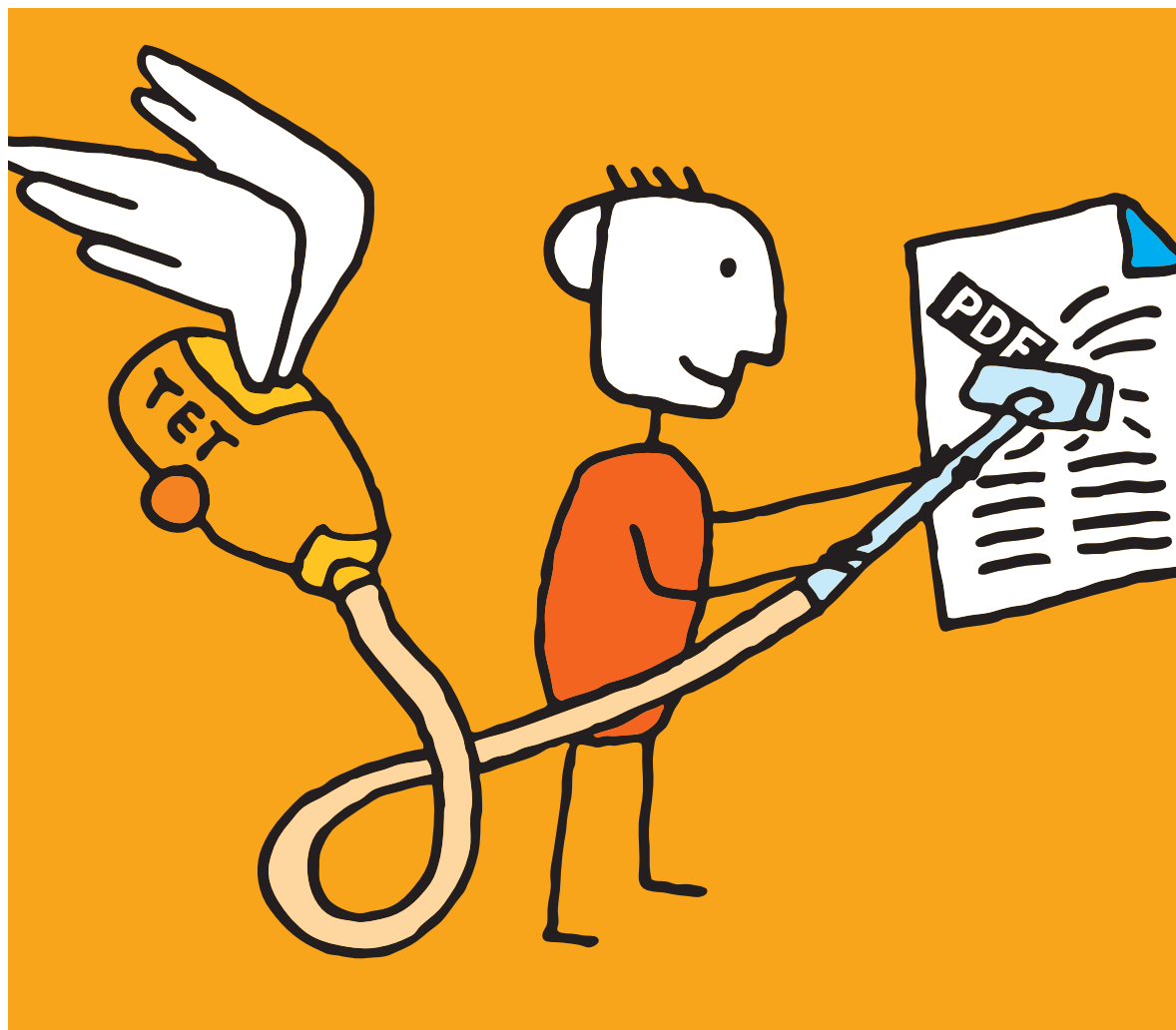


# Text Extraction Toolkit (TET)

Version 4.1

PDF 文書からテキスト・画像・メタデータを抽出するためのツールキット



Copyright © 1997-2012 PDFlib GmbH. All rights reserved.  
Protected by European and U.S. patents.

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

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

ライセンスご希望の際の連絡先 : [jp.sales@pdflib.com](mailto:jp.sales@pdflib.com)  
商用 PDFlib ライセンス保持者向けサポート : [jp.support@pdflib.com](mailto:jp.support@pdflib.com) (お使いのライセンス番号をお書きく  
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わず、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) 保証も行わず、そして、いかなるそしてすべての売買可能性の保証と、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・Windows・  
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。

TET は以下のサードパーティソフトウェアの変更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  
TIFFlib 画像ライブラリ、Copyright © 1988-1997 Sam Leffler, Copyright © 1991-1997 Silicon Graphics, Inc.  
Eric Young の書いた Cryptographic ソフトウェア、Copyright © 1995-1998 Eric Young ([ey@cryptsoft.com](mailto:ey@cryptsoft.com))  
Independent JPEG Group の JPEG ソフトウェア、Copyright © 1991-1998, Thomas G. Lane  
Cryptographic ソフトウェア、Copyright © 1998-2002 The OpenSSL Project ([www.openssl.org](http://www.openssl.org))  
Expat XML パーサ、Copyright © 1998, 1999, 2000 Thai Open Source Software Center Ltd  
ICU International Components for Unicode、Copyright © 1995-2009 International Business Machines  
Corporation and others

TET 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

## ○ TET の第一歩 7

o.1 ソフトウェアをインストール 7

o.2 TET ライセンスキーを適用 9

## 1 はじめに 13

1.1 TET 機能概要 13

1.2 TET のさまざまな使用法 15

1.3 文書とサンプルへのロードマップ 16

1.4 TET 4.0 の新機能 17

1.5 TET 4.1 の新機能 18

## 2 TET コマンドラインツール 19

2.1 コマンドラインオプション 19

2.2 TET コマンドラインを構築 22

2.3 コマンドラインの例 24

2.3.1 テキストを抽出 24

2.3.2 画像を抽出 24

2.3.3 TETML を生成 25

2.3.4 高度なオプション 25

## 3 TET ライブラリの言語バインディング 27

3.1 例外処理 27

3.2 C バインディング 29

3.3 C++ バインディング 32

3.4 COM バインディング 35

3.5 Java バインディング 36

3.6 .NET バインディング 38

3.7 Objective-C バインディング 39

3.8 Perl バインディング 41

3.9 PHP バインディング 42

3.10 Python バインディング 44

3.11 REALbasic バインディング 45

3.12 Ruby バインディング 46

3.13 RPG バインディング 48

## 4 TET コネクタ 51

4.1 Adobe Acrobat 用無償 TET Plugin 51

4.2 Lucene 検索エンジン用 TET コネクタ 53

4.3 Solr 検索サーバ用 TET コネクタ 56

4.4 Oracle 用 TET コネクタ 57

4.5 Microsoft 製品用 TET PDF IFilter 60

4.6 Apache TIKA ツールキット用 TET コネクタ 63

4.7 MediaWiki 用 TET コネクタ 65

## 5 設定 67

5.1 暗号化 PDF から内容を抽出 67

5.2 リソース設定とファイル検索 69

5.3 代表的シナリオのための推奨方策 73

## 6 テキスト抽出 77

6.1 PDF のさまざまな文書領域 77

6.2 ページとテキストの視覚情報 81

6.3 日本語・中国語・韓国語テキスト 87

6.3.1 日中韓エンコーディング・CMap 87

6.3.2 日中韓テキストの単語境界 87

6.3.3 縦書き 87

6.3.4 日中韓分解：narrow・wide・vertical 等 88

6.4 双方向アラビア文字・ヘブライ文字テキスト 90

6.4.1 双方向の一般的性質 90

6.4.2 アラビア文字テキストを後処理 90

6.5 内容分析 92

6.6 レイアウト分析 96

## 7 高度な Unicode 処理 99

7.1 Unicode のさまざまな重要概念 99

7.2 Unicode 前処理（フィルタリング）102

7.2.1 すべての粒度のためのフィルタ 102

7.2.2 粒度 word 以上のためのフィルタ 103

7.3	Unicode 後処理	105
7.3.1	Unicode 字形統合	105
7.3.2	Unicode 分解	108
7.3.3	Unicode 正規化	112
7.4	追加キャラクタとサロゲート	114
7.5	グリフに対する Unicode マッピング	115
<b>8</b>	<b>画像抽出</b>	<b>121</b>
8.1	画像抽出の基本	121
8.2	画像の連結とフィルタリング	123
8.3	配置画像と画像リソース	125
8.4	ページベースとリソースベースの画像ループ	126
8.5	配置画像の視覚情報	127
8.6	制約と注意	129
<b>9</b>	<b>TET マークアップ言語 (TETML)</b>	<b>131</b>
9.1	TETML を生成	131
9.2	TETML の詳細を制御	135
9.3	TETML のさまざまな要素と TETML スキーマ	139
9.4	TETML を XSLT で変換	143
9.5	さまざまな XSLT サンプル	146
<b>10</b>	<b>TET ライブラリ API リファレンス</b>	<b>151</b>
10.1	オプションリスト	151
10.2	オプションリスト文法	151
10.3	基本型	154
10.4	図形型	157
10.5	一般関数	158
10.5.1	オプション処理	158
10.5.2	セットアップ	161
10.5.3	PDFlib 仮想ファイルシステム (PVF)	162
10.5.4	Unicode 変換関数	165
10.5.5	例外処理	167
10.5.6	ログ記録	169
10.6	文書関数	171
10.7	ページ関数	178

- 10.8 テキスト・メトリクス抽出関数 187
- 10.9 画像抽出関数 192
- 10.10 TET マークアップ言語 (TETML) 関数 196
- 10.11 pCOS 関数 199

## A TET ライブラリクイックリファレンス 203

## B 更新履歴 205

## 索引 207

# ○ TET の第一歩

## 0.1 ソフトウェアをインストール

TET は、Windows システム群に対しては MSI インストーラの形で頒布され、それ以外のすべての対応オペレーティングシステムに対しては圧縮アーカイブの形で頒布されます。すべての TET パッケージの中には TET コマンドラインツールと TET ライブラリ / コンポーネント、およびサポートファイル群・説明書・使用例群が含まれます。TET をインストールまたは解凍した後は、以下の手順を推奨します。

- ▶ TET コマンドラインツールのユーザはその実行形式をただちに利用可能です。指定可能なオプションは 19 ページの 2.1「コマンドラインオプション」で解説しますが、TET コマンドラインツールをオプションなしで実行したときにも表示されます。
- ▶ TET ライブラリ / コンポーネントのユーザは 27 ページの 3 章「TET ライブラリの言語バインディング」の中の、使いたい開発環境に対応する節を読み、インストールされた使用例を眺めるとよいでしょう。Windows の場合、TET プログラミング例はスタートメニューから (COM と .NET の場合)、あるいはインストールディレクトリから (その他の言語バインディングの場合) 開くことができます。

商用 TET ライセンスを得た場合には、自分の TET ライセンスキーを 9 ページの 0.2「TET ライセンスキーを適用」に従って入力する必要があります。

**日中韓設定** レガシエンコーディングで符号化された日本語・中国語・韓国語 (日中韓) テキストを抽出するためには TET はそれに対応する日中韓エンコーディングを Unicode にマップする CMap ファイルを必要とします。この CMap ファイル群はすべての TET パッケージに含まれており、TET のインストールディレクトリ内の *resource/cmap* ディレクトリにインストールされています。Windows システム群では、TET インストール時にフルインストールのオプションを選ぶだけで OK です。この CMap ファイル群はレジストリを介して自動的に検索されます。

それ以外のシステムでは、この CMap ファイル群を以下のように手動で設定する必要があります。

- ▶ TET コマンドラインツールに対しては、この CMap ファイル群があるディレクトリの名前を `--searchpath` オプションで与えます。
- ▶ TET ライブラリ / コンポーネントに対しては、実行時に次のように *searchpath* を設定します。

```
set_option("searchpath=/CMap/リソース/への/パス");
```

また別の方法として、この日中韓 CMap ファイル群へのアクセスを設定するには、適切な *searchpath* 定義を書いた UPR 設定ファイルの場所を *TETRESOURCEFILE* 環境変数に設定するという方法もあります。

**IBM i5/iSeries でのグリフリスト設定** IBM i5/iSeries 上では、ディレクトリ *resource/glyphlst* 内のグリフリストが TET から利用可能である必要があります。これらのテーブルは、TET が標準ディレクトリ内にインストールされた場合は自動的に利用可能に設定されます。

**評価版の制約事項** TET 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とも完全機能の評価版として使うことができます。非ライセンス版はいずれもすべての機能に対応していますが、最大 10 ページ・1 MB 容量までの PDF 文書だけを処理できるようになっています。TET の評価版はいずれも製品評価の目的にのみ使用が許されていますので、実用目的には使用しないで下さい。TET を実用目的に使用するには有効な TET ライセンスが必要です。



## 0.2 TET ライセンスキーを適用

TET を実用目的に使用するには有効な TET ライセンスキーが必要です。TET ライセンスを購入されたら、そのライセンスキーを適用することによって、任意の大きさの文書进行处理できるようにする必要があります。ライセンスキーの適用方法は数通りありますので、以下に述べる方式のいずれかを選択して下さい。

**注記** TET ライセンスキーはプラットフォーム依存なので、それぞれ購入対象となったプラットフォーム上でのみ使用することができます。

**Windows インストーラ** Windows インストーラで作業をしている場合は、製品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。インストーラはライセンスキーをレジストリに追加します（後述）。

**ライセンスファイルでの作業** PDFlib はライセンスキー群をライセンスファイルから読み取ります。ライセンスファイルとは、後述の書式に従ったテキストファイルです。すべての TET ディストリビューションに入っているテンプレート *licensekeys.txt* を使うこともできます。「#」キャラクタで始まる行はコメントであり無視されます。2 行目はライセンスファイル自体のバージョン情報を内容として持ちます：

```
# Licensing information for PDFlib GmbH products
PDFlib license file 1.0
TET 4.1 ...あなたのライセンスキー ...
```

ライセンスファイルは、複数の PDFlib GmbH 製品に対するライセンスキーを、別々の行に内容として持つことが可能です。また、複数のプラットフォームに対するライセンスキーを内容として持つこともできますので、一つのライセンスキーを複数プラットフォームで共用することが可能です。ライセンスファイルは以下の方法で設定できます：

- ▶ *licensekeys.txt* というファイルが、すべてのデフォルトディレクトリ内で検索されます（10 ページの「デフォルトファイル検索パス」参照）。
- ▶ *set\_option()* API 関数で *licensefile* オプションを指定することもできます：

```
tet.set_option("licensefile", "/path/to/licensekeys.txt");
```

この *licensefile* オプションは、TET オブジェクトをインスタンス化した直後に、すなわち *TET\_new()* を呼び出した後に（C の場合）、あるいは TET オブジェクトを生成した後に設定する必要があります。

- ▶ TET コマンドラインツールの *--teto* オプションを与え、*licensefile* オプションでライセンスファイルの名前を与えます：

```
tet --teto "licensefile /path/to/your/licensekeys.txt" ...
```

パス名が空白キャラクタを含む場合には、パスを中括弧で囲む必要があります：

```
tet --teto "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使って、「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"
```

IBM i5/iSeries では、ライセンスファイルは下記のように指定できます（このコマンドは、スタートアッププログラム *QSTRUP* 内で指定することができ、すべての PDFlib GmbH 製品に対して効力を持ちます）：

```
ADDENVVAR ENVVAR(PDFLIBLICENSEFILE) VALUE(<... パス ...>) LEVEL(*SYS)
```

**レジストリ内にライセンスキー** Windows では、ライセンスファイルの名前を下記のレジストリキーに記入することもできます：

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

あるいは、ライセンスキーを直接下記のレジストリキーに記入することもできます：

```
HKLM\SOFTWARE\PDFlib\TET4\license  
HKLM\SOFTWARE\PDFlib\TET4\4.1\license
```

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

**注記** 64ビット Windows システム上で手作業でレジストリを操作する際には注意が必要です。通常、64ビット PDFlib バイナリは Windows レジストリの 64ビットビューとともに動作するのに対して、64ビットシステム上で走る 32ビット PDFlib バイナリはレジストリの 32ビットビューとともに動作します。32ビット製品に対するレジストリキーを手作業で追加する必要がある場合には、必ず、*regedit* ツールの 32ビットバージョンを使用してください。これは「スタート」→「ファイル名を指定して実行...」ダイアログから下記のように呼び出すことができます：

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

**デフォルトファイル検索パス** Unix・Linux・Mac OS X・i5/iSeries システム上では、パスやディレクトリ名を一切指定していない場合でも、さまざまなファイルがいくつかのディレクトリでデフォルトで検索されます。UPR ファイル（その中にさらなる検索パス群を含めておくこともできます）を検索して読み取る前に、下記のディレクトリが検索されます：

```
<rootpath>/PDFlib/TET/4.1/resource/cmap  
<rootpath>/PDFlib/TET/4.1/resource/codelist  
<rootpath>/PDFlib/TET/4.1/resource/glyphlst  
<rootpath>/PDFlib/TET/4.1  
<rootpath>/PDFlib/TET  
<rootpath>/PDFlib
```

Unix・Linux・Mac OS X の場合、*<rootpath>* は、まず */usr/local* で置き換えられ、ついで HOME ディレクトリで置き換えられます。i5/iSeries の場合、*<rootpath>* は空です。

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

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

この機能を利用すると、環境変数やランタイムオプションを一切指定しなくてもライセンスファイルを扱うことができます。

**TET コマンドラインツールに対するオプション内でライセンスキーを設定** TET コマンドラインツールを使用する場合には、ライセンスファイルまたはライセンスキー自体の名前を内容として持つオプションを与えることもできます：

```
tet --teto "license ...あなたのライセンスキー ..." ...さらなるオプション群...
```

**TET API 呼び出しでライセンスキーを設定** TET API を使用する場合は、スクリプトやプログラムに以下のような API 呼び出しを加えて、実行時にライセンスキーを設定させることもできます：

▶ COM/VBScript の場合：

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

▶ C の場合：

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

▶ C++・.NET/C#・Java・Ruby の場合：

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

▶ Perl・Python・PHP の場合：

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

▶ RPG の場合：

```
d licensekey      s           20
d licenseval      s           50
c                  eval       licenseopt='license=... あなたのライセンスキー ...'+x'00'
c                  callp      TET_set_option(TET:licenseopt:0)
```

この *license* オプションは、TET オブジェクトを実体化させた直後に設定する必要があります。これは具体的には、*TET\_new()* を呼び出した直後 (C の場合)、または TET オブジェクトを生成させた直後ということになります。

**i5/iSeries・zSeries 上での複数システムライセンスファイル** i5/iSeries・zSeries に対するライセンスキーはシステム固有ですので、複数のシステムで共用することはできません。リソース共有を実現して、複数のシステムで共用できる一つのライセンスファイルを扱うためには、下記のライセンスキー書式を用いて、複数のシステム固有キーを一つのファイル内に保持することができます：

PDFlib license file 2.0

# Licensing information for PDFlib GmbH products

```
TET      4.1      ...あなたのライセンスキー ...      ...マシン1に対するシリアル番号...
TET      4.1      ...あなたのライセンスキー ...      ...マシン2に対するシリアル番号...
```

1 行目のバージョン番号が変わっていることと、複数のライセンスキーが存在していることに留意してください。ライセンスキーの後にはそれぞれ、16 進 8 桁のシリアル番号 (i5/iSeries の場合) か、16 進 4 桁の CPU ID (zSeries の場合) が続きます。

**さまざまなライセンシングオプション** 一つないし複数のコンピュータ上で TET を使用したり、TET 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、サポート契約やソースコード契約も提供しています。ライセンシングの詳細や購入注文フォームは、TET ディストリビューショ

ン内にあります。商用ライセンスの取得にご関心がある場合、または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](http://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：[jp.sales@pdflib.com](mailto:jp.sales@pdflib.com)

PDFlib ライセンス保持者向けサポート：[jp.support@pdflib.com](mailto:jp.support@pdflib.com)

# 1 はじめに

PDFlib Text Extraction Toolkit (TET) は、PDF 文書からテキスト・画像を抽出することを目的としていますが、それ以外の情報を PDF から取り出すために利用することもできます。TET は、以下のような目的を実現するための基盤要素として活用できます：

- ▶ PDF のテキスト内容を検索
- ▶ PDF 内に含まれる全用語の一覧（コンコーダンス）を生成
- ▶ 大量 PDF ファイルを処理可能な検索エンジンを実装
- ▶ PDF からテキスト抽出して保管・翻訳・その他再利用目的に活用
- ▶ PDF のテキスト内容を他形式へ変換
- ▶ PDF をそれぞれ内容にしたがって処理または変更
- ▶ 複数 PDF 文書のテキスト内容を比較
- ▶ PDF からラスタ画像を再利用のため抽出
- ▶ PDF からメタデータ等の情報を抽出

TET は、スタンドアロン使用のために設計されており、いかなるサードパーティソフトウェアをも必要としません。堅牢であり、マルチスレッドのサーバ用途にも適しています。

## 1.1 TET 機能概要

**対応する PDF 入力** TET は、さまざまな作成元による数百万種の PDF テストファイルに対して動作試験済です。PDF 1.0 から PDF 1.7 拡張レベル 8 (ISO 32000 を含む) を受け入れ可能です。これは Acrobat 1 ～ X に対応します。また、暗号化された文書も受け入れ可能です。

**Unicode 対応** TET には、あらゆるテキストに対して信頼性の高い Unicode マッピングを実現するに足る十分な数のアルゴリズムとデータが内蔵されています。PDF 文書内のテキストは Unicode で符号化されているとは限りませんが、TET は PDF 文書から抽出したテキストを Unicode に正規化します：

- ▶ TET はすべてのテキスト内容を Unicode に変換します。C の場合、テキストは UTF-8 形式か UTF-16 形式で返されます。それ以外の言語の場合はネイティブ Unicode 文字列として返されます。
- ▶ 合字などの複数文字グリフは、その構成素を並べた Unicode キャラクタ列へと分解されます。
- ▶ ベンダ依存 Unicode 値 (Corporate Use Subarea, CUS) については識別情報が与えられるとともに、厳密な定義済みの意味を持つキャラクタへ可能な限りマップされます。
- ▶ Unicode マッピング情報を欠いたグリフについてはそのように識別情報が与えられるとともに、ユーザ指定可能な置換キャラクタへマップされます。
- ▶ 基本多言語面 (BMP) 外のキャラクタに対する UTF-16 サロゲートペアは正しく解釈され保持されます。サロゲートペアと UTF-32 値はすべての言語バインディングにおいて抽出可能です。

PDF 文書のなかには、信頼性の高い Unicode マッピングの実現に必要な情報を十分に持たないものがあります。そのような場合でもテキストを正しく抽出できるよう、TET にはさまざまな設定オプションが用意されており、それを用いることによって、正しい Unicode マッピングの実現に必要な補助的情報を与えることができます。必要なマッピングテーブルの作成支援のため、PDFlib FontReporter という Adobe Acrobat 用無料プラグインを提供

いたしております。このプラグインを用いれば、PDF 内のフォント・エンコーディング・グリフの解析が可能です。

**日中韓対応** TET は、日本語・中国語・韓国語テキストの抽出に完全対応しています：

- ▶ あらゆる定義済み日中韓 CMap (エンコーディング) が認識され、日中韓テキストは Unicode に変換されます。日中韓エンコーディング変換のための CMap ファイル群が、TET ディストリビューションに同梱されています。
- ▶ 特殊なキャラクタ字形 (全角・半角・縦中横等) を、対応する通常の字形に変換 (字形統合) することもできます。
- ▶ 横書きと縦書きに対応しています。
- ▶ 日中韓フォント名は Unicode へ正規化されます。

**双方向ヘブライ文字・アラビア文字テキスト対応** TET は、双方向テキストを扱うための以下の機能を含んでいます：

- ▶ 右書き・双方向テキストを論理順に並べ替え
- ▶ ページの主要テキスト方向を決定
- ▶ アラビア文字の表示形を正規化し、合字を分解
- ▶ 単語の引き伸ばしに使われているアラビア文字のタトゥィールキャラクタを除去

**Unicode 後処理** TET の Unicode 後処理機能は以下を含みます：

- ▶ 字形統合：一つないし複数のキャラクタに対して温存・置換・除去のいずれかを行います。対象キャラクタ群を Unicode 集合として指定することができるので便利です。
- ▶ 分解：Unicode 規格で定義されている正準分解または互換分解を適用することも可能です。環境によってはこれによってテキストがより有用になる場合があります。たとえば、アクセント付きキャラクタや分数、商標記号のような記号を温存したり分割したりすることができます。
- ▶ 正規化：出力を、Unicode 規格で定義されている Unicode 正規形 NFC・NFD・NFKC・NFKD のいずれかへ変換します。これによって TET は、データベースや検索エンジンなどいくつかの環境が入力として求める形式と正確に一致した形式を生成することができます。

**画像抽出** TET は PDF からラスタ画像を抽出します。断片化された画像の隣接部分は再結合され、後処理と再利用を可能にします (いくつかのアプリケーションによって生成されたマルチストリップ画像等)。小さい画像は、微小な画像素片で出力が散らかることを防ぐために、フィルタをかけて取り除くこともできます。

画像は、広く用いられている TIFF・JPEG・JPEG 2000 のいずれかの形式で抽出されます。

**位置情報** TET は、テキストのページ上の位置やグリフ幅やテキスト方向といった厳密なメトリクスを出力します。テキスト抽出処理に際してはページ上の特定領域を外したり含めたりすることにより、たとえばヘッダ・フッタや余白を無視させたりすることが可能です。

画像については、ピクセルサイズ・物理サイズ・色空間が得られるほか、位置や角度も得られます。

**単語検出と内容解析** TET には、低次元のグリフ情報を抽出する機能だけでなく、高次元の内容解析を行う高度なアルゴリズムも内蔵されています：

- ▶ 単語区切りを検出して文字群でなく単語を抽出。
- ▶ ハイフンで区切られた単語の各部を再結合 (デハイフネーション)。

- ▶ 影付きや擬似太字テキスト等によるテキストのダブリを除去。
- ▶ 段落群を読み順に再連結。
- ▶ ページ上に配置されたテキスト群の順序認識。
- ▶ テキストの行を再構成。
- ▶ ページ上の表構造を認識。
- ▶ 上付き・下付き・ドロップキャップ（段落頭の大きい先頭キャラクタ群）を認識。

**PDF オブジェクトを容易に指し示すための pCOS インタフェース** TET は、任意の PDF オブジェクトを取得するための pCOS (*PDFlib Comprehensive Object System*) インタフェースを含んでいます。pCOS を使うと、PDF 文書から、PDF メタデータやインタラクティブ要素（しおりテキストやフォームフィールドの内容など）、その他あらゆる情報を、簡単な取得インタフェースで取得することができます。pCOS クエリパスの文法は、別途 pCOS パスリファレンスに記述しています。

**TET Markup Language (TETML)** PDF 文書から取得した情報は、TET Markup Language すなわち TETML という XML 形式で表現することもできます。これは標準的な XML ツール群で処理が可能です。TETML は、テキスト・画像・メタデータ情報を内容として持つほか、オプションとしてフォント・位置関連情報を内容として持つこともできます。

**テキストとは何か** TET はさまざまな種類の PDF 文書を扱いますが、目に見えるテキストはすべてうまく抽出できるとは限りません。そのテキストは PDF のテキスト・エンコーディング機能を用いて符号化されている必要があります（すなわちそれはフォントに基づいたものでなければなりません）。以下の種類のテキストはページ上で目に見えてはいても TET で抽出することはできません：

- ▶ ページのスキャナ画像等、ラスタライズされた（ピクセル画像の）テキスト
- ▶ フォントなしにベクトル要素で直接表されたテキスト

なお、メタデータとハイパーテキスト要素（しおり・フォームフィールド・ノート・注釈等）内テキストは pCOS インタフェースで抽出できます。逆に TET は、ページ上で目に見えていないテキストを抽出することもあります。これは以下のような場合に起こる可能性があります：

- ▶ PDF の不可視属性を用いたテキスト（ただし、この種類のテキストをテキスト抽出処理の対象外にするオプションはあります）。
- ▶ テキストが、ページ上の画像等何か他の要素によって隠されたり切り取られたりしている場合。
- ▶ PDF のレイヤーは無視され、各レイヤーの表示 / 非表示にかかわらず、TET はすべてのレイヤーからテキストを抽出します。

## 1.2 TET のさまざまな使用法

TET は、さまざまな開発環境用のプログラミングライブラリ（コンポーネント）としても提供されますし、バッチ操作のためのコマンドラインツールとしても提供されます。両者の機能は同様ですが、それぞれ実用に適する場面が異なっています。TET ライブラリとコマンドラインツールはどちらも、TET の XML ベースの出力形式である TETML を生成することができます。

どちらの TET を選べばよいか、以下にそのガイドラインを掲げます。

- ▶ TET プログラミングライブラリはデスクトップアプリケーションやサーバアプリケーションに組み込んで使うことができます。さまざまなプログラミング言語に対応して

います。TET パッケージには、TET ライブラリのすべての対応言語バインディングにおける使用例が同梱されています。

- ▶ TETコマンドラインツールはPDF文書のバッチ処理に適しています。プログラミングは一切不要で、かわりにさまざまなコマンドラインオプションをそなえており、それを用いて複雑なワークフローでの活用が可能です。
- ▶ TETML出力は、XMLベースのワークフローと、さまざまなXML処理ツールやXSLT等のXML処理言語に通じた開発者に適しています。
- ▶ TET コネクタは、データベースや検索エンジン等、広く利用されているさまざまなソフトウェアパッケージにTETを統合させるために適しています。
- ▶ TET Pluginは、TETを対話的に利用できるようにする、Adobe Acrobat用の無償拡張です（詳しくは51ページの4.1「Adobe Acrobat用無償TET Plugin」を参照）。

## 1.3 文書とサンプルへのロードマップ

**TET ライブラリ用ミニサンプル** TET 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プログラミング作成例を含んでいます。これらのミニサンプルは、自分のアプリケーションのための出発点として利用したり、自分のTETインストールを試験したりするために利用できます。以下のアプリケーションのためのソースコードから成っています：

- ▶ **extractor** サンプルは、PDF文書からテキストを抽出するための基本的なループを演示しています。
- ▶ **image\_resources** サンプルは、PDF文書から画像をリソース志向な方式で抽出するための基本的なループを演示しています。
- ▶ **dumper** サンプルは、PDF文書に関する一般情報を取得するための内蔵 pCOS インタフェースの使用を示しています。
- ▶ **fontfilter** サンプルは、フォント名や文字サイズといったフォント関連情報の処理方法を示しています。
- ▶ **glyphinfo** サンプルは、グリフに関する情報（フォント・サイズ・位置等）や、**dropcap**・**shadow**・**hyphenation** といったテキスト属性の取得方法を演示しています。
- ▶ **tetml** サンプルは、PDF文書からTETML（PDFの内容を表現するためのTETのXML言語）を生成するためのプロトタイプのコードを含んでいます。
- ▶ **get\_attachments** サンプル（すべての言語バインディングで得られるわけではありません）は、PDFファイル添付、すなわち別のPDF文書内に埋め込まれているPDF文書の処理方法を演示しています。

注記 Windows Vista・Windows 7 では、ミニサンプルはデフォルトで「Program Files」ディレクトリにインストールされます。Windows Vista の新しい保護方式によって、これらのサンプルによって生成されるPDF出力ファイルは、「互換性ファイル」下でしか表示されません。推奨回避法：サンプルをユーザディレクトリへ複製します。

**XSLT サンプル** TET ディストリビューションは、いくつかのXSLTスタイルシートを含んでいます。これらは、さまざまな目的を実現するためのTETMLの処理方法を演示しています：

- ▶ **concordance.xsl**：文書内の単語を頻度の多い順に並べたリストを生成します。
- ▶ **fontfilter.xsl**：文書内の、指定した値よりも大きなサイズで特定のフォントを用いているすべての単語をリストします。



- ▶ *fontfinder.xml* : 文書内のすべてのフォントについて、すべての出現箇所をページ番号と位置情報とともにリストします。
- ▶ *fontstat.xml* : フォントとグリフの統計を生成します。
- ▶ *index.xml* : アルファベット順の索引を生成します。
- ▶ *metadata.xml* : TETML 内に含まれている文書レベル XMP メタデータから、選択したフィールド群を抽出します。
- ▶ *solr.xml* : Solr エンタプライズ検索サーバ用の入力を生成します。
- ▶ *table.xml* : 表を CSV ファイル (カンマ区切り値) へ抽出します。
- ▶ *tetml2html.xml* : TETML をシンプルな HTML へ変換します。
- ▶ *textonly.xml* : TETML 入力から生テキストを抽出します。

**TET クックブック** TET クックブックは、特定の応用問題を TET ライブラリで解決するためのソースコード作成例の集合です。このクックブックは Java 言語で書かれていますが、他の言語でも容易に利用できます。なぜなら TET API はすべての対応言語バインディングでほぼ等価だからです。いくつかのクックブックサンプルは XSLT 言語で書かれています。TET クックブックは以下のグループにまとめられています：

- ▶ Text : テキスト抽出関連のサンプル群
- ▶ Font : フォント属性に焦点をあてたテキスト関連のサンプル群
- ▶ Image : 画像抽出関連のサンプル群
- ▶ TET & PDFlib+PDI:PDF から TET で情報を抽出して、新たな PDF を元 PDF と抽出情報に基づいて構築するサンプル群。これらのサンプルには、TET に加えて PDFlib+PDI 製品が必要です。
- ▶ TETML : TETML 処理用の XSLT サンプル群
- ▶ Special : その他のサンプル群

TET クックブックは下記 URL で利用可能です：  
[www.pdfliib.com/tet-cookbook](http://www.pdfliib.com/tet-cookbook)

**pCOS クックブック** pCOS クックブックは、TET に内蔵されている pCOS インタフェースのためのコードの集合です。下記 URL で利用可能です：  
[www.pdfliib.com/pcos-cookbook](http://www.pdfliib.com/pcos-cookbook)

pCOS インタフェースの詳細は、TET パッケージに含まれている pCOS パスリファレンスに示されています。

## 1.4 TET 4.0 の新機能

TET 4.0 の新機能と大幅改善機能を以下に示します：

- ▶ パフォーマンス向上：さまざまな種別の文書でより速くなりました。
- ▶ 数十万ページの超大文書でもより速く、よりメモリ消費が少なくなりました。
- ▶ アラビア文字・ヘブライ文字等の右書き・双方向テキストを抽出。
- ▶ 正規化・字形統合・分解の制御による Unicode 後処理。
- ▶ 影付き除去・単語境界検出・ハイフン除去の改善。
- ▶ 上付き・下付き検出の改善。
- ▶ 非準拠 PDF 文書に対する回避策の導入による堅牢性の向上。
- ▶ 破損 PDF からうまくテキストを抽出するための修復モードの改善。

- ▶ XML 出力 (TETML) 内の情報の増加 : ハイフン除去・ドロップキャップ・影付き・上付き / 下付き。下向き座標系・PDF/A-2・PDF/E・フォントサブセット。
- ▶ C++・Perl 言語バインディングの改善。

## 1.5 TET 4.1 の新機能

TET 4.1 の新機能と大幅改善機能を以下に示します :

- ▶ PDF 1.7 拡張レベル 8 に対応 (ISO 32000-2 で仕様化された暗号化方式)
- ▶ 擬似オブジェクトが増え (フォントの詳細など)、暗号化された添付の取り扱い方を明確にした pCOS インタフェース 8 へのアップデート
- ▶ TETML 出力内に、PDF 文書とフォントに関するさらなる情報
- ▶ 新たに Objective-C・Ruby 用の言語バインディング
- ▶ 表意文字の日中韓テキストに対する単語境界検出の改良 (オプション *ideographic*)
- ▶ 新たな API 関数 `TET_convert_to_unicode()`・`TET_info_pvf()`
- ▶ Lucene・Solr 用のコネクタをアップデート
- ▶ メタデータと構造化テキスト内容の検出と抽出のためのTIKAツールキット用の新たなコネクタ
- ▶ パフォーマンスの向上、とりわけ大規模文書に対して
- ▶ さまざまなバグ修正と小規模な改良

# 2 TET コマンドラインツール

## 2.1 コマンドラインオプション

TET コマンドラインツールを利用すれば、プログラミングを一切必要とせずに、1つないし複数の PDF 文書からテキスト・画像を抽出することができます。出力は、プレーンテキスト (Unicode) 形式か、または TET の XML ベースの出力形式である TETML 形式で生成することができます。この TET プログラムはたくさんのコマンドラインオプションで制御できます。このプログラムは各単語の後に空白キャラクタ (U+0020) を挿入し、また各行の後に U+000A を、各ページの後に U+000C を挿入します。これを呼び出すには次のように1つないし複数の PDF ファイルを指定します。

```
tet [<オプション群>] <ファイル名>...
```

この TET コマンドラインツールは TET ライブラリの最上位の上に構築されています。`--docopt`・`--teto`・`--imageopt`・`--pageopt` オプションを用いれば、151 ページの 10 章「TET ライブラリ API リファレンス」のオプション一覧表に従ったさまざまなオプションをライブラリに与えることが可能です。表 2.1 にすべての TET コマンドラインオプションを挙げます (この一覧は、この TET プログラムをオプションなしで走らせたときにも表示されます)。

注記 日中韓テキストを抽出するには、7 ページの 0.1「ソフトウェアをインストール」に従って TET 同梱の CMap ファイル群の置き場所を設定する必要があります。

表 2.1 TET コマンドラインオプション一覧

オプション	引数	機能
--		オプションのリストを終了させます。これは、ファイル名が - キャラクタで始まっているときに有用です。
@filename <sup>1</sup>		オプション群を持つレスポンスファイルを指定します。文法の説明は 22 ページの「レスポンスファイル」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、かつ最初のファイル名の前でのみ認識されます。レスポンスファイルを用いて他のオプションの引数を置き換えることはできず、オプションに引数を補完するためにのみ用いることができます。
--docopt	<オプションリスト>	open_document( ) に対する追加のオプションリスト (172 ページの表 10.8 参照)。tetml オプションの filename サブオプションはここでは使えません。
--firstpage -f	<integer>   last	テキスト抽出を開始するページの番号。キーワード last で末尾ページを指定、last-1 で末尾ページの前のページ、等々。デフォルト : 1
--format	utf8   utf16	テキスト出力の形式を指定 (デフォルト : utf8) : <b>utf8</b> BOM (バイトオーダーマーク) 付き UTF-8 <b>utf16</b> ネイティブバイト順序の BOM 付き UTF-16 このオプションは、つねに UTF-8 で生成される TETML 出力には効力を持ちません。
--help, -? (またはオプションなし)		利用可能なオプションをまとめたヘルプを表示

表 2.1 TET コマンドラインオプション一覧

オプション	引数	機能
<code>--image<sup>2</sup></code> <code>-i</code>		文書から画像を抽出します。抽出された画像の命名規則は <code>--imagemap</code> オプションに依存します。
<code>--imagemap</code>	page   resource	<p><code>--image</code> オプションによる画像抽出の数え上げグループの種類を指定します (デフォルト: <code>--tetml</code> を指定しているなら resource、そうでないなら page) :</p> <p><b>page</b> 選択されたページ群にあるすべての画像を数え上げます。複数回配置されている画像は複数回抽出されます。抽出された画像は下記のパターンで命名されます:          &lt;ファイル名&gt;_p&lt; ページ番号 &gt;_&lt; 画像番号 &gt;.[tif jpg jpx]</p> <p><b>resource</b> 文書内のすべての (結合した) 画像リソースを数え上げます。各画像リソースは、文書内に現れる回数にかかわらず、一度だけ抽出されます。画像抽出にあたっては <code>--firstpage</code> <code>--lastpage</code> オプションは無視されます。抽出された画像は下記のパターンで命名されます:          &lt;ファイル名&gt;_I&lt; 画像 ID&gt;.[tif jpg jpx]          注記: I&lt; 画像 ID&gt; は TETML 属性 Image/@id でも用いられます。</p>
<code>--imageopt</code>	< オプションリスト >	write_image_file( ) に対する追加のオプションリスト (194 ページの表 10.17 を参照)
<code>--lastpage</code> <code>-l</code>	<integer>   last	テキスト抽出を終了するページの番号。キーワード last で末尾ページを指定、last-1 で末尾ページの前のページ、等々。デフォルト: last
<code>--outfile</code> <code>-o</code>	< ファイル名 >	(複数の入力ファイル名を与えているときは不可) テキスト出力または TETML 出力のファイル名。入力ファイルを一個だけ与えているときのみ、ファイル名「-」を用いて標準出力を指定可能。デフォルト: 入力ファイル名の .pdf・.PDF を .txt (テキスト出力時) か .tetml (TETML 出力時) に置き換えた名前。
<code>--pageopt</code>	< オプションリスト >	テキスト出力生成なら open_page( ) に、TETML 出力生成なら process_page( ) に与えられる追加のオプションリスト。使えるオプションについては 179 ページの表 10.10 と 196 ページの表 10.18 を参照。テキスト出力の場合、オプション granularity はつねに page に設定されます。
<code>--password,</code> <code>-p</code>	< パスワード >	暗号化文書に対するユーザ・マスタ・添付いずれかのパスワード。場合によっては、シュラッグ機能を利用すると、パスを与えずに暗号化文書をインデックスすることができます (67 ページの 5.1 「暗号化 PDF から内容を抽出」を参照)。
<code>--searchpath<sup>1</sup></code> <code>-s</code>	< パス >...	ファイル (CMap 等) を検索する 1 つないし複数のディレクトリの名前。デフォルト: インストール環境に依存
<code>--targetdir</code> <code>-t</code>	< ディレクトリ名 >	生成されるテキスト・TETML・画像ファイルの出力ディレクトリ。存在しているディレクトリである必要があります。デフォルト: . (すなわちカレント作業ディレクトリ)
<code>--tetml</code> <code>-m</code>	glyph   word   wordplus   line   page	<p>(<code>--text</code> との組み合わせ不可) テキスト・画像情報を含む TET 3 スキーマに従った TETML 出力を生成します。TETML はつねに UTF-8 で生成されます。与える引数で、いくつかの種類の一つを選びます (135 ページの 9.2 「TETML の詳細を制御」を参照) :</p> <p><b>glyph</b> グリフ位置情報とフォント情報を持ったグリフベースの TETML</p> <p><b>word</b> 単語枠を持った単語ベースの TETML</p> <p><b>wordplus</b> 単語枠とすべてのグリフ情報を持った単語ベースの TETML</p> <p><b>ine</b> 行ベースの TETML (テキストのみ)</p> <p><b>page</b> ページベースの TETML (テキストのみ)</p>

表 2.1 TET コマンドラインオプション一覧

オプション	引数	機能
<b>--teto</b>	< オプションリスト >	set_option( ) に対する追加のオプションリスト (158 ページの表 10.2 を参照)。オプション outputformat は無視されます (かわりに --format を用いてください)。
<b>--text<sup>2</sup></b>		(--tetml との組み合わせ不可) 文書からテキストを抽出します (デフォルトで有効)
<b>--verbose</b> <b>-v</b>	0   1   2   3	冗長レベル (デフォルト : 1) : <b>0</b> 一切出力なし <b>1</b> エラーのみ出力 <b>2</b> エラーとファイル名を出力 <b>3</b> 詳細レポート
<b>--version, -V</b>		TET バージョン番号を印字します。

1. このオプションは複数回与えることもできます。

2. オプション --image はテキスト抽出を無効にしますが、それは --text ・ --tetml と組み合わせることもできます。

## 2.2 TET コマンドラインを構築

TET コマンドラインの作成にあたっては、以下の規則を守る必要があります。

- ▶ 入力ファイルは、*searchpath* で指定し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オプション形が利用可能で、長オプション形との混在も可能です。
- ▶ 長オプションは先頭の数文字だけで表すことも可能です。ただしその省略形が一意である場合に限りです。
- ▶ 入力ファイルの暗号化ステータスによっては、テキストを正しく抽出するためにユーザパスワードかマスタパスワードが必要となる場合があります。これは *--password* オプションで与える必要があります。TET はこのパスワードでテキスト抽出が許されるかどうかを調べ、だめならエラーを生成します。

TET はコマンドライン全体をまず調べてからファイルの処理を開始します。コマンドライン上どここのオプション内であれエラーが出たならば、ファイルはどれも一切処理されないままとなります。

**ファイル名** 空白キャラクタを含むファイル名は、TET のようなコマンドラインツールで用いる際には若干特殊な取り扱いを要します。空白キャラクタを含んだファイル名を処理させるには、ファイル名全体をダブルクォート文字 " でくくらなければなりません。ワイルドカードが標準的用法に従って使えます。たとえば *\*.pdf* は、所与のディレクトリ内でファイル名接尾辞 *.pdf* を持つすべてのファイルを表します。システムによって、大文字小文字を区別するものとししないものがあることに注意しましょう (*\*.pdf* が *\*.PDF* と違ったり等)。また Windows システム群ではワイルドカードは、空白キャラクタを含むファイル名には効かないので注意して下さい。ワイルドカードはカレントディレクトリ内で評価され、検索パスディレクトリ内では評価されません。

Windows では、すべてのファイル名オプションが Unicode 文字列を受け入れます。たとえばこれは、エクスプローラ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際に発生します。

**レスポンスファイル** オプション群をコマンドラインで直接与える方法以外に、オプション群をレスポンスファイル内で与える方法もあります。レスポンスファイルの内容は、コマンドライン内で *@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持った単純なテキストファイルです。これは以下の文法規則に従う必要があります：

- ▶ オプション値群は空白文字、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。
- ▶ 空白文字を含む値は、ダブルクォーテーションマーク " で囲む必要があります。
- ▶ 値の先頭と末尾のダブルクォーテーションマークは無視されます。
- ▶ 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は、もう一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、*@filename* 文法はそれ自体をレスポンスファイルで用いることもできます。

レスポンスファイルは、ファイル名引数に対して Unicode 文字列を持つこともできます。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 で、それ以外のすべてのシステムでは ISO 8859-1 (Latin-1) で解釈されます。

**終了コード** TET コマンドラインツールは以下に示す終了コードを返すので、これを利用すれば、要求した操作が正しく遂行でき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の処理が成功し完了した。
- ▶ 終了コード 1：ファイル処理エラーが 1 件ないし複数発生、しかし処理は継続された。
- ▶ 終了コード 2：コマンドラインオプションの中に何らかのエラーが発見された。処理はその不良オプションの位置で停止したため、入力ファイルは一切処理されなかった。

## 2.3 コマンドラインの例

以下の例は、TET 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演示しています。作成例は 2 種類示されています。1 種類目はすべてのオプションの長形式を用いており、一方 2 種類目はそれと同等な短オプション形式を用いています。

### 2.3.1 テキストを抽出

PDF 文書 *file.pdf* からテキストを UTF-8 形式で抽出し、それを *file.txt* に格納します：

```
tet file.pdf
```

テキスト抽出から先頭ページと末尾ページを除外します：

```
tet --firstpage 2 --lastpage last-1 file.pdf  
tet -f 2 -l last-1 file.pdf
```

日中韓 CMap が置かれているディレクトリを与えます（日中韓テキスト抽出では必須です）：

```
tet --searchpath /usr/local/cmaps file.pdf  
tet -s /usr/local/cmaps file.pdf
```

PDF からテキストを UTF-16 形式で抽出し、それをファイル *file.utf16* に格納します：

```
tet --format utf16 --outfile file.utf16 file.pdf  
tet --format utf16 -o file.utf16 file.pdf
```

ディレクトリ内のすべての PDF ファイルからテキストを抽出し、生成された *\*.txt* ファイル群を別のディレクトリ（すでに存在している必要があります）に格納します：

```
tet --targetdir out in/*.pdf  
tet -t out in/*.pdf
```

テキスト抽出をページ上の特定の領域に限定します。これは適切なページオプションのリストを与えることで実現できます：

```
tet --pageopt "includebox={{0 0 200 200}}" file.pdf
```

さまざまなコマンドラインオプションを内容として持つレスポンスファイルを用いて、カレントディレクトリ内のすべての PDF 文書を処理します（ファイル *options* にコマンドラインオプション群が入っています）：

```
tet @options *.pdf
```

### 2.3.2 画像を抽出

*file.pdf* から画像群をページ志向な方式で抽出し、それらをディレクトリ *out* 内の *file\*.tif*/*file\*.jpg* に格納します：

```
tet --targetdir out --image file.pdf  
tet -t out -i file.pdf
```

*file.pdf* から画像群をリソース志向な方式で抽出し、それらをディレクトリ *out* 内の *file\*.tif*/*file\*.jpg* に格納します：



```
tet --targetdir out --image --imageloop resource file.pdf
tet -t out -i --imageloop resource file.pdf
```

*file.pdf* から画像群を画像結合なしに抽出します。これは、画像処理に関連した適切なページオプションのリストを与えることで実現できます：

```
tet --targetdir out --image --pageopt "imageanalysis={merge={disable}}" file.pdf
tet -t out -i --pageopt "imageanalysis={merge={disable}}" file.pdf
```

### 2.3.3 TETML を生成

PDF 文書 *file.pdf* に対する TETML 出力を単語モードで生成し、それを *file.tetml* に格納します：

```
tet --tetml word file.pdf
tet -m word file.pdf
```

TETML 出力を、*Options* 要素一切なしで生成します。これは、適切な文書オプションのリストを与えることで実現できます：

```
tet --docopt "tetml={elements={options=false}}" --tetml word file.pdf
```

TETML 出力を、すべてのグリフ情報を持った単語モードで生成し、それを *file.tetml* に格納します：

```
tet --tetml word --pageopt "tetml={glyphdetails={all}}" file.pdf
tet -m word --pageopt "tetml={glyphdetails={all}}" file.pdf
```

画像群を抽出して TETML を生成することを、一回の呼び出しで行います：

```
tet --image --tetml word file.pdf
tet -i -m word file.pdf
```

TETML 出力を下向き座標で生成します：

```
tet --tetml word --pageopt "topdown={output}" file.pdf
tet -m word --pageopt "topdown={output}" file.pdf
```

### 2.3.4 高度なオプション

ある種の TeX 由来の PDF 文書に対して、文書オプション *checkglyphlists* を与えて Unicode マッピングを改善します：

```
tet --docopt checkglyphlists file.pdf
```

Unicode 字形統合を適用します。たとえば空白字形統合：すべての種類の Unicode 空白キャラクターを U+0020 へマップします：

```
tet --docopt "fold={{[:blank:]} U+0020}" file.pdf
```

句読点を単語境界として無効にします：

```
tet --pageopt "contentanalysis={punctuationbreaks=false}" file.pdf
```



# 3 TETライブラリの言語バインディング

この章では、TET ライブラリで供給されるさまざまな言語バインディングの仕様について解説します。TET ディストリビューションには、対応しているすべての言語バインディングによるいくつかの小さな TET アプリケーションのための完全なサンプルコードが含まれています。

## 3.1 例外処理

ある種のエラーは多くの言語において例外と呼ばれています。それにはもっともな理由があり、そうしたエラーは単なる例外で、プログラムが始まって終わるまでの間にそう頻繁には起こらないと予想されるからです。一般的なさばき方としては、まずいことが頻繁に起こりそうな関数呼び出しについては通常のエラー報告のしくみ（要は特別なエラーコードを返す）を採用するとして、めったに起こらなさそうなものについてまでそれをやるとコードがあらゆる場合分けによって条件文だらけになってしまうので、それを避けるためにそうしたものについては特殊な例外のしくみを採用するのが普通です。TET でもまさにその通りのことを行っています。操作によっては、割と頻繁にまずいことが起きると予想できるものがあります。たとえば：

- ▶ 正しいパスワードを知らない PDF 文書を開こうとした（ただし、67 ページの 5.1「暗号化 PDF から内容を抽出」で説明するシュラッグ機能も参照）。
- ▶ 誤ったファイル名で PDF 文書を開こうとした。
- ▶ 修復できないほど破損している PDF 文書を開こうとした。

TET はこのようなエラーを、値 -1 を返すことによって知らせます。API リファレンスに記述している通りです。一方、有害と考えられる出来事ではあっても起きる頻度は割と低いものもあります。たとえば：

- ▶ 仮想メモリが足りなくなった。
- ▶ 関数に与えたパラメタがおかしい（文書のハンドルが無効だった等）。
- ▶ オプションリストを与えたら形式が間違っていた。
- ▶ 必要なリソースなのに（日中韓テキスト抽出には CMap ファイルが必要等）見つからない。

TET がこのような状況を検知した場合には、特別なエラー値が呼び出し元に返るのではなく、例外が発生します。例外にネイティブに対応している言語では、例外の発生は、その言語・環境と与える標準手段を用いて行われます。C 言語バインディングの場合には、TET はカスタムの例外処理のしくみを提供しているので、クライアントはそれを用いる必要があります（29 ページの 3.2「C バインディング」参照）。

重要な留意点としては、例外が起きたら文書の処理は止めなければなりません。例外の後に呼んでも安全なメソッドは `delete()`・`get_apiname()`・`get_errnum()`・`get_errmsg()` だけです。これ以外のメソッドを例外の後に呼ぶと予測できない結果につながる可能性があります。この例外は以下の情報を持っています。

- ▶ 一意のエラー番号。
- ▶ 例外の原因となった API 関数の名前。
- ▶ 問題点を説明した文。

**失敗した関数呼び出しの原因を取得** TET の関数によっては、たとえば `open_document()` や `open_page()` 等、呼び出して失敗しても例外が発生しないものがあります（そうした関

数はエラー時には -1 を返します)。そのような場合には、失敗した関数呼び出しの直後に関数 `get_errnum()`・`get_errmsg()`・`get_apiname()` を呼び出せば、問題の性質に関する詳しい情報を得ることができます。

## 3.2 C バインディング

TET は C 言語で、いくつかの C++ モジュールを用いて書かれています。C バインディングを利用するには、静的または共有ライブラリのいずれかを使用することができ (Windows・MVS の場合は DLL)、自分のソースモジュールにインクルードするためのセントラル TET インクルードファイル `tetlib.h` が必要です。あるいは、`tetlibdl.h` を用いて TET DLL を実行時に動的に読み込むこともできます (詳しくは次項参照)。

**注記** C 用の TET バインディングを用いるアプリケーションは、C++ コンパイラでリンクする必要があります。なぜならこのライブラリには C++ で実装されている部分があるからです。C リンカを使用すると、アプリケーションを明示的に必要な C++ サポートライブラリ群にリンクしないかぎり、非決定的外部参照を引き起こす可能性があります。

**実行時に読み込まれる DLL として TET を使用** 多くのクライアントでは TET を、リンク時に結合される静的結合ライブラリか動的ライブラリとして利用しますが、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。これはとりわけ、DLL を必要に応じてのみ読み込むために有用であり、また、MVS 上ではライブラリは TET に対して明示的にリンクされることなく DLL として実行時に個別に読み込まれるので有用です。TET では、この動的な利用法を実現するために特化した機構を用意しています。これは以下の規則に従って動かします：

- ▶ `tetlib.h` でなく `tetlibdl.h` をインクルード。
- ▶ `TET_new()`・`TET_delete()` でなく `TET_new_dl()`・`TET_delete_dl()` を使用。
- ▶ `TET_TRY()`・`TET_CATCH()` でなく `TET_TRY_DL()`・`TET_CATCH_DL()` を使用。
- ▶ その他すべての TET 呼び出しに関数ポインタを使用。
- ▶ 追加モジュール `tetlibdl.c` をコンパイルして、生成されるオブジェクトファイルに自分のアプリケーションをリンク。

動的読み込みのしくみを `extractordl.c` サンプルで演示しています。

**注記** DLL の実行時読み込みは、選ばれたプラットフォームでのみ対応しています。

**例外処理** TET の API は、C 言語にはネイティブな例外処理がないのを補うために、このライブラリが発生させた例外に対処するためのしくみを提供しています。`TET_TRY()` マクロと `TET_CATCH()` マクロを用いることにより、例外発生時にエラー処理やクリーンアップを実行させるためのコードを入れ込んだクライアントコードを組み立てることができるのです。これらのマクロはコード内に 2 つの部分に分けます。1 つは try 節で、例外が発生させる可能性のあるコードを持ち、もう 1 つは catch 節で、例外に対処するコードを持ちます。try ブロックの中から呼び出された API 関数のいずれかが例外が発生させたとき、プログラムの実行はただちに catch ブロックの先頭ステートメントに移ります。TET のクライアントのコードは以下の規則に従う必要があります：

- ▶ `TET_TRY()` と `TET_CATCH()` は必ず対にしなければなりません。
- ▶ `TET_new()` は決して例外を発生させません。try ブロックは、有効な TET オブジェクトハンドルを用いなければ開始させることができないので、`TET_new()` はすべての try ブロックの外で呼び出さなければなりません。
- ▶ `TET_delete()` は決して例外を発生させません。よって、すべての try ブロックの外で安全に呼び出すことができます。catch 節の中で呼び出すこともできます。
- ▶ try ブロックと catch ブロックの両方の中で用いられている変数については特別な注意を払う必要があります。コンパイラは、制御がブロックからブロックへ移ることは知

らないので、そのような場合、不適切なコードを生成する可能性があります (レジスタ変数最適化等)。

幸い、この種の問題を避ける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** と宣言する必要があります。 **volatile** キーワードの使用はコンパイラに、危険な最適化を変数に行ってはならないということを知らせませ

- ▶ try ブロックを抜ける場合には (たとえば return ステートメントによって、ひいては対応する **TET\_CATCH()** の実行をバイパスして)、return ステートメントの前に **TET\_EXIT\_TRY()** マクロを呼び出して、例外のしくみにそのことを通知する必要があります。
- ▶ TET のあらゆる言語バインディングの場合と同様、例外が発生したら文書処理は止めなければなりません。

以下のコードに、これらの規則の遵守例を示します。クライアントのコード内で TET の例外を扱う際の典型的な書き方もあわせて示しています (完全なサンプルは TET パッケージ内にあります) :

```
volatile int pageno;
...
if ((tet = TET_new()) == (TET *) 0)
{
    printf("メモリがいっぱいです\n");
    return(2);
}
TET_TRY(tet)
{
    for (pageno = 1; pageno <= n_pages; ++pageno)
    {
        /* ページ処理 */

        if (/* エラーが起きた */)
        {
            TET_EXIT_TRY(tet);
            return -1;
        }
    }
    /* API関数群を直接・間接に呼び出すステートメント群 */
}
TET_CATCH(tet)
{
    printf("エラー %d が %s() 内で %d ページ上で発生しました: %s\n",
          TET_get_errnum(tet), TET_get_apiname(tet), pageno, TET_get_errmsg(tet));
}
TET_delete(tet);
```

