は、PDF/X-1a/3/4/5（ISO 15930）印刷業務規格群と、トランザクション印刷のための PDF/VT-1/2（ISO 16612-2）と、アクセシブル PDF のための PDF/UA-1（ISO 14289）に対応しています。

さまざまな署名規格

- ISO 32000-1 と将来の ISO 32000-2 に従った標準の PDF 署名
- Acrobat XI に従った長期検証（LTV）のための署名
- ETSI TS 102 778 パート 2・3・4 と CAdES（ETSI TS 101 733）に従った PAdES（PDF 高度電子署名）。ETSI TS 103 172 に拠る PAdES 準拠レベル PAdES-B（基本）・PAdES-T（信頼時間）・PAdES-LT（長期）・PAdES-LTA（アーカイブタイムスタンプ付長期）を含みます。PAdES パート 3 に従った PAdES-BES（基本電子署名）と PAdES-EPES（明示的ポリシーベース電子署名）の両方に対応しています。

暗号署名の詳細

- RSA・DSA アルゴリズムに加え、楕円曲線暗号法に基づく楕円曲線電子署名アルゴリズム（ECDSA）に従った署名。NIST が推奨している楕円曲線群に加え、Brainpool 等の曲線にも対応しています。
- NSA の Suite B 暗号法に従った強固な署名・ハッシュ関数。
- 生成される署名内に完全な証明書チェーンを埋め込み。したがって、Adobe 認定信頼リスト（AATL）に載っている CA（認証局）からの証明書を持った署名を、クライアント側で何ら構成を必要とせず Acrobat・Adobe Reader 内で検証できます。
- オンライン証明書状態プロトコル（OCSP.。RFC 2560・RFC 6960 に拠る）応答と証明書失効リスト（CRL。RFC 3280 に拠る）を長期検証（LTV）のための失効情報として埋め込み。

タイムスタンピング

- RFC 3161 に従って、信頼されたタイムスタンプ局（TSA）からタイムスタンプを取得し、生成する署名内に埋め込み。TSA の詳細を AATL 証明書から読み取って構成を何ら要せずタイムスタンプを作成することも可能です。
- ISO 32000-2 と PAdES パート 4 に従って文書レベルタイムスタンプ署名を作成。文書レベルタイムスタンプは、個人署名を行うことなく文書の状態を保証します。
- タイムスタンプ *policy* 引数と、広く利用されているすべてのタイムスタンプハッシュ関数に対応。

さまざまな署名エンジン　PLOP DS は、複数の暗号化エンジンに、すなわち電子署名を生成するためのコンポーネントに対応しています：

- 内蔵のエンジンは , 必要な暗号化機能を PLOP DS 内に、一切の外部依存なく実装しています。この内蔵エンジンは、広く用いられている PKCS#12・PFX 形式のソフトウェアベースのデジタル ID に対応しています。
- PLOP DS は暗号化トークンを、標準の PKCS#11 インタフェースを通じて紐付けることができます。この方法で、スマートカード・USB スティック等セキュアデバイス上のデジタル ID を用いて署名を行うことが可能です。セキュアな PIN 入力のためのキーボードが付いた機器でも同様です。
- Windows では PLOP DS は、Windows が Microsoft Cryptographic API（MS CAPI）を通じて提供している暗号化インフラストラクチャを活用することができます。ソフトウェアベースのデジタル ID やセキュアハードウェアトークンに加えて Windows 証明書ストアからのデジタル ID を用いて署名を行うことが可能です。ただし LTV 等 MSCAPI エンジンでは利用できない署名機能もあります。

PLOP DS でできないこと　以下の制約に留意してください：

- PLOP DS を用いて PDF 文書を Reader 有効化する（Adobe Reader での注釈作成を許す等）ことはできません。なぜならこれには特定の Adobe 署名が必要だからです。
- 静的または動的 XFA フォームに署名することはできません。

2.2 PLOP DS の評価のための準備

PDFlib デモ CA 証明書を Acrobat にインストール　以下の手順は、PLOP DS を用いて電子書名を作成するために必須ではありません。しかし PLOP DS を、そのパッケージ内で提供されているサンプル証明書群を用いて評価しようとするならば、以下の説明のように Acrobat を構成することを推奨します。これは、Acrobat の信頼済み証明書のリスト

(「Acrobat における信頼済みルート証明書」(70 ページ) 参照) にインストールされている商用 CA からの証明書で作業する場合には必要ありません。

PLOP DS に含まれているサンプル証明書群は PDFlib Demo CA によって発行・署名されています。この CA の自己署名ルート証明書を Acrobat で利用可能にすれば、生成される署名群は Acrobat 内で完全に有効として受け入れられます。PDFlib Demo CA 証明書を Acrobat XI にインストールするには以下のように操作します:

- **「編集」→「環境設定」→「一般 ...」→「署名」→「*ID* と信頼済み証明書」→「詳細 ...」→「信頼済み証明書」→「追加」→「参照 ...」**をクリック
- ***bind/data/PDFlibDemoCA.crt*** (PLOP インストレーションの一部) へブラウズし、**「追加」→「*OK*」**をクリック。
- これで、信頼済み証明書のリストの中にエントリ**「*PDFlib GmbH Demo CA*」**が現れます。
- このエントリを選択し、**「信頼を編集」**をクリックして、ボタン**「この証明書を信頼済みのルートとして使用」**・**「証明済み文書」**を有効にし、**「*OK*」**をクリックします。

デモデジタル ID を Windows に取り込む Windows 上で PLOP DS の MSCAPI ベースの署名エンジンを試すには、Windows 証明書ストアでデジタル ID を利用可能にする必要があります。デモデジタル ID を取り込むには、照応する ***.p12*** ファイルをダブルクリックして「証明書のインポートウィザード」を起動し、その画面に従います。

2.3 PLOP DS で文書に署名

署名を行うにはデジタル ID が必要です。デジタル ID は、ファイルとして、または Windows 証明書ストア内で、あるいは暗号トークン (スマートカードや USB スティック等) 上で利用できます。ファイルの場合には、そのデジタル ID にアクセスするにはパスワードが必要ですが、Windows 証明書ストアは通常、Windows ログインによって保護されており、パスワードを必要としません。暗号トークンは多くの場合、PIN によって保護されており、その PIN は、署名側ソフトウェアによって、あるいはそのトークンの内蔵キーボード上で直接、与えられる必要があります。

電子署名を作成するには ***PLOP_prepare_signature()*** を用います。いくつかのオプションを使えます。その後、その署名を行うには ***PLOP_create_document()*** を用います。PDF 文書に署名を行うためのサンプルコードが、すべての PLOP パッケージに含まれている ***sign・multisign*** ミニサンプル内にあります。PLOP コマンドラインツールに対してこれと同等のオプションは***--signopt*** です。

基本的な署名のオプションリストの例 PDF 文書に対して、ファイル ***demorsa2048.p12*** からのデジタル ID を用いて、不可視署名を作成します。このデジタル ID に対するパスワード ***demo*** がファイル ***pw.txt*** の内容となっています:

```
plop --signopt "digitalid={filename=demorsa2048.p12} passwordfile=pw.txt" ↵
        --outfile signed.pdf input.pdf
```

(Windows のみ) PDF 文書に対して、Windows 証明書ストアからの (デフォルトストア ***My*** からの) 証明書を用いて、不可視署名を作成します。これは、このデジタル ID は自分の Windows ログインによって保護されているので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={store=My subject={PDFlib Demo PLOP User 2048}}" ↵
         --outfile signed.pdf input.pdf
```

（PKCS#11 対応を有するプラットフォームのみ）PDF 文書に対して、暗号トークンからのデジタル ID を用いて、不可視署名を作成します。このトークンに対する PKCS#11 インタフェースは、そのスマートカードサプライヤーによって提供される必要のある ***cryptoki.dll*** ライブラリ内に実装されています。このデジタル ID に対するパスワードはファイル ***pw.txt*** 内に含まれています：

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

さらに詳しくは 6.2 節「PLOP DS の各種暗号化エンジン」（72 ページ）をごらんください。

2.4 証明用署名

証明用または作成者署名は、その文書をその作成者が作成した時点におけるその状態を証明すると同時に、その証明を破ることなくある種の変更を許すものです。***certification*** オプションは、フォーム記入許可等、その証明された文書に対してその署名を破ることなく行うことのできる変更群を指定します：

```
plop --signopt "digitalid={filename=demorsa2048.p12} passwordfile=pw.txt ↵
           certification=formfilling" ↵
           --outfile certified.pdf input.pdf
```

2.5 タイムスタンプ

署名にタイムスタンプを追加するには、タイムスタンプ局の URL が必要であり、それを ***timestamp*** オプションに与える必要があります：

```
plop --signopt "digitalid={filename=demorsa2048.p12} passwordfile=pw.txt ↵
       timestamp={source={url={http://timestamp.acme.com/tsa-noauth/tsa}}}" ↵
       --outfile signed.pdf input.pdf
```

同様に、文書レベルタイムスタンプを、***doctimestamp*** オプションを用いて適用できます：

```
plop --signopt ↵
       "doctimestamp={source={url={http://timestamp.acme.com/tsa-noauth/tsa}}}" ↵
       --outfile signed.pdf input.pdf
```

さらに詳しくは 6.5 節「タイムスタンプ」（94 ページ）をごらんください。

2.6 LTV 有効化署名

長期検証（long-term validation ＝ LTV）への対応には、チェーン内のすべての証明書が利用可能であり、かつ、証明書失効情報が署名の作成時にオンラインかディスクファイルから取得できることが必須です。このためには、然るべき OCSP または CRL サーバ群が PKI によって提供されている必要があります。多くの場合（とりわけ AATL 証明書）、必要なネットワーク情報は署名証明書から読み取ることが可能です。そうでない場合には、然るべきネットワークリソースを、***ocsp*** および／または ***crl/crlfile/crldir*** オプションを通じて与える必要があります。証明書チェーン全体へのアクセスを与えるために、そのルート CA 証明書を内容とする PEM ファイルの名前をオプション ***rootcertfile*** で与える必要があります。

LTV 有効化署名は通常、用いられている証明書群に対するオンライン PKI リソース群（CRL または OCSP）を必要とします。これは PLOP デモ証明書に対しては利用できません。次善策として、ディストリビューション内で提供されている CRL ファイル *PDFlibDemoCA.crl* を利用できます（この CRL の失効日は、業務環境であれば受け入れがたいような非常に遠い未来になっています）。これに照応する、LTV 有効化署名を作成するためのコマンドライン呼び出しは、以下のようになります：

```
plop --signopt "digitalid={filename=demorsa2048.p12} password=demo ltv=full ↵
        crlfile=PDFlibDemoCA.crl rootcertfile=PDFlibDemoCA.pem" ↵
        --outfile ltv-signed.pdf input.pdf
```

次の例では、必要な OCSP または CRL 取得情報は署名証明書内に存在していると前提しています。商用証明書では通常そのようになっています。これらの条件下においては、オプション *ltv=full* を与えることにより、必ず LTV 有効化署名が作成されるようにすることができます：

```
plop --signopt "digitalid={filename=signer.p12} passwordfile=pw.txt ltv=full ↵
        rootcertfile=RootCA.pem" --outfile ltv-signed.pdf input.pdf
```

関与する PKI の内容によってはこれでは充分でない可能性があることに留意してください。特に、失効情報は、OCSP/CRL 証明者とタイムスタンプ局に対しても利用可能となっている必要があります。

2.7 PAdES 署名

各種 PAdES 署名は、PDF 署名を改良して、EU の諸要請を満たすようにしたものです。さまざまな署名オプションを用いて、さまざまな種類の PAdES に従った署名を作成することができます。たとえば、以下のコマンドラインは、PAdES パート 3（PAdES-B）に従った基本署名を作成します：

```
plop --signopt "digitalid={filename=demorsa2048.p12} passwordfile=pw.txt ↵
        sigtype=cades" ↵
        --outfile signed.pdf input.pdf
```

以下のコマンドラインは、明示的ポリシー識別子を伴った PAdES パート 3（PAdES-EPES）に従った署名を作成します：

```
plop --signopt "digitalid={filename=demorsa2048.p12} passwordfile=pw.txt ↵
        sigtype=cades policy={oid=2.16.276.1.89.1.1.1.1.3 commitmenttype=origin}" ↵
        --outfile signed.pdf input.pdf
```

さらに詳しくは 6.7 節「CAdES・PAdES 署名規格」（104 ページ）をごらんください。

2.8 電子署名を視覚化

電子署名を、企業ロゴや手書き署名のスキャン等によって、視覚化することができます。その視覚表現は、PDF 文書として与えられる必要があります。これは署名フォームフィールド内に配置されます。入力文書が署名フィールドをまだ含んでいない場合には、然るべきフィールド座標を与える必要があります。以下のコマンドラインは、視覚化文書 *signing_man.pdf* をフィールド長方形内に配置します：

```
plop --signopt "digitalid={filename=demorsa2048.p12} passwordfile=pw.txt ↵
        field={name=Signature1 rect={10 10 adapt adapt}}" --visdoc signing_man.pdf ↵
        --outfile signed.pdf input.pdf
```

さらに詳しくは 6.3.1 節「署名をグラフィックかロゴで視覚化」（79 ページ）をごらんください。

2.9 署名プロパティをクエリ

PLOP DS に内蔵されている pCOS プログラミングインタフェースを用いて、PDF 文書の署名設定をクエリすることができます。pCOS クックブックのトピック ***interactive_elements/signatures*** が、署名の種別と内容をクエリする方法を演示しています。pCOS を用いて文書情報をクエリするためのサンプルコードが、すべての PLOP パッケージに含まれている ***dumper*** ミニサンプル内にあります。PLOP コマンドラインツールに対してこれに照応するオプションは***--info*** です（1.5 節「pCOS で文書情報を取得」（19 ページ）を参照）：

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

このプログラム呼び出しは、以下のような出力を生成します：

```
       File name: hellosign.pdf
     PDF version: 1.7
      Encryption: No encryption
      ...
      Tagged PDF: false
      Signatures: 1
        signature field 'Signature1': invisible approval signature, CAdES
  Reader-enabled: false
```

3 PLOP・PLOP DS コマンドラインツール

3.1 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（角カッコ内のエントリはオプショナルです）：

```
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す。ライブラリのオプションを、7 章「PLOP・PLOP DS ライブラリ API リファレンス」（107 ページ）のオプション一覧に従って、*--inputopt*・*--outputopt*・*--plopopt* オプションを使って与えることができます。表 3.1 にすべての PLOP コマンドラインオプションを挙げます。

表 3.1 PLOP コマンドラインオプション

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 – キャラクタで始まる場合に有用です。
@filename[1]		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」（36 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。
--help, -?（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。
--info, -i		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。
--inputopt	< オプションリスト >	PLOP_open_document( ) 用のオプションリストを指定できます（表 7.3（113 ページ）参照）
--master[2]*, -m*	< 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。
--noreplace, -n		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す。

表 3.1 PLOP コマンドラインオプション

オプション	引数	機能
--outfile, -o	< ファイル名 >	(--info の場合を除いて、ちょうど 1 個の入力文書が必須です。--outfile と --targetdir 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。
--outputopt	< オプションリスト >	PLOP_create_document( ) 用のオプションリストを指定できます (表 7.5 (116 ページ) 参照)
--password[2], -p	< 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。
--permissions[2]	< 権限群 >	(--master 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 noprint・nomodify・nocopy・noannots・noassemble・noforms・noaccessible・nohiresprint・plainmetadata を任意の数含みます (表 5.3 (62 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) : ***keep*** 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 : keep noprint
--plopopt	< オプションリスト >	PLOP_set_option( ) 用のオプションリストを指定できます (表 7.11 (133 ページ) 参照)。これを使って、license または licensefile オプションを渡すことが可能です。
--recsize, -R	< 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)
--searchpath, -s[1]	< パス >	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
--signopt, -S	< 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、PLOP_prepare_signature( ) に対するオプションリスト (表 7.7 (122 ページ) 参照)。
--targetdir, -t	< ディレクトリ名 >	(--outfile と --targetdir 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。
--tempdirname	< 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
--tempfilename, -T	< 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
--user, -u[2]	< パスワード >	(--master を必要とします) 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。
--verbose, -v	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

表 3.1 PLOP コマンドラインオプション

オプション	引数	機能
--visdoc	〈ファイル名〉	（--signopt とともにのみ）電子署名を視覚化するために用いられるページを含んだ PDF ファイルの名前。
--visdocopt	〈オプションリスト〉	（--visdoc とともにのみ）署名視覚化文書を開くために用いられる、PLOP_open_document( ) に対するオプション群（表 7.3（113 ページ）参照）
--webopt, -w		PDF 出力を Web 配信のために線形化します。これは Web 最適化としても知られています。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。

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す。
- 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：***--plopopt*** 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 3.1 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かを調べ（表 5.2 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継続します。

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 ″ で囲う必要があるのです。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。ワイルドカードは、検索パスディレクトリ内では一切評価されず、カレントディレクトリ内で評価されます。

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。

3.2 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 つの形式で示してあり、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「PLOP の諸機能」（15 ページ）（さまざまな節）
- 5.3 節「コマンドラインで PDF 文書を保護」（64 ページ）
- 6.2.2 節「内蔵エンジン」（73 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。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 *demorsa2048.p12* の中のデジタル 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=demorsa2048.p12} passwordfile=pw.txt" ↵
        --outfile signed.pdf input.pdf
plop -S "digitalid={filename=demorsa2048.p12} passwordfile=pw.txt" -o signed.pdf ↵
        input.pdf
```

署名を作成し、手書き署名を内容とする既存 PDF をその署名を視覚化するために使用：

```
plop --signopt "digitalid={filename=demorsa2048.p12} passwordfile=pw.txt ↵
        field={rect={100 100 300 adapt}}" --visdoc signature.pdf ↵
        -outfile signed.pdf input.pdf
```

4 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。

4.1 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、***ploplibdl.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。

エラー処理 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います。
- 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例外を扱う典型的イディ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。**

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 *ploplibdl.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 *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。

4.2 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パが 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++ オブジェクトラッパを利用することで、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。

C++ の文字列処理 PLOP 4.1 で、新しい Unicode 対応の C++ バインディングを導入しました。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 キャラクタを持つことができます。これは PLOP 4.1 以来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。

注記 EBCDIC ベースのシステム群では、`wstring` ベースのインタフェースのためのオプションリスト文字列のフォーマッティングは、オプションリスト内に EBCDIC と UTF-16 の `wstring` が混在することを避けるために、さらなる変換を必要とします。この変換のための簡便なコードと使用法が、追加モジュール `utf16num_ebcdic.hpp` 内にあります。

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、以下のようにして変更することができます：

- PLOP 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べきです：

```
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・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 {
        ...各種PLOP命令...
} catch (PLOP::Exception &ex) {
            wcerr << L"Error " << ex.get_errnum()
            << L" in " << ex.get_apiname()
            << L"(): " << ex.get_errmsg() << endl;
}
```

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」（40 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してあってもなくても、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。

4.3 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エントリを作成し、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（4.5 節「.NET バインディング」（48 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。

4.4 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 ラッパライブラリと 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* ファイルの中にあり、*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。
- 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 サーブレットと 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、とりわけサーブレットに完全に適合しています。特定のサー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：

- サー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、サー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。自分のサー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。
- サー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。サー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 をサー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;
        /* ... 各種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.get_errmsg());
} finally {
        /* PLOPオブジェクトを削除 */
        if (plop != null) plop.delete();
}
```

4.5 .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ラッパを付けたもの）。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断させます。

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;
```

4.6 Objective-C バインディング

C・C++ 言語バインディングを Objective-C[1] で使用することもできますが、Objective-C 用の純正の言語バインディングも利用できます。以下の種類の PLOP フレームワークがあります：

- ***PLOP***：OS X で使用
- ***PLOP_ios***：iOS で使用

どちらのフレームワークも、C・C++・Objective-C 用の言語バインディングを内容として持っています。

PLOP の Objective-C 版を OS X にインストール　自分のアプリケーションの中で PLOP を使用するには、***PLOP.framework*** か ***PLOP_ios.framework*** をディレクトリ ***/Library/Frameworks*** へ複製する必要があります。他の場所へ PLOP フレームワークをインストールすることも可能ですが、Apple の ***install_name_tool*** を使用する必要があります。それについてここでは説明しません。PLOP メソッド宣言を有する ***PLOP_objc.h*** ヘッダファイルをアプリケーションソースコードへインポートする必要があります：

```
#import "PLOP/PLOP_objc.h"
```

または

```
#import "PLOP_ios/PLOP_objc.h"
```

引数の命名規則　PLOP メソッドの呼び出しの際には、引数を以下の規則に従って与える必要があります：

- 1 個目の引数の値は、メソッド名の直後に、コロンキャラクタ 1 個で区切って与えます。
- それより後の各引数については、それぞれ、その引数の名前とその値を（これもコロンキャラクタ 1 個で互いを区切って）与える必要があります。さまざまな引数の名前は、7 章「PLOP・PLOP DS ライブラリ API リファレンス」（107 ページ）と ***PLOP_objc.h*** 内にあります。

たとえば、API 解説における以下の行は：

```
int open_page(int doc, int pagenumber, String optlist)
int open_document(wstring filename, wstring optlist)
```

以下の Objective-C メソッドに照応します：

```
- (NSInteger) open_document: (NSString *) filename optlist: (NSString *) optlist;
```

ですので、アプリケーションからは以下のような呼び出しを行う必要があります：

```
doc = [plop open_document:filename optlist:pageoptlist];
```

コード補完のための XCode Code Sense を PLOP フレームワークで利用できます。

Objective-C におけるエラー処理　Objective-C バインディングは、PLOP エラーをネイティブ Objective-C 例外へ翻訳します。実行時の問題が起きた場合には、PLOP はクラス

1.　developer.apple.com/library/mac/documentation/Cocoa/Conceptual/ProgrammingWithObjectiveC/Introduction/Introduction.html を参照

PLOPException のネイティブ Objective-C 例外を発生させます。これらの例外は、通常の *try/catch* 機構を用いて処理できます：

```
@try {
        ...各種PLOP命令...
}
@catch (PLOPException *ex) {
        NSString * errorMessage =
            [NSString stringWithFormat:@"PLOPエラー %d が '%@'で発生しました: %@",
            [ex get_errnum], [ex get_apiname], [ex get_errmsg]];
        NSAlert *alert = [[NSAlert alloc] init];
        [alert setMessageText: errorMessage];
        [alert runModal];
        [alert release];
}
@catch (NSException *ex) {
        NSAlert *alert = [[NSAlert alloc] init];
        [alert setMessageText: [ex reason]];
        [alert runModal];
        [alert release];
}
@finally {
        [plop release];
}
```

この *get_errmsg* 方式以外にも、例外オブジェクトの *reason* フィールドを用いてエラーメッセージを取得することもできます。

4.7 Perl バインディング

Perl 用 PLOP ラッパは、1 個の C ラッパと 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 ラッパとモジュール ***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***（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のようになります：

```
/usr/lib/perl5/site_perl/5.16/i686-linux
```

Windows　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のようになります：

```
C:\Program Files\Perl5.16\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 {
        ...各種PLOP命令...
};
die "例外をキャッチしました: $@" if $@;
```

4.8 PHP バインディング

注記 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、ディストリビューションパッケージ内に含まれている、PDFlib の Web サイトにもある *PDFlib-in-.NET-HowTo.pdf* 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。

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。

PHP が外部の PLOP ライブラリを認識するよう PHP を構成する必要があります。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・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 の場合はそこ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 and Digital Signature) =>
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・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 版とでは、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 での例外処理　PHP 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 {

...各種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;
}
```

Eclipse と Zend Studio で開発　PHP Development Tools（PDT）[1] は、Eclipse と Zend Studio を用いた PHP 開発に対応しています。PDT は、以下に概略示す操作によって、文脈依存ヘルプに対応するよう構成することもできます。

PLOP がすべての PHP プロジェクトから認識されるよう PLOP を Eclipse 設定に追加します：

- **「ウィンドウ」→「設定」→「*PHP*」→「*PHP* ライブラリー」→「新規 ...」**を選択します。すると、ウィザードが起動します。
- **「ユーザー・ライブラリ名」**に**「*PLOP*」**と入力し、**「外部フォルダーの追加 ...」**をクリックし、フォルダ ***bind\php\Eclipse PDT*** を選択します。

既存または新規の PHP プロジェクトにおいて、PLOP ライブラリへの参照を以下のように追加できます：

- PHP エクスプローラー内でその PHP プロジェクトを右クリックし、**「インクルード・パス」→「インクルード・パスの構成 ...」**を選択します。
- **「ライブラリー」**タブへ移動し、**「ライブラリーの追加」**をクリックして、**「ユーザー・ライブラリー」→「*PLOP*」**を選択します。

これらの操作の後、PLOP メソッドの一覧を、PHP エクスプローラービュー内の ***PHP Include Path/PLOP/PLOP*** ノード下で閲覧することができます。新規の PHP コードを書く時、Eclipse は、すべての PDFlib メソッドについて、コード補完と文脈依存ヘルプで支援します。

