PDFlib GmbH

Text and Image Extraction Toolkit (TET)

Version 5.6

Toolkit for extracting Text, Images, and other items from PDF

Copyright © 2002–2024 PDFlib GmbH. All rights reserved.
Protected by European and U.S. patents.

PDFlib GmbH
Franziska-Bilek-Weg 9, 80339 München, Germany
www.pdflib.com

sales@pdflib.com
PDFlib_support@pdftron.com (please include your license number)

This publication and the information herein is furnished as is, is subject to change without notice, and should not be construed as a commitment by PDFlib GmbH. PDFlib GmbH assumes no responsibility or liability for any errors or inaccuracies, makes no warranty of any kind (express, implied or statutory) with respect to this publication, and expressly disclaims any and all warranties of merchantability, fitness for particular purposes and noninfringement of third party rights.

TET contains the following third-party components:

Adobe CMap resources, Copyright © 1990-2019 Adobe
AES, Arcfour and SHA algorithms, Copyright © 1995-1998 Eric Young
Brotli decompression code, Copyright © 2009, 2010, 2013-2016 by the Brotli Authors
Expat XML parser, Copyright © 2001-2022 Expat maintainers
ICClib, Copyright © 1997-2002 Graeme W. Gill
ICU International Components for Unicode, Copyright © 1991-2020 Unicode, Inc.
libjpeg, Copyright © 1991-2019, Thomas G. Lane, Guido Vollbeding
Libtiff image library, Copyright © 1988-1997 Sam Leffler, Copyright © 1991-1997 Silicon Graphics, Inc.
MD5 message digest, Copyright © 1991-2, RSA Data Security, Inc.
OpenJPEG, Copyright © 2002-2014, Université catholique de Louvain (UCL), Belgium
WOFF2 font decompression, Copyright © 2013-2017 by the WOFF2 Authors
Zlib compression library, Copyright © 1995-2022 Jean-loup Gailly and Mark Adler

0 First Steps with TET 7

0.1 Installing the Software 7

0.2 Applying the TET License Key 8

1 Introduction 11

1.1 Overview of TET Features 11

1.2 Many ways to use TET 13

1.3 Roadmap to Documentation and Samples 14

1.4 What's new in TET 5.1? 15

1.5 What's new in TET 5.2? 15

1.6 What's new in TET 5.3? 16

1.7 What's new in TET 5.4? 16

1.8 What's new in TET 5.5? 17

2 TET Command-Line Tool 19

2.1 Command-Line Options 19

2.2 Constructing TET Command Lines 22

2.3 Command-Line Examples 23

2.3.1 Extracting Text 23

2.3.2 Extracting Images 23

2.3.3 Generating TETML 24

2.3.4 Advanced Options 24

3 TET Library Language Bindings 25

3.1 C Binding 25

3.2 C++ Binding 28

3.3 Java Binding 30

3.4 .NET Binding 32

3.5 Objective-C Binding 35

3.6 Perl Binding 37

3.7 PHP Binding 38

3.8 Python Binding 40

3.9 Ruby Binding 41

3.10 RPG Binding 43

4 TET Connectors 45

4.1 Free TET Plugin for Adobe Acrobat 45

4.2 TET Connector for the Lucene Search Engine 46

4.3 TET Connector for the Solr Search Server 49

4.4 TET Connector for Oracle 50

4.5 TET PDF IFilter for Microsoft Products 53
4.6 TET Connector for the Apache TIKA Toolkit 55
4.7 TET Connector for MediaWiki 57

5 Configuration 59

5.1 Extracting Content from protected PDF 59
5.2 Resource Configuration and File Searching 61
5.3 Recommendations for common Scenarios 65

6 Text Extraction 69

6.1 PDF Document Domains 69
6.2 Page and Text Geometry 74
6.3 Text Color 80
6.4 Chinese, Japanese, and Korean Text 82
6.4.1 CJK Encodings and CMaps 82
6.4.2 Word Boundaries for CJK Text 82
6.4.3 Vertical Writing Mode 82
6.4.4 CJK Decompositions: Narrow, wide, vertical, etc. 83
6.5 Bidirectional Arabic and Hebrew Text 84
6.5.1 General Bidi Topics 84
6.5.2 Postprocessing Arabic Text 84
6.6 Content Analysis 86
6.7 Layout Analysis and Document Styles 89
6.8 Table and List Detection 92
6.9 Check whether an Area is empty 94
6.10 Annotation Contents 95

7 Advanced Unicode Handling 97

7.1 Important Unicode Concepts 97
7.2 Text Preprocessing (Filtering) 100
7.2.1 Filters for all Granularities 100
7.2.2 Filters for Granularity Word and above 101
7.3 Unicode Postprocessing 103
7.3.1 Unicode Folding 103
7.3.2 Unicode Decomposition 106
7.3.3 Unicode Normalization 110
7.4 Supplementary Characters and Surrogates 112
7.5 Unicode Mapping for Glyphs 113

8 Image Extraction 119

8.1 Image Extraction Basics 119
8.2 Extracting Images 122

8.2.1 Placed Images and Image Resources 122
8.2.2 Page-based and Resource-based Image Retrieval 123
8.2.3 Geometry of Placed Images 125
8.3 Merging Fragmented Images 127
8.4 Small and Large Image Filtering 129
8.5 Image Colors and Masking 130
8.5.1 Color Spaces 130
8.5.2 Image Masks and Soft Masks 131

9 TET Markup Language (TETML) 133

9.1 Creating TETML 133
9.2 TETML Examples 135
9.3 Controlling TETML Details 139
9.4 TETML Elements and the TETML Schema 143
9.5 Transforming TETML with XSLT 152
9.6 XSLT Samples 155

10 TET Library API Reference 159

10.1 Option Lists and other general Topics 159
10.1.1 Option List Syntax 159
10.1.2 Basic Types 161
10.1.3 Geometric Types 164
10.1.4 Unicode Support in Language Bindings 165
10.1.5 Encoding Names 165
10.1.6 Multi-threaded Programming 166
10.2 General Methods 167
10.2.1 Option Handling 167
10.2.2 Setup 170
10.2.3 PDFlib Virtual Filesystem (PVF) 171
10.2.4 Unicode Conversion 173
10.2.5 Exception Handling 175
10.2.6 Logging 177
10.3 Document Methods 179
10.4 Page Methods 188
10.5 Text and Glyph Details Retrieval Methods 198
10.6 Image Retrieval 204
10.7 TET Markup Language (TETML) 208
10.8 pCOS Methods 211

A TET Library Quick Reference 215

B Revision History 217

Index 219

0 First Steps with TET

0.1 Installing the Software

TET is delivered as an installer or compressed package for Windows and as a compressed archive for all other supported operating systems. All TET packages contain the TET command-line tool and the TET library/component, plus support files, documentation, and examples. After installing or unpacking TET the following steps are recommended:

- Users of the TET command-line tool can use the executable right away. The available options are discussed in Section 2.1, »Command-Line Options«, page 19, and are also displayed when you execute the TET command-line tool without any options.
- Users of the TET library/component should read one of the sections in Chapter 3, »TET Library Language Bindings«, page 25, corresponding to their preferred development environment, and review the installed examples.