**名前文字列に対する Unicode 処理** C 言語はネイティブで Unicode に対応していません。API 関数に対する文字列パラメタのなかには**名前文字列**として宣言されているものがあります。これらは **length** パラメタと、文字列の先頭の BOM の有無によって処理のされ方が異なります。C の場合、**length** パラメタが 0 でないときは文字列は UTF-16 と解釈されます。**length** パラメタが 0 のときは、文字列が UTF-8 BOM で始まっていれば UTF-8 と解釈され、EBCDIC UTF-8 BOM で始まっていれば EBCDIC UTF-8 と解釈され、BOM がなければ **host** エンコーディング (ただし EMBDIC ベースの全プラットフォーム上では **ebcdic**) と解釈されます。

**オプションリストに対する Unicode 処理** オプションリストに文字列を含めるときは特別な注意が必要です。なぜなら、それは UTF-16 形式の Unicode 文字列として表すことはできず、バイト列として表すことしかできないからです。そのため、Unicode のオプションに対しては UTF-8 が用いられます。オプションの先頭の BOM を見て、TET はそれをどう解釈するかを決めます。この BOM を用いて文字列の形式が判定されます。具体的には、文字列オプションの解釈動作は以下の通りです：

- ▶ オプションが UTF-8 BOM (`\xEF\xBB\xBF`) で始まっていれば UTF-8 と解釈されます。
- ▶ オプションが EBCDIC UTF-8 BOM (`\x57\x8B\xAB`) で始まっていれば EBCDIC UTF-8 と解釈されます。
- ▶ BOM がなければ文字列は *winansi* (ただし EBCDIC ベースのプラットフォーム上では *ebcdic*) として扱われます。

**注記** `TET_convert_to_unicode()` ユーティリティ関数を用いると、UTF-16 文字列を UTF-8 文字列に変換することができます。Unicode 値を持つオプションリストを作りたいときに便利です。

## 3.3 C++ バインディング

**注記** C++ で書かれた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、TET .NET DLL に直接アクセスすることを推奨します（ただし C++ バインディングを用いるべきクロスプラットフォームアプリケーションを除く）。TET 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 で使用する C++ サンプルコードを含んでいます。

*tetlib.h* C ヘッダファイルに加えて、C++ 用のオブジェクト指向ラップも提供されており、TET のクライアントから利用することができます。このラップはヘッダファイル *tetlib.h* を必要とします。このヘッダファイルは *tetlib.h* をインクルードします。*tet.hpp* はテンプレートベースの実装を内容として持ちますので、対応する *tet.cpp* モジュールは必要ありません。C++ オブジェクトラップを使うことは、API 関数と、すべての TET 関数名の *TET\_* 接頭辞とによる関数的アプローチを、よりオブジェクト志向なアプローチへ置き換えます。

**実行時に読み込まれる DLL として TET を利用** C 言語バインディングと同様に、C++ バインディングでは、TET を実行時に動的に自分のアプリケーションにアタッチすることができます（29 ページの「実行時に読み込まれる DLL として TET を使用」を参照）。動的読み込みを有効にするには、*tet.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時に下記のようにします：

```
#define TETCPP_DL 1
```

これに加えて、追加モジュール *tetlibdl.c* をコンパイルして、その結果できるオブジェクトファイルに自分のアプリケーションをリンクさせる必要があります。動的読み込みの詳細は、TET オブジェクト内に隠蔽されていますので、C++ API には影響を与えません：すべてのメソッド呼び出しは、動的読み込みが有効になっているかどうかにかかわらず、同じ見た目になります。動的読み込みのしくみを、提供する Makefile 内の *extractordl* サンプルで演示しています。

**注記** DLL の実行時読み込みには、一部プラットフォームでのみ対応しています。

**C++ の文字列処理** TET のテンプレートベースの文字列処理アプローチでは、文字列処理に関して以下の使用パターンが使えます：

- ▶ 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 TET 4.0 以降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かぎり、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かぎり、用いることができます。
- ▶ プレーン C++ 文字列を、TET 3.0 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、TET 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* をつ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 ベースのシステムの場合、*wstring* ベースのインタフェースに対するオプションリスト文字列を作成する際には、オプションリスト内に EBCDIC と UTF-16 の *wstring* が混在することを防ぐため、さらに変換を行う必要があります。この変換に便利なコードと利用法が、追加モジュール *utf16num\_ebcdic.hpp* 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** TET 3.0 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、以下のようにして適合させることができます：

- ▶ TET C++ クラスは *pdflib* 名前空間内に入りましたので、クラス名を修飾する必要があります。いちいち *pdflib::TET* 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 TET メソッドを使う前に下記を追加すべきです：

```
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* 接頭辞を頭につける必要があります。たとえば

```
const wstring pageoptlist = L"granularity=page";
```

- ▶ TET のエラーメッセージと例外文字列 (*TET::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。
- ▶ TET C++ バインディングで *tet.cpp* モジュールは必要なくなりました。TET 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 TET 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：

- ▶ *tet.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 TETCPP_TET_WSTRING 0
```

- ▶ *tet.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 TETCPP_USE_PDFLIB_NAMESPACE 0
```

**C++ のエラー処理** TET 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、TET クラスはパブリックな *TET::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 TET 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。

TET 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、TET が発生させた例外をキャッチします：

```
try {
    ...いろいろなTET命令...
} catch (TET::Exception &ex) {
    wcerr << L"Error " << ex.get_errnum()
    << L" in " << ex.get_apiname()
    << L"(): " << ex.get_errmsg() << endl;
}
```

## 3.4 COM バインディング

**TET の COM エディションをインストール** TET は COM コンポーネントに対応したすべての環境で利用することができます。TET のインストール方法は簡単で単純です。以下の点に留意して下さい：

- ▶ NTFS パーティションにインストールする場合は、すべての TET ユーザに、そのインストールディレクトリの読み取り権限と、...\*TET 4.132-bit\bind\COM\bin\tet\_com.dll* の実行権限を持たせる必要があります。
- ▶ インストーラはシステムレジストリに対して書き込み権限を持たなければなりません。Administrators グループか Power Users グループの権限であれば通常充分でしょう。

**例外処理** TET の COM コンポーネントに対する例外処理は COM の規則に従って行われます。すなわち、TET の例外が起きると、COM の例外が発生し、そのエラーの説明テキストとともに伝達されます。この COM の例外は、そのクライアント環境が対応している任意の COM のエラー処理方式で捕捉・処理することができます。

**TET の COM エディションを .NET で使う** TET の .NET エディション (38 ページの 3.6 「.NET バインディング」参照) のかわりとして、TET の COM エディションは .NET で使用することも可能です。まず、*tlbimp.exe* ユーティリティを用いて TET の COM エディションから .NET アセンブリを作成する必要があります：

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

.NET アプリケーション内ではこのアセンブリを使うことができます。Visual Studio .NET 内から *tet\_com.dll* への参照を追加すればアセンブリは自動作成されます。C# における TET の COM エディションの使用法を以下のコードに示します：

```
using TET_com;  
...  
static TET_com.ITET tet;  
...  
tet = New TET();  
...
```

上記に示した以外のコードはすべて TET の .NET エディションと同様に動作します。

## 3.5 Java バインディング

**TET の Java エディションをインストール** TET は `com.pdflib.TET` という名前で Java パッケージとしてまとめられています。このパッケージはネイティブ JNI ライブラリに依存しますので、両者は適切に設定しておく必要があります。

JNI ライブラリを可能にするために必要な操作は、以下のようにプラットフォームによって異なります：

- ▶ Unix システム群の場合には、`libtet_java.so` (Mac OS X では `libtet_java.jnilib`) ライブラリを、共有ライブラリ用のデフォルトの場所、ないし適切に設定したディレクトリに置く必要があります。
- ▶ Windows の場合には、`pdf_tet.dll` ライブラリを、Windows のシステムディレクトリ、ないし PATH 環境変数に挙げたディレクトリに置く必要があります。

この TET の Java パッケージはファイル `tet.jar` 内に存在し、ただ一つのクラス `tet` を持ちます。このパッケージを自分のアプリケーションで利用可能にするには、`tet.jar` を `CLASSPATH` 環境変数に追加するか、Java コンパイラへの呼び出しの中に `-classpath tet.jar` オプションを加えるか、またはこれと等価な操作を Java IDE で行う必要があります。JDK で Java VM の設定を行うことによって、ネイティブライブラリが所与のディレクトリ内で検索されるようにするには、`java.library.path` にそのディレクトリの名前を適切に設定します。たとえば

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

このプロパティの値は下記のようにして調べられます：

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

**TET を J2EE アプリケーションサーバとサーブレットコンテナで使う** TET はサーバサイド Java アプリケーションに完全に適合しています。TET ディストリビューションは、TET を J2EE 環境で使うためのサンプルコードと設定を含んでいます。以下の設定事項に従う必要があります：

- ▶ サーバがネイティブライブラリを検索するディレクトリはベンダによって異なります。よくある場所としては、システムディレクトリや、背後の Java VM に固有のディレクトリや、ローカルサーバディレクトリを候補に挙げることができます。サーバベンダが提供している説明書を調べて下さい。
- ▶ アプリケーションサーバとサーブレットコンテナはしばしば、特殊なクラスローダを用いており、そのクラスローダは制限されているか、あるいは専用のクラスパスを用いている可能性があります。サーバによっては、特別なクラスパスを定義して、TET パッケージが必ず見つかるようにしておくことが必要です。

TET を個別のサーブレットエンジンとアプリケーションサーバで使う際のより詳細な注意点は、TET ディストリビューションの J2EE ディレクトリ内の追加文書に記してあります。

**Unicode とレガシエンコーディングの変換** TET ユーザの便宜のため、有用な文字列変換メソッドをここでいくつか挙げます。下記のコンストラクタは、プラットフォームのデフォルトエンコーディングを用いて、バイト列から Unicode 文字列を生成します：

```
String(byte[] bytes)
```

下記のコンストラクタは、*enc* 引数で与えたエンコーディング (*SJIS*・*UTF8*・*UTF-16* 等) を用いて、バイト列から Unicode 文字列を生成します：

```
String(byte[] bytes, String enc)
```

String クラスの下記のメソッドは、Unicode 文字列を、*enc* 引数で指定したエンコーディングに従った文字列へ変換します：

```
byte[] getBytes(String enc)
```

**TET の Javadoc 文書** TET パッケージは TET の Javadoc 文書を含んでいます。この Javadoc は、すべての TET API メソッドの短縮された説明のみを含んでいます。詳しくは、151 ページの 10 章「TET ライブラリ API リファレンス」を参照してください。

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

- ▶ パッケージエクスプローラで Java プロジェクトを右クリックし、「**Javadoc ロケーション**」を選択します。
- ▶ 「**ブラウザ ...**」をクリックし、Javadoc が置かれているパス (TET パッケージ内) を選択します。

これらの操作を行なった後は、「**Java ブラウズ**」パースペクティブや「**ヘルプ**」メニューなどから TET の Javadoc を閲覧できます。

**例外処理** TET の Java 言語バインディングはクラス *TETException* のネイティブな Java 例外を発生させます。TET のクライアントコードは標準的な Java 例外の文法を用いる必要があります：

```
TET tet = null;

try {

...さまざまなTETメソッド呼び出し...

} catch (TETException e) {
    System.err.print("TETの例外が発生しました:\n");
    System.err.print("[ " + e.get_errnum() + " ] " + e.get_apiname() + ": " +
        e.get_errmsg() + "\n");
} catch (Exception e) {
    System.err.println(e.getMessage());
} finally {
    if (tet != null) {
        tet.delete();                /* TETオブジェクトを削除 */
    }
}
```

TET は適切な *throws* 節を宣言するので、クライアントコードはすべての可能な例外を捕捉するか、またはそれらを自身で宣言する必要があります。

## 3.6 .NET バインディング

注記 TET を .NET フレームワークで利用するためのさまざまな方式やオプションに関する詳しい情報が、ディストリビューションパッケージ内と PDFlib Web サイトにある PDFlib-in-.NET-HowTo.pdf 文書にあります。

TET の .NET エディションはそれらすべての .NET 概念に対応しています。専門用語でいえば、TET.NET エディションは .NET Framework の制御下で走る C++ クラス (非マネージ TET コアライブラリのためのマネージャラップを備えた) です。それはストロング名を持つ静的アセンブリとしてパッケージされています。この TET アセンブリ (*TET\_dotnet.dll*) は、ライブラリ本体とメタ情報を持ちます。

**TET の .NET エディションをインストール** TET 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。TET.NET MSI インストーラは、TET アセンブリと補足データファイル・サンプルを対話的にマシンにインストールします。このインストーラはまた、Visual Studio .NET の「参照の追加」ダイアログボックスの .NET タブで TET を簡単に参照できるよう登録も行います。

**エラー処理** TET.NET は .NET の例外に対応しており、実行時に問題が起きると例外を発生させ、詳細なエラーメッセージを伝達します。こうした例外を捕捉して適切にそれに対処することはクライアント側の領分です。それをしない場合には .NET Framework がその例外を捕捉し、多くの場合アプリケーションを中断させます。

例外関連情報の伝達のために TET は独自の例外クラス *TET\_dotnet.TETException* を定義しています。このクラスはメンバ *get\_errnum*・*get\_errmsg*・*get\_apiname* を持ちます。

**C++ と CLI で TET を利用** C++ で書かれた .NET アプリケーション (共通言語基盤 CLI をベースとした) は、TET C++ バインディングなしで直接 TET.NET DLL にアクセスできます。そのソースコードから下記のように TET を参照する必要があります：

```
using namespace TET_dotnet;
```

## 3.7 Objective-C バインディング

C・C++言語バインディングをObjective-C<sup>1</sup>で使用することもできますが、真正なObjective-C 用言語バインディングも用意してあります。下記の種類の TET フレームワークが利用可能です：

- ▶ Mac OS X で利用するための *TET*
- ▶ iOS で利用するための *TET\_ios*

いずれのフレームワークも、C・C++・Objective-C 用言語バインディングを含んでいます。

**Objective-C 版 TET を Mac OS X にインストール** TET を自分のアプリケーションで使うには、*TET.framework* か *TET.framework* をディレクトリ */Library/Frameworks* へコピーする必要があります。別の場所へ TET 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* を用いる必要があります。それについてはここでは記述しません。TET メソッド宣言群を含んだ *TET\_objc.h* ヘッダファイルをアプリケーションのソースコードで取り込む必要があります：

```
#import "TET/TET_objc.h"
```

または

```
#import "TET_ios/TET_objc.h"
```

**引数命名規則** TET メソッド呼び出しの際には、引数を以下の規則に従って与える必要があります：

- ▶ 第一引数の値はメソッド名の直後に、コロンキャラクタで区切って与えます。
- ▶ その後の各引数については、引数の名前と値を（これもコロンキャラクタで互いを区切って）与える必要があります。引数の名前は 151 ページの 10 章「TET ライブラリ API リファレンス」と *TET\_objc.h* にあります。

たとえば、API 記述における下記の行は：

```
int open_page(int doc, int pagenumber, String optlist)
```

下記の Objective-C となります：

```
- (NSInteger) open_page: (NSInteger) doc pagenumber: (NSInteger) pagenumber optlist: (NSString *) optlist;
```

つまりアプリケーションでは、下記のように呼び出しを行う必要があることとなります：

```
page = [tet open_page:doc pagenumber:pageno optlist:pageoptlist];
```

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

**Objective-C のエラー処理** Objective-C バインディングは、TET エラーを、ネイティブな Objective-C 例外へ翻訳します。実行時の問題の場合には、TET はクラス *TETException* 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* 機構で処理できます：

1. [developer.apple.com/library/mac/#documentation/Cocoa/Conceptual/ObjectiveC/Introduction/introObjectiveC.html](http://developer.apple.com/library/mac/#documentation/Cocoa/Conceptual/ObjectiveC/Introduction/introObjectiveC.html) 参照。

```
@try {
    ...いろいろなTET命令...
}
@catch (TETException *ex) {
    NSString * errorMessage =
        [NSString stringWithFormat:@"TETエラー %d in '%@': %@",
            [ex get_errnum], [ex get_apiname], [ex get_errmsg]];
    UIAlertView *alert = [[UIAlertView alloc] init];
    [alert setMessageText: errorMessage];
    [alert runModal];
    [alert release];
}
@catch (NSException *ex) {
    UIAlertView *alert = [[UIAlertView alloc] init];
    [alert setMessageText: [ex reason]];
    [alert runModal];
    [alert release];
}
@finally {
    [tet release];
}
```

`get_errmsg` メソッドの他に、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可能です。



## 3.8 Perl バインディング

Perl 用 TET ラップは、1 個の C ラップと 2 個の Perl パッケージモジュールから成ります。このモジュールの一つは各 TET API 関数と同等のものを Perl で提供するもので、もう一つは TET 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つ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。

**TET の Perl エディションをインストール** 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)。

TET バインディングが動作するためには、Perl インタプリタは TET Perl ラップとモジュール `tetlib_pl.pm`・`PDFlib/TET.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/tet extractor.pl
```

**Unix** Perl は、`tetlib_pl.so` (Mac OS X では `tetlib_pl.bundle`)・`tetlib_pl.pm`・`PDFlib/TET.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/tetlib_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/5.10/i686-linux
```

**Windows** TET は、Perl 5 の Windows に対する ActiveState ポートにも対応しています。これは ActivePerl とも呼ばれます。DLL `tetlib_pl.dll` とモジュール `tetlib_pl.pm`・`PDFlib/TET.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\Perl5.10\site\lib
```

**Perl の例外処理** TET 例外が発生した際には、Perl 例外が発生します。これは、`eval` シーケンスを用いてキャッチ・対処することができます：

```
eval {  
    ...いろいろなTET命令...  
};  
die "例外をキャッチしました: $@" if $@;
```

## 3.9 PHP バインディング

**TET の PHP エディションをインストール** TET は、PHP に動的に結合できる C ライブラリとして実装されています。TET は、PHP のいくつかのバージョンに対応しています。使っている PHP のバージョンに応じて、適切な TET ライブラリを、アンパックした TET アーカイブから選ぶ必要があります。

TET を PHP で利用する際のさまざまな考慮点やオプションについては、たとえば PHP 用のローダブル TET モジュールを用いるべきかどうかといった疑問も含めて、PDFlib Web サイトから入手できる文書 *PDFlib-in-PHP-HowTo* を参照してください。この文書では主に PDFlib の PHP での利用について論じていますが、その内容は、TET の PHP での利用についてもあてはまるものです。

PHP の設定を行って、外部の TET ライブラリについて PHP に知らせる必要があります。次の 2 通りの方法があります：

- ▶ 以下の行のうちのいずれかを *php.ini* に追加する：

```
extension=php_tet.dll      ; Windowsの場合
extension=php_tet.so      ; Unix・Mac OS Xの場合
extension=php_tet.sl      ; HP-UXの場合
```

PHP は、Unix の場合は *php.ini* 内の変数 *extension\_dir* で指定されたディレクトリ内で、Windows の場合はそれに加えて標準システムディレクトリ群の中でも、ライブラリを検索します。下記の一行 PHP スクリプトを用いれば、PHP TET バインディングのどのバージョンがインストールされているかを調べることができます：

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

これを実行すると、現在の PHP の設定に関する長い情報ページが表示されます。このページ上で、*tet* というタイトルのセクションを見てください。もしこのセクションに

```
PDFlib TET Support      enabled
```

というフレーズ（および TET のバージョン番号）があれば、PHP 用の TET が正しくインストールされています。

- ▶ あるいは、スクリプトの冒頭に以下の行のうちのいずれかを書くことで、TET を実行時に読み込むこともできます：

```
dl("php_tet.dll");      # Windowsの場合
dl("php_tet.so");      # Unix・Mac OS Xの場合
dl("php_tet.sl");      # HP-UXの場合
```

**PHP のファイル名処理** 非限定のファイル名（パス要素のないもの）と相対ファイル名は、PHP の Unix バージョンと Windows バージョンとで扱いが異なります：

- ▶ Unix システムの PHP では、パス要素のないファイルは、スクリプトが読み込まれたディレクトリの中で検索されます。
- ▶ Windows の PHP では、パス要素のないファイルは、PHP DLL が読み込まれたディレクトリの中でのみ検索されます。

**例外処理** PHP 5 は構造化された例外処理に対応しているので、TET の例外は PHP の例外として発生します。通常の *try/catch* 技法を用いて、TET の例外を扱うことができます：

```
try {
```

```
...いろいろなTET命令...
} catch (TETException $e) {
    print "TET例外発生:\n";
    print "[" . $e->get_errnum() . "]" . $e->get_apiname() . ": "
        $e->get_errmsg() . "\n";
}
catch (Exception $e) {
    print $e;
}
```

## 3.10 Python バインディング

**Python 版 TET 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。TET バインディングが動作するためには、Python インタプリタがTET Python ラッパを利用可能である必要があります。このラッパは、PYTHONPATH 環境変数内に挙げられているディレクトリ群の中で検索されます。Python ラッパの名前はプラットフォームによって異なります：

- ▶ Unix・Mac OS X : `tetlib_py.so`
- ▶ Windows : `tetlib_py.pyd`

**Python のエラー処理** Python バインディングは、TET 例外をネイティブな Python 例外へ翻訳します。この Python 例外は、通常の try/catch 技法で扱えます：

```
try:
    ...いろいろなTET命令...
except TETException:
    print("TET例外発生:\n[%d] %s: %s" %
          ((tet.get_errnum()), tet.get_apiname(), tet.get_errmsg()))
```

## 3.11 REALbasic バインディング

**TET の REALbasic エディションをインストール** TET は、Mac と Windows の REALbasic 開発環境 (REALbasic 2006 以上) に対応しています。

Mac でも Windows でも、TET の REALbasic エディション (*TET.rbx*) は、REALbasic アプリケーションが入っているのと同じフォルダ内の *Plugins* というフォルダへ複製する必要があります。Mac OS X では、*TET.framework* を */Library/Frameworks* へインストールする必要があります。REALbasic 用 TET は一つのパッケージで配布され、以下の種類を含んでいます：

- ▶ Mac OS X PowerPC
- ▶ Mac OS X Intel
- ▶ Windows

このことは、Mac 版・Windows 版どちらを用いても、Mac 用・Windows 用の両方のアプリケーションをビルドできることを意味します。スタンドアロンなアプリケーションが生成される際には、REALbasic は TET の適切な部分を選んで、プラットフォーム固有の部分だけを生成アプリケーション内へ含めます。

**追加の REALbasic クラス** TET は、REALbasic のオブジェクト階層構造に 2 個の新しいクラスを追加します：

- ▶ *TET* クラスはすべての TET API メソッドを含んでいます。
- ▶ *TETException* クラスは、*RuntimeException* から派生したもので、TET が発生させる例外を扱うために使うことができます (後述)。

TET は、GUI アプリケーションの作成にも、コンソールアプリケーションの作成にも用いることができます。TET はコントロールではありませんので、これは REALbasic のコントロールパレットに新しいアイコンをインストールはしません。しかし TET が利用可能なときは、REALbasic は TET クラスとその関連メソッド群を認識します。たとえば、命令補完や引数チェックは、TET API メソッド群に対して完全に動作します。

**REALbasic のエラー処理** 例外発生時には、TET はクラス *TETException* のネイティブな REALbasic 例外を発生させます。TET 例外は、標準的な REALbasic の技法で扱えます：*try/catch* ブロックを用いるか (これを推奨しますが、REALbasic 5.5 以上が必要)、あるいは *Exception* ブロックでそれを処理します。後者を以下のコードで演示します：

Exception err As TETException

```
MsgBox("TET例外がextractorサンプル内で発生しました: [" + _  
    Str(err.get_errnum()) + "]" + err.get_apiname() + ": " + err.get_errmsg())
```

この例で示しているように、REALbasic 開発者は、エラー番号、エラーメッセージ、例外を発生させた API 関数の名前を取得するための *TETException* メソッド群を用いて詳細なエラー情報を得ることができます。

## 3.12 Ruby バインディング

**Ruby 版 TET をインストール** Ruby<sup>1</sup>の拡張機構は、共有ライブラリを実行時に読み込むことによって動作します。TET 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 TET 拡張ライブラリへのアクセスを有する必要があります。このライブラリ (Windows・Unix の場合：*TET.so*、Mac OS X の場合：*TET.bundle*) がインストールされる先は通常、local ruby インストールディレクトリの *site\_ruby* ブランチ内であり、すなわち下記のような名前のディレクトリ内になります：

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

しかし、Ruby は他のディレクトリでも拡張を検索します。このディレクトリの一覧を取得するには下記の ruby 呼び出しを用います：

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

この一覧は通常、カレントディレクトリを含みますので、試験目的には、TET 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よいでしょう。

**Ruby のエラー処理** Ruby バインディングは、TET 例外をネイティブな 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
  ...いろいろなTET命令...
rescue TETException => pe
  print pe.backtrace.join("\n") + "\n"
  print "エラー [" + pe.get_errnum.to_s + "]" + pe.get_apiname + ": " + pe.get_errmsg
  print " on page pageno" if (pageno != 0)
  print "\n"
rescue Exception => e
  print e.backtrace.join("\n") + "\n" + e.to_s + "\n"
ensure
  tet.delete() if tet
end
```

**Ruby on Rails** Ruby on Rails<sup>2</sup>は、Ruby による Web 開発を支援するオープンソースフレームワークです。Ruby 用 TET 拡張は Ruby on Rails で使えます。TET 作成例を Ruby on Rails で走らせるには以下の手順に従います：

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

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

- ▶ *config/routes.rb* を編集：

1. [www.ruby-lang.org/en](http://www.ruby-lang.org/en) 参照。

2. [www.rubyonrails.org](http://www.rubyonrails.org) 参照。

```
...
# public/index.htmlの削除を忘れないように
root :to => "home#demo"
```

- ▶ `app/controllers/home_controller.rb`を以下のように編集して、PDF内容を抽出するためのTETコードを挿入します。出発点として `extractor-rails.rb` サンプルのコードを利用できます：

```
class HomeController < ApplicationController
  def demo
    require "TET"
    begin
      p = TET.new
      doc = tet.open_document(infile, docoptlist)
      ...TETアプリケーションコード、extractor-rails.rb参照...
      ...
      # そして抽出したテキストを最後に表示
      send_data text, :type => "text/plain", :disposition => "inline"
      rescue TETException => pe
      # エラー処理
    end
  end
end
```

- ▶ 自分のインストレーションをテストするには、下記コマンドでWEBrickサーバを開始させ、

```
$ rails server
```

ブラウザで `http://0.0.0.0:3000` を表示させます。PDF 文書から抽出されたテキストがブラウザに表示されます。

**TET をローカルにインストール** TET を Ruby on Rails でのみ利用したい場合で、TET を Ruby 全体で利用できるようグローバルにインストールすることができないときは、Rails ツリー内の `vendors` ディレクトリ内に TET をローカルにインストールすることも可能です。これはとりわけ、全体で利用できる Ruby 拡張をインストールする権限を有していないけれども TET を Rails で利用したいときに有用です。

## 3.13 RPG バインディング

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

**Unicode 文字列処理** TET 関数はすべて可変長の Unicode 文字列を引数として用いますので、*%ucs2* ビルトイン関数を用いてシングルバイト文字列を Unicode 文字列へ変換する必要があります。TET 関数が返す文字列はすべて可変長の Unicode 文字列です。この Unicode 文字列をシングルバイト文字列へ変換するには *%char* ビルトイン関数を用います。

**注記** *%CHAR*・*%UCS2* 関数は、カレントジョブの CCSID を用いて Unicode との文字列相互変換を行います。TET とともに提供されている作成例は、CCSID 37 (US EBCDIC) をベースとしています。他のコードページでこれらの作成例を走らせた場合には、オプションリスト内のいくつかの特殊キャラクタ ([[]]等) が正しく翻訳されない可能性があります。

文字列はすべて可変長文字列として渡されますので、さまざまな関数で明示的な文字列長をとる *length* 引数を渡してはいけません (可変長文字列の長さは文字列の先頭 2 バイトに格納されています)。

**TET に対する RPG プログラムをコンパイル・バインド** RPG で TET の関数を利用するには、コンパイルされた TET サービスプログラムが必要です。TET の定義群をコンパイル時にインクルードするには、ILE-RPG プログラムの *D* スペック内でその名前を指定する必要があります：

```
d/copy QRPGLSRC,TETLIB
```

TET のソースファイルライブラリがライブラリリストのトップにない場合は、ライブラリも指定する必要があります：

```
d/copy tetsrc/lib/QRPGLSRC,TETLIB
```

ILE-RPG プログラムのコンパイルの際には、開始前にバインディングディレクトリを作成し、そこに TET 付属の TETLIB サービスプログラムを入れておく必要があります。たとえば、ライブラリ TETLIB の中に TETLIB というバインディングディレクトリを作成したいときは、次のように指定します：

```
CRTBNDDIR BNDDIR(TETLIB/TETLIB) TEXT('TETlib Binding Directory')
```

バインディングディレクトリを作成したら、TETLIB サービスプログラムをバインディングディレクトリに追加する必要があります。たとえば、ライブラリ TETLIB 中のサービスプログラム TETLIB を、さきに作成したバインディングディレクトリに追加したいときは、次のように指定します：

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

そして、*CRTBNDRPG* コマンドを用いて (または PDM でオプション 14 を用いて) プログラムをコンパイルすれば完了です：

```
CRTBNDRPG PGM(TETLIB/EXTRACTOR) SRCFILE(TETLIB/QRPGLSRC) SRCMBR(*PGM) DFTACTGRP(*NO) BNDDIR(TETLIB/TETLIB)
```



**RPGのエラー処理** ILE-RPG で書かれた TET クライアントは、ILE-RPG が提供する *monitor/on-error/endmon* エラー処理機構を利用することができます。例外を見張るもう一つの方法は、ILE-RPG 内の \*PSSR グローバルエラー処理サブルーチンを用いることです。例外が発生した際には、ジョブログは、エラー番号、失敗した関数、例外の理由を示します。TET は、呼び出し側プログラムへエスケープメッセージを送ります。

```
c   eval      p=TET_new
*
c   monitor
*
c   callp     TET_set_option(tet:globaloptlist)
c   eval      doc=TET_open_document(tet:%ucs2(%trim(parm1)):docoptlist)
:
:
*   Error Handling
c   on-error
*   Do something with this error
*   don't forget to free the TET object
c   callp     TET_delete(tet)
c   endmon
```



# 4 TET コネクタ

TET コネクタは、TET を他のソフトウェアとインタフェースするために必要なグルーコードを提供します。TET コネクタは、TET ライブラリか TET コマンドラインツールをベースにしています。

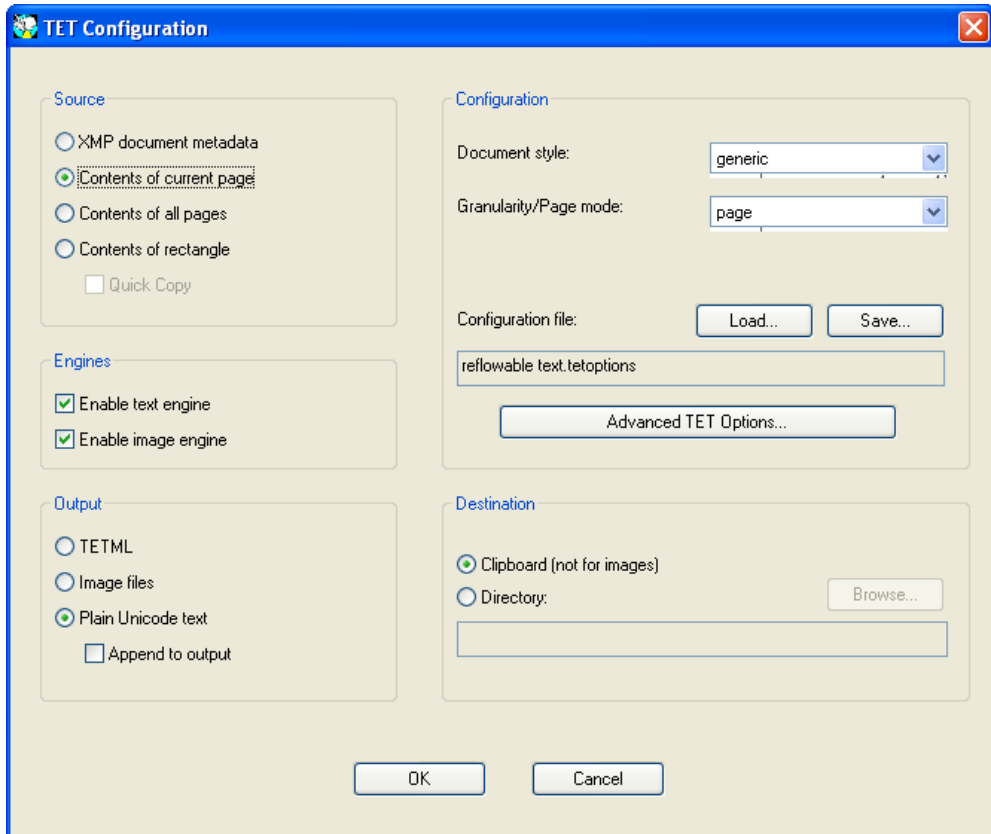
## 4.1 Adobe Acrobat 用無償 TET Plugin

この節では、Adobe Acrobat での試験と、任意の PDF 文書との TET の対話的使用のために利用できる TET の無償入手可能なパッケージングである TET Plugin を説明します。TET Plugin は、Acrobat 8/9/X Standard・Pro・Pro Extended で動作します（しかし無償の Adobe Reader では動作しません）。これは下記の場所から無償でダウンロードできます：

[www.pdflib.com/products/tet-plugin](http://www.pdflib.com/products/tet-plugin)

**TET Plugin とは** TET Plugin は、TET へのシンプルな対話的アクセスを提供します。TET Plugin は Acrobat のプラグインとして動作しますが、背後の内容抽出機能は Acrobat の機能を使わず、完全に TET をベースとしています。TET Plugin は、PDFlib TET のパワーを演示する無償ツールとして提供されています。TET は Acrobat 内蔵のテキスト・画像抽出

図 4.1  
TET Plugin の設定画面



ツールよりも強力で、多くの便利なユーザインタフェース機能も提供していますので、Acrobat 内蔵のコピー・検索機能のかわりとして有用です。テキストを抽出しようとしたときに、Acrobat ならただのゴミしか返さないような場合でも、PDFlib TET なら多くの文書をうまく処理することができます。TET Plugin は以下の機能を提供しています：

- ▶ PDF 文書内のテキストを、システムのクリップボードかディスクファイルへプレーンテキストとして複製。改良されたクリップボード制御により、コピー / 貼り付けの利用が実現しています。
- ▶ PDF を TETML に変換して、それをクリップボードかディスクファイルに入れます。
- ▶ XMP 文書メタデータをクリップボードかディスクファイルへ複製。
- ▶ 文書内の単語を検索。
- ▶ ページ上の検索文字列をすべて同時にハイライト。
- ▶ 文書内の画像を、TIFF・JPEG・JPEG 2000 のいずれかのファイルとして抽出。
- ▶ 画像の色空間・位置情報を表示。
- ▶ 必要に応じてテキスト・画像抽出を調整するための詳細な設定が利用可能です。設定セットは保存して再読込することも可能です。

**Acrobat のコピー機能にまさっている点** TET Plugin は、Acrobat 内蔵のコピー機能に対していくつかまさっている点があります：

- ▶ 出力を、さまざまなアプリケーションの要請に合わせてカスタマイズすることができます。
- ▶ Acrobat がゴミだけをクリップボードへ複製する場合にも、TET は多くの場合においてテキストを正しく解釈することができます。
- ▶ 未知のグリフ (正しい Unicode マッピングが確立できないもの) は赤色でハイライトされ、ユーザが選んだキャラクタ (クエスションマーク等) で置き換え可能です。
- ▶ TET は文書を Acrobat よりずっと速く処理します。
- ▶ 書き出す画像群を対話的に選ぶこともできますし、ページ内または文書内のすべての画像を抽出することもできます。
- ▶ 微小な画像素片を結合して、利用可能な画像にします。

## 4.2 Lucene 検索エンジン用 TET コネクタ

Lucene はオープンソースの検索エンジンです。Lucene は元来 Java プロジェクトですが、C バージョンも利用可能で、.NET 用バージョンも開発中です。Lucene について詳しくは [lucene.apache.org](http://lucene.apache.org) を参照してください。

**注記** 暗号化文書は、特定の条件下では *shrug* オプションでインデックスできます（詳しくは、67 ページの 5.1 「暗号化 PDF から内容を抽出」を参照）。これは Connector ファイル群の中で用意されますが、このオプションを手作業で有効にする必要があります。

**要件とインストール** TET ディストリビューションは、Lucene Java で PDF インデクシングを可能にするために利用できる TET コネクタを含んでいます。以下、この Lucene Java 用コネクタについて詳しく説明します。以下の要件が満たされていることが前提です：

- ▶ Lucene 3.0.x に対し JDK 1.5 以降。
- ▶ Ant ビルドツールが動作するようインストールされている。
- ▶ Lucene コア JAR ファイルを持つ Lucene ディストリビューション。TET とともに配布される Ant ビルドファイルはファイル *lucene-core-3.0.2.jar* を前提しています。このファイルは Lucene ディストリビューションに含まれています。
- ▶ Unix・Linux・Mac・Windows いずれか用の TET 配布パッケージがインストールされている。

Lucene 用 TET コネクタを実装するには、コマンドプロンプトで以下の操作を行います：

- ▶ ディレクトリ `<TET インストールディレクトリ>/connectors/lucene` へ *cd*。
- ▶ ファイル *lucene-core-x.x.x.jar* をこのディレクトリへ複製。
- ▶ グローバルな、あるいは文書関連・ページ関連の TET オプションを *TetReader.java* に追加することによって設定をカスタマイズすることもできます。たとえば、グローバルなオプションリストを用いて、リソースへの正しい *searchpath* を与えることが可能です（日中韓 CMap がデフォルトインストールと異なるディレクトリにインストールされている場合等に）。

*PdfDocument.java* モジュールは、ディスクファイルかメモリバッファ（Web クローラによって与えられた等）に格納されている PDF 文書を処理する方法を演示しています。クラス *com.pdflib.tet.lucene.IndexPdfFiles* 内で、Lucene エンジンのターゲットバージョンを *LUCENE\_VERSION* 変数でカスタマイズすることも可能です。

- ▶ コマンド *ant index* を実行。これはソースコードをコンパイルし、`<TET インストールディレクトリ>/bind/data` ディレクトリ内に含まれる PDF ファイル群に対してインデクサを走らせます。
- ▶ コマンドライン検索を開始するには、コマンド *ant search* を実行します。ここでは、Lucene クエリ言語でクエリを入力できます。

**TET と Lucene をコマンドライン検索クライアントでテスト** 以下のサンプルセッションは、TET と Lucene でインデクシングを行い、生成されたインデックスを Lucene コマンドラインクエリツールでテストする場合のコマンドと出力を演示しています。この操作はコマンド *ant index* を実行することから始まります：

```
devserver (1)$ ant index
Buildfile: build.xml
...
index:
    [java] adding ../data/Whitepaper-XMP-metadata-in-PDFlib-products.pdf
    [java] adding ../data/Whitepaper-PDFA-with-PDFlib-products.pdf
```

```
[java] adding ../data/FontReporter.pdf
[java] adding ../data/TET-PDF-IFilter-datasheet.pdf
[java] adding ../data/PDFlib-datasheet.pdf
[java] 1255 total milliseconds
```

BUILD SUCCESSFUL

Total time: 2 seconds

devserver (1)\$ ant search

Buildfile: build.xml

compile:

search:

```
[java] Enter query:
```

PDFlib

```
[java] Searching for: pdflib
```

```
[java] 5 total matching documents
```

```
[java] 1. ../data/PDFlib-datasheet.pdf
```

```
[java] Title: PDFlib, PDFlib+PDI, Personalization Server Datasheet
```

```
[java] 2. ../data/Whitepaper-PDFA-with-PDFlib-products.pdf
```

```
[java] Title: Whitepaper: Creating PDF/A with PDFlib
```

```
[java] 3. ../data/FontReporter.pdf
```

```
[java] Title: PDFlib FontReporter 1.3 Manual
```

```
[java] 4. ../data/TET-PDF-IFilter-datasheet.pdf
```

```
[java] Title: PDFlib TET PDF IFilter Datasheet
```

```
[java] 5. ../data/Whitepaper-XMP-metadata-in-PDFlib-products.pdf
```

```
[java] Title: Whitepaper: XMP Metadata support in PDFlib Products
```

```
[java] Press (q)uit or enter number to jump to a page.
```

q

```
[java] Enter query:
```

title:FontReporter

```
[java] Searching for: title:fontreporter
```

```
[java] 1 total matching documents
```

```
[java] 1. ../data/FontReporter.pdf
```

```
[java] Title: PDFlib FontReporter 1.3 Manual
```

```
[java] Press (q)uit or enter number to jump to a page.
```

q

```
[java] Enter query:
```

BUILD SUCCESSFUL

Total time: 57 seconds

2つのクエリが実行されています：1つはテキスト内の単語 **PDFlib** に対して、もう1つは **title** フィールド内の単語 **FontReporter** に対してです。結果ページングモードを抜けて次のクエリを始められるようにするには **q** を入力する必要がある点に留意してください。

Ant *build.xml* ファイル内のパスとファイル名はすべて、プロパティを通じて定義されています。これは、このファイルをさまざまな環境で使えるようにするためです。すなわち、プロパティ群をコマンドラインで与えることもできますし、上書きさせたいプロパティ群をファイル *build.properties* 内に、あるいはプラットフォーム個別のプロパティをファイル *windows.properties* か *unix.properties* 内に記入することもできます。たとえば、Ant を下記のように呼び出せば、*/tmp* 下にインストールされている Lucene JAR ファイルでサンプルを実行することができます：

ant -Dlucene.jar=/tmp/lucene-core-x.x.x.jar index

**TET と Lucene を Demo Web アプリケーションでテスト** Lucene デモ Web アプリケーションは、Tomcat や GlassFish をはじめとする任意の Java サーブレットコンテナ上に展開することができます。必要な操作は、Lucene に付いて来る HTML 文書で説明されています。オンラインで [lucene.apache.org/java/3\\_0\\_2/demo3.html](http://lucene.apache.org/java/3_0_2/demo3.html) でも得られます。

そのページのステップ *Configuration* に留意してください。ここでインデックスの場所を Web アプリケーションに知らせるために、それをファイル *configuration.jsp* に記入する必要があります。ここで追加するパスは、Ant が Lucene インデックスの場所に関するプロパティを上書きせず動作させられている場合は `<TET インストールディレクトリ >/bind/lucene/index` となるでしょう。

**メタデータフィールドをインデックス** Lucene 用 TET コネクタは以下のメタデータフィールドをインデックスします：

- ▶ *path* (トークン化フィールド)：文書のパス名
- ▶ *modified* (DateField)：最終更新日
- ▶ *contents* (Reader フィールド)：文書の全テキスト内容
- ▶ Title・Subject・Author 等、定義済み・カスタムの PDF 文書情報項目すべて。文書情報項目は、TET に内蔵されている pCOS インタフェースで取得することができます (pCOS について詳しくは pCOS パスリファレンスを参照してください)。たとえば

```
String objType = tet.pcos_get_string(tetHandle, "type:/Info/Subject");
if (!objType.equals("null"))
{
    doc.add(new Field("summary", tet.pcos_get_string(tetHandle,
        "/Info/Subject"), Field.Store.YES, Field.Index.ANALYZED));
}
```

- ▶ *font*：PDF 文書内のすべてのフォントの名前

*PdfDocument.java* 内で、インデックスする文書情報項目のセットを変更、あるいは pCOS に基づいて情報を追加することによって、メタデータフィールドをカスタマイズすることも可能です。

**PDF ファイル添付** Lucene 用 TET コネクタは、文書内のすべての PDF ファイル添付を再帰的に処理し、各添付のテキストとメタデータをインデックスできるよう Lucene 検索エンジンに与えます。これによって、検索テキストがメイン文書内になく添付内にあるときでも、検索ヒットが生成されます。再帰的添付横断はとりわけ、PDF パッケージ・ポートフォリオに対して重要です。

## 4.3 Solr 検索サーバ用 TET コネクタ

Solr は高パフォーマンスなオープンソースのエンタプライズ検索サーバで、Lucene 検索ライブラリをベースとしています。XML/HTTP・JSON/Python/Ruby API を有し、ヒットハイライト・ファセット検索・キャッシュ化・レプリケーション・Web 管理インタフェースをそなえています。Java サブレットコンテナ内で動作します ([lucene.apache.org/solr](http://lucene.apache.org/solr) を参照)。

Solr は Lucene コアエンジンを取り巻く追加レイヤとしてふるまいます。インデックスされたデータをシンプルな XML 形式で受け付けます。Solr 入力 は TETML をベースに非常に簡単に生成できます。TETML は TET が生成する一種の XML です。Solr 用 TET コネクタは、TETML を Solr が受け付ける XML 形式へ変換する XSLT スタイルシートから成ります。このスタイルシートのための TETML 入力は、TET ライブラリか TET コマンドラインツールで生成できます (131 ページの 9.1 「TETML を生成」を参照)。

**注記** 暗号化文書は、特定の条件下では *shrug* オプションでインデックスできます (詳しくは、67 ページの 5.1 「暗号化 PDF から内容を抽出」を参照)。暗号化文書をインデックスするには、Solr のための TETML 入力を生成する際に、TET ライブラリか TET コマンドラインツールでこのオプションを有効にする必要があります。

**メタデータフィールドをインデックス** Solr 用 TET コネクタは、すべての標準文書情報フィールドをインデックスします。各フィールドのキーがフィールド名として用いられます。

**PDF ファイル添付** Solr 用 TET コネクタは、文書内のすべての PDF ファイル添付を再帰的に処理し、各添付のテキストとメタデータをインデックスできるよう検索エンジンに与えます。これによって、検索テキストがメイン文書内になく添付内にあるときでも、検索ヒットが生成されます。再帰的添付横断はとりわけ、PDF パッケージ・ポートフォリオに対して重要です。

**TETML を変換するための XSLT スタイルシート** *solr.xsl* スタイルシートは、*glyph* 以外の任意のモードの TETML 入力を受け付けます。これは、検索サーバに入力データを与えるために必要な XML を生成します。文書情報項目群は、その情報項目の名前 (に文字列値であることを示す *\_s* 接尾辞をつけたもの) を保持したフィールドとして与えられ、メインテキストは多数のテキストフィールドで与えられます。文書内の PDF 添付 (PDF パッケージ・ポートフォリオを含む) は再帰的に処理されます：

```
<?xml version="1.0" encoding="UTF-8"?>
<add>
<doc>
<field name="id">PDFlib-FontReporter-E.pdf</field>
<field name="Author_s">PDFlib GmbH</field>
<field name="CreationDate_s">2008-07-08T15:05:39+00:00</field>
<field name="Creator_s">FrameMaker 7.0</field>
<field name="ModDate_s">2008-07-08T15:05:39+00:00</field>
<field name="Producer_s">Acrobat Distiller 7.0.5 (Windows)</field>
<field name="Subject_s">PDFlib FontReporter</field>
<field name="Title_s">PDFlib FontReporter 1.3 Manual</field>
<field name="text">PDFlib</field>
<field name="text">GmbH</field>
<field name="text">Munchen</field>
...
```



## 4.4 Oracle 用 TET コネクタ

Oracle 用 TET コネクタは、TET を Oracle データベースに結合して、PDF 文書を Oracle Text でインデックスしクエリできるようにします。PDF 文書は、データベース内のそのパス名を通じて参照することもできますし、データベース内に BLOB として直接格納することもできます。

**注記** 暗号化文書は、特定の条件下では *shrug* オプションでインデックスできます（詳しくは、67 ページの 5.1 「暗号化 PDF から内容を抽出」を参照）。これは Connector ファイル群の中で用意されますが、このオプションを手作業で有効にする必要があります。

**要件とインストール** TET コネクタは Oracle 10i と Oracle 11g でテストされています。TET コネクタを利用するためには、データベースを作成する際に **AL32UTF8** データベース文字集合を指定する必要があります。これは、Oracle Express の Universal 版ではつねにそうなります（しかし Western European 版では異なります）。AL32UTF8 は Oracle が推奨しているデータベース文字集合であり、TET で PDF 文書をインデックスする場合にも最良の動作をします。ただし、以下の方式のいずれかに従えば、他の文字集合で TET を Oracle Text に接続することも可能です：

- ▶ Oracle Text 11.1.0.7 からは、必要な文字集合変換をデータベースが行えます。下記にある Oracle Text 11.1.0.7 文書の「Using USER\_FILTER with Charset and Format Columns」節を参照してください：

[download.oracle.com/docs/cd/B28359\\_01/text.111/b28304/cdatadic.htm#sthref497](http://download.oracle.com/docs/cd/B28359_01/text.111/b28304/cdatadic.htm#sthref497)

- ▶ Oracle Text 11.1.0.6 までは、TET フィルタスクリプトが生成する UTF-8 テキストをデータベース文字集合へ変換する必要があります。これは、文字集合変換コマンドを *tetfilter.sh* に追加することによって実現できます：

Unix：*iconv*（オープンソースソフトウェア）か *uconv*（無償の ICU Unicode ライブラリに含まれています）を呼び出します。

Windows：*tetfilter.bat* 内の適切なコードページコンバータを呼び出します。

Oracle 用 TET コネクタを活用できるようにするには、以下のようにして TET フィルタスクリプトを Oracle から利用可能にする必要があります：

- ▶ TET フィルタスクリプトを、Oracle がそれを見つけることができるディレクトリへ複製：

Unix：*connectors/Oracle/tetfilter.sh* を *\$ORACLE\_HOME/ctx/bin* へ複製

Windows：*connectors/Oracle/tetfilter.bat* を *%ORACLE\_HOME%\bin* へ複製

- ▶ TET フィルタスクリプト（それぞれ *tetfilter.sh*・*tetfilter.bat*）内の *TETDIR* 変数が必ず TET インストールディレクトリを指しているようにします。
- ▶ 必要に応じて、追加の TET オプション群を、グローバル・文書レベル・ページレベルのいずれかについて *TETOPT*・*DOCOPT*・*PAGEOPT* 変数内で与えることもできます（オプションリストについて詳しくは、151 ページの 10 章「TET ライブラリ API リファレンス」を参照）。これは特に TET ライセンスキーを与えるために有用です。例：

```
TETOPT="license=aaaaaaa-bbbbbb-cccccc-dddddd-eeeeee"
```

TET ライセンスキーを与えるための他の選択肢については 9 ページの 0.2 「TET ライセンスキーを適用」を参照してください。

**Oracle ユーザに権限を付与** 以下の例はいずれも、Oracle ユーザがインデックスを作成しクエリする適切な権限を持っていることを前提としています。以下のコマンド群は、

ユーザ *HR* に適切な権限群を付与します（これらのコマンドは、*system* として発行する必要があります、かつ適切に調整する必要があります）：

```
SQL> GRANT CTXAPP TO HR;
SQL> GRANT EXECUTE ON CTX_CLS TO HR;
SQL> GRANT EXECUTE ON CTX_DDL TO HR;
SQL> GRANT EXECUTE ON CTX_DOC TO HR;
SQL> GRANT EXECUTE ON CTX_OUTPUT TO HR;
SQL> GRANT EXECUTE ON CTX_QUERY TO HR;
SQL> GRANT EXECUTE ON CTX_REPORT TO HR;
SQL> GRANT EXECUTE ON CTX_THES TO HR;
```

**例 A : PDF 文書のパス名をデータベースに格納** この例は、インデックスされた PDF 文書群へのファイル名参照をデータベース内に格納します。以下のように操作します：

- ▶ コマンドプロンプトで下記のディレクトリへ移動：

```
<TETインストレーションディレクトリ>/connectors/Oracle
```

- ▶ *tetsetup\_a.sql* スクリプト内の *tetpath* 変数を、TET がインストールされているディレクトリを指すように変えます。
- ▶ データベースを用意:Oracle の *sqlplus* プログラムを使って、テーブル *pdftable\_a* を作成し、このテーブルに PDF 文書群のパス名を記入し、インデックス *tetindex\_a* を作成します（なお、*tetsetup\_a.sql* スクリプトの内容は若干プラットフォーム依存です。パス文法が異なるためです）：

```
SQL> @tetsetup_a.sql
```

- ▶ データベースを、インデックスを用いてクエリ：

```
SQL> select * from pdftable_a where CONTAINS(pdffile, 'Whitepaper', 1) > 0;
```

- ▶ インデックスを更新（文書を追加した後に必要です）：

```
SQL> execute ctx_ddl.sync_index('tetindex_a')
```

- ▶ データベースをクリーンアップ（インデックスとテーブルを削除）することもできます：

```
SQL> @tetcleanup_a.sql
```

**例 B : PDF 文書を BLOB としてデータベースに格納してメタデータを追加** この例は、PDF 文書本体を BLOB としてデータベースに格納します。PDF データに加えて、いくつかのメタデータを pCOS インタフェースで抽出し、それ用のデータベース列に格納します。*tet\_pdf\_loader* Java プログラムは、PDF 文書群を BLOB としてデータベースに格納します。メタデータ処理を演示するため、このプログラムは、pCOS インタフェースを用いて、文書タイトル（pCOS パス */Info/Title* を通じて）と文書内のページ数（pCOS パス *length:pages* を通じて）を抽出します。この文書タイトルとページ数はデータベース内の別コラムに格納されます。この例を動作させるには以下のように操作します：

- ▶ コマンドプロンプトで下記のディレクトリへ移動：

```
<TETインストレーションディレクトリ>/connectors/Oracle
```

- ▶ データベースを用意:Oracle の *sqlplus* プログラムを使って、テーブル *pdftable\_b* とそのインデックス *tetindex\_b* を作成します：

```
SQL> @tetsetup_b.sql
```

- ▶ データベースに内容を入れます: このテーブルに、JDBC を通じて PDF 文書とメタデータを入れます (これはストアドプロシージャではできない点に留意)。TET パッケージとともに供給されている ant ビルドファイルは、Oracle JDBC ドライバに対する *ojdbc14.jar* ファイルが *tet\_pdf\_loader.java* ソースコードと同じディレクトリにあると前提しています。適切な JDBC 接続文字列を *ant* コマンドで指定します。このビルドファイルは、すべてのプロパティの記述を含んでおり、これを用いて Ant ビルドのオプション群を指定することができます。これらのオプションに対する値をコマンドラインで与えることができます。下記の例では、ホスト名として *localhost*、ポート番号 1521、データベース名として *xe*、ユーザ名・パスワードとして *HR* を用いています (自分のデータベース設定に合わせて適切に変えてください) :

```
ant -Dtet.jdbc.connection=jdbc:oracle:thin:@localhost:1521:xe ←  
-Dtet.jdbc.user=HR -Dtet.jdbc.password=HR
```

- ▶ インデックスを更新 (最初と、文書を追加した後に必要です) :

```
SQL> execute ctx_ddl.sync_index('tetindex_b')
```

- ▶ データベースを、インデックスを用いてクエリ :

```
SQL> select * from pdftable_b where CONTAINS(pdffile, 'Whitepaper', 1) > 0;
```

- ▶ データベースをクリーンアップ (インデックスとテーブルを削除) することもできます :