1. www.eclipse.org/pdt を参照

4.9 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・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:
        ...各種PLOP命令...
except PLOPException:
        print 'PLOP例外をキャッチしました!'
```

4.10 Ruby バインディング

PLOP の Ruby 版をインストール　Ruby[1] 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、Ruby インタプリタが Ruby 用 PLOP 拡張ライブラリへアクセスできるようになっている必要があります。このライブラリ（Windows・Unix では ***PLOP.so***。OS X では ***PLOP.bundle***）は通常、ローカルの Ruby インストレーションディレクトリの ***site_ruby*** ブランチ内に、すなわち以下のような名前のディレクトリの中にインストールされます：

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

ただし、Ruby は他のディレクトリ群へも拡張を探しに行きます。このディレクトリのリストを取得するには以下のルビー呼び出しを使用できます：

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

このリストは通常、カレントディレクトリを含んでいますので、試験目的のためには、単に PDFlib 拡張ライブラリとスクリプト群を同一ディレクトリ内に置けば足ります。

データ型　引数は、PLOP API へ、に挙げるデータ型に従って渡す必要があります。

表 4.1　Ruby バインディングにおけるデータ型

API データ型	Ruby バインディングにおけるデータ型
文字列データ型	string
バイナリデータ型	string

Ruby におけるエラー処理　Ruby バインディングは、PLOP 例外をネイティブ Ruby 例外へ翻訳するエラーハンドラをインストールします。この Ruby 例外は通常の ***rescue*** 技法で扱えます：

```
begin
        ...各種PLOP命令...
rescue PLOPException => pe
  print "PLOP例外が暗号化サンプル内で発生しました:\n"
  print "[" + pe.get_errnum.to_s + "] " + pe.get_apiname + ": " + pe.get_errmsg + "\n"
end
```

1. www.ruby-lang.org/ja を参照

5 PDF 暗号化・復号

5.1 さまざまな PDF 暗号化機能

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。

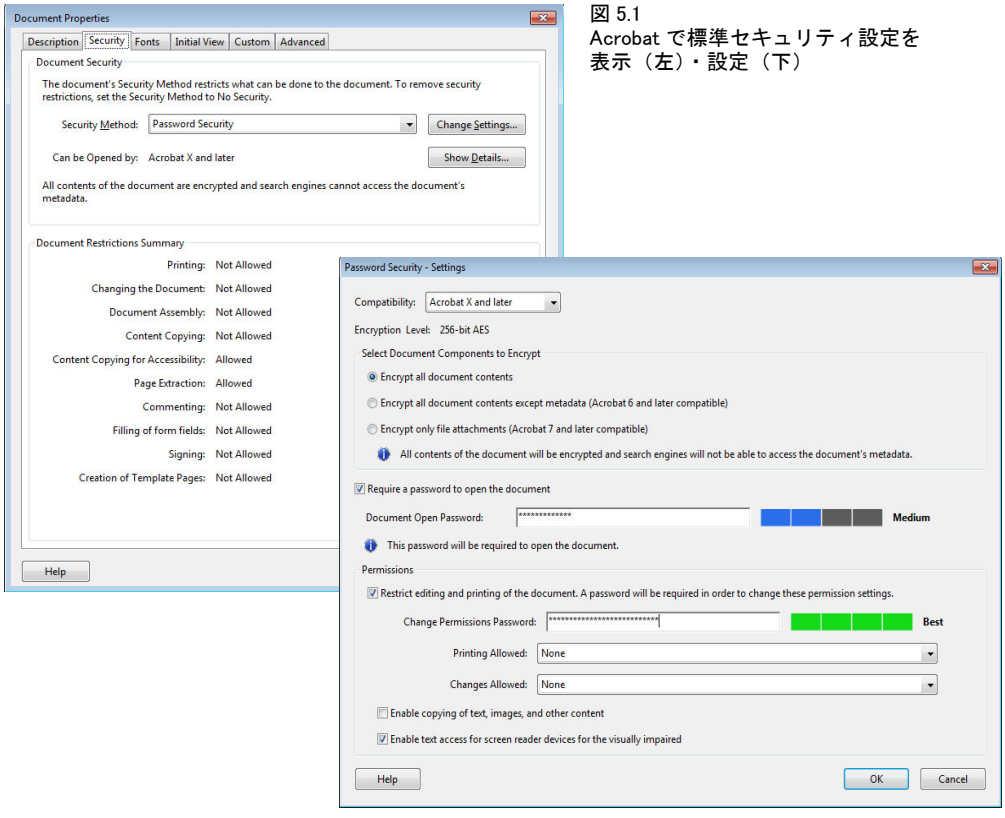


図 5.1
Acrobat で標準セキュリティ設定を表示（左）・設定（下）

図 5.1 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。128 ビットまたは 256 ビットキーを用いた AES 暗号化は Suite B 暗号法において必須です。

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 5.1 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 5.1 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 RC4 128 ビット、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（使用するべきではありません）	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 5.1 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と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、パスワードの長さと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マ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す：

「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込むことができます。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いてはいけません。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。
- 弱い RC4 アルゴリズムと、PDF 1.7ext3（Acrobat 9）に従った AES-256 は避けるべきです。パスワード確認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/XI と PLOP 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/XI 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。

5.2 PLOP による PDF 暗号化

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　文書を保護するための暗号化の詳細は、入力文書の PDF バージョンと、*PLOP_create_document()* の *encryption* オプションに依存します。

PLOP は、弱い RC4 暗号化アルゴリズムを決して使用しません。PDF 1.7ext3（Acrobat 9）に従った AES-256 は、パスワード確認アルゴリズム内に脆弱性を持っており、それがパスワードに対するブルートフォース攻撃を可能にします。このため、Acrobat X/XI と PLOP は決して暗号化に Acrobat 9 暗号化を使用しません（既存文書の復号にのみ使用します）。

デフォルトでは PLOP は PDF 1.6 以上を生成します。他の機能、とりわけ電子署名が、PDF 出力バージョンをさらに押し上げる場合があります。この場合には、その押し上げられたバージョン番号が、元の PDF バージョンのかわりに基本として用いられます。暗号化アルゴリズムの選択の具体的過程は以下の通りです：

- PDF 1.7ext3 以下の入力、または *encryption=algo4* が与えられている場合：PDF バージョンは、必要であれば PDF 1.6 へ押し上げられ、かつ、Acrobat 7/8（pCOS アルゴリズム 4）に従った AES-128 暗号化が使用されます。パスワードは Latin-1 キャラクタのみを内容とすることができ、32 キャラクタへ切り詰められます。
- PDF 1.7ext8・PDF 2.0 入力、または *encryption=algo11* が与えられている場合：PDF バージョンは、必要であれば PDF 1.7ext8 へ押し上げられ、かつ、Acrobat X/XI（pCOS アルゴリズム 11）に従った AES-256 暗号化が使用されます。パスワードは Unicode キャラクタを内容とすることができ、127 UTF-8 バイトへ切り詰められます。

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（1.2 節「Web 最適化（線形化）PDF」（16 ページ）参照）、マスターパスワードが必要です。

表 5.2 に、すべての操作について何が必要かをまとめました。

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 5.2 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

表 5.2 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
マスター	可能	可能	可能	可能

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えることができ、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。これよりも古い暗号化アルゴリズムでは、Latin-1 文字集合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 文字集合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 5.3 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 5.3 に、使える権限キーワードを列挙します。

表 5.3 PLOP_create_document( ) の permissions オプションに対するアクセス制限キーワード

キーワード	説明
noprint	Acrobat でそのファイルが印刷できなくなります。
nomodify	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。
nocopy	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。
noannots	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。
noforms	（暗黙に noannots と見なされます）Acrobat でフォームフィールドへの記入ができなくなります。noannots が指定されていなくても同様です。
noaccessible	（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 5.3 PLOP_create_document( ) の permissions オプションに対するアクセス制限キーワード

キーワード	説明
noassemble	（暗黙に nomodify と見なされます）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。
nohiresprint	Acrobat で高解像度印刷ができなくなります。noprint 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。
plain-metadata	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。

5.3 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_document()* 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_document()* 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 文字集合外の Unicode キャラクタを含むことも可能です。

さまざまな暗号化の例 以下のさまざまなサンプルコマンドライン呼び出しは、長いコマンドラインオプションで示しています。短縮形のコマンドラインオプションについては 3.1 節「PLOP・PLOP DS コマンドラインオプション」（33 ページ）を参照してください。

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
```

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
```

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
```

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
```

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。PLOP はデフォルトで、強い AES 暗号化を使用します。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」（60 ページ）参照）：

```
plop --password old --master DEMO --outputfile 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 暗号化が適用されます。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 --master DEMO ↵
       --permissions "noprint nocopy noannots" --targetdir 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
```

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
```

6 PLOP DS による電子署名

注記 PDF 文書に電子的に署名を行う機能は、PDFlib PLOP DS でのみ利用可能であり、PLOP 基本製品では利用できません。

6.1 はじめに

6.1.1 署名の諸概念

電子署名を詳細に解説することはこのマニュアルの範囲外です。しかし、PLOP DS を用いて PDF 文書に電子的に署名を行う際に役割を担ういくつかの重要な構成要素について触れます。これらの構成要素は、全体として公開鍵基盤（Public Key Infrastructure ＝ PKI）を形成します。

電子署名は、公開鍵暗号法に基づいています。公開鍵暗号法のことを非対象暗号化ともいいます。これは、文書に署名した人のみが入手可能な秘密鍵と、万人がその署名を検証できるよう万人が入手可能な公開鍵によって働きます。

証明書 公開鍵は一般に、その署名者の公開鍵と彼の名前と連絡先詳細を内容とするいわゆる証明書ファイルの中で頒布されます。偽造証明書を防ぐために、この情報パッケージはさらに、人物やその他の企業やサーバ等といった主体へ証明書を発行す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のことを、認証局（Certificate Authority ＝ CA）やセキュリティセンター（Trust Center ＝ TC）といいます。CA 自身の証明書のことを、ルート証明書といいます。それは多くの場合、万人がダウンロードできるようその CA のウェブサイト上で公開されています。証明書は通常、X.509 形式で保管されます。

証明書チェーン ある CA によって発行された署名用証明書は、その発行した CA が、または、より高次の、その中間 CA の証明書を発行した CA が、信頼できると見なされるならば、信頼できると見なされます。ルート CA から、文書を署名するために実際に使用されている最下端のエンドユーザー証明書まで、順繰りに次の証明書に署名することによってつながっている証明書群のリストのことを、証明書チェーンといいます。このチェーンの中の最上位の CA の証明書のことを、ルート証明書といいます。ある署名が有効であると見なされるためには、そのチェーンの中のすべての証明書が有効である必要があります。

デジタル ID 証明書を、その証明書とそれに照応する秘密鍵の両方を内容とするパッケージと区別することは、重要です。このパッケージをデジタル ID といいます。証明書は万人に自由に頒布できるのに対し、デジタル ID は注意深く保護する必要があります。なぜならデジタル ID は秘密情報（秘密鍵）を内容としているからです。電子署名を行うためにデジタル ID 内の秘密鍵にアクセスするには通常、パスワードかパスフレーズが必要です。デジタル ID の保管形式として広く用いられているのは PKCS#12（Windows では PFX ともいいます）です。証明書とデジタル ID は常にはっきり区別されるわけではないことに留意してください：正確には**「デジタル *ID* を用いて文書に署名する」**と言うべきところを**「証明書を用いて署名する」**と言っていることはちまたに多いです。

証明書失効確認 証明書は、ある特定の期間にわたり有効です。その失効日が過ぎれば、あるいは、その CA によって明示的に失効させられた場合には、ただちに無効になります。

証明書を失効させることは、その証明書保持者が関連組織を去った、あるいは、その秘密鍵が破られたという理由で、必要となる場合があります。

証明書確認は通常、OCSP（Online Certificate Status Protocol ＝オンライン証明書ステータスプロトコル）または証明書失効リスト（certificate revocation list ＝ CRL）というプロトコルを使用したオンラインクエリの過程を経ます。両方式に関して詳しくは、「OCSP の概要」（88 ページ）・「CRL の概要」（90 ページ）を参照してください。

タイムスタンピング　タイムスタンプは、ある特定の時点の表現に対して電子署名を行います。この際に、その時刻は、信頼された正確な時刻情報源から取得されることができます。タイムスタンプを通常の署名の中に内蔵させることによって、ある特定の時点よりも前にその署名（ひいてはその署名がされた文書）が存在していたことを保証することも可能です。タイムスタンプを PDF 文書に対して別途行うことも可能です。タイムスタンピングサーバとプロトコルの詳細に関する詳しい情報については、6.5.1 節「構成」（94 ページ）を参照してください。

デジタル ID の取得元　デジタル ID は、さまざまな取得元から取得できます。多くの ID は電子メールに署名することを意図されています。これらの電子メール ID を用いて PLOP DS で PDF 文書に署名することも可能です。デジタル ID についてどの取得元を選ぶかは、必要な ID の数（従業員ごとに 1 つずつか、それとも会社 ID 1 個のみか等）と、求める制御の程度によって異なります：

- 商用またはフリーの ID を発行しているパブリック CA のうちの一つからデジタル ID を取得します。Acrobat による署名検証を可能とするために、Acrobat 内で信頼済みルートとしてインストールされている CA からのデジタル ID を用いて署名を作成することを推奨します（「Acrobat における信頼済みルート証明書」（70 ページ）参照）。
- より大きな組織の場合：自前のプライベート CA を構築することにより、デジタル ID を自分で作成できるようにします。CA を構築するためのさまざまなソフトウェアパッケージがあります。たとえばフリーの OpenSSL ソフトウェア（***www.openssl.org*** 参照）や、Java の構成要素である ***keytool*** アプリケーション、Microsoft Windows Server オペレーティングシステムに含まれている Certificate Services 等です。
- 試験目的の場合や、制御された、または小さなユーザーグループ内でのやりとり：自己署名した証明書からのデジタル ID を作成します。Acrobat で自己署名の証明書を作成するには以下のようにします：
  Acrobat XI：**「編集」→「環境設定」→「一般」→「署名」→「*ID* と信頼済み証明書」→「詳細 ...」→「*ID* を追加」→「今すぐデジタル *ID* を新規作成」**
  Actobat X：**「ツール」→「電子署名」→「その他の電子署名」→「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」**。
  その次の操作で、PKCS#12 ディスクファイルか Windows 証明書ストアをターゲットとして指定できます。PLOP DS は両方の方式に対応しています。

6.1.2 Acrobat と PDF におけるさまざまな署名

PDF では、以下に述べるさまざまな種類の電子署名を使えます。署名は PDF 内にフォームフィールドとして実装されています。PDF 署名は、常に文書全体に（個々のページにではなく）紐付いており、2 つの種類があります：

- 不可視署名は、ページ上の領域を全く占めません。これを Acrobat で表示するには**「署名」**パネルを表示させます（Acrobat X/XI：**「表示」→「表示切り替え」→「ナビゲーションパネル」→「署名 ...」**）。

- 可視署名は、文書内のページ上のどこかに位置付けられている長方形のフォームフィールドを用います。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。

どちらの種類の署名に対しても、場所、署名理由、連絡先情報等、さらなる特性を指定することが可能です。1

承認署名　PDF で最も広く用いられる署名の種類は、承認署名というものです。1 個の PDF 文書は、1 個または複数の承認署名を内容とすることができます。1 個の承認署名が、種別**「署名」**のフォームフィールド 1 個の中に配置されます。このフィールドは、不可視とすることも可視とすることもできます。承認署名は、その文書がそのデジタル ID の保有者によって署名されていることを保証するとともに、文書変更が必ず検知される働きを持ちます。その文書に何か変更が加えられると、その署名は必ず無効になります。承認署名は、その署名を作成する個人か主体に紐付いています。他のなんぴとたりとも、その必要な各種証明書類へのアクセスを有しませんので、その署名者は、署名時点におけるその文書の状態を否定できません（否認防止）。

承認署名を持った文書を開く際には、Acrobat は通常、上端付近に青いバーを表示します（ただしその文書が PDF/A に準拠している場合には、この署名バーよりも PDF/A バーのほうが優先されて表示されます）。その署名が有効の場合には、この青いバーは緑のチェックマークを含みます。この署名は Acrobat の**「署名」**パネル内にも表示されます。

承認署名は、長期検証のために証明書失効情報とタイムスタンプを内容として持つこともできます。いずれのアイテムも、信頼されたサーバからネットワークを通じて取得されます。

承認署名は PLOP DS におけるデフォルトの署名種別です。これは少なくとも PDF 1.6 出力を必須とします。必要な場合には、PLOP DS は PDF バージョンを然るべく押し上げます。

承認署名は pCOS では *signaturefields[...]/sigtype=approval* として報告されます。

証明用署名　1 つの文書の中の最初の署名は、証明用署名とすることもできます。この種類を作成者署名ともいいます。なぜならこれは、その作成者がその文書を作成した通りのその文書の状態を証明するからです。この文書作成者は、その文書に対して、その署名を破ることなくある種の変更が行われることを許容することもできます。ですので証明用署名を、改変検知・防止（Modification Detection and Prevention ＝ MDP）署名ともいいます。許容する変更としては以下の種類を指定できます（表 6.6 参照）：

- 変更を一切許可しない：プレスリリースや政府刊行物といった典型的なリードオンリー文書で有用です。この場合には、作成者または文書レベルタイムスタンプ署名を追加するだけでも、その証明書署名は無効になります。
- フォームフィールドの入力と電子署名の追加（Acrobat のメニュー項目からでなく、署名フィールドをクリックすることによる）を許可：この証明書署名は、購入注文フォーム等において、フォームユーザーが必ず真正な文書で操作を行えるようにします。ユーザーが、編集可能なフォームフィールドへの入力を行なっても、または、作成者署名を行なっても、その証明書署名は無効になりません。ページをページテンプレートから産み出すことによって追加する（手動でページを追加するのではなく）ことも許容されますが、この技法が使われることはまれです。
- フォームフィールドの入力と電子署名・注釈の追加を許可：これはたとえば、公証人が、署名された文書に、その信証の性質に関する詳細を内容とする註を付けたい場合に利用できます。

証明用署名を持った文書を開いた時、Acrobat は、上端付近の青い署名バーの中に青いリボンを表示します。その署名は Acrobat の**「署名」**パネル内にも表示されます（それが有効ならば青いリボンとともに）。

証明用署名を PLOP DS で作成するには *certification* 署名オプションを用います（6.3.5 節「証明用署名」（85 ページ）参照）。証明用署名には少なくとも PDF 1.6 出力が必須です。必要ならば PLOP DS は PDF バージョンを然るべく押し上げます。

文書レベルタイムスタンプ署名　タイムスタンプ署名は、承認署名または証明用署名の中に埋め込まれたタイムスタンプと混同しないよう注意が必要です。1 個の文書が任意の数のタイムスタンプ署名を内容とすることができます。タイムスタンプ署名は、その文書がある特定の時点において存在していたことを確かなものとします。このタイムスタンプは、信頼されたサーバからネットワークを通じて取得され、その文書に署名した個人や主体へは関わりを持ちません。タイムスタンプ署名は長期検証のために重要な役割を果たします。なぜならタイムスタンプ署名を使って既存の署名をリフレッシュさせることができるからです。タイムスタンプ署名はフォームフィールド内に配置されますが、常に不可視です。

タイムスタンプ署名を持った文書を開いた時、Acrobat は、上端付近の青い署名バーの中に緑のチェックマークを表示します。その署名は Acrobat の**「署名」**パネル内にも表示されます（それが有効ならば時計とスタンプのアイコンとともに）。

タイムスタンプ署名を PLOP DS で作成するには *doctimestamp* 署名オプションを用います（6.5.3 節「文書レベルタイムスタンプ署名」（96 ページ）参照）。タイムスタンプ署名には少なくとも PDF 1.7ext8 出力が必須です。必要ならば PLOP DS は PDF バージョンを然るべく押し上げます。

使用権限署名　1 個の文書が 2 個までの使用権限署名を内容とすることができます。これを利用すると、いくつかの編集機能を Adobe Reader で使える、いわゆる Reader 有効化された PDF 文書を作成できます。使用権限署名は、署名フォームフィールドに結び付いてはおらず、Acrobat の**「署名」**パネルに表示されません。

使用権限署名は、PLOP DS で作成することはできませんが、pCOS 擬似オブジェクト *usagerights* を用いてクエリすることはできます。

Acrobat における信頼済みルート証明書　Adobe Reader と Acrobat は、以下の取得元からの信頼済みルート証明書を受け付けます：

- Adobe 認定信頼リスト（Adobe Approved Trust List ＝ AATL）[1]からのルート証明書。この AATL は、世界中の多くの国からの商用・機関・政府認証局（CA）を内容としています。照応する CA 証明書群が Adobe Reader・Acrobat 内に含まれています。この AATL の中のルート証明書のうちのいずれかにチェーンしている証明書はすべて、信頼できると見なされます。この AATL リストは、Acrobat・Adobe Reader に内蔵されており、時の経つにつれて拡張される場合があります。更新されたリストを、Acrobat で自動的に、または手動でダウンロードするには、**「編集」→「環境設定」→「信頼性管理マネージャ」→「*Adobe* のサーバーから信頼済みのルート証明書を読み込む」**を選択します。執筆時点で、50 局近い CA がこの AATL プログラムに参加しています。
- AATL の前にあった、これよりも古い、2005 年に導入された Adobe の Certified Document Services（CDS）[2] プログラムからのルート証明書。AATL 証明書は、Acrobat 内で直接、信頼済みルート CA として扱われますが、CDS 証明書は必ず、Adobe Root 証明書へ

1. さらに詳しい情報と参加 CA の一覧については helpx.adobe.com/acrobat/kb/approved-trust-list2.html を参照。
2. helpx.adobe.com/acrobat/kb/certified-document-services.html を参照。

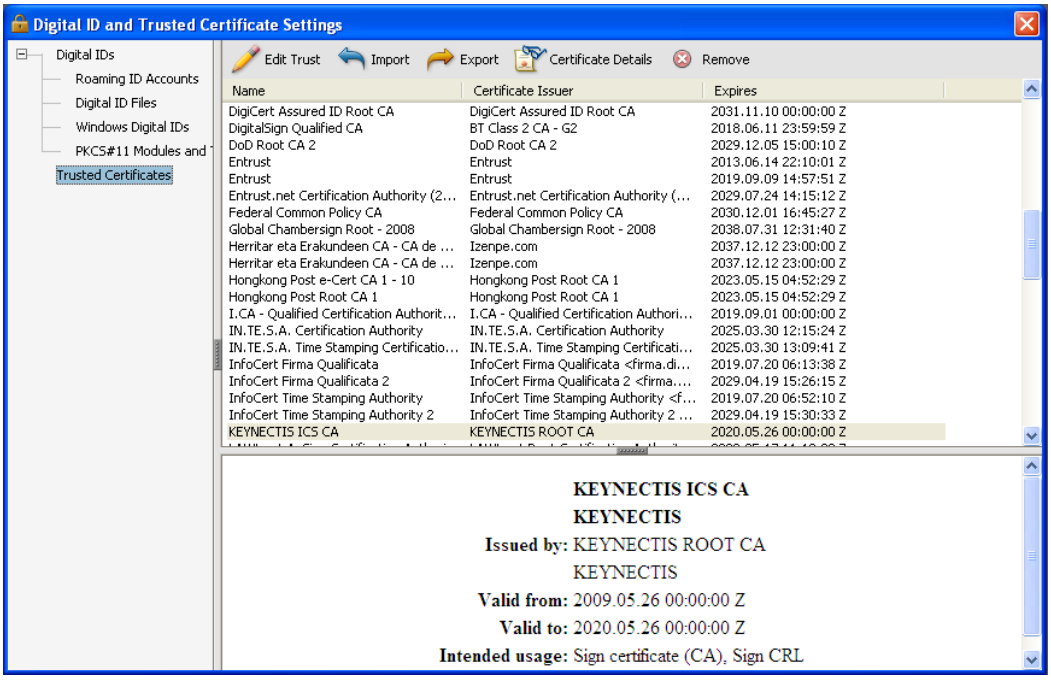


図 6.1
Acrobat の信頼済み証明書のリスト

チェーンしています。以下の CA がこの CDS プログラムに入っています：Entrust・GlobalSign・Keynectis・Post.Trust・Symantec。

- Adobe Reader と Acrobat ではバージョン 11.0.6 から、ETSI TS 119 612[1] に従った European Union Trust List（EUTL）からの信頼済みルート証明書のダウンロードに対応しています。これは**「編集」→「環境設定」→「信頼管理マネージャー」→「*Automatic European Union Approved Trusted Certificates Updates*」**を通じて制御できます。この機能はまだ動作しませんが、近い将来、国の信頼リストの追加のために使用できるようになることを期待しています。
- Acrobat または Adobe Reader へユーザーによって、**「編集」→「環境設定」→「署名」→「*ID* と信頼済み証明書」→「詳細 ...」→「信頼済み証明書」**を通じて手動で取り込まれたルート証明書。この証明書は、**「連絡先の信頼を設定」→「信頼」**タブを通じて**「この証明書を信頼済みのルートとして使用」**を有効にすることによって、信頼済みとして構成される必要があります。この方法は、任意のルート証明書に対して使えますが、ユーザー側による手作業を必要とするため、ワークフローによっては望ましくありません。
- Acrobat は、Windows 証明書ストア内の証明書を信頼済みとして扱うこともできます。これは、Acrobat XI で**「環境設定」→「署名」→「検証」→「詳細 ...」→「*Windows* 統合」**を通じて制御できます。

Acrobat の信頼済みルート証明書のリストを表示するには、**「編集」→「環境設定」→「署名」→「*ID* と信頼済み証明書」→「詳細 ...」→「信頼済み証明書」**を用います（図 6.1 参照）。