If you obtained a commercial TET license you must enter your TET license key according to Section 0.2, »Applying the TET License Key«, page 8.

CJK configuration. In order to extract Chinese, Japanese, or Korean (CJK) text which is encoded with legacy encodings TET requires the corresponding CMap files for mapping CJK encodings to Unicode. The CMap files are contained in all TET packages, and are installed in the *resource/cmap* directory within the TET installation directory.

On non-Windows systems you must manually configure the CMap files:

- For the TET command-line tool this can be achieved by supplying the name of the directory holding the CMap files with the *--searchpath* option.
- For the TET library/component you can set the *searchpath* at runtime:

```
tet.set_option("searchpath={/path/to/resource/cmap}");
```

As an alternative method for configuring access to the CJK CMap files you can set the *TETRESOURCEFILE* environment variable to point to a UPR configuration file which contains a suitable *searchpath* definition.

Restrictions of the evaluation version. The TET command-line tool and library can be used as fully functional evaluation versions even without a commercial license. Unlicensed versions support all features, but will only process PDF documents with up to 10 pages and 1 MB size. Evaluation versions of TET must not be used for production purposes, but only for evaluating the product. Using TET for production purposes requires a valid TET license.

0.2 Applying the TET License Key

Using TET for production purposes requires a valid TET license key. Once you purchased a TET license you must apply your license key in order to allow processing of arbitrarily large documents. There are several methods for applying the license key; choose one of the methods detailed below.

Note TET license keys are platform-dependent, and can only be used on the platform for which they have been purchased.

Windows installer. If you are working with the Windows installer you can enter the license key when you install the product. The installer will add the license key to the registry (see below).

Working with a license file. PDFlib products read license keys from a license file, which is a text file according to the format shown below. You can use the template *licensekeys.txt* which is contained in all TET distributions. Lines beginning with a '#' character contain comments and are ignored; the second line contains version information for the license file itself:

```
# Licensing information for PDFlib GmbH products
PDFlib license file 1.0
TET 5.6 ...your license key...
```

The license file may contain license keys for multiple PDFlib GmbH products on separate lines. It may also contain license keys for multiple platforms so that the same license file can be shared among platforms. License files can be configured in the following ways:

- A file called *licensekeys.txt* is searched in all default locations (see »Default file search paths«, page 9).
- You can specify the *licensefile* option with the *set_option()* API method:

  ```
  tet.set_option("licensefile={/path/to/licensekeys.txt}");
  ```

  The *licensefile* option must be set immediately after instantiating the TET object, i.e., after calling *TET_new()* (in C) or creating a TET object.
- Supply the *--tetopt* option of the TET command-line tool and supply the *licensefile* option with the name of a license file:

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

  If the path name contains space characters you must enclose the path with braces:

  ```
  tet --tetopt "licensefile={/path/to/your license file.txt}" ...
  ```

- You can set an environment (shell) variable which points to a license file. On Windows use the system control panel and choose *System, Advanced, Environment Variables*; on Unix apply a command similar to the following:

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

  On IBM System i the license file can be specified as follows (this command can be specified in the startup program *QSTRUP* and will work for all PDFlib GmbH products):

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

License keys in the registry. On Windows you can also enter the name of the license file in the following registry key:

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

As another alternative you can enter the license key directly in one of the following registry keys:

```
HKLM\SOFTWARE\PDFlib\TET5\license
HKLM\SOFTWARE\PDFlib\TET5\5.6\license
```

The installer will write the license key provided at install time in the last of these entries.

Note *Be careful when manually accessing the registry on 64-bit Windows systems: as usual, 64-bit binaries work with the 64-bit view of the Windows registry, while 32-bit binaries running on a 64-bit system work with the 32-bit view of the registry. If you must add registry keys for a 32-bit product manually, make sure to use the 32-bit version of the* `regedit` *tool. It can be invoked as follows from the* `Start, Run...` *dialog:*

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

Default file search paths. On Unix, Linux, macOS and IBM System i systems some directories is searched for files by default even without specifying any path and directory names. Before searching and reading the UPR file (which may contain additional search paths), the following directories are searched:

```
<rootpath>/PDFlib/TET/5.6/resource/cmap
<rootpath>/PDFlib/TET/5.6/resource/codelist
<rootpath>/PDFlib/TET/5.6/resource/glyphlst
<rootpath>/PDFlib/TET/5.6
<rootpath>/PDFlib/TET
<rootpath>/PDFlib
```

On Unix, Linux, and macOS <*rootpath*> will first be replaced with */usr/local* and then with the HOME directory. On IBM System i <*roothpath*> is empty.

Default file names for license and resource files. By default, the following file names are searched for in the default search path directories:

```
licensekeys.txt            (license file)
tet.upr                    (resource file)
```

This feature can be used to work with a license file without setting any environment variable or runtime option.

Setting the license key in an option for the TET command-line tool. If you use the TET command-line tool you can supply an option which contains the name of a license file or the license key itself:

```
tet --tetopt "license ...your license key..." ...more options...
```

Setting the license key with a TET API call. If you use the TET API you can add an API call to your script or program which sets the license key at runtime:

- In C:

```
TET_set_option(tet, "license=...your license key...");
```

- In C++, .NET/C#, Java, and Ruby:

```
tet.set_option("license=...your license key...");
```

- In Perl, Python and PHP:

```
tet->set_option("license=...your license key...");
```

- In RPG:

```
d licensekey      s             20
d licenseval      s             50
c                   eval      licenseopt='license=... your license key ...'+x'00'
c                   callp     TET_set_option(TET:licenseopt:0)
```

The *license* option must be set immediately after instantiating the TET object, i.e., after calling *TET_new()* (in C) or creating a TET object.

Licensing options. Different licensing options are available for TET use on one or more computers, and for redistributing TET with your own products. We also offer support and source code contracts. Licensing details and the purchase order form can be found in the TET distribution. Please contact us if you are interested in obtaining a commercial license or have any questions.