```
SQL> @tetcleanup_b.sql
```

## 4.5 Microsoft 製品用 TET PDF IFilter

この節では、PDFlib TET をベースとして構築された別製品である TET PDF IFilter を説明します。TET PDF IFilter の詳しい情報と配布パッケージは [www.pdfli.com/products/tet-pdf-ifilter](http://www.pdfli.com/products/tet-pdf-ifilter) で入手可能です。

TET PDF IFilter は、非商用デスクトップ用途については無償で利用可能です。デスクトップシステム上での商用利用とサーバ上での展開には商用ライセンスが必要です。

**PDFlib TET PDF IFilter とは** TET PDF IFilter は、PDF 文書からテキストとメタデータを抽出し、それを Windows 上の検索ソフトウェアで利用可能にします。これによって、PDF 文書をローカルデスクトップや企業サーバ、あるいは Web で検索することが可能になります。TET PDF IFilter は、特許を受けた PDFlib Text Extraction Toolkit (TET) をベースとしています。TET は、PDF 文書から確実にテキストを抽出するための定評ある開発者向け製品です。

TET PDF IFilter は、Microsoft の IFilter インデクシングインタフェースの堅牢な実装です。これは SharePoint や SQL Server 等、IFilter インタフェースに対応しているすべての検索製品とともに動作します。そうした製品は HTML 等、特定のファイル形式に対する形式個別のフィルタプログラムを用いており、このフィルタプログラムを IFilter と呼びます。TET PDF IFilter はそのようなプログラムの一つで、PDF 文書に特化したものです。文書を検索するためのインタフェースは、Windows Explorer や Web やデータベースのフロントエンドであってもよいですし、クエリスクリプトやカスタムアプリケーションとすることもできます。対話的検索だけでなく、ユーザインタフェース一切なしでクエリをプログラマ的に発することもできます。

**特長** TET PDF IFilter は以下の利点を提供します：

- ▶ 欧文テキスト、日本語・中国語・韓国語（日中韓）テキスト、アラビア文字・ヘブライ文字等の右書き言語に対応
- ▶ 暗号化文書をインデックスし、また、Acrobat が失敗する PDF からテキストを抽出
- ▶ Unicode 字形統合・分解・正規化に対応
- ▶ 展開：スレッドセーフ、高速・堅牢、32・64 ビット版
- ▶ 自動用字系・言語検出による検索向上

**エンタプライズ PDF 検索** TET PDF IFilter は、完全にスレッドセーフなネイティブ 32・64 ビット版として利用可能です。TET PDF IFilter と以下の製品を用いて、エンタプライズ PDF 検索ソリューションを実装できます：

- ▶ Microsoft SharePoint Server および SharePoint 用 FAST サーバ
- ▶ Microsoft Search Server
- ▶ Microsoft SQL Server
- ▶ Microsoft Exchange Server
- ▶ Microsoft Site Server

TET PDF IFilter は、IFilter インタフェースに対応している他のすべての Microsoft・サードパーティ製品とともに利用できます。

**デスクトップ PDF 検索** TET PDF IFilter を利用すると、たとえば以下のような製品とともに、デスクトップ PDF 検索を実装することも可能です：

- ▶ Windows Search : Windows Vista/7 に内蔵されています。Windows XP 用の無償アドオンとしても利用可能です。

- ▶ Windows Indexing Service

TET PDF IFilter は、デスクトップオペレーティングシステム上での非商用利用については無償ですので、気軽にテストや評価を行うことができます。

**受け入れ可能な PDF 入力** TET PDF IFilter は、あらゆる種類の PDF 入力に対応していません：

- ▶ Acrobat X までのすべての PDF バージョン (ISO 32000 も含め)
- ▶ 文書を開くパスワードを必要としない暗号化 PDF
- ▶ 破損した PDF 文書は修復されます。

**Unicode 後処理** TET PDF IFilter はさまざまな Unicode 後処理に対応しており、これを利用して検索結果を向上させることができます：

- ▶ 字形統合：キャラクタに対して温存・除去・置換のいずれかを行います。たとえば句読点や、無関係な用字系のキャラクタを除去することができます。
- ▶ 分解：一つのキャラクタを、等価な他のキャラクタないしキャラクタ列へ置き換えます。たとえば漢字を、それと正準等価な Unicode キャラクタへ置き換えることができます。
- ▶ テキストを、4 種類の Unicode 正規形のいずれへも変換できます。たとえば、データベースの要請に合うよう NFC 形式を出力することができます。

**国際化** 欧文テキストに加えて、TET PDF IFilter は日本語・中国語・韓国語 (日中韓) テキストに完全対応しています。すべての日中韓エンコーディングが認識されます。横書き・縦書きに対応しています。テキストのロケール ID (言語・地域識別子) の自動検出が、Microsoft の単語区切り・語幹処理アルゴリズムの結果を向上させ、このことはとりわけ東アジアテキストについて重要です。

ヘブライ文字・アラビア文字といった右書き言語にも対応しています。位置依存表示形は正規化され、テキストは論理順に発出されます。

**PDF は単なるページの寄せ集めではない** TET PDF IFilter は PDF 文書を、単なるページ群だけの他にも多くの情報を含む可能性のあるコンテナとして扱います。TET PDF IFilter は PDF 文書内の関連する項目をすべてインデックスします：

- ▶ ページ内容
- ▶ しおり内のテキスト
- ▶ メタデータ (後述)
- ▶ 埋め込まれた PDF と PDF パッケージ / ポートフォリオは、埋め込まれているすべての PDF 文書内のテキストが検索できるよう、再帰的に処理されます。

**XMP メタデータと文書情報** TET PDF IFilter の高度なメタデータ実装は、Windows のメタデータのためのプロパティシステムに対応しています。これは XMP メタデータと、標準・カスタム文書情報項目をインデックスします。メタデータインデクシングはいくつかのレベルで設定できます：

- ▶ 文書情報項目群と Dublin Core フィールド群、およびその他の広く用いられる XMP プロパティ群は、同等の Windows プロパティへマップされます。例：**Title**・**Subject**・**Author**。
- ▶ TET PDF IFilter は、有用な PDF 独自の擬似プロパティ群を追加します。例：ページサイズ・PDF/A 準拠レベル・フォント名。
- ▶ すべての定義済み XMP プロパティ群を検索できます。

- ▶ ユーザ定義 XMP プロパティ群を検索できます。例：企業独自の分類プロパティ、PDF/A 拡張スキーマ。

TET PDF IFilter は、メタデータを全文テキストインデックス内に統合することもできます。結果として、メタデータ対応を持たない全文テキスト検索エンジン（SQL Server 等）であっても、メタデータの検索が可能になります。

## 4.6 Apache TIKA ツールキット用 TET コネクタ

TIKA は、オープンソースの「既存のパースライブラリを用いてさまざまな文書からメタデータ・構造化テキスト内容を検出・抽出するためのツールキット」です。TIKA の詳細については [tika.apache.org](http://tika.apache.org) を参照してください。Tika 用 TET コネクタは、Tika に設定されているデフォルトの PDF パーサを置き換え、TET を PDF 形式用のパーサとして接続します。TET コネクタは以下の項目を Tika に与えます：

- ▶ 全ページの非整形のテキスト内容
- ▶ 定義済み・カスタムの文書情報フィールド群
- ▶ 文書のページ数

**注記** 暗号化文書は、特定の条件下では *shrug* オプションでインデックスできます（詳しくは、67 ページの 5.1 「暗号化 PDF から内容を抽出」を参照）。これは Connector ファイル群の中で用意されますが、このオプションを手作業で有効にする必要があります。TETPDFParser.java ではこれに加えて、*shrug* オプションでは不足の場合にパスワードを与える方法を提供しています。

**要件とインストール** TET ディストリビューションは、Tika ツールキット用 TET コネクタを含んでいます。以下の説明において `<tet-dir>` は、TET パッケージをアンパックしたディレクトリを意味します。下記の要件を満たす必要があります：

- ▶ JDK 1.5 以降
- ▶ *Ant* ビルドツールの動作中のインストール
- ▶ Unix・Linux・Mac・Windows いずれか用のインストール済み TET ディストリビューションパッケージ。
- ▶ 名前 *tika-app-1.x.jar* の Tika 用プレビルト JAR ファイル。このファイルのためのダウンロード情報は下記の場所にあります：

[tika.apache.org/download.html](http://tika.apache.org/download.html)

**Tika 用 TET コネクタをビルドしてテスト** Tika 用 TET コネクタをビルドしてテストするには以下の手順に従います：

- ▶ *tika-app-1.x.jar* をディレクトリ `<tet-dir>/connectors/Tika` へコピー。
- ▶ `<tet-dir>/connectors/Tika` へ移動し、Tika 用 TET コネクタをビルド：

```
ant
```

Tika jar ファイルの名前が *tika-app-1.0.jar* でないときは、jar ファイル名をコマンドラインで与える必要があります：

```
ant -Dtika-app.jar=tika-app-1.5.jar
```

- ▶ ビルドファイルは、Tika 用 TET コネクタでのテストを走らせるターゲットを含んでいません：

```
ant test
```

このコマンドは、`<tet-dir>/bind/data/FontReporter.pdf` の内容を XHTML として標準出力に生成するはずですが、自分の選んだ PDF ファイルでテストを行うには、下記のように Ant プロパティ *test.inputfile* をコマンドラインで与えます：

```
ant -Dtest.inputfile=/自分の/ファイル/への/パス.pdf test
```

暗号化文書にパスワードを与える機能については下記のようにテストできます：

```
ant -Dtest.inputfile=<暗号化ファイル.pdf> -Dtest.outputfile=<出力ファイル名> ←  
-Dtest.password=<パスワード> api-test
```

- ▶ Tika用TETコネクタが実際にMIMEタイプ *application/pdf* に対して用いられるかどうかを検証するには、<tet-dir>/connectors/Tika ディレクトリで、Unix・Mac OS X システムの場合は下記コマンドを実行します：

```
java -Djava.library.path=<tet-dir>/bind/java -classpath ←  
<tet-dir>/bind/java/TET.jar:tika-app-1.0.jar:tet-tika.jar ←  
org.apache.tika.cli.TikaCLI --list-parser-details
```

Windows の場合：

```
java -Djava.library.path=<tet-dir>/bind/java -classpath ←  
<tet-dir>/bind/java/TET.jar;tika-app-1.0.jar;tet-tika.jar ←  
org.apache.tika.cli.TikaCLI --list-parser-details
```

生成出力内に下記記述が現れるはずです：

```
com.pdflib.tet.tika.TETPDFParser  
application/pdf
```

- ▶ Tika GUIアプリケーションをTETコネクタとともに動作させるには、ディレクトリ <tet-dir>/connectors/Tika で下記コマンドを実行します：

Unix・Mac OS X システムの場合：

```
java -Djava.library.path=<tet-dir>/bind/java -classpath ←  
<tet-dir>/bind/java/TET.jar:tika-app-1.0.jar:tet-tika.jar ←  
org.apache.tika.cli.TikaCLI
```

Windows の場合：

```
java -Djava.library.path=<tet-dir>\bind\java -classpath ←  
<tet-dir>\bind\java\TET.jar;tika-app-1.0.jar;tet-tika.jar ←  
org.apache.tika.cli.TikaCLI
```

**Tike 用 TET コネクタをカスタマイズ** Tikeコネクタは、*TETPDFParser.java* ソースモジュール内で下記のようにカスタマイズすることもできます：

- ▶ *DOC\_OPT\_LIST* 変数に文書オプション群を追加。例：shrug オプションで暗号化文書を処理。
- ▶ *PAGE\_OPT\_LIST* 変数にページオプション群を追加。
- ▶ *SEARCHPATH* 変数で、日中韓 CMap などのリソースへの検索パスをカスタマイズ。あるいは、PDF 文書を処理する際に *tet.searchpath* プロパティを与えることもできます。



## 4.7 MediaWiki 用 TET コネクタ

MediaWiki は無償のウィキソフトウェアであり、Wikipedia をはじめとする多くのコミュニティ Web サイトを動作させるために利用されています。MediaWiki に関する詳しい情報は下記にあります：

[www.mediawiki.org/wiki/MediaWiki](http://www.mediawiki.org/wiki/MediaWiki)

**注記** 暗号化文書は、特定の条件下では *shrug* オプションでインデックスできます（詳しくは、67 ページの 5.1 「暗号化 PDF から内容を抽出」を参照）。これは Connector ファイル群の中で用意されますが、このオプションを手作業で有効にする必要があります。

**要件とインストール** TET ディストリビューションは、MediaWiki サイトへアップロードされる PDF 文書をインデックスするために利用できる TET コネクタを含んでいます。MediaWiki はネイティブに PDF 文書に対応はしていませんが、PDF を「画像」としてアップロードすれば受け付けます。MediaWiki 用 TET コネクタは、すべての PDF 文書を、それがアップロードされる際にインデックスします。MediaWiki 内にすでに存在している PDF 文書はインデックスされません。以下の必要条件を満たしている必要があります：

- ▶ PHP 5.0 以上
- ▶ MediaWiki 1.11.2 以上（これ以外のバージョンについては後述）
- ▶ Unix・Linux・Mac・Windows のいずれか用の TET 配布パッケージ

MediaWiki 用 TET コネクタを実装するには以下の操作を行います：

- ▶ PHP 用 TET バインディングを、42 ページの 3.9 「PHP バインディング」の説明のようにインストール。
- ▶ <TET インストールディレクトリ>/connectors/MediaWiki/PDFIndexer.php を <MediaWiki インストールディレクトリ>/extensions/PDFIndexer/PDFIndexer.php へ複製。
- ▶ 日中韓テキストへの対応が必要な場合は、<TET インストールディレクトリ>/resource/cmap 内の CMap ファイル群を <MediaWiki インストールディレクトリ>/extensions/PDFIndexer/resource/cmap へ複製。
- ▶ MediaWiki 設定ファイル *LocalSettings.php* に以下の行群を追加：

```
# アップロードされるPDFをインデックスして検索可能にします
include("extensions/PDFIndexer/PDFIndexer.php");
```

- ▶ PDF 文書をアップロードする際に警告が出ないようにするには、<MediaWiki インストールディレクトリ>/includes/DefaultSettings.php に以下の行群を追加して *.pdf* を既知のファイル種別拡張子にすることを推奨します：

```
/**
 * これはアップロードするファイルの好ましい拡張子のリストです。このリストにない拡張子
 * を持つファイルをアップロードすると警告が発生します。
 */
$wgFileExtensions = array( 'png', 'gif', 'jpg', 'jpeg', 'pdf' );
```

**MediaWiki 用 TET コネクタはどのように動作するか** MediaWiki 用 TET コネクタは、PHP モジュール *PfIndexer.php* から成ります。これは MediaWiki の定義済みフックの一つを用いて、新規 PDF 文書がアップロードされる際にいつも呼び出されるようフックアップされます。これは PDF 文書からテキストとメタデータを抽出して、それを、アップロードされる文書にオプションに付属する、ユーザが与えるコメントの後に追加します。このテキストは、ユーザが文書コメントを表示する際に表示されないよう、HTML コメント内

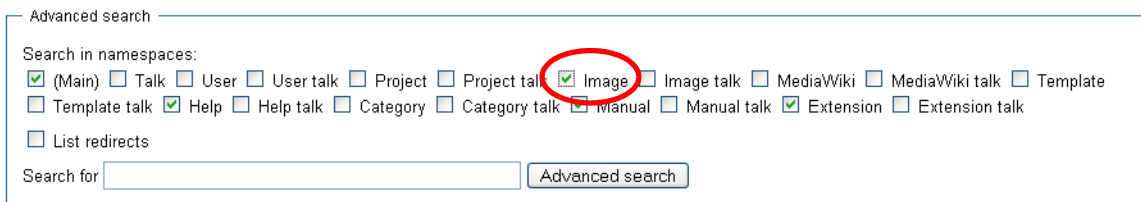


図 4.2 MediaWiki 内で PDF 文書を検索

に隠されています。MediaWiki はコメントの内容全文をインデックスします（隠されている全文テキストも含めて）ので、PDF のテキスト内容もインデックスされます。インデックス用テキストは以下のように構築されます：

- ▶ TET コネクタは、すべての文書情報フィールドの値をインデックスへ与えます。
- ▶ すべてのページのテキスト内容が抽出されて結合されます。
- ▶ 抽出されたテキストのサイズが制限未満であれば、それはまるごとインデックスへ与えられます。この方式の利点は、検索結果が検索語を文脈の中で表示する点です。
- ▶ 抽出されたテキストのサイズが制限を超えている場合は、テキストはユニークな単語群へ減量されます（すなわち、同一の単語が複数回出現しているものについて、その単語を 1 回だけに減量されます）。
- ▶ 減量したテキストのサイズが制限未満であれば、それはインデックスへ与えられます。そうでない場合には、それは切り落とされます。すなわち、その文書の終わりのほうのテキストはインデックスされなくなります。

定義済みの制限は 512 KB ですが、これは PDFIndexer.php 内で変更することもできます。上記のサイズテストの一つが制限に触れた場合には、MediaWiki のログ記録が有効にされているならば、警告メッセージが MediaWiki の *DebugLogFile* へ書き込まれます。

**PDF 文書を検索** PDF 文書は MediaWiki によって画像として扱われますので、*Image* 名前空間内を検索する必要があります。これは、*Advanced search* ダイアログ（図 4.2 参照）の名前空間の一覧にある *Image* チェックボックスをオンにすることで実現できます。ただし、この設定は *LocalSettings.php* 設定ファイル内で以下のように有効にすることも可能です：

```
$wgNamespacesToBeSearchedDefault = array(  
    NS_MAIN          => true,  
    NS_IMAGE        => true,  
);
```

検索結果は、検索語を含む文書の一覧を表示します。テキスト全文がインデックスされている場合（文書が長くて単語リストへ減量された場合ではなく）には、検索語の前後にいくつかの語が表示されて文脈を示します。PDF テキスト内容は MediaWiki インデックスに HTML 形式で与えられますので、テキストの頭に行番号が表示されます。これらの行番号は PDF 文書については意味がありませんので、無視して差し支えありません。

**メタデータフィールドをインデックス** MediaWiki 用 TET コネクタは、すべての標準文書情報フィールドをインデックスします。各フィールドの値が、検索で利用できるよう、インデックスへ与えられます。MediaWiki はメタデータベースの検索に対応していませんので、文書情報項目を直接検索することはできず、全文テキストの一部としてのみ情報項目を検索できます。

# 5 設定

## 5.1 暗号化 PDF から内容を抽出

**PDF のセキュリティ機能** PDF 文書は、以下の保護機能を提供するパスワードセキュリティによって保護することができます：

- ▶ ファイルを開いて閲覧するためにユーザパスワード（開くパスワードともいいます）が必要。
- ▶ 権限群やユーザ／マスタパスワードといったセキュリティ設定群のうちのいずれかを変更するためにマスタパスワード（所有者／権限パスワードともいいます）が必要。ユーザパスワードとマスタパスワードのあるファイルは、いずれかのパスワードを与えれば開いて閲覧できます。
- ▶ PDF 文書に対する印刷やテキスト抽出といった特定のアクションを権限設定が制限。
- ▶ 添付パスワードを与えて、文書自体の内容ではなくファイル添付だけを暗号化可能。

PDF 文書がこれらの保護機能のうちのいずれかを用いているとき、それは暗号化されます。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、「ファイル」→「プロパティ ...」→「セキュリティ」→それぞれ「詳細を表示 ...」「設定を変更 ...」をクリックします。

TET は PDF 権限設定に従います。dumper サンプルで演示しているように、パスワードと権限設定を、pCOS パス `encrypt/master`・`encrypt/user`・`encrypt/nocopy` 等で取得することができます。pCOS は `pcosmode` 擬似オブジェクトも提供しており、これを用いると、特定の文書についてどの操作が許されているかを知ることができます。

**内容抽出ステータス** デフォルトでは、文書をうまく開くことができたなら、TET によるテキストと画像の抽出は可能になります (`open_document()` の `requiredmode` オプションを与えた場合にはこの限りではありません)。`nocopy` 権限設定によって、制限 pCOS モードでの内容抽出が許されるか否かが決まります (フル pCOS モードでは内容抽出はつねに許されます)。以下の条件を用いれば、内容抽出が許されているかどうかを調べることができます：

```
if ((int) tet.pcos_get_number(doc, "encrypt/nocopy") == 0)
{
    /* 内容抽出は許されている */
}
```

**暗号化文書処理する必要** PDF 権限設定は、文書作成者がその内容の作成者としての権利を強制することを支援し、PDF 文書の利用者は、テキスト・画像内容を抽出する際にはその文書作成者の権利を尊重する必要があります。デフォルトでは、TET は制限モードで動作し、そのような暗号化文書からいかなる内容をも抽出することを拒否します。しかし、内容抽出はあらゆる場合においてただちに作成者の権利の侵害を構成するわけではありません。内容抽出が許可能な状況として以下のような場合が挙げられます：

- ▶ 少量の内容を引用目的で抽出する場合（「フェアユース」）。
- ▶ 組織に出入りする文書の中に特定のキーワードがないかを調べたい（文書スクリーニング）かもしれません。それを超える内容の再利用は行いません。
- ▶ 文書作成者自身がマスタパスワードを紛失したかもしれません。

- ▶ 検索エンジンが暗号化文書をインデックスする場合。その文書内容をユーザに直接利用可能にはしません（元の PDF へのリンクを示すことによって間接的にのみ利用可能とします）。

最後の例がとりわけ重要です：利用者が暗号化 PDF の内容抽出を許されていない場合であっても、企業内検索や Web ベース検索でその文書を見つけ出すことは可能であるべきです。抽出したテキストを利用者に直接利用可能とせず、文書を見つけ出せるよう検索エンジンのインデックスに供するためにのみ用いるならば、内容を抽出することは許容されるでしょう。利用者は元の暗号化 PDF をのみ利用可能となりますので（検索エンジンが内容をインデックスしてヒットリストが PDF へのリンクを含んだ後には）、利用者がその PDF を利用する際には、その文書の内部権限設定が通常通りその文書を保護することになります。

**暗号化文書に対する「シュラッグ」機能** TET は、TET ユーザが文書作成者の権利を尊重することについて責任を受け入れる前提のもとに、暗号化文書からテキストと画像を抽出するために利用できる機能を提供しています。この機能は**シュラッグ**と呼ばれ、次のように動作します：`open_document()` に `shrug` オプションを与えることによって、ユーザは、自身がいかなる文書作成者の権利をも侵害する意志を持たないことを言明します。PDFlib GmbH の取引条件は、TET のお客様が PDF 権限設定を尊重するべきであると定めています。

以下のすべての条件が真であるとき、**シュラッグ**機能は有効化されます：

- ▶ `open_document()` に `shrug` オプションが与えられている。
- ▶ 文書がマスタパスワードを必要としているが、それが `open_document()` に与えられていない。
- ▶ 文書がユーザ（文書を開く）パスワードを必要としている場合には、それが `open_document()` に与えられている必要があります。
- ▶ 文書の権限設定でテキスト抽出が許されていない、すなわち `nocopy=true`。

**シュラッグ**機能は以下の効力を持ちます：

- ▶ `nocopy=true` であっても文書からの内容抽出が許されます。ユーザは文書作成者の権利を尊重する責任を負います。
- ▶ pCOS 擬似オブジェクト `shrug` が `true/1` に設定されます。
- ▶ pCOS がフルモードで動作します（制限モードではなく）。すなわち `pcosmode` 擬似オブジェクトが 2 に設定されます。

`shrug` 擬似オブジェクトを下記の形に従って用いれば、内容をユーザに直接利用可能としてよいか、それともインデックスすることやその類の間接的用途にのみ利用するべきかを決定することができます：

```
int doc = tet.open_document(filename, "shrug");
...
if ((int) tet.pcos_get_number(doc, "shrug") == 1)
{
    /* インデックスすることのみ許される */
}
else
{
    /* 内容をユーザに渡してもよい */
}
```

## 5.2 リソース設定とファイル検索

**UPR ファイルとリソースカテゴリ** 場合によっては TET は、エンコーディング定義やグリフ名マッピングテーブルといったリソースの場所を知って利用する必要があります。リソースの取り扱いをプラットフォームに依存せずかつカスタマイズ可能なものにするために、設定ファイルを与えてそこに利用可能なリソース群とそれぞれのディスクファイル名を記述することができます。静的な設定ファイルだけでなく、`set_option()` を用いてリソースを追加することにより動的な設定も行うことが可能です。設定ファイルに関しては、*Unix PostScript Resource* (UPR) というテキスト形式が用いられます。TET で用いられる形の UPR ファイル形式を以下に説明します。TET は、表 5.1 に挙げるリソースカテゴリに対応しています。

表 5.1 リソースカテゴリ (ファイル名はすべて UTF-8 で指定する必要がある)

カテゴリ	形式 <sup>1</sup>	説明
cmap	キー = 値	CMap のリソース名とファイル名
codelist	キー = 値	コードリストのリソース名とファイル名
encoding	キー = 値	エンコーディングのリソース名とファイル名
glyphlist	キー = 値	グリフリストのリソース名とファイル名
glyphmapping	オプションリスト	175 ページの表 10.9 に従ってグリフマッピング方式を記述したオプションリスト。このリソースは <code>open_document()</code> 内で評価され、その結果は、 <code>open_document()</code> のオプション <code>glyphmapping</code> で指定されたマッピングの後に追加されます。
hostfont	キー = 値	埋め込まれていないフォントに対して用いるべきホストフォントリソースの名前 (キーは PDF フォント名。値は UTF-8 エンコーディングによるホストフォント名)
fontoutline	キー = 値	埋め込まれていないフォントに対して用いるべき TrueType フォントまたは OpenType フォントのフォントとファイル名
searchpath	値	データファイル群を含むディレクトリの相対パスまたは絶対パス

1. UPR 文法では等号「=」を名前と値の間に必要としますが、`set_option()` でリソースを指定する際はこのキャラクタは必要なく、かつ入れてはいけません。

**UPR ファイル形式** PDF ファイルは非常に簡単な構造を持ったテキストファイルであり、テキストエディタで書いたり自動的に生成させたりすることが容易にできます。まずはその文法について見てみましょう：

- ▶ 1 行に書けるのは最長 255 キャラクタまでです。
- ▶ バックスラッシュ「\」は行末キャラクタをエスケープします。これは行を延長するのに使えます。
- ▶ ピリオド「.」だけの行はセクションの終了を示します。
- ▶ 注釈行はパーセント「%」で始めることができ、行末で終了します。
- ▶ 空白は無視されます。ただしリソース名とファイル名の中の空白は無視されません。

UPR ファイルは以下の構成要素でできています：

- ▶ ファイルを識別させるための魔法行。以下の形をとります。

PS-Resources-1.0

- ▶ ファイル内に記述されるすべてのリソースカテゴリを列挙したセクション。各行にリソースカテゴリを1つずつ記述します。この列挙はピリオド1つの行によって終了します。
- ▶ ファイル冒頭で列挙された各リソースカテゴリごとにセクション1つずつ。各セクションの先頭行でリソースカテゴリを示し、その後に、利用可能なリソースを記述した行を何行でも列挙することができます。この列挙はピリオド1つの行によって終了します。各リソースデータ行にはリソースの名前を書きます（等号はクォートではさむ必要あり）。リソースがファイル名を必要とする場合は、その名前を等号の後に加える必要があります。リソース項目内に列挙されたファイルを TET が検索する際には *searchpath*（後述）が適用されます。

**UPR ファイルの記述例** 以下に UPR 設定ファイルの記述例を示します：

```
PS-Resources-1.0
searchpath
glyphlist
codelist
encoding
.
searchpath
/usr/local/lib/cmaps
/users/kurt/myfonts
.
glyphlist
myglyphlist=/usr/lib/sample.gl
.
codelist
mycodelist=/usr/lib/sample.cl
.
encoding
myencoding=sample.enc
.
```

**ファイル検索と *searchpath* リソースカテゴリ** 絶対パスや相対パスだけでなく、パス指定を一切つけずにファイル名を TET に与えることもできます。*searchpath* リソースカテゴリを用いれば、必要なデータファイル群を含むディレクトリのパス名を列挙指定することができます。ファイルを開く必要があるとき、TET はまず与えられた通りのファイル名を用いてファイルを開こうとします。それが失敗すると、TET は *searchpath* リソースカテゴリに指定されたディレクトリの中でファイルを開こうと試み、1つのディレクトリで失敗すれば次の指定ディレクトリを試し、成功するまで次々と試していきます。複数の *searchpath* 項目は蓄積させることが可能で、逆順に検索されます（後で設定したパスが、前に設定したパスよりも先に検索されます）。この検索を行わせたくないときは、TET 関数でフルパスを指定します。

Windows の場合、TET は *searchpath* リソースカテゴリを、以下のレジストリキーから読んだ値で初期化します：

```
HKLM\SOFTWARE\PDFlib\TET4\4.1\SearchPath
HKLM\SOFTWARE\PDFlib\TET4\SearchPath
HKLM\SOFTWARE\PDFlib\SearchPath
```

これらのレジストリ項目は複数のパスをセミコロン「;」で区切って持つことができます。Windows インストーラはこの *SearchPath* レジストリ項目を以下のディレクトリ名で初期化

します (TET をカスタムのディレクトリにインストールした場合はこれと同様のディレクトリ名) :

```
C:\Program Files\PDFlib\TET 4.1 32bit\resource
C:\Program Files\PDFlib\TET 4.1 32bit\resource\cmap
```

IBM iSeries の場合、*searchpath* リソースカテゴリは以下の値で初期化されます :

```
/PDFlib/TET/4.1/resource/cmap
/PDFlib/TET/4.1/resource/codelist
/PDFlib/TET/4.1/resource/glyphlst
/PDFlib/TET/4.1
/PDFlib/TET
/PDFlib
```

MVS では、この *searchpath* 機能には対応していません。

**デフォルトファイル検索パス** Unix・Linux・Mac OS X・i5/iSeries システムの場合、パス・ディレクトリ名を何も指定しなくてもいくつかのディレクトリでファイルが検索されます。UPR ファイル (これがさらに検索パスを含む場合もあります) を検索して読み取る前に、以下のディレクトリが検索されます :

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

Unix・Linux・Mac OS X の場合 *<rootpath>* は、まず */usr/local* で、ついで HOME ディレクトリで置き換えられます。i5/iSeries の場合 *<rootpath>* は空です。

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

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

この機能を利用すれば、ライセンスファイルを、環境変数や実行時オプションを何も設定せずに扱うことができます。

**UPR リソースファイルを検索** リソースファイルを用いる必要がある場合には、*set\_option()* を呼び出してそれを指定することもできますし (後述)、または UPR リソースファイルで指定することもできます。TET はこのファイルを、初めてのリソースが要求された時点で自動的に読み込みます。具体的には以下のように処理されます :

- ▶ *TETRESOURCEFILE* 環境変数が定義されている場合、TET はその値を、読み込むべき UPR ファイルの名前として採用します。そのファイルが読み込めなかったときは例外が発生します。
- ▶ *TETRESOURCEFILE* 環境変数が定義されていない場合、TET は以下の名前のファイルを開こうとします :

upr (MVSの場合。データセットが期待される)  
/tet/4.1/tet.upr (iSeriesの場合)  
tet.upr (Windows・Unix、その他すべてのシステムの場合)

このファイルが読み込めなかったときに例外は発生しません。

- ▶ Windows の場合、TET はさらに以下のレジストリ項目をも読もうとします：

```
HKLM\SOFTWARE\PDFlib\TET\4.1\resourcefile
```

そしてこのキー (TET のインストーラはこのキーを作成して値をインストールディレクトリ `>/tet.upr` を与えますが、それ以外の手段で作成することもできます) の値を、読み込むべきリソースファイルの名前として採用します。このファイルが読み込めなかったときは例外が発生します。

- ▶ クライアント側で実行時に TET に強制してリソースファイルを読み込ませるには、次のように `resourcefile` オプションを明示的に設定します：

```
set_option("resourcefile=/パス/です/tet.upr");
```

この呼び出しは何回でも繰り返すことができます。リソース項目が蓄積されます。

**リソースを実行時に設定** 設定のために UPR ファイルを用いるだけでなく、`set_option()` を用いて個々のリソースを直接設定することも可能です。この関数は、リソースカテゴリ名と、対応するリソース名・値の対 (複数可) とを、UPR リソースファイル内のそのカテゴリのセクションに書くのと同じ形で受け入れます。たとえば：

```
set_option("glyphlist={myglyphnames=/usr/local/glyphnames.g1}");
```

1つのオプションリストの中で、1つのリソースカテゴリオプションに対する複数のリソース名を設定することも可能です (ただし `set_option()` への1度の呼び出しの中で、同じリソースカテゴリオプションを複数回繰り返すことはできません)。あるいは、複数回呼び出してリソース設定を蓄積させることもできます。

**テキストファイルでのエスケープシーケンス** エスケープシーケンスは、UPR ファイルと CMap ファイル以外のすべてのテキストファイル内で使用できます。特殊なキャラクタシーケンスを用いると、テキストファイル内に印字不能キャラクタを含めることができます。すべてのシーケンスはバックスラッシュ「\」キャラクタで始まります：

- ▶ `\x` は 2 桁の 16 進数 (`0 ~ 9`、`A ~ F`、`a ~ f`) を開始します。例：`\x0D`
- ▶ `\nnn` は 3 桁の 8 進数 (`0 ~ 7`) を表します。例：`\015`。シーケンス `\000` は無視されます。
- ▶ シーケンス `\\` は 1 個のバックスラッシュを表します。
- ▶ 行末のバックスラッシュは行末キャラクタをキャンセルします。



## 5.3 代表的シナリオのための推奨方策

TETにはさまざまなオプションがあり、それらを活用することで、操作のさまざまな面を制御することが可能です。この項では、TETの典型的な応用シナリオについて、いくつかの推奨方策を示します。以下でふれる関数やオプションについて詳しくは、151ページの10章「TETライブラリAPIリファレンス」を参照して下さい。

**速度を最適化** 場合によっては、とくに検索エンジンのためにPDFをインデックスする際には、テキスト抽出の速度が最重要であり、最適出力を得ることよりも優先されます。TETのデフォルト設定は、可能なかぎり最良の出力が得られるように選択されていますが、処理を速めるように調整することも可能です。`open_page()`でオプションを選ぶことによりテキスト抽出のスループットを最大化する方法をいくつか挙げます：

▶ `docstyle=searchengine`

検索エンジンのためのインデックス処理に影響を及ぼさないやり方で出力品質を落とすことによって動作を高速化するようにいくつかの内部パラメタが設定されます。

▶ `skipengines={image}`

画像抽出が必要ない場合、動作を高速化するために内部の画像処理をスキップできます。

▶ `contentanalysis={merge=0}`

これは、時間のかかるストリップと区域の結合ステップを無効にするもので、代表的なファイルに対する処理時間をデフォルト設定の60%程度に低減します。ただし、内容がページ上に任意の順序でばらまかれているような文書では、テキストが論理順に抽出されなくなるところがあるかもしれません。

▶ `contentanalysis={dehyphenate=false}`

これは、ハイフン分割されている単語の再結合を無効にします。ハイフン除去が必要でない場合であれば、このオプションで処理時間を若干短縮できる場合があります。

▶ `contentanalysis={shadowdetect=false}`

これは、冗長な影付き・擬似太字テキストの検出を無効にします。これによっても、処理時間を短縮できる場合があります。

**単語か行レイアウトか折り返し可能テキストか** 以下のように、応用の種類によって、望ましい出力の種類は異なります(ハイフネーションされた単語はつねにこれらの設定でハイフン除去されます)：

▶ 個々の単語(レイアウト無視)：検索エンジンでは、レイアウトがらみの事柄は関心の対象とはならず、テキストの中の単語だけが関心の対象になります。このような場合には、`open_page()`で`granularity=word`を用いて、`get_text()`を1回呼び出すごとに1個の単語が抽出されるようにします。

▶ 行レイアウトを温存：`get_text()`を1回呼び出すごとに1つのページ全体のテキスト内容が抽出されるようにするには、`open_page()`で`granularity=page`を用います。テキストの行と行の間はそれぞれ、ラインフィードキャラクタで区切られるので、既存の行構造が保持されます。

▶ 折り返し可能テキスト：改行を避け、抽出されたテキストの折り返しを実現するためには、`open_page()`で`contentanalysis={lineseparator=U+0020}`と`granularity=page`を用います。`get_text()`を一回呼び出せばページ内容全体が取得できます。区域どうしはラインフィードキャラクタで区切られ、区域内の行どうしの間には空白キャラクタが挿入されます。

**検索エンジンやインデクサを書く** インデクサは通常、テキストのページ上における位置には関心を持っていません（検索された用語をハイライト表示させたい場合を除き）。多くの場合、インデクサは Unicode マッピングにおいて起こるエラーを許容し、得られるテキスト内容すべてを処理します。推奨方策：

- ▶ `open_page()` で `granularity=word` を用います。
- ▶ 句読点を処理する方法をアプリケーションが知っている場合には、ページオプション `contentanalysis={punctuationbreaks=false}` を設定すれば、句読点を隣接テキストと一緒にしておくことができます。

**位置情報** 応用の種類によっては、位置情報に関する機能が有用でしょう：

- ▶ `get_char_info()` インタフェースが必要になるのは、テキストのページ上における位置や、それぞれのフォント名や、その他詳細情報を必要とする場合だけです。テキストの座標に関心がない場合には、`get_text()` を呼び出せば充分です。
- ▶ ページのレイアウトに関して事前情報がある場合は、`open_page()` で `includebox` オプションや `excludebox` オプションを用いて、ヘッダ・フッタやその他本文テキストには含まれない部位を除外することができます。

**未知のキャラクタ** TET は、1 つないし複数のキャラクタに対する正しい Unicode マッピングを決定できないときには、それを Unicode 置き換えキャラクタ U+FFFD で表します。自分のアプリケーションがこうしたマップ不能キャラクタについて関心を持たない場合は、このキャラクタが現れても単にすべて捨ててしまえば済むことです。アプリケーションがそれより精巧な結果を求める場合は、対応するフォントを考慮に入れることができ、それを用いてマップ不能キャラクタの処理方法を決めることもできるでしょう。すべてのマップ不能キャラクタをクエスチョンマークへ置き換えるには下記の文書オプションを用います：

```
unknownchar=?
```

すべてのマップ不能キャラクタを出力から除くには下記の文書オプションを用います：

```
fold={{[:Private_Use:] remove} {[U+FFFD] remove} default}
```

**複雑なレイアウト** ある種の文書は、非常に凝ったページレイアウトを用いることがよくあります。たとえば雑誌などでは、TET はページ上の段組みどうしの関係を正しく決めることができない場合があります。このような場合には、処理時間をかけることによって、抽出されるテキストを向上させることが可能です。この目的に適したオプションは、96 ページの 6.6 「レイアウト分析」にまとめてあります。関連するオプションについて詳しくは、183 ページの表 10.12 を参照してください。

**法律文書** 法律文書を扱う際には通常、Unicode マッピングの誤りは一切許容されません。それによって文書の内容や解釈が変わってしまう危険があるためです。多くの場合テキストの位置は必要ではなく、テキストは単語ごとに抽出される必要があります。推奨方策：

- ▶ `open_page()` で `granularity=word` オプションを用います。
- ▶ 開くためにパスワードを必要とする文書进行处理する必要があるときは、`open_document()` で `password` オプションを用いて正しい文書パスワードを指定します。あるいは、内容抽出が権限設定で許可されていないときは、その文書からテキストを抽出する合法的立場に自分があるならば、`shrug` オプションを用います（68 ページの「暗号化文書に対する「シュラッグ」機能」を参照）。

- ▶ テキストの厳密さを正確に保つには：`get_char_info()` が返すキャラクタ情報構造のフィールドが1であったとき、または`get_text()` が返す文字列の中に Unicode 置き換えキャラクタが入っていたときには、ただちに処理を停止させます。

テキストモード `glyph` または `wordplus` による TETML 内では、この状況は `Glyph` 要素の下記の属性で特定できます：

`unknown="true"`

`unknownchar` オプションをよくあるキャラクタに設定しないようにします。正しくマップされたキャラクタとの区別が、`unknown` フィールドを調べないかぎりできなくなってしまうためです。

- ▶ テキストの厳密さを保つもう一つの方策としては、ページ上に表示されていないテキストについてはテキスト抽出を無効化するとよいでしょう：

`ignoreinvisibletext=true`

**PDFlib+PDI で文書を処理** PDFlib+PDI を用いて PDF 文書をページごとに処理している場合には、TET をそこに組み合わせて分割や結合の処理を制御することもできます。たとえば、PDF 文書をページ上の内容に従って分割することが可能になります。作成処理に関与している場合には、テキスト内にそのための適当な処理命令を持った区切りページを挿入することもできます。TET クックブックには、TET で文書を分析してからそれを PDFlib+PDI で処理する例が含まれています。

**Unicode 値を持たないレガシ PDF 文書** 場合によっては、レガシアプリケーションによって生成された PDF 文書を処理しなければならないことがあります。そのような PDF は、正しい Unicode マッピングに必要な十分な情報を持たないことがあります。デフォルト設定を用いた場合には、TET はそのテキスト内容の一部ないし全部抽出できないかもしれません。推奨方策：

- ▶ まずはテキストをデフォルト設定で抽出してみて、その結果を解析します。正しい Unicode マッピングに必要な十分な情報を与えないフォントを見つけます。
- ▶ このフォントの問題を解決するため、カスタムのエンコーディングテーブルとグリフ名リストを書きます。PDFlib FontReporter を利用してフォントを解析し、Unicode マッピングテーブルを作成します。
- ▶ このカスタムのマッピングテーブルを設定して、テキストをまた抽出してみます。この時は文書の量をはじめより多くしてみます。なおもマップ不能なグリフやフォントが存在する場合には、マッピングテーブルを適切に調整します。
- ▶ マップ不能なフォントを持つ文書が大量にある場合は、必要なマッピングテーブルの作成について PDFlib GmbH が支援できる場合もあります。

**PDF 文書を他の形式に変換** PDF 文書のページ内容を、できるだけ多くの情報を保ちながら自分のアプリケーションに取り込みたい場合には、正確なキャラクタメトリックが必要になります。推奨方策：

- ▶ `get_char_info()` を用いて、正確なキャラクタメトリックとフォント名を抽出します。`uv` フィールドを用いて各キャラクタの Unicode 値を抽出している場合でも、`char_info` 構造に内容を取り込むために `get_text()` を呼び出す必要があります。
- ▶ `open_page()` で `granularity=glyph` か `word` のいずれかを用います。自分のアプリケーションに合っている方を選びましょう。`granularity=glyph` で処理をすると、テキストの視覚レイアウトと、TET が処理して生成する論理的テキストとの間に齟齬が生じる場合があります（たとえば、合字グリフによって生成された 2 個のキャラクタは、その合字と同じ幅には収まらないかもしれません）。

**カスタムエンコーディングのロゴを持つ企業フォント** 多くの場合、カスタムのロゴを含んだ企業フォントは、そのロゴに対する Unicode マッピング情報を持っていないか、あるいは持っても誤っています。このようなフォントが含まれた PDF 文書が大量にある場合には、正しい Unicode 値でカスタムのマッピングテーブルを作成することを推奨します。

まず、そのフォントを含んだ PDF に対するフォントレポートを生成させ（116 ページの「PDFlib FontReporter Plugin で PDF 文書を分析」を参照）、マップが誤っているグリフをそのフォントレポート内で見つけます。そのフォントの種類で利用できる設定テーブルを選び、それを用いて、足りない Unicode マッピングを与えることができます。ロゴタイプフォントに対するコードリストの詳しい作成例は、117 ページの「コードリストリソースはあらゆる種類のフォントに利用可能」を参照。

**TeX 文書** TeX 文書によって生成された PDF 文書は、数値グリフ名や Type 3 フォントをはじめとする、他の製品がテキストをうまく抽出できない原因となる機能を含んでいることがよくあります。TET は、このような文書を扱うための多くのヒューリスティックと回避策を含んでいます。しかしある種の TeX 文書は、処理時間をより多く要する、デフォルトでは無効になっている回避策でのみ処理することが可能です。下記の文書オプションを用いれば、こうした文書のための、CPU 消費のより多いフォント処理を有効にすることができます：

```
checkglyphlists=true
```

# 6 テキスト抽出

## 6.1 PDF のさまざまな文書領域

PDF 文書は、ページ内容だけではなく、それ以外の多くの場所にテキストを含んでいる可能性があります。たいていのアプリケーションはページ内容のみを扱いますが、他の文書領域が必要な状況も多くあります。

ページ内容は、最も活躍する関数 `get_text()` と `get_image()` で取得することができ、一方、他の文書領域からテキストを取得するには内蔵の pCOS インタフェースが主要な役割を務めます。

この節では以下、TET ライブラリと TETML での領域検索について情報を提供します。あわせて、これらの文書領域を Acrobat 8/9/X で検索する方法もまとめます。これは Acrobat で検索ヒットを見つけるために重要です。

**ページ上のテキスト** ページ内容は PDF 内の主要なテキストです。ページ上のテキストはフォントで視覚表現され、PDF 内で利用可能な多様なエンコーディング技法の一つを用いて符号化されています。

- ▶ Acrobat 8/9/X での表示方法：ページ内容はつねに表示されています。
- ▶ Acrobat 8/9/X で一つの PDF を検索する方法：「編集」→「検索」または「編集」→「高度な検索」。Acrobat がグリフを Unicode 値へ正しくマップできない文書内のテキストを、TET は処理できる可能性があります。このような状況では、TET をベースにした TET Plugin を利用することができます (51 ページの 4.1 「Adobe Acrobat 用無償 TET Plugin」を参照)。TET Plugin は、その自前の検索ダイアログを「Plug-Ins」→「PDFlib TET Plugin...」→「TET Find」で提供します。ただしこれは、完全な検索機能を提供するようには意図されていません。
- ▶ Acrobat 8/9/X で複数の PDF を検索する方法：「編集」→「検索」または「編集」→「高度な検索」の後、「検索する場所を指定してください。」の中で「以下の場所にあるすべての PDF 文書」を選択し、PDF 文書群が入っているフォルダへブラウザ。
- ▶ TET ライブラリ用サンプルコード：`extractor` ミニサンプル
- ▶ TETML 要素：`/TET/Document/Pages/Page`

**定義済み文書情報項目** 固有の文書情報項目はキー / 値対です。

- ▶ Acrobat 8/9/X での表示方法：「ファイル」→「プロパティ...」
- ▶ Acrobat 8/9/X で一つの PDF を検索する方法：なし
- ▶ Acrobat 8/9/X で複数の PDF を検索する方法：「編集」→「検索」または「編集」→「高度な検索」の後、ダイアログの下端近くの「高度な検索オプションを使用」をクリック。「検索する場所：」プルダウンで PDF 文書群のフォルダを選択し、プルダウンメニュー「その他の条件」で「作成日」「更新日」「作成者」「タイトル」「サブタイトル」「キーワード」のいずれかを選択。
- ▶ TET ライブラリ用サンプルコード：`dumper` ミニサンプル
- ▶ TETML 要素：`/TET/Document/DocInfo`

**カスタム文書情報項目** 標準項目に加えて、カスタム文書情報項目を定義することもできます。

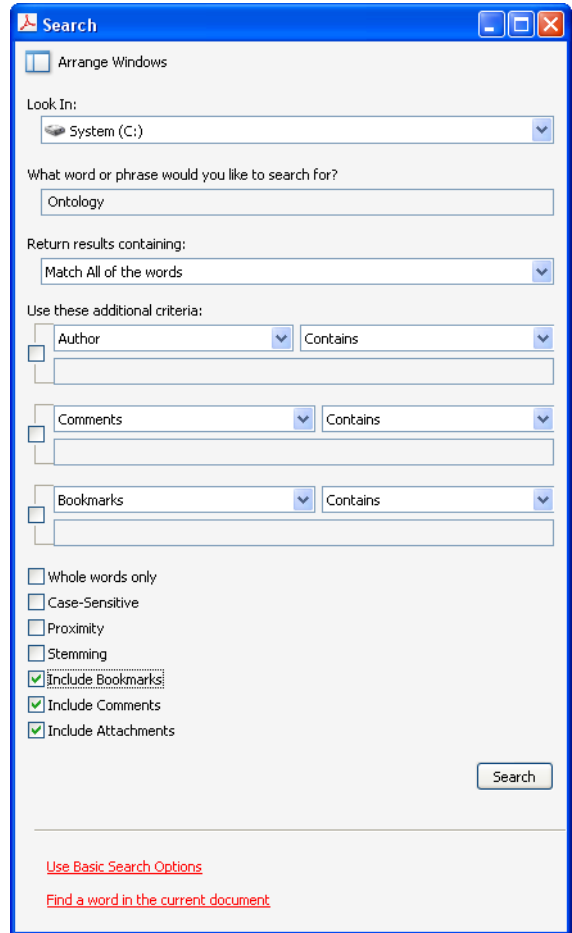


図 6.1  
Acrobat の高度な  
検索ダイアログ

- ▶ Acrobat 8/9/X での表示方法：「ファイル」→「プロパティ...」→「カスタム」（無償の Acrobat Reader では利用できません）
- ▶ Acrobat 8/9/X での検索方法：なし
- ▶ TET ライブラリ用サンプルコード：*dumper* ミニサンプル
- ▶ TETML 要素：*/TET/Document/DocInfo/Custom*

**文書レベルの XMP メタデータ** XMP メタデータは、拡張されたメタデータを内容として持つ XML ストリームから成ります。

- ▶ Acrobat 8/9/X での表示方法：「ファイル」→「プロパティ...」→「その他のメタデータ...」（無償の Acrobat Reader では利用できません）
- ▶ Acrobat 8/9/X で一つの PDF を検索する方法：なし
- ▶ Acrobat 8/9/X で複数の PDF を検索する方法：「編集」→「検索」または「編集」→「高度な検索」の後、「高度な検索オプションを使用」をクリック。「検索する場所：」プルダウンで PDF 文書群のフォルダを選択し、プルダウンメニュー「その他の条件」で「XMP メタデータ」を選択（無償の Acrobat Reader では利用できません）。
- ▶ TET ライブラリ用サンプルコード：*dumper* ミニサンプル
- ▶ TETML 要素：*/TET/Document/Metadata*

**画像レベルの XMP メタデータ** XMP メタデータは、画像・ページ・フォントといった文書構成要素にも付けることが可能です。しかし、XMP が通常見つかるのは画像レベルにおいてのみです（文書レベルに加えて）。

- ▶ Acrobat 8/9 での表示方法：「ツール」→「高度な編集」→「TouchUp オブジェクトツール」→画像を選択→右クリック→「メタデータを表示 ...」（無償の Acrobat Reader では利用できません）
- ▶ Acrobat X での表示方法：「ツール」→「コンテンツ」→「オブジェクトを編集」→画像を選択→右クリック→「メタデータを表示 ...」（無償の Acrobat Reader では利用できません）
- ▶ Acrobat 8/9/X での検索方法：なし
- ▶ TET ライブラリ用サンプルコード：pCOS クックブックのトピック *image\_metadata*
- ▶ TETML 要素：/TET/Document/Pages/Resources/Images/Image/Metadata

**フォームフィールド内のテキスト** フォームフィールドはページにかぶさって表示されます。しかし、技術的にはこれはページ内容の一部ではなく、別途のデータ構造によって表現されます。

- ▶ Acrobat 8 での表示方法：「表示」→「ナビゲーションパネル」→「フィールド」
- ▶ Acrobat 9 での表示方法：「フォーム」→「フィールドを追加または編集 ...」
- ▶ Acrobat X での表示方法：「ツール」→「フォーム」→「編集」（無償の Acrobat Reader では利用できません）
- ▶ Acrobat 8/9/X での検索方法：なし
- ▶ TET ライブラリ用サンプルコード：pCOS クックブックのトピック *fields*
- ▶ TETML 要素：なし

**注釈内のテキスト** フォームフィールドと同様、注釈（ノート・テキスト注釈等）は別のデータ構造によって表現されます。注釈の興味対象となるテキスト内容はその種類によって異なります。たとえば、Web リンクの場合はその興味対象となる部分は URL でしょうし、それ以外の種類の注釈では印字されるテキスト内容が意味を持つでしょう。

- ▶ Acrobat 8/9 での表示方法：「表示」→「ナビゲーションパネル」→「注釈」
- ▶ Acrobat X での表示方法：「注釈」→「注釈のリスト」
- ▶ Acrobat 8/9/X で一つの PDF を検索する方法：「編集」→「検索」→ボックス「注釈を含める」をチェック、または注釈のリストツールバーで「検索」ボタンをクリック
- ▶ Acrobat 8/9/X で複数の PDF を検索する方法：「編集」→「検索」または「編集」→「高度な検索」の後、「高度な検索オプションを使用」をクリック。「検索する場所：」プルダウンで PDF 文書群のフォルダを選択し、プルダウンメニュー「その他の条件」で「注釈」を選択。
- ▶ TET ライブラリ用サンプルコード：pCOS クックブックのトピック *annotations*
- ▶ TETML 要素：なし

**しおり内のテキスト** しおりは直接にはページと結びついていませんが、特定のページへ飛ぶアクションを内容として持つ場合があります。しおりは階層構造を形成するように入れ子にすることも可能です。

- ▶ Acrobat 8/9 での表示方法：「表示」→「ナビゲーションパネル」→「しおり」
- ▶ Acrobat X での表示方法：「表示」→「表示切り替え」→「ナビゲーションパネル」→「しおり」
- ▶ Acrobat 8/9/X で一つの PDF を検索する方法：「編集」→「検索」または「編集」→「高度な検索」の後、ボックス「しおりを含める」をチェック

- ▶ Acrobat 8/9/X で複数の PDF を検索する方法：「編集」→「検索」または「編集」→「高度な検索」の後、「高度な検索オプションを使用」をクリック。「検索する場所：」プルダウンで PDF 文書群のフォルダを選択し、プルダウンメニュー「その他の条件」で「しおり」を選択（無償の Acrobat Reader では利用できません）
- ▶ TET ライブラリ用サンプルコード：pCOS クックブックのトピック *bookmarks*
- ▶ TETML 要素：なし

**ファイル添付** PDF 文書はファイル添付を含むこともでき（文書レベルかページレベルで）、そのファイル添付自体が PDF 文書であることも可能です。

- ▶ Acrobat 8/9 での表示方法：「表示」→「ナビゲーションパネル」→「添付ファイル」
- ▶ Acrobat X での表示方法：「表示」→「表示切り替え」→「ナビゲーションパネル」→「添付ファイル」
- ▶ Acrobat 8/9/X での検索方法：「編集」→「検索」または「編集」→「高度な検索」の後、ボックス「添付ファイルを含める」をチェック（無償の Acrobat Reader では利用できません）。入れ子になった添付は再帰的に検索されません。
- ▶ TET ライブラリ用サンプルコード：*get\_attachments* ミニサンプル
- ▶ TETML 要素：*/TET/Document/Attachments/Attachment/Document*

**PDF パッケージ・ポートフォリオ** Acrobat 8 では、PDF パッケージという概念が導入されました。これはファイル添付に追加のプロパティを与えたものです。Acrobat 9 ではこの概念を拡張して、PDF ポートフォリオが導入されました。

- ▶ Acrobat 8/9/X での表示方法：Acrobat は、PDF パッケージ専用のユーザインタフェースで、パッケージ / ポートフォリオの表紙と中身の PDF 文書群を表現します。
- ▶ Acrobat 8/9 で一つの PDF パッケージを検索する方法：「編集」→「検索」→「検索する場所：」プルダウンで「PDF パッケージ全体」を選択
- ▶ Acrobat X で一つの PDF パッケージを検索する方法：「編集」→「ポートフォリオ全体を検索」
- ▶ Acrobat 8/9/X で複数の PDF パッケージを検索する方法：なし
- ▶ TET ライブラリ用サンプルコード：*get\_attachments* ミニサンプル
- ▶ TETML 要素：*/TET/Document/Attachments/Attachment/Document*

**PDF の各種規格とその他各種 PDF プロパティ** この領域は明示的にテキストを含んではおらず、PDF/X・PDF/A ステータス、タグ付き PDF ステータスといった、PDF 文書のさまざまな固有プロパティを集めたコンテナとして用いられます。

- ▶ Acrobat 8 での表示方法：なし
- ▶ Acrobat 9：「表示」→「ナビゲーションパネル」→「規格」（規格準拠 PDF でのみ現れます）
- ▶ Acrobat X：「表示」→「表示切り替え」→「ナビゲーションパネル」→「規格」（規格準拠 PDF でのみ現れます）
- ▶ Acrobat 8/9/X での検索方法：なし
- ▶ TET ライブラリ用サンプルコード：*dumper* ミニサンプル
- ▶ TETML 要素・属性：*/TET/Document/@pdfa*・*/TET/Document/@pdfe*・*/TET/Document/@pdfua*・*/TET/Document/@pdfvt*・*/TET/Document/@pdfx*



## 6.2 ページとテキストの視覚情報

**デフォルト座標系** デフォルトではTETは、ページとテキストのすべての視覚情報をPDFの標準座標系で表します。ただし、座標系の原点（ページの外にある場合もある）は、ページの表示可能領域の左下隅に一致させます。正確には、**CropBox** が存在するときはその左下隅、そうでないときは **MediaBox** の左下隅が原点になります。ページが **Rotate** キーを持つ場合はページ回転が行われます。この座標系は DTP ポイントを単位として用います。

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

第1座標は右へ増加し、第2座標は上へ増加します。TET に与える座標はすべてこの座標系で表したものでなければなりませんし、TET が返す座標もすべてこの座標系で表されたものになります。PDF 文書内でその座標が実際どのように表されているかは関係ありません。PDF のページサイズを得る方法については pCOS パスリファレンスを参照してください。

**下向き座標系** PDF の上向き座標系とは異なり、グラフィック環境によっては上向き座標系を用いているものがあります。開発者によってはこの方を好む場合があります。下向き座標系の利用を実現するため、TET では代替座標系を使うことができます。この代替座標系では、すべての関連する座標はページの左下隅ではなく左上隅に対して解釈され、**y** 座標は下向きに増加します。この**下向き機能**は、TET のユーザが下向き座標系をごく自然に扱えるようにするために設計されています。追加の利点として、下向き座標は Acrobat で表示される座標値と等しくなります（後述）。ページで下向き座標を有効にするには、**topdown** ページオプションを用います。

**Acrobat で座標を表示** Acrobatでページ座標を表示するには以下のように操作します(図 6.2 参照) :

- ▶ カーソル座標を表示するには、Acrobat X では「表示」→「表示切り替え」→「カーソル座標」(Acrobat 9 : 「表示」→「カーソル座標」。Acrobat 8 : 「表示」→「ナビゲーションタブ」→「情報」) を用います。
- ▶ 座標は、Acrobat で現在選択されている単位で表示されます。Acrobat 8/9/X で表示単位をポイントに変更する (TET で用いられているのと同様に) には、次のように操作します : 「編集」→「環境設定」(→「一般 ...」) →「単位とガイド」→「単位」へ行き、「ポイント」を選択。

この表示されている座標はページの左上隅を原点としており、左下隅を原点に持つ PDF・TET のデフォルト座標系とは異なることに留意してください。Acrobat の座標系と揃う下向き座標系を選ぶための方法については前項を参照してください。

**テキスト抽出の領域** デフォルトでは TET は、目に見えるページ領域のテキストをすべて抽出します。**open\_page()** の **clippingarea** オプション (179 ページの表 10.10 を参照) を用いると、これを任意の PDF ページ枠項目 (TrimBox 等) へ変更することができます。キーワード **unlimited** を用いると、いかなるページ枠にもかかわらずすべてのテキストを抽出することができます。デフォルト値 **cropbox** は、Acrobat で目に見える領域内のテキストを抽出するよう TET に指示します。

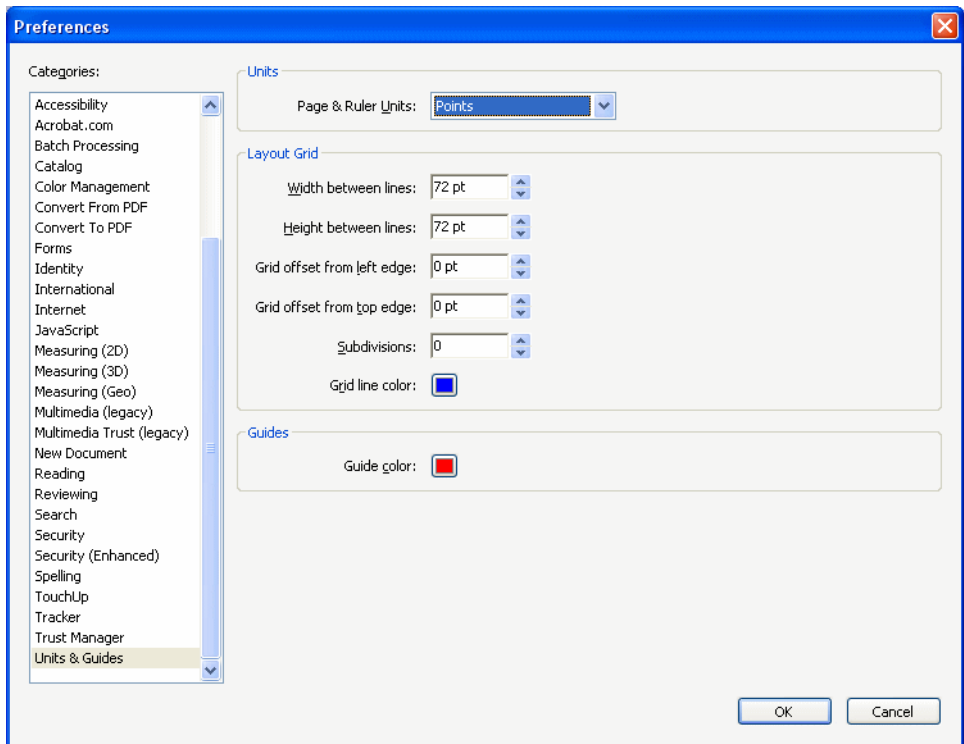


図 6.2 Acrobat の座標表示を設定。カーソル座標を表示するには「表示」→「カーソル座標」を用います。

テキスト抽出の領域はもっと細かく、任意の数の矩形領域を *open\_page()* の *includebox-excludebox* オプションで与えることによって指定することもできます。これは部分的なページ内容（選んだ段組等）を抽出したり、あるいは必要ない部分（余白・ヘッダ・フッタ等）を除いたりするために有用です。最終的な切り抜き領域は、*includebox* オプションで指定されたすべての矩形の和を算出し、そこから *excludebox* オプションで指定されたすべての矩形の和を除外することによって構築されます。キャラクタは、その参照点が切り抜き領域の中にあるとき、その切り抜き領域内にあると判定されます。これはすなわち、キャラクタの一部が切り抜き領域の外に出ていてもそれがその切り抜き領域内にあると判定される場合があります、またその逆の場合もあることを意味します。

**グリフメトリック** *get\_char\_info()* を用いると、任意のグリフに対して返されたキャラクタ群のフォント情報やメトリック情報を取得することができます。出力内の各キャラクタに対して、それぞれ以下の値が得られます（図 6.3 と 189 ページの表 10.15 を参照）：

- ▶ **uv** フィールドは、情報取得対象のキャラクタ（カレントキャラクタ）の UTF-32 Unicode 値を持ちます。このフィールドはつねに UTF-32 を持ちます。ネイティブな Unicode 文字列内で UTF-16 文字列しか扱えない言語バインディングにおいても同様です。この **uv** フィールドを利用すると、BMP 領域外のキャラクタをアプリケーションで取り扱いたいときに、サロゲートペアを解釈する必要なしに取り扱えるようになります。サロゲートペアは 2 個の別々のキャラクタとして報告されるので、1 つ目の値の **uv** フィー

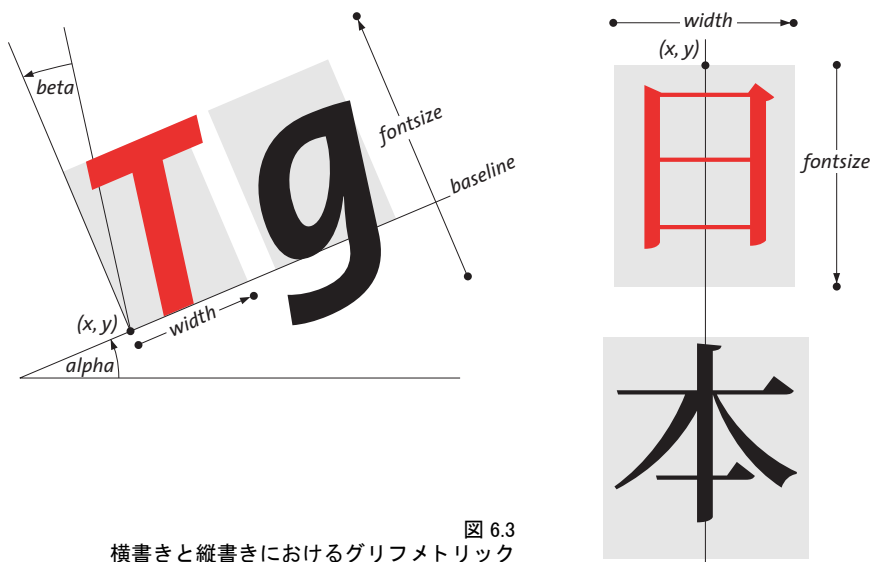


図 6.3  
横書きと縦書きにおけるグリフメトリック

ルドが実際の Unicode 値 (U+FFFF を超える) を持ち、2 つ目の値の *uv* フィールドは無形キャラクターとして扱われて *uv* 値 0 を持ちます。

- ▶ **type** フィールドは、そのキャラクターの種別を表します。有形キャラクターと無形キャラクターの 2 種類があります。有形キャラクターに分類されるものとしては、通常のキャラクター (1 つのグリフまるごとの結果等) と、1 つのグリフに対応する複数のキャラクターの列の先頭キャラクター (合字の 1 文字目等) が挙げられます。無形キャラクターに分類されるものとしては、複数のキャラクターの列の中の後続キャラクター (合字の 2 文字目等) と、挿入された区切りキャラクターが挙げられます。無形キャラクターの場合、位置  $(x, y)$  はもっとも最近の有形キャラクターの終了点を指し、幅 *width* は 0 になり、その他のフィールドは *uv* 以外すべてもっとも最近の有形キャラクターと同じになります。この終了点は、方向が *alpha* (横書き) の場合は  $(x, y)$  に *width* を加えた点になり、方向が  $-90^\circ$  (縦書き) の場合は *fontsize* を加えた点になります。
- ▶ **unknown** フィールドは通常は false (C・C++ では 0) ですが、ただし元のグリフが Unicode にマップすることができないために *unknownchar* オプションによる指定キャラクターに置き換えられたときは値 true (C・C++ では 1) を持ちます。このフィールドを用いると、クエスチョンマークやスペースなどのありがちなキャラクターを *unknownchar* として指定したような場合に、実際の文書の内容と置き換えキャラクターとを区別することが可能になります。
- ▶ **attributes** フィールドは、TET の内容分析アルゴリズムで決定された下付き・上付き・ドロップキャップ・影付きステータスに関する情報を内容として持ちます。
- ▶  **$(x, y)$**  両フィールドは、そのグリフの参照点の位置を指します。この参照点は横書きではグリフ矩形の左下隅になり、縦書きでは上端中央になります (87 ページの 6.3.1 「日中韓エンコーディング・CMap」参照)。無形キャラクターについては、ページ上に対応するグリフがないので、点  $(x, y)$  はもっとも最近の有形キャラクターの終了点を指します。*y* の値は *topdown* ページオプションに依存します。
- ▶ **width** フィールドは、そのグリフに対応するフォントメトリックと文字間隔・水平倍率等のテキスト出力パラメタ群とに従ったグリフの幅を表します。こうしたパラメタが次のグリフの位置を制御するので、隣り合う 2 つのグリフの参照点間の間隔は *width*

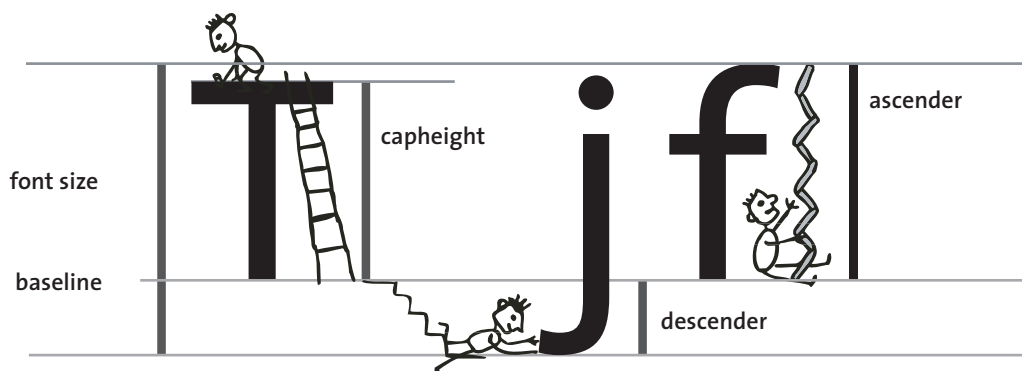


図 6.4 フォント固有メトリック

とは異なる場合があります。ゼロ幅キャラクタの場合 *width* はゼロになることがあります。逆に斜体テキスト等、実際のアウトラインの幅がそのグリフの *width* 値より広がる場合もあります。

無形キャラクタの場合 *width* は 0 になります。

- ▶ 角度 *alpha* は、行内のテキストの進行方向を与えます。これは、標準方向からの偏移で表されます。この標準方向は横書きでは  $0^\circ$  になり、縦書きでは  $-90^\circ$  になります（縦書きについて詳しくは後述）。よって角度 *alpha* は、通常の横書きテキストと通常の縦書きテキストに対してはともに  $0^\circ$  になります。 *alpha* ・ *beta* の値は *topdown* ページオプションに依存します。
- ▶ 角度 *beta* は、斜体（擬似イタリック）テキスト等の、テキストにかかった傾斜を表します。この角度は *alpha* に対する垂線から測られます。通常の正立したテキストに対しては  $0^\circ$  になります（横書きでも縦書きでも）。 *beta* の絶対値が  $90^\circ$  を超える場合、そのテキストはベースラインを軸に反転して裏返っていることになります。
- ▶ *fontid* フィールドは、そのグリフに対して用いられているフォントの pCOS ID を持ちます。この ID を用いると、フォント名・埋め込み状況・記述方向（横書きか縦書きか）といった詳しいフォント情報を取得することができます。pCOS パスリファレンスに、こうしたフォント情報取得のためのサンプルコードがあります。
- ▶ *fontsize* フィールドは、テキストのサイズをポイント単位で表します。これは正規化されるので、つねに正の値になります。
- ▶ *textrendering* フィールドは、描線・塗り・不可視等、グリフのレンダリングの種類を表します。これは PDF ページ記述について定義されているテキストレンダリングモードの数値を反映します（189 ページの表 10.15 を参照）。不可視のテキストはデフォルトでは抽出されますが、 *open\_page()* で *ignoreinvisibletext* オプションを用いればそうでなくすることもできます。

**フォント固有メトリック** TET は、PostScript と PDF が用いているグリフ・フォントメトリックシステムを用いています。ここで簡単に説明しましょう。

文字サイズには通常、キャラクタの各部が重なり合わないために必要な、隣り合う行どうしの最小限の間隔が選ばれます。文字サイズは一般に、フォント内の個々のキャラクタよりも大きくなります。なぜなら、文字サイズはアセンダとディセンダにわたるうえ、さらに行間の追加のアキが加わる場合もあるからです。

キャップハイトは、多くの欧文フォントの *T* や *H* のような大文字の高さです。x ハイトは、多くの欧文フォントの *x* のような小文字の高さです。アセンダは、多くの欧文フォントの *f* や *d* のような小文字の高さです。ディセンダは、多くの欧文フォントの *j* や *p* のような小文字のベースラインから下端までの間隔です。ディセンダは通常、負の値です。x ハイト・キャップハイト・アセンダ・ディセンダの値は文字サイズの 1000 分の 1 を単位に測られます。

これらの値はフォントによって異なっており、pCOS インタフェースで取得できます。たとえば、以下のコードはアセンダとディセンダの値を取得します：