1. www.etsi.org/deliver/etsi_ts/119600_119699/119612/01.01.01_60/ts_119612v010101p.pdf を参照

6.2 PLOP DS の各種暗号化エンジン

6.2.1 概要

PLOP DS では、文書に電子的に署名を行うために必要な公開鍵とハッシュ化アルゴリズムを実装した複数の暗号化エンジンを使用できます。署名を PLOP DS ライブラリで作成するには、*PLOP_prepare_signature()*・*PLOP_create_document()* API 関数を用いるか、PLOP DS コマンドラインツールのオプション *--signopt*（短縮記法：*-S*）を用います。

PLOP DS を用いて電子署名を行うにはデジタル ID が必要です。デジタル ID ファイルかトークンを使用する場合には、照応するパスワードが必要です。Windows 証明書ストア内の個人的な（アカウント毎の）デジタル ID を使用する場合には、その ID は通常、その人の Windows ログインによって保護されています。

電子署名を作成するための各種暗号化エンジン PLOP DS ではさまざまな暗号化エンジンを使用できます。暗号化エンジンとは、電子署名を生成するために必要なさまざまな暗号化機能を実装した一片のソフトウェアまたはハードウェアです。どの暗号化エンジンを選択するかによって、デジタル ID の形式・保管場所、および他のソフトウェアとオペレーティングシステムとの統合の様式が異なります。PLOP DS では以下の暗号化エンジンを使用できます：

- *builtin* エンジンは、すべてのプラットフォームで利用できます。これは、必要な暗号化機能群を PLOP DS カーネル内に直接実装しており、外部依存を一切必要としません。このエンジンはデフォルトで有効ですが、署名オプション *engine=builtin* を用いて明示的に選択することもできます。
- *pkcs#11* エンジンは、暗号トークンへの統一的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参照します。ここでトークンとは、スマートカードや USB スティック等の暗号デバイスを意味します。トークンは、ソフトウェア証明書よりも高いセキュリティを提供し、多くの場合、PIN を用いて保護されています。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。この *PKCS#11* エンジンを選択するには *engine=pkcs#11* 署名オプションを用います。
- *mscapi* エンジンは、Microsoft Cryptographic API（Windows 上でのみ利用可能）を参照します。この API はこのオペレーティングシステムに含まれています。これにより PLOP DS は、Windows によって提供されている暗号化インフラストラクチャと、あるいは、CAPI ドライバを通じて結合されているサードパーティソフトウェアまたはハードウェアと相互作用することが可能になります。この *mscapi* エンジンを選択するには *engine=mscapi* 署名オプションを用います。

使用できる各種デジタル ID PLOP DS は、PDF 文書に署名を行うためにはデジタル ID を必要とします。デジタル ID は、その署名者のデジタル証明書と、照応する秘密鍵を内容として持ち、通常、パスワード等の手段によって保護されています。PLOP DS では以下の種類のデジタル ID を使用できます：

- 各種プラットフォームで *engine=builtin* を使用：PKCS#12 形式のデジタル ID ファイル（通常、*.p12*・*.pfx*・*.cer* のいずれか）
- PLCS#11 対応のプラットフォームで *engine=pkcs#11* を使用：コンピュータに接続されたスマートカード等の暗号トークン（デバイス）に保管されているデジタル ID。
- Windows で *engine=mscapi* を使用：Windows 証明書ストア内のデジタル ID。

6.2.2 内蔵エンジン

内蔵エンジンはデフォルトエンジンです。これはファイルベースの証明書とともに働き、完全な機能と制御を提供します。

秘密鍵を解除　デジタル ID（より正確には：デジタル ID 内に含まれている秘密鍵）は通常、パスワードかパスフレーズか PIN を用いて保護されています。なぜなら電子署名を作成するための機密である秘密鍵を内容として持っているからです。デジタル ID を PLOP DS で使用するために解錠するには、正しい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ると、PLOP DS は例外を発生させます。デジタル ID の具体的な解錠の方法は、どの暗号化エンジンを選択するかによって異なります：

照応するパスワードを、*password* 署名オプションを用いて与える必要があります。PLOP DS コマンドラインツールを使用している場合には、パスワードについては、間接的に外部ファイル内で、*passwordfile* サブオプションを用いて与えることを強く推奨します。もしパスワードをパスワードファイル内でなく直接与えてしまうと、他のユーザーが読めるおそれがあります。なぜならコマンドラインはマルチユーザーシステム上では他のユーザーから見える場合があるからです。

オプションリスト例　以下の例に、PLOP DS コマンドラインツールとライブラリを用いて PDF 文書に電子的に署名を行う方法を示します。自分自身のプログラムの中から署名を作成するために、*--signopt* に与えるオプションリストを、PLOP DS の API 関数 *PLOP_prepare_signature()* に与えることもできます。対応しているすべての言語バインディングのための完全なプログラミング例が PLOP DS パッケージ内にあります。これらの例では、パスワード *demo* のデジタル ID ファイル *demorsa2048.p12* を使用します。このファイルはディストリビューションパッケージに含まれています。

ファイル *demorsa2048.p12* からのデジタル ID を使用して、PDF 文書に対して不可視の署名を作成します。このデジタル ID に対するパスワードはファイル *pw.txt* の内容となっています：

```
plop --signopt "digitalid={filename=demorsa2048.p12} passwordfile=pw.txt" ↵
       --outfile signed.pdf input.pdf
```

6.2.3 スマートカード等の暗号トークンのための PKCS#11 エンジン

PLOP DS 内の PLCS#11 エンジンを利用すると、スマートカードや USB スティック等の暗号トークン上の証明書を使用できます。そのようなトークンを使用して署名を作成するには、トークン個別のプロトコルを実装した DLL または共有ライブラリが必要です。この PKCS#11 DLL/SO が、そのトークンプロバイダーによって、照応するソフトウェアパッケージに含まれて提供されている必要があります。それがシステムにインストールされている必要があり、かつ、PLOP DS から利用可能になっている必要があります。Windows ではすなわち、その DLL が、Windows システムディレクトリか、PATH 環境変数内に含まれているディレクトリか、アプリケーションのカレントディレクトリへ複製されている必要があります。PKCS#11 DLL/SO が他の DLL 群に依存している場合もあることに留意してください。この場合には、そのトークンベンダーによって提供されているすべての必須 DLL が PLOP DS から利用可能になっている必要があります。

秘密鍵を選択　1 つの署名トークンが、複数のデジタル ID を内容として持っている場合もあります。たとえば 1 つは電子メールを暗号化するための物、もう 1 つは文書に電子的に署名するための物といったようにです。そのトークン上に複数の署名証明書がある場合には、*digitalid* オプションのサブオプション *issuer*・*label*・*serial*・*subject* のうちのいずれ

図 6.2
キーボード付きスマートカードリーダ

か 1 つを与えることによって、こうした判定基準のうちの 1 つによって適切な証明書を選択する必要があります。1 つの鍵に対して、そのトークン用の管理ソフトウェアを用いて、1 つのラベルを割り当てることも可能です。発行者・シリアル番号・サブジェクトは、証明書の固有のフィールドです。

トークン上の秘密鍵を解錠　暗号トークンが、パスワードまたは PIN がソフトウェアによって与えられることを許容している場合には、*engine=builtin* の場合と同様に *password* 署名オプションを与える必要があります。そのトークンが、直接的な PIN またはパスワード入力を要求する場合（キーボード付きのスマートカードリーダ等）には、この *password* オプションを省略する（または空文字列を与える）ことができ、そのトークンのキーボードに手動でその PIN を入力する必要があります。パスワード／ PIN 処理の詳細は暗号トークンによって異なる可能性があります。

PKCS#11 セッションとマスチスレッディング　大量署名のパフォーマンスを向上させるために、PLOP DS は、PKCS#11 DLL/SO に対するロード／アンロード操作の数を最小化させ、各 PKCS#11 セッションの持続期間を最大化しています。このためには、アプリケーションは以下の条件に従う必要があります：

- いかなる時点においても、読み込める PKCS#11 DLL/SO は、そのライブラリを使用した最後の PLOP オブジェクトに対して *PLOP_delete()* が呼び出されるまで、ただ 1 つです。最後の PLOP オブジェクトを削除した後には、別の PKCS#11 DLL/SO を *PLOP_prepare_signature()* で指定できます。言い換えれば、任意の数の PKCS#11 スロットを、すべてのトークンスロットが同一の DLL/SO によって提供されている（これは通常、同一種別のトークンを意味します）限り、マルチスレッドな方式で指定することが可能です。
- ある特定のスレッドの中の *PLOP_prepare_signature()* は、別のスレッドからすでにアクセスされている PLCS#11 スロットへアクセスしてはいけません。

ある特定のスロットに対する *PLOP_prepare_signature()* への最初の呼び出しの中で新規セッションが作成され、それと同じスレッドの中で *PLOP_prepare_signature()* が再び呼び出されるまで維持されます。ですのでアプリケーションは、可能な限り多数の出力文書に対して、*PLOP_prepare_signature()* を 1 回のみ呼び出すべきです。たとえば、同一の PKCS#11 スロットが指定されている限り、かつそのトークンの制約（署名の最大数や、連続署名に対する最長時間等）が満たされている限りにおいては、*PLOP_prepare_signature()* へのさらなる呼び出しは必要となりません。

大量署名を効率的に行うための完全コードサンプルが、すべての PLOP DS パッケージに含まれている *multisign* サンプルの中にあります。

PKCS#11 のオプションリスト例　以下の例においては、ベンダー固有の PKCS#11 DLL を *cryptoki.dll* としています。実際の DLL 名はこれとは異なる可能性があります。

PKCS#11 を通じて指定されたトークンからのデジタル ID を用いて、PDF 文書に不可視署名を作成します。このトークンに対する PIN はファイル *pw.txt* の内容となっています：

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

PKCS#11 を通じて指定されたトークンからのデジタル ID を用いて、PDF 文書に不可視署名を作成します。このコマンドでは、PIN を与えていませんので、このトークンに対する PIN を、トークンに付いているキーボードへ打ち込む必要があります：

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

6.2.4 Windows 上の MSCAPI エンジン

MSCAPI エンジンを利用すると、Windows オペレーティングシステムに内蔵されている署名機能を活用できます。最も重要な点は、Windows 証明書ストア内のデジタル ID 群へアクセスできる点です。他方で、この MSCAPI エンジンには、他の暗号化エンジン群にはないいくつかの制約があります。たとえば、MSCAPI は ECDSA に対応していません。

注記 OCSP・CRL 埋め込みと、タイムスタンピングは、`engine=mscapi` では使えません。ですので、MSCAPI エンジンでは LTV 対応署名を作成できません。

秘密鍵を解錠　その人の証明書の設定によっては、Windows 証明書ストア内のデジタル ID は、その人の Windows ログインによって保護されており、さらなるパスワードが必要ない場合もあります。その証明書を Windows 証明書ストア内へ取り込む際に高セキュリティを有効にした場合には、その証明書を使用して署名を行うたびに必ずそのパスワードを求められます。これは当然バッチアプリケーションには不向きです。

MSCAPI のためのオプションリスト例　以下の例では、署名を行うためのデジタル ID は Windows 証明書ストア内で得られると前提しています。PLOP DS デモ証明書でこれを実現するには、ファイル *demorsa2048.p12* 内のデジタル ID をダブルクリックして Windows 証明書ストア内へインストールする必要があります。

Windows 証明書ストアからの（デフォルトストア *My* からの、デフォルトストア位置 *current_user* からの）証明書を使用して、PDF 文書に不可視署名を作成します。これは、パスワードを与える必要がないよう、そのデジタル ID がその人の 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=mscapi digitalid={store=My subject={PDFlib Demo PLOP User 2048}}" ↵
         --outfile signed.pdf input.pdf
```

ファイル *demorsa2048.p12* 内の証明書を使用して、PDF 文書に不可視署名を作成します：

```
plop --signopt "engine=mscapi digitalid={filename=demorsa2048.p12} passwordfile=pw.txt" ↵
        --outfile signed.pdf input.pdf
```

PDF 暗号化のためのマスターパスワード ***SECRET*** と、デジタル ID にアクセスするためのパスワード ***demo*** を用いて、不可視署名を作成し、文書を暗号化します：

```
plop --master SECRET --signopt "digitalid={filename=demorsa2048.p12} password={demo}" ↩
        --outfile signed.pdf input.pdf
```

Windows 証明書ストアを管理　Windows オペレーティングシステムは、いくつかの証明書ストア内に編成された証明書群を保持することが可能です。PKCS#12 形式の証明書を新規にインストールするには、単にその証明書ファイルをダブルクリックして、**「証明書のインポート ウィザード」**に従います。PLOP DS パッケージ内のデモ証明書群で、パスワード ***demo*** を用いてこれを試すことができます。

以下のように、Microsoft 管理コンソール（MMC）を用いて証明書群を表示・編成することもできます：

- **「スタート」**をクリックし、プログラム名のための枠に**「*mmc*」**と打ち込むことによってこのプログラムを起動します。
- **「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」**を選択して**「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」**と**「完了」**を選択します。あるいは、**「サービス アカウント」**か**「コンピューター アカウント」**が自分の証明書のストア位置であるならば、それを用います。
- **「*OK*」**をクリックします。

これで、インストールされた証明書をブラウズできるようになりました。自分の証明書は**「個人」**カテゴリ内にあり、これを PLOP DS で以下のオプションリストを用いて（***--signopt*** コマンドラインオプションに、または、***PLOP_prepare_ signature()*** の署名オプションとして与えて）指定できます：

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

証明書の詳細を表示するには、MMC 内で証明書をダブルクリックします。証明書を PFX 形式で書き出すには、リスト内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。

管理コンソールを使用して証明書を取り込むこともできます：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。

6.2.5 署名と各種ハッシュアルゴリズム

電子署名は、暗号化アルゴリズムと、ハッシュアルゴリズムと、両者に対するパラメータ群によって特徴付けられます。

署名を生成する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、そのデジタル ID のための公開鍵／秘密鍵ペアを作成する際に指定されます。PLOP DS は以下の署名アルゴリズムに対応しています：

- 範囲 1024 ～ 4096（2048 ビット以上が推奨されます）のキー長を持った RSA。RSA は、インターネット等多くの応用分野で広く利用されています。
- 範囲 1024 ～ 4096（2048 ビット以上が推奨されます）のキー長を持った DSA。DSA は広く利用されていません。DSA は SHA-1 の使用を必須としており、SHA-1 はもや安全とは見なされていませんので、DSA の使用については安全性の懸念があります。

- ECDSA（楕円曲線電子署名アルゴリズム）は、RSA の近年の後継者です。ECDSA の強度は曲線によって決まります。曲線は、パラメータ群によって特徴付けられますが、名前で特徴付けられることのほうが多いです。RFC 5480 では、NIST によって推奨されている 15 種類の名前付き曲線を定義していますが、それらの曲線のうち Acrobat XI が対応しているのは 3 種類のみです。これらの Acrobat 準拠の曲線の名前は ***P-256/P-384/P-521*** です。RFC 5639 ではこれに加えて、Brainpool 曲線群という曲線の集合を定義しています。Brainpool 曲線を使用した署名を、Acrobat を用いて検証することはできず、専用の検証ソフトウェアを必要とします。Suite B 暗号法に対しては、曲線 ***P-256*** または ***P-384*** を用いた ECDSA 署名が必須となっています。

ハッシュアルゴリズムは、署名されたデータに対するメッセージダイジェストを生成するために使用されます。広く用いられているハッシュアルゴリズムは、SHA-1（もはや安全とは見なされていません）と、それよりも強い、SHA-2 ファミリ内の SHA-256・SHA-384・SHA-512 を含むアルゴリズム群です。Suite B 暗号法に対しては、ハッシュ関数 SHA-256 または SHA-384 が必須となっています。署名に対して用いられているハッシュアルゴリズムを Acrobat XI で表示するには以下のようにします：

- **「署名」**パネルを開きます。
- 署名を選択し、**「署名」**メニューで**「署名のプロパティを表示 ...」**を選択します。
- **「署名者の証明書を表示」**をクリックします。
- すると、**「証明書ビューア」**というタイトルのダイアログが表示され、その中の**「詳細」**の中にそのハッシュアルゴリズムが表示されます。

表 6.1 に、各種署名アルゴリズムと、照応するハッシュ関数を挙げます。この表には、署名を検証するために最低限必要な Acrobat バージョンも挙げてあります。Acrobat を用いて PDF 署名を検証しようとする場合には、署名を行うために用いられたデジタル ID の署名特性に Acrobat バージョンを必ず合致させる必要があります。表 6.1 には、各署名アルゴリズムから生成される最低限の PDF 出力バージョンもあわせて挙げてあります。入力文書がそれよりも低い PDF バージョン番号を用いている場合には、PLOP DS は、その PDF バージョンを、この表に挙げてある値まで押し上げます。

表 6.1 署名アルゴリズム・ハッシュアルゴリズム・PDF 出力バージョン・必須 Acrobat バージョン

署名アルゴリズム	engine=builtin と engine=mscapi の場合のハッシュアルゴリズム[1]	註	PDF 出力バージョンと検証のために必要な最小 Acrobat バージョン[2]
承認・証明用署名			
4096 ビット以下の RSA	SHA-256	engine=mscapi：Enhanced Cryptographic Provider が必要です	PDF 1.6 ／ Acrobat 7 sigtype=cades の場合： PDF 1.7ext8 ／ Acrobat X
4096 ビット以下の DSA	SHA-1（DSA で必須）	engine=mscapi：1024 ビット以下のみ engine=pkcs#11：非対応	PDF 1.6 ／ Acrobat 7 sigtype=cades の場合： PDF 1.7ext8 ／ Acrobat X
NIST 曲線（RFC 5480）P-256/P-384/P-521[3] を用いた ECDSA	その曲線の強度によって、SHA-256・SHA-384・SHA-512 のいずれか	engine=mscapi：非対応	PDF 1.7ext8 ／ Acrobat XI
P-256/P-384/P-521 以外の NIST 曲線を用いた ECDSA	その曲線の強度によって、SHA-256・SHA-384・SHA-512 のいずれか	engine=mscapi：非対応 conformance=extended が必須	PDF 1.7ext8 ／ Windows 版 Acrobat XI で一部の曲線のみを検証可能

表 6.1 署名アルゴリズム・ハッシュアルゴリズム・PDF 出力バージョン・必須 Acrobat バージョン

署名アルゴリズム	engine=builtin と engine=mscapi の場合のハッシュアルゴリズム[1]	註	PDF 出力バージョンと検証のために必要な最小 Acrobat バージョン[2]
14 種の Brainpool 曲線を用いた ECDSA（RFC 5639）	その曲線の強度によって、SHA-256・SHA-384・SHA-512 のいずれか	engine=mscapi：非対応 conformance=extended が必須	PDF 1.7ext8 ／ Acrobat XI で検証できません
文書レベルタイムスタンプ			
TSA によって選択されます	デフォルトでは SHA-256 ですが、doctimestamp サブオプション hash を用いて変更することも可能	builtin エンジンの場合にはタイムスタンプは常に作成されます	PAdES パート 4 拡張を伴う PDF 1.7ext8 ／ Acrobat X
OCSP リクエスト・レスポンス（証明書識別）			
OCSP レスポンダによって選択されます	デフォルトでは SHA-1 ですが、ocsp サブオプション hash を用いて変更することも可能	選択されたハッシュアルゴリズムに OCSP レスポンダが対応している必要があります	Acrobat XI 以下は OCSP に対して SHA-1 にのみ対応しています

1. engine=pkcs#11 の場合のハッシュ関数はトークンによって与えられます。ある特定のトークンモデルによって使用されているハッシュ関数を知るには、そのトークンベンダーの説明書を参照してください。
2. PDF/A・PDF/X モードでは、入力文書の PDF バージョンが変更されないまま保たれます。
3. これらの曲線は、secp256r1（または prime256v1）／ secp384r1 ／ secp521r1 という名前でも知られています。

6.3 PDF の署名の各種設定内容

6.3.1 署名をグラフィックかロゴで視覚化

認証・証明用署名は文書内で以下の方式で表現できます：

- 不可視署名は、ページ上に何ら表現を持ちません。Acrobat の**「署名」**パネル内にのみ表示されます。
- 可視署名は、ページ上の特定位置にその署名を表示するための任意のテキストかグラフィックを内容とすることができます。既存の PDF 文書からのページを用いてその署名の視覚表現を作成することができます。可視署名は**「署名」**パネルにも表示されます。このページを採られた文書を、視覚化文書といいます。

文書レベルタイムスタンプ署名は常に不可視署名として作成されます。

署名視覚化文書　署名視覚化のために用いられる PDF ページは、スキャンされた手書きの署名や、公的印鑑や企業ロゴや、署名用証明書の保有者の写真等、その署名された文書の受け手にとって有用となりうる任意の視覚表現を内容とすることができます。

視覚化文書が、署名された入力文書よりも高い PDF バージョンを用いている場合には、生成される出力の PDF バージョンは然るべく調整されます。***PDF 1.7ext3***（Acrobat 9）と***PDF 1.7ext8***（Acrobat X/XI）の文書は、その視覚化ページとしての用途に関する限り、PDF 1.7 と互換です。

注記 PDF/A のための署名視覚化は、その視覚化文書にいくつか特定の制限を課します（「PDF/A 準拠」（81 ページ）を参照）。PDF/UA・PDF/X・PDF/VT モードでは電子署名の視覚化に対応していません。

視覚化文書を、*PLOP_open_document()* を用いて開く必要があります。その文書ハンドルを、*field* オプションの *visdoc* サブオプションに与える必要があります：

```
field={visdoc=< ハンドル > rect={100 100 300 150}}
```

署名フィールドの位置と寸法　*field*署名オプションは、ページ上でのその署名の表現を制御します。署名された文書の可視ページ上での署名視覚化ページの位置と寸法を、この *field* オプションの *rect* サブオプションを用いて指定できます。この寸法を、明示的に指定することもできますし、1 つの隅と、他の寸法のうちの 1 つか 2 つを指定することによって暗黙的に指定することもできます。抜けている値は、キーワード *adapt* を用いて指定することによって、変倍がかからないよう自動的に算出されます。この *adapt* キーワードを用いれば、視覚化ページを署名長方形の任意の隅に寄せることができます。できあがる長方形がページをはみ出してはいけません。以下の例でさまざまな組み合わせを演示します。

- 最もシンプルはアプローチは、視覚化ページを、求める寸法で作成しておくことです。この場合には、単にフィールドの左下隅の座標を与えれば、PLOP DS はページの原寸を用いて署名視覚化を行います：

  ```
  rect={100 100 adapt adapt}
  ```

- 左下隅に寄せて、幅を保ち、高さを調節することによって変倍を防ぎます：

  ```
  rect={100 100 300 adapt}
  ```

- 左下隅に寄せて、幅を調節して高さを保つことによって変倍を防ぎます：

```
rect={100 100 adapt 200}
```

- ページを長方形に強制的に合わせます。すなわち、長方形の幅と高さを両方とも保ちます。ページと長方形の縦横比が異なる場合には、その視覚化ページには変倍がかかります：

```
rect={100 100 300 200}
```

然るべき署名フィールド長方形を、視覚化ページの寸法に応じて動的に算出するには、pCOS インタフェースを用いてそのページ寸法をクエリすることができます（pCOS ページ番号は 0 から始まることに留意してください）：

```
width  = plop.pcos_get_number(visdoc, "pages[" + (vispage-1) + "]/width");
height = plop.pcos_get_number(visdoc, "pages[" + (vispage-1) + "]/height");
```

既存のフォームフィールド内に署名する　入力文書がすでに署名フィールドを内容として持っている場合には、そのフィールドをその署名と視覚化のために使用することもできます。これを実現するには、その既存のフィールドの名前がわかっている場合には、それを与えます：

```
field={name=MyExistingFieldName visdoc=< ハンドル >}
```

そのフィールド名がわからない場合には、以下のように、PLOP に対して、既存の署名フィールドを使用するよう命令することもできます：

```
field={fillexisting visdoc=< ハンドル >}
```

既存のフィールド内に署名する場合であっても、*rect* フィールドオプションを用いてその位置と寸法を変更することは可能です。既存のフィールド内に署名を作成した場合、かつ、そのフィールドがページ上の可視の長方形を用いている場合には、*visdoc* オプションを与える必要があります（あるいは、フィールドオプション *rect={0 0 0 0}* を用いてそのフィールドを不可視にします）。

署名フィールド内における視覚化ページの位置を指定　視覚化ページは、署名フィールド内に配置され、それがその長方形内に収まり、かつその縦横比が保たれるように拡縮されます。これはとりわけ、署名を既存のフォームフィールド内へ配置したい場合、かつ、そのフィールドと視覚化ページの縦横比が一致しない場合に有用です。

field オプションの *position* サブオプションを用いると、その署名フィールド内における視覚化ページの位置を指定することができます。

デフォルトでは、視覚化ページはフィールド内で縦横ともに中央に配置されます。これを変更して、たとえば視覚化ページを署名フィールドの左下隅に配置させることもできます：