1 Introduction

The PDFlib Text and Image Extraction Toolkit (TET) is targeted at extracting text and images from PDF documents, but can also be used to retrieve other information from PDF. TET can be used as a base component for realizing the following tasks:

- search the text contents of PDF
- create a list of all words contained in a PDF (concordance)
- implement a search engine for processing large numbers of PDF files
- extract text from PDF to store, translate, or otherwise repurpose it
- convert the text contents of PDF to other formats
- process or enhance PDFs based on their contents
- compare the text contents of multiple PDF documents
- extract the raster images from PDF
- extract metadata and other information from PDF

TET has been designed for stand-alone use, and does not require any third-party software. It is robust and suitable for multi-threaded server use.

1.1 Overview of TET Features

Supported PDF input. TET has been tested against millions of PDF test files from various sources. It accepts PDF 1.0 up to PDF 1.7 extension level 8 and PDF 2.0, corresponding to Acrobat 1-DC including encrypted documents. TET attempts to repair various kinds of malformed and damaged PDF documents.

Note TET does not support XFA forms. XFA is a separate format which is not part of the PDF standard ISO 32000. Since XFA is packaged inside a small PDF wrapper XFA forms are often confused with PDF documents although XFA is a completely different file format which requires dedicated software.

Unicode support. TET includes a considerable number of algorithms and data to achieve reliable Unicode mappings for all text. Since text in PDF documents is not usually encoded in Unicode, TET normalizes the text from a PDF document to Unicode:

- TET converts all text contents to Unicode. In C the text is returned in UTF-8 or UTF-16 format; in other language bindings as native Unicode strings.
- Ligatures and other multi-character glyphs are decomposed into a sequence of their constituent Unicode characters.
- Vendor-specific Unicode values *(Corporate Use Subarea*, CUS) are identified and mapped to characters with precisely defined meanings if possible.
- Glyphs which are lacking Unicode mapping information are identified and mapped to a configurable replacement character.
- UTF-16 surrogate pairs for characters outside the Basic Multilingual Plane (BMP) are interpreted and maintained. Surrogate pairs and UTF-32 values can be retrieved in all language bindings.

Some PDF documents do not contain enough information for reliable Unicode mapping. In order to successfully extract the text nevertheless TET offers various configuration options which can be used to supply auxiliary information for proper Unicode mappings. In order to facilitate writing the required mapping tables we make available

PDFlib FontReporter, a free plugin for Adobe Acrobat. This plugin can be used for analyzing fonts, encodings, and glyphs in PDF.

CJK support. TET includes full support for extracting Chinese, Japanese, and Korean text:

- All predefined CJK CMaps (encodings) are recognized; CJK text is converted to Unicode.
- Special character forms (e.g. wide, narrow, prerotated glyphs for vertical text) can optionally be converted (folded) to the corresponding regular forms
- Horizontal and vertical writing modes are supported.
- CJK font names are normalized to Unicode.

Support for Bidirectional Hebrew and Arabic Text. TET includes the following features for dealing with Bidi text:

- Re-order right-to-left and Bidi text to logical ordering
- Determine dominant text direction of the page
- Normalize Arabic presentation forms and decompose ligatures
- Remove Arabic Tatweel character used for stretching words

Unicode postprocessing. TET's Unicode postprocessing features include the following:

- Folding: preserve, replace, or remove one or more characters; affected characters can conveniently be specified as Unicode sets;
- Decomposition: optionally apply canonical or compatibility decompositions as defined in the Unicode standard. This may make the text better usable in some environments. For example, you can keep or split accented characters, fractions, or symbols like the trademark symbol.
- Normalization: convert the output to Unicode normalization formats NFC, NFD, NFKC, or NFKD as defined in the Unicode standard. This way TET can produce the exact format required as input in some environments, e.g. databases or search engines.

Image extraction. TET extracts raster images from PDF. Adjacent parts of a segmented image are combined to facilitate postprocessing and re-use (e.g. multi-strip images created by some applications). Small images can be filtered in order to exclude tiny image fragments from cluttering the output. If a mask is attached to an image, the mask can be extracted as well.

Images are extracted in TIFF, JPEG, JPEG 2000, or JBIG2 format.

Geometry. TET provides precise metrics for the text, such as the position on the page, glyph widths, and text direction. Specific areas on the page can be excluded or included in the text extraction process, e.g. to ignore headers and footers or margins.

For images the pixel size, physical size, and color space are available as well as position and angle.

Text color. TET provides information about the color of glyphs. The color spaces for filling and stroking and the corresponding color values can be retrieved. A convenient shortcut is available for easily comparing the colors of multiple glyphs without having to deal with the complexities of PDF color spaces.

Word detection and content analysis. TET can be used to retrieve low-level glyph information, but also includes advanced algorithms for high-level content and layout analysis:

- Detect word boundaries to retrieve words instead of characters.
- Recombine the parts of hyphenated words (dehyphenation).
- Remove duplicate instances of text, e.g. shadow and fake bold text.
- Recombine paragraphs into reading order.
- Reorder text which is scattered over the page.
- Reconstruct lines of text.
- Recognize tabular structures on the page.
- Recognize superscript, subscript and drop caps (large initial characters at the start of a paragraph).

TET Markup Language (TETML). The information retrieved from a PDF document can be presented in an XML format called TET Markup Language (TETML) for processing with standard XML tools. TETML contains text, image, and metadata information and can optionally also contain font- and geometry-related details. TETML also contains color and color space information as well as interactive elements such as form fields, annotations, bookmarks, etc.

pCOS interface for simple access to PDF objects. TET includes the pCOS interface *(PDFlib Comprehensive Object System)* for retrieving arbitrary PDF objects. With pCOS you can retrieve PDF metadata, interactive elements (e.g. bookmark text, contents of form fields), or any other information from a PDF document with a simple query interface. The syntax of pCOS query path is described separately in the pCOS Path Reference.

What is text? While TET deals with a large class of PDF documents, in some cases visible text cannot be extracted. The text must be encoded using PDF's text and encoding facilities (i.e., it must be based on a font). Although the following flavors of text may be visible on the page they cannot be extracted with TET:

- Rasterized (pixel image) text, e.g. scanned pages;
- Text which is represented by vector elements without any font.

Note that metadata and text in hypertext elements (such as bookmarks, form fields, notes, or annotations) can be retrieved with TETML or the pCOS interface; see Section 6.1, »PDF Document Domains«, page 69, for details. On the other hand, TET may extract some text which is *not* visible on the page. This may happen in the following situations:

- Text using PDF's *invisible* attribute (however, there is an option to exclude this kind of text from the text retrieval process)
- Text which is obscured by some other element on the page, e.g. an image.

1.2 Many ways to use TET

TET is available as a programming library (component) for various development environments, and as a command-line tool for batch operations. Both offer similar features, but are suitable for different deployment tasks. Both the TET library and command-line tool can create TETML, TET's XML-based output format.