```
/* アセンダとディセンダの値を取得 */
path = "fonts[" + i + "]/ascender";
System.out.println("アセンダ=" + p.pcos_get_number(doc, path));

path = "fonts[" + i + "]/descender";
System.out.println("ディセンダ=" + p.pcos_get_number(doc, path));
```

なお、アセンダ等のフォントメトリック値は、このフォントによるグリフに対して `get_char_info()` を呼び出した後のみ取得するべきです。言い換えれば、`get_char_info()` から返されたフォント ID を用いることは安全ですが、`fonts[]` 配列内のすべてのフォントを評価することは必ずしも埋め込まれたフォントデータからのメトリック値を与えず、PDF *FontDescriptor* 辞書からの不正確な値を与える可能性があります。詳しくは pCOS パスリファレンスを参照してください。

**グリフと単語の終了点** ハイライトを正しく行うためには、単語内の末尾キャラクタの末尾位置が必要になります。`get_char_info()` が返す開始点座標 *x, y* と `width · alpha` 値を用いれば、横書きでのグリフの終了点を、すなわちそのグリフのアドバンスベクトルの終点（グリフ枠の右下隅）を決定することができます：

```
x_end = lrx = x + 幅 * cos(alpha)
y_end = lry = y + 幅 * sin(alpha)
```

通常の横書きテキスト（すなわち `alpha=0`）では、これは次の形に省略できます：

```
x_end = lrx = x + 幅
y_end = lry = y
```

より一般的には、グリフ枠のサイズを、その右上隅の座標を決定することによって算出できます（この算式は、グリフが角度 *beta* によって斜形化されている可能性を考慮に入れていません）：

```
urx = x + 幅 * cos(alpha) - 向き * 高さ * sin(alpha)
ury = y + 幅 * sin(alpha) + 向き * 高さ * cos(alpha)
```

ここで、`topdown=true` ならば向き = -1、`topdown=false` ならば向き = 1 です（81 ページの「下向き座標系」参照）。高さの値は文字サイズとフォントの幾何情報に依存します。下記は、広く用いられている多くのフォントについて、使える値を与えます（アセンダ値の取得については 84 ページの「フォント固有メトリック」を参照）：

```
高さ = 文字サイズ * アセンダ / 1000
```

多くのグラフィック開発環境において、グリフ変換は以下のように表すことができます：

```
translate(x,y);
rotate(alpha);
skew(0, -beta);
if (abs(beta) > 90)
    scale(1 -1);
```

これらの変換を行なった後は、グリフ枠の右上隅は以下のように表すことができます：

```
urx = x + 幅
ury = y + 向き * 高さ
```

**縦書きでのグリフ計算** 縦書きの日中韓テキストについては、終了点の計算は以下のように行われます：

```
Xend = x
Yend = y - 文字サイズ
```

グリフ枠の左上隅と右下隅は以下のように算出できます：

```
ulx = x - 幅/2 * cos(alpha)
uly = y - 幅/2 * sin(alpha)

lrx = ulx + 幅 * cos(alpha) + 向き * 文字サイズ * sin(alpha)
lry = uly + 幅 * sin(alpha) - 向き * 文字サイズ * cos(alpha)
```

ここで、*topdown=true* ならば向き = -1、*topdown=false* ならば向き = 1 です（81 ページの「下向き座標系」参照）。

## 6.3 日本語・中国語・韓国語テキスト

### 6.3.1 日中韓エンコーディング・CMap

TET は日本語・中国語・韓国語（日中韓）のテキストに対応しており、横書き・縦書きの、任意のレガシエンコーディング（CMap）の日中韓テキストを Unicode に変換します。TET は Adobe のすべての日中韓キャラクタ集合に対応しています：

- ▶ 日本語：Adobe-Japan1 ～ 6
- ▶ 中国語繁体字：Adobe-CNS1 ～ 5
- ▶ 中国語簡体字：Adobe-GB1 ～ 5
- ▶ 韓国語：Adobe-Korea1 ～ 2

これらの PDF CMap は、今日用いられているすべての日中韓キャラクタエンコーディングをカバーしています。たとえば Shift-JIS・EUC・Big-5・KSC 等、多数のエンコーディングです。ロケール独自エンコーディングで符号化された日中韓フォント名（Shift-JIS で符号化された和文フォント名など）は、Unicode へ正規化されます。

**注記** レガシエンコーディングで符号化された日中韓テキストを抽出するには、TET に同梱の CMap ファイル群の場所を、7 ページの 0.1 「ソフトウェアをインストール」に従って設定する必要があります。

### 6.3.2 日中韓テキストの単語境界

日中韓テキストの単語境界検出は *ideographic* ページオプションで制御できます：

- ▶ *ideographic=split* にすると、表意文字キャラクタはつねに単語境界を構成します。すなわち、*granularity=word* の場合に表意文字が 1 個ずつ返されます。表意文字日中韓キャラクタは単語境界と見なされますが、カタカナキャラクタは単語境界として扱われません。
- ▶ *ideographic=keep* にすると、表意文字キャラクタは一般に単語境界を構成しません。句読点と、表意文字キャラクタと非表意文字キャラクタとの間の切り替わりは、なお単語境界を構成します。*granularity=word* の場合、表意文字読点 *U+3001* と表意文字句点 *U+3002* も単語境界を構成します。*granularity=page* の場合、行末に行区切りが挿入されません。

互換上の理由からデフォルト値は *ideographic=split* ですが、日中韓テキストに対してはテキスト抽出向上のため *ideographic=keep* を用いることを強く推奨します。

### 6.3.3 縦書き

TET は横書きにも縦書きにも対応しており、それぞれについて適切なメトリック計算をすべて行います。縦書きのテキストを扱う際には以下のことに留意して下さい。

- ▶ グリフの参照点は縦書きの場合にはグリフ矩形の上端中央にあります。テキストの位置は下へ向かって進みます。その進行幅はフォントサイズと文字間隔によって決定され、グリフ幅には依存しません（図 6.3 参照）。
- ▶ 角度 *alpha* は通常の縦書きテキストについては  $0^\circ$  です。いいかえれば、縦書きのフォントで *alpha=0^\circ* ならば下へ、すなわち  $-90^\circ$  の方向に向かって進むということです。
- ▶ 上述の違いのため、クライアントコード側では下記の pCOS コードを用いて記述方向を考慮に入れる必要があります（テキストが縦に並んでいるからといって、それが実際には縦書きのフォントを用いているとは限らないことに留意して下さい）：

```

count = p.pcos_get_number(doc, "length:fonts");
for (i=0; i < count; i++)
{
    if (p.pcos_get_number(doc, "fonts[" + id + "]/vertical"))
    {
        /* フォントは縦書きを用いている */
        vertical = true;
    }
}

```

- ▶ 縦書きのテキストと句読点の回転済みグリフは、それぞれ対応する未回転 Unicode キャラクタへマップされます。回転済みキャラクタを温存するには下記の文書オプションを uses :

```
decompose={vertical=_none}
```

- ▶ 縦書きテキストのテキスト行検出は下記のページオプションで改善できることが多いです :

```
layoutanalysis={forcelayoutanalysis=0}
```

### 6.3.4 日中韓分解 : narrow · wide · vertical 等

Unicode と多くのレガシエンコーディングでは、全角と半角のキャラクタという概念が使えます (ダブルバイト・シングルバイトキャラクタと呼ばれる時もあります)。デフォルトでは TET は、全角・半角キャラクタをそれぞれ対応する標準幅のキャラクタへ置き換える Unicode 分解 *wide* · *narrow* を行います。

元の全角・半角キャラクタを温存するには、*decompose* 文書オプションを用いてそれぞれの分解を無効化します :

```
decompose={wide=_none narrow=_none}
```

同様に、*small* · *square* · *vertical* 分解も日中韓キャラクタに対して効力を持ちます。これらの分解は (*wide* と *narrow* も含めて) デフォルトではすべて有効になっていますので、キャラクタはそれぞれ対応する標準的キャラクタへ変換されます。元のキャラクタを温存するには、それぞれの分解を無効化します。下記の文書オプションはすべての分解を無効化します :

```
decompose={none}
```

表 6.1 に、日中韓分解を例とともに示します。分解について詳しくは、108 ページの 7.3.2 「Unicode 分解」を参照してください。



表 6.1 日中韓互換分解の例 (decompose オプションのサブオプション)

分解名	説明	対象 Unicode キャラクタ	分解が有効のとき (デフォルト)	分解が無効のとき
<i>narrow</i>	半角キャラクタ	U+FF61 ~ U+FFDC、 U+FFE8 ~ U+FFEE	ㄱ U+30F2	ㄱ U+FF66
<i>small</i>	CNS 11643 互換の ための小型字体	U+FE50 ~ U+FE6B	꺾 U+002C	꺾 U+FE50
<i>square</i>	日中韓の組文字	U+3250、 U+32CC ~ U+32CF、 U+3300 ~ U+3357、 U+3371 ~ U+33DF、 U+337B ~ U+337F、 U+33FF、 U+1F131 ~ U+1F14E、 U+1F190、 U+1F200、 U+1F210 ~ U+1F231	ㄱ ㅁ U+30AD U+30ED	ㄱ ㅁ U+3314
<i>vertical</i>	縦書き字体	U+309F、 U+30FF、 U+FE10 ~ U+FE19、 U+FE30 ~ U+FE48	ㄱ U+FE37	{ U+007B
<i>wide</i>	全角字体	U+3000、 U+FF01 ~ U+FF60、 U+FFE0 ~ U+FFE6	ㄱ U+00A3	ㄱ U+FFE1

## 6.4 双方向アラビア文字・ヘブライ文字テキスト

TET は、アラビア文字やヘブライ文字といった右書き用字系による文書からテキストを正しく抽出するために、追加の処理を行います。こうした用字系ではしばしば左書きテキストが挿入されますので（数等）、そのような文書は、双方向であるといえます。双方向テキストの抽出には、以下に説明する処理ステップの一つないし複数が関与します。

### 6.4.1 双方向の一般的性質

**右書き・双方向テキストを並べ替え** 右書きの並びと左書きの並びは、論理的なテキストの正しい並びを形成するよう並べ替える必要があります。粒度が **word** 以上の場合、下記のページオプションを用いると、TET はテキストを論理順に発出します（これがデフォルト設定です）：

```
contentanalysis={bidi=logical}
```

双方向処理は下記ページオプションで明示的に無効化することもできます：

```
contentanalysis={bidi=visual}
```

**ページの優勢テキスト向きを決定** 双方向並べ替えは、単語内のキャラクタ群と行内の単語群に対して効力を持つばかりではなく、それ以外のページレイアウト認識の諸側面に対しても効力を持ちます。混在双方向行は場合によっては、ページ全体が右書きなのか左書きなのかを考慮に入れることなしには安心して並べ替えできないこともあります。この決定を自動的に行うために、TET はページの優勢テキスト向きを調べ、そのページが主に左書きと見なすべきかそれとも主に右書きと見なすべきかにそのアルゴリズムを合わせます。

この決定は **bidilevel** オプションで上書きすることもできます。たとえば下記のオプションリストは、テキストの大多数が左書きのページ上であっても右書き処理を強制します：

```
contentanalysis={bidilevel=rtl}
```

**グリフ順序** `get_char_info()` と TETML 内の **Glyph** 要素によって返されるグリフ情報は、つねに視覚的順序に従って、すなわち普通の水平ベースラインについては左書きとして並べられます。この左書きグリフ順序によって、クライアントアプリケーションは、テキストの双方向ステータスを調べる必要なしに決め打ちの順序でグリフ座標を受け取ることができます。この動作は、実際のテキスト向きは右書きであるという事実にもかかわらず、アラビア文字やヘブライ文字のフォント内のグリフは概して、その左辺に参照点があり右へ進行しているという現実を反映しています。

### 6.4.2 アラビア文字テキストを後処理

**アラビア文字の表示形を正規化し合字を分解** アラビア文字のキャラクタは、最高 4 種類の形で存在しています。単独形・語頭形・語中形・語尾形です。これらの形は、意味的には同一のキャラクタを表していますが、別々の Unicode 値を持っている場合があります。デフォルトでは TET は、すべての表示形を、それぞれ対応する正準形へ変換します。表 6.2 に示すように、**decompose** オプションを用いて表示形を温存することもできます（108 ページの 7.3.2 「Unicode 分解」を参照）。

PDF 文書では各表示形は、単独形の Unicode キャラクタへマップされていることもあれば、表示形のうちのひとつ（たとえばその文書の ToUnicode CMap 内の）へマップされて

いることもあるため、たとえ分解が無効化されていても、出力が表示形を含むことを TET は保証はできません。

表 6.2 アラビア文字の表示形を decompose オプションで処理

説明とオプションリスト	分解前	分解後 (論理順で)
語尾形・語頭形・単独形・語中形を分解 : decompose オプションなし (デフォルト) または decompose=none	س U+FEB2	س U+0633
または decompose={final=_all medial=_all initial=_all isolated=_all}	س U+FEB3	س U+0633
	س U+FD0E	س ر U+0633 U+0631
	س U+FEB4	س U+0633
語尾形・語頭形・単独形・語中形を温存 : decompose={final=_none medial=_none initial=_none isolated=_none}	س U+FEB2	س U+FEB2
	س U+FEB3	س U+FEB3
	س U+FD0E	س U+FD0E
	س U+FEB4	س U+FEB4

**アラビア文字のタトウィールキャラクタを除去** タトウィールキャラクタ U+0640 (カシーダとも呼ばれます) は、単語を伸ばして行を埋めつくすようにするために頻繁に用いられます。タトウィールはそれ自体は何のテキスト情報も持ちませんので、通常これは、抽出されるテキスト内では必要ではありません。表 6.3 に示すように、*fold* オプションを用いてタトウィールキャラクタを温存することもできます (105 ページの 7.3.1 「Unicode 字形統合」を参照)。

表 6.3 タトウィールキャラクタ U+0640 を *fold* オプションで処理

説明とオプションリスト	字形統合前	字形統合後
アラビア文字のタトウィールキャラクタを除去 : fold オプションなし (デフォルト) または fold={{[U+0640] remove}} または fold={default}	- U+0640	なし
アラビア文字のタトウィールキャラクタ (デフォルトでは除去される) を温存 : fold={{[U+0640] preserve}}	- U+0640	- U+0640

## 6.5 内容分析

PDF 文書は、テキストの個々のキャラクタのページ上における位置のみならず、そのセマンティックス (Unicode マッピング) をも提供します。しかし一般に PDF 文書では、単語・行・コラム・その他高次のテキストユニットに関する情報は伝えてくれません。ページ上のテキストを構成するそれぞれの断片には、個々のキャラクタや音節や行、ないしそれらの任意の混合が含まれている可能性があります、明示的に単語や行やコラムの始まりや終わりを示す印はそこには一切ついていないのです。

さらに悪いことには、ページ上のテキスト断片の順序は、論理的な (読む) 順序とは違っている可能性があります。テキストの各部分をページ上に配置する際には規則など何もないのです。たとえば 2 段組みのページの場合でも、その内容の生成順はまず左コラムの 1 行目、次は右コラムの 1 行目、左コラムの 2 行目、右コラムの 2 行目、…となっているかもしれません。が、論理的な順序ならば、まず左コラム内のテキストをすべて処理した後右コラムのテキストを処理しなければならないわけです。このような文書からテキストを抽出する場合には、ただ PDF ページ上の命令群を再生していただければ、一般には望ましくありません。テキストの論理構造が失われているからです。

TET の内容解析エンジンは、テキスト断片の内容・位置・関係を解析して、以下の目標を達成しようとしています：

- ▶ キャラクタ群から単語を再構成し、単語間に区切りキャラクタを (望まれていれば) 挿入
- ▶ 冗長テキストを除去 (たとえば影付きに見せるためのダブリ等)
- ▶ 複数行にまたがってハイフネーションされている単語の各部を再結合
- ▶ テキストのコラム (区域) を認識
- ▶ 区域内のテキスト断片をソート。ページ内の区域もソート

これらの動作について以下詳しく説明します。こうした内容処理を制御するオプションについても解説します。

**テキストの粒度** `open_page()` で `granularity` オプションを用いると、`get_text()` を 1 回呼び出すごとに返されるテキストの量を指定することができます。

- ▶ `granularity=glyph` と設定すると、各断片はそれぞれ 1 つのグリフをマップした結果を持ちます。これは複数のキャラクタになることもあります (合字の場合等)。このモードでは内容分析は無効にされ、TET は、ページ上の元通りのテキスト断片をその元通りの順序のまま返します。これは最も速いモードではありますが、これが有用なのは、TET クライアント側で精巧な後処理を行おうとするときのみです (あるいはテキストの論理構造には関心がなくて位置にのみ興味があるとき)。なぜならこのモードではテキストがページ全体に散らばってしまっている可能性があるからです。
- ▶ `granularity=word` と設定すると、単語検出機能アルゴリズムがキャラクタ群をまとめて論理的な単語を再構成します。各断片はそれぞれ 1 つの単語を持ちます。孤立した句読点 (カンマ・コロンのクエスチョンマーク・クオート等) はデフォルトでは独立した断片として返され、連続する句読点キャラクタは一個の単語としてグループ化されます (ピリオドキャラクタを連続させて点線のように見せかけている場合等)。ただし、句読点処理は変更することも可能です (後述の「欧文テキストの単語境界検出」を参照)。
- ▶ `granularity=line` と設定すると、単語検出機能によって認識された単語群をまとめて行を再構成します。ハイフン除去が有効のとき (デフォルト) は、行末でハイフンで分割された単語は再結合され、そのハイフン除去された完全な単語はその行に繰り込まれます。

- ▶ `granularity=page` と設定すると、ページ上に含まれるすべての単語が1つの断片で返されます。

複数の単語・行・区域の間には、選択された粒度がその単位より大きければ、それぞれ区切りキャラクタが挿入されます。たとえば `granularity=word` の場合、`TET_get_text()` は呼び出されるごとに単語をきっかり1つずつ返すのですから、区切りキャラクタを挿入する必要などないわけです。

区切りキャラクタを指定するには、`open_page()` の `wordseparator`・`lineseparator` オプションを用います (区切りキャラクタを無効にするには `U+0000` を用います)。たとえば：

```
lineseparator==U+000A
```

デフォルトでは、`granularity=glyph` の場合にはすべての内容分析動作が無効にされ、それ以外の粒度設定では有効にされます。しかし、区切りオプションを用いればよりきめ細かな制御も可能です (後述)。

**欧文テキストの単語境界検出** 単語検出機能は、`glyph` を除くすべての粒度モードで有効にされ、ページ全体にでたらめな順序で散らばっているかもしれない複数のグリフをまとめて論理的な単語を再構成します。欧文テキストの単語境界は2つの判定基準によって認識されます：

- ▶ 精巧なアルゴリズムがグリフどうしの位置関係を解析して、キャラクタのグループを検出し、単語を再構成します。このアルゴリズムはさまざまな属性や特例を考慮して、レイアウトが複雑な場合やページ上のテキスト順序がばらばらな場合でも単語を正確に認識できるよう努めます。
- ▶ スペースや句読点 (コロン・カンマ・ピリオド・括弧等) といったある種のキャラクタは、その幅・位置にかかわらずつねに単語境界と認識されます。`open_page()` の `punctuationbreaks` オプションを `false` に設定すると、単語検出機能は句読点キャラクタを単語境界として扱わなくなります：

```
contentanalysis={punctuationbreaks=false}
```

単語境界検出の際に句読点キャラクタを無視することは、たとえば、Web URL を扱う際に有用でしょう。URL では通常、ピリオド・スラッシュキャラクタは語の一部と見なされるからです (図 6.5 参照)。

**注記** 表意文字キャラクタによるテキストに対する単語境界検出は動作が異なります。詳しくは 87 ページの 6.3.2 「日中韓テキストの単語境界」を参照してください。



図 6.5  
デフォルト設定 `punctuationbreaks=true` では URL は各部に分解されますが (上)、`punctuationbreaks=false` では各部はひとまとまりのまま保持されます (下)。

**ハイフン除去** 行末でハイフン区切りされた単語というものは通常、抽出したテキストを論理レベルで処理したいアプリケーションにとっては好ましくありません。そのため TET はハイフン除去、すなわちハイフン区切りされた単語の各部分の再結合を行います。より正確にいうと、行末の単語の末尾にハイフンキャラクタがあり、かつその次の行の最初の単語の先頭が小文字ならば、ハイフンは除去され、単語の前半部は次行後半部と結合されます。ただし同じ区域内に少なくともあと 1 行存在する場合に限りです。ダッシュキャラクタ（ハイフンではない）は変更されないまま温存されます。ハイフン区切りされた単語の各部分に変更が加えられることはなく、ただハイフンが除去されるだけです。ハイフン除去を無効にするには、`open_page()` で下記のオプションリストを用います：

```
contentanalysis={dehyphenate=false}
```

strategische Grundsätze – der  
der Nutzung von Synergie-  
in Branchen sowie in Unter-  
dukterstellung. So verringert  
bei der Produkterstellung –  
g – seit längerem nicht nur

**注記** 区域の末尾でハイフン区切りされた単語は認識されませんので、ハイフン除去はそれに対しては一切行われません（すなわちそのハイフンはテキストの中に残ります）。

**影付き・擬似太字テキスト除去** PDF 文書はときに、ページの意味内容には貢献しない、ただある種の視覚効果を生み出すためだけの冗長テキストを含んでいます。影付きテキスト効果を生み出すにはたいてい、テキスト本体を少しずつ位置をずらしながら複数複製して重ねるといった方法が採られます。この各層のテキストに不透明な色をつければ、下層のテキストはほとんど隠れ、見える部分が影付き効果を生み出すように見かけ上見えるのです。

同様に、ワープロソフトはときに擬似太字テキスト生成機能に対応しています。ボール

# Introduction

ドフォントが入手不可能なときにもテキストを見かけ上太字に見せかけるため、そのテキストはページ上に同じ色で何度も配置されます。その際にほんの少し位置をずらしていけば、太字テキストであるかのように見せられるわけです。

擬似影付きや擬似太字テキスト、ないし同様の擬似視覚効果は、テキストを抽出して再利用しようとする際に困った問題をひき起こします。ただ視覚効果のためだけの冗長テキスト内容が、ページ内容としては不要であるにもかかわらず一緒に処理されてしまうためです。

単語検出機能が有効にされている場合、デフォルトでは TET はこうした冗長な擬似視覚効果を認識して除去します。影付き除去を無効にするには、`open_page()` で下記のオプションリストを用います：

```
contentanalysis={shadowdetect=false}
```

**アクセント付きキャラクタ** 多くの言語で、アクセントなどの分音記号が、別のキャラクタのそばに配置されて複合キャラクタを形成します。TeX をはじめとするいくつかの組版プログラムでは、2 つのキャラクタ（字母キャラクタとアクセント）を別々に出力して

1 個の複合キャラクタを生成します。たとえば、**ä** キャラクタを生成するために、まず文字 **a** がページ上に配置され、ついで分音符キャラクタ **¨** がその上に配置されます。TET はこの状況を検出し、2 個のキャラクタを再結合して正しい複合キャラクタを形成します。

## 6.6 レイアウト分析

TET は、最も可能性の高いテキスト抽出順序を決定するために、ページ上のテキストのレイアウトを分析します。この自動処理は、いくつかのオプションで支援することも可能です。処理する文書の性質について事前の知識がある場合には、適切なオプションを与えることによってテキスト抽出結果を向上させることができます。

**各種文書スタイル** さまざまなレイアウトやスタイルの文書进行处理するために、いくつかの内部パラメタが利用可能です。たとえば、新聞のページは大量のテキストが多段組になっていることが多く、事業報告書は余白に注釈がしばしば現れる、などです。TET は、いくつかの種類の文書に対する定義済み設定を含んでいます。これらの設定は、下記のように `open_page()` でオプションリストを用いて有効にすることができます：

`docstyle=papers`

入力文書の種類がわかっている場合には、`docstyle` ページオプションおよび（あてはまる場合には）`layouthint` ページオプションの適切な値を与えることを強く推奨します。`docstyle` オプションを与えれば、高度なレイアウト認識アルゴリズムが有効になります。ただし、このオプションに不適切な値を与えてしまうと逆に生成結果が悪化する場合があります。

`docstyle` オプションでは以下の種類が利用可能です（表 6.4 にいくつか典型的な文書スタイルの例を挙げます）：

- ▶ `book`：通常のページによる典型的な書籍レイアウト
- ▶ `business`：ビジネス文書
- ▶ `fancy`：複雑な、またときにイレギュラーなレイアウトを持つ、意匠を凝らしたページ
- ▶ `forms`：構造化されたフォーム
- ▶ `generic`：とくに分類のない最も一般的な文書
- ▶ `magazines`：雑誌の記事。多くは3段組ないしそれ以上で、画像やグラフィックがちりばめられている
- ▶ `papers`：多段組・大紙面・小活字の新聞
- ▶ `science`：科学記事。多くは2段組ないしそれ以上で、画像・式・表組などがちりばめられている
- ▶ `searchengine`：この分類は、入力文書の特定の種類を指すものではなく、TET を検索エンジンのためのインデксаの典型的要請に最適化します。生テキストのみを発出し、処理を高速化するために、いくつかのレイアウト検出機能が無効化されます。たとえば、表組・ページ構造認識が無効化されます。
- ▶ `spacegrid`：この分類は、メインフレームシステムでよく生成されるリスト指向のレポートを対象としています。この種の文書の特徴は、視覚レイアウトが、テキストの明示的な位置合わせではなく、空白キャラクタによって作り出されていることです。この種の文書进行处理の際には、いくつかの処理ステップ（影付き検出など）をスキップしうるので、テキスト抽出が高速化されえます。

最も適切な文書スタイルを選ぶことによって、処理を速め、テキスト抽出結果を向上させられる可能性があります。

**複雑なレイアウト** 文書の種類によっては、非常に凝ったページレイアウトを用いるものがあります。たとえば雑誌や定期行物では、TET はページ上の段組間の関係を正しく決定することができない場合があります。このような状況では、処理時間を長くかけるこ



表 6.4 いろいろな文書スタイル

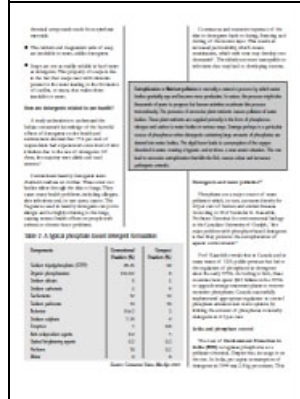
docstyle=book



docstyle=business



docstyle=fancy



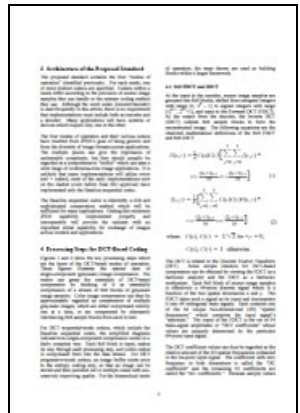
docstyle=magazines



docstyle=papers



docstyle=science



docstyle=spacegrid

2007 Census of Population and Housing - Summary File 1

Table W1. Race: Male 18+ (Single with partner)

18+ Male 18+ (Single with partner) by race and marital status

Race	Total		Married		Never married	
	Male	Female	Male	Female	Male	Female
White	1,011,191	1,011,191	687,144	687,144	324,047	324,047
Black	142,142	142,142	94,761	94,761	47,381	47,381
Hispanic	142,142	142,142	94,761	94,761	47,381	47,381
Other	142,142	142,142	94,761	94,761	47,381	47,381

... [Additional tables and data points follow in a similar format]

とにより、抽出されるテキストを改善することが可能です。これは *structureanalysis*・*layoutanalysis* ページオプションで制御できます。例：

```
structureanalysis={list=true bullets={{fontname=ZapfDingbats}}}  
layoutanalysis = {layoutrowhint={full separation=preservecolumns}}  
layoutdetect=2  
layouteffort=high
```

**表組検出** TET は、ページ上の表構造を検出して、表組の内容を表行・列・セルに構造化します。表組に関してページ上で検出された情報は API によって直接提供されることはなく、下記の例のように TETML 出力内でのみ利用可能となります：

```
<Table>  
<Row>  
<Cell colSpan="5">  
  <Para>  
    <Word>  
      <Text>5</Text>  
      <Box llx="317.28" lly="637.14" urx="324.59" ury="650.29"/>  
    </Word>  
    <Word>  
      <Text>.</Text>  
      <Box llx="324.60" lly="637.14" urx="328.25" ury="650.29"/>  
    </Word>  
    <Word>  
      <Text>REFERENCES</Text>  
      <Box llx="335.04" lly="637.14" urx="407.64" ury="647.47"/>  
    </Word>  
  </Para>  
</Cell>  
</Row>  
...  
</Table>
```

# 7 高度な Unicode 処理

## 7.1 Unicode のさまざまな重要概念

この節では、Unicode に関する基本的な情報を提供します。TET におけるテキスト処理は Unicode 規格に大いに依存しているからです。Unicode Web サイトではさらに詳しい情報が大量に提供されています：

[www.unicode.org](http://www.unicode.org)

**キャラクタとグリフ** テキストを扱う際には以下の概念をはっきり区別することが重要になります：

- ▶ **キャラクタ**は、言語の中で情報を伝える最小の単位です。代表例としてはラテンアルファベットの音素文字や、日本語の音節文字、中国語の表意文字が挙げられます。キャラクタは意味を持ちます。すなわち意味的な実体であるといえます。
- ▶ **グリフ**は、1つないし複数のキャラクタを表すさまざまな図形です。グリフは外観を持ちます。すなわち表象的な実体であるといえます。

キャラクタとグリフとの間には一対一の対応は存在しません。たとえば、合字は1つのグリフですが複数のキャラクタで表されます。その一方では、1つのグリフを場面に応じて異なるキャラクタを表すために使いまわしたりすることもあります（つまりキャラクタどろしの形が同じという場合が存在します。図 7.1 参照）。

TET の Unicode 後処理によって、グリフ群と生成キャラクタ群との関係はさらに変わることもあります。たとえば、分解によって1個のキャラクタが複数のキャラクタに変換されることがあり、また、字形統合によってキャラクタが除去されることもあります。こうした理由から、キャラクタとグリフの間にはいかなる特定の関係をも前提することはできません。

**BMP と PUA** Unicode ベースの環境では以下の用語が頻出します：

キャラクタ

グリフ

U+0067 LATIN SMALL LETTER G

U+0066 LATIN SMALL LETTER F +  
U+0069 LATIN SMALL LETTER I

U+2126 OHM SIGN または  
U+03A9 GREEK CAPITAL LETTER OMEGA

U+2167 ROMAN NUMERAL EIGHT または  
U+0056 V U+0049 I U+0049 I U+0049 I

図 7.1  
グリフとキャラクタ  
の関係

- ▶ **基本多言語面 (BMP)** は、Unicode 範囲 U+0000 ~ U+FFFF 内のコード点から成ります。Unicode 規格は、さらに多くのコードを追加面群に、すなわち範囲 U+10000 ~ U+10FFFF 内に含んでいます。
- ▶ **私用領域 (PUA)** は、私用のために予約されているいくつかの範囲の一つです。Unicode 規格はこの範囲内にいかなるキャラクタも指定していませんので、PUA コード点は一般的なやりとりには使えません。基本多言語面は PUA を範囲 U+E000 ~ U+F8FF 内に含んでいます。第十五面 (U+F0000 ~ U+FFFFFD) と第十六面 (U+100000 ~ U+10FFFFD) はまるごと私用のために予約されています。

**さまざまな Unicode 符号化形式 (UTF 形式)** Unicode 規格では、各キャラクタに番号 (コード点) を割り当てています。これらの番号をコンピュータ処理で利用するためには、これらを何らかの方法で表す必要があります。Unicode 規格ではこれを符号化形式と呼びます (以前は変換形式と呼んでいました)。この用語はフォントの符号化方式 (エンコーディング) と混同してはいけません。Unicode では以下の符号化形式を定義しています：

- ▶ **UTF-8** : これは、コード点を 1 ~ 4 バイトで表す可変長形式です。範囲 U+0000 ~ U+007F 内の ASCII キャラクタは範囲 00 ~ 7F 内のシングルバイトで表されます。範囲 U+00A0 ~ U+00FF 内の Latin-1 キャラクタは 2 バイトで表され、その第一バイトは必ず 0xC2 か 0xC3 (これらの値は Latin-1 では  $\tilde{A} \cdot \tilde{A}$  を表します) になります。
- ▶ **UTF-16** : 基本多言語面 (BMP) 内のコード点を 16 ビット値 1 個で表します。追加面群内のコード点、すなわち範囲 U+10000 ~ U+10FFFF 内のものは、16 ビット値の対で表します。この対をサロゲートペア (代用対) といいます。サロゲートペア 1 個は、範囲 D800 ~ DBFF 内の高位サロゲート値 1 個と、範囲 DC00 ~ DFFF 内の低位サロゲート値 1 個から成ります。高・低サロゲート値はサロゲートペアの一部としてのみ現れることができ、それ以外の場面では現れることができません。
- ▶ **UTF-32** : 各コード点を 32 ビット値 1 個で表します。

**Unicode 符号化スキームと BOM (Byte Order Mark)** コンピュータアーキテクチャによってバイトの順序は異なります。すなわち、大きな値 (16 ビットや 32 ビット) を構成するバイト群を、最高位バイトを先頭に格納するか (ビッグエンディアン)、それとも最低位バイトを最初に格納するか (リトルエンディアン) という違いがあります。ビッグエンディアンアーキテクチャのよく知られた例は PowerPC であり、一方 x86 アーキテクチャはリトルエンディアンです。UTF-8 と UTF-16 はシングルバイトよりも大きな値に基づいていますので、バイト順序を考慮する必要がここで生じてきます。符号化スキーム (上述の符号化形式とは異なることに留意) は、符号化形式に加えてバイト順序を指定します。たとえば、UTF-16BE はビッグエンディアンバイト順序の UTF-16 を意味します。バイト順序が事前にわからない場合には、コード点 U+FEFF を用いて指定することもできます。これを BOM (Byte Order Mark) といいます。UTF-8 では BOM は必須ではありませんが、存在することもでき、その場合はバイト列を UTF-8 として識別するために用いることができます。表 7.1 に、さまざまな符号化形式における BOM の表現を挙げます。

表 7.1 さまざまな Unicode 符号化形式における BOM

符号化形式	BOM (16 進)	WinAnsi での視覚表現 <sup>1</sup>
UTF-8	EF BB BF	ï»¿
UTF-16 big-endian	FE FF	þÿ
UTF-16 little-endian	FF FE	ÿþ
UTF-32 big-endian	00 00 FE FF	■■þÿ
UTF-32 little-endian	FF FE 00 00	ÿþ■■

1. 黒四角■は null バイトの意。

**組文字とキャラクタ列** グリフのなかには、複数のキャラクタの列にマップされるものがあります。たとえば合字は、その構成キャラクタ群に従って複数のキャラクタにマップされます。しかし組文字は分割してよい場合とよくない場合があります（図 7.1 のローマ数字等）、それはフォント内・PDF 内の情報によって決まるとともに、*decompose* 文書オプションにも依存します（105 ページの 7.3 「Unicode 後処理」を参照）。

適切ならば、TET は組文字を構成キャラクタ列に分割します。このキャラクタ列は、*get\_text()* が返すテキストに組み込まれます。各キャラクタごとに、その元となったグリフ（群）に関する情報は *get\_char\_info()* で得ることができますが、その際、そのキャラクタがキャラクタ列内の先頭なのかそれとも後続なのかという情報も一緒に得られるようになっています。位置情報はキャラクタ列内の先頭キャラクタに対してのみ返されます。キャラクタ列内の後続キャラクタは固有の位置情報・幅情報というものを一切持たないので、先頭キャラクタと併せて処理する必要があります。

**対応グリフを持たないキャラクタ** ページ上のグリフは、そのすべてが 1 つないし複数の Unicode キャラクタにマップされますが、逆に TET が出力するキャラクタは、そのすべてが実在のグリフにマップされているとは限りません。対応するグリフを持つキャラクタを有形キャラクタ（real character）と呼び、持たないものを無形キャラクタ（artificial character）と呼びます。このような、直接対応するグリフがないのに出力される無形キャラクタは、いくつかの種類に分類できます。

- ▶ 組文字（上述）は複数の Unicode キャラクタの列にマップされますが、このとき列内の先頭キャラクタは実在のグリフに対応しますが、残りのキャラクタは対応グリフを持ちません。
- ▶ *linseparator/wordseparator* オプションで挿入される区切りキャラクタは、対応グリフを持たない無形キャラクタです。

## 7.2 Unicode 前処理（フィルタリング）

TET は、有用でなさそうなテキストを除去するためにいくつかのフィルタを適用します。これらのフィルタはテキストを、いかなる Unicode 後処理ステップをも適用する前に変更します。いくつかのフィルタはつねに有効であり、それ以外は単語検出機能を必要とします。その `granularity=word` 以上の場合にのみ有効です。

### 7.2.1 すべての粒度のためのフィルタ

以下のフィルタはすべての粒度で使用できます。

**異常な文字サイズのテキスト** 非常に小さなテキストや非常に大きなテキストは無視させることもできます。たとえばページ背景の大きなキャラクタです。上限・下限は `fontsize` ページオプションで制御できます。デフォルトでは、すべての文字サイズのテキストが抽出されます。

下記のページオプションは、抽出するテキストの文字サイズの範囲を 10 ~ 50 ポイントに制限します。これを外れている文字サイズのテキストは無視されます：

```
fontsize={10 50}
```

**不可視テキスト** 不可視テキスト（すなわち `textrendering=3` のテキスト）はデフォルトでは抽出されます。PDF 内のテキストは、`textrendering` プロパティ以外にもさまざまな理由で見えなくなっている可能性があることに留意してください。たとえばテキストの色が背景色と同一であるかもしれませんし、テキストがページ上の他のオブジェクトに隠れているかもしれません。ここで記述する動作は `textrendering=3` のテキストにのみ効力を持ちます。この PDF 技法は OCR の生成テキストに対して広く用いられています。スキャンされたラスタ画像の「後ろ」にテキストが見えない形で存在しているのです。

不可視テキストは、`get_char_info()` が返す `TET_char_info` 構造の `textrendering` メンバによって識別することもできますし（189 ページの表 10.15 を参照）、あるいは TETML 内の `Glyph/@textrendering` 属性によって識別することもできます。

不可視テキストを無視したいときは下記のページオプションを用います：

```
ignoreinvisibletext=true
```

**特定フォント名またはフォント種別のテキストを完全無視** 場合によっては、1 個ないし複数のフォントを名前で指定してそのテキストを完全に無視させることが有用でしょう。たとえば、意味のあるテキストをまったく構成しない記号フォントなどです。あるいは、問題のあるフォントをフォント種別で指定することもできます。これは、装飾のために用いられることのある Type 3 フォントに対して主に有用です。このフィルタは、`glyphmapping` 文書オプションの `remove` サブオプションによって制御できます。

たとえば、Type 3 フォントのテキストをすべて無視：

```
glyphmapping={{fonttype={Type3} remove}}
```

Webdings・Wingdings・Wingdings 2・Wingdings 3 フォントのテキストをすべて無視：

```
glyphmapping={{fontname=Webdings remove} {fontname=Wingdings* remove}}
```

フォント名に対する条件とフォント種別に対する条件とを組み合わせることも可能です。たとえば、文字 A で始まるすべての Type 3 フォントのテキストを無視：

```
glyphmapping={{fonttype={Type3} fontname=A* remove}}
```

## 7.2.2 粒度 word 以上のためのフィルタ

以下のフィルタは `granularity=word · line · page` でのみ使用できます。

**ハイフン除去** ハイフン除去は、ハイフンを削除して、ハイフンで分割されていた単語の各部分を結合させます。

単語を複数行に分割するために用いられているハイフンは、`TET_char_info` 構造の `attributes` メンバによって識別することもできますし (189 ページの表 10.15 を参照)、あるいは TETML 内の `Glyph/@hyphenation` 属性によって識別することもできます。

ハイフン除去は下記のページオプションによって無効化することができます：

```
contentanalysis={dehyphenate=false}
```

**ハイフン報告** ハイフン除去が有効になっているとき、生成されるグリフリスト内、すなわち `get_char_info()` が返すグリフのリスト内で、あるいは TETML 内の `Glyph` 要素群で、ハイフネーションされた単語の各部の間のハイフンキャラクタが報告されるかどうかを選ぶことができます。デフォルトではハイフンは除去されます。

しかし、応用によっては、ページ上のハイフンの正確な位置を知る必要がある場合があります。たとえば、TET クックブックの `highlight_search_terms · search_and_replace_text` トピックでは、元の単語の上に注釈または置換テキストをかぶせる際にハイフングリフを考慮に入れています。このような状況の場合には、下記のページオプションを用いれば、ハイフン除去処理によって検知されたすべてのハイフンを含めるよう TET に指示することができます：

```
contentanalysis={keeplyphenglyphs=true}
```

ハイフンは、`get_char_info()` が返す `TET_char_info` 構造内の `attributes` メンバの `TET_ATTR_DEHYPHENATION_ARTIFACT` フラグによって識別することもできますし (189 ページの表 10.15 を参照)、あるいは、TETML 内で `Glyph/@dehyphenation` 属性の値 `artifact` で識別することもできます。

**影付き除去** 影付きや擬似太字といった視覚効果のためだけの冗長テキストは削除されます。

影付き・擬似太字テキストは、`TET_char_info` 構造の `attributes` メンバによって識別することもできますし (189 ページの表 10.15 を参照)、あるいは TETML 内で `Glyph/@shadow` 属性によって識別することもできます。

影付き除去は下記のページオプションで無効化できます：

```
contentanalysis={shadowdetect=false}
```

**マップなしグリフ** Unicode へマップできないグリフは、私用領域内のキャラクタへ置き換えられます (115 ページの「マップ不能グリフ」の項を参照)。場合によっては、使える Unicode 値をグリフに割り当てるために必要な十分な情報を PDF 文書が含んでいない (あるいは情報があっても整合していない) ことがあります。このような場合には、`unknownchar` 文書オプションで指定されたキャラクタが割り当てられます。

すべての PUA キャラクタは、デフォルトでは Unicode 未知キャラクタ U+FFFF へ置き換えられます。この動作は、`fold` 文書オプションで変えることもできます。下記のオプションリストは、すべての未知のキャラクタを、すなわち PUA キャラクタと、有用な Unicode 値を決定できなかったキャラクタとを、除去します：

```
fold={{[:Private_Use:] remove} {[U+FFFF] remove} default}
```

マップなしグリフ（すなわち、ページ上では見えているが、TET で抽出できないキャラクタ）は、*TET\_char\_info* 構造の *unknown* メンバによって識別することもできますし、あるいは、TETML 内で *Glyph/@unknown* 属性で識別することもできます。



## 7.3 Unicode 後処理

TET は、抽出したテキストを構成する Unicode キャラクタ群を微調整するためのさまざまな制御手段を提供しています。この節で述べる各種の後処理ステップは、Unicode 規格で定義されているものです。これらは TET で利用可能であり、以下の順序で処理されます：

- ▶ 字形統合は、**fold** 文書オプションによって制御され、特定のキャラクタ群に対して温存・除去・置換のいずれかを行います。例：単語の分割に用いられているハイフンを除去。アラビア文字のタトウィールキャラクタを除去。
- ▶ 分解は、**decompose** 文書オプションによって制御され、1個のキャラクタを1個ないし複数の等価なキャラクタへ置き換えます。例：合字を分割。全角英数・記号をおのの対応する非全角キャラクタへマップ。
- ▶ 正規化は、**normalize** 文書オプションによって制御され、テキストを正規化 Unicode 形式の一つへ変換します。例：字母キャラクタと分音キャラクタを結合して通用キャラクタ化。オーム記号をギリシャ文字オメガへマップ。

### 7.3.1 Unicode 字形統合

字形統合は、1個ないし複数の Unicode キャラクタを処理して、各キャラクタに対して特定の操作を適用します。以下の操作が利用可能です：

- ▶ キャラクタを温存。
- ▶ キャラクタを除去。
- ▶ 別の（固定の）キャラクタへ置き換え。

字形統合は連鎖されません：字形統合の出力が、利用可能な字形統合によってさらに処理されることはありません。字形統合は、Unicode テキスト出力に対してのみ効力を持ち、**TET\_char\_info** 構造内で、または TETML の **<Glyph>** 要素群で報告されるグリフの集合に対しては効力を持ちません。たとえば、ある Unicode キャラクタ群が字形統合によって除去された場合でも、その元のキャラクタ群を生み出したそれに対応するグリフ群は報告されます。

読みやすくするために、以下の表の中の例では、**fold** オプションリストのサブオプションを個別に記しています。複数の字形統合を適用したいときには、これらのサブオプションを連結して単一の大きな**fold**オプションリストにする必要があることに留意してください。**fold** オプションを複数回与えてはいけません。たとえば、下記は誤りです：

```
fold= { [:blank:] U+0020 } fold= { [_dehyphenation remove] }      誤り!
```

下記のオプションリストは、複数の字形統合に対する正しい文法を示しています：

```
fold= { [:blank:] U+0020 } {[_dehyphenation remove] }
```

**字形統合のさまざまな例** 表 7.2 に、字形統合のさまざまな応用を演示するさまざまな **fold** オプションの例を挙げます。これらのサンプルのオプションは、**open\_document()** に対するオプションリストの中で与える必要があります。TET は、全 Unicode キャラクタか

ら選択された部分集合に対して字形統合を適用することもできます。これを Unicode 集合といいます。その文法は、154 ページの「Unicode 集合」で解説しています。

表 7.2 さまざまな fold オプションの例

説明とオプションリスト	字形統合前	字形統合後
<b>1 個の Unicode 集合内のすべてのキャラクタを除去</b>		
ISO 8859-1 (Latin-1) 内のキャラクタのみを出力内に保持、すなわち、基本 欧文ブロック外のすべてのキャラクタを除去： fold={{[^U+0020-U+00FF] remove}}	A U+0104	なし
すべての非アルファベットキャラクタ (句読点・数字など) を除去： fold={{[:Alphabetic=No:] remove}}	7 U+0037 A U+0041	なし A U+0041
数字以外のすべてのキャラクタを除去： fold={{^[[:General_Category=Decimal_Number:]] remove}}	7 U+0037 A U+0041	7 U+0037 なし
すべての未知キャラクタを、すなわち、PUA キャラクタ群と、有用な Uni- code 値が決定できなかったキャラクタ群とを除去 (残りのデフォルト字形統 合は再有効化されます)： fold={{[:Private_Use:] remove} {[U+FFFF] remove} default}	U+FFFF	なし
すべてのダッシュ約物キャラクタを除去します： fold={{[:General_Category=Dash_Punctuation:] remove}}	- U+002D	なし
すべての双方向制御キャラクタを除去します： fold={{[:Bidi_Control:] remove}}	U+200E	なし
<b>1 個の Unicode 集合内のすべてのキャラクタを 1 個の特定のキャラクタへ置き換え</b>		
空白字形統合：Unicode 空白キャラクタのすべての種類を U+0020 へマップ： fold={{[:blank:] U+0020}}	U+00A0	U+0020
ダッシュ字形統合：Unicode ダッシュキャラクタのすべての種類を U+002D へ マップ：fold={{[:Dash:] U+002D}}	- U+2011	- U+002D
すべての未割り当てキャラクタ (すなわち、キャラクタが割り当てられてい ない Unicode コード点) を U+FFFD へマップ：fold={{[:Unassigned:] U+FFFD}}	U+03A2	◆ U+FFFD
<b>個別のキャラクタに対して特別処理</b>		
改行位置のすべてのハイフンキャラクタを温存しつつ、残りのデフォルト字 形統合を保持。これらのキャラクタは TET によって内部的に識別されていま すので (固定の Unicode プロパティを持つのではなく)、この字形統合のド メインを指定するにはキーワード <code>_dehyphenation</code> を用います： fold={{_dehyphenation preserve}}	- U+002D	- U+002D
アラビア文字のタトウィールキャラクタ (デフォルトでは除去されます) を 温存：fold={{[U+0640] preserve}}	- U+0640	- U+0640
さまざまな約物キャラクタをおのおの対応する ASCII キャラクタへ置き換 え： fold={{ {[U+2018] U+0027} {[U+2019] U+0027} {[U+201C] U+0022} [U+201D] U+0022}}	" U+201C	" U+0022

**デフォルト字形統合** `granularity=glyph` の場合を除き、TET はデフォルトで表 7.3 に挙げる字形統合をすべて適用します。カスタムの字形統合を初期デフォルト字形統合と結合するには、カスタム字形統合オプション群の後にキーワード *default* を与える必要があります。例：