```
field={name=MyExistingFieldName visdoc=< ハンドル > position={left bottom} }
```

pCOS　署名の可視性は pCOS において *signaturefields[...]/visible=true* として報告されます。署名フィールドがすでに署名を内容として持っているかどうかの情報をクエリするには *signaturefields[...]/sigtype != none* を用います。

6.3.2 PDF/A・PDF/UA・PDF/X・PDF/VT 準拠

このマニュアルで別途特記しない限り、すべての PLOP 操作は、PDF/A・PDF/UA・PDF/X・PDF/VT の諸規定に準拠していますので、PLOP 操作によって規格準拠が維持されま

す。ただしこの規則にはいくつか例外として、ある特定の規格によって PLOP 操作が禁じられている場合もあります。たとえば PDF/A で暗号化は禁じられています。そのような場合には、自分の優先順位を考慮する必要があります：

- 規格準拠を維持する必要がある場合には、その操作は PLOP によって拒絶されます。これがデフォルト動作です。
- その操作（暗号化等）が規格準拠よりも大切である場合には、*sacrifice* オプションを用いて規格識別子を除去することが可能です。

関連する規格ごとの注意点を以下に挙げます。

PDF/A 準拠　PDF/A 規格では、CMS・CAdES ベースの署名を許容しており、タイムスタンプと、失効情報と、証明書チェーンのうち可能な限り多くを含めることを推奨しています。

PDF/A モードでは、すなわち、入力文書が PDF/A に準拠しており、かつ、*pdfa* に対して *sacrifice* オプションが設定されていない場合には、取り込まれる視覚化文書は、その PDF/A 諸特性について互換である必要があります：

- 視覚化文書の PDF/A レベルが互換である必要があります（表 6.2）。
- 視覚化文書の出力インテントが互換である必要があります（表 6.3）。

備考：出力インテントを持たない PDF/A-1a 視覚化文書（表 6.2 と表 6.3 で赤で囲ってあります）は、すべての PDF/A パート・互換レベル・出力インテント種別と互換です。PLOP DS ディストリビューションは、これらの特性を持ったサンプル視覚化ファイル *signing_man_pdfa1a.pdf* を含んでいます。これを、すべての種類の PDF/A とともに試すための視覚化文書として利用できます。出力インテントを持たない PDF/A-1b 視覚化文書は、すべての PDF/A パートの *b* 準拠レベル群と互換です。

PDF/A 準拠にこだわらない場合には、その規格準拠エントリを除去することもできます：

```
sacrifice={pdfa}
```

表 6.2　さまざまな PDF/A 入力レベルに対して互換な視覚化文書の PDF/A レベル

	視覚化文書の *PDF/A* レベル				
入力文書の PDF/A レベル	**PDF/A-1a:2005**	**PDF/A-1b:2005**	**PDF/A-2a・PDF/A-3a**	**PDF/A-2b・PDF/A-3b**	**PDF/A-2u・PDF/A-3u**
PDF/A-1a:2005	許容	—	—	—	—
PDF/A-1b:2005	許容	許容	—	—	—
PDF/A-2a・PDF/A-3a	許容	—	許容	—	—
PDF/A-2b・PDF/A-3b	許容	許容	許容	許容	許容
PDF/A-2u・PDF/A-3u	許容	—	許容	—	許容

表 6.3　視覚化文書の PDF/A 出力インテント互換性（すべての PDF/A 互換レベルに対して）

	視覚化文書の出力インテント種別			
入力文書の出力インテント種別	**なし**	**グレースケール**	**RGB**	**CMYK**
なし	許容	—	—	—
グレースケール *ICC* プロファイル	許容	許容[1]	—	—
RGB ICC プロファイル	許容	—	許容[1]	—
CMYK ICC プロファイル	許容	—	—	許容[1]

1. 視覚化文書の出力インテントと入力文書の出力インテントとが同一である必要があります。

PDF/UA 準拠　PDF/UA モードでは署名視覚化には対応していません。その上 PDF/UA では、不可視署名フィールドですらも「その構造ツリー内に、正しい読み取り順序で表現されている」ことを必須としています。PLOP が署名フィールドについて正しい読み取り順序を決定することはできませんので、入力文書内で然るべきフォームフィールドを用意する必要があります。Acrobat XI でこれを既存の PDF/UA 文書に対して行うには以下のように操作します：

- **「ツール」**パネルを開き、**「フォーム」**サブセクションを開きます。**「作成」**を選択します。
- 現れたダイアログで、**「既存の文書から」→「次へ」→「現在の文書」**を選択します。
- **「フォーム」**パネルの**「タスク」**セクションで、**「新しいフィールドを追加」→「電子署名」**をクリックして、ページ上にフォームフィールド長方形を描きます。
- **「フォームの編集を閉じる」**をクリックして、**「タグ」**パネルを開きます。
- この**「タグ」**パネルの上端にあるオプションボタンをクリックして、**「検索 ...」**を選択します。
- 現れたダイアログで、**「マークされていない注釈」**を選択し、**「検索」**をクリックします。
- 作成したばかりの署名フィールドがハイライトされていることを確認してください。**「エレメントを検索」**ダイアログで**「タグエレメント」**をクリックし、**「種別：*Form*」**を選択します。ここでフィールドタイトルを与えることもできます。そして**「*OK*」**をクリックします。
- 「タグ」パネル内で、新たに作成された ***Form*** 構造エレメントが、タグリストの末尾に現れているはずです。そのタグを選択し、それをタグヒエラルキー内の、その署名フィールドが読み取られてほしい位置に照応する、然るべき位置へ移動させます。

署名フィールドに ***Signature1*** という名前が付けてあるとして、新規の署名のためのターゲットフィールドとして、署名オプションリスト内でその名前を参照できます：

```
field={name=Signature1}
```

あるいは、既存のフィールドの中へ、その名前にかかわらず、署名を配置するよう PLOP DS に命じることもできます：

```
field={fillexisting}
```

field 署名オプションの ***tooltip*** サブオプションを用いると、その署名フィールドの、スクリーンリーダソフトウェアによって使われるための、然るべき代替説明を与えることができます。

PDF/UA 準拠にこだわらない場合には、その規格準拠エントリを除去することもできます：

```
sacrifice={pdfua}
```

PDF/X・PDF/VT 準拠　PDF/X・PDF/VT モードでは署名視覚化には対応していません。

PDF/X 準拠にこだわらない場合には、その規格準拠エントリを除去することもできます（PDF/VT についても同様です）：

```
sacrifice={pdfx}
```

6.3.3 文書セキュリティストア（DSS）

文書セキュリティストア（Document Security Store ＝ DSS）という専用の PDF データ構造は、証明書と、関連する失効情報を保持することができます。このデータを、まとめて検証情報といい、長期検証のために重要な役割を果たします。この DSS は、PAdES パート 4

で導入されたものであり、ISO 32000-2 に盛り込まれる予定です。Acrobat X 以上ではこれに対応しています。この DSS は、認証・証明用署名に対してはオプショナルですが、文書タイムスタンプとタイムスタンプ付き署名の長期検証を可能にするには必須です。

署名でなく DSS 内に検証情報を保管すると、ファイルサイズを削減できます。なぜなら署名オブジェクトと異なり、DSS は圧縮することが可能であり、かつ ASCII 表現（署名のサイズを倍増させる）を必要としないからです。さらに、DSS は、複数の署名を検証するためのデータを保持することが可能です（たとえば、署名用証明書と TSA 証明書のための共通のルート CA 証明書を 1 個だけ保管すれば済みます）が、署名は、ただ 1 個の署名のための検証情報しか保持できません。

検証情報のなかには、署名オブジェクト内にしか保管できない項目もあり、DSS 内にしか保管できない項目もあり、どちらにも保管できる項目もあります。どちらにも保管できる項目については、署名オプション ***dss*** を用いると、その位置を制御することができます。この 2 種類の場所の比較を表 6.4 に挙げます。

表 6.4 検証情報のさまざまな項目の保管場所

	署名オブジェクト	文書セキュリティストア（DSS）	dss オプションによる制御
署名用証明書・TSA 証明書	可	—	—
署名用証明書・TSA 証明書以外の証明書（証明用証明書の発行者等）、および照応する OCSP レスポンスと CRL	可	可	可
署名用証明書のための OCSP レスポンスと CRL	可	可	可
TSA 証明書のための OCSP レスポンスと CRL[1]	—	可	—

1. タイムスタンプのための検証情報を埋め込む必要がある場合には、PLOP DS は常に、DSS を増分更新として追加します。

PLOP DS は、以前の署名のための検証情報を持った既存の DSS が入力文書内にあれば、それを温存します。これによって、既存の署名の LTV ステータスが必ず有効に保たれます。

Acrobat X 以上で、署名済み文書に DSS を追加するには、**「署名」**パネルを開き、**「オプション」**メニュー内で**「検証情報の追加」**をクリックします。

pCOS DSS の存在は、pCOS パス ***type:/Root/DSS*** を用いてチェックできます。DSS そのものがただちに LTV ステータスを保証するわけではないことに留意してください。なぜならそれは、関与している証明書群と失効情報の一部しか含んでいないかもしれないからです。

6.3.4 署名と増分 PDF 更新

デフォルトでは PLOP DS は、電子署名を入力文書に、増分更新として知られている PDF 技法を用いて追加します。増分更新では、入力文書の複製を作成し、署名データをその末尾に追加することによって、元の文書の内容と構造を温存します。署名オプション ***update=false*** を用いると、PDF DS は、増分 PDF 更新を追加するのでなく、PDF オブジェクト群のヒエラルキーを書き換えます。更新モードと書き換えモードにおける署名の比較を表 6.5 に挙げます。

暗号化文書に署名を行う 暗号化された入力文書に署名を行うことも可能です。ただし、***update=true*** を用いたデフォルトの署名モードでは、暗号化の諸特性を変えることはできません。これにより、以下の制約があります：

表 6.5 更新モードと書き換えモードにおける署名の比較

	更新モード (update=true)	書き換えモード (update=false)
既存の署名群	温存されます	失われます
認証・証明用署名のための DSS を追加	可	可
文書レベルタイムスタンプ・タイムスタンプ付き署名のための DSS（LTV のためには必須）を追加	可	—[1]
既存の DSS を温存	する	する
新しいパラメータを用いて暗号化 (userpassword・masterpassword・permissions)	—	可
最適化	—	可
破損した入力文書のための修復モード	—	可
署名速度	やや速い	やや遅い

1. タイムスタンプのための検証情報を埋め込む必要がある場合には、PLOP DS は常に、DSS を増分更新として追加します。

- 入力文書のマスターパスワードを *password* オプション内で与える必要があります。
- *userpassword*・*masterpassword*・*permissions* オプションは許されません。入力文書の照応する値が出力文書のために用いられます。
- *encryption* オプションは許されません。なぜなら、入力文書と同じ暗号化アルゴリズムを使用する必要があるからです。PLOP の一般的な戦略と異なり、これによって、もしも入力がすでに弱いアルゴリズムによって暗号化されていた場合には、弱く暗号化された出力が生成されてしまいます。

署名処理中に何らかの暗号化特性を変更したい場合には、*update=false* を用いて書名を行う必要があります。

署名済み文書の旧版へ復帰　増分更新は文書に情報を追加するだけですので、入力文書の構造は温存されます。署名された文書に変更が加えられた場合には、その署名済みバージョンを、その増分更新群を除去することによって再構築することもできます。Acrobat XI でこれを行うには以下のように操作します：

- 署名ページを開き、署名を選択します。
- **「署名バージョンを表示」**を選択すると、署名済みバージョンへ復帰します。

署名が別の増分更新の中の DSS を通じて LTV 対応にされている場合には、その更新は、署名済みバージョンへ復帰することによって除去されます。結果として、旧版内の署名は LTV 対応ですと表示されなくなりますが、文書全体の中ではこの同じ署名が LTV 対応ですと表示されます。これは、増分 PDF 更新を除去した結果であり、その文書全体の中の署名群の実際の LTV ステータスには影響を与えません。なお、この問題はタイムスタンプ署名では起こりません。なぜなら Acrobat XI は TSA に対しては完全検証を必須としないからです。

この現象は、DSS 内の検証情報が増分更新内に追加されている場合にのみ発生しますので、2 通りの方法で回避できます：

- *dss=false* と設定することにより DSS を避ける。
- *update=false* と設定することにより増分更新を避ける。

どちらの選択肢も、文書レベルタイムスタンプと、埋め込まれたタイムスタンプに対しては効果がありません。なぜならこれらは常に増分更新内に DSS を必要とするからです。

pCOS　増分更新による文書の版の数はpCOS擬似オブジェクト *revisions* で報告されます。署名はそれぞれが新たな版を生み出しますが、版は他の変更によって生み出されることもあります。たとえば DSS の追加です。ですので、版の数は、その文書内の署名の数よりも大きくなる場合があります。

6.3.5 証明用署名

証明用（作成者）署名については、「証明用署名」（69 ページ）でさわりを紹介しました。証明用署名を持った文書を開くと、Acrobat は、上端付近の青い署名バーの中に青いリボンを表示し、Acrobat の「署名」パネルの中にもその署名を表示します（それが有効な場合には青いリボンとともに）。証明用署名は、その署名を無効にすることなく、その文書に対してどの種類の変更を行うことができるかを指定します（図 6.3・表 6.6 参照）。証明用署名を PLOP DS で作成するには *certification* オプションを用います。

表 6.6 証明用署名を無効にすることなく許される文書変更

署名の種類（オプションリスト）	署名を無効にすることなく許される変更				
	フォームフィールドに値を記入	電子署名・ページ追加[1]	注釈を作成・削除・変更	署名フィールドを追加[2]	その他の変更すべて
certification=nochanges	—	—	—	—	—
certification=formfilling	可	可[3]	—	—	—
certification=formsandannotations	可	可[3]	可	—	—
certification=none（すなわち認証署名か文書レベルタイムスタンプ署名）	可	可	可	可	—

1. ページテンプレートから産み出すという、まれにしか使われない技法を用いてページを追加することが可能かどうかです。「ツール」→「ページ」→「ページの挿入」を用いて手動でページを追加することはできません。
2. 「入力と署名」→「署名を配置」を通じて署名を追加することが可能かどうかです。「ツール」→「フォーム」→「編集」を用いてフォームフィールドを追加することはできません。
3. 署名フィールドをクリックすることによって署名を行うことのみ可能です。Acrobat のメニュー項目を通じて行うことはできません。

以下の署名オプションは、その署名を無効にすることなくフォーム記入が許される証明用署名を作成します：

```
digitalid={filename=demorsa2048.p12} passwordfile=pw.txt certification=formfilling
```

preventchanges サブオプションを用いると、Acrobat ユーザーインタフェース内の、注釈ツール群等、署名を無効にしてしまうであろうツールを無効にすることができます。こうしておけば、ユーザーが、証明用署名を無効にしてしまうであろう変更を行おうとするおそれはなくなります。Acrobat を用いて文書を証明する際には変更は常に防止されます。この *preventchanges* オプションはデフォルトで *true* に設定されています。*preventchanges=false* の場合には、Acrobat は、上端付近に青い証明バーを表示しなくなり、すべての編集ツールを有効にします。しかし、許されない変更は証明用署名をやはり無効にします。

証明用署名は、必ず 1 個の文書の中の最初の署名であるべきですので、すでに署名を含んでいる文書に対しては行うべきではありません。

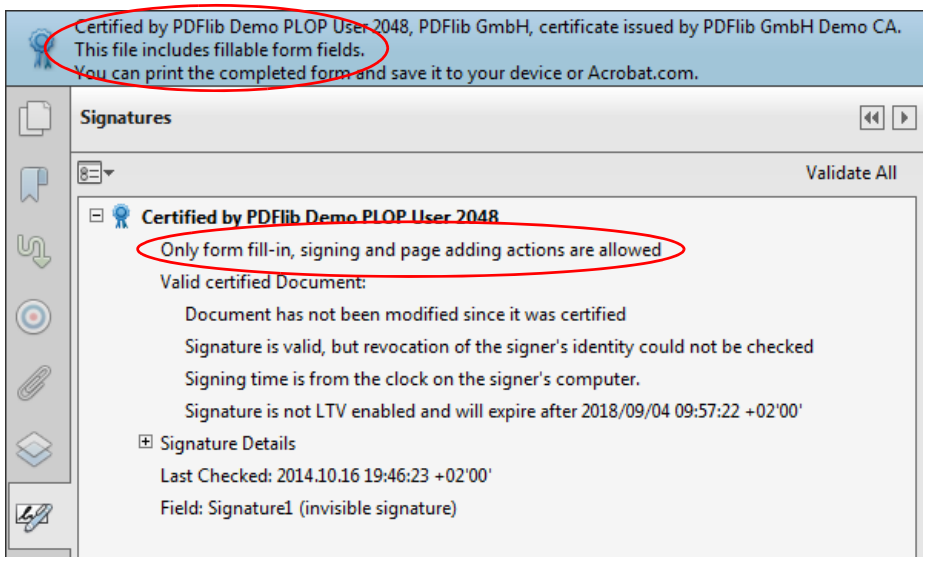
Acrobat における証明用署名の妥当性　証明用署名が技術的に有効な場合であっても、Acrobat において証明済み文書の利点を完全に活用するには、さらなる必要事項がいくつかあります：

- 証明用署名は、AATL CA からの証明書を用いると（「Acrobat における信頼済みルート証明書」（70 ページ）参照）、最も簡単に作成できます。Adobe Root CA は自動的に必要な信頼設定を持っていますので、構成手順は一切必要ありません。
- PKI からの、Adobe Root 下で働かないエンドユーザー証明書で証明用署名を作成しようとする場合には、Acrobat で、必要な信頼レベルをそのルート証明書に割り当てることを推奨します：
  「編集」→「環境設定 ...」→「署名」→「*ID* と信頼済み証明書」→「詳細 ...」→「信頼済み証明書」→そのルート証明書を選択→**「信頼を編集」→「証明済み文書」**を有効に。結果として、この選択されたルートの下の証明書を用いて作成される照明用署名はすべて、有効として受け入れられます。
- 個別の証明書に対して、必要な信頼レベルを設定することもできます。ただし、これは常道ではなく、推奨されません。以下のように操作します：
  「署名」パネルを開き、その証明用署名を選択し、**「証明書を表示 ...」**→証明書チェーン内で署名用証明書（すなわちリストの一番下のもの）を選択し、**「信頼」**タブを開き、**「信頼済み証明書に追加 ...」**をクリックし、通知メッセージダイアログで**「*OK*」**をクリックして、信頼設定を編集します。

上記の方法のいずれも行わないと、Acrobat はその証明用署名に、青いリボンでなく黄色い三角の印を付け、テキスト**「署名者の証明書は証明済み文書を作成する目的では信頼されていません」**を加えます。

pCOS　証明用署名はpCOSで*signaturefields[...]/sigtype= certification*として報告されます。許されている変更の種類は *signaturefields[...]/permissions* を用いてクエリできます。これはキーワード *nochanges*・*formfilling*・*formsandannotations* のいずれかを返します。

図 6.3 Acrobat で「フォームフィールドの入力と署名フィールドに署名を許可」とした証明用署名



pCOS 擬似オブジェクト ***signaturefields[...]/preventchanges*** を用いると、禁じられた変更を行うことにより証明用署名がうっかり無効にされてしまわないよう Acrobat のユーザーインタフェース要素が無効にされるかどうかをチェックできます。

6.4 証明書失効情報

署名は、その署名用証明書の失効ステータスに関する情報を含むこともできます。この情報は、署名検証ソフトウェアによって、その証明書が署名の時点においていまだ有効であった（失効させられていなかった）ことを保証するために用いられることができます。これを行うには 2 通りの方法があります。

Acrobat XI では、以下のようにして署名ビューア内で失効情報を確認できます：**「署名」**パネルを開き、その署名を右クリックして、**「署名のプロパティを表示 …」→「証明書を表示 …」**を選択し、**「失効」**タブへ行きます（図 6.4・図 6.5 参照）。

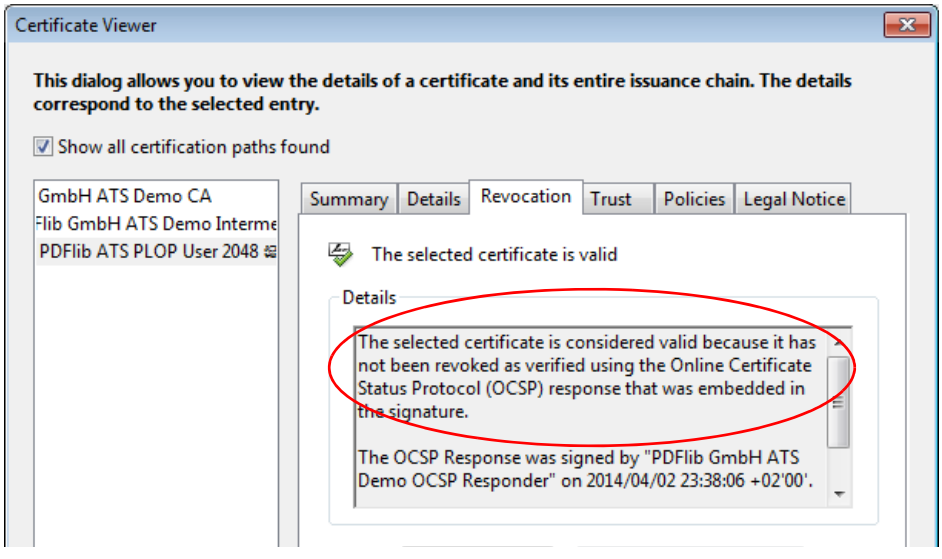
6.4.1 オンライン証明書ステータスプロトコル（OCSP）

注記 `engine=mscapi` **の場合には OCSP レスポンスの埋め込みには対応していません。**

OCSP の概要　RFC 2560・RFC 6960 に従った OCSP が使用されている場合、署名を行うソフトウェアは、その証明書のステータスをリアルタイムでクエリするために OCSP サーバ（OCSP レスポンダともいいます）へネットワークリクエストを送信します。OCSP レスポンダは、発行されたり失効させられたりした証明書群の、その CA のデータベースへのリアルタイムのアクセスを持つサーバです。この OCSP レスポンダは、そのクエリの時点でその証明書が有効かどうかを確認して、その結果を持った署名済みの応答を返します。この OCSP レスポンスはその署名に埋め込まれます。

証明書は、RFC 3280 に従った *ocsp* アクセス方式を持った *Authority Info Access*（AIA）という拡張を内容として持っていることもあります。それは、その証明書を発行した CA に関連付けられた OCSP レスポンダへの URL を内容として持ちます。あるいはこの URL を、*ocsp* 署名オプションを通じて与えることもできます。PLOP DS が、ある特定の証明書のための OCSP リクエストを送信した時には、その OCSP サーバは、その証明書に対するステータス**「有効」・「失効」・「不明」**のいずれかを持った応答を返します。**「有効」**な OCSP レスポンスを生み出すには、以下の条件がすべて満たされる必要があります：

図 6.4
Acrobat に表示された OCSP 情報



- *ocsp*アクセス方式を持ったAIA拡張がデジタルID内に存在する必要があります。あるいは、*ocsp* 署名オプションの *source* サブオプションを与える必要があります。
- OCSP レスポンダが、指定された URL においてネットワークを通じて到達可能であり、かつ、*ocsp* 署名オプションの *source* サブオプションの *timeout* サブオプションで指定された時間内に応答を送信すること。
- OCSPレスポンスが、そのOCSPレスポンダによって受け持たれているCAによってその証明書が発行されていること、かつ有効である（すなわちその有効期限に達していない）こと、かつ失効させられていないことを必要とするステータス**「有効」**を内容としていること。

OCSP レスポンスが**「有効」**である場合には、PLOP DS は、その応答を、生成される署名の中へ埋め込みます。そうでない場合には、*critical* サブオプションに応じて、その使用不能な応答は無視されるか、あるいは署名は生成されません。デフォルトでは PLOP DS は、署名者のデジタル ID の中にもし AIA 拡張があればその中の OCSP レスポンダの URL を用い、有効な OCSP レスポンスのない状況を黙殺します。しかし、*ocsp* オプションを用いて OCSP レスポンスの埋め込みが明示的に要求されている場合には、*critical* オプションが *false* に設定されていない限り、署名を生成するには、**「有効」**な応答が必要です。

OCSP の構成　使用している PKI に応じて、OCSP レスポンスに関する構成について以下の点を考慮する必要があります：

- 証明書内にAIA拡張がない場合には、*ocsp*オプションの*source*サブオプションを用いてOCSP レスポンダを与える必要があります。
- OCSP リクエストを作成するには、署名者の証明書の発行者のための有効な証明書が必要です。これは多くの場合、署名者のデジタル ID の中に含まれています。そうでない場合には別途、*rootcertdir* ／ *rootcertfile* ／ *certfile* 署名オプションのいずれかを用いて与える必要があります。
- OCSP レスポンダは、ネットワークコミュニケーションが成功するために認証を必要とする場合がありますので、OCSP リクエストではいくつかの認証オプションに対応しています。
- OCSP の**ノンス**機能は、反射攻撃を防ぐ一方で、キャッシングを妨げるのでパフォーマンスを低下させます。OCSP レスポンダの構成によっては、*nonce* オプションを用いる必要がある場合があります。以下のようなメッセージを受け取った場合、その OCSP レスポンダはノンス機能に対応していません。この場合には、オプション *nonce=false* を与えれば、ノンス機能を無効にできます：