- The TET programming library can be used for integration into your desktop or server application. Many different programming languages are supported. Examples for

using the TET library with all supported language bindings are included in the TET package.

- The TET command-line tool is suited for batch processing PDF documents. It doesn't require any programming, but offers command-line options which can be used to integrate it into complex workflows.
- TETML output is suited for XML-based workflows and developers who are familiar with the wide range of XML processing tools and languages, e.g. XSLT.
- TET connectors are suited for integrating TET in various common software packages, e.g. databases and search engines.
- The TET Plugin is a free extension for Adobe Acrobat which makes TET available for interactive use (see Section 4.1, »Free TET Plugin for Adobe Acrobat«, page 45, for more information).

1.3 Roadmap to Documentation and Samples

Programming samples for the TET library. The TET distribution contains programming examples for all supported language bindings. These samples can serve as a starting point for your own applications, or to test your TET installation. They comprise source code for the following applications:

- The *extractor* sample demonstrates the basic loop for extracting text from a PDF document.
- The *images_per_page* sample extracts the images on each page and reports about their geometry and other properties.
- The *image_resources* sample demonstrates the basic loop for extracting images from a PDF document in a resource-oriented way (no geometric information available).
- The *dumper* sample shows the use of the integrated pCOS interface for querying general information about a PDF document.
- The *fontfilter* sample shows how to process font-related information, such as font name and font size.
- The *glyphinfo* sample demonstrates how to retrieve detailed information about glyphs (font, size, position, etc.) as well as text attributes such as *dropcap*, *shadow*, *hyphenation*, etc. It also shows how to access text color information.
- The *tetml* sample contains code for generating TETML (TET's XML language for expressing PDF contents) from a PDF document.
- The *get_attachments* sample demonstrates how to process PDF file attachments, i.e. PDF documents which are embedded in another PDF document.

XSLT samples. The TET distribution contains several XSLT stylesheets. They demonstrate how to process TETML to achieve various goals:

- *concordance.xsl:* create list of unique words in a document sorted by descending frequency.
- *fontfilter.xsl:* List all words in a document which use a particular font in a size larger than a specified value.
- *fontfinder.xsl:* For all fonts in a document, list all occurrences along with page number and position information.
- *fontstat.xsl:* generate font and glyph statistics.
- *index.xsl:* create an alphabetically sorted »back-of-the-book« index.

- *metadata.xsl:* extract selected properties from document-level XMP metadata included in TETML.
- *solr.xsl:* generate input for the Solr enterprise search server.
- *table.xsl:* Extract a table to a CSV file (comma-separated values).
- *tetml2html.xsl*: convert TETML to HTML.
- *textonly.xsl:* extract the raw text from TETML input.

TET Cookbook. The TET Cookbook is a collection of source code examples for solving specific application problems with the TET library. The Cookbook examples are written in the Java language, but can easily be adjusted to other programming languages since the TET API is almost identical for all supported language bindings. Some Cookbook samples are written in the XSLT language. The TET Cookbook is organized in the following groups:

- Text: samples related to text extraction
- Font: samples related to text with a focus on font properties
- Image: samples related to image extraction
- TET & PDFlib+PDI: samples which extract information from a PDF with TET and construct a new PDF based on the original PDF and the extracted information. These samples require the PDFlib+PDI product in addition to TET.
- TETML: XSLT samples for processing TETML
- Special: other samples

The TET Cookbook is available at the following URL:
www.pdflib.com/tet-cookbook.

pCOS Cookbook. The *pCOS Cookbook* is a collection of code fragments for the pCOS interface which is integrated in TET. It is available at the following URL:
www.pdflib.com/pcos-cookbook.

Details of the pCOS interface are documented in the pCOS Path Reference which is included in the TET package.

1.4 What's new in TET 5.1?

The features below are new or considerably improved in TET 5.1:

- numbered and unnumbered lists are identified and expressed in TETML (with page option *structureanalysis={list=true}*)
- repair mode for damaged input documents with cross-reference streams
- improved workarounds for non-conforming input documents
- improved performance for disabled image, color, and vector engines as well as for documents without layers
- reduced memory requirements
- pCOS interface updated to version 11 with support for certificate security
- other bug fixes
- updated language bindings

1.5 What's new in TET 5.2?

The features below are new or considerably improved in TET 5.2:

- improved table detection with row and column span identification

- mark Artifacts (irrelevant text and images) in TETML and the API
- extract text and images from annotations and patterns
- support for inline images and images in soft masks (graphics state with a Transparency Group XObject)
- new language binding for .NET Core
- enhancements in all language bindings and updates for the latest language versions
- many bug fixes, improvements and workarounds for damaged PDF
- security updates for third-party libraries
- optionally retrieve *Separation* and *DeviceN* text colors in the simpler alternate color space instead of the rather complex native color space
- minor pCOS extensions pCOS interface 12)

1.6 What's new in TET 5.3?

The features below are new or considerably improved in TET 5.3:

- optimized PDF resource handling to improve performance for documents with excessive numbers of images, patterns or other resources
- security and performance updates of all third-party components
- harden processing of damaged and illegal PDF documents by testing the full »Issue Tracker« PDF corpus with tens of thousands of »stressful PDF files«
- expanded platform and CPU support including macOS on ARM64 and Linux on ARM64
- timeout can be specified to limit processing time for large or complex files
- enhancements in all language bindings and updates for the latest language versions including .NET 5, PHP 8, Perl 5.32 and Ruby 3.0
- support for native UTF-8, UTF-16 and UTF-32 Unicode strings in C++17 and C++20
- implement detection of certain kinds of attacks using legal PDF constructs which try to construct overly large data structure
- improved TETML output for edge cases
- improved word boundary, list and paragraph detection
- support for Unicode 13
- improved performance of the Classic .NET binding
- many minor bug fixes and improvements
- updated character collections and CMaps for PDF 2.0

1.7 What's new in TET 5.4?

The features below are new or considerably improved in TET 5.4:

- security and performance updates of third-party components
- enhancements in all language bindings and updates for the latest language versions including .NET 6/7, PHP 8.1/8.2, Perl 5.34/5.36 and Ruby 3.1
- added support for arm64/x86_64 bindings on macOS
- improved TIKA and MediaWiki connectors
- many minor bug fixes and improvements

1.8 What's new in TET 5.5?

The features below are new or considerably improved in TET 5.5:

- security and performance updates of third-party components
- enhancements in all language bindings and updates for the latest language versions including .NET 8, PHP 8.3, Perl 5.38 and Ruby 3.2
- several minor bug fixes and improvements

2 TET Command-Line Tool

2.1 Command-Line Options

The TET command-line tool allows you to extract text and images from one or more PDF documents without the need for any programming. Output can be generated in plain text (Unicode) format or in TETML, TET's XML-based output format. The TET program can be controlled via command-line options. The program inserts space characters (U+0020) after each word, U+000A after each line, and U+000C after each page. It is called as follows for one or more input PDF files:

```
tet [<options>] <filename>...
```

The TET command-line tool is built on top of the TET library. You can supply library options using the *--docopt, --tetopt, --imageopt,* and *--pageopt* options according to the option list tables in Chapter 10, »TET Library API Reference«, page 159. Table 2.1 lists all TET command-line options (this list is also displayed if you run the TET program without any options).