```
fold={ { _dehyphenation preserve} default }
```

すべてのデフォルト字形統合を明示的に無効化したい場合でない限り、*fold* オプションリストにはキーワード *default* を追加することを推奨します。

表 7.3 *fold* オプションのデフォルト値一覧

説明とオプションリスト	入力例	出力
空白字形統合：Unicode 空白キャラクタのすべての種類を U+0020 ヘマッパ： fold={{[:blank:] U+0020}}	U+00A0	U+0020
私用領域 (PUA) 内のすべてのキャラクタを未知キャラクタ (デフォルトではこれは U+FFFD ですが、 <code>unknownchar</code> オプションで変更することもできます) ヘマッパ：fold={{[:Private_Use:] unknownchar}}	U+E001	◆ U+FFFD
ハイフン除去された単語の中のすべてのハイフンを除去： fold={{_dehyphenation remove}}	- U+002D	なし
アラビアモジのタトウォールキャラクタをすべて除去： fold={{[U+0640] remove}}	- U+0640	なし
すべての制御キャラクタと、Unicode で割り当てられていないすべてのキャラクタとを除去 (TETML 出力を生成する際には、これらの字形統合は、他のすべての字形統合の後つねに行われます)： fold={{[:Control:] remove} {[:Unassigned:] remove}}	U+000C U+03A2	なし

## 7.3.2 Unicode 分解

分解は、1 個のキャラクタを、他の 1 個ないし複数の等価なキャラクタ列へ置き換えます。ある Unicode キャラクタが、他の 1 個ないし複数のキャラクタと実際には同じ意味でありながら、歴史的理由により Unicode 内で別々のコードを与えられているとき（多くはレガシ符号化形式との可逆のからみで）、その Unicode キャラクタはそのキャラクタ（群）と（互換あるいは正準）等価であるといいます。分解は情報を破壊します。これは、元のキャラクタとその等価キャラクタとの使い分けに関心がない場合には有用です。しかし使い分けに関心がある場合には、その分解は適用するべきではありません。Unicode 分解に関する完全な説明は下記を参照してください：

[www.unicode.org/versions/Unicode5.2.0/ch03.pdf#G729](http://www.unicode.org/versions/Unicode5.2.0/ch03.pdf#G729)

**注記** ここで用いている「分解」という用語は、Unicode 規格で定義されているものですが、実際には多くの分解は、1 個のキャラクタを複数の部分へ分割するのではなく、1 個のキャラクタを別の 1 個のキャラクタへ変換します。

**正準分解** 互いに正準等価であるキャラクタまたはキャラクタ列は、同一の抽象キャラクタを表現しており、したがって必ず同一の体裁と動作を持つべきものです。よくある例としては、合成済みキャラクタ（例： $\text{Ä}$ ）と連結キャラクタ列（例： $\text{A} \text{¨}$ ）が挙げられます： $\text{Ä}$  (U+00C4) と  $\text{A} \text{¨}$  (U+0041 U+0308) が挙げられます：2 つの表現は互いに正準等価です。1 つの表現から別の表現へ切り替えても情報は失われません。正準分解は、1 つの表現を、正準表現と見なされる別の表現へ置き換えます。

Unicode コードチャート<sup>1</sup>では、正準マッピングには記号 IDENTICAL TO  $\equiv$  (U+2261) が付けられています（キャラクタテーブルにはなし）。分解名 *<canonical>* と暗黙的に見なされません。表 7.4 にいくつかの例を挙げます。

表 7.4 正準分解：decompose オプションのサブオプション（正準等価なキャラクタには、Unicode コードチャートで記号 IDENTICAL TO  $\equiv$  (U+2261) が付けられています）

分解名	説明	分解前	分解後
<i>canonical</i> <sup>f</sup>	正準分解	$\text{Ä}$ U+00C0	$\text{A} \text{¨}$ U+0041 U+0300
		$\text{林}$ U+F9F4	$\text{林}$ U+6797
		$\text{Ω}$ U+2126	$\text{Ω}$ U+03A9
		$\text{ば}$ U+3070	$\text{は} \text{¨}$ U+306F U+3099
		$\text{ℵ}$ U+FB2F	$\text{ℵ}$ U+05D0 U+05B8

1. デフォルトではこの分解は、特定のキャラクタ群を温存するために、すべてのキャラクタに対しては適用されません。詳しくは、107 ページの「デフォルト字形統合」を参照してください。

1. [www.unicode.org/Public/5.2.0/charts/](http://www.unicode.org/Public/5.2.0/charts/) を参照。

**互換分解** 互換等価なキャラクタどうしは、同一の抽象キャラクタを表しますが、その体裁や動作は互いに異なっている可能性があります。例としては、アラビア文字キャラクタの単独形 (س) U+0633 等) と位置依存表示形 (س · س · س) U+FEB2 U+FEB4 U+FEB3 等) が挙げられます。互換等価キャラクタは組版上互いに異なるものです。この組版情報を除去すると、情報が失われる可能性があります。ある種の応用の場合 (検索等) には処理を単純化できるでしょう。互換分解は組版情報を除去します。

Unicode コードチャートでは、互換マッピングには記号 ALMOST EQUAL TO U+2248 が付いており、その後、分解名 (「タグ」ともいいます) が `<noBreak>` のように山括弧にくくって書かれています。タグ名が書かれていないものについては、`<compat>` と見なされます。このタグ名は、表 7.5 のオプション名と同一です。いくつかの例に見られるように、分解の結果、1 個のキャラクタが複数キャラクタ列へ変換されることもあります。

注記 表 7.5 の項目はすべて互換分解を記しているのに対し、「compat」タグは「他の」互換分解のみを、すなわち特定の名前を持たないもののみを内容として持ちます。

注記 PDF 文書によっては、グリフが、分解されていない Unicode 値へではなく、分解済みキャラクタ列へすでにマップされている場合もあることに留意してください。その場合には `decompose` オプションは出力に対して効力を持ちません。

**分解のさまざまな例** TET における分解は、文書オプション `decompose` で制御できます。1 種類の分解が、全 Unicode キャラクタに対してではなく、いくつかのキャラクタに対してのみ適用されるよう制限することもできます。ある分解の適用対象となる部分集合を、そのドメインといいます。表 7.5 に、すべての Unicode 分解に対するサブオプションを例とともに挙げます。

以下の `decompose` オプションの例は、`open_document()` に対するオプションリストの中で与える必要があります。`decompose` オプションリスト内の分解名は表 7.5 から取られます。

すべての分解を無効化：

```
decompose={none}
```

全角 (ダブルバイト)・半角キャラクタを温存：

```
decompose={wide=_none narrow=_none}
```

すべての正準等価キャラクタをおのおの対応キャラクタへマップ：

```
decompose={canonical=_all}
```

下記のオプションリストは `circle` 分解を有効化しますが、それ以外の分解はすべて無効化します：

```
decompose={none circle=_all}
```

反対に、下記のオプションリストはすべての分解を有効化します (なぜなら、他のオプションを省略するとデフォルトが有効になるからです)：

```
decompose={circle=_all}
```

表 7.5 互換分解 : decompose オプションのサブオプション一覧 (互換等価なキャラクタには、Unicode コードチャートで記号 ALMOST EQUAL TO  $\approx$  U+2248 が付けられています)

分解名	説明	分解前	分解後 (論理順で)
<i>circle</i>	丸囲みキャラクタ	① U+3251	2 1 U+0032 U+0031
<i>compat</i> <sup>1</sup>	その他の互換分解。例：広く用いられている合字	fi U+FB01	f i U+0066 U+0069
<i>final</i>	語尾形。特にアラビア文字の	س U+FEB2	س U+0633
<i>font</i>	フォントバリエーション。例：数学の集合の文字、ヘブライ文字の合字	Ⓒ U+2102	Ⓒ U+0043
<i>fraction</i> <sup>1</sup>	俗用分数字体	¼ U+00BC	1 / 4 U+0031 U+2044 U+0034
<i>initial</i>	語頭形。特にアラビア文字の	س U+FEB3	س U+0633
<i>isolated</i>	単独形。特にアラビア文字の	سر U+FD0E	ر س U+0633 U+0631
<i>medial</i>	語中形。特にアラビア文字の	س U+FEB4	س U+0633
<i>narrow</i>	半角キャラクタ	ㄣ U+FF66	ㄣ U+30F2
<i>nobreak</i>	改行禁止キャラクタ		
		U+00A0	U+0020
<i>none</i>	明示的に decompose オプションリストで指定されていないすべての分解を無効化		(すべてのキャラクタを変更せず温存)
<i>small</i>	CNS 11643 互換のための小型字体	Ꞁ U+FE50	Ꞁ U+002C
<i>square</i>	日中韓の組文字	ㄱ U+3314	ㄱ ㄴ U+30AD U+30ED
<i>sub</i> <sup>1</sup>	下付き字体	₁ U+2081	₁ U+0031
<i>super</i> <sup>1</sup>	上付き字体	ₐ U+00AA ™ U+2122	ₐ U+0061 ™ M U+0054 U+004D
<i>vertical</i>	縦書き字体	⸍ U+FE37	{ U+007B
<i>wide</i>	全角字体	₤ U+FFE1	₤ U+00A3

1. デフォルトではこの分解は、特定のキャラクタ群を温存するために、すべてのキャラクタに対しては適用されません。詳しくは、107 ページの「デフォルト字形統合」を参照してください。

**デフォルト分解** デフォルトでは、*fraction* 以外のすべての分解が有効になっています。多くのデフォルト分解は *\_all* ドメインに対して（すなわちすべてのキャラクタに対して）適用されますが、いくつかは、表 7.6 に従った、より小さなデフォルトドメインに対して適用されます。分解を扱う正攻法は正規化によることです（112 ページの 7.3.3 「Unicode 正規化」を参照）。*granularity=glyph* のときは、Unicode 後処理は完全に無効化されますので、その場合には分解はいずれも有効にはなりません。

表 7.6 Unicode 分解（decompose オプションのサブオプション）のデフォルトドメイン一覧

分解	TET におけるデフォルト
<i>canonical</i>	canonical={ [U+0374 U+037E U+0387 U+1FBE U+1FEF U+1FFD U+2000 U+2001 U+2126 U+212A U+212B U+2329-U+232A] } デフォルトドメインは正準重複（シングルトン）を含みますが、それ以外の正準等価キャラクタを含みません。デフォルトが <i>_all</i> でないのは、 $\text{Ä}$ <small>U+00C4</small> のようなキャラクタを温存するためです。
<i>compat</i>	compat={ [U+FB00-U+FB17] } デフォルトドメインは欧文とアルメニア文字の合字を含みますが、それ以外の互換キャラクタを含みません。デフォルトが <i>_all</i> でないのは、 $\text{IJ}$ <small>U+0132</small> のようなキャラクタを温存するためです。
<i>fraction</i>	fraction= <i>_none</i> 分数はデフォルトでは分解されません。分解すると整数部と分数部が合わさって不適切な数字列になるおそれがあるからです。たとえば、クライアントアプリケーションがキャラクタ列 $\text{9 } \frac{1}{2}$ <small>U+0039 U+00BD</small> （数値 9.5 を表す）を $\text{9 } \frac{1}{2}$ <small>U+0039 U+0031 U+2044 U+0032</small> と勘違いして、数値 $(91)/2=45.5$ と解釈してしまうおそれがあります。
<i>sub</i> <i>super</i>	sub={ [U+208A-U+208E] } super={ [U+207A-U+207E] } デフォルトドメインは数学記号のみを含みます。上付き・下付き数字はデフォルトでは分解されません。これは、上で <i>fraction</i> について述べたのと同様の数値解釈上の問題を避けるためです。商標記号 $\text{TM}$ <small>U+2122</small> のようなキャラクタはデフォルトでは $\text{T M}$ <small>U+0054 U+004D</small> へ分解されません。
<i>all others</i>	circle= <i>_all</i> final= <i>_all</i> ... vertical= <i>_all</i> wide= <i>_all</i> その他のすべての分解は、デフォルトですべてのキャラクタに対して有効になっています。

### 7.3.3 Unicode 正規化

Unicode 規格では、正準等価と互換等価の概念に基づき（これらは 108 ページの 7.3.2 「Unicode 分解」で説明しています）、4 種類の正規形を定義しています。すべての正規形は、結合記号を特定の順序に置き、分解と合成を異なる方式で適用します：

- ▶ 正規形 C (NFC) は、正準分解の後に正準合成を適用します。
- ▶ 正規形 D (NFD) は、正準分解を適用します。
- ▶ 正規形 KC (NFKC) は、互換分解の後に正準合成を適用します。
- ▶ 正規形 KD (NFKD) は、互換分解を適用します。

正規形は、Unicode 規格付録 #15 「Unicode Normalization Forms」で定義されています ([www.unicode.org/versions/Unicode5.2.0/cho3.pdf#G21796](http://www.unicode.org/versions/Unicode5.2.0/cho3.pdf#G21796)・[www.unicode.org/reports/tr15/](http://www.unicode.org/reports/tr15/)を参照)。

TET は 4 種類の Unicode 正規形すべてに対応しています。Unicode 正規化は *normalize* 文書オプションで制御できます。例：

```
normalize=nfc
```

TET はデフォルトでは正規化を適用しません。*decompose* オプションと *normalize* オプションはかち合うおそれがあることから、*normalize* オプションを *none* 以外の値に設定するとデフォルト分解は無効化されます。

正規形の選択はアプリケーションの要請に依存します。たとえば、いくつかのデータベースはテキストを NFC で受け付けます。NFC は、Web 上の Unicode テキストについて推奨される形式でもあります。表 7.7 に、さまざまなキャラクタに対する正規化の効果を示します。

表 7.7 Unicode 正規形：さまざまな例

正規化前	NFC	NFD	NFKC	NFKD
Ä U+00C4	Ä U+00C4	À ¨ U+0041 U+0308	Ä U+00C4	À ¨ U+0041 U+0308
À ¨ U+0041 U+0308	Ä U+00C4	À ¨ U+0041 U+0308	Ä U+00C4	À ¨ U+0041 U+0308
¨ À U+0308 U+0041	¨ À U+0308 U+0041	¨ À U+0308 U+0041	¨ À U+0308 U+0041	¨ À U+0308 U+0041
fi U+FB01	fi U+FB01	fi U+FB01	f i U+0066 U+0069	f i U+0066 U+0069
3 5 U+0033 U+2075	3 5 U+0033 U+2075	3 5 U+0033 U+2075	3 5 U+0033 U+0035	3 5 U+0033 U+0035
Å U+212B	Å U+00C5	À ° U+0041 U+030A	Å U+00C5	À ° U+0041 U+030A
TM U+2122	TM U+2122	TM U+2122	T M U+0054 U+004D	T M U+0054 U+004D
IV U+2163	IV U+2163	IV U+2163	I V U+0049 U+0056	I V U+0049 U+0056



表 7.7 Unicode 正規形 : ささまざまな例

正規化前	NFC	NFD	NFKC	NFKD
ㄱ U+FB48	ㄱ U+05E8 U+05BC	ㄱ U+05E8 U+05BC	ㄱ U+05E8 U+05BC	ㄱ U+05E8 U+05BC
가 U+AC00	가 U+AC00	ㄱ ㅏ U+1100 U+1161	가 U+AC00	ㄱ ㅏ U+1100 U+1161
ぢ U+3062	ぢ U+3062	ぢ ㄱ U+3061 U+3099	ぢ U+3062	ぢ ㄱ U+3061 U+3099
10月 U+32C9	10月 U+32C9	10月 U+32C9	1 0 月 U+0031 U+0030 U+6708	1 0 月 U+0031 U+0030 U+6708

## 7.4 追加キャラクターとサロゲート

Unicode の基本多言語面 (BMP) の外にある追加キャラクター、すなわち Unicode 値が **U+FFFF** を超えるキャラクターは、1 つの UTF-16 値では表すことができず、サロゲートペアと呼ばれる 2 つの UTF-16 値を必要とします。追加キャラクターの例としては、**U+1DXXX** にある数学記号・音楽記号や、**U+20000** から始まる日中韓の拡張キャラクターが挙げられます。

TET は追加キャラクターを解釈・保持し、それに対応する UTF-32 値を、ネイティブな Unicode 文字列が UTF-16 にしか対応していない言語バインディングにおいても利用可能にしています。1 番目のサロゲート値に対して `get_char_info()` が返す `uv` フィールドは、対応する UTF-32 値を持ちます。これを利用すれば、UTF-32 に対応していない UTF-16 環境で作業をしている場合でも追加キャラクターの UTF-32 値を直接利用可能です。

1 番目の (高位) サロゲートと 2 番目の (低位) サロゲートは保持されます。`get_text()` が返す文字列は 2 個の UTF-16 値を内容として持ちます。

## 7.5 グリフに対する Unicode マッピング

PDF 内のテキストは、さまざまなフォントやエンコーディング方式によって表されている可能性があるのに対し、TET は、PDF 内の元のテキスト表現にかかわらず、グリフから抽象情報を抽出し、すべてのテキストを Unicode キャラクタへ正規化します。PDF 内で見つかった情報を、おのおの対応する Unicode 値へ変換することを、**Unicode マッピング**といい、これはテキストの意味を理解する（テキストの視覚表現を画面上や紙に展開するのではなく）ために必須です。正しい Unicode マッピングを与えるために、TET は、PDF 文書内で見つかるさまざまなデータ構造を照会したり、埋め込まれた、あるいは外部のフォントファイルを照会したり、内蔵の、あるいはユーザが与えたテーブルを照会したりします。さらに TET は、非標準なグリフ名に対して Unicode マッピングを決定するいくつかの方式も適用します。

あらゆる方策を尽くしてもなお、一部のテキストが Unicode へマップできない PDF 文書もあります。このような場合に対処するため、TET は、問題のある PDF ファイルに対する Unicode マッピングを制御するために利用できるさまざまな設定機能を提供しています。

**マップ不能グリフ** PDF 内のテキストが Unicode へうまくマップできない場合、その原因は何種類かあります。たとえば、Type 1 フォントが未知のグリフ名を含んでいることが原因かもしれません。あるいは、TrueType・OpenType・CID のいずれかのフォントの場合には、グリフ ID で指定されていて、フォントまたは PDF の中に Unicode 値が含まれていないことが原因かもしれません。TET はそのようなマップなしキャラクタに対して、私用領域内のコード点を割り当てます。この PUA 値は **fold** オプションで除去または置き換えが可能です。デフォルトでは PUA キャラクタは U+FFFD へ、すなわち Unicode 未知キャラクタへマップされます。このキャラクタに対して自分のコードは対処するべきです。Unicode マッピングの問題を意に介しない場合であれば、単に U+FFFD を無視してもよいですし、あるいは下記の文書オプションを用いてこれを除去することもできます：

```
fold={[:Private_Use:] remove} }
```

マップ不能グリフかどうかを調べるには、`get_char_info()` が返す **unknown** フィールドを利用します。

**Unicode マッピング制御の概要** TET には、実際には Unicode 値を持たない PDF 文書から、それでも何とかしてテキストをうまく抽出できるよう処理するための、さまざまな代替機能が備わっています。しかしながらそれでもなお、PDF やフォントデータの構造内で十分な情報が得られないためにテキストを抽出することができない文書というものは存在します。TET は、Unicode マッピング情報を追加供給するために活用することのできるさまざまな設定機能を持っています。この節ではそうした機能について解説します。

`open_document()` で **glyphmapping** オプションを用いると (171 ページの 10.6 「文書関数」を参照)、グリフに対する Unicode マッピングを何通りかの方式で制御することができます。利用可能な方式の概要を以下に列挙します (組み合わせで利用することも可能)。こうした制御は、フォントごとに適用することもできますし、1 つの文書内のすべてのフォントに対してグローバルに適用することもできます：

- ▶ **forceencoding** サブオプションを用いると、PDF における定義済みのエンコーディング **WinAnsiEncoding**・**MacRomanEncoding** が用いられるたびに、すべて別のエンコーディングで処理させることができます。
- ▶ **codelist**・**tounicodecmap** サブオプションを用いると、Unicode 値のリストをシンプルテキスト形式 (**codelist** リソース) で与えることができます。

- ▶ *glyphlist* サブオプションを用いると、非標準グリフ名を Unicode 値にマップしたリストを与えることができます。
- ▶ *glyphrule* サブオプションを用いると、グリフ名の数値から Unicode 値をアルゴリズム的に導き出すための規則を定義することができます。 *encodinghint* オプションを用いると、内部規則を制御することができます。
- ▶ 定義済みのエンコーディングはすでに何ダースもありますが、それに加えてカスタムのエンコーディングを定義することも可能で、これは *encodinghint* オプションで、または *glyphrule* オプションの *encoding* サブオプションで利用することができます。
- ▶ PDF が十分な情報を与えずフォントが PDF に埋め込まれてもいないときに外部のフォントから Unicode マッピング情報が得られるよう設定しておくことも可能です。

**PDFlib FontReporter Plugin<sup>1</sup> で PDF 文書を分析** 正しい Unicode マッピングテーブルを作りたいとき、それに必要な情報を得るには、問題を含むその PDF 文書を分析しなければなりません。

PDFlib GmbH はこうした場面で役立つ TET の無償派生製品 PDFlib FontReporter を提供しています。これは、フォント・エンコーディング・グリフの情報を簡単に収集できる Adobe Acrobat プラグインです。このプラグインは、実際の各グリフに以下の情報を付した詳しいフォントレポートを作成します。

- ▶ その対応するコード: 16 進の 1 桁目が左端の列に示され、16 進の 2 桁目が上端の行に示されます。CID フォントの場合はヘッダに印字されるオフセットを加えると、グリフに対応するコードが得られます。
- ▶ そのグリフ名: もしあれば。
- ▶ そのグリフに対応する (1 つないし複数の) Unicode 値 (Acrobat がそれを決定できる場合)。

こうしたさまざまな情報は、TET のグリフマッピング制御に対して重要な役割を演じます。あるフォントレポートの一例から 2 つのページを抜き出して図 7.2 に示します。FontReporter Plugin によって作成されたフォントレポートを利用すれば、PDF フォントを分析し、ひいては TET がテキストを正しく抽出できるようにマッピングテーブルを作ることが可能です。TET のテキスト抽出を制御するために Unicode マッピングテーブルやグリフ名ヒューリスティックスを書こうと思うならば、それに対応するフォントレポートに目を通しておくことを強く推奨します。

1. PDFlib FontReporter Plugin は [www.pdflib.com/products/fontreporter](http://www.pdflib.com/products/fontreporter) から無償ダウンロード可能です。

PDFlib FontReporter																PDFlib FontReporter															
PIAPOC+ThesisAntiqua-Normal, Type1, Embedded, Subset Encoding: custom, symbol bit, ToUnicode table, CharSet table																ShinGo-Medium, CIDFontType0 Encoding: Identity-H Character collection: Adobe-Japan1-2 CID x0300 (50 glyphs)															
x0	x1	x2	x3	x4	x5	x6	x7	x8	x9	xA	xB	xC	xD	xE	xF	x0	x1	x2	x3	x4	x5	x6	x7	x8	x9	xA	xB	xC	xD	xE	xF
0x																															
1x																															
2x										(	)																				
3x																															
4x	i	2	3		5	6	7	8																							
5x	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O																
6x	p	R	S	T	U	V	W																								
7x	a	b	c	d	e	f	g	h	i		k	l	m	n	o																
8x	p	q	r	s	t	u	v	w	x	y	z																				
9x																															
Ax																															
Bx																															
Cx																															
Dx																															
Ex																															
Fx																															

図 7.2 Adobe Acrobat 用 PDFlib FontReporter Plugin が作成するフォントレポートの例

**優先規則** TET はグリフマッピング制御を以下の順番で適用します：

- ▶ codelist・ToUnicode CMap リソースがまず照会されます。
- ▶ フォントが内部 ToUnicode CMap を持つならそれが次に考慮されます。
- ▶ グリフ名について TET は、外部か内部のグリフ名マッピング規則を、どちらかフォントとグリフ名の一致するものが見つかれば適用します。
- ▶ 最後の手段として、ユーザが与えたグリフリストが適用されます。

**コードリストリソースはあらゆる種類のフォントに利用可能** コードリストはグリフリストに似ており、ただ各コードのグリフ名でなく Unicode 値を指定する点が違います。同じ鋳造場で作られた複数のフォントならば、互いに同一のコード割り当てを使用していることもあるかもしれませんが、一般にはコード（グリフ ID ともいう）というのはフォントごとに異なるものです。その結果、フォントごとに別々のコードリストが必要になります。コードリストはテキストファイルであり、各行ごとに、1つのコードに対する Unicode マッピングを以下の規則に従って記述しています：

- ▶ パーセント記号「%」の後のテキストは無視されます。これは注釈に使えます。
- ▶ 1列目にはグリフのコードを10進か16進記法で書きます。これは1バイトフォントの場合は0～255の範囲、CIDフォントの場合は0～65535の範囲内の値でなければなりません。



図 7.3  
あるロゴタイプのフォントに対するフォントレポートを見たところ、このフォントは誤った Unicode マッピングを持っていることがわかった。これはカスタムのコードリストで直すことが可能。

- ▶ その後につづけてその行には、そのコードに対する Unicode コード点を 7 つまで書くことができます。値は 10 進か（前に *x* か *ox* をつけて）16 進記法で与えることができます。UTF-32 に対応していますので、サロゲートペアも使えます。

コードリストはファイル名の拡張子として *.cl* を用いるきまりになっています。コードリストは *codelist* リソースを用いて設定を行うことができます。コードリストリソースが何も明示的に指定されていない場合、TET は *<mycodelist>.gl*（ここで *<mycodelist>* はリソース名）という名前のファイルを *searchpath* ヒエラルキーの中で探します（詳しくは、69 ページの 5.2 「リソース設定とファイル検索」を参照）。つまりいいかえれば、リソース名とファイル名（から拡張子 *.cl* を除いたもの）が同一のときはリソースを設定する必要はないということで、なぜならその場合 TET は下記の呼び出しと等価な動作を暗黙に行うからです（ここで *name* は任意のリソース名）：

```
set_option("codelist {name name.cl}");
```

次の例はコードリストの使用例です。ここでは、図 7.3 に示すようにロゴタイプのグリフが誤ってマップされている場合を想定します。このフォントでは、1 つのグリフが実際には複数のキャラクタを表しており、キャラクタをすべて並べると会社のロゴタイプが形成されるようになっているのですが、各グリフが誤ってキャラクタ *a・b・c・d・e* にマップされてしまっているのです。これを直すには以下のコードリストを作ればよいでしょう：

% GlobeLogosOne フォントの各コードに対する Unicode マッピング

```
x61      x0054 x0068 x0065 x0020      % The
x62      x0042 x006F
x63      x0073 x0074 x006F x006E x0020 % ston
x64      x0047 x006C x006F
x65      x0062 x0065      % be
```

そして *open\_document()* で下記のオプションを用いてコードリストを与えます（このコードリストは *GlobeLogosOne.cl* というファイルにあり、検索パスで見つかるようにしてあるとして）：

```
glyphmapping {{fontname=GlobeLogosOne codelist=GlobeLogosOne}}
```

**ToUnicode CMap リソースはあらゆる種類のフォントに利用可能** PDF は ToUnicode CMap というデータ構造に対応しています。これを用いると、フォントの各グリフに対する Unicode 値を与えることができます。このデータ構造が PDF ファイル内に存在する場合、TET はそれを利用します。あるいは、ToUnicode CMap を外部ファイルとして与えることも可能です。これは PDF 内の ToUnicode CMap が不完全な場合や、誤った内容を含んでいるときや、存在しない場合に有用です。ToUnicode CMap はコードリストよりも優先されます。ただ、コードリストのほうが ToUnicode CMap よりも形式が簡単なため、好んで用いられます。

CMap では、ファイル名に拡張子をつけないきまりになっています。ToUnicode CMap は `cmap` リソースを用いて設定を行うことができます (69 ページの 5.2 「リソース設定とファイル検索」を参照)。`cmap` リソースの内容は、標準的な CMap の文法に従う必要があります。<sup>1</sup> ToUnicode CMap を `Warnock` ファミリのすべてのフォントに適用するには、`open_document()` で下記のオプションを用います：

```
glyphmapping {{fontname=Warnock* tounicodecmap=warnock}}
```

**グリフリストリソースは 1 バイトフォントに利用可能** グリフリスト (グリフ名リストの略) を用いると、非標準のグリフ名に対してカスタムの Unicode 値を与えたり、あるいは標準のグリフ名に対する既存の値を無視させて別の値を使わせたりすることができます。グリフリストはテキストファイルであり、以下の規則に従って各行ごとに、1 つのグリフ名に対する 1 つの Unicode マッピングを記述します：

- ▶ パーセント記号「%」の後のテキストは無視されます。これは注釈に使えます。
- ▶ 1 列目にはグリフ名を書きます。フォント内で用いられている任意のグリフ名を書くことができます (すなわち、標準のグリフ名の Unicode 値を無視させて別の値を使わせることも可能)。グリフ名の中にパーセント記号を入れたい場合は `1%` とシーケンスで書く必要があります (パーセント記号は注釈を開始させる役割を持つので)。
- ▶ 1 つのグリフ名に対しては 1 つのマッピングだけが許されます。同一のグリフ名に対して複数のマッピングがある場合はエラーとして扱われます。
- ▶ その後につづけてその行には、そのグリフ名に対する Unicode コード点を 7 つまで書くことができます。値は 10 進記法か (前に `x` か `ox` をつけて) 16 進記法で与えることができます。UTF-32 に対応していますので、サロゲートペアも使えます。
- ▶ グリフ名の中に印字不能キャラクタを入れたい場合はテキストファイル用のエスケープシーケンスを用います (69 ページの 5.2 「リソース設定とファイル検索」を参照)。

グリフリストはファイル名の拡張子として `.gl` を用いるきまりになっています。グリフリストは `glyphlist` リソースを用いて設定を行うことができます。グリフリストリソースが何も明示的に指定されていない場合、TET は `<myglyphlist>.gl` (ここで `<myglyphlist>` はリソース名) という名前のファイルを `searchpath` ヒエラルキーの中で探します (詳しくは、69 ページの 5.2 「リソース設定とファイル検索」を参照)。つまりいいかえれば、リソース名とファイル名 (から拡張子 `.gl` を除いたもの) とが同一のときはリソースを設定する必要はないということで、なぜならその場合 TET は下記の呼び出しと等価な動作を暗黙に行うからです (ここで `name` は任意のリソース名)：

```
set_option("glyphlist {name name.gl}");
```

グリフマッピングの優先規則によって、フォントが ToUnicode CMap を含んでいる場合にはグリフリストは照会されません。以下の例はグリフリストの使用例です：

% TeX 文書で用いられるグリフ名に対する Unicode 値

```
precedesequal 0x227C
similarequal  0x2243
negationslash 0x2044
union         0x222A
prime        0x2032
```

1. [partners.adobe.com/public/developer/en/acrobat/5411.ToUnicode.pdf](http://partners.adobe.com/public/developer/en/acrobat/5411.ToUnicode.pdf) を参照

CMSY で始まるすべてのフォント名に対してグリフリストを適用したいときは、`open_document()` で下記のオプションを用います：

```
glyphmapping {{fontname=CMSY* glyphlist=tarski}}
```

**1 バイトフォントの数値グリフ名を規則で解釈させる** PDF 文書内のグリフ名はときに、何らかの定義済みのリストから採って来られないで、アルゴリズム的に生成されていることがあります。これはその PDF を生成したアプリケーションの「機能」かもしれませんが、あるいはプリンタドライバがフォントを別の形式に変換したことによるものかもしれませんが、その過程で元のグリフ名が失われ、*Go0, Go1, Go2, …* のような規則的な名前に置き換えられていることがあるのです。TET は、普及しているさまざまなアプリケーションやドライバが生成する数値グリフ名を処理するためのグリフ名規則を内蔵しています。ただ、グリフ名が同じであってもそれが別々のエンコーディングに対して生成される場合もあるので、`open_document()` で `encodinghint` オプションを与えて、文書内に現れる規則的なグリフ名が従うべきエンコーディングを指定することもできるようになっています。たとえば文書にロシア語のテキストが入っていることがわかっているのに、PDF 内の情報不足でうまくそれが抽出できない場合は、オプション `encodinghint=cp1250` を与えてキリル文字コードページを指定することができます。

こうした数値グリフ名を解釈する内蔵の規則のほかに、自分でカスタムの規則を定義することもできます。それには `open_document()` で `glyphmapping` オプションに `fontname·glyphrule` サブオプションを用います。その際には以下の情報を与える必要があります：

- ▶ その規則を適用するべきフォントの完全名か短縮名 (`fontname` オプション)
- ▶ グリフ名の接頭辞、すなわち数値部の前につくキャラクタ列 (`prefix` サブオプション)
- ▶ 数値を解釈する際の基数 = 10 進か 16 進か (`base` サブオプション)
- ▶ できた数値コードを解釈するべきエンコーディング (`encoding` サブオプション)

たとえばフォント *T1, T2, T3, …* 内のグリフ名が *co0, co1, co2, …, cFF* で、各グリフ名が WinAnsi キャラクタの 16 進位置 (*oo, …, FF*) にそれぞれ対応していることがわかった (PDFlib FontReporter を利用するなどして) 場合には、`open_document()` で下記のオプションを用います：

```
glyphmapping {{fontname=T* glyphrule={prefix=c base=hex encoding=winansi} }}
```

**外部フォントファイルとシステムフォント** PDF が Unicode マッピングのための十分な情報を持たず、しかもフォントが埋め込まれていないときは、TET が Unicode マッピングの導出に利用できるフォントデータが追加されるよう設定することもできます。フォントデータはディスク上の TrueType か OpenType のフォントファイルから来るようにすることもでき、その場合は `fontoutline` リソースカテゴリを用いてそのように設定することが可能です。あるいは Mac や Windows システム上では、TET はホストオペレーティングシステム上にインストールされたフォントを利用することもできます。こうしたホストフォントを利用させないようにするには `open_document()` で `fontoutline` オプションを用います。

ディスクファイルを *WarnockPro* フォントのために設定するには下記の呼び出しを行います：

```
TET_set_option("fontoutline {WarnockPro WarnockPro.otf}");
```

外部フォントファイルの設定については、詳しくは、69 ページの 5.2 「リソース設定とファイル検索」を参照してください。



# 8 画像抽出

## 8.1 画像抽出の基本

**さまざまな画像形式** TET は PDF ページ群からラスタ画像群を抽出し、その抽出した画像を、以下のいずれかの形式で格納します：

- ▶ TIFF (*.tif*) 画像は多くの場合に生成されます。TET が生成する多くの TIFF 画像は、多くの TIFF ビューア・コンシューマで利用できます。ただし、いくつかの高度な TIFF 機能には、すべての画像ビューアが対応しているわけではありません。Windows XP の画像ビューアは、TIFF で広く使われている Flate 圧縮に対応していないことに留意してください。当社では Adobe Photoshop を、TIFF 画像の有効性に関するベンチマークとしています。
- ▶ JPEG (*.jpg*) は、PDF 内ですでに JPEG アルゴリズム (*DCTDecode* フィルタ) で圧縮されている画像について生成されます。ただし、DCT 圧縮された画像を TIFF として抽出する必要がある場合もいくつかあります。PDF 色処理のすべての側面が JPEG で表現できるわけではないからです。
- ▶ JPEG 2000 (*.jpx*) は、PDF 内ですでに JPEG 2000 アルゴリズム (*JPXDecode* フィルタ) を用いて圧縮されている画像について生成されます。

**画像をディスクまたはメモリへ抽出** TET API は、PDF 文書から抽出した画像を 2 種類の方式で受け渡すことができます：

- ▶ *write\_image\_file()* API 関数は、画像ファイルをディスク上に生成します。この画像ファイルのベースファイル名は *filename* オプションで指定する必要があります。TET は画像種別に応じて適切な接尾辞を自動的に付加します。
- ▶ *get\_image\_data()* API 関数は、画像データをメモリ内で受け渡します。これは、画像データを他の処理構成要素へ、ディスクファイルを扱う必要なしに受け渡したいときに便利です。

詳細は、自分の画像抽出上の要請に依存します (126 ページの 8.4 「ページベースとリソーススペースの画像ループ」を参照)。いずれの場合にも、抽出画像の種別を知ることが可能です (次項参照)。

**抽出画像のファイル種別を知る** 画像ファイル種別は、TETML 内の *Image/@extractedAs* 属性で報告されています。API レベルでは、以下の慣用表現を用いて抽出画像の種別を知ることができます。

```
int imageType = tet.write_image_file(doc, tet.imageid, "typeonly");
```

```
/* 画像種別を表す数値を種別名へマップ */  
String imageFormat;  
switch (imageType) {  
case 10:  
    imageFormat = "TIFF";  
    break;  
  
case 20:  
    imageFormat = "JPEG";  
    break;
```

```

case 30:
    imageFormat = "JPEG2000";
    break;

case 40:
    imageFormat = "RAW";
    break;

default:
    System.err.println("write_image_file() が未知の値を返しました "
        + imageType + ", 画像をスキップします, エラー: "
        + tet.get_errmsg());
}

```

**画像の XMP メタデータ** PDF では、XMP 形式を用いて、文書全体または文書の一部に対してメタデータを付与しています。XMP とその用途に関して詳しくは右記を参照してください：[www.pdflib.com/knowledge-base/xmp-metadata/](http://www.pdflib.com/knowledge-base/xmp-metadata/)

画像オブジェクトには、PDF 文書の中で、XMP メタデータが関連づけられていることがあります。XMP メタデータが存在しているときは、TET はデフォルトでは、出力形式 JPEG・TIFF の抽出画像について、その中にそれを埋め込みます。この動作は、`write_image_file()・get_image_data()` の `keepxmp` オプションで制御することができます。このオプションが `false` に設定されているときは、TET は、画像出力ファイルを生成する際に画像メタデータを無視します。

pCOS クックブック内の `image_metadata` トピックでは、画像ファイルを一切生成せずに pCOS インタフェースを用いて画像メタデータを直接抽出する方法を示しています。

## 8.2 画像の連結とフィルタリング

**画像連結** 画像を、PDF 文書内で表現されているとおりに抽出することが望ましくない場合もあります：一見 1 個の画像であるものが、実はたくさんの小画像を並べたものである場合が多くあります。このように画像を断片化する理由としてよくあるのは以下のとおりです：

- ▶ アプリケーションやドライバのなかには、マルチストリップ TIFF 画像を断片化 PDF 画像へ変換するものがあります。ストリップの数は数ダースから何百個にも及ぶ場合があります。
- ▶ スキャンソフトウェアのなかには、スキャンしたページを小さな断片（ストリップまたはタイル）に分割するものがあります。断片の数は通常、数ダースを超えることはありません。
- ▶ アプリケーションのなかには、印刷出力や PDF 出力を生成する際に画像を小さな断片に分割するものがあります。極端な例としては、とりわけ Microsoft Office アプリケーション群によって作成された文書では、1つのページが数千もの小画像断片を含んでいる場合もあります。

TET の画像連結エンジンは、このような状況を検出して、画像断片を再結合して 1 つの大きな有用な画像にします。画像群が連結候補と見なされるためには、いくつかの条件が満たされる必要があります：

- ▶ 画像断片群が水平または垂直に並んでおり（任意の角度でではなく）、サブ画像群の矩形グリッドを形成していること。
- ▶ 構成要素あたりのビット数が同じであること。
- ▶ 色空間が同じ、ないし互換であること。
- ▶ 色空間と圧縮方式（特に JPEG 2000 圧縮）の組み合わせによっては、画像連結ができません。

連結候補群が、結合して大きな画像にできる場合には、それらは連結されます。連結された画像は、`images[/mergetype` pCOS 擬似オブジェクトによってそのように識別できます。それは、連結された画像については値 1（擬似）を持ち、連結処理によって消費された画像については 2（消費済）を持ちます。消費済画像は一般に、受け取ったアプリケーション側では無視するべきです。

画像連結を完全に無効化するには下記のページオプションを用います：



図 8.1  
この画像はたくさんの小さなストリップから成っていますが、TET はこれを 1 個の再利用可能な画像として抽出します。

```
imageanalysis={merge={disable}}
```

**画像はいつ連結されるか** ページ上の画像の分析と連結は、おのこの `open_page()` への呼び出しによって引き起こされます。これは、以下の重要な結果につながります：

- ▶ pCOS `images[]` 配列内の項目数は、すなわち `length:images` 擬似オブジェクトの数は、増えていく可能性があります：ページが順次処理されていくにつれて、画像連結によってできた擬似画像が配列に加わっていくからです。ですので、すべての連結済画像を抽出するためには、`length:images` を取得して画像データを抽出する前に、文書内のすべてのページを開く必要があります。擬似（連結済）画像は、`images[]/mergetype` 擬似オブジェクトで、それに対応するフラグ `artificial`（数値 1）でマークされています。
- ▶ 一方、`images[]` 配列内の要素のなかには、連結済画像の一部分としてのみ用いられるものがあります。しかし、消費済の項目は、`images[]` 配列から除去されるのではなく、`images[]/mergetype` 擬似オブジェクトで、それに対応するフラグ `consumed`（数値 2）でマークされています。

**文書内に画像がいくつあるか** なんと、この簡単な質問には簡単な答えがありません。答えは以下の決定によって変わります：

- ▶ 画像リソースを数えたいか、それとも配置された画像を数えたいのか。
- ▶ 連結済画像の一部分としてのみ用いられている、単独では配置されていない画像を数に入れたいか。

TET と pCOS 擬似オブジェクトを用いれば、これらすべての種類の画像数の答えを知ることができます。TET クックブック内の `image_count` トピックでは、画像のさまざまな数え方の可能性を演示しています。それは以下のような出力を生成します：

```
No of raw image resources before merging: 82
No of placed images: 12
No of images after merging (all types): 83
  normal images: 1
  artificial (merged) images: 1
  consumed images: 81
No of relevant (normal or artificial) image resources: 2
```

**小画像フィルタリング** TET は、ページ上に微小画像がたくさんあるときは、それらを無視します。多くの小画像は、画像連結処理によって結合されて 1 つの大画像にされることが多いですので、小画像除去は画像除去の後に行われます。かつそれらは、`imageanalysis` ページオプションの `smallimages` サブオプションの `maxarea`・`maxcount` サブオプションで指定できる大きさと数に関する条件を満たす必要があります。小画像除去を完全に無効化するには、下記のページオプションを用います：

```
imageanalysis={smallimages={disable}}
```

## 8.3 配置画像と画像リソース

TET では、配置画像と画像リソースを区別します：

- ▶ **配置画像**は、ページ上の画像に対応します。配置画像は視覚特性を持ちます：特定の位置に配置されており、寸法（ポイント・ミリメートルなどの絶対単位で測られる）を持っています。多くの場合その画像はページ上で可視になっていますが、場合によっては、ページ上の他のオブジェクトによって隠されていたり、可視ページ領域の外に配置されていたり、全体的ないし部分的にクリップされていたりして不可視になっています。配置画像は、TETML 内で *PlacedImage* 要素で表されています。
- ▶ **画像リソース**は、実際のピクセルデータ・色空間・要素数・要素あたりビット数などを表現するリソースです。配置画像と違って、画像リソースは明示的な視覚情報を持ちません。ただし、幅と高さの特性（ピクセル単位で測られる）は持っています。画像リソースはそれぞれ一意な ID を持っており、それを用いてそのピクセルデータを抽出することができます。画像リソースは、TETML 内で *Image* 要素で表されています。

1 個の画像リソースは、その文書内の任意の数の配置画像のもととして使われることがあります。通常、画像リソースはそれぞれ 1 回だけ配置されますが、同一ページ上に、または複数ページ上に繰り返し配置されることもあります。たとえば、文書内の各ページのヘッダに繰り返し用いられる企業ロゴの画像を考えてみましょう。ページ上のロゴはそれぞれ、1 個の配置画像から成っていますが、最適化された PDF においては、それらの配置画像がすべて同一の画像リソースに紐付けられていることがあります。一方、最適化されていない PDF においては、配置ロゴはそれぞれ、同一の画像リソースの自分用の複製を元に行っていることがあります。これは見た目は同じ結果になりますが、PDF 文書のファイルサイズは大きくなります。最適化されていない PDF 文書は、どのページからも参照すらされていない画像リソース（すなわち未使用リソース）を含んでいることもあります。

表 8.1 で、配置画像と画像リソースのさまざまな特性を比較しています。

表 8.1 配置画像と画像リソースの比較

特性	配置画像	画像リソース
TETML 要素	PlacedImage	Image
画像連結による影響	あり	あり
ページとの紐付け	あり	-
ピクセル単位の幅と高さ	あり	あり
ポイント単位の幅と高さ	あり	-
ページ上の位置	あり	-
視覚上の出現回数	1	0、1 ないしそれ以上
一意な ID	なし : <code>get_image_info( )</code> が返す <code>imageid</code> メンバと、TETML 内の <code>PlacedImage/@image</code> 属性は、背後の画像リソースを特定するにすぎません	はい : <code>get_image_info( )</code> が返す <code>imageid</code> メンバと、TETML 内の <code>Image/@id</code> 属性
TET コマンドラインツールにおけるファイル命名規則	<ファイル名>_p< ページ番号 >_< 画像番号 >. [tif jpg jpx]	<ファイル名>_I< 画像 ID>. [tif jpg jpx]

## 8.4 ページベースとリソースベースの画像ループ

配置画像と画像リソースの違いは、すなわち、画像抽出について 2 種類の根本的に異なるアプローチをもたらします：ページベースとリソースベースの画像抽出ループです。どちらの方式も、画像をディスクファイルかメモリへ抽出するために用いることができます。

**ページベースの画像抽出ループ** この場合には、アプリケーションは正確なページレイアウトと配置画像に関心があり、画像データの重複はいとわなないこととなります。ページベースのループで画像を抽出すると、配置画像ごとに 1 つずつ画像ファイルが生成されますので、複数の抽出配置画像に同一の画像データが含まれることがあります。アプリケーション側で画像 ID の重複をチェックすることで画像の重複を避けることもできます。しかし、一意な画像リソースを抽出したいなら、リソースベースの画像抽出ループを用いるほうが簡単です（後述）。

ページベースの画像抽出ループは、TET コマンドラインツールではオプション `--imageloop page` で有効にすることができます。API レベルでのページベースの画像抽出のためのコードは、TET クックブックの `images_per_page · images_in_memory` トピック内で演示しています。`images_per_page` クックブックトピックで、ページ上の画像の座標を取得する方法も示しています。

ページベースの画像抽出ループの詳細（上述のサンプルコードを参照してください）：`get_image_info()` が、配置画像に関する視覚情報と、その背景画像データの pCOS 画像 ID (`imageid` フィールドで) を取得します。この ID を用いて、`pcos_get_number()` で画像の色空間や、幅・高さをピクセル単位で表したものといたさらなる詳細を取得したり、`write_image_file()` または `get_image_data()` でその画像のピクセルデータ本体を取得したりすることができます。同一の画像が 1 つないし複数のページ上で複数回参照されている場合、その対応する ID は同一になります。

**リソースベースの画像抽出ループ** この場合には、アプリケーションは文書内の画像リソースに関心があり、どの画像がどのページで用いられているかはいとわなないこととなります。複数回（1 つないし複数のページ上に）配置されている画像リソースが 1 回だけ抽出されます。その反面、どのページにも全く配置されていない画像も抽出されます。

リソースベースの画像抽出ループは、TET コマンドラインツールではオプション `--imageloop resource` で有効にすることができます。API レベルでのリソースベースの画像抽出のためのコードは、`image_resources` ミニサンプル・クックブックトピック内で演示しています。pCOS パスリファレンスには、pCOS インタフェースに関してさらに詳しい情報があります。

リソースベースの画像抽出ループの詳細（上述のサンプルコードを参照してください）：画像リソースを抽出する前に、画像連結が必ず有効になるように、すべてのページが開かれます。画像連結が関係ない場合はこのステップはスキップできます。画像を抽出するためには、その対応する画像 ID が必要です。コードは、0 から最大画像 ID までのすべての値を評価します。この最大画像 ID は、`pcos_get_number()` を用いて、pCOS `length:images` の値として取得されます。連結済画像の消費済部分（マルチストリップ画像のストリップなど）をスキップするために、`mergetype` pCOS 擬似オブジェクトを用いて各画像リソースの種別が調べられます。これによって、画像連結処理によって消費された画像部分をスキップすることができます（最終的な連結済画像にのみ関心があるからです）。ひとたび画像 ID がわかれば、関数 `write_image_file()` または `get_image_data()` を呼び出せば、前者なら画像データをディスクファイルへ書き出すことができ、後者ならピクセルデータをメモリ内で受け渡すことができます。

## 8.5 配置画像の視覚情報

`get_image_info()` を用いると、配置画像の視覚情報を取得することができます。各画像について、`image_info` 構造内で以下の値が得られます (図 8.2 参照) :

- ▶ `x・y` フィールドは、画像参照点の座標です。参照点は通常、画像の左下隅です。ただし、ページ上の座標系変換によって参照点が変わることもあります。たとえば、画像が水平反転されていれば、参照点は画像の左上隅になるでしょう。`y` の値は `topdown` ページオプションに依存します。
- ▶ `width・height` フィールドは、ページ上の配置画像の物理的寸法に対応します。これらはポイント (すなわち 1/72 インチ) 単位で与えられます。
- ▶ 角度 `alpha` は、ピクセル行の向きを記述します。この角度は範囲  $-180^\circ < \alpha < +180^\circ$  をとります。角度 `alpha` は、画像をその参照点を中心に回転させます。正立した画像については `alpha` は  $0^\circ$  になります。`alpha` と `beta` の値は `topdown` ページオプションに依存します。
- ▶ 角度 `beta` は、ピクセル列の向きを記述します。これは `alpha` の垂線に平行です。この角度は範囲  $-180^\circ < \beta < +180^\circ$ 、ただし  $\pm 90^\circ$  以外の値をとります。角度 `beta` は画像を斜形化し、`beta=180^\circ` は画像を `x` 軸で反転します。正立した画像については、`beta` は範囲  $-90^\circ < \beta < +90^\circ$  をとります。`abs(beta) > 90^\circ` なら、その画像はベースラインで反転されています。
- ▶ `imageid` フィールドは、画像の pCOS ID を内容として持ちます。これを用いると、pCOS 関数群で詳しい画像情報を取得したり、`write_image_file()` または `get_image_data()` で画像ピクセルデータ本体を取得したりすることができます。

画像変形の結果として、抽出画像の向きは誤りに見える場合があります。なぜなら、抽出される画像データは、PDF 内の画像オブジェクトに基づいているからです。PDF ページ上で配置画像に対して適用されている回転や反転といった変形はいずれも、ピクセルデータには適用されず、元のピクセルデータが抽出されます。

**画像解像度** 画像解像度を dpi (dots per inch) 単位で算出するには、ピクセル単位の画像幅をポイント単位の画像幅で割り、それに 72 をかける必要があります :

```
while (tet.get_image_info(page) == 1) {
    String imagePath = "images[" + tet.imageid + "]";
    int width = (int) tet.pcos_get_number(doc, imagePath + "/Width");
    int height = (int) tet.pcos_get_number(doc, imagePath + "/Height");

    double xDpi = 72 * width / tet.width;
```

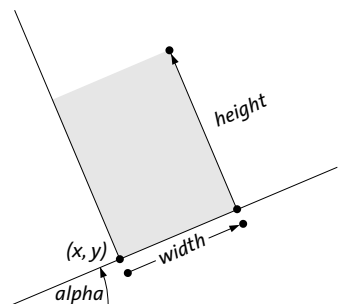


図 8.2  
画像の視覚情報

```
double yDpi = 72 * height / tet.height;
...
}
```

回転されたり斜形化されたりしている画像については、dpi 値は無意味かもしれないことに留意してください。画像の dpi 計算のための完全なコードは、TET クックブックの *determine\_image\_resolution* トピック内にあります。



## 8.6 制約と注意

**画像の色再現性** TETは画像を抽出する際に、画像品質を下げません：

- ▶ ラスタ画像がダウンサンプルされることはありません。
- ▶ 画像の色空間は、出力内で温存されます。CMYKからRGBへの変換や、同様の色変換をTETが適用することはありません。
- ▶ 色要素の数はつねに不変となります。たとえば、RGB画像は、灰色しか含んでいなくても、グレースケールへ変更されません。

**各種画像対応** 場合によっては、抽出された画像の色の見た目が、PDF ページ上での見た目と異なることがあります。画像の輪郭は温存されているながらも、色は以下のような理由で違って見える場合があります：

- ▶ 画像マスクが適用されている。
- ▶ 着色されたグレースケール画像は、その色なしで、グレースケール画像として抽出されます。
- ▶ DeviceN カラーは TIFF では対応していませんので、DeviceN 色空間を持つ画像は、 $N=1 \cdot 3 \cdot 4$  の場合にはそれぞれグレースケール・RGB・CMYK 画像として抽出されます。 $N>4$  のものについては、1 個ないし複数のアルファチャンネルを持つ CMYK TIFF 画像が生成されます。
- ▶ 分版色空間を持つ画像は、グレースケール画像として抽出されます。画像の着色に用いられていた特色は失われます。
- ▶ インデックス付き ICCBased 色空間を持つ画像：ICC プロファイルは無視されます。

**画像抽出の予期せぬ結果** 場合によっては、抽出された画像の輪郭が PDF ページと違って見えることがあります：

- ▶ 画像が水平に（上下逆）、または垂直に反転して見えることがあります。これは、PDF ページ上で何らかの変形が画像に適用されていても、TET はそれにかかわらず、その画像の元のピクセルデータを抽出していることによります。
- ▶ 画像マスクは無視されますので、マスク効果は抽出画像へは反映されません。

**非対応の画像種別** 以下の種別の PDF 画像は抽出できません。すなわち、`write_image_file()` はこれらの場合には -1 を返します：

- ▶ PDF インライン画像：これはまれにしか用いられない種類の PDF 画像であり、ときたま小さなラスタ画像に対して用いられています。
- ▶ JBIG2 圧縮を用いた画像
- ▶ インデックス付き Lab 色空間を用いた画像



# 9 TET マークアップ言語 (TETML)

## 9.1 TETML を生成

PDF 文書文書の内容をプログラミングインタフェース経由で提供することのほかに、TET では、これと同じ情報を表す XML 出力を生成することもできます。TET が生成するこの XML 出力を TET マークアップ言語 (TETML) と呼んでいます。TETML は、PDF ページ群のテキスト内容を含んでいるほか、テキスト位置・フォント・文字サイズなどの情報も含んでいることがあります。TET がページ上に表組のような構造を検出したときは、その表組は TETML 内で表・表行・セル要素の階層構造として表されます。なお、表組情報は TET プログラミングインタフェースでは得ることができず、TETML を通じてのみ得ることができます。TETML は画像と色空間に関する情報も含んでいます。

PDF 文書を TETML へ変換するには、TET コマンドラインツールか TET ライブラリのいずれかを用いることができます。いずれの場合にも、TETML 生成の詳細を制御するために利用できるさまざまなオプションがあります。

**TET コマンドラインツールで TETML を生成** TET コマンドラインツールを用いる場合は、`--tetml` オプションで TETML オプションを生成することができます。下記のコマンドは TETML 出力文書 `file.tetml` を生成します：

```
tet --tetml word file.pdf
```

さまざまなオプションを用いて、文書の一部分のページのみを変換したり、処理オプションを与えたりすることもできます。詳しくは、19 ページの 2.1 「コマンドラインオプション」を参照してください。

**TET ライブラリで TETML を生成** TET ライブラリでは、API 呼び出しの簡単な連鎖を用いて TETML 出力を生成することができます。`tetml` サンプルプログラムでは、プログラムの TETML を生成するための正統的な指示列を演示しています。このサンプルプログラムは、すべての対応言語バインディングで利用可能です。

TETML 出力は、ディスクファイル上にもメモリ内にも生成できます。生成された TETML ストリームは、現在利用されている多くのプログラミング言語が提供している XML 対応を用いて XML ツリーへパースすることができます。`tetml` サンプルプログラムでは、TETML ツリーを処理する様子も演示しています。

**TETML の中身** TETML 出力は UTF-8 (USS または MVS を用いた zSeries 上 :EBCDIC-UTF-8。 [www.unicode.org/reports/tr16](http://www.unicode.org/reports/tr16) を参照) で符号化されており、以下の情報を含んでいます (これらの項目のいくつかはオプションです)：

- ▶ 一般文書情報：暗号化ステータス・PDF 規格・タグ付き PDF など
- ▶ 文書情報フィールド・XMP メタデータ
- ▶ 各ページのテキスト内容 (単語または段落。行を含めることも可能)
- ▶ グリフのフォントと位置情報 (フォント名・サイズ・座標)
- ▶ グリフのレイアウト属性 (下 / 上付き・ドロップキャップ・影付き)
- ▶ ハイフネーション属性
- ▶ 構造情報。例：表組
- ▶ ページ上の配置画像に関する情報

- ▶ リソース情報、すなわちフォント・色空間・画像
- ▶ PDF 処理中に例外が発生した場合はエラーメッセージ

TETML 内では、さまざまな要素と属性がオプション的です。詳しくは、135 ページの 9.2 「TETML の詳細を制御」を参照してください。

**さまざまな TETML の例** 以下に、TETML 文書の最も重要な各部分を抜き出して示します：

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Created by the PDFlib Text Extraction Toolkit TET (www.pdflib.com) -->
<TET xmlns="http://www.pdflib.com/XML/TET3/TET-3.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.pdflib.com/XML/TET3/TET-3.0
  http://www.pdflib.com/XML/TET3/TET-3.0.xsd"
  version="4.1">
<Creation platform="Linux-x86_64" tetVersion="4.1dev" date="2012-01-27T11:16:43+01:00" />
<Document filename="FontReporter.pdf" pageCount="9" filesize="132437" linearized="true"
  pdfVersion="1.6">
<DocInfo>
<Author>PDFlib GmbH</Author>
<CreationDate>2010-07-06T22:51:50+00:00</CreationDate>
<Creator>FrameMaker 7.0</Creator>
<ModDate>2010-07-06T23:07:59+02:00</ModDate>
<Producer>Acrobat Distiller 9.3.3 (Windows)</Producer>
<Subject>PDFlib FontReporter</Subject>
<Title>PDFlib FontReporter Manual</Title>
</DocInfo>
<Metadata>
<x:xmpmeta xmlns:x="adobe:ns:meta/" x:xmptk="Adobe XMP Core 4.2.1-c043 52.372728, 2009/
01/18-15:08:04" >
  <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
    ...XMPメタデータ...
  </rdf:RDF>
</x:xmpmeta>
</Metadata>
<Options>tetml={} </Options>
<Pages>
<Page number="1" width="485" height="714">
<Options>tetml={} granularity=word </Options>
<Content granularity="word" dehyphenation="false" dropcap="false" font="false"
geometry="false" shadow="false" sub="false" sup="false">
<Para>
  <Word>
    <Text>FontReporter</Text>
    <Box llx="28.32" lly="613.53" urx="214.98" ury="643.53"/>
  </Word>
</Para>
<Para>
  <Word>
    <Text>Version</Text>
    <Box llx="28.32" lly="582.87" urx="100.24" ury="604.83"/>
  </Word>
<Word>
    <Text>1.4</Text>
    <Box llx="105.05" lly="582.87" urx="128.79" ury="604.83"/>
  </Word>
</Para>
</Pages>
</TET>
```

```

</Word>
</Para>
...さらなるページ内容...
</Content>
</Page>
...さらなるページ群...
<Resources>
<Fonts>
  <Font id="F0" name="PDFlibLogo-Regular" fullname="MMOHKN+PDFlibLogo-Regular"
    type="TrueType" embedded="true" ascender="1000" capheight="700" italicangle="0"
    descender="0" weight="400" xheight="500"/>
  <Font id="F1" name="ThesisAntiqua-Bold" fullname="MMOHKO+ThesisAntiqua-Bold"
    type="Type 1 CFF" embedded="true" ascender="741" capheight="679" italicangle="0"
    descender="-250" weight="606" xheight="505"/>
...さらなるフォント群...
</Fonts>
<Images>
  <Image id="I0" extractedAs=".tif" width="595" height="750" colorspace="CS3"
    bitsPerComponent="8"/>
  <Image id="I1" extractedAs=".tif" width="595" height="750" colorspace="CS3"
    bitsPerComponent="8"/>
</Images>
<ColorSpaces>
  <ColorSpace id="CS0" name="DeviceGray" components="1"/>
  <ColorSpace id="CS1" name="DeviceCMYK" components="4"/>
...さらなるカラースペース群...
</ColorSpaces>
</Resources>
</Pages>
</Document>
</TET>