```
OCSP response from URL 'http://ocsp.acme.com' for certificate 'CN = PDFlib GmbH...'
does not contain nonce although it was requested
```

- *ocsp*オプションの*hash*サブオプションを用いると、OCSPリクエストとレスポンスの中で証明書を識別するために使用される然るべきハッシュ関数を選択できます。ただし Acrobat XI は、SHA-1 ハッシュ関数を使用した OCSP レスポンスしか取り扱うことができず、他のハッシュ関数を用いた OCSP レスポンスを署名検証のために使うことができません。*ocsp* オプションのサブオプション *hash=sha1* を使えば必ず、Acrobat が失効ステータスを正しく決定できます。

OCSP レスポンダに対する失効確認　OCSP レスポンダの署名用証明書は、OCSP レスポンスを作成した時点において有効である必要があります。堂々巡り（OCSP レスポンダの証明書はさらなる OCSP レスポンスを必要とすることになる）を避けるため、OCSP レスポンダの証明書の中には、RFC 2560 に従った *id-pkix-ocsp-nocheck* 拡張を含めることが推

奨されています。ほとんどすべての商用 OCSP レスポンダではそのようになっています。あるいは、この証明書は CRL 配布点（*CRLdp*）拡張を内容とすることもできます。

OCSP のオプションリスト例　以下の例では、オプションリストの中の、OCSP レスポンスの埋め込みに関係する部分のみを示します。他の署名オプション群も適切に加える必要があります。

署名者のデジタル ID の中にある URL を用いて OCSP レスポンスの埋め込みを試み、*ocsp* アクセス方式を持った AIA 拡張がそのデジタル ID の中で得られない場合にはエラーを発して失敗：

```
ocsp={source={}}
```

または同等表現：

```
ocsp={}
```

AIA 拡張を用いた OCSP レスポンスの埋め込みが可能であれば行うよう要求し、しかしエラーが出たら黙殺：

```
ocsp={source={} critical=false}
```

または同等表現：

```
ocsp={critical=false}
```

デジタル ID 内に AIA 拡張があっても OCSP レスポンスを埋め込まない：

```
ocsp=none
```

OCSP レスポンダに対する URL とタイムアウト 1 秒を明示的に与え、たとえデジタル ID 内に AIA 拡張があってもその中のエントリをオーバライド：

```
ocsp={source={url={http://ocsp.acme.com/} timeout=1000} }
```

OCSP 試行が成功しなかった場合にはその署名処理を絶対に行わないようにし、かつ、ノンス機能に対応していない OCSP レスポンダに対してはノンス機能を無効に：

```
ocsp={critical nonce=false}
```

6.4.2 証明書失効リスト（CRL）

注記 `engine=mscapi` **の場合には CRL の埋め込みには対応していません。**

CRL の概要　RFC 3280 に従った CRL が用いられている場合には、CA は定期的に（1 日 1 回等）、まだ期限が切れてはいないけれども失効させられている証明書群の署名済みリストを作成します。このリストは、署名ソフトウェアからの入手を可能にされ、そして署名内へ埋め込まれます。このリストは、ネットワークを通じて取得することもできますし、ローカルに保管することもできます。CRL は、ある特定の継続期間（1 日等）を持っており、その継続期間が尽きる前にリフレッシュされる必要があります。CRL は、任意の数の失効させられた証明書を取り扱うことができますので、通常、OCSP レスポンスよりはるかに大きく（数メガバイトにも）なり、しかもそのサイズは事前にはわかりません。

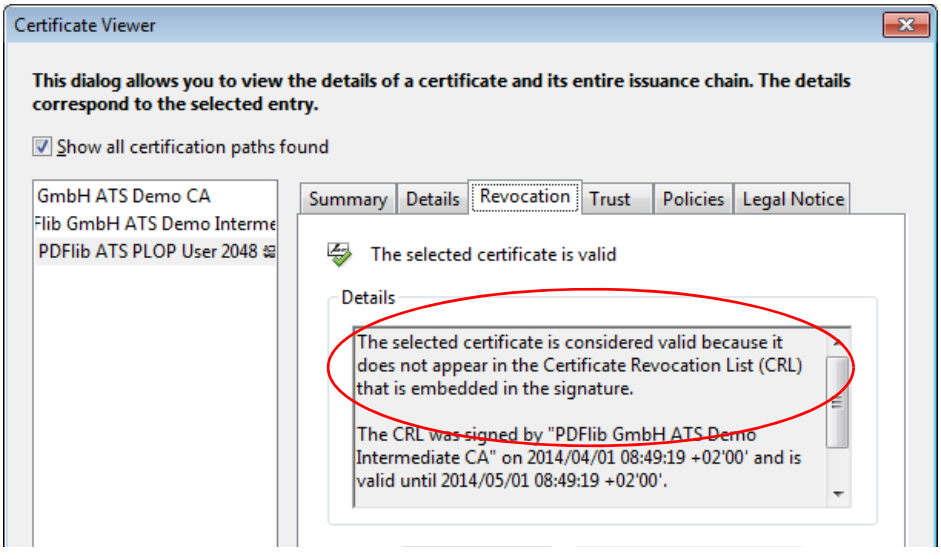
CRL はまるごと PDF 出力内へ埋め込まれますので、この種の失効情報は署名済み PDF 文書を肥大化させます。PLOP DS は CRL をいくつかの取得源から取得できます：

- 証明書は、***CRL 配布点***（***CRLdp***）という拡張を内容として持っていることがあります。これは 1 個ないし複数の CRL リソースのネットワーク URL を内容としています。PLOP DS は、CRL を取得できるまで、この ***CRLdp*** 拡張の中のすべてのエントリを試します。使用可能な CRL が見つかった場合には、それは署名内へ、あるいは文書セキュリティストア（Document Security Store ＝ DSS）内へ埋め込まれます（6.3.3 節「文書セキュリティストア（DSS）」（82 ページ）参照）。この ***CRLdp*** 拡張は、CRL を必要とする証明書それぞれについて、OCSP レスポンスの入手可能性と、照応する ***critical*** オプションに応じて、評価されます。
- この ***CRLdp*** 拡張のかわりに、署名用証明書のための CRL の取得を、***crl*** オプションを用いて構成することもできます。そのサブオプション ***source*** は、CRL が動的に取得されるネットワークアドレスを指し示します。サブオプション ***filename*** は、DER 形式の静的なローカル CRL ファイルを指し示します。
- 署名用証明書とその他すべての関与する証明書のための 1 個ないし複数のローカル CRL ファイルを、***crldir/crlfile*** 署名オプションを用いて PEM 形式で与えることもできます。

署名者の証明書が CRL に含まれている場合、それはその発行した CA によって失効させられており、すなわち、もはやそれを使用して有効な署名を作成することはできなくなっています。この場合、***PLOP_prepare_signature()*** は以下のようなエラーメッセージを発して失敗します：

```
Certificate verification failure for certificate with subject 'C = DE, L = Munich, O =
PDFlib GmbH, CN = PDFlib Demo PLOP User 2048': certificate revoked
```

図 6.5
Acrobat に表示された CRL 情報



PLOP DS は、CRL を、それが期限切れになるまで保持します。ある特定の CRL が、その継続期間が尽きたためにもう使えなくなった時にのみ、PLOP DS は新しい CRL をサーバから取り寄せます。

CRL のオプションリスト例　以下の例では、オプションリストの中の、CRL の埋め込みに関係する部分のみを示します。他の署名オプション群も適切に加える必要があります。

署名者のデジタル ID の中にある URL を用いて CRL の埋め込みを試み、***CRLdp*** 拡張がそのデジタル ID の中で得られない場合にはエラーを発して失敗：

```
crl={source={}}
```

または同等表現：

```
crl={}
```

CRLdp 拡張を用いた CRL の埋め込みが可能であれば行うよう要求し、しかしエラーが出たら黙殺：

```
crl={source={} critical=false}
```

または同等表現：

```
crl={critical=false}
```

デジタル ID 内に ***CRLdp*** 拡張があっても署名用証明書やその他任意の証明書に対して CRL を取得しようと試みない。これは、署名を行うコンピュータがオフラインである場合等、オンライン取得が必ず失敗することがわかりきっている場合には合理的です：

```
crl=none
```

CRL サーバに対する URL とタイムアウト 1 秒を明示的に与え、たとえデジタル ID 内に ***CRLdp*** 拡張があってもその中のエントリをオーバライド：

```
crl={source={url={http://crl.acme.com/} timeout=1000} }
```

ローカルディスクファイルの内容である CRL を与える：

```
crlfile={certs.pem}
```

6.4.3 OCSP か CRL か

失効情報を含める方式として最も然るべきものを選ぶにあたっては、以下の要素が意味を持ちます：

- OCSPはリアルタイムは証明書ステータス情報を提供します。1つのOCSPレスポンスはただ 1 つの証明書のみを取り扱いますので、そのサイズはわずか数キロバイトであると予測できます。他方で、OCSP は必ず OCSP レスポンダへのネットワーク接続を必要とします。
- CRL がOCSPに勝る利点は、ローカルに保管できるのでネットワークオーバヘッドを避けることができる点です。難点は、ローカルに保管された CRL は、頻繁にリニューアル（すなわち発行・取り寄せ）されない限り、内容が古くなってしまうおそれがある点です。

- CA の証明書を失効させる必要が生じることはまれですので、CA に対する CRL は通常、エンドユーザー証明書に対する CRL よりもはるかに小さくなります。
- 法令や、署名に関する内規によって、両方式のうちのいずれかが義務付けられている、あるいは禁止されている場合がありえます。

デフォルトでは PLOP DS は、有効な OCSP レスポンスが得られない場合にのみ CRL を署名内へ埋め込みますが、この動作は、***ocsp・crl*** オプションと ***critical*** サブオプションを用いて変えることもできます。

失効情報が必ず埋め込まれるようにするには以下のオプションリストを用います。この場合、OCSP が有効な応答を与えない場合にのみ CRL が取得されます：

```
ocsp={critical=false source={url={http://ocsp.acme.com/}}} ↵
 crl={critical=true  source={url={ http://crl.acme.com/}}}
```

この ***ocsp・crl*** オプションが失効情報の埋め込みを制御するのは、署名用証明書に対してのみであり、CA または TSA 証明書群が関与していてもそれらに対しては制御しないことに留意してください。

6.5 タイムスタンプ

6.5.1 構成

電子署名は、信頼された時刻サーバから取得された日時情報を含むこともできます。このようなサーバを、タイムスタンプ局（Time-Stamp Authority ＝ TSA）ともいいます。署名を行うコンピュータから採られた時刻（容易にごまかしが可能）とは異なり、信頼されたサーバから取得されたタイムスタンプは、署名の時点について、署名済みで信頼に足る情報源を提供します。PLOP DS は、RFC 3161 に従ったタイムスタンピングに対応しています。タイムスタンピング要求は、生成された署名のハッシュを含んでいますので、タイムスタンプは、その署名が特定の時点において作成されていることを確証します。タイムスタンプは、生成される PDF 署名の中へ埋め込まれます。

選択した TSA に応じ、タイムスタンプを作成するための構成について以下の点を考慮する必要があります：

- 最も重要な情報は、TSA へ到達できるネットワークアドレスです。これを与えるには *source* サブオプションの *url* サブオプションを用います。あるいは署名者のデジタル ID から採ることも可能です（「デジタル ID 内のタイムスタンプ拡張」（95 ページ）参照）。
- TSAを信頼するには、そのTSA証明書を発行したCAが信頼されている必要があります。この TSA の CA 証明書は、署名を検証する際、他の CA 証明書群と同様に処理される必要があります。詳しくは「中間 CA 証明書（群）を構成」（100 ページ）を参照してください。これは特に LTV 対応署名を作成する際に重要です。AATL ヒエラルキー群（「Acrobat における信頼済みルート証明書」（70 ページ）参照）のいずれかの下にある TSA を利用する場合には、その TSA 証明書の発行者、ないし発行者群のチェーンは、Acrobat に信頼済みルートとして知られています。ただし、その TSA CA 証明書を *certfile* オプションで与える必要がある場合があります。
- TSA は、クライアントがタイムスタンプを要求する際にある特定のハッシュアルゴリズムを使用するよう要求する場合があります。デフォルトでは PLOP DS は SHA-256 アルゴリズムを使用しており、これは現在のすべての TSA で動作します。これ以外のハッシュ関数を与えるには *hash* サブオプションを用います。
- TSA のなかには、自由にアクセスできるものもあれば、アクセスを制限するためにユーザー名とパスワードを要求するものもあります。権限のないアクセスには以下のようなメッセージが返されます：

  ```
  Network response from URL 'https://timestamp.acme.com/tsa' has bad status code 401
  ('Unauthorized'))
  ```

  権限パラメータ群を与えるには、URL に含めるか、あるいは *source* ネットワークサブオプションの *username/password* サブオプションを用います。
- TSA が SSL アクセス（すなわち *https*）を要求する場合には、そのサーバの SSL ルート証明書を、*sslcertdir/sslcertfile* オプションを用いて与える必要があります。そうしないと以下のようなメッセージを返されてしまいます：

  ```
  Document time-stamp request to 'https://timestamp.acme.com/tsa' failed
  ('Peer certificate cannot be authenticated with given CA certificates')
  ```

  必要なサーバ証明書を与えるのではなく、オプション *sslverifypeer=false* を用いることによって、サーバ証明書の確認をスキップすることもできます。ただし、そうすることのセキュリティ上の意味を認識している場合に限ります。
- TSA のなかには、ポリシー OID（オブジェクト識別子）を要求するものもあり、これを与えるには *policy* サブオプションを用います。この OID の適切な値についてはその

TSA とすり合わせておく必要があります。このポリシー OID は、Acrobat の**「署名のプロパティ」**ダイアログ→**「証明書を表示 ...」**に表示されます。

6.5.2 タイムスタンプ付き署名

注記 `engine=mscapi` **の場合にはタイムスタンプ付き署名には対応していません。**

認証・証明用署名は、埋め込まれたタイムスタンプを内容として持つこともできます。Acrobat 7 以上がタイムスタンプ付き署名に対応しています。

デジタル ID 内のタイムスタンプ拡張　デジタル ID は、タイムスタンプ局の URL を内容とする *TimeStamp* 拡張を内容として持っていることがあります。これにより、TSA の詳細を与える必要なく、タイムスタンプを埋め込んだ署名を行うことが可能になります。この *TimeStamp* 拡張は Adobe によって、そのパートナーCA 群のために仕様化されたものであり、通常、AATL（***Adobe* 認定信頼リスト**）プロバイダ群によって発行される証明書（「Acrobat における信頼済みルート証明書」（70 ページ）参照）の中には含まれています。

この *TimeStamp* 拡張があり、かつ認証を要求しない URL を内容としている場合には、PLOP DS はタイムスタンプを作成するために、その指定された TSA へアクセスを試みます。この場合には、タイムスタンプを作成するために *url* サブオプションを与える必要はありません。しかし、認証を要求する TSA を使用するには、たとえ *TimeStamp* 拡張内でその TSA が指定されていても、その TSA の完全な詳細を明示的にオプションリストで指定する必要があります（後述の例を参照）

タイムスタンピングのオプションリスト例　以下の例では、オプションリストの中の、タイムスタンプの埋め込みに関係する部分のみを示します。他の署名オプション群も適切に加える必要があります。

指定した URL にある TSA から取得したタイムスタンプを用いて、デフォルトハッシュアルゴリズム SHA-256 を使用して署名にスタンプ：

```
timestamp={source={url={http://timestamp.acme.com/tsa}}}
```

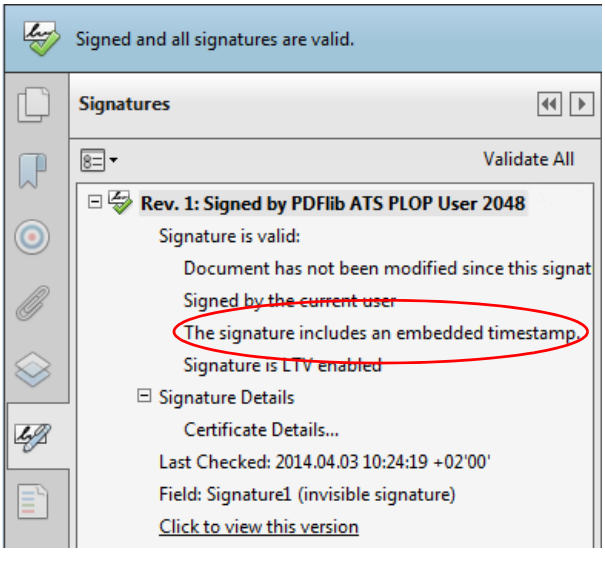


図 6.6
Acrobat におけるタイムスタンプ付き署名

タイムスタンプを取得するには TSA がユーザー名とパスワードを要求する所のタイムスタンプを用いて署名にスタンプ：

```
timestamp={source={url={http://timestamp.acme.com/tsa} username=demo password=demo}}
```

TSA がダイジェスト認証を要求する所のタイムスタンプを用いて署名にスタンプ

```
timestamp={source={url={http://timestamp.acme.com/tsa} httpauthentication=digest ↵
        username=demo password=demo }}
```

TSA が SSL を通じてアクセスされる必要がある場合には、そのサーバの SSL 証明書を、オプション *sslcertdir/sslcertfile* を用いて与える必要があります。そのサーバの SSL 証明書が手に入らない場合には、オプションを用いてサーバ認証をスキップすることも可能です。ただし、そうすることのセキュリティ上の意味を認識している場合に限ります：

```
timestamp={source={url={https://timestamp.acme.com/tsa}} sslverifypeer=false}
```

署名者のデジタル ID の中にある URL を用いてタイムスタンプを埋め込もうと試み、そのデジタル ID の中で *TimeStamp* 拡張が得られなければエラーを発して失敗：

```
timestamp={source={}}
```

または同等表現：

```
timestamp={}
```

たとえデジタル ID 内に *TimeStamp* 拡張があってもタイムスタンプを埋め込まない：

```
timestamp=none
```

6.5.3 文書レベルタイムスタンプ署名

文書レベルタイムスタンプは、PAdES パート 4 で導入されており、ISO 32000-2 に盛り込まれる予定です。Acrobat X 以上が文書レベルタイムスタンプに対応しています。

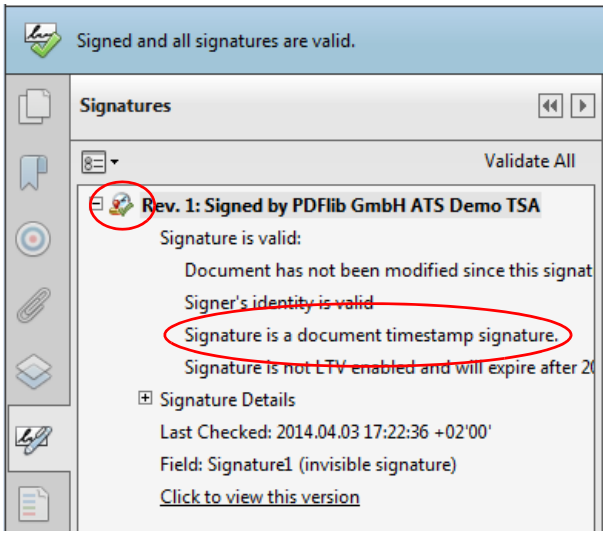


図 6.7
Acrobat における
文書レベルタイムスタンプ

タイムスタンプ付き署名と文書レベルタイムスタンプの違い　タイムスタンプ付き署名と同様に、文書レベルタイムスタンプは、ある特定の時点に紐付いたステータス情報を提供します。ただし、前者ではタイムスタンプは主たる署名の属性であるのに対して、文書レベルタイムスタンプは有効な署名そのものです。それはデジタル ID を必要としません。なぜなら署名する人や主体というものがないからです。かわりに文書レベルタイムスタンプは、タイムスタンプ局（TSA）へのネットワーク要求を通じて作成されます。文書レベルタイムスタンプは、ある特定の文書が、そのタイムスタンプで示された時刻に存在していたことを保証します。

注記 `engine=mscapi` **の場合には文書レベルタイムスタンプ署名には対応していません。**

文書レベルタイムスタンピングのオプションリスト例　以下の例では、文書タイムスタンプを作成するための完全な署名オプションリストを示します。署名用証明書が必要ありませんので、他の署名オプションは一切必要ありません。

指定したURLにあるTSAから取得した文書レベルタイムスタンプを、デフォルトハッシュアルゴリズム SHA-256 を使用して追加：

```
doctimestamp={source={url={http://timestamp.acme.com/tsa}}}
```

ユーザー名とパスワードを要求する TSA からの文書レベルタイムスタンプを追加：

```
doctimestamp={source={url={http://timestamp.acme.com/tsa}} username=demo password=demo}
```

ダイジェスト認証を要求する TSA からの文書レベルタイムスタンプを追加：

```
doctimestamp={source={url={http://timestamp.acme.com/tsa} httpauthentication=digest ↵
        username=demo password=demo}}
```

pCOS　文書レベルタイムスタンプは pCOS で *signaturefields[...]/sigtype=doctimestamp* として報告されます。

6.5.4 対応していない TSA

以下に述べる状況では、TSA を使用して PLOP DS で PDF 文書に署名を行うことはできません。

新しいタイムスタンピングプロトコル　PDF は、RFC 3161 に従ったタイムスタンピングプロトコルに対応していますが、新しい RFC 5816 には対応していません。この新しいプロトコルは、*ESSCertIDv2* と *SigningCertificateV2* を導入した RFC 5035 に基づいています。このような TSA を使用して PDF 文書に署名を行うことはできません。PLOP DS は以下のエラーメッセージを発します：

```
Signature verification of time-stamp failed: ess signing certificate error
```

属性証明書　PLOP DS は属性証明書には対応していません。TSA がそれを使用している場合、PLOP DS は以下のエラーメッセージを発します：

```
Time-stamp authority 'http://adobe-tsa.entrust.net/TSS/HttpTspServer'
uses unsupported protocol ('wrong tag')
```

属性証明書はとりわけ、TSA の時刻監査証明書（Time Auditing Certificate ＝ TAC）のために用いられます。TSA 製品のなかには、RFC 2630 に従った新しい CMS 文法を用いて TAC

を符号化するものがあります。PLOP DS はこれに対応していません。ただし、RFC 3126 に従った署名済み属性の中に TAC を入れる等の代替方式を用いて TAC を符号化するようそのような製品を構成することも可能です。

鍵用途拡張内に「クリティカル」フラグがない

タイムスタンピングプロトコル RFC 3161 では、TSA 証明書が**「拡張鍵用途」**フィールド拡張を含み、その値が**「タイムスタンピング」**で、かつこの拡張がクリティカルとして標識されていることを必須としています。この拡張が TSA 証明書内にありながら、クリティカルとして標識されていない場合には、Acrobat はその署名を無効として拒絶します。

PLOP DS は、このような TSA 証明書を使用して生成されたタイムスタンプを、以下のエラーメッセージとともに拒絶します：

```
Signature verification of time-stamp failed: certificate verify error:
Verify error:unsupported certificate purpose
```

「拡張鍵用途」フィールドに「クリティカル」フラグが設定されていない TSA 証明書を使用して Acrobat で文書タイムスタンプを作成しようとすると、以下のエラーメッセージが出ます：

```
Error encountered while signing:
Certificate is not valid for the usage
```

そのような TSA を使用してタイムスタンプを埋め込んだ証明用または認証署名を Acrobat で作成すると、成功しますが、検証の際には、生成された署名の中のタイムスタンプが以下のメッセージとともに拒絶されます：

```
The signature includes an embedded timestamp but it is invalid
```

Authenticode タイムスタンプサーバ Authenticode は、Microsoft のタイムスタンピングプロトコルであり、コード署名をその主用途としています。Authenticode は、RFC 3161 でなく、古い RFC 2985/PKCS#9 に基づいていますので、PDF も PLOP DS もこれには対応していません。

PLOP DS は、Authenticode TSA を使用して生成されたタイムスタンプを、以下のようなエラーメッセージとともに拒絶します。

```
Unexpected content type 'text/html;charset=ISO-8859-1' in reply to time-stamp request to
URL 'http://timestamp.entrust.net/TSS/AuthenticodeTS'
(expected content type 'application/timestamp-reply')
```

または

```
Unexpected content type 'application/timestamp-query' in reply to time-stamp request to
URL 'http://timestamp.verisign.com/scripts/timstamp.dll'
(expected content type 'application/timestamp-reply')
```

Authenticode TSA を Acrobat で使用しようとすると以下のエラーメッセージが出ます：