Note *In order to extract CJK text from certain PDF documents you must configure access to the CMap files which are shipped with TET (see Section 0.1, »Installing the Software«, page 7).*

Table 2.1 TET command-line options

option	*parameters*	*function*
--		*End the list of options; this is useful if file names start with a - character.*
@filename[1]		*Specify a response file with options; for a syntax description see »Response files«, page 22. Response files are only recognized before the* `--` *option and before the first filename. Response files cannot be used to replace the parameter for another option, but must contain complete option/parameter combinations.*
--docopt	<*option list*>	*Additional option list for* TET_open_document( ) *(see Table 10.8, page 180). The* `filename` *suboption of the* `tetml` *option cannot be used here.*
--firstpage ***-f***	<*integer*> | `last`	*(Ignored for* `--imageloop resource`*) The number of the page where content extraction will start. The keyword* `last` *specifies the last page,* `last-1` *the page before the last page, etc. Default: 1*
--format	`utf8` | `utf16`	*Specifies the format for text output (default:* `utf8`*):* ***utf8*** *UTF-8 with BOM (byte order mark)* ***utf16*** *UTF-16 in native byte ordering with BOM* *This option does not affect TETML output which is always created in UTF-8.*
--help, -? *(or no option)*		*Display help with a summary of available options.*
--image[2] ***-i***		*Extract images from the whole document (with* `--imageloop resource`*) or the selected pages (with* `--imageloop page`*). The file names of extracted images depend on the* `--imageloop` *option (see below).*

Table 2.1 TET command-line options

option	parameters	function
--imageloop	`page` | `resource`	*Specifies the kind of enumeration for extracting images with the* `--image` *option (default:* `page`*, but forced to* `resource` *if* `--tetml` *is specified):* ***page*** *Extract all images on the selected page(s). Image resources which are placed multiply are extracted multiply.* ***resource*** *Extract all plain and merged image resources in the document. Each image resource is extracted once, regardless of the number of occurrences in the document. Since no size information is available for image resources, a dummy value of 72 dpi is embedded in generated TIFF images.*
--imageopt	<*option list*>	*Additional option list for TET_write_image_file( ) (see Table 10.20, page 206)*
--lastpage ***-l***	<*integer*> | `last`	*(Ignored for* `--imageloop resource`*) The number of the page where content extraction will finish. The keyword* `last` *specifies the last page,* `last-1` *the page before the last page, etc. Default:* `last`
--outfile ***-o***	<*filename*>	*(Not allowed if multiple input file names are supplied) File name for text or TETML output. The file name »-« can be used to designate standard output provided only a single input file has been supplied. Default: name of the input file, with* `.pdf` *or* `.PDF` *replaced with* `.txt` *(for text output) or* `.tetml` *(for TETML output).*
--pagecount		*Print the number of pages in the document, i.e. the value of the pCOS path* `length:pages`*, to* `stdout` *or the file provided with* `--outfile`*.*
--pageopt	<*option list*>	*Additional page option list for TET_open_page( ) if text output is generated, or for TET_process_page( ) if TETML output is generated (see Table 10.10, page 188, and Table 10.21, page 208). For text output the option* `granularity` *is always set to* `page`*.*
--password, ***-p***	<*password*>	*User, master or attachment password for encrypted documents. In some situations the* `shrug` *feature can be used to index protected documents without supplying a password (see Section 5.1, »Extracting Content from protected PDF«, page 59).*
--samedir		*Create output files in the same directory as the input file(s).*
--searchpath[1] ***-s***	<*path*>...	*Name of one or more directories where files (e.g. CMaps) are searched. Default: installation-specific*
--targetdir ***-t***	<*dirname*>	*Output directory for generated text, TETML, and image files. The directory must exist. This option is ignored if* `--samedir` *is specified. Default: . (i.e. the current working directory)*
--tetml ***-m***	`glyph` | `image` | `word` | `wordplus`[3] | `line` | `page`	*(Cannot be combined with* `--text`*) Create TETML output with information about text, images, and interactive elements. TETML is created in UTF-8 format. The supplied parameter selects one of several variants (see Section 9.3, »Controlling TETML Details«, page 139):* ***glyph*** *Glyph-based TETML with glyph geometry and font details* ***image*** *TETML with image information, but without text and interactive elements* ***line*** *Line-based TETML* ***page*** *Page-based TETML* ***word*** *Word-based TETML with word boxes* ***wordplus*** *Word-based TETML with word boxes plus all glyph details* *Attachments, bookmarks and destinations are included in the TETML only if all pages of the documents are processed (see* `--firstpage` *and* `--lastpage`*).*

Table 2.1 TET command-line options

option	*parameters*	*function*
--tetopt	*<option list>*	*Additional option list for TET_set_option( ) (see Table 10.2, page 167). The option* `outputformat` *is ignored (use* `--format` *instead).*
--text[2]		*(Cannot be combined with* `--tetml`*) Extract text from the document (enabled by default)*
--verbose ***-v***	`0 | 1 | 2 | 3`	*verbosity level (default: 1):* ***0*** *no output at all* ***1*** *emit only errors* ***2*** *emit errors and file names* ***3*** *detailed reporting*
--version, -V		*Print the TET version number.*

1. This option can be supplied more than once.
2. The option `--image` *disables text extraction by default, but it can be combined with* `--text` *or* `--tetml`.
3. The option `--tetml wordplus` *internally sets the page option* `tetml={glyphdetails={all}}`. *If the* `tetml` *page option is also supplied with* `--pageopt`, *e.g.* `--pageopt glyphdetails={all}`, *the internal option is overriddedn, i.e. no glyph details are emitted just like in* `word` *mode.*

Image file names. The file names for extracted images depend on the *--imageloop* option. For *--imageloop page* the extracted placed images are named according to the following pattern:

```
<filename>_p<pagenumber>_<imagenumber>.[tif|jpg|jp2|jpf|j2k|jbig2]
```

If an image is designated as Artifact (irrelevant content) the following file name pattern is used:

```
<filename>_p<pagenumber>_<imagenumber>_artifact.[tif|jpg|jp2|jpf|j2k|jbig2]
```

Images which are used as mask for another image are named according to the following pattern *(imagenumber* is the number of the masked image):

```
<filename>_p<pagenumber>_<imagenumber>_mask[_artifact].[tif|jpg|jp2|jpf|j2k|jbig2]
```

For *--imageloop resource* the extracted image resources (including images used as mask for another image) are named according to the following pattern:

```
<filename>_I<imageid>.[tif|jpg|jp2|jpf|j2k|jbig2]
```

Here *imageid* is the index in the *images[ ]* resource array. The same image file name patterns are used in the TETML attribute *Image/@filename*.