```

選択する TETML モードに応じて、さらなる詳細を TETML 内で表すこともできます。TETML モードについて詳しくは、136 ページの「テキストモードを選択」で説明します。上記サンプルにグリフ詳細が加わった 1 つの変化形を以下に示します。**Glyph** 要素は、フォント・位置情報を含んでいます：

```

<Word>
  <Text>PDFlib</Text>
  <Box llx="111.48" lly="636.33" urx="161.14" ury="654.33">
    <Glyph font="F1" size="18" x="111.48" y="636.33" width="9.65">P</Glyph>
    <Glyph font="F1" size="18" x="121.12" y="636.33" width="11.88">D</Glyph>
    <Glyph font="F1" size="18" x="133.00" y="636.33" width="8.33">F</Glyph>
    <Glyph font="F1" size="18" x="141.33" y="636.33" width="4.88">l</Glyph>
    <Glyph font="F1" size="18" x="146.21" y="636.33" width="4.88">i</Glyph>
    <Glyph font="F1" size="18" x="151.08" y="636.33" width="10.06">b</Glyph>
  </Box>
</Word>
<Word>
  <Text>GmbH</Text>
  <Box llx="165.06" lly="636.33" urx="214.84" ury="654.33">
    <Glyph font="F1" size="18" x="165.06" y="636.33" width="12.06">G</Glyph>
    <Glyph font="F1" size="18" x="177.12" y="636.33" width="15.44">m</Glyph>
    <Glyph font="F1" size="18" x="192.56" y="636.33" width="10.06">b</Glyph>
    <Glyph font="F1" size="18" x="202.61" y="636.33" width="12.22">H</Glyph>
  </Box>
</Word>

```

```
</Box>
</Word>
<Word>
  <Text>Munchen</Text>
  <Box llx="218.75" lly="636.33" urx="292.23" ury="654.33">
    <Glyph font="F1" size="18" x="218.75" y="636.33" width="15.77">M</Glyph>
    <Glyph font="F1" size="18" x="234.52" y="636.33" width="10.19">u</Glyph>
    <Glyph font="F1" size="18" x="244.70" y="636.33" width="10.22">n</Glyph>
    <Glyph font="F1" size="18" x="254.92" y="636.33" width="7.52">c</Glyph>
    <Glyph font="F1" size="18" x="262.44" y="636.33" width="10.22">h</Glyph>
    <Glyph font="F1" size="18" x="272.66" y="636.33" width="9.34">e</Glyph>
    <Glyph font="F1" size="18" x="282.00" y="636.33" width="10.22">n</Glyph>
  </Box>
</Word>
```

## 9.2 TETML の詳細を制御

**さまざまな TETML テキストモード** TETML は、さまざまなテキストモードで生成することができます。各モードは、含むフォント・位置情報の量が異なり、また、テキストをより大きな単位（粒度）にまとめるやり方が異なっています。テキストモードは、各ページに対して個別に指定することが可能です。しかし多くの場合、TETML ファイルは全ページに対するデータを同一のモードで含みます。以下のテキストモードが利用可能です：

- ▶ **glyph** モードは、各グリフのテキスト・フォント・座標を含み、単語グループ化や構造情報を一切含まない低レベルな種類です。これはページ上の元のテキスト情報を表していますので、デバッグや分析の用途を想定しています。
- ▶ **word** モードでは、テキストは単語ごとにまとめられ、各単語の座標を持つ **Box** 要素が加わります。フォント情報は一切得られません。このモードは、単語ベースで動作するアプリケーションに適しています。約物キャラクタはデフォルトでは独立した単語として扱われますが、この動作はページオプションで変えることもできます（93 ページの「欧文テキストの単語境界検出」を参照）。テキストの行を **Line** 要素で特定させることも可能です。これは **tetml** ページオプションで制御されます。
- ▶ **wordplus** モードは、**word** モードと似ていますが、単語内の全グリフのフォント・座標の詳細が加わります。座標は、**topdown** ページオプションに従って、左下隅か左上隅からの相対位置で表されます。**wordplus** モードでは、フォントの使われ方を分析し、単語内でのフォントや文字サイズなどの変化を追うことが可能です。**wordplus** は、関連する TETML 要素をすべて含む唯一のテキストモードですので、あらゆる種類の処理作業に適しています。その反面、これは TETML 内に大量の情報を含みますので、最も大きな量の出力を生成します。
- ▶ **line** モードは、個別の **Line** 要素を構成するすべてのテキストを含みます。そのうえ、複数の行が 1 個の **Para** 要素内にまとめられることもあります。**line** モードは、ページ内容が行にまとめられていることがわかっている場合か、または受取先アプリケーションが行ベースのテキスト入力しか扱えない場合にのみ推奨します。
- ▶ **page** モードは、段落レベルから始まる構造情報を含みますが、フォント・座標の詳細は一切含みません。

表 9.1 に、各種テキストモード内に存在する TETML 要素を挙げます。

表 9.1 各種テキストモード内の TETML 要素一覧

テキストモード	構造	表組	テキスト位置	テキスト詳細
<b>glyph</b>	-	-	-	Glyph
<b>word</b>	Para・Word オプション： Line	Table・Row・Cell	Box	-
<b>wordplus</b>	Para・Word オプション： Line	Table・Row・Cell	Box	Glyph
<b>line</b>	Para・Line	-	-	-
<b>page</b>	Para	Table・Row・Cell	-	-

**テキストモードを選択** TET コマンドラインツール (19 ページの 2.1「コマンドラインオプション」を参照) では、望むページモードを、`--tetml` オプションに対する引数として指定することができます。下記のコマンドは、*wordplus* モードで TETML 出力を生成します。

```
tet --tetml wordplus file.pdf
```

TET ライブラリでは、テキストモードは直接指定することはできず、オプション群の組み合わせとして指定します：

- ▶ 最小要素内のテキストの量を、`process_page()` の *granularity* オプションで指定することができます。
- ▶ `granularity=glyph` または `word` の場合には、さらにグリフ詳細の量を指定することも可能です。グリフ情報の一部分が必要ないときは、`tetml` オプションの *glyphdetails* サブオプションでそれを省略させることもできます。

下記のページオプションリストは、すべてのグリフ詳細を含めた *wordplus* モードで TETML 出力を生成します：

```
granularity=word tetml={ glyphdetails={all} }
```

表 9.2 に、各種ページモードを作成するためのオプションをまとめます。

表 9.2 TET ライブラリで各種 TETML テキストモードを作成

テキストモード	<code>process_page()</code> の <i>granularity</i> オプション	<code>process_page()</code> の <i>tetml</i> オプション
<i>glyph</i>	<code>granularity=glyph</code>	<code>tetml={glyphdetails={all}}</code>
<i>word</i>	<code>granularity=word</code>	-
<i>wordplus</i>	<code>granularity=word</code>	<code>tetml={glyphdetails={all}}</code>
<i>Line</i> 要素付き <i>word</i>	<code>granularity=word</code>	<code>tetml={elements={line}}</code>
<i>Line</i> 要素付き <i>wordplus</i>	<code>granularity=word</code>	<code>tetml={glyphdetails={all} elements={line}}</code>
<i>line</i>	<code>granularity=line</code>	-
<i>page</i>	<code>granularity=page</code>	-

**TETML 出力を制御するための文書オプション** この項では、生成される TETML 出力を直接制御するさまざまなオプションの効力をまとめます。他のすべての文書オプションは、処理の詳細を制御するために用いることができます。文書オプションの完全な説明は、172 ページの表 10.8 にあります。

文書関連のオプションは、`--docopt` コマンドラインオプションに、または `open_document()` 関数に与える必要があります。

`tetml` オプションは、TETML の一般的な諸側面を制御します。いくつかの TETML 要素は、必要なければ、`elements` サブオプションを用いてなくすこともできます。下記の文書オプションリストは、生成される TETML 出力内に文書レベルの XML メタデータがないようにしています：

```
tetml={ elements={nodocxmp} }
```



**engines** オプションは、テキスト・画像抽出エンジンを有効化または無効化します。下記のオプションリストは、テキスト内容を処理しますが、画像処理は無効化しています：

```
engines={noimage}
```

TETML を生成する際に与えられたすべての文書オプションは、下記の文書オプションで無効化されないかぎり、*/TET/Document/Options* 要素内に記録されます：

```
tetml={ elements={nooptions} }
```

**TETML 出力を制御するためのページオプション** ページオプションの完全な説明は表 10.10 にあります。ページ関連のオプションは、**--pageopt** コマンドラインオプションに、または **process\_page()** 関数に与える必要があります。

**tetml** オプションは、**Glyph** 要素内の座標・フォント関連情報を有効化または無効化します。下記のページオプションリストは、**Glyph** 要素内のフォント詳細を有効化していますが、それ以外のグリフ属性はないようにしています：

```
tetml={ glyphdetails={font} }
```

下記のページオプションリストは、TETML 出力に **Line** 要素を加えています：

```
tetml={ glyphdetails={font} elements={line} }
```

下記のページオプションは、下付きと上付きを示すために、**Glyph** 要素に **sub**・**sup** 属性を追加しています：

```
tetml={ glyphdetails={sub sup} }
```

下記のページオプションは、**all** を用いて、**Glyph** 要素に可能なすべての属性を生成しています：

```
tetml={ glyphdetails={all} }
```

下記のページオプションは、デフォルトの上向き座標ではなく下向き座標を要請しています：

```
topdown={output}
```

下記のページオプションリストは、TET に対して、約物キャラクタを隣接する単語に結合するよう指示しています。すなわちこの場合、約物キャラクタは独立した単語としては扱われません：

```
contentanalysis={nopunctuationbreaks}
```

下記のページオプションは、**page** モードでのみ意味を持ちます。デフォルト区切りキャラクタをラインフィードからスペースへ変更します：

```
contentanalysis={lineseparator=U+0020}
```

TETML を生成する際に与えられたすべてのページオプションは、下記の文書オプションで無効化されないかぎり、*/TET/Document/Pages/Page/Options* 要素内に（各ページごとに個別に）記録されます：

```
tetml={ elements={nooptions} }
```

**例外処理** PDF 解析中にエラーが発生した場合、TET は一般には、可能ならばその問題を修復または無視しようとし、可能でないなら例外を発生させます。しかし、TET で TETML 出力を生成している際には、PDF 解析上の問題は通常、TETML 内の *Exception* 要素として報告されます：

```
<Exception errnum="4506">Object 'objects[49]/Subtype' does not exist</Exception>
```

TETML 出力を処理している際に、期待した要素でなく *Exception* 要素が現れた場合をいかに処理するかは、アプリケーション側で手はずを整える必要があります。

TETML 出力ファイルの生成を妨げる問題の場合は（出力ファイルへの書き込み権限がないなど）、例外が発生し、有効な TETML 出力は生成されません。

## 9.3 TETML のさまざまな要素と TETML スキーマ

すべての TETML 要素・属性とそれらの関係についての正式な XML スキーマ定義 (XSD) が、TET ディストリビューションに含まれています。TET 名前空間は下記のとおりです：

<http://www.pdflib.com/XML/TET3/TET-3.0>

このスキーマは Web 上の下記 URL からダウンロードすることもできます：

<http://www.pdflib.com/XML/TET3/TET-3.0.xsd>

各 TETML 文書のルート要素内には、TETML 名前空間とスキーマロケーションがともに存在しています。

表 9.3 に、すべての TETML 要素の役割を説明します。TET 4.1 と TET 4.0 で導入された属性にはその旨記しています。図 9.1 に、TETML 要素群の XML 階層構造を図示します。

表 9.3 TETML 要素・属性一覧

TETML 要素	説明と属性
<b>Attachment</b>	PDF 添付については、入れ子になった Document 要素内にその内容を記述します。非 PDF 添付にすいては、その名前のみが挙げられ、内容は記述されません。 属性：name・level・pagenumber
<b>Attachments</b>	Attachment 要素群のコンテナ
<b>Box</b>	単語の座標を記述します。属性 llx と lly は Box の左下隅を、urx と ury は右上隅を、標準 PDF 座標で記述します。Box が表す矩形の各辺がページの各辺と平行な場合には、4 個の値 llx・lly・urx・ury は左下隅と右上隅を表します。そうでない場合には 4 個の隅すべてが存在します。1 個の単語が複数の Box 要素を含む場合もあります。たとえばハイフン区切りされた単語が複数のテキスト行にわたっている場合や、単語の先頭が大きなキャラクタになっている場合などです。 属性：llx・lly <sup>1</sup> ・urx・ury <sup>1</sup> ・ulx・uly <sup>1</sup> ・lrx・lry <sup>1</sup>
<b>Cell</b>	1 個の表セルの内容を記述します。 属性：colSpan
<b>ColorSpace</b>	PDF 色空間を記述します。 属性：alternate・base・components・id・name
<b>ColorSpaces</b>	ColorSpace 要素群のコンテナ
<b>Content</b>	ページ内容を階層構造として記述します。 属性：granularity・dehyphenation (TET 4.0)・dropcap (TET 4.0)・font・geometry・shadow (TET 4.0)・sub (TET 4.0)・sup (TET 4.0)
<b>Creation</b>	TET 実行に関する日付とオペレーティングシステムプラットフォームと、TET のバージョン番号を記述します。 属性：platform・tetVersion・date
<b>DocInfo</b>	定義済・カスタム文書情報項目
<b>Document</b>	PDF ファイル名・サイズ・PDF バージョン番号を含む一般的文書情報を記述します。 属性：filename・pageCount・filesize・linearized・pdfVersion・pdfa (TET 4.0 : PDF/A-2 に対する新しい値群。TET 4.1 : PDF/A-3 に対する新しい値群)・pdfe (TET 4.0・TET 4.1 : PDF/E-2 に対する新しい値群)・pdfx・pdfua (TET 4.1)・pdfvt (TET 4.1)・pdfx (TET 4.1 : 列挙値群)・tagged

表 9.3 TETML 要素・属性一覧

TETML 要素	説明と属性
<b>Encryption</b>	さまざまなセキュリティ設定を記述します。 属性 : keylength · algorithm (TET 4.1 : 新しい値 8 ~ 11) · attachment (TET 4.1) · description (TET 4.1 : アルゴリズムに対する新しい値 8 ~ 11) · masterpassword · userpassword · noprint · nomodify · nocopy · noannots · noassemble · noforms · noaccessible · nohiresprint · plainmetadata
<b>Exception</b>	TET が発生させた例外に関連するエラーメッセージ・番号。PDF データ構造が正しくないために入力から十分な情報を抽出できない場合には、この Exception 要素が他の要素を置き換えることがあります。 属性 : errnum
<b>Font</b>	フォントリソースを記述します。必須の name 属性は正準フォント名を内容として持ち、オプションな fullname 属性はサブセット接頭辞を含むフォント名を持ちます。 属性 : ascender (TET 4.1) · capheight (TET 4.1) · descender (TET 4.1) · embedded · fullname (TET 4.0) · id · italicangle (TET 4.1) · type · name · vertical · weight (TET 4.1) · xheight (TET 4.1)
<b>Fonts</b>	Font 要素群のコンテナ
<b>Glyph</b>	1 個のグリフに対するフォント・位置情報の詳細を記述します。この要素は、このグリフが生み出す Unicode キャラクタ (1 個ないし複数) を内容として持ちます。1 個のグリフが複数のキャラクタを生成することもあります。たとえば合字の場合などです。単語に対する Glyph 要素群は、1 個ないし複数の Box 要素内にまとめられています。 属性 : x · y <sup>1</sup> · width · alpha <sup>1</sup> · beta <sup>1</sup> · shadow (TET 4.0) · dropcap (TET 4.0) · font · size · sub (TET 4.0) · sup (TET 4.0) · textrendering · unknown · dehyphenation (TET 4.0)
<b>Image</b>	画像リソースを、すなわち、画像を構成するピクセル配列本体を記述します。 属性 : bitsPerComponent · colorspace · extractedAs (TET 4.0) · height · id · mask · maskonly · mergetype · width
<b>Images</b>	Image 要素群のコンテナ
<b>Line</b>	1 個の行に対するテキスト。TET 4.0 : Line は Word 要素群を含むこともありえます。
<b>Metadata</b>	文書・フォント・画像のいずれかと紐付けられる XMP メタデータ。
<b>Options</b>	TETML を生成するために用いられた文書またはページオプション。
<b>Page</b>	1 個のページの内容。 属性 : number · height · width · topdown (TET 4.0)
<b>Pages</b>	Page 要素群のコンテナ
<b>Para</b>	1 個の段落を構成するテキスト。
<b>PlacedImage</b>	ページ上に配置されている画像のインスタンスを記述します。 属性 : alpha <sup>1</sup> · beta <sup>1</sup> · height · image · width · x · y <sup>1</sup>
<b>Resources</b>	色空間リソース・フォントリソース・画像リソース。
<b>Row</b>	1 個ないし複数の表セル。
<b>Table</b>	1 個ないし複数の表行。
<b>TET</b>	ルート要素 属性 : version (TET 4.1 は 4.1 を生成します。TET 4.0 は 4.0 を生成しました。TET 3 は 3 を生成しました)

表 9.3 TETML 要素・属性一覧

TETML 要素	説明と属性
<i>Text</i>	単語などの要素のテキスト内容
<i>Word</i>	1 個の単語

1. 垂直座標と角度はすべて、topdown ページオブションに従って、左下隅か左上隅のいずれかからの相対位置で表されます。

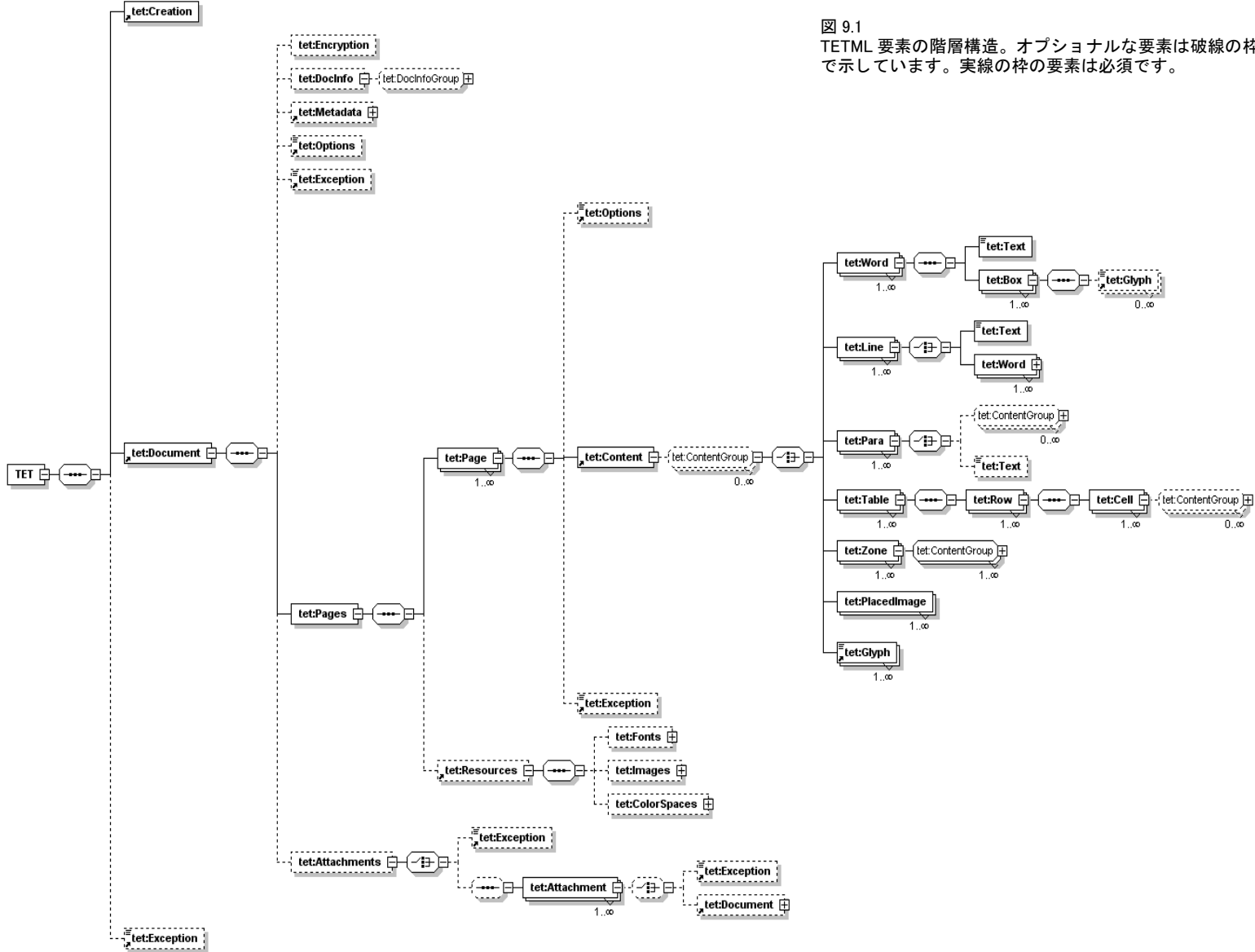


図 9.1  
TETML 要素の階層構造。オプションな要素は破線の枠  
で示しています。実線の枠の要素は必須です。

## 9.4 TETML を XSLT で変換

**XSLT のごく簡単な概略** XSLT (*eXtensible Stylesheet Language Transformations* の略) は、XML 文書を他の文書へ変換するための言語です。その入力はずっと XML 文書ですが (私たちの場合は TETML 文書)、出力は必ずしも XML である必要はありません。XSLT では、任意の計算を行うこともでき、プレーンテキストや HTML 出力を生成することもできます。XSLT スタイルシートを用いて TETML 入力を処理し、その入力に基づいた新しいデータセット (テキスト・XML・CSV・HTML のいずれかの形式で提供) を生成しましょう。この TETML 入力は、PDF 文書の内容を反映しています。TETML 文書は、131 ページの 9.1 「TETML を生成」で説明したように、TET コマンドラインツールで、または TET ライブラリで生成されています。

XSLT は非常に強力ですが、通常のプログラミング言語とはかなり異なっています。XSLT プログラミングの初歩をこの節で解説しようとするつもりはありません。それに關してはさまざまな書籍や Web の情報がありますのでそちらを参照してください。私たちのサンプルは XSLT 1.0 だけにしています。いろいろな XSLT 2.0 実装が入手可能ではありますが、XSLT 1.0 に比べるとまだ広く普及しているとはいえない現状です。XSLT 1.0 の仕様は [www.w3.org/TR/xslt](http://www.w3.org/TR/xslt) にあります。

とはいえ、読者が TETML 文書の XSLT 処理にすばやく着手し実現できるよう手助けすることは私たちの望みです。この節では、XSLT スタイルシートを動作させるために最も重要な環境を説明するとともに、この目的のために広く利用されているソフトウェアを挙げます。XSLT スタイルシートを XML 文書に適用するためには、XSLT プロセッサが必要です。無償や商用のさまざまな XSLT プロセッサが入手可能であり、そのなかにはスタンドアロンなやり方で利用できるものもあれば、プログラミング言語の助けによって自分のプログラムの中で利用可能なものもあります。

XSLT スタイルシートでは、処理の詳細を制御するために環境からスタイルシートへ渡されるパラメータを活用することも可能です。私たちの XSLT サンプルのなかにもスタイルシートパラメータを利用しているものがありますので、さまざまな環境でパラメータをスタイルシートへ受け渡す方法についても情報を提供します。

さまざまなパッケージングで利用できる、広く利用されている XSLT プロセッサには、以下のようなものがあります：

- ▶ Microsoft の MSXML という XML 実装。Windows 2000 SP4 からオペレーティングシステムに内蔵されています。
- ▶ Microsoft の .NET Framework 2.0 XSLT 実装
- ▶ Saxon。無償版と商用版が入手可能です。
- ▶ Xalan。Apache ファウンデーションがホストしているオープンソースプロジェクトです (C++ 実装と Java が入手可能です)。
- ▶ GNOME プロジェクトのオープンソースの *libxslt* ライブラリ
- ▶ Sablotron。オープンソースの XSLT ツールキットです。

**コマンドラインで XSLT** XSLT スタイルシートをコマンドラインから適用することは、便利な開発・試験環境を提供します。以下のさまざまな例は、XSLT スタイルシートをコマンドライン上で適用する方法を示しています。サンプルはすべて、入力ファイル *FontReporter.tetml* をスタイルシート *tetml2html.xsl* で処理しており、その際、XSLT パラメータ *toc-generate* (スタイルシート内で用いられている) を値 *o* に設定しており、そして生成された出力を *FontReporter.html* へ送っています：

- ▶ Java ベースの Saxon プロセッサ ([www.saxonica.com](http://www.saxonica.com) 参照) は下記のように使えます：

```
java -jar saxon9.jar -o FontReporter.html FontReporter.tetml tetml2html.xml ←
toc-generate=0
```

- ▶ xsltproc ツールは、多くの Linux ディストリビューションに含まれています。[xmlsoft.org/XSLT](http://xmlsoft.org/XSLT) を参照してください。スタイルシートを TETML 文書に適用するには下記のコマンドを用います：

```
xsltproc --output FontReporter.html --param toc-generate 0 tetml2html.xml ←
FontReporter.tetml
```

- ▶ Xalan C++ は、コマンドラインツールを提供しており、これは下記のようにして起動できます：

```
Xalan -o FontReporter.html -p toc-generate 0 FontReporter.tetml tetml2html.xml
```

- ▶ MSXML パーサを持つ Windows システムでは、Microsoft が提供している無償の *msxsl.exe* プログラムを利用できます。このプログラムは（ソースコードも含め）下記の場所で入手可能です：

[www.microsoft.com/download/en/details.aspx?displaylang=en&id=21714](http://www.microsoft.com/download/en/details.aspx?displaylang=en&id=21714)

このプログラムは下記のように動作させます：

```
msxsl.exe FontReporter.tetml tetml2html.xml -o FontReporter.html toc-generate=0
```

**自分のアプリケーション内で XSLT** 自分のアプリケーションの中に XSLT 処理を組み込みたい場合には、XSLT プロセッサの選択は当然、使うプログラミング言語と環境に依存します。TET ディストリビューションは、さまざまな重要な環境のためのサンプルコードを含んでいます。*runxslt* サンプル群は、TETML 文書を読み込み、XSLT スタイルシートをパラメタ付きで適用し、生成された出力をファイルへ書き出す方法を演示しています。これらのプログラムは、引数なしで実行された場合には、TET ディストリビューションとともに提供されているすべての XSLT サンプルを実行します。あるいは、TETML 入力ファイル名・XSLT スタイルシート名・出力ファイル名・パラメタ / 値対群について引数群を与えることもできます。*runxslt* サンプル群は、XSLT 処理を自分のアプリケーションへ組み込むための出発点として活用することができます：

- ▶ Java 開発者は、*javax.xml.transform* パッケージ内のメソッド群を利用できます。これは *runxslt.java* サンプルで演示しています。また、*ant* ビルドツールでは、コーディングなしで Java ベースの XSLT を実行できます。TET ディストリビューション内の *build.xml* ファイルは、全サンプル用の XSLT タスク群を含んでいます。
- ▶ .NET 開発者は、*System.Xml.Xsl.XslTransform* 名前空間内のメソッド群を利用できます。これは *runxslt.ps1* PowerShell スクリプトで演示しています。同様のコードを、C# や他の .NET 言語で利用できます。
- ▶ COM オートメーションに対応しているすべての Windows ベースのプログラミング言語は、MSXML パーサが提供している *MSXML2.DOMDocument* オートメーションクラスのメソッド群を利用できます。これは *runxslt.vbs* サンプルで演示しています。同様のコードを、他の COM 対応言語で利用できます。

Perl など他の多くのプログラミング言語では、XSLT 拡張が利用可能です。

**Web サーバで XSLT** XML から HTML への変換は XSLT の用途として代表的なものですので、XSLT スタイルシートを Web サーバ上で動作させたいときも多くあります。いくつかの重要なシナリオ：



- ▶ ASPまたはASP.NETを持つWindowsベースのWebサーバでは、上述のCOMまたは.NETインタフェースを利用できます。
- ▶ Java ベースの Web サーバでは *javax.xml.transform* パッケージを利用できます。
- ▶ PHP ベースの Web サーバでは Sablotron プロセッサを利用できます。 [www.php.net/manual/en/intro.xsl.php](http://www.php.net/manual/en/intro.xsl.php) を参照してください。

**Web ブラウザで XSLT** 現在普及している多くのブラウザも、XSLT 変換に対応していません。XSLT スタイルシートを TETML 文書に適用するようブラウザに指示するには、その TETML 文書の *xml* 処理命令を含む先頭行の後、かつルート要素の前に、適切な処理命令を持つ行を追加します。そしてそれをブラウザへ読み込めば、ブラウザはそのスタイルシートを適用し、結果の出力を表示します（なお、Internet Explorer では、ローカルディスクからのファイル进行处理の際にはファイル名接尾辞 *.xml* が必要です）：

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="tetml2html.xsl" version="1.0"?>
<TET xmlns="http://www.pdflib.com/XML/TET3/TET-3.0"
...
```

ブラウザは、XSLT スタイルシートを TETML 文書に適用し、そしてその結果のテキスト・HTML・XML のいずれかの出力を表示します。あるいは、ブラウザでの XSLT 処理は、JavaScript コードから引き起こさせることも可能です。

Firefox 2 以上では、*xslt-param* 処理命令を用いてパラメータを XSLT スタイルシートに与えることもできます：

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="tetml2html.xsl" version="1.0"?>
<?xslt-param name="toc-generate" value="0"?>
<TET xmlns="http://www.pdflib.com/XML/TET3/TET-3.0"
...
```

## 9.5 さまざまな XSLT サンプル

TET ディストリビューションは、XSLT を TETML に適用した場合の強力を演示する XSLT スタイルシートをいくつか含んでいます。これらは、TETML アプリケーションの出発点として活用することもできます。この節では、この XSLT サンプル群の概要を紹介するとともに、サンプル出力を掲載します。143 ページの 9.4 「TETML を XSLT で変換」で、XSLT スタイルシートを動作させるための多くのオプションを説明しています。このスタイルシート群の機能と内部動作に関して詳しくは、XSLT コードのコメントに記してあります。このスタイルシートサンプル群のいくつかの一般的側面：

- ▶ 多くの XSLT サンプルはパラメタに対応しており、これを用いてさまざまな処理詳細を制御することができます。これらのパラメタは、XSLT コード内で設定することもできますし、環境 (*ant* など) から上書きすることもできます。
- ▶ 多くの XSLT サンプルでは、特定のテキストモード (たとえば *word* モード。詳しくは、135 ページの「さまざまな TETML テキストモード」を参照) の TETML 入力を必要とします。それらのサンプルは、誤った入力から自己を守るために、与えられた TETML 入力が必要に準拠しているかどうかをチェックして、準拠していないならエラーを報告します。
- ▶ XSLT サンプルのなかには、文書内の PDF 添付を再帰的に処理するものがあります (これは後述の説明の中でその旨記しています)。しかし多くのサンプルは PDF 添付を無視します。それらも、添付を処理するように拡張することが容易なように書かれています。*Attachments* 要素内の対象要素を選択すれば充分です。関連する *xsl:template* 要素自体に変更を加える必要はありません。
- ▶ XSLT サンプルはすべて XSLT 1 で動作します。サンプルによっては XSLT 2 の機能を使えばもっと簡単化できるものもありますが、使いやすさを考えて XSLT 1 にこだわりたいと考えました。

**コンコーダンスを生成** *concordance.xsl* スタイルシートは、*word* または *wordplus* モードの TETML 入力を受け付けます。これはコンコーダンスを、すなわち、文書内の単語を出現頻度の高い順に並べた一覧を生成します。これは、言語分析のためのコンコーダンスや、翻訳者のための相互参照や、整合性チェックなどを生成するのに有用でしょう。

List of words in the document along with the number of occurrences:

```
the 207
font 107
of 100
a 92
in 83
and 75
fonts 64
PDF 60
FontReporter 58
...
```

**フォントフィルタリング** *fontfilter.xsl* スタイルシートは、*glyph* または *wordplus* モードの TETML 入力を受け付けます。これは文書内の、特定のフォントを用いた、かつ指定された値より大きなサイズの単語の一覧を作ります。これは、特定のフォント / サイズの組み合わせを検出したり、品質管理のために有用でしょう。同じコンセプトを用いて、大きな文字サイズを用いたテキスト部分に基づいた目次を生成することもできます。

Text containing font 'TheSansBold-Plain' with size greater than 10:

```
[TheSansBold-Plain/24] Contents
[TheSansBold-Plain/13.98] 1
[TheSansBold-Plain/13.98] Installing
[TheSansBold-Plain/13.98] PDFlib
[TheSansBold-Plain/13.98] FontReporter
[TheSansBold-Plain/13.98] 2
[TheSansBold-Plain/13.98] Working
[TheSansBold-Plain/13.98] with
[TheSansBold-Plain/13.98] FontReporter
[TheSansBold-Plain/13.98] A
[TheSansBold-Plain/13.98] Revision
[TheSansBold-Plain/13.98] History
[TheSansBold-Plain/24] 1
[TheSansBold-Plain/24] Installing
[TheSansBold-Plain/24] PDFlib
[TheSansBold-Plain/24] FontReporter
...
```

**フォントの使用箇所を検索** *fontfinder.xsl* スタイルシートは、*glyph* または *wordplus* モードの TETML 入力を受け付けます。文書内のすべてのフォントについて、特定のフォントを用いているテキストの出現箇所をすべて、そのページ番号とページ上の位置とともに一覧にします。これは、望ましくないフォントを検出して整合性をチェックしたり、特定の悪い文字サイズの使用を見つけたりするために有用でしょう。

TheSansExtraBold-Plain used on:

page 1:

(111, 636), (165, 636), (219, 636), (292, 636), (301, 636), (178, 603), (221, 603), (226, 603), (272, 603), (277, 603), (102, 375), (252, 375), (261, 375), (267, 375)

TheSans-Plain used on:

page 1:

(102, 266), (119, 266), (179, 266), (208, 266), (296, 266), (346, 266), (367, 266)

...

**フォント統計** *fontstat.xsl* スタイルシートは、*glyph* または *wordplus* モードの TETML 入力を受け付けます。これはフォントとグリフの統計を生成します。これは、品質管理に有用なほか、各フォントについてマップなしグリフ（すなわち、いかなる Unicode キャラクターをもマップできないグリフ）も報告されますので、アクセシビリティ試験にも有用でしょう。

19894 total glyphs in the document; breakdown by font:

```
68.71% ThesisAntiqua-Normal: 13669 glyphs
22.89% TheSans-Italic: 4553 glyphs
6.38% TheSansBold-Plain: 1269 glyphs
0.9% TheSansMonoCondensed-Plain: 179 glyphs
0.49% TheSansBold-Italic: 98 glyphs
0.27% TheSansExtraBold-Plain: 54 glyphs
0.21% TheSerif-Caps: 42 glyphs
0.15% TheSans-Plain: 29 glyphs
0.01% Gen_TheSans-Plain: 1 glyphs
```

**索引を生成** *index.xml* スタイルシートは、*word* または *wordplus* モードの TETML 入力を受け付けます。これは索引を、すなわち、文書内の単語をアルファベット順に並べ、おのこのページ番号とともに一覧にしたものを生成します。数字と約物キャラクターは無視されます。

Alphabetical list of words in the document along with their page number:

A  
about 2 7 8  
access 8 12  
accessible 11  
achieving 9 12  
Acrobat 2 5 7 8 9 10 11 14 15 17  
ActiveX 2  
actual 9 12  
actually 11 12 14  
addition 9  
Additional 12  
additions 17  
address 9 12  
addressed 9  
addressing 9  
Adobe 2 5 8 12 14  
...

**XMP メタデータを抽出** *metadata.xml* スタイルシートは、任意のモードの TETML 入力を受け付けます。これは文書レベルの XMP メタデータを対象としており、XMP からいくつかのメタデータプロパティを抽出します。文書内の PDF 添付 (PDF パッケージ・ポートフォリオを含め) は再帰的に処理されます:

```
dc:creator = PDFlib GmbH  
xmp:CreatorTool = FrameMaker 7.0
```

**表組内容を CSV 形式で抽出** *table.xml* スタイルシートは、*word*・*wordplus*・*page* のいずれかのモードの TETML 入力を受け付けます。これは、指定された表組の内容を抽出して、その表組内容を含む CSV ファイル (カンマ区切り値) を生成します。CSV ファイルは、あらゆる表計算アプリケーションで開くことができます。これは、PDF 文書内の表組の内容を再利用するのに有用でしょう。

**TETML を HTML へ変換** *tetml2html.xml* スタイルシートは、*wordplus* モードの TETML 入力を受け付けます。これは TETML を、ブラウザで表示できる HTML へ変換します。このコンバータは、PDF 文書と同等の視覚表現を生成しようとするのではなく、以下の側面を演示するものです:

- ▶ 見出し要素 (*H1*・*H2* など) を、設定可能な文字サイズに基づいて生成します。
- ▶ TETML 内の表組要素を、おのおの HTML 表組構造へマップして、表組をブラウザで表示可能にします。
- ▶ HTML ページの冒頭に目次を生成します。その各項目は、文書内の何らかの見出しに基づいており、おのおの見出しへ飛ぶアクティブリンクを含んでいます。
- ▶ 各ページについて画像の一覧を生成します。おのおの、TET コマンドラインツールによって *resource* 画像ループモードで (例: *tet --image --tetml file.pdf*) 生成された画像ファイル名を用いて画像ファイルへリンクされています。

**生テキストを TETML から抽出** `textonly.xsl` スタイルシートは、任意のモードの TETML 入力を受け付けます。これは、すべての `Text` 要素を取得しつつ、他のすべての要素を無視することにより、生テキスト内容を抽出します。文書内の PDF 添付 (PDF パッケージ・ポートフォリオを含め) は再帰的に処理されます。



# 10 TET ライブラリ API リファレンス

## 10.1 オプションリスト

オプションリストは、さまざまな API 関数呼び出しを制御するための、強力で、それでいて簡単な方式です。多くの API メソッドでは、オプションリストが使えるようになっていますので、関数に膨大な引数を与える必要がありません。略して *optlist* ともいいます。これは、任意の数のオプションを内容として持つ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、リストのような複合データに対応しています。多くの言語バインディングでは、オプションリストは、必要なキーワードと値を連結していくことによって、簡単に構築することができます。

**バインディング** C 言語バインディング : *sprintf()* 関数を用いてオプションリストを構築するとよいでしょう。

**バインディング** .NET 言語バインディング : C# プログラマーは、*AppendFormat()* *StringBuilder* メソッドは整形項目を表すのに中括弧 `{}` を用いて、それが引数の文字列表現へ置き換えられることに留意する必要があります。一方 *Append()* メソッドでは、中括弧キャラクタにいかなる特別な意味をも持たせていません。オプションリスト文法は中括弧キャラクタを利用していますので、*AppendFormat()* と *Append()* のどちらのメソッドを使うかを選ぶときには注意が必要です。

## 10.2 オプションリスト文法

**正式なオプションリスト文法定義** オプションリストは、以下の規則に従って構築する必要があります :

- ▶ オプションリスト内のすべての要素 (キーと値) は、1 個ないし複数の右記区切りキャラクタによって区切る必要があります : スペース・タブ・キャリッジリターン・ニューライン・等号 [=]。
- ▶ 一番外側の囲み中括弧は、要素には含まれません。 `{}` と書けば空要素を意味します。
- ▶ 一番外側の中括弧の中の区切りキャラクタは、要素を分割する効力をもはや持たず、要素の一部となります。ですので、区切りキャラクタを含む要素は、中括弧で囲む必要があります。
- ▶ 先頭または末尾に中括弧を持つ要素は、中括弧で囲む必要があります。
- ▶ 要素が片方だけの中括弧を含む場合には、その中括弧は直前にバックスラッシュキャラクタを付けて保護する必要があります。要素の閉じ中括弧の直前のバックスラッシュも、直前にバックスラッシュキャラクタを付ける必要があります。
- ▶ オプションリストはバイナリゼロ値を含んではいけません。

オプションリストは、この PDFlib リファレンス内の仕様に従い、リスト値を持つことができます。リスト値は 1 個ないし複数の要素 (これ自体もまたリストである場合もあります) を内容として持ちます。それらは上述の規則に従って区切られますが、等号は区切りキャラクタとしてもはや見なされないという点が唯一異なります。

**単純オプションリスト** 多くの場合、オプションリストは 1 個ないし複数のキー / 値対を内容として持ちます。キーと値は、また複数のキー / 値対も同様に、1 個ないし複数の

空白キャラクタ（スペース・タブ・キャリッジリターン・ニューライン）で区切る必要があります。あるいは、キーは等号「=」で値と区切ることもできます：

```
key=value
key = value
key value
key1 = value1 key2 = value2
```

可読性を増すため、キーと値の間には等号を用い、隣り合うキー/値対の間には空白を用いることを推奨します。

オプションリストは左から右へと評価されますので、オプションは同一リスト内で複数回与えることもできます。この場合、最後に現れたオプションがそれ以前の上書きします。下記の例では、1番目のオプションでの割り当ては2番目によって上書きされますので、オプションリスト処理後に **key** は値 **value2** を持つことになります：

```
key=value1 key=value2
```

**リスト値** リストは、1個ないし複数の区切られた値を内容として持ちます。この値は単純値であることもありますし、それ自身がリスト値であることもあります。リストは中括弧 **{}** で囲まれており、リスト内の値群は空白キャラクタで区切る必要があります。例：

```
searchpath={/usr/lib/tet d:\tet}          (ディレクトリ名2個を持つリスト)
```

リストは、入れ子になったリストを内容として持つこともあります。この場合、各リストの間は空白で区切る必要があります。区切りキャラクタは、隣り合うキャラクタ **}** と **{** の間には挿入する必要がありますが、同じ種類の中括弧どうしの間では省くこともできます：

```
fold= { [:Private_Use:] remove } { [U+FFFD] remove } } (リスト2個を持つリスト)
```

リストがちょうど1個のリストを内容として持つときも、入れ子になったリストの中括弧群を省くことはできません：

```
fold= { [:Private_Use:] remove } } (入れ子のリスト1個を持つリスト)
```

**入れ子になったオプションリストとリスト値** オプションによっては、オプションリスト型またはオプションリストのリスト型を受け付けるものがあります。オプションリスト型のオプションは、1個ないし複数の子オプションリストを内容として持ちます。オプションリストのリスト型のオプションは、1個ないし複数の入れ子になったオプションリストを内容として持ちます。入れ子になったオプションリストを扱う際には、囲む中括弧の数を正しく指定することが重要です。いくつかの例を以下に挙げます。

オプション **contentanalysis** の値はオプションリストであり、そのオプションリスト自体が1個のオプション **punctuationbreaks** を内容として持ちます：

```
contentanalysis={punctuationbreaks=false}
```

下記の例で、オプション **glyphmapping** の値は、オプションリストただ1個を内容として持つオプションリストのリストです：

```
glyphmapping= { {fontname=GlobeLogosOne codelist=GlobeLogosOne} }
```

下記の例で、オプション **glyphmapping** の値は、オプションリスト2個を内容として持つオプションリストのリストです：



```
glyphmapping { {fontname=CMSY* glyphlist=tarski} {fontname=ZEH* glyphlist=zeh}}
```

オプションリスト 1 個を内容として持つリストにおいて、そのオプションリストの中の *fontname* 値がスペースを含んでいるので、さらに中括弧で囲む必要があります：

```
glyphmapping={ {fontname={Globe Logos One} codelist=GlobeLogosOne} }
```

キーワード 2 個を内容として持つリスト：

```
fonttype={Type1 TrueType}
```

異なる型が混在したリスト。内側のリスト群は、Unicode 集合 1 個とキーワード 1 個を内容として持っており、外側のリストは、オプションリスト 2 個とキーワード *default* を内容として持っています：

```
fold={ {[:Private_Use:] remove} {[U+FFFF] remove} default }
```

矩形 1 個を内容として持つリスト：

```
includeboxes={{10 20 30 40}}
```

**はまりやすい罫** この項では、オプションリスト文法についてよくある誤りを挙げます。中括弧は区切りキャラクタではありませんので、下記は誤りです：

```
key1 {value1}key2 {value2}                      誤り!
```

これはエラーメッセージ *Unknown option 'value2'* を引き起こします。同様に、下記は区切りキャラクタが抜けているので誤りです：

```
key{value}                                      誤り!  
key={{value1}{value2}}                      誤り!
```

中括弧は照応している必要がありますので、下記は誤りです：

```
key={open brace { }                              誤り!
```

これはエラーメッセージ *Braces aren't balanced in option list 'key={open brace {}'* を引き起こします。文字列の中のただ 1 個の中括弧は、バックスラッシュキャラクタを直前に付ける必要があります：

```
key={closing brace \} and open brace \{ }      正しい!
```

文字列値の末尾のバックスラッシュは、直後が閉じ中括弧キャラクタである場合には、もう 1 個のバックスラッシュを直前に付ける必要があります：

```
filename={C:\path\name\}                      誤り!  
filename={C:\path\name\\}                      正しい!
```

## 10.3 基本型

**文字列** 文字列は、一般に非ローカライズのキーワードのために用いられるプレーン ASCII 文字列です (EBCDIC プラットフォームでは EBCDIC 文字列)。空白または「=」キャラクターを含む文字列は、`{}` で囲む必要があります：

```
password={ secret string }           (3個の空白を含む文字列値)  
contents={length=3mm}                (1個の等号を含む文字列値)
```

キャラクター `{}` は、文字列の一部としたい場合には、\キャラクターを直前に付ける必要があります：

```
password={weird\}string}             (右中括弧を含む文字列値)
```

要素の閉じ中括弧の直前のバックスラッシュは、直前にバックスラッシュキャラクターを付ける必要があります：

```
filename={C:\path\name\}            (1個のバックスラッシュで終わる文字列)
```

空文字列は、中括弧の対で構築できます：

```
{}
```

内容文字列・ハイパーテキスト文字列・名前文字列：これらは、さまざまな形式の Unicode 内容を持つことができます。パラメタ `escapesequence` が設定されている場合には、シングルバイトはエスケープシーケンスで表すこともできます。これらの文字列型と、文字列オプションに対するエンコーディング選択について詳しくは、*PDFlib チュートリアル* を参照してください。

非Unicode対応言語バインディング：オプションリストの先頭が [EBCDIC-]UTF-8 BOM である場合には、オプションリストの内容・ハイパーテキスト・名前文字列はそれぞれ [EBCDIC-]UTF-8 文字列として解釈されます。

**Unichar** Unichar は、1 個の Unicode 値であり、いくつかの種類の文法に対応しています：10 以上の 10 進値 (例：173)、接頭辞 `x`・`X`・`ox`・`oX`・`U+` のいずれかを付けた 16 進値 (`xAD`・`oxAD`・`U+00AD`)、数値参照、文字参照、グリフ名参照から「&」・「;」修飾を除いたもの (`shy`・`#xAD`・`#173`)。あるいは、リテラルなキャラクターを与えることもできます。Unichar は、範囲 0 ~ 1 114 111 (0 ~ 0x10FFFF) 内である必要があります：例：

```
unknownchar=?                         (リテラル)  
unknownchar=63                        (10進)  
unknownchar=x3F                       (16進)  
unknownchar=0x3F                     (16進)  
unknownchar=U+003F                   (Unicode記法)  
lineseparator={CRLF}                 (標準グリフ名参照)
```

1 個の数字キャラクターは、10 進 Unicode 値としてではなく、リテラルに扱われます：

```
replacementchar=3                     (U+0033 THREE。U+0003ではありません!)
```

**Unicode 集合** Unicode 集合は、以下の構成要素によって構築することができます：

- ▶ パターンは、一連のキャラクターを角括弧でまとめたものであり、Unicode キャラクター群と Unicode プロパティ集合群のリストを内容として持ちます。

- ▶ リストは、Unicode キャラクタの連鎖であり、2 個のキャラクタの間に「-」で示した範囲を持つことができます。例：`U+FB00-U+FB17`。連鎖は、Unicode 順に左から右までのすべてのキャラクタの範囲を示します。複数の Unicode キャラクタは、間を空白で区切らずに直接つなげて書く必要があります。例：`U+0048U+006C`。
- ▶ リスト内の Unicode キャラクタは以下のように指定できます：
  - ASCII キャラクタはリテラルとして指定できます
  - ちょうど 4 桁の 16 進数字：`\uhhhh` または `U+hhhh`
  - ちょうど 5 桁の 16 進数字：`U+hhhhh`
  - 1～6 桁の 16 進数字：`\x{hhhhhh}`
  - ちょうど 8 桁の 16 進数字：`\Uhhhhhhhh`
  - エスケープされたバックスラッシュ：`\\`
- ▶ Unicode プロパティ集合は、Unicode プロパティによって指定されます。プロパティ名を指定するための文法は、POSIX と Perl 文法の拡張であり、ここで **type** は Unicode プロパティの名前を ([www.unicode.org/Public/UNIDATA/PropertyAliases.txt](http://www.unicode.org/Public/UNIDATA/PropertyAliases.txt) 参照)、**value** はその値を ([www.unicode.org/Public/UNIDATA/PropertyValueAliases.txt](http://www.unicode.org/Public/UNIDATA/PropertyValueAliases.txt) 参照) 表します：
  - POSIX 式文法：`[:type=value:]`
  - POSIX 式文法で否定：`[.^type=value:]`
  - Perl スタイル文法：`\p{type=value}`
  - Perl スタイル文法で否定：`\P{type=value}`**type=** は、Category・Script プロパティについては省けますが、それ以外のプロパティについては必要です。
- ▶ 集合演算をパターンに適用することもできます：
  - 2 個の集合の和をとるには、単純にそれらを連結します：`[[:letter:][:number:]]`
  - 2 個の集合の積をとるには、「&」演算子を用います：`[[:letter:] & [U+0061-U+007A]]`
  - 2 個の集合の差をとるには、「-」演算子を用います：`[[:letter:]-[U+0061-U+007A]]`
  - 補集合を得るには、開き「[」の直後に「^」を入れます：`[^U+0061-U+007A]`。それ以外の場所では、「^」は特別な意味を持ちません。

表 10.1 にいろいろな Unicode 集合の例を挙げます。下記 Web サイトを利用すると、いろいろな Unicode 集合表現を対話的に試すことができます：

[unicode.org/cldr/utility/list-unicodeset.jsp](http://unicode.org/cldr/utility/list-unicodeset.jsp)

**論理値** 論理値は、値 *true* または *false* をとります。論理値オプションで値を省略すると、値 *true* と見なされます。**name=false** の短縮記法として **noname** を用いることも可能です：

```
usehostfonts      (usehostfonts=trueと同等)
nousehostfonts    (usehostfonts=falseと同等)
```

**キーワード** キーワード型のオプションは、決められたキーワードの定義済リストの 1 つをとることができます。例：

```
clippingarea=cropbox
```

オプションによっては、数値とキーワードのいずれかの値をとるものもあります。

**数値** オプションリストは、いくつかの数値型に対応しています。

表 10.1 いろいろな Unicode 集合の例

Unicode 集合の指定	その Unicode 集合内のキャラクタ群
[U+0061-U+007A]	a から z までの小文字
[U+0640]	アラビア文字タトウィールの 1 キャラクタのみ
[\x{0640}]	アラビア文字タトウィールの 1 キャラクタのみ
[U+FB00-U+FB17]	欧文・アルメニア文字の合字群
[^U+0061-U+007A]	a から z まで以外のすべてのキャラクタ
[:Lu:] [:UppercaseLetter:]	すべての大文字 (Unicode 集合の短形式と長形式)
[:L:] [:Letter:]	L で始まるすべての Unicode カテゴリ (Unicode 集合の短形式と長形式)
[:General_Category=Dash_Punctuation:]	一般カテゴリ Dash_Punctuation 内のすべてのキャラクタ
[:Alphabetic=No:]	すべての非アルファベットキャラクタ
[:Private_Use:]	私用領域 (PUA) 内のすべてのキャラクタ

整数型は、10 進と 16 進の整数をとります。x・X・0x・0X のいずれかで始まる正の整数は 16 進値を意味します：

```
-12345
0
0xFF
```

float は、10 進浮動小数点数または整数をとります。ピリオドまたはカンマを浮動小数点値の小数点として使えます。指数記法にも対応しています。以下の値はすべて同等です：

```
size = -123.45
size = -123,45
size = -1.2345E2
size = -1.2345e+2
```

## 10.4 図形型

**矩形** 矩形は、矩形の左下隅と右上隅の  $x \cdot y$  座標を指定する float 値 4 個のリストです。座標を解釈するための座標系（デフォルトまたはユーザ座標系）はオプションによって異なりますので、都度説明してあります。例：

```
includebox = {{0 0 500 100} {0 500 500 600}}
```

## 10.5 一般関数

### 10.5.1 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 RB** `Sub set_option(optlist As String)`

**C** `void TET_set_option(TET *tet, const char *optlist)`

---

TET 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 10.2 に従ってグローバルオプション群を指定したオプションリスト。1 つのオプションが複数回与えられたときは、最後に出現したものがそれ以前のものを上書きします。1 個のオプションに複数の値を与えるためには (*searchpath* など)、このオプションのリスト引数内ですべての値を与えてください。

右記のオプションが使えます：*asciifile*・*cmap*・*codelist*・*encoding*・*filenamehandling*・*fontoutline*・*glyphlist*・*license*・*licensefile*・*logging*・*userlog*・*outputformat*・*resourcefile*・*searchpath*

**詳細** 表 10.2 でその旨記されているオプションについては、この関数を複数呼び出して値を蓄積させることもできます。その旨記していないオプションでは、新しい値が古い値を上書きします。

表 10.2 TET\_set\_option() のグローバルオプション一覧

オプション	説明
<i>asciifile</i>	(論理値。i5/iSeries・zSeries 上でのみ対応) ASCII エンコーディングのテキストファイル (UPR 設定ファイル・グリフリスト・コードリストなど) を受け付けます。デフォルト：i5/iSeries では true、zSeries では false
<i>cmap</i> <sup>1,2</sup>	(名前文字列のリスト) 文字列対のリスト。各対は、CMap リソース 1 個の名前と値を内容として持ちます (69 ページの 5.2 「リソース設定とファイル検索」を参照)。
<i>codelist</i> <sup>1,2</sup>	(名前文字列のリスト) 文字列対のリスト。各対は、コードリストリソース 1 個の名前と値を内容として持ちます (69 ページの 5.2 「リソース設定とファイル検索」を参照)。
<i>encoding</i> <sup>1,2</sup>	(名前文字列のリスト) 文字列対のリスト。各対は、エンコーディングリソース 1 個の名前と値を内容として持ちます (69 ページの 5.2 「リソース設定とファイル検索」を参照)。