```
Error encountered while signing:
Error encountered while BER decoding
```

6.6 長期検証（LTV）

6.6.1 LTV の概念と Acrobat の対応

長期検証（Long-Term Validation ＝ LTV）は、署名を、その署名用証明書が期限切れになったり失効させられたりした後でもなお検証できるという意味を持ちます。これは、署名済み文書を長期間にわたってアーカイブするためには重要な特徴です。この LTV の概念は、PAdES パート 4（ETSI TS 102 778-4）で論じられており、Acrobat XI はこれに対応しています。

署名を LTV 対応にするには、その完全な証明書チェーンと、関与するすべての証明書に対する失効情報、すなわちまとめて検証情報と呼ばれるものを、その署名の中へ、あるいは DSS（6.3.3 節「文書セキュリティストア（DSS）」（82 ページ）参照）内へ埋め込む必要があります。LTV のために署名関連データを追加で埋め込む必要があることから、その署名済み文書は概して、LTV 対応でない署名よりも大きくなります。

注記 `engine=mscapi` **の場合には、LTV 対応署名には対応していません。**

LTV 対応署名は、埋め込まれたタイムスタンプを含むべきですが、これは厳格な要請ではありません。LTV 対応署名の継続期間を、その関与する証明書群のうちのいずれかが期限切れになるか失効させられる前に文書レベルタイムスタンプ署名を追加することによって延ばすことも可能です。

LTV ステータスは、絶対的に定義されるのではなく、信頼済みルート証明書群の 1 つの集合との関連において定義されます。構成によって、ある特定の署名が、ある構成では LTV 対応と見なされ、別の構成では LTV 対応でないと見なされることもありえます。たとえば、PLOP DS 内と Acrobat 内とで別々の信頼済みルート群を構成すれば、LTV ステータスは異なる可能性があります。

Acrobat における LTV ステータス　Acrobat XI は**「署名」**パネル内に、ステータス行「署名は LTV 対応です」または「署名は LTV 対応ではなく、… を過ぎると有効期限が切れます」を表示します（図 6.8 参照）。LTV ステータスに関して以下のことに留意してください：

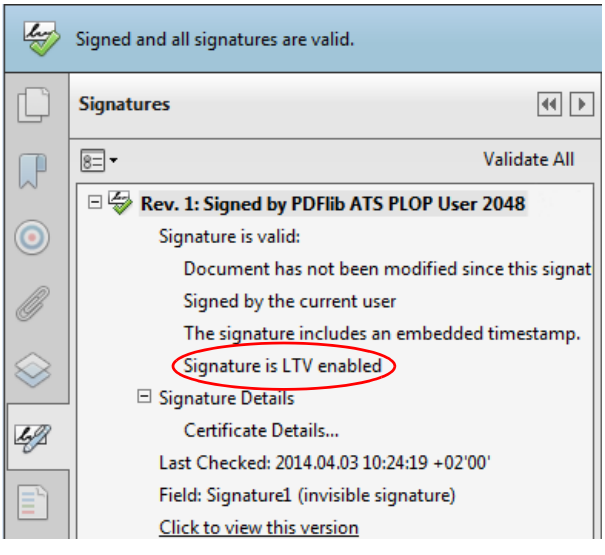


図 6.8
Acrobat における LTV 対応の署名

- 関与するすべての証明書に対するルート証明書（群）が Acrobat 内で信頼済みとして構成されている必要があります（「Acrobat における信頼済みルート証明書」（70 ページ）参照）。
- Acrobat では、任意の有効な署名について、その直接の署名用 CA 証明書を信頼済みルートストアへ追加することによって、LTV 対応ですと強制的に表示させることができます。このことは、その構成を考えに入れていないとその LTV ステータスについて混乱を招きかねません。これはまた、自己署名証明書を使用して作成された証明書が信頼済みルート証明書に追加されれば LTV 対応として扱われることにもつながります。
- Acrobat XI は、埋め込みタイムスタンプがなくても LTV ステータスを認めます。タイムスタンプが埋め込まれている場合、Acrobat はその TSA 証明書に対して検証情報を要求しません。PLOP DS はもっと厳格であり、かつ、TSA 証明書に対して完全な検証情報の要求も行います。
- Acrobat XI は、OCSP レスポンスについて、SHA-1 ハッシュ関数にのみ対応しています（「OCSP の構成」（89 ページ）参照）。ですので、それ以外のハッシュ関数が使用されている場合、完全な検証情報が実際には利用可能でありながら、Acrobat はその LTV ステータスを正しく表示しないことがあります。
- Acrobat で以下の設定を有効にすると、LTV ステータスが表示されなくなりますので、有効にしてはいけません：**「環境設定」→「署名」→「検証」→「詳細 ...」→「検証時刻」→「署名の検証に使用する時刻 ：現在の時刻」**。
- LTV ステータスは、以前の版へ復帰することによって失われる場合があります。詳しくは「署名済み文書の旧版へ復帰」（84 ページ）を参照してください。

6.6.2 PLOP DS を用いた LTV 対応署名

以下のオプションを与えると、PLOP DS は、すべての検証情報が与えられていれば、LTV 対応署名を作成します。そうでなければ署名を作成せずエラーを発します：

```
ltv=full
```

このオプション単独では、必ず LTV 対応署名になるわけではなく、すべての必要事項が満たされていることを確認するだけです。証明書が見つからない場合、または検証情報を取得できなかった場合には、PLOP DS はエラーメッセージを発します。ですので、すべてのエラーメッセージを徹底的に分析することが重要です。

デフォルト設定 ***ltv=try*** の場合には、入手可能なすべての失効情報が、署名される文書の中へ埋め込まれますが、たとえその情報が LTV ステータスの実現には不充分であってもその署名呼び出しは失敗しません。

すべてのチェーンに対して信頼済みルート証明書を構成　関与しているすべての証明書を完全に検証するために、PLOP DS はすべての証明書に対してトラストアンカーを必要とします。その正確な数は PKI 構成の内容によります。信頼済みルート証明書群を、***rootcertdir*** または ***rootcertfile*** オプションを用いて与える必要があります。これは最低限、署名用証明書のためのチェーンの最上位にあるルート CA の証明書については行う必要があります。関与するすべての証明書チェーンの最上位にただ 1 つのルート CA があるのでない限り、その他に TSA 用等のルート証明書群も必要になる場合があります。

中間 CA 証明書（群）を構成　残りの証明書チェーン（すなわち、ルート証明書と署名用証明書等関与している証明書との間のすべての中間 CA）が、PLOP DS がそれを署名内へ埋め込めるよう、入手可能である必要があります。署名用証明書や、その他、OCSP レス

ポンダ証明書や TSA 証明書等関与している証明書に対する CA 証明書群は、以下の場所で検索されます：

- CA 証明書群を、*certfile* オプションを用いて与えることもできます。
- （*engine=mscapi* の場合には不可）署名用証明書に対する CA 証明書群を、その署名者のデジタル ID を内容として持つ PKCS#12 ファイルの中に含めることもできます。
- （*engine=mscapi* の場合のみ）CA 証明書群は、Windows 証明書ストア内で検索されます。
- 証明書は、RFC 3280 に従った *calssuers*（認証局発行者）アクセス方式を持った *Authority Info Access*（AIA）拡張を内容として持っていることがあります。この拡張は、署名用証明書を発行した CA の証明書と、場合によっては中間 CA 証明書群を発行した CA の証明書を取得できる 1 個ないし複数の URL を内容としています。プロトコル *http*・*https*・*ftp* に対応しています。

証明書はその AIA 拡張内で LDAP プロトコルを指定している場合がありあすが、現在のところ PLOP DS はそれには対応していません。この場合には、LDAP ブラウザ[1] を使って手作業で LDAP を通じてその CA 証明書を取得すれば、それを上述のオプション群に与えることができます。これは署名用証明書について 1 回だけ行えばよいので現実的です。

どの CA 証明書群を構成する必要があるか　LTV ステータスを実現するための具体的な必要事項は PKI 構成によって異なります。多くの場合は以下の手順で充分です：

- 多くの商用 CA によって発行された証明書は *calssuers* アクセス方式を持った AIA 拡張を含んでいます。ですので PLOP DS は、その署名用証明書に対する CA 証明書群のチェーンを自動的に取得できます。そのルート CA 証明書だけを、*rootcertdir* または *rootcertfile* オプションを用いて与える必要があります。
  calssuers アクセス方式を持った AIA 拡張が署名用証明書内にないときは、多くの場合、必要なルート証明書（群）をその CA のウェブサイトからダウンロードできます。
- TSA と OCSP レスポンダの証明書は自動的に取得されます。これらの証明書、ないしはいずれかの中間証明書が署名用証明書とは異なるルート CA によって発行されている場合には、そのルート証明書を、*rootcertdir* または *rootcertfile* オプションを用いて与える必要があります。
- CRL は多くの場合、クエリされている証明書を発行したのと同じ CA によって署名されています。しかし、もしも CRL が別の CA によって署名されている場合には、その照応する CA 証明書を、*certfile* オプションを用いて与える必要があります。なぜならそれを自動的に取得することはできないからです。CRL を署名するために使用された証明書が、署名用証明書とは異なるルート CA によって発行されている場合（別の PKI に基づく TSA に対する CRL 等）には、そのルート証明書を、*rootcertdir* または *rootcertdir* オプションを用いて与える必要があります。

validate=full または *ltv=full* の場合、PLOP DS は、必要な CA 証明書が見つからないとき、エラー「*unable to get local issuer certificate*」を発します。この場合には、その足りない証明書を、オプション *rootcertdir*・*rootcertfile*・*certfile* のいずれかで与える必要があります。以下のメッセージ：

```
Certificate verification failure for certificate with subject '...':
self signed certificate in certificate chain
```

は通常、信頼済み自己署名証明書を、*rootcertfile* または *rootcertdir* オプションでなく *certfile* オプションで与えたときに発せられます。

1. たとえば www.ldapbrowser.com/ で入手可能なフリーの Softerra LDAP Browser。

署名用証明書に対する失効情報　署名用証明書に対する失効情報を、以下のいずれかの手段によって与える必要があります：

- 署名者の証明書の中の *AIA* 拡張か *ocsp* オプションを通じた OCSP。
- 署名者の証明書の中の*CRLdp*拡張か*crl*オプションを通じたCRL。既存のCRL群を、*crldir*・*crlfile* オプションを用いて与えることもできます。

ocsp・*crl* オプションの *critical* サブオプションを用いると、必ずその署名用証明書に対するOCSPまたはCRL情報の取得が成功した場合にのみ署名が作成されるようにすることができます。詳しくは 6.4 節「証明書失効情報」（88 ページ）を参照してください。

その他の関与している証明書群に対する失効情報　証明書チェーン内のすべての CA の証明書に対する失効情報と、CRL と OCSP レスポンスに署名を行うために使用されているすべての CA の証明書に対する失効情報も、入手可能である必要があります。ただし例外として以下の場合には失効情報は必要ありません：

- *rootcertdir* または *rootcertfile* オプションへ与えたルート CA 証明書群。
- *id-pkix-ocsp-nocheck* 拡張を含んでいる（通常は含んでいます）、OCSP レスポンダの証明書。

署名用証明書以外の証明書群に対する失効情報を、以下のいずれかの手段によって与えることができます：

- 証明書内の *AIA* 拡張を通じた OCSP。
- 証明書内の *CRLdp* 拡張を通じた CRL。既存の CRL 群を、*crldir*・*crlfile* オプションを用いて与えることもできます。

LTV のオプションリスト例　PKI が以下のように設定されている場合を考えます：

- 署名者のデジタル ID が、CA 証明書群のチェーンを、その PKCS#12 ファイル内に内容として持っているか、あるいは、ルート証明書以外の各証明書が、*caIssuers* アクセス方式を持った AIA 拡張を内容として持っている。
- 署名者のデジタル ID と、ルート証明書以外のチェーン内のすべての CA 証明書が、*ocsp* アクセス方式を持った AIA 拡張か、*CRLdp* 拡張を内容として持っている。
- OCSP レスポンダの証明書が *id-pkix-ocsp-nocheck* 拡張を内容として持っている。

この場合には、LTV ステータスを実現するには *rootcertfile* オプションだけが（そのデジタル ID を取り扱うためのオプション群に加えて）必要です。オプション *ltv=full* を用いると、LTV の必要事項への違反が必ず検出されるようになり、LTV ステータスが実現できない場合には署名が作成されなくなります：

```
digitalid={filename=demorsa2048.p12} passwordfile=pw.txt ltv=full rootcertfile=root1.pem
```

タイムスタンプを埋め込むためには、LTV ステータスを実現するには、その TSA 証明書に対する失効情報も入手可能である必要があります。その TSA 証明書が、*ocsp* アクセスメソッドを持った AIA 拡張か、*CRLdp* 拡張を内容として持っており、かつ、そのルート CA が署名用証明書と同じであることが理想です。この場合には、LTV ステータスを実現するためにさらなるオプション指定は必要ありません：

```
digitalid={filename=demorsa2048.p12} passwordfile=pw.txt ltv=full ↵
        rootcertfile=root1.pem timestamp={source={url={http://timestamp.acme.com/tsa}}}
```

しかし、そのTSAが別のルートCAに基づいている場合には、そのTSAルートを*rootcertfile*オプションで与える必要があります（ファイル *root1+2.pem* が、必要なルート証明書をPEM形式で両方とも内容としていると前提）：

```
digitalid={filename=demorsa2048.p12} passwordfile=pw.txt ltv=full ↵
        rootcertfile=root1+2.pem timestamp={source={url={http://timestamp.acme.com/tsa}}}
```

6.7 CAdES・PAdES 署名規格

6.7.1 CMS・CAdES 署名

欧州電気通信標準化機構（ETSI）[1]は、電子署名に関する欧州指令の採用と実装の促進および EU 加盟諸国間の電子署名の調和を図るため、数多くの電子署名規格を発行しました。この ETSI 規格群は世界のそれ以外の地域でも非常に影響力があります。たとえば、Acrobat X 以上がこれらに対応しており、PDF 2.0 標準 ISO 32000-2 内で参照されており、さまざまな RFC にも取り入れられています。

CMS・CAdES 署名　長い間、PDF 署名は CMS（Cryptographic Message Syntax ＝暗号メッセージ構文）に基づいていました。CMS は、古い PKCS#7 バージョン 1.5 形式に基づいています。これは、RFC 5652 で仕様化されており、インターネットで広く使われています。PDF 内の CMS 署名は、署名辞書内において ***adbe.pkcs7.detached*** かもっと古い何らかの非推奨のサブフィルタエントリを使用しています。CMS 署名は、すべてのバージョンの Acrobat で作成・検証できます。

CAdES（***CMS Advanced Electronic Signatures*** ＝ CMS 高度電子署名）は、ETSI TS 101 733（技術的に RFC 5126 と同等）で仕様化されており、CMS にいくつかの機能を追加したものです。最も重要な点として、これは、証明書置換と呼ばれる脅威シナリオに対して、署名用証明書への参照を署名内へ含める（***signing-certificate-v2*** 属性を用いて）ことによって防御します。PDF 内の CAdES 署名は、PDF 署名辞書内において ***ETSI.CAdES.detached*** サブフィルタを必要とします。CAdES 署名の作成・検証には Acrobat X 以上で対応しています。

pCOS　CAdES 署名は pCOS で ***signaturefields[...]/cades=true*** として報告されます。

各種 PAdES 署名　PAdES（***PDF Advanced Electronic Signatures*** ＝ PDF 高度電子署名）は ETSI TS 102 778 で仕様化されています。これは、PDF 1.7（ISO 32000-1）で定義されている通りに、PDF 署名に各種オプションと各種制約を追加することによって、CAdES を PDF に適用します。PAdES ではさらなる各種 PDF データ構造の仕様も定めており、これらは PDF 2.0（ISO 32000-2）に盛り込まれる予定です。PAdES はいくつかのパートから成ります（ここで関係のないパート群は以下省略しています）。

- PAdESパート2はETSI TS 102 778-2で仕様化されています。これはCMSに基づいているので PAdES-CMS とも呼ばれ、ISO 32000-1 に準拠していますがそのいくつかのオプショナルな機能を禁じています。
- PAdESパート3はETSI TS 102 778-3で仕様化されています。これはCAdESに基づいており、2 つの種類 BES（***Basic Electronic Signature*** ＝基本電子署名）と EPES（***Explicit Policy-based Electronic Signature*** ＝明示的ポリシーベース電子署名）を定義しています。EPES は BES を、署名にポリシー識別子と、オプショナルな関与種別表出とを追加することによって拡張しています。この ***policy*** 属性は、その署名がいかなる署名ポリシーのもとで作成されたかを指定します。この ***commitment-type*** 属性を、署名辞書内の ***Reason*** エントリのかわりとして使うこともできます。CAdES では、**「発信証明」・「受取証明」・「認可証明」**等一連の汎用の関与種別を定義しています。
- PAdES パート 4 は、ETSI TS 102 778-4 で仕様化されており、長期検証のための手段を提供します。長期検証については 6.6 節「長期検証（LTV）」（99 ページ）で詳しく説明します。パート 4 では、文書レベルタイムスタンプと DSS（6.3.3 節「文書セキュリ

1. ETSI の規格群は www.etsi.org/standards から無償で入手可能です。

ティストア（DSS）」（82 ページ）参照）を導入しています。PAdES パート 4 で定義されている各種概念は、PAdES パート 2・パート 3 署名に適用できます。

各種 PAdES 準拠レベル　さまざまなワークフローやアーカイブシナリオの要請に応えるために、いくつかの種類の PAdES 署名が定義されています。以下の各種 PAdES 準拠レベルが ETSI TS 103 172 で仕様として定められており、その各レベルは直前レベルの上に構築されています：

- PAdES-B（基本）は PAdES 署名の基礎的要素です。レベル B は委員会決定 2011/130/EU に準拠していると信じられています。これは、署名ポリシー識別子を持つ署名と持たない署名、すなわち EPES と BES に対応しています。
- PAdES-T（署名存在のための信頼時間）は、ある特定の日時にその署名が存在したことを証明するために PAdES-B にタイムスタンプを追加します。
- PAdES-LT（長期）は、長期検証を保証するために PAdES-T に検証情報を追加します。
- PAdES-LTA（アーカイブタイムスタンプ付長期）は、検証データの入手可能性と整合性を保証するために PAdES-LT に文書レベルタイムスタンプを追加します。

Acrobat における CAdES・PAdES 対応　デフォルトでは Acrobat 署名は PAdES パート 2（CMS）に準拠しています。Acrobat X・XI は、CAdES のために以下のように構成すれば、PAdES パート 3 の BES 署名を作成できます：

- Acrobat XI：**「編集」→「環境設定」→「署名」→「作成と表示方法」→「詳細 ...」→「デフォルトの署名形式：*CAdES* 相当」**
- Acrobat X：**「編集」→「環境設定」→「セキュリティ」→「詳細環境設定 ...」→「作成」→「デフォルトの署名形式：*CAdES* 相当」**

ポリシー識別子への対応がありませんので、PAdES パート 3 の EPES を Acrobat XI で作成することはできません。しかし、BES も EPES も Acrobat で検証できます。

Acrobat X と Acrobat XI は、PAdES パート 4 に以下の機能で対応しています：

- Acrobat XI：長期検証ステータス情報。詳しくは ≫LTV status in Acrobat≪, page 103 を参照してください。
- Acrobat X・XI：署名を LTV 対応に。**「署名」**パネルを開き、**「オプション」**メニューで**「検証情報の追加」**をクリックすることによって可能です。
- Acrobat X・XI：文書レベルタイムスタンプ。

6.7.2 PLOP DS を用いた PAdES 署名

PLOP DS は、上述の作成者署名と認証署名のためのすべての形式に対応しています。署名種別を選択するには *sigtype* オプションを用います。各種 PAdES 準拠レベルのための機能については個別のオプションで有効にします。デフォルトでは、PLOP DS 署名は PAdES パート 2（CMS）に準拠しています。PLOP DS で各種 PAdES 準拠レベルを実現するために必要なオプションを表 6.7 に挙げます。

注記 **engine=mscapi の場合には PAdES パート 3・パート 4 には対応していません。**

表 6.7 PAdES のパートと準拠レベル

署名種別	該当 PAdES パート	オプション
CMS	PAdES パート 2	sigtype=cms（デフォルト）
CMS-LTV（長期検証）	PAdES パート 2・パート 4	sigtype=cms ltv=full rootcertfile/rootcertdir[1]

表 6.7 PAdES のパートと準拠レベル

署名種別	該当 PAdES パート	オプション
PAdES-B（基本）	PAdES パート 3	PAdES-BES：sigtype=cades PAdES-EPES：PAdES-BES と同様に加えて policy
PAdES-T（信頼時間）	PAdES パート 3	PAdES-B と同様に加えて、timestamp を critical=true で
PAdES-LT（長期）	PAdES パート 4	PAdES-T と同様に加えて ltv=full rootcertfile/rootcertdir[1]
PAdES-LTA（アーカイブタイムスタンプ付長期）	PAdES パート 4	PAdES-LT と同様に加えて doctimestamp

1. LTV ステータスを実現するには他にも certfile・ocsp・crl 等オプションが必要な場合があります。6.6.2 節「PLOP DS を用いた LTV 対応署名」（100 ページ）を参照してください。

PAdES のオプションリスト例　以下の署名オプション（その他関連オプション群に加えて）は、PAdES-B BES に従った署名を作成します：

```
sigtype=cades
```

以下の部分的な署名オプションリストは、PAdES-B EPES に従った署名を作成します（架空の署名ポリシー識別子を用いています）：

```
sigtype=cades policy={oid=2.16.276.1.89.1.1.1.1.3 commitmenttype=origin}
```

以下の部分的な署名オプションリストは、PAdES-T に従った署名を作成します：

```
sigtype=cades timestamp={critical source={url={http://timestamp.acme.com/tsa}}}
```

以下の部分的な署名オプションリストは、PAdES-LT に従った署名を作成します：

```
sigtype=cades timestamp={critical source={url={http://timestamp.acme.com/tsa}}} ltv=full
```

以下の部分的な署名オプションリストは、PAdES-LTA に従った署名を作成します：

```
sigtype=cades timestamp={critical source={url={http://timestamp.acme.com/tsa}}} ↵
ltv=full doctimestamp={source={url={http://timestamp.acme.com/tsa}}}
```

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／値ペアどうしの間は、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、***{*** と ***}*** 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 対応の言語バインディングでは、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 非対応の言語バインディングでは、文字列を 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、zSeries の場合は ***ebcdic***、Unix・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。

長方形　長方形は、長方形の左下隅と右上隅の *x・y* 座標を指定した 4 個の浮動小数点値のリストです。これらの座標は、デフォルト PDF 座標系において、すなわち、ページの左下隅を原点として、ポイントをユニットとして解釈されます。例：