2.2 Constructing TET Command Lines

The following rules must be observed for constructing TET command lines:

- Input files are searched in all directories specified as *searchpath*.
- Short forms are available for some options, and can be mixed with long options.
- Long options can be abbreviated provided the abbreviation is unique.
- Depending on the encryption status of the input file, a user or master password may be required for successfully extracting text. It must be supplied with the *--password* option. TET will check whether this password is sufficient for content extraction, and will generate an error if it isn't.

TET checks the full command line before processing any file. If an error is encountered in the options anywhere on the command line, no files are processed at all.

File names. File names which contain blank characters require some special handling when used with command-line tools like TET. In order to process a file name with blank characters you should enclose the complete file name with double quote " characters. Wildcards can be used according to standard practice. For example, **.pdf* denotes all files in a given directory which have a *.pdf* file name suffix. Note that on some systems case is significant, while on others it isn't (i.e., **.pdf* may be different from **.PDF)*. Also note that on Windows systems wildcards do not work for file names containing blank characters. Wildcards are evaluated in the current directory, not any *searchpath* directory.

On Windows all file name options accept Unicode strings, e.g. as a result of dragging files from the Explorer to a command prompt window.

Response files. In addition to options supplied directly on the command-line, options can also be supplied in a response file. The contents of a response file are inserted in the command-line at the location where the *@filename* option was found.

A response file is a simple text file with options and parameters. It must adhere to the following syntax rules:

- Option values must be separated with whitespace, i.e. space, linefeed, return, or tab.
- Values which contain whitespace must be enclosed with double quotation marks: "
- Double quotation marks at the beginning and end of a value are omitted.
- A double quotation mark must be masked with a backslash to use it literally: \"
- A backslash character must be masked with another backslash to use it literally: \\

Response files can be nested, i.e. the *@filename* syntax can itself be used in a response file.

Response files may contain Unicode strings for file name arguments. Response files can be encoded in UTF-8, EBCDIC-UTF-8, or UTF-16 format and must start with the corresponding BOM. If no BOM is found, the contents of the response file are interpreted in EBCDIC on Z, and in ISO 8859-1 (Latin-1) on all other systems.

Exit codes. The TET command-line tool returns with an exit code which can be used to check whether or not the requested operations could be successfully carried out:

- Exit code 0: all command-line options could be successfully and fully processed.
- Exit code 1: one or more file processing errors occurred, but processing continued.
- Exit code 2: some error was found in the command-line options. Processing stopped at the particular bad option, and no input file has been processed.

2.3 Command-Line Examples

The following examples demonstrate some useful combinations of TET command-line options.

2.3.1 Extracting Text

Extract the text from a PDF document *file.pdf* in UTF-8 format and store it in *file.txt:*

```
tet file.pdf
```

Exclude the first and last page from text extraction:

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

Supply a directory where the CJK CMaps are located (required for CJK text extraction):

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

Extract the text from a PDF in UTF-16 format and store it in *file.utf16:*

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

Extract the text from all PDF files in a directory and store the generated **.txt* files in another directory (which must already exist):

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

Extract the text from all PDF files from two directories and store the generated **.txt* files in the same directory as the corresponding input document:

```
tet --samedir dir1/*.pdf dir2/*.pdf
```

Restrict text extraction to a particular area on the page:

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

Use a response file which contains various command-line options and process all PDF documents in the current directory (the file *options* contains command-line options):

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

2.3.2 Extracting Images

Extract images from *file.pdf* in a page-oriented manner and store them in the directory *out:*

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

Extract images from *file.pdf* in a resource-oriented manner and store them in the directory *out:*

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

Extract images from *file.pdf* without image merging; this can be achieved by supplying a list of page options for image processing:

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

2.3.3 Generating TETML

Generate TETML output in word mode for PDF document *file.pdf* and store it in *file.tetml:*

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

Generate TETML output without any *Options* elements; this can be achieved by supplying a suitable list of document options:

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

Generate TETML output in word mode with all glyph details and store it in *file.tetml:*

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

Extract images and generate TETML with text and image information:

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

Extract images and generate TETML image information, but no text:

```
tet --tetml image --image file.pdf
```

Generate TETML output with topdown coordinates:

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

Generate TETML output with improved table detection:

```
tet --tetml word --pageopt "vectoranalysis={structures=tables}" file.pdf
```

2.3.4 Advanced Options

Apply Unicode foldings, e.g. space folding: map all variants of Unicode space characters to *U+0020:*

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

Disable punctuation as word boundary:

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

3 TET Library Language Bindings

Note *It is strongly recommended to take a look at the examples which are contained in all TET packages. They provide a convenient starting point for your own application development and cover many important aspects of TET programming.*

3.1 C Binding

In order to use the C binding you can use a static or shared library (DLL on Windows and MVS), and you need the central TET include file *tetlib.h* for inclusion in your client source modules.

Note *Applications which use the TET binding for C must be linked with a C++ linker since the library includes parts which are implemented in C++. Using a C linker may result in unresolved externals unless the application is linked against the required C++ support libraries.*

Exception handling. The TET API provides a mechanism for acting upon exceptions thrown by the library in order to compensate for the lack of native exception handling in the C language. Using the *TET_TRY( )* and *TET_CATCH( )* macros client code can be set up such that a dedicated piece of code is invoked for error handling and cleanup when an exception occurs. These macros set up two code sections: the try clause with code which may throw an exception, and the catch clause with code which acts upon an exception. If any of the API methods called in the try block throws an exception, program execution will continue at the first statement of the catch block immediately. The following rules must be obeyed in TET client code:

- *TET_TRY( )* and *TET_CATCH( )* must always be paired.
- *TET_new( )* will never throw an exception; since a try block can only be started with a valid TET object handle, *TET_new( )* must be called outside of any try block.
- *TET_delete( )* will never throw an exception, and therefore can safely be called outside of any try block. It can also be called in a catch clause.
- Special care must be taken about variables that are used in both the try and catch blocks. Since the compiler doesn't know about the transfer of control from one block to the other, it might produce inappropriate code (e.g., register variable optimizations) in this situation.
  Fortunately, there is a simple rule to avoid this kind of problem: Variables used in both the *try* and *catch* blocks must be declared *volatile*. Using the *volatile* keyword signals to the compiler that it must not apply dangerous optimizations to the variable.
- If a try block is left (e.g., with a *return* statement, thus bypassing the invocation of the corresponding *TET_CATCH( )*), the *TET_EXIT_TRY( )* macro must be called before the return statement to inform the exception machinery.
- Document processing must stop when an exception was thrown.