表 10.2 TET\_set\_option() のグローバルオプション一覧

オプション	説明
<b>filename-handling</b>	<p>(キーワード。Windows では必須ではありません) ファイル名のターゲットエンコーディング。Windows の場合このオプションは、与えられたファイル名には適用されますが、生成されるファイルの名前には適用されません (デフォルト : Mac OS X では unicode、それ以外では honorlang) :</p> <p><b>ascii</b> 7 ビット ASCII</p> <p><b>basicebcdic</b> コードページ 1047 に従った、ただし Unicode 値 <math>\leq</math> U+007E のみの基本 EBCDIC</p> <p><b>basicebcdic_37</b> コードページ 0037 に従った、ただし Unicode 値 <math>\leq</math> U+007E のみの基本 EBCDIC</p> <p><b>honorlang</b> 環境変数 LC_ALL・LC_CTYPE・LANG を解釈し、それが utf8・UTF-8・cpXXXX・CPXXXX・iso8859-x・ISO-8859-x のいずれかを指定しているならファイル名へ適用。</p> <p><b>legacy</b> auto エンコーディング (すなわちカレント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、honorlang パラメタが設定されているなら LANG 変数を解釈。</p> <p><b>unicode</b> (EBCDIC-) UTF-8 形式の Unicode エンコーディング</p> <p>すべての有効なエンコーディング名</p> <p>TET が認識する任意の (内蔵またはユーザ定義の) エンコーディング</p> <p>Unicode 非対応言語バインディングで UTF-8 BOM なしで length=0 で与えられたファイル名はこの filenamehandling オプションに従って解釈されます。</p>
<b>fontoutline<sup>1,2</sup></b>	(名前文字列のリスト) 文字列対のリスト。各対は、FontOutline リソース 1 個の名前と値を内容として持ちます (69 ページの 5.2 「リソース設定とファイル検索」を参照)。
<b>glyphlist<sup>1,2</sup></b>	(名前文字列のリスト) 文字列対のリスト。各対は、グリフリストリソース 1 個の名前と値を内容として持ちます (69 ページの 5.2 「リソース設定とファイル検索」を参照)。
<b>hostfont<sup>1,2</sup></b>	(名前文字列のリスト) 文字列対のリスト。各対は、埋めこまれていないフォントに対して用いたいホストフォントの PDF フォント名と UTF-8 符号化された名前を内容として持ちます。
<b>license</b>	(文字列) ライセンスキーを設定します。これは、open_document*( ) を初めて呼び出すより前に設定する必要があります。
<b>licensefile</b>	(文字列) ライセンスキー (群) を内容として持つファイルの名前を設定します。このライセンスファイルは、TET_open_document*( ) を初めて呼び出すより前に 1 回だけ設定することができます。あるいは、ライセンスファイルの名前は、PDFLIBLICENSEFILE 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できます。
<b>logging<sup>1</sup></b>	(オプションリスト。非サポート) 表 10.7 に従ってログ記録出力を指定するオプションリスト。あるいは、ログ記録オプション群は、TETLOGGING 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できます。空のオプションリストにすると、前回の呼び出しで設定したオプション群によってログ記録が有効になります。環境変数が設定されている場合には、ログ記録は、TET_new( ) への最初の呼び出しの直後から開始されます。
<b>userlog</b>	(名前文字列。非サポート) ログ記録が有効にされている場合にログファイルへ書き込まれる任意の文字列。
<b>output-format</b>	<p>(キーワード。C・Ruby・Perl・Python・PHP 言語バインディングのみ) TET_get_text( ) が返すテキストの形式を指定します :</p> <p><b>utf8</b> 文字列は、(C の場合はヌル終端の) UTF-8 形式で返されます。</p> <p><b>utf16</b> 文字列は、そのマシンのネイティブバイト順序の UTF-16 形式で返されます。</p> <p><b>utf32</b> 文字列は、そのマシンのネイティブバイト順序の UTF-32 形式で返されます。</p> <p><b>ebcdicutf8</b> (EBCDIC ベースのシステムでのみ利用可能) 文字列は、ヌル終端 EBCDIC 符号化 UTF-8 形式で返されます。i5/iSeries ではコードページ 37 が、zSeries ではコードページ 1047 が用いられます。</p> <p>デフォルト : C・Ruby・Perl・Python・PHP の場合は utf8、i5/iSeries・zSeries 上の C の場合は ebcdicutf8</p>

表 10.2 TET\_set\_option() のグローバルオプション一覧

オプション	説明
<i>resourcefile</i>	<p>(名前文字列) UPR リソースファイルの相対または絶対ファイル名。このリソースファイルはただちに読み込まれます。既存のリソースは温存され、その値は再設定された場合には新しいもので上書きされます。明示的なリソースオプションは、リソースファイル内の項目の後に評価されず。</p> <p>リソースファイル名は、環境変数 TETRESOURCEFILE で、または Windows のレジストリキーで与えることもできます (69 ページの 5.2 「リソース設定とファイル検索」を参照)。デフォルト: tet.upr (MVS では upr)</p>
<i>searchpath</i> <sup>1</sup>	<p>(名前文字列のリスト) 読み込みたいファイル群を含むディレクトリの相対または絶対パス名 (群)。この検索パスは複数回設定することもでき、その場合には項目は蓄積され、設定した順に使用されます (69 ページの 5.2 「リソース設定とファイル検索」を参照)。空文字列にすると、既存のすべての検索パス項目が削除されます。Windows では、検索パスはレジストリ項目を通じて設定することもできます。デフォルト: 空</p>
<i>shutdown-strategy</i>	<p>(整数) すべての TET オブジェクトに対して 1 回割り当てられるグローバルリソースの解放の方式。グローバルリソースはそれぞれ、それが初めて必要とされた時点で初期化されます。このオプションは 1 個のプロセス内のすべての TET オブジェクトに対して同じ値に設定する必要があります。そうでない場合は動作は未定義です (デフォルト: 0):</p> <ul style="list-style-type: none"> <li>o そのリソースを何個の PLOP オブジェクトが使用しているかを参照カウンタが追跡します。最後の TET オブジェクトが削除されて参照カウンタが 0 に落ちた時に、そのリソースは解放されます。</li> <li>1 リソースはそのプロセスの終わりまで保持されます。これはパフォーマンスを若干向上させる可能性があります。最後の TET オブジェクトが削除された後に必要なメモリが増加します。</li> </ul>

1. 複数回呼び出すとオプション値を蓄積できます。
2. UPR 文法とは異なり、名前と値の間の等号「=」は必須ではなく、許容もされません。



## 10.5.2 セットアップ

---

**C** *TET \*TET\_new(void)*

---

新規 TET オブジェクトを作成します。

**戻り値** 後続の呼び出しで使用する TET オブジェクトへのハンドル。メモリが得られなかったためにこの関数が成功しなかったときは NULL を返します。

**バインディング** オブジェクト指向言語バインディングでは、この関数は TET コンストラクタ内に隠蔽されていますので、利用できません。

---

**Java** *void delete()*

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

**C** *void TET\_delete(TET \*tet)*

---

TET オブジェクトを削除して、関連する内部リソースをすべて解放します。

**詳細** TET オブジェクトを削除すると、その開いている文書も自動的にすべて閉じられます。TET オブジェクトは、閉じられた後は、いかなる関数でも使用してはいけません。

**バインディング** オブジェクト指向言語バインディングでは、この関数は TET デストラクタ内に隠蔽されていますので、一般に必要ではありません。ただし Java では、自動的なガベージコレクションに加えて明示的なクリーンアップを可能にするために、利用可能となっています。.NET では、処理の最後に、非マネージのリソースをクリーンアップするために *Dispose()* を呼び出すべきです。

## 10.5.3 PDFlib 仮想ファイルシステム (PVF)

---

```
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 RB Sub create_pvf(filename As String, data, optlist As String)
C void TET_create_pvf(TET *tet,
    const char *filename, int len, const void *data, size_t size, const char *optlist)
```

---

メモリ内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、後続する TET 呼び出しにおいてこの仮想ファイルを参照するために使える任意の文字列です。

**len** (C 言語バインディングのみ) **filename** が UTF-16 文字列の場合の長さ (バイト単位で)。**len=0** ならば、ヌル終端文字列を与える必要があります。

**data** 仮想ファイルのためのデータへの参照。COM では、これは仮想ファイルを構成するデータを内容として持つ Byte の Variant です。C・C++ では、これはメモリ位置へのポインタです。Java では、これは byte 配列です。Perl・PHP では、これは文字列です。

**size** (C・C++ のみ) このデータを内容として持つメモリブロックの長さをバイト単位で。

**optlist** 表 10.3 に従ったオプションリスト。右記のオプションが使えます：**copy**

**詳細** 仮想ファイル名は、入力ファイル名をとる任意の API 関数に対して与えることができます。こうした関数のなかには、そのデータがもう必要なくなるまでその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、**TET\_delete\_pvf()** によって明示的に、あるいは **TET\_delete()** で自動的に削除されるまで、メモリ内に保持されます。

各 TET オブジェクトは、それぞれ自分自身の PVF ファイルのセットを保持します。仮想ファイルを、異なる TET オブジェクト間で共有することはできません。複数スレッドで個別の TET オブジェクトを操作している場合、PVF の使用を同期する必要はありません。**filename** で、すでにある仮想ファイルと同じ名前を指定すると例外が発生します。この関数では、**filename** がすでに通常のディスクファイルで使わ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かぎり、与えたデータを、それに対応する **TET\_delete\_pvf()** への呼び出しが成功する前に、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。この決まりに従わないとクラッシュするおそれが高いです。

表 10.3 TET\_create\_pvf() のオプション一覧

オプション	説明
<b>copy</b>	(論理値) TET は、与えられたデータの内部コピーをただちに作成します。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廃棄してもかまいません。この <b>copy</b> オプションは、COM・NET・Java バインディングでは自動的にに設定されます (それ以外のバインディングでのデフォルト：false)。それ以外のバインディングでは、 <b>copy</b> オプションを与えないかぎりデータは複製されません。

---

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

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

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

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

**C** *int TET\_delete\_pvf(TET \*tet, const char \*filename, int len)*

---

指名された仮想ファイルを削除し、そのデータ構造を解放します（ただしその内容は解放しません）。

**filename**（名前文字列）*TET\_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* が UTF-16 文字列の場合の長さ（バイト単位で）。*len=0* ならば、ヌル終端文字列を与える必要があります。

**戻り値** 指定された仮想ファイルが存在しているがロックされている場合は -1、それ以外の場合は 1。

**詳細** ファイルがロックされていなければ、TET はただちに、*filename* に関連づけられたデータ構造を削除します。*filename* という名前の有効な仮想ファイルが存在しない場合には、この関数は何も警告など出さずに終了します。この関数を呼び出して成功した後は、その *filename* は再利用することもできます。仮想ファイルはすべて、*TET\_delete()* で自動的に削除されます。

詳細な動作は、これに対応する *TET\_create\_pvf()* への呼び出しの際に *copy* オプションを与えていたかどうかによって依存します：*copy* オプションを与えていた場合には、ファイルの管理データ構造とファイル内容本体（データ）の両方が解放されますが、そうでなかった場合には、内容は解放されません。後者の場合にはクライアント側で内容を解放することが期待されています。

---

**C++** *int info\_pvf(wstring filename, wstring keyword)*

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

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

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

**C** *int TET\_info\_pvf(TET \*tet, const char \*filename, int len, const char \*keyword)*

---

仮想ファイルまたは PDFlib 仮想ファイルシステム (PVF) のプロパティを取得します。

**filename**（名前文字列）仮想ファイルの名前。*keyword=filecount* の場合この *filename* は空とすることができます。

**len**（C 言語バインディングのみ）*filename* が UTF-16 文字列の場合の長さ（バイト単位で）。*len=0* ならば、ヌル終端文字列を与える必要があります。

**keyword** 表 10.4 に従ったキーワード。

**詳細** この関数は、仮想ファイルまたは PDFlib 仮想ファイルシステム (PVF) のさまざまなプロパティを返します。プロパティをキーワードで指定します。

表 10.4 TET\_info\_pvf() のキーワード一覧

オプション	説明
<i>filecount</i>	カレント TET 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内のファイルの総数。filename 引数は無視されます。
<i>exists</i>	そのファイルが PDFlib 仮想ファイルシステム内に存在しているなら（かつ削除されていないなら）1、そうでないなら 0
<i>size</i>	（存在する仮想ファイルに対してのみ）指定した仮想ファイルのサイズをバイト単位で。
<i>iscopy</i>	（存在す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ていたなら 1、そうでないなら 0
<i>lockcount</i>	（存在す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 TET 関数によって内部的にセットされたロックの数。このロックカウントが 0 にならなければそのファイルは削除されることができません。

## 10.5.4 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 RB** `Function convert_to_unicode(inputformat As String, input As String, optlist As String) As String`

**C** `const char *TET_convert_to_unicode(TET *tet,  
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 テキストの各形式：`utf8`・`ebcdicutf8`・`utf16`・`utf16le`・`utf16be`・`utf32`
- ▶ すべての内部的に知られる 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` は右記の動作を指定します：入力文字列に UTF-8 か UTF-16 の BOM がある場合には、それを用いて正しい形式が決定され、そうでない場合はカレントシステムコードページと見なされます。

**input** Unicode へ変換したい文字列。

**input** Unicode へ変換したいデータを内容とするバリエーション (REALbasic ならメモリブロック)。

**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` ならば、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) を格納させたいメモリ位置への C スタイルポインタ。

**optlist** 表 10.5 に従ってオプションを指定したオプションリスト：

- ▶ 入力フィルタオプション：`charref`・`escapesequence`
- ▶ Unicode 変換オプション：`bom`・`errorpolicy`・`inflate`・`outputformat`

**戻り値** 指定し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、指定した入力形式に従っていない (無効な UTF-8 文字列など) ときは、`errorpolicy=return` であれば空の出力文字列が返され、`errorpolicy=exception` であれば例外が発生します。

**詳細** この関数は Unicode 文字列変換全般に有用でしょう。これは、適当な Unicode 変換機能を提供していない環境で作業するユーザの利便のために提供されています。

**バインディング** C バインディング：返される文字列は、最大 10 項目を持つリングバッファに格納されます。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 1 個の中にはこの関数への呼び出しを最大 10 個まで引数として入れることができます。10 個を超える文字列が同時に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++ バインディング : *inputformat* と *optlist* は通常通り *wstring* として渡す必要がありますが、*input* と戻りデータは *string* 型を持つ必要があります。

Python バインディング : UTF-8 の戻り値は文字列として返ります。Python 3 : 非 UTF-8 の戻り値はバイト列として返ります。

表 10.5 TET\_convert\_to\_unicode() のオプション一覧

オプション	説明
<i>charref</i>	(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
<i>bom</i>	(キーワード。outputformat=utf32 のときは無視されます) 出力文字列にバイト順序マーク (BOM) を付加する方式。使えるキーワード (デフォルト : none) : <i>add</i> BOM を付加。 <i>keep</i> 入力文字列に BOM があるなら BOM を付加。 <i>none</i> BOM を付加しない。 <i>optimize</i> outputformat=utf8 または ebcdicutf8 かつ出力文字列が範囲 < U+007F のキャラクターのみを含む場合を除いて BOM を付加。
<i>errorpolicy</i>	(キーワード) 変換エラーの際の動作 (デフォルト : exception) : <i>return</i> 文字参照が解決できないとき、または指定されたフォント内に内蔵コードかグリフ ID が存在しないとき、置換キャラクタ U+FFFD が使用されます。変換エラーの場合には空文字列が返されます。 <i>exception</i> 変換エラーの場合には例外が発生します。
<i>escape-sequence</i>	(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
<i>inflate</i>	(論理値。inputformat=utf8 の場合のみ。outputformat=utf8 のときは無視されます) true の場合、無効な UTF-8 入力文字列は例外を発生させず、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。これはデバッグに有用でしょう。デフォルト : false
<i>output-format</i>	(キーワード) 生成したい文字列の 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 : 出力形式は強制的に utf16 になります。 C++ 言語バインディング : 右記の出力形式のみ許されます : ebcdicutf8・utf8・utf16・utf32。

## 10.5.5 例外処理

---

**C++** *wstring get\_apiname()*

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

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

**VB RB** *Function get\_apiname() As String*

**C** *const char \*TET\_get\_apiname(TET \*tet)*

---

例外を発生させた、または失敗した API 関数の名前を得ます。

**戻り値** 例外を発生させた関数の名前、または、呼び出されてエラーコードを持って失敗した最近の関数の名前。エラーがなかった場合は空文字列が返されます。

---

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

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

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

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

**C** *const char \*TET\_get\_errmsg(TET \*tet)*

---

最後に発生した例外の、または失敗した関数呼び出しの原因のテキストを得ます。

**戻り値** 最後に発生した例外の説明を、または、呼び出されてエラーコードを持って失敗した最近の関数の原因を内容として持つテキスト。エラーがなかった場合は空文字列が返されず。

---

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

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

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

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

**C** *int TET\_get\_errnum(TET \*tet)*

---

最後に発生した例外の、または、失敗した関数呼び出しの原因の番号を得ます。

Get the number of the last thrown exception or the reason for a failed function call.

**戻り値** 例外の番号、または、呼び出されてエラーコードを持って失敗した最近の関数のエラーコード。エラーがなかった場合はこの関数は 0 を返します。

---

**C** *TET\_TRY(tet)*

**C** *TET\_CATCH(tet)*

**C** *TET\_RETHROW(tet)*

**C** *TET\_EXIT\_TRY(tet)*

---

例外処理ブロックをセットアップします。例外をキャッチします。例外を再び投げます。例外機構に対し、対応する *TET\_CATCH()* ブロックに入ることなく *TET\_TRY()* ブロックから

抜けることを通知します。`TET_RETHROW()` を用いると、例外をキャッチした後に、より高いレベルの関数へそれを再び投げることができます。

詳細 (C 言語バインディングのみ) 29 ページの 3.2 「C バインディング」を参照。



## 10.5.6 ログ記録

ログ記録機能を利用すると、API 呼び出し群を追跡することができます。そのログファイルの内容は、デバッグ目的に有用なほか、PDFlib GmbH サポートから求められることがあります。表 10.6 に、`TET_set_option()` (158 ページの 10.5.1 「オプション処理」を参照) でログ記録機能を有効にするためのオプションを挙げます。

表 10.6 `TET_set_option()` のログ記録関連キー一覧

キー	説明
<code>logging</code>	表 10.7 に従ったログ記録オプション群を持つオプションリスト
<code>userlog</code>	ログファイルへ複製される文字列

ログ記録オプションは、以下の方法で与えることができます：

- ▶ `TET_set_option()` の `logging` オプションに対するオプションリストとして。例：  
`tet.set_option("logging", "filename=debug.log remove")`
- ▶ `TETLOGGING` という環境変数で。この場合、ログ記録出力は、API 関数のいずれかを最初に呼び出した時から開始されます。

表 10.7 `TET_set_option()` の `logging` オプションのサブオプション一覧

キー	説明
(空リスト)	<code>disable</code> で無効化されているログ出力を有効にします。
<code>disable</code>	(論理値) ログ出力を無効化します。デフォルト : <code>false</code>
<code>enable</code>	(論理値) ログ出力を有効にします。
<code>filename</code>	(文字列) ログファイルの名前 ( <code>stdout</code> と <code>stderr</code> も受け付けます)。すでに内容があるとき、出力はその末尾に追加されます。ログファイル名はあるいは、 <code>TETLOGFILENAME</code> という環境変数で与えることもできます (この場合、このオプション <code>filename</code> はつねに無視されます)。デフォルト : <code>tet.log</code> (Windows と Mac では / ディレクトリ内。Unix では /tmp 内)
<code>flush</code>	(論理値) <code>true</code> の場合、ログファイルは、必ず出力が実際にフラッシュされるよう、出力のたびに閉じられ、次の出力の時にまた開かれます。プログラムのクラッシュを追跡する際に、ログファイルが途中で終わってしまっている場合に有用でしょう。ただし速度はかなり遅くなります。 <code>false</code> の場合、ログファイルは 1 回だけ開かれます。デフォルト : <code>false</code>
<code>remove</code>	(論理値) <code>true</code> の場合は、新しい出力を書きこむ際に、既存のログファイルは削除されます。デフォルト : <code>false</code>
<code>stringlimit</code>	(整数) テキスト文字列内のキャラクタの数の上限。0 なら無制限。デフォルト : 0

表 10.7 TET\_set\_option() の logging オプションのサブオプション一覧

キー	説明
<b>classes</b>	(オプションリスト) 整数型のオプション群を内容として持つリスト。ここで各オプションはログ記録クラスを記述し、おのおのの値はその粒度を記述します。レベル 0 はログ記録クラスを無効化し、正の数値はクラスを有効にします。レベルが上がるほど詳細な出力を与えます。以下のオプションが使えます (デフォルト: {api=1 warning=1}):
<b>api</b>	すべての API 呼び出しを、その引数と結果とともにログ記録します。api=2 の場合は、すべての API 追跡行の頭にタイムスタンプが生成されるとともに、非推奨の関数・オプションにはその旨注記されます。api=3 の場合は、try/catch 呼び出しがログ記録されます (入れ子の例外処理によるデバッグ問題に対して有用です)。
<b>filesearch</b>	SearchPath または PVF を通じたファイルの場所特定に関連するすべての試みをログ記録します。
<b>resource</b>	Windows レジストリ・UPR 定義を通じたリソースの場所特定のすべての試みを、そのリソース検索の結果とともにログ記録します。
<b>user</b>	userlog オプションで与えられたユーザ指定のログ記録出力。
<b>warning</b>	すべての警告を、すなわち、無視または内部修復できるエラー状況をログ記録します。warning=2 の場合は、例外を発生させずに、TET_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、ファイルを開こうとした試みの失敗すべての原因もログ記録されます。

## 10.6 文書関数

---

**C++** *int open\_document(wstring filename, wstring optlist)*

**C# Java** *int open\_document(String filename, String optlist)*

**Perl PHP** *long open\_document(string filename, string optlist)*

**VB RB** *Function open\_document(filename As String, optlist As String) As Long*

**C** *int TET\_open\_document(TET \*tet, const char \*filename, int len, const char \*optlist)*

---

内容を抽出したい、ディスクベースの、または仮想の PDF 文書を開きます。

**filename** (名前文字列) 処理させたい PDF 入力ファイルの絶対または相対名。このファイルは、*searchpath* リソースカテゴリ内で指定されているすべてのディレクトリ内で検索されます。Windows では、UNC パスと、マップされたネットワークドライブも使えます。PHP では、Unicode ファイル名は UTF-8 である必要があります。

非 Unicode 言語バインディングでは、**len = 0** の場合、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。ただし、UTF-8 BOM が頭に付いている場合は、UTF-8 か 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** が UTF-16 文字列の場合の長さ (バイト単位で)。**len=0** ならば、ヌル終端文字列を与える必要があります。

**optlist** 表 10.8 に従って文書オプション群を指定したオプションリスト。右記のオプションが使えます: *checkglyphlists* · *decompose* · *encodinghint* · *fold* · *glyphmapping* · *lineseparator* · *normalize* · *inmemory* · *password* · *repair* · *requiredmode* · *shrug* · *tetml* · *usehostfonts* · *wordseparator* · *zonesteparator*

**戻り値** エラー時は -1、そうでないなら文書ハンドル。たとえば、入力文書または TETML 出力ファイルを開くことができないときはエラーになります。-1 が返された場合には、*TET\_get\_errmsg()* を呼び出してエラーの詳細を知ることがを推奨します。

**詳細** 1 個の TET オブジェクト内で、任意の数の文書を同時に開いておくことができます。しかし、1 個の TET オブジェクトを複数のスレッドで同時に、アクセスを同期するロック機構なしで使用してはいけません。

暗号化: 文書が暗号化されている場合は、その権限設定が内容抽出を許しているならば、そのユーザパスワードを *password* オプションで与える必要があります。権限設定が内容抽出を許していない場合には、その文書のマスタパスワードを与える必要があります。*requiredmode* オプションを指定している場合は、正しいパスワードがなくても文書を開くことができますが、操作は制限されます。*shrug* オプションを用いると、保護された文書から一定の条件下で内容抽出を可能にすることができます (67 ページの 5.1 「暗号化 PDF から内容を抽出」を参照)。

i5/iSeries 上での対応ファイルシステム: TET は、PC タイプのファイルシステムでのみテストされています。ですので、入力・出力ファイルは、IFS (統合ファイルシステム) 内の PC タイプファイル内になければなりません。*QSYS.lib* ファイルシステム内の入力ファイルはテストされておらず、サポートされていません。*QSYS.lib* ファイルは多くの場合、レコードベースまたはデータベースオブジェクトのために用いられますので、TET を *QSYS.lib* オブジェクトとともに使用すると予期しない動作結果を招くおそれがあります。TET ファイル I/O ストリームはつねにストリームベースであり、レコードベースではありません。

表 10.8 TET\_open\_document()・TET\_open\_document\_callback()の文書オプション一覧

オプション	説明
<b>check-glyphlists</b>	(論理値) true ならば、TET は、テキスト抽出が始まる前に、すべての condition=allfonts の内蔵グリフマッピング規則をチェックします。そうでないなら、グローバルなグリフマッピング規則は適用されません。このオプションは処理速度を低下させますが、デフォルトで Unicode ヘマップできないグリフ名を持つある種の TeX 文書に対して有用です。デフォルト : false
<b>decompose</b>	<p>(キーワードかオプションリスト) 指定した Unicode 分解タグを持つ、かつ、指定した Unicode 集合の一部であるすべてのキャラクタに適用される Unicode 分解。これらの条件はサブオプションの名前と値で与えられます。分解を用いると、等価な Unicode キャラクタ間の違いを除去するか温存することができます (105 ページの 7.3 「Unicode 後処理」を参照)。</p> <p>デフォルト : 111 ページの「デフォルト分解」を参照してください。ただし、normalize オプションが none 以外の値を持つときは、すべてのデフォルト分解は無効化されます。すなわち、normalize オプションを設定するとデフォルトが decompose=none に設定されます。ただし、ユーザ指定の分解はなお適用することができます。</p> <p>リストに換えて、以下のキーワードを与えることもできます :</p> <p><b>none</b> 何の分解も適用されません。</p> <p><b>default</b> 他の指定した分解の前に、デフォルト分解 (111 ページの「デフォルト分解」を参照) が適用されます。</p> <p>分解のための以下のサブオプションが使えます :</p> <p><b>canonical · circle · compat · final · font · fraction · initial · isolated · medial · narrow · nobreak · small · square · sub · super · vertical · wide</b></p> <p>これらの各サブオプションは、分解のドメインを、すなわち、分解が適用される Unicode キャラクタの集合を指定する文字列またはキーワードを受け付けます。文字列は、ドメインの Unicode 集合を指定します。これを用いると、指定した分解タグを持つキャラクタの部分集合に分解を制限することができます。このドメインの外のキャラクタは変更されません。</p> <p>Unicode 集合の文字列に換えて、以下のキーワードを与えることもできます :</p> <p><b>_all</b> 全 Unicode キャラクタの集合。すなわち、分解は、指定した分解タグを持つすべてのキャラクタに適用されます。</p> <p><b>_none</b> 空集合。すなわち、分解はまったく適用されません。</p>
<b>encodinghint</b>	(文字列 <sup>1</sup> ) 標準規則ではマップできず、定義済内部グリフマッピング規則によってのみマップできるグリフ名に対する Unicode マッピングを決定するために用いられるエンコーディングの名前。キーワード none を用いると、すべての定義済規則を無効化することができます。デフォルト : winansi

表 10.8 TET\_open\_document()・TET\_open\_document\_callback()の文書オプション一覧

オプション	説明
<b>fold</b>	<p>(キーワード、またはリストのリスト。それぞれの内側のリストの1番目の要素はUnicode集合またはキーワード、2番目の要素はUnicharまたはキーワード)Unicode集合として指定した字形統合ドメイン内のすべてのキャラクタに対し、字形統合(等価マッピング)を適用します。この字形統合は、lineseparator または wordseparator オプションで追加した区切りキャラクタを除くすべてのテキストに対して適用されます(105ページの7.3「Unicode後処理」を参照)。デフォルト:107ページの表7.3を参照。</p> <p>リストに換えて、以下のキーワードを与えることもできます:</p> <p><b>none</b> 字形統合は一切適用されません。</p> <p>サブリストに換えて、以下のキーワードを与えることもできます:</p> <p><b>default</b> 他の指定した字形統合の前に、デフォルト字形統合が適用されます。</p> <p>各リストの1番目の要素は、字形統合のドメインを、すなわちその字形統合が適用されるUnicodeキャラクタの集合を指定します。文字列は、ドメインのUnicode集合を指定します。1個のキャラクタが、foldオプション内で指定している複数の集合に含まれている場合には、最初に一致した集合の定義が他のすべてに優先します。問題を避けるため、重ならない集合を用いることを推奨します。</p> <p>Unicode集合に換えて、以下のキーワードを与えることもできます:</p> <p><b>_dehyphenation</b></p> <p>字形統合は、改行位置でハイフン区切りされた単語内で見つかったハイフンキャラクタに適用されます。これらのキャラクタは、TET_get_char_info()が返すattributesメンバ内と、TETML内の@dehyphenation属性でフラグが立ちます。</p> <p>各リストの2番目の要素は、字形統合のターゲットキャラクタまたはアクションを内容として持ちます。これは以下の種類のいずれかで指定します:</p> <p><b>(Unichar)</b> そのドメイン内のすべてのキャラクタを、指定したUnicodeキャラクタへ置き換えます。</p> <p><b>remove</b> ドメイン内のすべてのキャラクタが除去されます。</p> <p><b>preserve</b> ドメイン内のキャラクタは変更されません。</p> <p><b>unknownchar</b></p> <p>ドメイン内のすべてのキャラクタを、unknowncharオプションで指定したキャラクタへ置き換えます。</p>
<b>glyphmapping</b>	<p>(オプションリストのリスト)オプションリストのリスト。ここで各オプションリストは、標準メソッドできちんとマップできない1個ないし複数のフォント/エンコーディングの組み合わせに対するグリフマッピング方式を記述します。このマッピングは、設定された順に用いられます。最後のオプションリストがフォント名ワイルドカード「*」を含んでいる場合には、それ以前のマッピングは用いられなくなります。各規則は、表10.9に従ったオプションリストを内容として持ちます。特定のフォント名にマッチするすべてのグリフマッピングがこのフォントに適用されます(デフォルト:定義済内蔵グリフ規則群が適用されます)。</p> <p>なお、グリフマッピング規則は、UPRファイル内の外部リソースとして指定することもできます(69ページの5.2「リソース設定とファイル検索」を参照)。</p>
<b>lineseparator</b>	<p>(Unichar. granularity=zone・pageの場合のみ)行どうしの間に挿入させたいキャラクタ<sup>2</sup>。デフォルト:U+000A</p>
<b>normalize</b>	<p>(キーワード)テキスト出力を、Unicode正規化形の1つへ正規化します:</p> <p><b>none</b> いかなる正規化も適用しません。</p> <p><b>nfc</b> 正規化形C(NFC):正準分解の後に正準合成</p> <p><b>nfd</b> 正規化形D(NFD):正準分解</p> <p><b>nfkc</b> 正規化形KC(NFKC):互換分解の後に正準合成</p> <p><b>nfkd</b> 正規化形KD(NFKD):互換分解</p> <p>Unicode正規化形には正準分解と互換分解がかかってくるので、オプションdecomposeとnormalizeの組み合わせは注意深く行う必要があります。normalizeオプションをnone以外の値に設定すると、分解のデフォルトはdecompose=noneに設定されます。normalizeオプションはdecomposeオプションの後に処理されます。</p>

表 10.8 TET\_open\_document()・TET\_open\_document\_callback()の文書オプション一覧

オプション	説明
<b>inmemory</b>	(論理値。TET_open_document()のみ) true の場合、TET はファイル全体をメモリ内へ読み込み、それをそこで処理します。これは、システムによっては驚異的なパフォーマンス向上につながるがありますが(とくに MVS)、そのかわりメモリ使用量も増えます。false の場合、文書の個々の部分が必要に応じてディスクから読み込まれます。デフォルト : false
<b>password</b>	(文字列) 暗号化された文書に対するユーザパスワード・マスタパスワード・添付パスワードのいずれか。その文書の権限設定がテキストコピーを許している場合には、ユーザパスワードで充分ですが、そうでない場合はマスタパスワードを与える必要があります。 文書の暗号化ステータスを取得する方法と、ユーザまたはマスタパスワードを知らなくても適用できる pCOS 操作については、pCOS パスリファレンスを参照してください。 shrug オプションを用いると、保護された文書から一定の条件下で内容抽出を可能にすることができます(67 ページの 5.1「暗号化 PDF から内容を抽出」を参照)。
<b>repair</b>	(キーワード) 破損 PDF 文書の扱い方を指定します。文書の修復は、通常のパースよりも時間がかかりますが、ある種の破損 PDF が処理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場合もあります(デフォルト : auto) : <b>force</b> 文書に問題があってもなくても、無条件に文書の修復を試みます。 <b>auto</b> PDF を開く際に問題が検出されたときのみ文書を修復します。 <b>none</b> 文書の修復の試みは一切行われません。PDF 内に問題があるときは、関数呼び出しは失敗します。
<b>requiredmode</b>	(キーワード) 文書を開く際に受け入れられる最小 pCOS モード(最小 : minimum/ 制限 : restricted/ フル : full)。結果の pCOS モード(pCOS パスリファレンス参照)が、求めたモードよりも低くなる場合は呼び出しは失敗します。呼び出しが成功したときは、結果の pCOS モードが最低で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、それより高くなる場合もあります。たとえば、暗号化されていない文書に対して requiredmode=minimum を指定すると、結果はフルモードになります。デフォルト : full
<b>shrug</b>	(論理値) true の場合、シュラッグ機能が有効になり、保護された文書から一定の条件下で内容抽出が可能になります(67 ページの 5.1「暗号化 PDF から内容を抽出」を参照)。shrug オプションを用いる場合には、その PDF 文書作成者の権利を尊重してください。デフォルト : false
<b>tetml</b>	(オプションリスト) TETML 出力が開始され、そして TET_process_page() でページごとに作成できます。以下のサブオプションが使えます : <b>elements</b> (論理値のリスト) 出力内に特定の TETML 要素が含ませるかどうかを指定します(デフォルト : すべて true) : <b>docinfo</b> /TET/Document/DocInfo 要素 <b>docxmp</b> /TET/Document/Metadata 要素 <b>options</b> 要素 /TET/Document/Options と /TET/Document/Pages/Page/Options <b>encodingname</b> (キーワード) 生成される TETML のテキスト宣言の XML エンコーディング宣言内で用いたい名前。出力はつねに UTF-8 で生成されます(デフォルト : UTF-8) : <b>_none</b> エンコーディング宣言は作成されません。出力は同様に UTF-8 形式になります。 <b>UTF-8</b> 宣言 encoding="UTF-8" が作成されます。 これ以外のエンコーディング名は、エンコーディング宣言内にリテラルに用いられません。適切なエンコーディング名を与えるとともに、TET が TETML 出力を完成させた後に、生成された TETML (UTF-8 です) を、指定したエンコーディングへ変換することは、クライアント側の役割です。 <b>filename</b> (文字列) TETML ファイルの名前。filename を与えなかった場合には、出力はメモリ内に生成され、TET_get_xml_data() で取得することができます。関数呼び出しが失敗した(すなわち、PDF 入力文書をうまく開くことができなかった)場合には、TETML 出力は生成されません。

表 10.8 TET\_open\_document()・TET\_open\_document\_callback()の文書オプション一覧

オプション	説明
<b>unknown-char</b>	(Unichar) PDF 文書内の情報が整合しない、または存在しないために、Unicode ヘマッピングできないキャラクタに対する置き換えとして用いたいキャラクタ。U+0000 にすると、未知キャラクタは除去されます。デフォルト : U+FFFD (置換キャラクタ) 関連するオプション : 未知の (PUA) キャラクタをも、指定した unknownchar へ置き換えたいときは、fold={{[:Private_Use:] unknownchar}} を用います。それらを除去するには fold={{[:Private_Use:] remove}} を用います。
<b>usehostfonts</b>	(論理値) true の場合、埋めこまれていないが Unicode マッピングを決定するために必要なフォントのデータは、Mac または Windows ホストオペレーティングシステム上で検索されます。デフォルト : true
<b>wordseparator</b>	(Unichar. granularity=line・page の場合のみ) 単語どうしの間に挿入させたいキャラクタ <sup>2</sup> 。デフォルト : U+0020

1. 表 10.9 の脚注 2 を参照
2. 区切りキャラクタを無効化するには U+0000 を用います。

表 10.9 TET\_open\_document()・TET\_open\_document\_callback()の glyphmapping オプションのサブオプション一覧

オプション	説明
<b>codelist</b>	(文字列) フォントに適用させたいコードリソースの名前。これは、埋めこまれた ToUnicode CMap またはエンコーディング項目よりも優先されます。
<b>fontname</b>	(名前文字列) 規則に対して選択されるフォント (群) の名前の一部分ないし全体。部分集合接頭辞を与えているときは、指定した部分集合だけが選択されます。部分集合接頭辞を一切与えていないときは、名前 (部分集合接頭辞なし) がマッチするすべてのフォントが選択されます。限られたワイルドカード <sup>1</sup> が使えます。デフォルト : *
<b>fonttype</b>	(キーワードのリスト) グリフマッピングは、指定した種類のフォントに対してのみ適用されます : * (あらゆる種類のフォントを意味します)・Type1・MMType1・TrueType・CIDFontType2・CIDFontType0・Type3。デフォルト : *
<b>force-encoding</b>	(文字列 <sup>2</sup> 1 個か 2 個のリスト。名前 2 個があるときは、1 番目は winansi・macroman・Custom のいずれかでなければなりません) 8 ビットエンコーディングのフォント : 1 番目のエンコーディングを、2 番目の名前で指定したエンコーディングリソースへ置き換えます。項目が 1 個しか与えられていないときは、指定されたエンコーディングを用いて MacRoman・WinAnsi・MacExpert エンコーディングのすべてのインスタンスが置き換えられます。このオプションがフォントにマッチした場合には、その同じフォントに対しては他のグリフマッピングは一切適用されません。 CID フォント : 値 1 個 unicode にのみ対応しています。これは CID 値を Unicode 値として解釈します。
<b>forcetsymbol-encoding</b>	(キーワードまたは文字列 <sup>2</sup> ) 埋め込まれた、実際にはテキストフォントである擬似 TrueType 記号フォントに対する Unicode マッピングを決定するために使われるエンコーディングの名前か、または以下のキーワードの 1 つ (デフォルト : auto) : <b>auto</b> フォントの内蔵エンコーディング (後述) が、記号範囲 U+F0000 ~ U+F0FFF 内の Unicode キャラクタを少なくとも 1 個含んでいる場合には、encodinghint オプションで指定したエンコーディングを用いて擬似記号キャラクタが本当のテキストキャラクタへマップされます。そうでない場合には、encodinghint は使われず、キャラクタは builtin キーワードに従ってマップされます。 <b>builtin</b> フォントの post テーブル内のグリフ名の Unicode マッピングから得られる、フォントの内蔵エンコーディングを用います。 有名な TrueType フォント Wingdings* と Webdings* はつねに記号フォントとして扱われます。

表 10.9 TET\_open\_document()・TET\_open\_document\_callback() の glyphmapping オプションのサブオプション一覧

オプション	説明
<b>globalglyphlist</b>	(論理値) true の場合は、指定したグリフリストは TET オブジェクトの終了までメモリ内に保持されますので、複数の文書に対して適用することもできます。デフォルト: false
<b>glyphlist</b>	(文字列) 適用したいグリフリストリソースの名前
<b>glyphrule</b>	(オプションリスト) 数値グリフ名に対するマッピング規則 (定義済規則に加えて)。このオプションリストは以下のサブオプションを内容として持つ必要があります:
<b>prefix</b>	(文字列。空でも可) 規則が適用されるグリフ名の接頭辞。
<b>base</b>	(キーワード) グリフ名の解釈を指定します:
<b>ascii</b>	シングルバイトグリフ名はおのおのリテラル ASCII キャラクタとして解釈されます (例: 1 は U+0031 へマップされます)。
<b>auto</b>	グリフ名が 10 進と 16 進のどちらの値を表すのかを自動的に決定します。結果が一意でない場合には、10 進と見なされます。
<b>dec</b>	グリフ名はコードの 10 進表現として解釈されます。
<b>hex</b>	グリフ名はコードの 16 進表現として解釈されます。
<b>encoding</b>	(文字列) この規則に対して用いられるエンコーディングリソースの名前。キーワード none を指定すると、規則は無効になります。
<b>ignoreto-unicodemap</b>	(論理値) true の場合には、フォントに対する ToUnicode CMap は無視されます。デフォルト: false
<b>override</b>	(論理値。glyphlist または glyphrule オプションとともに用いてのみ意味を持ちます) true の場合には、グリフマッピング規則は、標準 (内蔵) グリフのマッピングより前に適用されます (すなわち、新しいマッピングが内蔵のものより優先されます)。そうでない場合はその逆です。デフォルト: true
<b>remove</b>	(論理値) true の場合、抽出されたテキストから、指定したフォント名 (複数可) および/またはフォント種別 (複数可) を用いているテキストはすべて除去されます。
<b>tounicode-cmap</b>	(文字列) フォントに適用させたい ToUnicode CMap リソースの名前。これは、埋め込まれた ToUnicode CMap またはエンコーディング項目よりも優先されます。

- 限られたワイルドカード: 単独キャラクタ「\*」は全フォントを意味します。接頭辞の後に「\*」を付けると (「MSTT\*」など)、指定した接頭辞で始まるすべてのフォントを意味します。
- 右記の定義済エンコーディング名は、追加設定なしで使えます: winansi・macroman・macroman\_apple・macroman\_euro・ebcdic・ebcdic\_37・iso8859-X・cpXXXX・U+XXXX。カスタムエンコーディングはリソースとして定義できます。

```

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 TET_open_document_callback(TET *tet, void *opaque, size_t filesize,
    size_t (*readproc)(void *opaque, void *buffer, size_t size),
    int (*seekproc)(void *opaque, long offset),
    const char *optlist)

```

内容抽出したい PDF 文書を、カスタムデータソースから開きます。

**opaque** 入力 PDF 文書に関連づけた何らかのユーザデータへのポインタ。このポインタは、コールバック関数群の 1 番目の引数として渡され、いかようにでも使うことができます。TET はこの不透明ポインタをいかなる形においても使用しません。

**filesize** PDF 文書全体のサイズをバイト単位で。



**readproc** *size* バイトを *buffer* で指し示されたメモリへ複製する C コールバック関数。文書の終わりに到達した場合には、求められたよりも少ないデータを複製することができます。この関数は、複製したバイト数を返す必要があります。

**seekproc** 文書内のカレント読み取り位置を設定する C コールバック関数。**offset** は文書の先頭からの位置を表します (0 を最初のバイトとして)。この関数は、成功したときには 0 を、そうでなかったときには -1 を返す必要があります。

**optlist** 表 10.8 に従って文書オプション群を指定したオプションリスト。

戻り値 *TET\_open\_document()* 参照。

詳細 *TET\_open\_document()* 参照。

バインディング この関数は、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)`

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

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

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

**C** `void TET_close_document(TET *tet, int doc)`

---

文書ハンドルと、その文書に関連づけられたすべての内部リソースを解放します。

**doc** *TET\_open\_document\*()* で得られた有効な文書ハンドル。

詳細 文書を閉じると、その開いているページ群もすべて自動的に閉じられます。*TET\_delete()* を呼び出すと、開いている文書とページはすべて自動的に閉じられます。とはいえ、文書が必要なくなった時点で明示的に閉じるのが良いプログラミング習慣です。閉じられた文書ハンドルは、その後はいかなる関数呼び出しにおいても使用してはいけません。

## 10.7 ページ関数

---

**C++** *int open\_page(int doc, int pagenumber, wstring optlist)*

**C# Java** *int open\_page(int doc, int pagenumber, String optlist)*

**Perl PHP** *long open\_page(long pagenumber, string optlist)*

**VB RB** *Function open\_page(doc As Long, pagenumber As Long, optlist As String) As Long*

**C** *int TET\_open\_page(TET \*tet, int doc, int pagenumber, const char \*optlist)*

---

内容抽出したいページを開きます。

**doc** *TET\_open\_document\*()* で得られた有効な文書ハンドル。

**pagenumber** 開きたいページの物理的番号。最初のページをページ番号 1 とします。総ページ数は *TET\_pcos\_get\_number()* と pCOS パス *length:pages* で取得できます。

**optlist** 表 10.10 に従ってページオプション群を指定したオプションリスト。右記のオプションが使えます：*clippingarea*・*contentanalysis*・*docstyle*・*excludebox*・*fontsize**range*・*granularity*・*ideographic*・*ignoreinvisibletext*・*imageanalysis*・*includebox*・*layoutanalysis*・*layouteffort*・*skipengines*・*structureanalysis*・*topdown*。

**戻り値** ページのハンドル。エラーの場合には -1。-1 が返された場合には、*TET\_get\_errmsg()* を呼び出してエラーの詳細を知ることが推奨します。

**詳細** 1 個の文書内で任意の数のページを同時に開いておくことができます。同じページを異なるオプションで複数回開くこともできます。しかし、1 つのページを処理している途中でオプションを変えることはできません。

ページ内にレイヤー定義 (オプション的な内容グループ) があっても考慮されません。レイヤーが可視にされているかどうかにかかわらず、ページ上のすべてのレイヤー上のすべてのテキストが抽出されます。

表 10.10 TET\_open\_page()・TET\_process\_page()のページオプション一覧

オプション	説明
<b>clippingarea</b>	(キーワード。includeboxを指定しているとは無視されます) テキストが抽出される領域を指定します (デフォルト : cropbox) : <b>mediabox</b> MediaBox を用います (これは必ず存在します) <b>cropbox</b> CropBox を用います (Acrobat で表示される領域です)。なければ MediaBox を用います <b>bleedbox</b> BleedBox を用います。なければ CropBox を用います <b>trimbox</b> TrimBox を用います。なければ CropBox を用います <b>artbox</b> ArtBox を用います。なければ CropBox を用います <b>unlimited</b> 位置にかかわらずすべてのテキストを考慮します。
<b>content-analysis</b>	(オプションリスト。granularity=glyphの場合には不可) 表 10.11 に従った、高レベル内容分析とテキスト処理のためのサブオプション群のリスト。
<b>docstyle</b>	(キーワード) レイアウト検出エンジンがさまざまなパラメタを選択するために用いるヒント。これらのパラメタは、文書が以下の分類の1つに属する状況においてレイアウト分析を最適化します : 文書がこれらの分類の1つにあてはまることがわかっている場合には、このオプションに適切な値を与えればレイアウト検出結果は非常に向上します。このオプションは高度なレイアウト認識を有効にします (デフォルト : none) : <b>book</b> 典型的な本 <b>business</b> ビジネス文書 <b>fancy</b> 複雑なレイアウトによる装飾的なページ <b>forms</b> 構造化されたフォーム <b>generic</b> 特に特徴を指定しない最も一般的な文書分類です。 <b>magazines</b> 雑誌記事 <b>none</b> 特定の文書スタイルがわかっておらず、高度なレイアウト認識は無効にされます。 <b>papers</b> 新聞 <b>science</b> 科学記事 <b>searchengine</b> アプリケーションは検索エンジンインデックス生成機能やそれに類似のアプリケーションであり、ページの単語一覧をなるべく速く取得することが主たる関心。表組・ページ構造認識は無効化されます。 <b>spacegrid</b> 視覚レイアウトがスペースキャラクタ群を用いて生成される、リスト指向のレポート (メインフレームシステムで生成されることが多い)。この種の文書に対しては、影付き検出や洗練された単語境界検出といった多くのヒューリスティックは必要ないので、このオプションによってテキスト抽出を高速化できます。
<b>excludebox</b>	(矩形のリスト) 指定した矩形を合わせた領域を、テキスト抽出から除外します。デフォルト : 空
<b>fontsize-range</b>	(float 2 個のリスト) テキストの最小・最大文字サイズを指定した 2 個の数値。この区間を外れたサイズのテキストは無視されます。最大値にキーワード unlimited を指定すると、上限なしという意味になります。デフォルト : { 0 unlimited }
<b>granularity</b>	(キーワード) TET_get_text() が返すテキスト断片の粒度。glyph 以外のすべてのモードで単語検出機能が有効になります。詳しくは 92 ページの「テキストの粒度」を参照してください (デフォルト : word) : <b>glyph</b> 各断片はそれぞれ 1 個のグリフをマップした結果を内容として持ちますが、それが複数のキャラクタになる場合もあります (合字の場合など)。 <b>word</b> 各断片はそれぞれ、単語検出機能によって決定された 1 個の単語を内容として持ちます。 <b>line</b> 各断片はそれぞれテキスト 1 行を、ないしはそれにできるだけ似たものを内容として持ちます。連続する 2 個の単語の間には単語区切りキャラクタが挿入されます。 <b>page</b> 各断片はそれぞれ 1 個のページを内容として持ちます。単語・行・区域区切りキャラクタが適宜挿入されます。

表 10.10 TET\_open\_page()・TET\_process\_page() のページオプション一覧

オプション	説明
<b>ideographic</b>	(キーワード) 表意文字キャラクタ群に対する単語境界検出を制御します。このオプションは互換上の理由からデフォルトは split ですが、keep に設定することを推奨します (デフォルト : split)。 <b>keep</b> 表意文字キャラクタは一般に単語境界を構成しません。ただし句読点、および表意文字キャラクタと非表意文字の間の切り替わりは、単語境界を構成します。 granularity=word の場合には、表意文字読点 U+3001 と表意文字句点 U+3002 も単語境界を構成します。granularity=page の場合には、行末に改行は挿入されません。 <b>split</b> 表意文字キャラクタはつねに単語境界を構成します。
<b>ignore-invisibletext</b>	(論理値) true の場合には、表現モード 3 (不可視) のテキストは無視されます。デフォルト : false (不可視テキストは、スキャンされたページとその OCR テキストを内容として持つ画像 + テキスト PDF で主に使用されているからです)
<b>image-analysis</b>	(オプションリスト) 表 10.13 に従った、高レベル画像処理を制御するためのサブオプション群のリスト。
<b>includebox</b>	(矩形のリスト) テキスト抽出を、指定した矩形を合わせた領域に限りませす。デフォルト : 切り抜き領域全体
<b>layout-analysis</b>	(オプションリスト。granularity=glyph の場合には不可) 表 10.12 に従った、レイアウト検出機能を制御するためのサブオプション群のリスト。
<b>layouteffort</b>	(キーワード) レイアウト認識の品質 / パフォーマンスのトレードオフを制御します。レイアウト認識は、努力を増せば向上できますが、これによって操作は遅くなるおそれがあります。このレイアウト認識努力を、キーワード none・low・medium・high・extra で制御できます。デフォルト : low
<b>layouthint</b>	(オプションリスト) 特定のページレイアウト要素の存在についてレイアウト認識エンジンに通知します : <b>subsummary</b> (キーワード) サブサマリ (傍注) の存在について、指定によってはその位置とともにエンジンに知らせます。使えるキーワード (デフォルト : none) : <b>auto</b> サブサマリ検出なし。 <b>left</b> ページの左脇にサブサマリ検出を試みます。 <b>none</b> 自動的にサブサマリ検出を試みます。 <b>right</b> ページの右脇にサブサマリ検出を試みます。 <b>header</b> (論理値) true の場合、エンジンはページヘッダの検出を試みます (デフォルト : false)。 <b>footer</b> (論理値) true の場合、エンジンはページフッタの検出を試みます (デフォルト : false)。

表 10.10 TET\_open\_page()・TET\_process\_page() のページオプション一覧

オプション	説明
<b>skipengines</b>	(キーワードのリスト) ページ内容に対して利用可能なパーサのいくつかをスキップします。スキップされたエンジンはこのページに対して何のデータも返しません。必要でないエンジンをスキップすると、このエンジンが出すデータを必要としないアプリケーションにおいてパフォーマンスが向上します (デフォルト: すべてのエンジンが有効): <b>text</b> (キーワード) テキスト抽出エンジンをスキップします。 <b>image</b> (キーワード) 画像抽出エンジンをスキップします。
<b>structure-analysis</b>	(オプションリスト。granularity=glyph の場合には不可) 表 10.14 に従った、ページ構造分析を制御するためのサブオプション群のリスト。
<b>topdown</b>	(オプションリスト) 可視ページの左上隅に原点を持つ、y 座標が下方増加する座標系を指定します。そうでないなら、左下隅に原点を持つデフォルト座標系が用いられます。下向き座標を有効にすると、Acrobat で表示される座標系と同じにすることができます。使えるサブオプション: <b>input</b> (論理値) true の場合には、以下の項目に対して座標が有効になります (デフォルト: false): ページオプション includebox・excludebox <b>output</b> (論理値) true の場合には、以下の項目に対して座標が有効になります (デフォルト: false): TET_char_info: y・alpha・beta TET_image_info: y・alpha・beta TETML: Glyph/@y・Glyph/@alpha・Glyph/@beta・Box/@lly・Box/@ury・PlacedImage/@y・PlacedImage/@alpha・PlacedImage/@beta

表 10.11 TET\_open\_page()・TET\_process\_page() の contentanalysis オプションのサブオプション一覧

オプション	説明
<b>bidl</b>	(キーワード。granularity=glyph の場合には無視されます。右書きキャラクタがページ上に存在している場合にのみ効力を持ちます) 断片内の右書き・左書きテキストを並べ替える反転双方向アルゴリズムを制御します (デフォルト: logical): <b>visual</b> 断片内の右書き・左書きキャラクタを視覚順に保ちます。すなわち、反転双方向アルゴリズムを適用しません。 <b>logical</b> 反転双方向アルゴリズムを適用して、断片内のキャラクタを論理順でもたらしめます。
<b>bidilevel</b>	(キーワード) 反転双方向アルゴリズムに対するページのベースレベル (すなわちテキスト進行の主要向き) を指定します (デフォルト: auto): <b>auto</b> テキスト進行の主要向きを、内容に基づいてヒューリスティックに決定します。 <b>ltr</b> テキスト進行の主要向きとして左書きを前提します (欧文文書など) <b>rtl</b> テキスト進行の主要向きとして右書きを前提します (ヘブライ文字・アラビア文字文書など)
<b>dehyphenate</b>	(論理値) true の場合には、ハイフン区切りされた単語が特定され、ハイフンを挟むテキスト断片どうしが連結されます。ハイフン自体は keeplyphens オプションに従って扱われます。デフォルト: true
<b>dropcapsize</b>	(float) 大きなグリフがドロップキャップとして認識される最小サイズ。ドロップキャップは、区域の先頭の大きなキャラクタで、数行にわたってぶら下がっているものです。それらは、その区域の残りとして連結され、その区域内の最初の単語の一部を成します。デフォルト: 35
<b>dropcapratio</b>	(float) ドロップキャップと隣接テキストの文字サイズの最小比率。大きなキャラクタは、そのサイズが dropcapsize を超えており、かつ、その文字サイズ比率が dropcapratio を超えている場合にドロップキャップとして認識されます。言い換えれば、これはドロップキャップがわたるテキスト行数です。デフォルト: 4 (ドロップキャップは 3 行にわたるものが非常に多いですが、行間の分も考慮に入れる必要があります)

表 10.11 TET\_open\_page()・TET\_process\_page() の contentanalysis オプションのサブオプション一覧

オプション	説明
<b>includebox-order</b>	<p>(整数) 複数の包含枠を与えているとき (オプション includebox 参照)、このオプションは、枠の順序が単語検出機能に対しどのような効力を持つかを制御します (デフォルト: 0):</p> <ul style="list-style-type: none"> <li>0 ページ内容を分析する際に、包含枠順序は無視します。結果は、あたかも包含枠群の外のテキストがすべて削除されたのと同じになります。これは、ほしくないテキスト (ヘッダ・フッタなど) を除去しつつ、単語検出機能には何の影響も与えないようにするために有用です。</li> <li>1 包含枠順序を、単語と区域を生成する際には考慮に入れますが、区域順序を決定する際には考慮しません。1 個の単語が複数の枠に属することは決してありません。できた区域群は、論理順に並べ替わります。枠が重なりあっている場合には、そのテキストはリスト内で最も先に現れた枠に属します。それ以外では、オプションリスト内における包含枠の順序は考慮されません。この設定は、テキストをフォームから抽出する時や、テキストを表組から抽出する時や、複雑なレイアウトにおいて包含枠が重なりあっている場合に有用です。</li> <li>2 包含枠順序を、すべての操作について考慮します。各包含枠の内容は他の枠とは独立に扱われ、できたテキストは包含枠の順序に従って連結されます。これは、フォームからテキストを特定の順序で抽出したい時や、雑誌レイアウト内の記事段組を定義済順序で抽出したい時に有用です。こうした場合には、包含枠を適切な順序で指定するために、そのページレイアウトに関する事前の知識が必要です。</li> </ul>
<b>keeplyphen-glyphs</b>	<p>(論理値) true かつ dehyphenate=true の場合には、ハイフン除去された単語の各部分の間のハイフングリフは、TET_get_char_info() が返すグリフのリスト内と TETML 内の Glyph 要素で温存されます。これは、ページ上のテキストを正確に置き換えたいなど、ハイフンの位置について詳しい情報を必要とするアプリケーションにおいて有用です。これは fold={[_dehyphenation remove]} とは異なることに留意してください。後者は、get_text() が返す論理テキストからハイフンを除去するだけであり、グリフに対しては効力を持ちません。デフォルト: false</p>
<b>linespacing</b>	<p>(キーワード) 段落内のテキスト行どうしの間典型的な縦間隔を指定します: small・medium・large のいずれかです (デフォルト: medium)</p>
<b>maxwords</b>	<p>(整数かキーワード) ページ上の単語の数が、指定した数以下のときは (キーワード unlimited を指定すると上限なしになります)、ページ上で検出された区域群は適切に連結され並べ替えられます。ページ上の単語の数が、指定した数を超過しているときは、区域は一切作成されず、単語群はページ内容の読み順で抽出されます。後者の場合には処理はより速くなりますが、抽出される単語群の順序は最適ではなくおそれがあります。新聞のような、多くの単語を含む大きなページに対しては、このオプションを unlimited に設定することを推奨します。デフォルト: 5000</p>
<b>merge</b>	<p>(整数) ストリップと区域の連結を制御します (デフォルト: 2):</p> <ul style="list-style-type: none"> <li>0 ストリップ作成後の連結なし。これはかなり処理速度を向上させますが、最適未満の出力を生成するおそれがあるほか、影によっては正しく検出されないおそれがあります。</li> <li>1 単純な、ストリップを区域へ入れ込む連結: ストリップは、それが区域に重なっており、それでいて、次のストリップ以外のストリップと重なり合っていない場合に (影なしの場合における区域の重なり合いを避けるため)、この区域内へ連結されます。</li> <li>2 順序破りなテキストのための高度な区域連結: merge=1 に加えて、複数の重なり合う区域は、両区域のテキスト内容が重なり合わないならば、結合されて 1 個の区域になります。</li> </ul>
<b>numeric-entities</b>	<p>(キーワード) 数値・分数・時刻のような数値実体に対する単語境界検出を制御します (デフォルト: keep):</p> <ul style="list-style-type: none"> <li><b>split</b> その実体を、punctuationbreaks サブオプションに従って分割します。</li> <li><b>keep</b> その実体を、単語まるごととして温存します。</li> </ul>