```
rect={ 100 100 200 150}
```

adapt キーワードを用いて自動的な寸法算出を行わせることもできます。6.3.1 節「署名をグラフィックかロゴで視覚化」（79 ページ）を参照してください。

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す。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(wstring filename, const void *data, size_t size, wstring 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アント型です。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。

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。ここうした関数のなかには、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

オプション	説明
copy	（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、copy オプションは自動的に true 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：false）。それ以外の言語バインディングでは、copy 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(wstring 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ますので解放されません。

C++ ***double info_pvf(wstring filename, wstring 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。

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

キーワード	説明
filecount	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。filename 引数は無視されます。
exists	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
size	（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。
iscopy	（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 copy オプションが与えられたなら 1、そうでないなら 0。
lockcount	（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 入力関数

C++ ***int open_document(wstring filename, wstring 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）を処理するために開きます。

filename 開きたい PDF ファイルのフルパス名。このファイルは *SearchPath* リソースを用いて検索されます。

Unicode 非対応言語バインディングでは、このファイル名は、*filenamehandling* オプションに従って UTF-8 へ変換されます（*filenamehandling=unicode* か、与えられたファイル名が UTF-8 BOM で始まっているのでない限り）。*len* が 0 以外の場合には（C 言語バインディングのみ）、このファイル名は、オプション *filenamehandling* にかかわらず、UTF-16 から UTF-8 へ変換されます。このファイル名が変換できないとき、またはこのファイル名が有効な UTF-8 か UTF-16 を構成していない場合には、エラーが発生します。

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。

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」（107 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 文書ハンドルは以下の目的に使用できます：

- 入力文書として、さらに *PLOP_create_document()* を用いて処理するために使用。
- 署名の体裁としてページを提供（署名オプション *field* とサブオプション *visdoc*）。
- pCOS を用いて文書情報をクエリ。

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

オプション	説明
inmemory	（論理値。PLOP_open_document( ) のみ）true 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）非常な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。false 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：false
password	（文字列。暗号化された文書に対しては、requiredmode 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 5.2（62 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 ebcdic エンコーディングか EBCDIC-UTF-8 で与える必要があります。update=true の場合には、これと同じパスワードが、生成される出力文書のためのマスターパスワードとして使用されます。
repair	（キーワード。update=true の場合には none が強制されます）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：auto）： ***force*** 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。 ***auto*** 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。 ***none*** 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。
requiredmode	（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（minimum/restricted/full）。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 requiredmode=minimum を指定した場合、結果は full モードになります。デフォルト：full
shrug	（論理値）以下の状況において各種アクセス制限が無視されます（すなわち PDF 処理が許されます）：文書がマスターパスワードを用いて暗号化されているが、そのユーザーパスワード（もしあれば）のみが与えられている。各種権限が無視されている時には、pCOS 擬似オブジェクト shrug は true に設定されます。デフォルト：false
xmppolicy	（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：rejectinvalid）： ***rejectinvalid*** 無効な XMP の場合には、XML パーサ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。 ***ignoreinvalid*** （sacrifice={pdfa pdfua pdfvt pdfx} 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 <pdfx:invalid_source_XMP_exception> 要素内に入れ込みます。 ***remove*** 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), wstring 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* へ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」（107 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++ ***void close_document(int doc, wstring optlist)***

C# Java ***close_document(int doc, String optlist)***

Perl PHP ***close_document(long doc, string optlist)***

VB ***Sub close_document(doc As Long, optlist As String))***

C ***void PLOP_close_document(PLOP *plop, int doc, const char *optlist)***

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。

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」（107 ページ）参照）。

詳細 この関数は、*PLOP_delete( )* を呼び出すより前に、*PLOP_open_document*( )* を用いて開かれている各文書について呼び出す必要があります。これは、与えられたハンドルに紐付いている文書を閉じて、関連するすべてのリソースを解放します。

表 7.4 PLOP_close_document( ) に対するオプション

オプション	説明
lastinthread	（論理値）カレントスレッド内の最後の文書を処理した後には、メモリリークを防ぐため、このオプションを true に設定するべきです。lastinthread=true の後に、その同じ PLOP オブジェクトに対して PLOP_create_document( ) を呼び出してはいけません。デフォルト：false

7.4 出力関数

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

PDF 出力文書を、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。

Unicode 非対応の言語バインディングの場合、*len=o* 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=o* の場合、ヌル終端文字列を与える必要があります。

optlist 表 7.5 に従ったオプションリスト（7.1 節「オプションリスト」（107 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。

電子文書が作成される場合、この関数呼び出しは、以下の場合には失敗します：

- タイムスタンプが取得できず、かつ、*critical* オプションが設定されている。
- 署名チェーンが再検証され、その間に証明書が期限切れになっているか失効させられている。
- ページに収まらない視覚化文書が与えられている。
- 入力文書が破損しており、かつ、署名が更新モードで作成されている。

詳細 この関数を呼び出す前に、*PLOP_open_document*( )* が呼び出されている必要があります。処理させたい文書を *input* オプションで与えます。ユーザーパスワードとマスターパスワードについて強いられる制約条件については 5.2 節「PLOP による PDF 暗号化」（61 ページ）を参照してください。

PLOP_create_document( ) は、その署名チェーンを、OCSP レスポンスがリクエストされてから期限切れになった等の理由で、再検証することがあります。

表 7.5 PLOP_create_document( ) に対するオプション

オプション	説明
docinfo	（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、標準文書情報項目は XMP へもミラーされます。オプションリスト内のそれぞれのペアには、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）： ***Subject*** 文書のサブタイトル ***Title*** 文書のタイトル ***Author*** 文書の作成者 ***Keywords*** 文書の内容を表すキーワード ***Trapped*** 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 True・False・Unknown です。PDF/X 入力の場合、sacrifice に pdfx か pdfvt があるなら、Unknown のみが許されます。 ***Creator*・*CreationDate*・*Producer*・*ModDate*・*GTS_PDFXVersion*・*GTS_PDFXConformance*・*ISO_PDFEVersion*** 以外の任意の名前 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いてはいけません）。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。
encryption	（キーワード。masterpassword が与えられている場合にのみ意味を持ちます。update=true の場合には不可）出力文書のために使用させたい暗号化アルゴリズム。使えるキーワード（デフォルト：PDF 1.7ext8 以上の入力の場合には algo11、そうでないなら algo4）： ***algo4*** Acrobat 7/8 に従った AES-128、すなわち pCOS アルゴリズム 4 を使用して暗号化。これは必要に応じて出力 PDF バージョンを PDF 1.6 へ押し上げます。パスワードは Latin-1 キャラクタ群のみを内容とすることができ、32 キャラクタへ切り詰められます。 ***algo11*** Acrobat X/XI に従った AES-256、すなわち pCOS アルゴリズム 11 を使用して暗号化。これは必要に応じて出力 PDF バージョンを PDF 1.7ext8 へ押し上げます。パスワードは Unicode キャラクタ群を内容とすることができ、127 UTF-8 バイトへ切り詰められます。
input	（PLOP_open_document*( ) を用いて取得された文書ハンドル。必須）処理させたい入力文書
limitcheck	true の場合、PDF/A-1/2/3・PDF/X-4/5 モードにおいて間接 PDF オブジェクト数の上限（8,388,607）が強制されます。デフォルト：true
linearize	（論理値。署名作成または metadata とともに使うことはできません）true 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 引数）と組み合わせることはできません。デフォルト：false
master-password[1]	（文字列。update=true の場合には不可）文書のため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。デフォルト：空

表 7.5 PLOP_create_document( ) に対するオプション

オプション	説明
metadata	（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・PDF/X 識別エントリは、この与える XMP の中では許されません。使えるサブオプション： ***filename*** （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。 ***validate*** （キーワード）与えた XMP メタデータはキーワードによって検証されます（PLOP は入力文書内の XMP メタデータを検証しないことに留意してください）： ***none*** 検証なし ***xmp2004*** XMP 2004 仕様に従って検証 ***xmp2005*** XMP 2005 仕様に従って検証 ***pdfa1*** xmp2004 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-1 に従って拡張スキーマの検証も行います。 ***pdfa2*** xmp2005 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-2・PDF/A-3（両規格はメタデータの必要事項が同じです）に従って拡張スキーマの検証も行います。 デフォルト：入力が PDF/A-1 に準拠していてかつ sacrifice オプションに pdfa が含まれていない場合は pdfa1。入力が PDF/A-2 か PDF/A-3 に準拠していてかつ sacrifice オプションに pdfa が含まれていない場合は pdfa2。それ以外なら none
objectstreams	（キーワード。linearize=true の場合と、PDF/A-1・PDF/X-1a/3 モードにおいては、none が強制されます）出力ファイルサイズを劇的に削減する圧縮オブジェクトストリームを生成（デフォルト：all）： ***all*** 文書情報辞書以外のすべての単純オブジェクトを圧縮オブジェクトストリーム内へ書き込み、圧縮相互参照ストリームを生成。 ***none*** 圧縮オブジェクトストリームも圧縮相互参照ストリームも生成しません。 ***xref*** 圧縮相互参照ストリームを生成しますが、それ以外の圧縮オブジェクトストリームを生成しません。
optimize	（キーワード。update=true の場合には無視されます）適用させたい最適化（デフォルト：none）： ***all*** 実装されているすべての最適化を適用。 ***none*** 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。
permissions	（キーワードリスト。masterpassword が必要です。update=true の場合には不可）文書のアクセス権限リスト。キーワード noprint・nomodify・nocopy・noannots・noassemble・noforms・noaccessible・nohiresprint・plainmetadata を任意の数含むことができます（表 5.3（62 ページ）参照）。デフォルト：空

表 7.5 PLOP_create_document( ) に対するオプション

オプション	説明
recordsize	（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）
sacrifice	（キーワードのリスト。update=true の場合には無視されます）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 7.6 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）：
tempdirname	（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
tempfilename	（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 引数は空にしてはいけません。デフォルト：空
user-password[1]	（文字列。masterpassword が必要です。update=true の場合には不可）文書のため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。デフォルト：空

1. AES-256（アルゴリズム 11）の場合には任意の Unicode キャラクタ群を与えることができますが、AES-128（アルゴリズム 4）の場合には Latin-1 キャラクタ群のみを与えることができます。与えたパスワードは、アルゴリズム 11 の場合には 127 UTF-8 バイトに、アルゴリズム 4 の場合には 32 キャラクタに切り詰められます。EBCDIC プラットフォーム上では、パスワードを ebcdic エンコーディングか EBCDIC-UTF-8 で与える必要があります。

表 7.6 PLOP_create_document( ) の sacrifice オプションに対するキーワード

キーワード	説明
encrypted-attachments	（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。 正しいパスワードを与えられていない暗号化ファイル添付を内容として持つ文書は、update=true を用いて署名を行う際には一切処理できません。
fields	（入力トリガ：文書の中に、NeedAppearances=true の、1 個ないし複数の非署名フォームフィールドがある。操作トリガ：署名作成）このキーワードを与えると、すべてのフォームフィールドが削除されます。
pdfa	（入力トリガ：文書が、PDF/A-1・PDF/A-2・PDF/A-3 のうちのいずれかの準拠レベルに準拠している。操作トリガ：オプション visdoc と非互換な視覚化文書を用いた署名作成、または、オプション userpassword・masterpassword・permissions のうちのいずれか）このキーワードを与えると、PDF/A 入力を処理できますが、PDF/A-1 準拠エントリは削除されます。
pdfua	（入力トリガ：文書が PDF/UA-1 に準拠している。操作トリガ：オプション permissions とキーワード noaccessible を用いた、または、field オプションの visdoc サブオプションを用いた署名作成）このキーワードを与えると、もはや PDF/UA に準拠しなくなった出力を生成でき、その PDF/UA 準拠エントリが削除されます。
pdfvt	（入力トリガ：文書が PDF/VT-1 か PDF/VT-2 に準拠している。操作トリガ：pdfx と同じ）このキーワードを与えると、PDF/VT 入力を処理できますが、その PDF/VT・PDF/X 準拠エントリは削除されます。

表 7.6 PLOP_create_document( ) の sacrifice オプションに対するキーワード

キーワード	説明
pdfx	（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：docinfo オプション内の Trapped=Unknown と組み合わせた、または、field オプションの visdoc サブオプションと組み合わせた署名作成、あるいはオプション userpassword・masterpassword・permissions のうちのいずれか）このキーワードを与えると、PDF/X 入力を処理できますが、その PDF/X 準拠エントリは削除されます。文書が PDF/VT-1 か PDF/VT-2 にも準拠している場合には、その PDF/VT 準拠エントリも削除されます。

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アント配列です。JavaScript で COM を使う場合、返されたバリアント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_document( )*** 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。

7.5 電子署名関数

注記 電子署名機能は製品 PLOP DS でのみ利用可能です。

C++ ***int prepare_signature(wstring optlist)***

C# Java ***int prepare_signature(String optlist)***

Perl PHP ***int prepare_signature(string optlist)***

VB ***Function prepare_signature(optlist As String)***

C ***int PLOP_prepare_signature(PLOP *plop, const char *optlist)***

PLOP に対する 1 個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 に従って署名オプション群を指定したオプションリスト：

- 署名用証明書（デジタル ID）のためのオプション：***digitalid*・*password*・*passwordfile***
- 署名の文脈に関する情報を与えるためのオプション：***contactinfo*・*location*・*policy*・*reason***
- タイムスタンピングのためのオプション：***doctimestamp*・*timestamp***
- 署名視覚化のためのオプション：***field***
- 検証情報を与えるためのオプション：***certfile*・*crl*・*crldir*・*crlfile*・*ocsp*・*rootcertdir*・*rootcertfile*・*validate***
- 証明用署名のためのオプション：***certification*・*preventchanges***
- 署名作成の詳細を制御するためのオプション：***conformance*・*engine*・*ltv*・*signature*・*sigtype***
- 署名の詳細を制御するためのオプション：***dss*・*update***

戻り値 エラー時には −1（PHP では 0）、それ以外なら 1。エラーの後には、*PLOP_get_errmsg()* を呼び出して、そのエラーについてもっと詳しく知ることを推奨します。この関数呼び出しは以下の場合には失敗することがあります：

- 署名者の証明書が見つからない、または秘密鍵へアクセスできない。パスワードや PIN が誤っている等が原因です。
- 検証が失敗。有効な OCSP レスポンスまたは CRL を取得できず、その照応する *critical* フラグが設定されている等が原因です。
- *ltv=full* または *validate=full* に対する必要事項を満たせない。

PLOP_prepare_ signature() への呼び出しが失敗した後には、それと同じオプション群を用いて再度呼び出しを行うことは避けることを推奨します。なぜならこれは、誤ったパスワード／ PIN を何度も与えること等によって PKCS#11 トークンを無効にしてしまう場合があるからです。

詳細 この関数を用いて準備した署名オプション群を使用して、*PLOP_create_document()* を用いて任意の数の署名を作成することができます。ここで与えた署名オプション群は、再度 *PLOP_prepare_ signature()* を（異なる署名オプション群かオプション *nosignature* とともに）呼び出すまで、*PLOP_create_document()* を用いて作成されるすべての署名に対して使用されます。

この署名準備オプションリストは、署名が作成される前の予測不能な時点群において、再度処理されることがあります。特に、多数の署名が作成された後に CRL または OCSP レスポンスが期限切れであることが見つかったときは、この署名オプション群は、証明書失効情報をリフレッシュするために再度処理されます。

証明書の形式　いくつかのオプションは、テキストベースの PEM 形式の証明書を受け付けます。EBCDIC プラットフォームでは PEM 証明書は EBCDIC で符号化されている必要があります。

証明書・CRL ファイルの命名規則　いくつかのオプションは、署名または CTL ファイルのためのファイル名としてハッシュ値を要求します。これらのハッシュファイル名は、OpenSSL 1.0.0 以上を用いて作成できます（これより古いバージョンでは別の命名規則を使用しているので作成できません）。OpenSSL は、1 個のディレクトリ内のすべての証明書に対するハッシュ化されたリンクを作成する ***c_rehash*** ユーティリティを含んでいます。OpenSSL はコマンド ***openssl x509 -hash*** と ***openssl crl -hash*** は、1 個の証明書または CRL に対するハッシュを作成します。ハッシュ化されたファイル名を手作業で作成する必要があるときは、以下の規則に従って行えます：

- 証明書の subject フィールドの DER 符号化された形式、または CRL の issuer フィールドの DER 符号化された形式を使用して、その SHA-1 ハッシュを取ります。
- 生成されたハッシュ値の先頭 4 バイトを見て、照応する最初の 8 桁 16 進値をそのファイル名の基部として使用します。
- 「.」（ピリオド）キャラクタ 1 個と 10 進数 0（ゼロ）を追加します。ハッシュ値どうしの間で衝突がある場合には、この数を増やして 0 のかわりにそれを用います。

オブジェクト識別子（OID）の構文　いくつかのオプションは OID を要求します。署名オプション ***timestamp*** のサブオプション ***policy*** 等です。OID は一連の数から成っており、これらの数が空白かピリオドキャラクタ「.」によって区切られています。

表 7.7 PLOP_prepare_signature( ) に対するオプション

オプション	説明
certfile	（文字列。engine=mscapi の場合には不可）完全な証明書チェーンを埋め込むことを検証するために必要となる可能性のある 1 個ないし複数の中間 CA 証明書を PEM 形式で内容として持つファイルの名前。
certification	（キーワード）指定した証明レベルを持った証明用（作成者）署名を作成。none 以外の値は、証明用証明書を作成し、1 個の文書の中の最初の署名に対してのみ用いるべきです（デフォルト：none）： ***none*** 通常の認証署名：文書は証明されません。 ***nochanges*** 証明用署名：何か変更が行われるとこの署名は無効になります。 ***formfilling*** 証明用署名：フォーム記入することと署名を行うこと（Acrobat のメニュー項目を通じてではなく、署名フィールドをクリックすることによって）は許容されます。ページテンプレートから産み出すことによってページを追加する（手作業でページを追加するのではなく）ことも許容されますが、この技法はめったに使われません。その他の変更は署名を無効化します。 ***formsandannotations*** 署名用署名：フォーム記入すること、署名すること、ページ追加すること、コメントを行うこと（すなわち注釈作成・削除・変更）は許容されます。その他の変更は署名を無効化します。

表 7.7 PLOP_prepare_signature( ) に対するオプション

オプション	説明
conformance	（キーワード）生成される署名の準拠。使えるキーワード（デフォルト：acrobat）： ***acrobat*** 署名を Acrobat で検証できます（必要な Acrobat バージョンについて表 6.1（77 ページ）を参照）。Acrobat で対応していない機能を署名用証明書が使用している場合にはこの関数呼び出しは失敗します。 ***extended*** 以下の機能のうちのいずれかを使用している署名用証明書をも受け入れます。署名を Acrobat で検証することはできない可能性があります： Brainpool 曲線のいずれかを使用した楕円曲線署名（RFC 5639） P-256/P-384/P-521 以外の NIST-15 曲線のいずれかを使用した楕円曲線署名（RFC 5480）
contactinfo	（テキスト文字列。digitalid に対してのみ意味を持ちます）受信者が署名を検証するためにその署名者に連絡をとることができるようその署名者によって提供される情報（電話番号等）。ただし、これは信頼を確立するためのスケーラブルな解決法としては推奨されません。Acrobat 8/9/X はこの連絡先情報を「署名者」タブの「署名のプロパティ」ダイアログ内に表示します。Acrobat XI はこの連絡先情報を表示しません。
crl	（オプションリストかキーワード。crl=none 以外は digitalid に対してのみ意味を持ちます。engine=mscapi の場合には不可）署名用証明書に対する証明書失効リスト（CRL）を取得して、有効な OCSP レスポンスが得られない場合にはそれを署名か DSS の中へ埋め込みます。使えるサブオプション（デフォルト：{source={ } critical=false}、すなわち、そのデジタル ID の中に CRLdp 拡張があればそれが使用されます）： ***critical*** （論理値）true にすると、署名用証明書に対して有効な CRL が取得できた場合にのみ署名が生成され、それ以外の場合にはエラーが返されて署名は作成されません。このオプションを false にすると、有効な CRL を取得できなかった場合には CRL 埋め込みは無警告で無視されます。デフォルト：true ***filename*** （文字列）署名用証明書に対する CRL を DER 形式で内容とするファイルの名前。この filename オプションを与えると、署名用証明書の中の CRLdp 拡張は無視されます。 ***source*** （ネットワークオプションリスト）署名用証明書に対する CRL 配布点を記述したオプションリスト。プロトコル http・https に対応しています。この source オプションの url サブオプションまたはこの source オプション自体を省略することもでき、その場合にはそのデジタル ID の中の CRL 配布点（CRLdp）拡張が情報源として使用されます。 デジタル ID の中に CRLdp 拡張が存在していない限り、この filename・source オプションのいずれか 1 つを必ず、かついずれか 1 つのみを、与える必要があります。 このオプションを crl=none とすると、たとえ CRLdp 拡張が存在していても CRL をネットワーク経由で取得しません。これはその署名用証明書だけでなく、関与するすべての証明書に対して効力を持ちます。
crldir	（文字列。engine=mscapi の場合には不可）関与する証明書群を検証するために必要となる可能性のある PEM 形式の CRL 群を内容とするディレクトリの名前。そのファイル名については表（122 ページ）を参照。
crlfile	（文字列。engine=mscapi の場合には不可）関与する証明書群を検証するために必要となる可能性のある PEM 形式の 1 個ないし複数の CRL を内容とするファイルの名前。
digitalid	（オプションリスト。認証・証明用署名に対しては必須）表 7.8 に従ったサブオプション群を用いて署名者のデジタル ID を指定。使えるサブオプションは、選択したエンジンによって異なります。
doc-timestamp	（オプションリスト。engine=mscapi の場合には不可）信頼済みタイムスランプ局から文書レベルタイムスタンプを生成（builtin エンジンを使用して）。使えるサブオプション：オプション ***timestamp*** を参照

表 7.7 PLOP_prepare_signature( ) に対するオプション

オプション	説明
dss	（論理値。engine=mscapi の場合には不可）true にすると、証明書群と失効情報を文書セキュリティストア（DSS）内へ埋め込みます（6.3.5 節「証明用署名」（85 ページ）参照）。それ以外にすると、このデータを署名の中へ埋め込みます。埋め込みタイムスタンプと文書タイムスタンプに対する検証情報は、このオプションにかかわらず常に DSS 内へ埋め込まれます。入力文書がすでに DSS を内容として持っている場合には、その既存の DSS の内容に加えてその新規の署名に対する検証情報を含む新しい DSS が作成されます。デフォルト：CAdES ベースの署名と、既存の DSS を持った入力文書に対しては true、それ以外なら false
engine	（キーワード）署名を行うために使用させたい暗号化エンジンを指定（デフォルト：builtin）： ***builtin*** 内蔵暗号化エンジンを使用。デジタル ID を仮想またはディスクファイルから取り寄せる必要があります。 ***mscapi*** （Windows のみ。Windows Vista 以上必須）Microsoft Crypto API を暗号化エンジンとして使用。デジタル ID を証明書ストアかディスクファイルから取り寄せることができます。 ***pkcs#11*** （Windows・Linux・OS X・Solaris のみ）PKCS#11 インタフェースを使用して暗号トークンから証明書を取り寄せます。そのトークンのための照応する PKCS#11 DLL ／共有ライブラリの名前を digitalid オプションの filename サブオプションで与える必要があります。
field	（オプションリスト。digitalid に対してのみ意味を持ちます）署名を保持するフォームフィールドの位置と内容を表 7.9 のサブオプション群に従って指定。デフォルト：不可視署名が作成されます
location	（テキスト文字列。digitalid に対してのみ意味を持ちます）署名が作成される物理的位置またはホスト名
ltv	（キーワード。engine=mscapi の場合には不可）署名済み文書を長期検証（LTV）のために準備するかどうかを指定（デフォルト：try）： ***full*** （validate=full を暗黙に前提）署名済み文書を LTV 対応にするために完全な検証情報を埋め込みます。LTV ステータスを実現するには通常、rootcertdir・rootcertfile オプションのいずれかが必須です。さらなる証明書・失効情報を与えるためのオプション certfile・ocsp・crl も必要になる場合があります。証明書のための必要な証明書または失効情報を取得できないときにはこの呼び出しは失敗します。 ***none*** 検証情報を埋め込みません。署名済み文書はより小さくなりますが、LTV 対応ではなくなります。 ***try*** 入手できる限り多くの検証情報を埋め込みます。入手可能な証明書と失効情報によって、署名済み文書は LTV 対応になることもあればならないこともあります。

表 7.7 PLOP_prepare_signature( ) に対するオプション

オプション	説明
ocsp	（オプションリストかキーワード。engine=mscapi の場合には不可）OCSP 処理を構成。使えるサブオプション（デフォルト：{source={ } critical=false}、すなわち、デジタル ID の中に AIA 拡張があればそれを使用）： ***critical*** （論理値。digitalid に対してのみ意味を持ちます）true にすると、署名用証明書に対する有効な OCSP レスポンスがステータス「有効」を持って返された場合にのみ署名が生成されます。それ以外の場合にはエラーが返され、署名は作成されません。このオプションを false にすると、有効な OCSP レスポンスを取得できなかった場合には OCSP レスポンスの埋め込みは無警告で無視されます。デフォルト：true ***hash*** （キーワード）すべての OCSP リクエスト・レスポンス内で証明書を識別するために使用されるハッシュアルゴリズム。そのアルゴリズムにその OCSP レスポンダが対応している必要があります（デフォルト：sha1）： sha1・sha256・sha384・sha512 のいずれか。 なお、Acrobat XI は sha1 にしか対応していません。 ***nonce*** （論理値）true にすると、ノンス（nonce ＝「number used only once ＝数は 1 回だけ使用される」）拡張がすべての OCSP リクエストの中へ含められ、かつそれと同じ値が OCSP レスポンス内に存在している必要があります。ノンス処理は、反射攻撃を防ぎますが、キャッシングを妨げますので、これに対応していない OCSP レスポンダもあります。デフォルト：true ***source*** （ネットワークオプションリスト。digitalid に対してのみ意味を持ちます）署名用証明書に対する OCSP レスポンスが要求される、そしてその後にその署名か DSS の中へ埋め込まれるサーバを記述したオプションリスト。プロトコル http・https に対応しています。この source オプションの url サブオプションまたはこの source オプション自体を省略することもでき、その場合にはその URL はそのデジタル ID の中の authorityInfoAccess 拡張（AIA）から採られます。 このオプションを ocsp=none とすると、たとえ AIA 拡張が存在していても OCSP レスポンスをネットワーク経由で取得しません。これはその署名用証明書だけでなく、関与するすべての証明書に対して効力を持ちます。
password	（文字列。空でも許容されます。engine=builtin の場合には、password か passwordfile のいずれか 1 つ、かついずれか 1 つのみが必須。その他のエンジンでは代替方式を使用することもできます）デジタル ID に対するパスワード・パスフレーズ・PIN のいずれかを指定。engine=pkcs#11 の場合には、このオプションはその暗号トークンに対する PIN を内容とする必要があります。ただし、その PIN をそのトークン自体で対話的に入力する必要がある場合（キーボード付きのスマートカードリーダ等）を除きます。EBCDIC プラットフォームではこのパスワードは ebcdic エンコーディングと見なされます。
passwordfile	（文字列。engine=builtin の場合には、password か passwordfile のいずれか 1 つ、かついずれか 1 つのみが必須。その他のエンジンでは代替方式を使用することもできます）ファイルの 1 行目（1 個ないし複数の改行キャラクタを除いて）が、そのデジタル ID に対するパスワード・パスフレーズ・PIN として使用されます。EBCDIC プラットフォームではこのパスワードファイルの内容は ebcdic エンコーディングと見なされます。

表 7.7 PLOP_prepare_signature( ) に対するオプション

オプション	説明
policy	（オプションリスト。sigtype=cades の場合にのみ可。reason を指定した場合には不可。PAdES-EPES に対しては必須）署名を検証するために使用させたい署名ポリシー。使えるサブオプション： ***commitmenttype*** （キーワード）指定したポリシーのスコープの範囲内の署名に紐付けられる関与の種別。使えるキーワード（デフォルト：none）： ***approval*** 署名者はそのメッセージの内容を認証した。 ***creation*** 署名者はそのメッセージを作成した（ただし必ずしも認証したとは限らず、送信したとも限らない）。 ***delivery*** 表出を提供する信頼済みサービスプロバイダは、メッセージを、そのメッセージの受信者がアクセスできるローカルストアの中で伝達した。 ***none*** 署名には関与種別を含めない。 ***origin*** 署名者は、そのメッセージを作成し、認証し、送信したことを認知している。 ***receipt*** 署名者は、そのメッセージの内容を受信したことを認知している。 ***sender*** 表出を提供する主体はそのメッセージを送信した（ただし必ずしもこれを作成したとは限らない）。 ***notice*** （テキスト文字列）署名ポリシーの、人に読めるテキスト説明 ***oid*** （文字列。必須）署名ポリシーのオブジェクト ID ***uri*** （文字列）署名ポリシーの URI
prevent-changes	（論理値。certification が none 以外の場合のみ）true にすると、certification オプションを用いて禁止している各種変更（証明用署名を無効にするである各種変更）が Acrobat で防止されます。すなわち、照応するユーザーインタフェース要素が無効にされます。デフォルト：true
reason	（テキスト文字列。digitalid に対してのみ意味を持ちます。policy とともには不可）文書に署名を行う理由
rootcertdir	（文字列。engine=mscapi の場合には不可）証明書チェーンを検証するために必要になる可能性のある PEM 形式の信頼済みルート CA 証明書群を内容とするディレクトリの名前。ファイルの命名規則について「証明書・CRL ファイルの命名規則」（122 ページ）参照。
rootcertfile	（文字列。engine=mscapi の場合には不可）証明書チェーンを検証するために必要になる可能性のある 1 個ないし複数の PEM 形式の信頼済みルート CA 証明書を内容とするファイルの名前。セキュリティ上の理由により、このファイルは searchpath 内で検索されません。
signature	（論理値）false にすると、署名は作成されません。これは、事前に PLOP_prepare_signature( ) を呼び出して署名オプション群を与えておきながらも署名とそれ以外の処理を切り替えたい場合に有用でしょう。デフォルト：true
sigtype	（キーワード。digitalid に対してのみ意味を持ちます。engine=mscapi の場合には不可）署名種別（デフォルト：cms）： ***cms*** ISO 32000-1 と PAdES パート 2（ETSI TS 102 778-2）に従った CMS ベースの署名 ***cades*** CAdES（ETSI TS 101 733）と RFC 5126 に従った CAdES ベースの署名。これは PAdES パート 3・パート 4 については必須です。

表 7.7 PLOP_prepare_signature( ) に対するオプション

オプション	説明
timestamp	（オプションリストかキーワード。engine=mscapi の場合には不可）署名は、信頼済みタイムスタンプ局（TSA）によって作成された埋め込みタイムスタンプを含みます。使えるサブオプション（デフォルト：{source={ } critical= false}、すなわち、そのデジタル ID の中に TimeStamp 拡張があればそれを使用）： ***critical*** （論理値。文書レベルタイムスタンプに対しては true を強制されます）true にすると、有効なタイムスタンプが取得できる場合にのみ署名が生成されます。そうでない場合にはエラーが返されます。このオプションを false にすると、有効なタイムスタンプ応答を取得できない場合にはタイムスタンピングは無警告で無視されます。デフォルト：true ***hash*** （キーワード）タイムスタンプを作成するためのハッシュアルゴリズム。その TSA がそのアルゴリズムに対応している必要があります（デフォルト：sha256）：sha1（非推奨）・sha256・sha384・sha512 のいずれか ***policy*** （文字列）TSA ポリシーの OID。そのポリシーのもとでタイムスタンプを作成します。指定したポリシーにその TSA が対応していない場合にはタイムスタンピングは失敗します。 ***source*** （表 7.10 に従ったネットワークオプションリスト）TSA を記述したオプションリスト。プロトコル http・https に対応しています。 文書レベルタイムスタンプではなく埋め込みタイムスタンプの場合のみ：この source オプションの url サブオプションまたはこの source オプション自体を省略することもでき、その場合にはそのデジタル ID の中の TimeStamp 拡張が使われます。 キーワード none にすると、たとえ署名用証明書の中に TimeStamp 拡張が存在していてもタイムスタンプを埋め込みません。
update	（論理値）true にすると、署名関連データが、1 個ないし複数の増分 PDF 更新セクションとして、元の文書の複製に追加されます。それ以外にすると、PDF オブジェクトヒエラルキーは書き直され、したがって既存の署名群は失われます。埋め込みタイムスタンプと文書タイムスタンプに対する検証情報は、このオプションにかかわらず、常に更新として追加されます。
validate	（キーワード）関与する証明書群の検証を制御（デフォルト：ltv=full の場合には full、それ以外の場合には formal）： ***formal*** 以下のチェックが行われます： クリティカルな拡張フラグ群・キー用途等がチェックされます。 OCSP レスポンスが要求された場合には取得され、ステータス「有効」を持った有効な応答を必須とします。 CRL が要求された場合には取得され、署名用証明書が CRL に照らしてチェックされます。CRL の日付がチェックされます。 ***full*** validate=formal の内容に加えて、証明書チェーンの完全な検証。これは、必要なすべてのルートおよび中間 CA 証明書が入手可能であり、かつ、関与するすべての証明書（ルート証明書と id-pkix-ocsp-nocheck 拡張を持つ OCSP レスポンダを除く）に対する OCSP または CRL 失効情報も入手可能であることを必須とします。

表 7.8 PLOP_prepare_signature( ) の digitalid オプションのサブオプション

オプション	説明
engine=builtin の場合のサブオプション：	
filename	（文字列。必須）PKCS#12 形式のディスクベースまたは仮想デジタル ID ファイルの名前。
engine=pkcs#11 の場合のサブオプション：	
filename	（文字列。必須）スマートカード等暗号トークンに対する PKCS#11 DLL ／共有ライブラリの名前。これは PVF ファイルでなくディスクベースのファイルである必要があります。例：cryptoki.dll

表 7.8 PLOP_prepare_signature( ) の digitalid オプションのサブオプション

オプション	説明
issuer	（文字列）与えたクエリに「issuer」フィールド（PKCS#11 属性 CKA_ISSUER）が一致するデジタル ID を選択。このクエリの形式の説明について後述の subject を参照してください。
label	（文字列）与えた値にデジタル ID のユーザーフレンドリーラベル（PKCS#11 属性 CKA_ LABEL）が一致するデジタル ID を選択。
serial	（文字列）与えた値にシリアル番号（PKCS#11 属性 CKA_SERIAL_NUMBER）が一致するデジタル ID を選択。このシリアル番号は 10 進文字列か 16 進文字列（頭に 0X を付ける）として与える必要があります。
slotid	（正の整数）トークンをインタフェースするスロットの番号。これを用いると、複数のスロットが利用可能な場合にスロットを直接指定できます。
subject	（文字列）与えたクエリに「subject」フィールド（PKCS#11 属性 CKA_ SUBJECT）が一致するデジタル ID を選択。このクエリは、形式 /type0=value0/type1=value1/type2=... である必要があります。キャラクタを \（バックスラッシュ）でエスケープすることもできます。属性の順序は意味を持ちます。トークンが複数のデジタル ID を内容としている場合には、オプション issuer・label・subject を用いて証明書選択が可能です。 例：subject={/C=DE/L=Munich/O=PDFlib GmbH/CN=PDFlib Demo PLOP User 2048}
threadsafe	（論理値）true にすると、PKCS#11 ライブラリはスレッドセーフ操作に対応している必要があり、スレッドセーフモードで初期化されます。PKCS#11 がスレッドセーフ操作に対応していない場合にはこの呼び出しは失敗します。false にすると、PKCS#11 ライブラリはシングルスレッドモードで初期化されます。これはシングルスレッドのアプリケーションに対してのみ許容されます。デフォルト：true
engine=mscapi の場合のサブオプション：	
filename	（文字列。filename・store のいずれかが必須）PKCS#12 形式のディスクベースまたは仮想デジタル ID ファイルの名前。
storelocation	（キーワード）証明書ストアの位置（デフォルト：current_user）： current_service・current_user・current_user_group_policy・local_machine・local_machine_enterprise・local_machine_group_policy・services・ users 以下のストア位置については遠隔で開くことができ、そのためにはそのストア名（オプション store）の頭にそのコンピュータ名を付けます（1 個のバックスラッシュキャラクタで区切って）：local_machine・local_machine_group_policy・services・users。
subject	（文字列。store を指定した場合には必須）与えた文字列を「subject」エントリが内容としているデジタル ID を検索。これは通常そのデジタル ID の「common name」（CN）エントリを保持しています。
store	（文字列。filename・store のいずれかが必須）証明書ストアの名前。例：My・Root・Trust。storelocation=services か storelocation=users の場合には、このストア名の頭にサービスまたはユーザー名を付ける必要があります（1 個のバックスラッシュキャラクタで区切って）。デフォルト：My

表 7.9 PLOP_prepare_signature( ) の field オプションのサブオプション

オプション	説明
fillexisting	（論理値。文書の中に 1 個ないし複数の署名フィールドがあり、かつ name オプションを与えていない場合にのみ意味を持ちます）true にすると、入力文書内の最初の署名フィールドを用いて署名が行われます。false にすると、パターン Signature# に基づいた一意な名前を用いて新規の署名フィールドが作成されます。デフォルト：false

表 7.9 PLOP_prepare_signature( ) の field オプションのサブオプション

オプション	説明
name	（テキスト文字列。末尾をピリオド「.」キャラクタにしてはいけません）既存または新規の署名フィールドの名前。文書がこの名前を持った署名フィールドを内容として持っている場合には、それが署名のために使われ（そして page は無視され）、そうでない場合にはそのフィールドが作成されます。この名前を持ったフィールドがあるが種別が署名でない場合にはエラーが発生します。 デフォルト（すなわち空または名前を指定しない）：署名フィールドがない場合には、名前 Signature1 を持った新規署名フィールドが作成されます。そうでない場合には、フィールド作成はオプション fillexisting によって制御されます。
page	（正の整数。既存の署名フィールドへの記入が行われる場合には無視されます）署名フィールドが作成されるページの番号。先頭ページの番号を 1 とします。デフォルト：1
position	（キーワード 2 個のリスト）フィールド内の視覚化ページの相対位置。この視覚化ページは、与えたキーワード群に従ってその長方形内に配置され、その縦横比を保ちながらその長方形内に完全にはめ込まれるように拡縮されます。1 個目のキーワードは横位置を、値 left・center・right のいずれか 1 つを用いて指定します。2 個目のキーワードは縦位置を、値 top・center・bottom のいずれか 1 つを用いて指定します。両方の値が同じの場合には、1 個のキーワードを指定すれば足ります。デフォルト：{center}
rect	（長方形）署名フィールドの左下隅と右上隅の座標を PDF 座標（1 単位は 1/72 インチで左下隅を原点とする）で表したもの。この指定した長方形が視覚化ページで完全に満たされます。変倍を防ぐために 1 個または 2 個の座標のかわりにキーワード adapt を与えることもできます。この場合には、変倍を避けるために足りない座標（群）が自動的に算出されます。少なくとも 1 つの隅を明示的に指定する必要があります。この長方形はそのページをはみ出してはいけません。はめ込み処理について詳しくは「署名フィールドの位置と寸法」（79 ページ）を参照してください。4 個のゼロ値を用いた空の長方形は不可視フィールドになります。 デフォルト：既存のフィールドが使われる場合にはその長方形がデフォルトとなります。そうでない場合には空の長方形（すなわち不可視署名）。
tooltip	（空でないテキスト文字列）可視署名フィールドのツールチップ（代替テキストともいいます）のテキスト。これはスクリーンリーダによってアクセシビリティを向上させるために利用される可能性があります。デフォルト：なし
visdoc	（PLOP_open_document( ) を用いて取得した文書ハンドル。PDF/UA・PDF/X・PDF/VT モードでは不可。空でないフィールド長方形に対してのみ可であり、かつこの場合には必須）ページ上の署名を視覚化するために使用するページを含む文書。PDF/A モードでは、この視覚化文書は、生成される出力に互換である必要があります（「PDF/A 準拠」（81 ページ）参照）。
vispage	（整数。visdoc を与えた場合にのみ意味を持ちます）署名を視覚化するために使用するページのその文書内の番号。デフォルト：1

ネットワークオプションリスト　TSA や OCSP レスポンダ等ネットワークリソースへのアクセスを必要とする機能は多種あります。サーバと、場合によってはそこへアクセスするための詳細については、表 7.10 のサブオプション群に従ったネットワークオプションリストの中で指定することができます。データ型「ネットワークオプションリスト」を使用するオプションはそれぞれ、対応するプロトコルのリストを指定しています。ネットワークオプションリストの使用例をいくつか挙げます（ネットワークオプションリストの部分を青で示しています）：