The following code fragment demonstrates these rules with the typical idiom for dealing with TET exceptions in client code (a full sample can be found in the TET package):

```
TET *tet;
volatile int pageno;
...
/* Create a new TET object */
```

```
if ((tet = TET_new()) == (TET *) 0)
{
        printf("Couldn't create TET object (out of memory)\n");
        return(2);
}
TET_TRY(tet)
{
        for (pageno = 1; pageno <= n_pages; ++pageno)
        {
            /* process page */
            if (/* error happened */)
            {
                TET_EXIT_TRY(tet);
                return -1;
            }
        }
        /* statements that directly or indirectly call API methods */
}
TET_CATCH(tet)
{
        printf("TET exception %d in %s() on page %d: %s\n",
            TET_get_errnum(tet), TET_get_apiname(tet), pageno, TET_get_errmsg(tet));
}
TET_delete(tet);
```

Volatile variables. Special care must be taken regarding variables that are used in both the *TET_TRY()* and the *TET_CATCH()* blocks. Since the compiler doesn't know about the control transfer from one block to the other, it might produce inappropriate code (e.g., register variable optimizations) in this situation. Fortunately, there is a simple rule to avoid these problems:

Note Variables used in both the TET_TRY() and TET_CATCH() blocks should be declared `volatile`.

Using the *volatile* keyword signals to the compiler that it must not apply (potentially dangerous) optimizations to the variable.

Unicode handling for name strings. The C programming language supports genuine Unicode strings only in version C11. Since this version is not yet widely supported, TET offers Unicode support based on the traditional *char* data type. Some string parameters for API methods may be declared as *name strings*. These are handled depending on the *length* parameter and the existence of a BOM at the beginning of the string. In C, if the *length* parameter is different from 0 the string is interpreted as UTF-16. If the *length* parameter is 0 the string is interpreted as UTF-8 if it starts with a UTF-8 BOM, or as EBCDIC UTF-8 if it starts with an EBCDIC UTF-8 BOM, or as *auto* encoding if no BOM is found (or *ebcdic* on EBCDIC-based platforms).

Unicode handling for option lists. Strings within option lists require special attention since they cannot be expressed as Unicode strings in UTF-16 format, but only as byte arrays. For this reason UTF-8 is used for Unicode options. By looking for a BOM at the beginning of an option TET decides how to interpret it. The BOM is used to determine the format of the string. More precisely, interpreting a string option works as follows:

- If the option starts with a UTF-8 BOM *(\xEF\xBB\xBF)* it is interpreted as UTF-8.
- If the option starts with an EBCDIC UTF-8 BOM *(\x57\x8B\xAB)* it is interpreted as EBCDIC UTF-8.

- If no BOM is found, the string is treated as *winansi* (or *ebcdic* on EBCDIC-based platforms).

Note *The TET_convert_to_unicode( ) utility function can be used to create UTF-8 strings from UTF-16 strings, which is useful for creating option lists with Unicode values.*

Using TET as a DLL loaded at runtime. While most clients will use TET as a statically bound library or a dynamic library which is bound at link time, you can also load the DLL at runtime and dynamically fetch pointers to all API methods. This is especially useful to load the DLL only on demand. TET supports a special mechanism to facilitate this dynamic usage. It works according to the following rules (this is demonstrated in the *extractordl.c* sample):

- Include *tetlibdl.h* instead of *tetlib.h*.
- Use *TET_new_dl( )* and *TET_delete_dl( )* instead of *TET_new( )* and *TET_delete( )*.
- Use *TET_TRY_DL( )* and *TET_CATCH_DL( )* instead of *TET_TRY( )* and *TET_CATCH( )*.
- Use function pointers for all other TET calls.
- Compile the auxiliary module *tetlibdl.c* and link your application against the resulting object file.

The dynamic loading mechanism is demonstrated in the *extractordl.c* sample.

3.2 C++ Binding

An object-oriented wrapper for C++ is available for TET clients. It requires the *tet.hpp* header file which in turn includes *tetlib.h*. The template-driven TET interface supports different string types for passing data from and to TET.

Default interface based on wstring. Strings of the C++ type *std::wstring* are used for all strings in the API. This is the default string type unless one of the types discussed below is selected.

The default interface assumes that all strings passed to and from TET methods are native *wstrings*. Depending on the size of the *wchar_t* data type, *wstrings* are assumed to contain Unicode strings encoded in UTF-16 (2-byte characters) or UTF-32 (4-byte characters). Literal strings in the source code must be prefixed with *L* to designate wide strings.

Use of the default *std::wstring* interface is demonstrated in the *extractor* sample and all other C++ samples.

Optional UTF-8/UTF-16/UTF-32 interfaces based on u8string/u16string/u32string. The *std::u8string* type requires C++20, while the *std::u16string* and *std::u32string* types require at least C++11. As the names imply they store Unicode characters in UTF-8, UTF-16 or UTF-32 format. These interfaces can be activated by setting one of the *TET_CPP_U8STRING, TET_CPP_U16STRING* or *TET_CPP_U32STRING* macros before including *tet.hpp*. Their use is demonstrated in the *extractor_u8string, extractor_u16string* and *extractor_u32string* samples.

Optional UTF-8 interface based on string. All C++ versions support the *std::string* data type. This interface can be activated by setting the *TET_CPP_STRING* macro before including *tet.hpp*. Unlike the *u8string* interface there is no UTF-8 compiler support with this string type. The client code is responsible for properly filling and interpreting strings with UTF-8.

Use of the UTF-8-based *std::string* interface is demonstrated in the *extractor_string* sample.

Note The API descriptions in Chapter 10, »TET Library API Reference«, page 159, use the generic type `String` *as a placeholder for one of the selected string data types.*

Error handling in C++. TET API methods throw a C++ exception in case of an error. These exceptions must be caught in the client code by using C++ *try/catch* clauses. In order to provide extended error information the TET class provides a public *TET::Exception* class which exposes methods for retrieving the detailed error message, the exception number, and the name of the TET API method which threw the exception.