表 10.11 TET\_open\_page()・TET\_process\_page() の contentanalysis オプションのサブオプション一覧

オプション	説明
<b>shadow-detect</b>	(論理値) true の場合には、影付きや偽ボールドテキストを作り出している、重なり合うテキスト断片群の冗長なインスタンスは検出され除去されます。デフォルト : true
<b>punctuation-breaks</b>	(論理値。granularity=word の場合のみ) true の場合には、文字のそばに配置されている約物キャラクタは単語境界として扱われ、そうでない場合にはそれは隣接する単語内へ含まれます。たとえば、このオプションは URL とメールアドレスの扱いに影響を与えます。デフォルト : true
<b>superscript</b>	(整数) 下付き・上付き検出を制御します (デフォルト : 2) : <b>0</b> 下付き・上付き検出なし <b>1</b> 単純な下付き・上付き検出 <b>2</b> 下付き・上付き検出のための高度なアルゴリズム

表 10.12 TET\_open\_page()・TET\_process\_page() の layoutanalysis オプションのサブオプション一覧

オプション	説明
<b>layout-astable</b>	(論理値) true の場合には、レイアウト認識エンジンはページ上の区域群を 1 個ないし複数の表組として扱います。連なりが表組と見なされるために必要な最小の列数は、文書のスタイルに依存します。false の場合には、スーパー表組認識は無効化されます (デフォルト : true)。
<b>layout-columnhint</b>	(キーワード) このオプションは、複雑なレイアウトにおける区域読み順検出を向上させる可能性があります。使えるキーワード (デフォルト : multicolumn) : <b>multicolumn</b> ページは多段組テキストを含んでいます。区域は段組から段組へ並べ替えされます。 <b>none</b> ヒントは何も得られません。区域順序はページ内容順序によって決定されます。 <b>singlecolumn</b> ページは 1 段組テキストを含んでいます。区域は行から行へ並べ替えされます。
<b>layoutdetect</b>	(整数) 再帰的レイアウト認識の深度を指定します (デフォルト : 1) : <b>0</b> レイアウト認識なし。 <b>1</b> ページ全体に対するレイアウト認識。ほとんどの文書がこれで充分です。 <b>2</b> レベル 1 の結果に対するレイアウト認識。これは、さまざまな多段組副レイアウトを持つレイアウトや、ページ上のさまざまな箇所タイトルがあるレイアウトや、複数段落表組に対して必要です。 <b>3</b> レベル 2 の結果に対するレイアウト認識。これは非常に複雑なレイアウトに対してのみ必要です。

表 10.12 TET\_open\_page()・TET\_process\_page()の layoutanalysis オプションのサブオプション一覧

オプション	説明
<b>layoutrow-hint</b>	(オプションリスト) レイアウト行処理を制御します。使えるオプション (デフォルト : none) : <b>full</b> レイアウト行処理を有効にします。 <b>none</b> レイアウト行処理を無効化します。 <b>separation</b> (キーワード) レイアウト行処理を有効にしますが、レイアウト認識がスーパー表組と推測した場合には無効化します。以下のサブオプションが使えます : <b>preservecolumns</b> 区域どうしの中の視覚関係に基づいて縦の段組を保つことを試みます。段組内の区域どうしが大きく引き離されている場合 (画像がはさまっているなどして) にはこれを推奨します。 <b>thick</b> 隣り合う区域どうしを連結して、それを同じレイアウト行内に配置しようと試みます。これによって、レイアウト行の数は少なくなり、一つ一つは大きくなります。段組内の段落どうしが互いに文字サイズよりも引き離されている雑誌や新聞のような複雑なレイアウトや、いくつかの多段組記事が縦に並んでいるレイアウトにはこれを推奨します。 <b>thin</b> 隣り合う区域どうしを引き離し、それらを別々のレイアウト行に配置しようと試みます。これによって、レイアウト行の数は多くなり、一つ一つは小さくなります。 例 : layoutanalysis = {layoutrowhint={full separation=thick}}
<b>mergetables</b>	(整数) 表行 1 個だけの表組は、表組認識中にスキップされ、通常の区域として扱われます。2 個の連続する区域が表組である場合には (表行 1 個だけであっても)、それらは結合することもできません (デフォルト : none) : <b>down</b> 下方へのみ結合します。 <b>none</b> 連結しません。 <b>up</b> 上方へのみ結合します。 <b>updown</b> 双方向に結合します。
<b>splithint</b>	(キーワードかオプションリスト) 2 ページ見開きスプレッド (ないしはさらなるスプレッドから成るページも) の特別な扱いを有効にします。ページを縦または横に、2 個以上のセクションに分割することができます。キーワード includebox を指定すると、分割領域群は includebox オプションによって定義されます。これに換えて、以下のオプションを与えることもできます : <b>x</b> (float) x 軸に関する除数。例 : 0.5 なら 2 ページ見開きスプレッド、0.33 なら 3 ページスプレッド。 <b>y</b> (float) y 軸に関する除数。
<b>standalone-fontsize</b>	(float) 巨大グリフに対する最小文字サイズ。巨大グリフは 1 グリフストリップを形成し、かつ、他の区域とは結合されません (デフォルト : 70)。
<b>supertable-columns</b>	(整数。layoutastable=true の場合のみ) 区域の連なりをスーパー表組と見なすためのレイアウト行内の最小の段組数。表組が段落群から作成されるとき、これらの段組は結合されずに、別々の区域として認識されます。この結果として、レイアウト認識はこれらの区域の連なりを表組として特定することができます (デフォルト : 4)。
<b>tabledetect</b>	(整数) 再帰的表組認識の深度を指定します (デフォルト : 1) : <b>0</b> 表組認識なし。 <b>1</b> 各区域に対する表組認識。 <b>2</b> レベル 1 で検出された各表セルに対する表組認識。これは、入れ子になった表組のために、また、複数行にわたるセルの解決のために必要です。



表 10.13 TET\_open\_page()・TET\_process\_page() の imageanalysis オプションのサブオプション一覧

オプション	説明
<b>smallimages</b>	(オプションリスト) 小画像除去を制御します。小画像は多くの場合擬似的なものであり、本当の画像ではないので無視する必要があります。使えるオプション： <b>disable</b> (論理値) true の場合には、小画像除去は無効化されます。デフォルト : false <b>maxarea</b> (float) 小画像と見なさせたい画像の最大面積 (幅×高さ) をピクセル単位で。デフォルト : 500 <b>maxcount</b> (整数) 小画像の最大許容数。これを超える数の小画像が見つかったときはそれらすべてが除去されます。デフォルト : 50
<b>merge</b>	(オプションリスト) 画像連結を制御します。この処理は、合わせて 1 個の大画像を形成できる隣接画像群を結合します。これは、PDF 内に個々のストリップが温存されているマルチストリップ画像に対して、また、多数の微小画像に分解されている背景画像に対して有効です。使えるオプション： <b>disable</b> (論理値) true の場合には、画像連結は無効化されます。デフォルト : false <b>gap</b> (float) 連結対象と見なさせたい 2 個の画像の間隔をポイント単位で。デフォルト : 1.0 (0.0 でないのは、位置計算における避けがたい誤差のためです)

表 10.14 TET\_open\_page()・TET\_process\_page() の structureanalysis オプションのサブオプション一覧

オプション	説明
<b>bullets</b>	(オプションリストのリスト。list=true の場合のみ) リスト内でビュレットキャラクタとして用いられる Unicode キャラクタとフォント名の組み合わせを指定します。使えるサブオプション： <b>bulletchars</b> (Unicode 値のリスト) ビュレットキャラクタのための 1 個ないし複数の Unicode 値。このサブオプションを与えない場合には、指定した fontname を用いているキャラクタすべてがビュレットキャラクタとして扱われます。 <b>fontname</b> (文字列) ビュレットキャラクタを使うフォントの名前。このサブオプションを与えない場合には、bulletchars サブオプションで指定したキャラクタはつねにビュレットキャラクタとして扱われます。  例： bullets={{fontname=ZapfDingbats}} bullets={{bulletchars={U+2022}}} bullets={{fontname=KozGoPro-Medium bulletchars={U+2460 U+2461 U+2462 U+2463 U+2464}}}
<b>list</b>	(論理値) リスト認識を有効にします (デフォルト : false)。false の場合には、リスト構造に関する情報は何も決定されません。
<b>paragraph</b>	(論理値) 段落認識を有効にします (デフォルト : true)。false の場合には、段落構造に関する情報は何も決定されません。
<b>table</b>	(論理値) 表組認識を有効にします (デフォルト : true)。false の場合には、表組認識エンジンは無効化されます。

---

**C++** `void close_page(int page)`  
**C# Java** `void close_page(int page)`  
**Perl PHP** `close_page(long page)`  
**VB RB** `Sub close_page(page As Long)`  
**C** `void TET_close_page(TET *tet, int page)`

---

ページハンドルと、関連するすべてのリソースを解放します。

`page` `TET_open_page()` で得られた有効なページハンドル。

**詳細** `TET_close_document()` を呼び出すと、その文書の開いているページはすべて自動的に閉じられます。とはいえ、ページがなくなっただけで明示的に閉じるのが良いプログラミング習慣です。閉じられたページハンドルは、その後はいかなる関数呼び出しにおいても使用してはいけません。

## 10.8 テキスト・メトリクス抽出関数

---

C++ *wstring* *get\_text(int page)*

C# *Java* *String* *get\_text(int page)*

Perl *PHP* *string* *get\_text(long page)*

VB *RB* *Function* *get\_text(page As Long) As String*

C *const char \*TET\_get\_text(TET \*tet, int page, int \*len)*

---

ページの内容から、次のテキスト断片を得ます。

*page* *TET\_open\_page()* で得られた有効なページハンドル。

*len* (C 言語バインディングのみ) 返される文字列の長さを、*TET\_set\_option()* の *outputformat* オプションに応じて保持する変数へのポインタ :

*outputformat=utf8* の場合、長さは Unicode キャラクタの数として報告されます。ヌル終端文字列のバイト数 (これは 8 ビットコードユニットの数に等しいです) は *strlen()* 関数で求められます。

*outputformat=utf16* の場合、長さは 16 ビットコードユニットの数として報告されます。サロゲートペアは 2 個のコードユニットとしてカウントされます。文字列のバイト数は  $2 * len$  です。

*outputformat=utf32* の場合、長さは 32 ビットコードユニットの数として報告されます (これは Unicode キャラクタの数に等しいです)。文字列のバイト数は  $4 * len$  です。

**戻り値** ページ上の、次のテキスト断片を内容として持つ文字列。この断片の長さは、*TET\_open\_page()* の *granularity* オプションによって決定されます。*granularity=glyph* の場合であっても、この文字列は複数のキャラクタを内容として持つ場合があります (99 ページの 7.1 「Unicode のさまざまな重要概念」を参照)。

ページ上のテキストがすべて取得されていた場合には、空文字列かヌルオブジェクトが返されます (後述)。この場合には、テキストがもうない理由はページ上のエラーによるものなのか、それともページの末尾に到達したからなのかを知るために、*TET\_get\_errnum()* を呼び出すべきです。

**バインディング** C 言語バインディング : 結果は、*TET\_set\_option()* の *outputformat* オプションに従って、ヌル終端 UTF-8 (デフォルト) か UTF-16/UTF-32 のいずれかの文字列で提供されます。i5/iSeries と zSeries では、EBCDIC 符号化された UTF-8 を選択することもでき、かつこれがデフォルトで有効になっています。テキストがそれ以上得られないときは、NULL ポインタと *\*len=0* が返されます。

C++・COM : 結果は、UTF-16 形式 (C++ では *wstring*) の Unicode 文字列として提供されます。テキストがそれ以上得られないときは、空文字列が返されます。

Java・.NET・Objective-C : 結果は、Unicode 文字列として提供されます。テキストがそれ以上得られないときは、ヌル (Objective-C では *nil*) 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、*TET\_set\_option()* の *outputformat* オプションに従って、UTF-8 (デフォルト) か UTF-16/UTF-32 のいずれかの文字列で提供されます。Python 3 では、UTF-16/UTF-32 の結果はバイト列として返されます。テキストがそれ以上得られないときは、ヌルオブジェクトが返されます。

REALbasic : 結果は Unicode 文字列として提供されます。テキストがそれ以上得られないときは、空文字列が返されます。

RPG 言語バインディング : 結果は、Unicode 文字列として提供されます。テキストがそれ以上得られないときは、NULL が返されます。

---

**C++** `const TET_char_info *get_char_info(int page)`

**C# Java** `int get_char_info(int page)`

**Perl PHP** `object get_char_info(long page)`

**VB RB** `Function get_char_info(int page) As Long`

**C** `const TET_char_info *TET_get_char_info(TET *tet, int page)`

---

最近のテキスト断片内の、次のグリフに対する詳しい情報を得ます。

`page TET_open_page()` で得られた有効なページハンドル。

**注記** この関数の名前は付け間違いです。ページ上の視覚的なグリフについての情報を報告する関数であって、それに対応する Unicode キャラクタについて報告するのではないのですから、`TET_get_glyph_info()` という名前にしておくべきでした。

**戻り値** `TET_get_text()` が返した最近のテキスト断片について、グリフがそれ以上得られないときは、バインディング依存の値が返されます。詳しくは後述の**バインディング**の項を参照してください。

**詳細** この関数は、`TET_get_text()` の後に 1 回ないし複数回呼び出すことができます。これは、与えられたページハンドルに関連付けられているカレントテキスト断片において、次のグリフへ進み (グリフがもうないときは何も返しません)、そしてそのグリフについての詳しい情報を提供します。テキスト断片が論理キャラクタを  $M$  個持ち、これについてこの関数への呼び出しが  $N$  回成功する (すなわちグリフが  $N$  個ある。  $N > 0$ ) とすると、 $N$  と  $M$  の関係は粒度に依存します :

- ▶ **granularity=glyph** の場合、各テキスト断片はそれぞれただ 1 個のグリフに対応します。すなわち  $N=1$  です。1 個のグリフは多くの場合、1 個のキャラクタに対応します。すなわち  $M=1$  です。ただし、合字グリフの場合には、ただ 1 個のグリフに対して複数のキャラクタに対応します。すなわち  $M > 1$  であり、`TET_get_char_info()` を複数回呼び出す必要があります。
- ▶ **glyph** 以外の粒度の場合には、グリフ列はキャラクタ列を生み出し、ここで各グリフがそれぞれ生み出すキャラクタは 0 個・1 個・複数のいずれでもありえます。このグリフ列は、Unicode キャラクタ列の原料となるわけです。言い換えれば、 $N$  と  $M$  の間の関係は事前にわかりません。 $N$  と  $M$  の間の関係は、内容分析 (ハイフン除去処理でハイフンが除去されるなど) や Unicode 後処理 (字形統合によってキャラクタが追加されたり削除されたりするなど) によって影響を受ける可能性があります。

**glyph** 以外の粒度の場合には、この関数は、`TET_get_text()` への最近の呼び出しが返したテキスト断片を構成する次のグリフへ進みます。これによって、単語検出機能が有効なときにグリフメトリックを取得することができ、また、1 個のテキスト断片には複数のキャラクタが含まれる場合があります。カレントテキスト断片についてすべてのグリフの詳細を取得するには、この関数を、情報をもう返さなくなるまで繰り返し呼び出す必要があります。

構造またはプロパティ/フィールド内のグリフ詳細は、同じページハンドルで次に `TET_get_char_info()` か `TET_close_page()` を呼び出す (先に行なったほう) まで有効です。グリフ情報プロパティ/フィールドのセットは、TET オブジェクトごとにただ 1 個しかありませんので、同じページについてであれ、別のページについてであれ、別の文書についてであれ、再び `TET_get_char_info()` を呼び出す前にクライアント側ですべてのグリフ情報を取得しておく必要があります。

**バインディング** C・C++ 言語バインディング : `TET_get_text()` が返した最近のテキスト断片について、グリフがそれ以上得られないときは、NULL ポインタが返されます。そうでないときは、1 個のグリフに関する情報を内容として持つ `TET_char_info` 構造へのポインタが返されます。このデータ構造のメンバについては表 10.15 で説明します。

COM・Java・.NET・Objective-C 言語バインディング : `TET_get_text()` が返した最近のテキスト断片について、グリフがそれ以上得られないときは、-1 が返されます。そうでないときは 1 が返されます。個々のグリフ情報は、表 10.15 に従った TET プロパティ/パブリックフィールドから取得することができます。関数が -1 を返したにもかかわらずプロパティ/フィールドの値を見た場合には、いずれも値 -1 をとります (`unknown` フィールドは `false` になります)。

Perl・Python 言語バインディング : `get_text()` が返した最近のテキスト断片について、グリフがそれ以上得られないときは、0 が返されます。そうでないときは、表 10.15 に挙げるキー群を含むハッシュが返されます。個々のグリフ情報は、このハッシュ内のキーによって取得することができます。

PHP 言語バインディング : `get_text()` が返した最近のテキスト断片について、グリフがそれ以上得られないときは、空 (ヌル) オブジェクトが返されます。そうでないときは、表 10.15 に挙げるフィールド群を内容として持つオブジェクトが返されます。個々のグリフ情報は、このオブジェクトのメンバフィールドから取得することができます。グリフ情報オブジェクト内の整数フィールドは、PHP 言語バインディングでは `long` として実装されています。

REALbasic バインディング : `get_text()` が返した最近のテキスト断片について、グリフがそれ以上得られないときは、`nil` が返されます。そうでないときは、表 10.15 に挙げるメンバ群を内容として持つ `TET_char_info` オブジェクトが返されます。個々のグリフ情報は、このオブジェクト内のキーによって取得することができます。`attributes` フィールドは REALbasic バインディングでは、REALbasic のインタフェースの問題を回避するために `attr` という名前になっています。

Ruby バインディング : グリフがそれ以上得られないときは、`nil` (ヌルオブジェクト) が返されます。そうでないときは、`TET_char_info` オブジェクトが返されます。

表 10.15 TET\_char\_info 構造のメンバ (C・C++・Ruby) と、同等のパブリックフィールド (Java・PHP・Objective-C)・キー (Perl)・プロパティ (COM・.NET)、およびその型と意味の一覧。詳しくは 82 ページの「グリフメトリック」を参照。

プロパティ/ フィールド名	説明
------------------	----

<code>uv</code>	(整数) カレントグリフに対する UTF-32 Unicode 値。glyph 以外の粒度の場合、これは、最終テキスト断片とはまったく無関係な擬似または中間値をとることがあります。granularity=glyph の場合、グリフに対する Unicode 値列は論理テキストに等しいですが、それ以外の粒度の場合には、それはさまざまな処理ステップによって変更されている可能性があります。
-----------------	--

表 10.15 TET\_char\_info 構造のメンバ (C・C++・Ruby) と、同等のパブリックフィールド (Java・PHP・Objective-C)・キー (Perl)・プロパティ (COM・.NET)、およびその型と意味の一覧。詳しくは 82 ページの「グリフメトリック」を参照。

プロパティ / フィールド名	説明
<b>type</b>	(整数) キャラクタの種別。以下の種別は、ページ上のグリフに対応する本当のキャラクタを記述します。これ以外のすべてのプロパティ / フィールドの値は、対応するグラフによって決定されます： <b>0</b> ちょうど 1 個のグリフに対応する通常のキャラクタ <b>1</b> キャラクタ列 (合字など) の先頭 以下の種別は、ページ上のグリフには対応しない擬似キャラクタを記述します。x・y フィールドは、最近の本当のキャラクタの終了点を表し、width フィールドは 0 になり、これ以外の uv を除くすべてのフィールドは、最近の本当のキャラクタに対応する値をとります： <b>10</b> キャラクタ列 (合字など) のつづき <b>11</b> (非推奨。使われていません) <b>12</b> 挿入された単語・行・区域区切りキャラクタ
<b>attributes<sup>1</sup></b>	(整数) グリフの属性をビットで表したものの。組み合わせることもできます： ビット <b>0</b> 視覚的または意味的な下付き ビット <b>1</b> 視覚的または意味的な上付き ビット <b>2</b> ドロップキャップキャラクタ (段落先頭の大きいキャラクタ) ビット <b>3</b> このグリフの、グリフまたは単語ベースの影重複は除去済 ビット <b>4</b> グリフは、ハイフン区切り箇所直前のキャラクタを表す ビット <b>5</b> contentanalysis={keeplyphenglyphs=true} が指定された場合以外除去されたハイフン区切りアーティファクト (すなわちハイフンキャラクタ) ビット <b>6</b> グリフは、ハイフン区切り箇所直後のキャラクタを表す
<b>unknown</b>	(論理値。C・C++・Perl では整数) 通常は false (0) ですが、元のグリフが Unicode ヘマップでなく、unknownchar として指定されたキャラクタへ置き換えられた場合には true (1) になります。
<b>x, y</b>	(double) グリフの参照点の位置。この参照点は、横書きではグリフ枠の左下隅であり、縦書きでは上端中央の点です。擬似キャラクタについてはこの x・y 座標は、最近のキャラクタの終了点の座標になります。
<b>width</b>	(double) 対応するグリフの幅 (横書きでも縦書きでも)。擬似キャラクタについてはこの幅は 0 になります。
<b>alpha</b>	(double) インラインテキスト進行の向きを度単位で反時計回りに測ったもの。横書きの場合にはこれはテキストのベースラインの向きであり、縦書きの場合にはこれは標準 -90° 向きに対する角度です。この角度は範囲 $-180^\circ < \alpha \leq +180^\circ$ 内になります。標準的な横書きテキストでも、縦書きの標準的テキストでも、この角度は $0^\circ$ になります。
<b>beta</b>	(double) テキスト斜形化角度を度単位で (反時計回り)、alpha の垂線に対して測ったもの。この角度は、正立テキストについては $0^\circ$ になり、斜体になった (斜形化された) テキストについては負になります。この角度は範囲 $-180^\circ < \beta \leq +180^\circ$ になりますが、ただし $\pm 90^\circ$ 以外の値をとります。abs(beta) > 90° なら、そのテキストはベースラインで反転されています。
<b>fontid</b>	(整数) fonts[ ] 擬似オブジェクト内におけるフォントの番号 (pCOS パスリファレンス参照)。fontid が負になることはありません。
<b>fontsize</b>	(double) 文字のサイズ (つねに正)。この値と、グリフの実際の高さとの比は固定ではなく、フォントデザインによって変動する可能性があります。多くのフォントでは、文字サイズは、すべてのアセンダ (アクセント付きキャラクタも含め) とディセンダを包含するように選ばれています。

表 10.15 TET\_char\_info 構造のメンバ (C・C++・Ruby) と、同等のパブリックフィールド (Java・PHP・Objective-C)・キー (Perl)・プロパティ (COM・.NET)、およびその型と意味の一覧。詳しくは 82 ページの「グリフメトリック」を参照。

**プロパティ /  
フィールド名 説明**

<i>textrendering</i>	(整数) テキスト表現モード :
0	テキストを塗る
1	テキスト (輪郭) を描線
2	テキストを塗って描線
3	不可視テキスト (しばしば OCR の結果に対して用いられる)
4	テキストを塗って、それをクリッピングパスに追加
5	テキストを描線し、それをクリッピングパスに追加
6	テキストを塗って描線し、それをクリッピングパスに追加
7	テキストをクリッピングパスに追加

1. REALbasic バインディングではこのフィールドは *attrs* という名前です。

## 10.9 画像抽出関数

---

**C++** `const TET_image_info *get_image_info(int page)`

**C# Java** `int get_image_info(int page)`

**Perl PHP** `object image_info get_image_info(long page)`

**VB RB** `Function get_image_info(int page) As Long`

**C** `const TET_image_info *TET_get_image_info(TET *tet, int page)`

---

ページ上の、次の画像に関する情報を取得します（ただしピクセルデータ本体は取得しません）。

`page` `TET_open_page()` で得られた有効なページハンドル。

**戻り値** 画像がそれ以上得られないときは、バインディング依存の値が返されます。そうでないときは、画像の詳細がバインディング依存の形で利用できます。詳しくは、後述の**バインディング**の項を参照してください。

**詳細** この関数は、与えられたページハンドルに関連付けられている次の画像へ進み（画像がもうないときは 0 か NULL を返します）、この画像についての詳しい情報を提供します。この関数は、画像連結機構によって生成された擬似画像をも返します。しかし、擬似画像を生成するために用いられた消費済画像は返されません。

構造またはプロパティ / フィールド内の画像詳細は、同じページハンドルで次に `TET_get_image_info()` か `TET_close_page()` を呼び出す（先に行なったほう）まで有効です。画像情報プロパティ / フィールドのセットは、TET オブジェクトごとにただ 1 個しかありませんので、同じページについてであれ、別のページについてであれ、別の文書についてであれ、再び `TET_get_image_info()` を呼び出す前にクライアント側ですべての画像情報を取得しておく必要があります。

**バインディング** C・C++ 言語バインディング：ページ上で画像がそれ以上得られないときは、NULL ポインタが返されます。そうでないときは、画像に関する情報を内容として持つ `TET_image_info` 構造へのポインタが返されます。このデータ構造のメンバについては表 10.16 で説明します。

COM・Java・.NET・Objective-C 言語バインディング：ページ上で画像がそれ以上得られないときは、-1 が返されます。そうでないときは 1 が返されます。個々の画像情報は、表 10.16 に従った TET プロパティ / フィールドから取得することができます。関数が -1 を返したにもかかわらずプロパティ / フィールドの値を見た場合には、いずれも値 -1 をとります。

Perl・Python 言語バインディング：ページ上で画像がそれ以上得られないときは、0 が返されます。そうでないときは、表 10.16 に挙げるキー群を含むハッシュが返されます。個々の画像情報は、このハッシュ内のキーによって取得することができます。

PHP 言語バインディング：ページ上で画像がそれ以上得られないときは、空 (ヌル) オブジェクトが返されます。そうでないときは、型 `TET_image_info` のオブジェクトが返されます。個々の画像情報は、表 10.16 に従ってそのフィールドから取得することができます。画像情報オブジェクト内の整数フィールドは、PHP 言語バインディングでは `long` として実装されています。



REALbasic バインディング：ページ上で画像がそれ以上得られないときは、*nil* が返されます。そうでないときは、表 10.16 に挙げるメンバ群を含む *TET\_image\_info* オブジェクトが返されます。個々の画像情報は、このオブジェクト内のメンバによって取得することができます。

Ruby バインディング：画像がそれ以上得られないときは、*nil* (ヌルオブジェクト) が返されます。そうでないときは、*TET\_image\_info* オブジェクトが返されます。

表 10.16 *TET\_image\_info* 構造のメンバ (C・C++・Ruby) と、同等のパブリックフィールド (Java・PHP・Objective-C)・キー (Perl)・プロパティ (COM・.NET)、およびその型と意味の一覧。詳しくは 121 ページの 8.1 「画像抽出の基本」を参照。

プロパティ / フィールド名	説明
<i>x, y</i>	(double) 画像の参照点の位置。参照点は画像の左下隅です。
<i>width, height</i>	(double) ページ上の画像の幅と高さをポイント単位で、画像の辺に沿って測ったもの。
<i>alpha</i>	(double) ピクセル行の向き。この角度は範囲 $-180^\circ < \alpha \leq +180^\circ$ 内になります。正立画像については $\alpha$ は $0^\circ$ になります。
<i>beta</i>	(double) ピクセル列の向きを、 $\alpha$ の垂線に対して測ったもの。この角度は範囲 $-180^\circ < \beta \leq +180^\circ$ になりますが、ただし $\pm 90^\circ$ 以外の値をとります。正立画像については $\beta$ は $-90^\circ < \beta < +90^\circ$ になります。 $ \beta  > 90^\circ$ なら、その画像はベースラインで反転されています。
<i>imageid</i>	(整数) pCOS 擬似オブジェクト <i>images[ ]</i> 内における画像の番号。この擬似オブジェクト内の項目群を通じて、詳細な画像プロパティを取得することができます (pCOS パスリファレンス参照)。

---

**C++** *int write\_image\_file(int doc, int imageid, wstring optlist)*  
**C# Java** *int write\_image\_file(int doc, int imageid, String optlist)*  
**Perl PHP** *long write\_image\_file(long doc, long imageid, string optlist)*  
**VB RB** *Function write\_image\_file(doc As Long, imageid As Long, optlist As String) As Long*  
**C** *int TET\_write\_image\_file(TET \*tet, int doc, int imageid, const char \*optlist)*

---

画像データをディスクへ書き出します。

**doc** *TET\_open\_document\**() で得られた有効な文書ハンドル。

**imageid** 画像の pCOD ID。この ID は、*TET\_get\_image\_info()* を呼び出して成功した後に *imageid* フィールドから、あるいは、*images* 擬似オブジェクト内のすべての項目をなめる (この配列内には *length:images* 項目があります) ことによって取得することができます。

**optlist** 表 10.17 に従って画像関連オプション群を指定したオプションリスト。右記のオプションが使えます：*compression*・*filename*・*keepxmp*・*typeonly*。

**戻り値** エラー時には -1、そうでないなら 0 より大きな値。-1 が返された場合には、*TET\_get\_errmsg()* を呼び出してエラーの詳細を知ることが推奨します。エラーの場合には何の画像出力も生成されません。まれにある非対応の形式の画像の場合も、エラーとして報告されます。戻り値が -1 以外の場合には、その戻り値が示すファイル形式でその画像を抽出できることを示します：

- ▶ -1: エラー発生。何の画像も抽出されません

- ▶ 10 : 画像は TIFF (.tif) として抽出された
- ▶ 20 : 画像は JPEG (.jpg) として抽出された
- ▶ 30 : 画像は JPEG 2000 (.jpx) として抽出された

**詳細** この関数は、指定した pCOS ID を持つ画像に対するピクセルデータを、いくつかの画像形式の 1 つへ変換し、その結果をディスクファイルへ書き出します。 **typeonly** オプションを与えた場合は、画像の種別だけが返され、画像ファイルは生成されません。

**バインディング** C・C++ : 戻り値のためのマクロが *tetlib.h* 内で得られます。

表 10.17 TET\_write\_image\_file()・TET\_get\_image\_data() のオプション一覧

オプション	説明
<b>compression</b>	(キーワード) ピクセルデータを圧縮するためのアルゴリズム (デフォルト : auto) : <b>auto</b> 適切な圧縮アルゴリズムを自動的に選択します。 <b>none</b> (TIFF 画像の場合のみ意味を持ちます) 可能ならば一切圧縮なしでピクセルデータを書き出します。
<b>filename</b> <sup>1</sup>	(文字列。typeonly も与えていないかぎり必須) ディスク上の画像ファイルの名前。この filename に、画像ファイル形式を示す接尾辞が追加されます。 TETML 内の Image/@id attribute 属性にマッチするため、下記のファイル名パターンを推奨します : I<imageid> ここで imageid は、imageid 引数の 10 進表現です。
<b>keepxmp</b>	(論理値) true の場合、かつ、その画像が PDF 内で関連付けられた XMP メタデータを持っている場合には、抽出される TIFF・JPEG 画像内にそのメタデータが埋め込まれます。デフォルト : true
<b>typeonly</b> <sup>1</sup>	(論理値) 与えたオプションに従って画像種別が決定されますが、画像ファイルは書き出されません。TET_get_image_data() 自体は画像種別を返さないの、これは、それが返した画像の種別を決定するために有用です。デフォルト : false

1. TET\_write\_image\_file( ) のみ

---

**C++** `const char *get_image_data(int doc, size_t *length, int imageid, wstring optlist)`  
**C# Java** `final byte[] get_image_data(int doc, int imageid, String optlist)`  
**Perl PHP** `string get_image_data(long doc, long imageid, string optlist)`  
**VB RB** `Function get_image_data(doc As Long, imageid As Long, optlist As String)`  
**C** `const char *TET_get_image_data(TET *tet, int doc, size_t *length, int imageid, const char *optlist)`

---

画像データをメモリから抽出します。

**doc** TET\_open\_document\*( ) で得られた有効な文書ハンドル。

**length** (C・C++ 言語バインディングのみ) 返されるデータの長さがバイト単位で格納されるメモリ位置への C スタイルポインタ。

**imageid** 画像の pCOS ID。この ID は、TET\_get\_image\_info() を呼び出して成功した後に imageid フィールドから、あるいは、images pCOS 配列内のすべての項目をなめる (この配列内には length:images 項目があります) ことによって取得することができます。

**optlist** 表 10.17 に従って画像関連オプション群を指定したオプションリスト。右記のオプションが使えます : compression・keepxmp。

**戻り値** 指定したオプション群に従って画像を表現したデータ。エラーの場合には（画像が抽出できない場合も含め）、C・C++ では NULL ポインタが返され、それ以外の言語バインディングでは空データが返されます。エラーが発生した場合には、`TET_get_errmsg()` を呼び出してエラーの詳細を知ることを推奨します。

**詳細** この関数は、指定した pCOS ID を持つ画像に対するピクセルデータを、いくつかの画像形式の 1 つへ変換し、そのデータをメモリ内で利用可能にします。

**バインディング** COM: 多くのクライアントプログラムでは、Variant 型を用いて画像データを保持します。

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使えます。

REALbasic: 結果は、エンコーディング -1（バイナリデータ）を持つ REALbasic 文字列として提供されます。テキストがそれ以上得られないときは、空文字列が返されます。

## 10.10 TET マークアップ言語 (TETML) 関数

---

<b>C++</b>	<code>int process_page(int doc, int pagenumber, wstring optlist)</code>
<b>C# Java</b>	<code>int process_page(int doc, int pagenumber, String optlist)</code>
<b>Perl PHP</b>	<code>long process_page(long doc, long pagenumber, string optlist)</code>
<b>VB RB</b>	<code>Function process_page(doc As Long, pagenumber As Long, optlist As String) As Int</code>
<b>C</b>	<code>int TET_process_page(TET *tet, int doc, int pagenumber, const char *optlist)</code>

---

ページを処理して TETML 出力を生成します。

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

**pagenumber** 処理したいページの物理的番号。最初のページをページ番号 1 とします。総ページ数は、`TET_pcos_get_number()` と pCOS バス `length:pages` で取得できます。`trailer=true` の場合、この `pagenumber` 引数は 0 にすることもできます。

**optlist** 以下のグループ内のオプション群を指定したオプションリスト：

- ▶ 表 10.10 に従った一般的なページ関連オプション (これらは `pagenumber=0` の場合には無視されます) : `clippingarea` · `contentanalysis` · `excludebox` · `fontsize range` · `granularity` · `ignoreinvisibletext` · `imageanalysis` · `includebox` · `layoutanalysis` · `skipengines`
- ▶ 表 10.18 に従った処理詳細を指定するオプション : `tetml`

表 10.18 `TET_process_page()` の追加オプション

オプション	説明
<b>tetml</b>	(オプションリスト) TETML の詳細を制御します。以下のオプションが使えます：
<b>elements</b>	(オプションリスト) オプションな TETML 要素を指定します：
<b>line</b>	( <code>granularity=word</code> の場合のみ) <code>true</code> の場合、TETML 出力は、Para レベルと Word レベルの間に Line 要素を含みます。デフォルト : <code>false</code>
<b>glyphdetails</b>	(オプションリスト。 <code>granularity=glyph</code> · <code>word</code> の場合のみ) 各 Glyph 要素に対して、どのグリフ属性が報告されるかを指定します (すべてのサブオプションのデフォルト : <code>false</code> )：
<b>all</b>	(論理値) すべての属性サブオプションを有効にします
<b>dehyphenation</b>	(論理値) 属性 <code>dehyphenation</code> を出力することによって、ハイフン区切りされた単語を示します。
<b>dropcap</b>	(論理値) 属性 <code>dropcap</code> を出力することによって、単語の先頭の大きなキャラクターを示します。
<b>geometry</b>	(論理値) 属性 <code>x</code> · <code>y</code> · <code>width</code> · <code>alpha</code> · <code>beta</code> を出力します。
<b>font</b>	(論理値) 属性 <code>font</code> · <code>fontsize</code> · <code>textrendering</code> · <code>unknown</code> を出力します。
<b>sub</b>	(論理値) 属性 <code>sub</code> を出力することによって、下付きを示します。
<b>sup</b>	(論理値) 属性 <code>sup</code> を出力することによって、上付きを示します。
<b>trailer</b>	(論理値) <code>true</code> の場合には、文書トレーラデータが、すなわち最終ページの後のデータが出力されます (これ以前に出力されたページ固有データにこれを連結する必要があります)。このオプションは、トレーラデータを出力するために、この関数を最後に呼び出す際に必要です。 <code>pagenumber=0</code> の場合には、トレーラデータのみが (ページ固有データなしで) 出力されます。 <code>trailer=true</code> を与えた後は、その同じ文書に対してはもう <code>TET_process_page()</code> を呼び出してはいけません。デフォルト : <code>false</code>

**戻り値** エラー時には-1、そうでないなら1。ただしTETMLモードでは、問題はTETMLの *Exception* 要素内で報告されますので、この関数はつねに成功します。

**詳細** この関数はページを開き、*TET\_open\_document\*()* に与えた形式関連オプション群に従って出力を生成して、ページを閉じます。生成されたデータは、*TET\_get\_xml\_data()* で取得することができます。

この関数は、対応する *TET\_open\_document\*()* への呼び出しでオプション *tetml* を与えた場合にのみ呼び出す必要があります。ヘッダデータは、すなわち、先頭ページの前の文書固有データは、*TET\_open\_document\*()* によって、先頭ページデータの前に生成されます。これは、*TET\_process\_page()* を初めて呼び出す前に *TET\_get\_xml\_data()* を呼び出すことによって別個に取得することもできますし、ページ関連データと組み合わせて取得することもできます。

トレーラデータ、すなわち、最終ページの後の文書固有データは、文書に対してこの関数を最後に呼び出す際に *trailer* サブオプションで要求する必要があります。トレーラデータは、最終ページの後に別個の呼び出しを行なって生成することもできますし (*pagenumber=0*)、最終ページと一緒に生成することもできます (*pagenumber* は0以外)。ページ群は任意の順序で抽出することができ、また、文書のページ群の任意の部分集合を抽出することができます。

トレーラを取得せずに *TET\_close\_document()* を呼び出すとエラーになります。トレーラを取得した後に *TET\_process\_page()* を呼び出してもエラーになります。

---

**C++** *const char \*get\_xml\_data(int doc, size\_t \*length, wstring optlist)*

**C# Java** *final byte[] get\_xml\_data(int doc, String optlist)*

**Perl PHP** *string get\_xml\_data(long doc, string optlist)*

**VB RB** *Function get\_xml\_data(doc As Long, optlist As String)*

**C** *const char \*TET\_get\_xml\_data(TET \*tet, int doc, size\_t \*length, const char \*optlist)*

---

TETML データをメモリから取得します。

**doc** *TET\_open\_document\*()* で得られた有効な文書ハンドル。

**length** (C・C++ 言語バインディングのみ) 返される文字列の長さをバイト単位で保持する変数へのポインタ。*length* は終端ヌルバイトを勘定しません。

**optlist** (現在、使えるオプションはありません。)

**戻り値** 指定したオプション群に従った、次のデータ断片を内容として持つバイト配列。バッファが空の場合には、空文字列が返されます (C では NULL ポインタかつ *\*len=0*)。

**詳細** この関数は、*TET\_open\_document\*()* と、1回ないし複数回の *TET\_process\_page()* への呼び出しによって生成された TETML データを取得します。TETML データは、*outputformat* オプションにかかわらず、つねに UTF-8 で符号化されています。内部バッファはこの呼び出しによってクリアされます。*TET\_process\_page()* を呼び出すたびに *TET\_get\_xml\_data()* を呼び出す必要はありません。クライアント側では、1個ないし複数のページに対する、ないし文書全体に対するデータをバッファ内に蓄積しておくことが可能です。

TETML モードでは、*TET\_close\_document()* の前にこの関数を少なくとも1回呼び出す必要があります。でないとデータは利用できなくなってしまう。 *TET\_get\_xml\_data()* をちょうど1回だけ呼び出す場合には (そのようなただ1回呼び出しは、*TET\_process\_*

`page()` への最後の呼び出しと `TET_close_document()` との間に行う必要があります)、バッファは文書全体に対する整形形式 TETML 文書を内容として持っていることが保証されます。この関数は、`TET_open_document*()` の `tetml` オプションに `filename` サブオプションを与えた場合には呼び出ししてはいけません。

**バインディング** C・C++ 言語バインディング：結果は、ヌル終端 UTF-8 として提供されます。i5/iSeries・zSeries では、EBCDIC 符号化された UTF-8 が返されます。返されたデータバッファは、次に `TET_get_xml_data()` を呼び出す時まで使えます。

Java・.NET バインディング：結果は、UTF-8 データを内容として持つバイト配列として提供されます。

COM：多くのクライアントプログラムは、Variant 型を用いて UTF-8 データを保持します。

REALbasic：結果は、エンコーディング UTF-8 の REALBasic String として返されます。

PHP 言語バインディング：結果は、UTF-8 文字列として提供されます。

Python：結果は、8 ビット文字列として返されます (Python 3：`bytes`)。

RPG 言語バインディング：結果は、ヌル終端 EBCDIC UTF-8 として返されます。

## 10.11 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 文法が使えます。詳しい説明は、別の文書としてある 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** `float pcos_get_number(int doc, string path)`

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

**C** `double TET_pcos_get_number(TET *tet, int doc, const char *path, ...)`

---

数値型か論理型の pCOS パスの値を得ます。

Get the value of a pCOS path with type number or boolean.

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

**path** 数値または論理値オブジェクトへの完全 pCOS パス。

**追加引数** (C 言語バインディングのみ) 任意の数の追加引数を、**key** 引数がそれに対応するプレースホルダを含んでいる場合には (文字列には `%s`、整数には `%d`。%% とするとパーセント記号 1 個)、与えることができます。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明示的に組み立てる手間が省けます。プレースホルダの数と型が、与える追加引数と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 パスによって特定されたオブジェクトの数値。論理値の場合は、それが `true` ならば 1 が返され、そうでないなら 0 が返されます。

---

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

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

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

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

**C** `const char *TET_pcos_get_string(TET *tet, int doc, const char *path, ...)`

---

名前型・文字列型・論理型のいずれかの pCOS パスの値を得ます。

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

**path** 文字列・名前・論理値のいずれかのオブジェクトへの完全 pCOS パス。

**追加引数** (C 言語バインディングのみ) 任意の数の追加引数を、**key** 引数がそれに対応するプレースホルダを含んでいる場合には (文字列には `%s`、整数には `%d`。%% とするとパーセント記号 1 個)、与えることができます。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明示的に組み立てる手間が省けます。プレースホルダの数と型が、与える追加引数と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 パスによって特定されたオブジェクトの値を持つ文字列。論理値の場合は、文字列 `true` か `false` が返されます。

**詳細** この関数は、pCOS がフルモードで動作しておらず、かつオブジェクトが文字列型の場合には、例外を発生させます (pCOS パスリファレンス参照)。例外として、オブジェクト /

*Info*/\* (文書情報キー群) は制限 pCOS モードでも *nocopy=false* か *plainmetadata=true* ならば取得することができ、また、*bookmarks[...]/Title* と *pages[...]/Annots/Contents* は制限 pCOS モードでも *nocopy=false* ならば取得できます。

この関数は、PDF 文書から取得する文字列がテキスト文字列であると前提しています。バイナリデータを内容として持つ文字列オブジェクトは、これに換えて、データにかなる変更をも加えない *TET\_pcos\_get\_stream()* で取得する必要があります。

**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 (zSeries・i5/iSeries では EBCDIC-UTF-8) で返されます。返される文字列は、最大 10 項目を持つリングバッファ内に格納されます。10 個を超える文字列が取得されたときには、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++ ラップのデフォルトの *wstring* 設定では *wstring* として返されます。zSeries・i5/iSeries の *string* 互換モードでは、結果は BOMなし EBCDIC-UTF-8 で返されます。

Java・.NET バインディング: 結果は、Unicode 文字列として提供されます。テキストがそれ以上得られないときは、ヌルオブジェクトが返されます。

Perl・PHP・Python 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。テキストがそれ以上得られないときは、ヌルオブジェクトが返されます。

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

---

**C++** *const unsigned char \*pcos\_get\_stream(int doc, int \*length, string optlist, wstring path)*

**C# Java** *final byte[] pcos\_get\_stream(int doc, String optlist, String path)*

**Perl PHP** *string pcos\_get\_stream(int doc, string optlist, string path)*

**VB RB** *Function pcos\_get\_stream(doc as Long, optlist As String, path As String)*

**C** *const unsigned char \*TET\_pcos\_get\_stream(TET \*tet, int doc, int \*length, const char \*optlist, const char \*path, ...)*

---

*stream* 型・*fstream* 型・文字列型のいずれかの pCOS パスの値を得ます。

*doc* *TET\_open\_document\*()* で得られ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取る変数へのポインタ。

*optlist* 表 10.19 に従ってストリーム取得オプション群を指定したオプションリスト。

*path* ストリームまたは文字列オブジェクトへの完全 pCOS パス。

**追加引数** (C 言語バインディングのみ) 任意の数の追加引数を、*key* 引数がそれに対応するプレースホルダを含んでいる場合には (文字列には *%s*、整数には *%d*。%% とするとパーセント記号 1 個)、与えることができます。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明示的に組み立てる手間が省けます。プレースホルダの数と型が、与える追加引数と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** ストリームまたは文字列の中に含まれた非暗号化データ。ストリームまたは文字列が空の場合、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場合には、返されるデータは空になります (C・C++ では NULL)。

オブジェクトが *stream* 型の場合には、*keepfilter=true* でないかぎり、すべてのフィルタがストリーム内容から除去されます (すなわち、生データ本体が返されます)。オブジェクトが *fstream* 型か **文字列型** の場合には、PDF ファイル内で見つかったとおりのデータがそのまま届けられますが、ただし例外として ASCII85・ASCIIHex フィルタは除去されます。

データの解凍と ASCII フィルタの除去に加えて、*convert* オプションに従ってテキスト変換が適用される場合もあります。

**詳細** この関数は、pCOS がフルモードで動作していないときには例外を発生させます (pCOS パスリファレンス参照)。例外として、オブジェクト */Root/Metadata* は制限 pCOS モードでも *nocopy=false* か *plainmetadata=true* ならば取得することができます。パスが *stream* 型・*fstream* 型・**文字列型** のいずれかのオブジェクトを指し示していない場合にも、例外が発生します。

この関数は、その名前にもかかわらず、**文字列型** のオブジェクトを取得するために使うこともできます。オブジェクトをテキスト文字列として扱う *TET\_pcos\_get\_string()* と違って、この関数は、返されたデータに対していかなる変更をも加えません。バイナリ文字列データは PDF 内でめったに用いられませんので、自動的にきちんとは検出できません。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、それともテキストとして取得するかを決めるのは、したがってユーザ側の役割です。

**バインディング** COM : 多くのクライアントプログラムは、Variant 型を用いてストリーム内容を保持します。COM による JavaScript は、返されたバリエーション配列の長さを取得することを許しません (しかしこれは他の言語と COM では動作します)。

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使えます。

Python : 結果は、8 ビット文字列として返されます (Python 3 : *bytes*)。

**注記** この関数を利用すると、PDF に埋め込まれているフォントデータを取得することができます。ユーザは、フォントは各フォントベンダの使用許諾の対象であり、それぞれの知的所有権保有者の明示的許諾なしでは再利用してはならないという事実にご留意してください。お使いのフォントベンダに連絡して、関連するライセンス契約を協議してください。

表 10.19 TET\_pcos\_get\_stream() のオプション一覧

オプション	説明
<b>convert</b>	(キーワード。非対応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内容が変換されるかどうかを制御します (デフォルト : none) :
<b>none</b>	内容をバイナリデータとして扱い、何の変換も行いません。
<b>unicode</b>	内容をテキストデータとして (すなわち、TET_pcos_get_string() におけるのと全く同じに) 扱い、Unicode へ正規化します。非 Unicode 対応言語バインディングでは、これは、データは BOM なし UTF-8 形式へ変換されることを意味します。このオプションは、めったに用いられない PDF 内のデータ型「テキストストリーム」(たとえばこれは JavaScript のために用いられる場合がありますが、JavaScript のほとんど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入っています) のために必要です。

表 10.19 TET\_pcos\_get\_stream( ) のオプション一覧

オプション	説明
<i>keepfilter</i>	(論理値。画像データストリームに対してのみ推奨されます。非対応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) true の場合には、ストリームデータは、その画像の <i>filterinfo</i> 擬似オブジェクト内で指定されているフィルタで圧縮されます (pCOS パスリファレンス参照)。false の場合には、ストリームデータは解凍されます。デフォルト : すべての非対応フィルタに対して true、それ以外には false

# A TETライブラリクイックリファレンス

以下の表に、すべての TET API 関数の概観を示します。頭に (C) が付いているものは、関数の C プロトタイプであり、Java 言語バインディングでは利用できないことを意味します。

## セットアップ関数

関数プロトタイプ	ページ
(C) <i>TET *TET_new(void)</i>	161
<i>void delete()</i>	161

## PVF 関数

関数プロトタイプ	ページ
<i>void create_pvf(String filename, byte[] data, String optlist)</i>	162
<i>int delete_pvf(String filename)</i>	163
<i>int info_pvf(String filename, String keyword)</i>	163

## Unicode 変換関数

関数プロトタイプ	ページ
<i>String convert_to_unicode(String inputformat, byte[] input, String optlist)</i>	165

## 例外処理関数

関数プロトタイプ	ページ
<i>String get_apiname()</i>	167
<i>String get_errmsg()</i>	167
<i>int get_errnum()</i>	167

## 文書関数

関数プロトタイプ	ページ
<i>int open_document(String filename, String optlist)</i>	171
(C) <i>int TET_open_document_callback(TET *tet, void *opaque, size_t filesize, size_t (*readproc)(void *opaque, void *buffer, size_t size), int (*seekproc)(void *opaque, long offset), const char *optlist)</i>	176
<i>void close_document(int doc)</i>	177

## ページ関数

関数プロトタイプ	ページ
<i>int open_page(int doc, int pagenumber, String optlist)</i>	178
<i>void close_page(int page)</i>	186

## テキスト・メトリック抽出関数

関数プロトタイプ	ページ
<i>String get_text(int page)</i>	187
<i>int get_char_info(int page)</i>	188

## 画像抽出関数

関数プロトタイプ	ページ
<i>int get_image_info(int page)</i>	192
<i>int write_image_file(int doc, int imageid, String optlist)</i>	193
<i>final byte[] get_image_data(int doc, int imageid, String optlist)</i>	194

## TET マークアップ言語 (TETML) 関数

関数プロトタイプ	ページ
<i>int process_page(int doc, int pagenumber, String optlist)</i>	196
<i>final byte[] get_xml_data(int doc, String optlist)</i>	197

## オプション処理

関数プロトタイプ	ページ
<i>void set_option(String optlist)</i>	158

## pCOS 関数

関数プロトタイプ	ページ
<i>double pcos_get_number(int doc, String path)</i>	199
<i>String pcos_get_string(int doc, String path)</i>	199
<i>final byte[] pcos_get_stream(int doc, String optlist, String path)</i>	200

## B 更新履歴

本マニュアルの更新履歴

日付	変更点
2012年4月04日	▶ <i>TET 4.1p1</i> に合わせて更新
2012年2月20日	▶ <i>TET 4.1</i> に合わせて更新
2010年9月22日	▶ <i>TET 4.op2</i> に合わせて更新
2010年7月27日	▶ <i>TET 4.0</i> に合わせて更新
2009年2月01日	▶ <i>TET 3.0</i> に合わせて更新
2008年1月16日	▶ <i>TET 2.3</i> に合わせてマニュアル更新
2007年1月23日	▶ <i>TET 2.0</i> に合わせた小幅追補
2005年12月14日	▶ <i>TET 2.1.0</i> に合わせて追加・修正。 <i>PHP・RPG</i> 言語バインディングに関する記述を追加
2005年6月20日	▶ <i>TET 2.0.0</i> に合わせてマニュアルを拡張・再構成
2003年10月14日	▶ <i>TET 1.1</i> に合わせてマニュアル更新
2002年11月23日	▶ <i>TET 1.0.2</i> に合わせて <i>TET_open_doc_callback()</i> の説明とページサイズ決定のコードサンプルを追加
2002年4月4日	▶ <i>TET 1</i> に合わせて第1版



# 索引

## A

API リファレンス 151

## B

BMP 100

BOM (Byte Order Mark) 100

## C

C++ と .NET 38

C++ バインディング 32

CLI 32

codelist 117

COM バインディング 35

CSV 形式 148

C バインディング 29

## D

Dispose() 161

## F

float・整数値

    オプションリスト内の 156

FontReporter Plugin 13, 116

## G

glyphlist 119

glyphrule 120

granularity 92

## H

HTML コンバータ (XSLT サンプル) 148

## I

IFilter

    Microsoft 製品用 60

## J

J2EE アプリケーションサーバ 36

Javadoc 37

Java バインディング 36

## L

Lucene 検索エンジン 53

## M

MediaWiki 65

## N

.NET バインディング 38

## O

Objective-C バインディング 39

Oracle Text 57

## P

pCOS

    API 関数 199

    クックブック 17

PDF のバージョン 13

Perl バインディング 41

PHP バインディング 42

PUA 100

Python バインディング 44

## R

REALbasic バインディング 45

resourcefile パラメタ 72

RPG バインディング 48

Ruby バインディング 46

## S

searchpath 70

Solr 検索サーバ 56

## T

tet.upr 72

TET\_CATCH() 167

TET\_close\_document() 177

TET\_close\_page() 186

TET\_convert\_to\_unicode() 165

TET\_create\_pvf() 162

TET\_delete() 161

TET\_delete\_pvf() 163

TET\_EXIT\_TRY() 30, 167

TET\_get\_apiname() 167

TET\_get\_char\_info() 188  
TET\_get\_errmsg() 167  
TET\_get\_errnum() 167  
TET\_get\_image\_data() 194  
TET\_get\_image\_info() 192  
TET\_get\_text() 187  
TET\_get\_xml\_data() 197  
TET\_info\_pvf() 163  
TET\_new() 161  
TET\_open\_document() 171  
TET\_open\_document\_callback() 176  
TET\_open\_page() 178  
TET\_pcos\_get\_number() 199  
TET\_pcos\_get\_stream() 200  
TET\_pcos\_get\_string() 199  
TET\_RETHROW() 167  
TET\_set\_option() 158  
TET\_TRY() 167  
TET\_write\_image\_file() 193  
TETML 131  
スキーマ 139  
TETRESOURCEFILE 環境変数 71  
TET コネクタ  
Lucene 用 53  
MediaWiki 用 65  
Microsoft 製品用 60  
Oracle 用 57  
Solr 用 56  
TIKA 用 63  
TET コマンドラインツール 19  
TET の機能 13  
TET クックブック 17  
TET プラグイン  
Adobe Acrobat 用 51  
TeX 文書 76  
TIKA ツールキット 63  
ToUnicode CMap 118

## U

Unichar 値  
オプションリスト内の 154  
Unicode  
BOM 100  
後処理 105  
オプションリスト内の 154  
概念 99  
正規化 112  
符号化形式 100  
符号化スキーム 100  
分解 108  
前処理 102  
前処理・後処理 102  
Unicode 字形統合 105  
UPR ファイル形式 69  
UTF-32 114

UTF 形式 100

## X

XMP メタデータ 78  
XSLT サンプル 148  
画像の 122  
XSD スキーマ  
TETML の 139  
XSLT 143  
サンプル 146, 16  
x ハイット 85

## あ

アセンダ 85  
後処理 102  
アラビア文字 90  
暗号化文書 67  
入れ子のオプションリスト 152  
インストール  
TET の 7  
インチ 81  
オプションリスト 151  
オプションリスト文法 151

## か

回転済みグリフ 88  
影付き除去 94  
画像  
XMP メタデータ 122  
色再現性 129  
解像度 127  
視覚情報 127  
小画像除去 124  
抽出 121  
ディスクまたはメモリへ抽出 121  
配置画像 125  
非対応の種別 129  
文書内の画像数 124  
ページベースの抽出ループ 126  
リソース 125  
リソースベースの抽出ループ 126  
連結 123  
キーワード  
オプションリスト内の 155  
基本多言語面 100  
キャップハイット 85  
キャラクタとグリフ 99  
キャラクタ列 101  
擬似太字除去 94  
矩形  
オプションリスト内の 157  
組文字 101



グリフ 99  
グリフ規則 120  
グリフメトリック 82  
グリフリスト 119  
権限パスワード 67  
コマンドラインツール 19  
コンコードダンス (XSLT サンプル) 146  
合字 101  
互換分解 109

## さ

サープレット 36  
最適化  
    速度の 73  
索引 (XSLT サンプル) 148  
作成例  
    XSLT 146  
サロゲート 100  
座標系 81  
シーケンス 101  
しおり 79  
視覚情報  
    画像の 127  
終了点  
    グリフと単語の 85  
シュラッグ機能 67  
小画像除去 124  
所有者パスワード 67  
私用領域 100  
字形統合 105  
数値  
    オプションリスト内の 155  
スキーマ 139  
正規化 112  
正準分解 108  
双方向テキスト 90  
速度を最適化 73

## た

縦書き 87  
単位 81  
単語境界検出 93  
単語検出機能 93  
注釈 79  
テキスト抽出ステータス 67  
テキストフィルタリング 102  
添付パスワード 67  
ディセンダ 85

## な

内容分析 92  
生テキスト抽出 (XSLT サンプル) 149

日中韓 (日本語・中国語・韓国語) 14, 87  
    互換形 88  
    設定 7  
    単語境界 87

## は

配置画像 125  
ハイフン除去 94  
ハイライト 85  
パスワード 67  
パッケージ 80  
表意文字テキスト  
    単語境界 87  
評価版 8  
表組検出 98  
表組抽出 (XSLT サンプル) 148  
ファイル検索 70  
ファイル添付 80  
フォームフィールド 79  
フォント統計 (XSLT サンプル) 147  
フォントの使用箇所を検索 (XSLT サンプル)  
    147  
フォントフィルタリング (XSLT サンプル) 146  
分解 108  
文書情報項目 77  
文書スタイル 96  
文書領域 77  
文法  
    オプションリストの 151  
ヘブライ文字 90  
ページベースの画像抽出ループ 126  
ポイント 81  
ポートフォリオ 80

## ま

前処理 102  
マスタパスワード 67  
マップ不能グリフ 115  
ミニサンプル 16  
ミリメートル 81  
文字列  
    オプションリスト内の 154

## や

ユーザパスワード 67

## ら

ライセンスキー 9  
リガチャ 101  
リスト値  
    オプションリスト内の 152

リソースカテゴリ 69  
リソースのコンフィギュレーション 69  
リソースベースの画像抽出ループ 126  
粒度 92  
領域  
    テキスト抽出の 87  
例  
    テキスト抽出ステータス 67  
例外処理 27  
    C の場合 29  
レスポンスファイル 22  
ログ記録 169  
論理値  
    オプションリスト内の 155



**PDFlib GmbH**

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

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

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

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

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