```
timestamp={source={url={http://timestamp.acme.com/}} hash=sha384} digitalid=...
ocsp={source={url={http://ocsp.acme.com/}} } digitalid=...
ocsp={source={timeout=1000}} digitalid=...
```

表 7.10 ネットワークオプションリストに対するサブオプション

オプション	説明
httpauthentication	（キーワード。http に対してのみ）試行させたい認証方式（群）。ある特定の認証方式に（ないしはどの方式にも）サーバが対応していない場合もあります。パフォーマンス上の利点の観点から、認証種別をデフォルトでなく明示的に設定するほうが望ましい場合があります。使えるキーワード（デフォルト：any）： ***any*** サーバが対応している最もセキュアな認証方式を選択。 ***anysafe*** any と同じ内容に加えて、ベーシック認証を除外。 ***basic*** ユーザー名とパスワードを用いたベーシック認証。この方式は、ユーザー名とパスワードがプレーンテキストのままネットワークへ送信されるので推奨されません。 ***digest*** RFC 2617 に基づいてハッシュ化されたユーザー名とパスワードを用いたダイジェスト認証。これはベーシック認証よりもセキュアです。 ***ntlm*** Microsoft 製品群で使用されているような NTLM 認証
password	（文字列）ベーシック・ダイジェスト認証のためのパスワード
sslcertdir	（文字列。https に対してのみ）SSL 接続を確立するために必要となる可能性のある PEM 形式の信頼済み CA 証明書群を内容とするディレクトリの名前。ファイル命名規則について「証明書・CRL ファイルの命名規則」（122 ページ）を参照してください。
sslcertfile	（文字列。https に対してのみ）SSL 接続を確立するために必要となる可能性のある PEM 形式の 1 個ないし複数の信頼済み CA 証明書を内容とするファイルの名前。
sslverifyhost	（論理値。https に対してのみ）true にすると、接続を確立するためにサーバ証明書内の Subject Alternate Name フィールドが URL 内のホスト名と一致することが必須になります。デフォルト：true
sslverifypeer	（論理値。https に対してのみ）true にすると、sslcertdir または sslcertfile オプションを用いて与えた信頼済み証明書の集合に対してサーバ証明書が検証可能であることが必須となります。false にすると、サーバ証明書のための既知の信頼済みルートが一切得られないために検証ができないサーバ証明書であっても受け入れられます。デフォルト：true
timeout	（整数）リソースへアクセスする際のタイムアウトをミリ秒単位で指定。値 0 とするとタイムアウトなしになります。デフォルト：15000
url	（文字列。通常は必須ですが、URL が文脈からわかる場合にはオプショナル）頭のプロトコル識別子を含めて完全に修飾されたネットワークリソースの URL。使えるプロトコルの集合は、ネットワークオプションリストを使用する各オプションの説明で指定されています。キャラクタを %20 等 URL エンコーディングで指定することもできます。この URL はユーザー名とパスワードを標準構文で含むこともできます。例： `http://user:password@timestamp.acme.com/`
username	（文字列）ベーシック・ダイジェスト認証のためのユーザー名

7.6 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）。言語ラッパは、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++ ***wstring 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++ ***wstring 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 「エラー処理」（39 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 「エラー処理」（39 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 「エラー処理」（39 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 「エラー処理」（39 ページ）参照。

7.7 オプション処理

C++ ***void set_option(wstring 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 7.11 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 7.11 で特記してあるオプションについては、この関数を複数回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 7.11 PLOP_set_option( ) に対するグローバルオプション

オプション	説明
filename-handling	（キーワード）ファイル名のエンコーディングを示します。Unicode 非対応言語バインディングにおいて UTF-8 BOM なしで関数引数として与えられたファイル名は、そのファイルシステムにおいて違法となるであろうキャラクタから保護するため、かつそのファイル名の Unicode 版を作成するために、このオプションに従って解釈されます。ファイル名が、ここで指定したエンコーディングの範囲外のキャラクタを含んでいる場合には、エラーが発生します。デフォルト：Windows・OS X では unicode、それ以外では honorlang： ***ascii*** 7 ビット ASCII ***basicebcdic*** コードページ 1047 に従った基本 EBCDIC、ただし Unicode 値 <= U+007E のみ ***basicebcdic_37*** コードページ 0037 に従った基本 EBCDIC、ただし Unicode 値 <= U+007E のみ ***honorlang*** 環境変数 LC_ALL・LC_CTYPE・LANG が解釈されます。LANG で指定されているコード集合がある場合にはそれがファイル名に適用されます。 ***legacy*** auto 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 honorlang パラメータが設定されている場合には LANG 変数を解釈します。 ***unicode*** （EBCDIC-）UTF-8 形式の Unicode エンコーディング すべての ***8*** ビット・***CJK*** エンコーディングの名前 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
license	（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。
licensefile	（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。
frontpage	（論理値）false 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。true の場合、0.1 節「ソフトウェアをインストール」（7 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、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。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：true

表 7.11 PLOP_set_option( ) に対するグローバルオプション

オプション	説明
logging[1]	（オプションリスト）表 7.12 に従ってログ記録出力を指定したオプションリスト。あるいはログ記録オプション群を。PLOPLOGGING という環境変数で、または Windows ではレジストリを通じて与えることも可能です。空のオプションリストにすると、以前の呼び出しで設定したオプション群を用いたログ記録を有効になります。環境変数が設定されている場合、ログ記録は、PLOP_new( ) への最初の呼び出しの直後に開始されます。
searchpath[1]	（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、エントリは蓄積されて、設定された順に使用されます。空白キャラクタを含むディレクトリ名による問題を避けるために、エントリが 1 個だけであっても二重中括弧を用いることを推奨します。空文字列（すなわち {{}}）を指定すると、デフォルトエントリを含む既存の検索パスエントリがすべて削除されます。Windows の場合、searchpath はレジストリエントリで設定することも可能です。デフォルト：空

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。

表 7.12 PLOP_set_option( ) の logging オプションに対するサブオプション

キー	説明
disable	（論理値）ログ記録出力を無効にします。デフォルト：false
filename	（文字列）ログファイルの名前（stdout・stderr も受け付けます）。出力は既存の内容に追加されます。このログファイル名を、PLOPLOGFILENAME という環境変数で与えることもできます（その場合にはこのオプション filename は常に無視されます）。デフォルト：plop.log（Windows・OS X では / ディレクトリ内、Unix では /tmp 内）
flush	（論理値）true にすると、ログファイルは出力ごとに閉じられ、次の出力の際に再度開かれることにより、出力が必ず実際に掃き出されるようにします。これは、プログラムのクラッシュを追跡する際にログファイルがどこで切れるかを見るために有用でしょう。ただし処理がかなり遅くなります。false にすると、ログファイルは 1 回だけ開かれます。デフォルト：false
remove	（論理値）true にすると、新たな出力を書き込む前に既存のログファイルが削除されます。デフォルト：false
restore	（論理値）すべてのログ記録分類レベル（この同じオプションリストで指定しているものを除き）の状態を、最も過去に保存された状態へ復帰させます。
save	（論理値）すべてのログ記録分類レベル（この同じオプションリストで指定しているものを除き）の状態を保存します。7 個までの保存レベルに対応しています。
stringlimit	（整数）テキスト文字列内のキャラクタ数の上限。0 とすると無制限になります。デフォルト：0

表 7.12 PLOP_set_option( ) の logging オプションに対するサブオプション

キー	説明
classes	（オプションリスト）オプションごとに 1 個のログ記録分類を記述し、照応する値がその粒度レベルを記述するオプションリスト。レベル 0 はそのログ記録クラスを無効にし、正の数はそのクラスを有効にします。レベルを大きくするほど出力が詳細になります。解説中にレベルの言及がないクラスについては値 1 を用いる必要があります（初期値：api=1）。 ***api*** すべての API 呼び出しを、その引数と戻り値とともにログ記録します。api=2 とすると、すべての API トレース行の頭にタイムスタンプが作成され、非推奨の関数とオプションが標識されます。api=3 とすると、try/catch 呼び出しがログ記録されます（ネストされた例外処理を伴う問題をデバッグするために有用です）。 ***digsig*** 電子署名作成に関する詳細をログ記録します： ***1*** 基本的情報 ***2*** 検証情報。OCSP・CRL の詳細。PKCS#11 ライブラリ・スロット・トークン情報 ***5*** 証明書の詳細 ***filesearch*** SearchPath または PVF を通じたファイル検索関連のすべての試行をログ記録します。 ***resource*** Windows レジストリ・UPR 定義を通じたリソース検索のすべての試行を、そのリソース検索の結果とともにログ記録します。 ***user*** userlog を用いて与えられるユーザー指定のログ出力。 ***warning*** すべての警告、すなわち、無視または内部的に修復できるエラー状態をログ記録します。warning=2 とすると、例外を発生させずに PLOP_get_errmsg( ) を通じて取得できるメッセージテキストを残す関数からのメッセージと、ファイルを開く試行（searchpath 内でファイルを検索）のすべての失敗の理由もログ記録されます。

7.8 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, wstring 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引数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、与える追加引数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, wstring 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・数値・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引数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、与える追加引数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。ただし、***/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、*pages[...]/annots[...]/* で始まるすべてのパスは、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改されません。

バインディング C 言語バインディング：文字列は BOM なしの UTF-8 形式で返されます。返される文字列は、最大 10 エントリ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++ 言語バインディング：文字列は、C++ ラッパのデフォルト *wstring* 構成における *wstring* として返されます。zSeries の *string* 互換モードでは、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す。

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・Python・Ruby 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。

C++ ***const unsigned char *pcos_get_stream(int doc, int *length, string optlist, wstring 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 7.13 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引数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、与える追加引数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、あるいは、暗号化されていない文書の中の暗号化された添付の内容がク

エリされてその添付パスワードが与えられていないときは、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アント型を用いています。JavaScript で COM を使う場合、返されたバリアント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 7.13 PLOP_pcos_get_stream( ) に対するオプション

オプション	説明
convert	（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：none）： ***none*** 内容をバイナリデータとして扱い、一切変換しません。 ***unicode*** 内容をテキストデータとして（すなわち PLOP_pcos_get_string( ) 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。

7.9 Unicode 変換関数

C++ ***string convert_to_unicode(wstring inputformat, string input, wstring 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 ではバリアント）

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 7.14 に従った入力フィルタオプション群：***charref*・*escapesequence***
- 表 7.14 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 エントリ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 7.14 PLOP_convert_to_unicode( ) に対するオプション

オプション	説明
bom	（キーワード。outputformat=utf32 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：none）： ***add*** BOM を加えます。 ***keep*** 入力文字列に BOM があるなら BOM を加えます。 ***none*** BOM を加えません。 ***optimize*** outputformat=utf8 または ebcdicutf8 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。
charref	（論理値）true 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：false
errorpolicy	（キーワード）変換エラーの場合の動作（デフォルト：exception）： ***return*** 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。 ***exception*** 変換エラーの場合に例外が発生します。
escape-sequence	（論理値）true 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：false
inflate	（論理値。inputformat=utf8 の場合のみ。outputformat=utf8 の場合には無視されます）true 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：false
output-format	（キーワード）生成される文字列の Unicode テキスト形式：utf8・ebcdicutf8・utf16・utf16le・utf16be・utf32。空文字列は utf16 と同等です。デフォルト：utf16 Unicode 対応言語バインディング：出力形式は utf16 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：utf8・utf16・utf32。

A PDFlib を PLOP DS と結合

PLOP DS は、PDF 文書を動的に生成してそれに署名を行うために、PDFlib と容易に相互動作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、非常に大きな文書を扱うのでない限り、通常は PLOP DS ライブラリを使ってそうするほうが効率的です。

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 DS 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、その生成された文書を *PLOP_open_ document()* で処理します。

文書をメモリ内に作成して電子的に署名　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 生成を利用します。
- 生成された PDF データの入ったバッファの内容を、*PDF_end_document()* の後に *PDF_get_buffer()* を呼び出すことによって取り出します。
- この PDF データに基づいて、*PLOP_create_pvf()* を呼び出すことによって PLOP 内に仮想ファイルを作成します。
- この PVF ファイルの名前を、*PLOP_open_document()* を用いて PLOP DS へ渡します。

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。

注記 PDFlib 7/8/9 はフォームフィールドに対する視覚ストリームを作成しませんので、PDFlib で生成された、非署名フォームフィールド群を内容として持つ文書については、`sacrifice` オプションを用いてそのフィールド群を除去した場合にのみ、PLOP DS を用いてその文書に署名を行うことができます。

署名視覚化文書を動的に作成　署名視覚化のために用いる文書を、PDFlib を用いて動的に作成することもできます（6.3.1 節「署名をグラフィックかロゴで視覚化」（79 ページ）参照）。これは、視覚化文書の中に現在日時等可変のテキストまたは画像構成要素を含める必要がある場合に有用でしょう。*dynamicsign* の基本構造は *hellosign* サンプルとよく似ています。

dynamicsign プログラミングサンプルは、PDFlib を用いて PDF 視覚化文書を動的に作成し、それを署名作成処理で使用するためにメモリ内で PLOP に渡す方法を演示しています。

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

文書入力・出力

エラー処理

オプション処理

pCOS 関数

Unicode 変換関数

関数プロトタイプ	ページ
string convert_to_unicode(string inputformat, byte[ ] input, string optlist)	*139*

C 変更履歴

このマニュアルの変更履歴

日付	変更
2015 年 02 月 27 日	► PLOP 5.0・PLOP DS 5.0 に関する変更
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

R

S

T

W

X

Z

あ

い

え

へ

ほ

ま

む

ゆ

ら

れ

PDFlib GmbH

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、PDF メーリングリストと、
groups.yahoo.com/neo/groups/pdflib/info 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

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