Native C++ exceptions thrown by TET methods behave as expected. The following code fragment catches exceptions thrown by TET:

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

Using TET as a DLL loaded at runtime. The C++ binding allows you to dynamically attach TET to your application at runtime (see »Using TET as a DLL loaded at runtime«, page 27). Dynamic loading can be enabled as follows when compiling the application module which includes *tet.hpp:*

```
#define TETCPP_DL    1
```

In addition you must compile the auxiliary module *tetlibdl.c* and link your application against the resulting object file (this is demonstrated in the *extractordl* sample project). Since the details of dynamic loading are hidden in the TET object it does not affect the C++ API: all method calls look the same regardless of whether or not dynamic loading is enabled.

3.3 Java Binding

Installing the TET Java edition. TET is organized as a Java package with the name *com.pdflib.TET*. This package relies on a native JNI library; both pieces must be configured appropriately.

In order to make the JNI library available the following platform-dependent steps must be performed:

- On Unix systems the library *libtet_java.so* (on macOS: *libtet_java.jnilib)* must be placed in one of the default locations for shared libraries, or in an appropriately configured directory.
- On Windows the library *tet_java.dll* must be placed in the Windows system directory, or a directory which is listed in the PATH environment variable.

The TET Java package is contained in the *TET.jar* file. In order to supply this package to your application, you must add *TET.jar* to your *CLASSPATH* environment variable, add the option *-classpath TET.jar* in your calls to the Java compiler, or perform equivalent steps in your Java IDE. In the JDK you can configure the Java VM to search for native libraries in a given directory by setting the *java.library.path* property to the name of the directory, e.g.

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

You can check the value of this property as follows:

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

Unicode and legacy encoding conversion. For the convenience of TET users we list some useful string conversion methods here. Please refer to the Java documentation for more details.

The following constructor creates a Unicode string from a byte array, using the platform's default encoding:

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

The following constructor creates a Unicode string from a byte array, using the encoding supplied in the *enc* parameter (e.g. *SJIS, UTF8, UTF-16):*

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

The following method of the String class converts a Unicode string to a string according to the encoding specified in the *enc* parameter:

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

Javadoc documentation for TET. The TET package contains Javadoc documentation for TET. The Javadoc contains only abbreviated descriptions of all TET API methods; please refer to Section 10, »TET Library API Reference«, page 159, for more details.

In order to configure Javadoc for TET in Eclipse proceed as follows:

- In the Package Explorer right-click on the Java project and select *Javadoc Location*.
- Click on *Browse...* and select the path where the Javadoc (which is part of the TET package) is located.

After these steps you can browse the Javadoc for TET, e.g. with the *Java Browsing* perspective or via the *Help* menu.

Exception handling. The TET binding for Java throws native Java exceptions of the class *TETException*. TET client code must use standard Java exception syntax:

```
try {
        tet = new TET();
        ...TET instructions...

} catch (TETException e) {
        System.err.println("TET exception occurred:");
        System.err.println("[" + e.get_errnum() + "] " + e.get_apiname() + ": " +
            e.get_errmsg());

} catch (Exception e) {
        System.err.println(e);

} finally {
        if (tet != null) {
            tet.delete();                    /* delete the TET object */
        }
}
```

Since TET declares appropriate *throws* clauses, client code must either catch all possible exceptions or declare those itself.

3.4 .NET Binding

Note The .NET binding is delivered as a universal package for all supported platforms. However, it still requires a platform-specific license key which cannot be transferred across platforms.

Table 3.1 describes general aspects of the .NET binding.

Table 3.1 General properties of the .NET binding

Topic	***Implementation of the .NET binding***
.NET support	*.NET Standard 2.0 and above, e.g. .NET Core 3.1, .NET 6/7*
.NET Framework support	*.NET Framework 4.6.1 and above*
target platforms	*Windows, Linux, macOS; see* `system-requirements.txt` *for exact list of platform/architecture combinations*
download package	*universal* `zip` *package for all supported platforms*
package contents	*NuGet package with managed and native libraries, documentation, and samples*
implementation	*C# assembly* `TET_dotnet.dll` *with managed code and platform-specific shared library* `TET_dotnet_native.[dll|so|dylib]` *with native code*
.NET integration	*C# Interop via explicit PInvoke*
versioning scheme	*The versioning scheme conforms to .NET rules. The .NET version numbers are visible e.g. in the NuGet cache and* `.csproj` *project files. They differ from the product's major and minor release numbers. A mapping between both versioning schemes can be found in* `compatibility.txt`.

Installing TET.NET. The product is supplied as NuGet package where the package name includes the version number. Internal the packages for .NET use semantic version according to .NET rules. As a result, the internal version numbers differ from the regular product version number. A mapping table for both numbering schemes is available in *compatibility.txt*.

The NuGet package can be installed locally using any of the following methods:

- The *dotnet* command-line tool (all platforms). This method is detailed in the next section.
- Visual Studio's Package Manager UI (Windows and macOS)
- Visual Studio's Package Manager Console (Windows)
- The *nuget* command-line tool (all platforms)

The project files for the supplied samples are prepared for target framework .NET 6.0 (target framework moniker TFM=*net6.0)*.

Installing TET for .NET with the dotnet command-line tool. We describe the installation, configuration and build process with the *dotnet* utility, using the supplied *extractor* project as an example:

- Unpack the product package in a directory of your choice.
- In a command shell *cd* to the *extractor* project directory:

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

- (This step is not required for the supplied samples which reference the package with a local *NuGet.Config* file) Copy the NuGet package to the application's project directory:

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

- (This step is not required for the supplied samples which already contain a reference to TET) Enter the following command with the appropriate version number (the version number can be found in the name of the *.nupkg* file):

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

  This command adds a TET reference to the *.csproj* project file. It also installs TET in the local NuGet package cache if it is not yet present, e.g.

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

  Because of this caching you must copy the **.nupkg* only for the first project. Subsequent projects don't require the package file since it is taken from the cache.
- Now you can build and run the *extractor* project to test it:

  ```
  dotnet build
  dotnet run
  ```

  As a result you will find the generated *TET-datasheet.txt* output document in the application directory.

Full examples with ready-to-use configuration are included in all packages. Once the .NET binding is properly referenced you can use the *TET_dotnet*.TET and *TET_dotnet.TETException* classes.

Installing TET.NET in a .NET Framework project. Proceed as follows to add TET.NET to a .NET Framework 4.x project with Visual Studio:

- Add a reference to *TET_dotnet.x.y.z.nupkg* with the NuGet Package manager. As a result a reference to *TET_dotnet* is added to your project as well as a *packages.config* management file.
- In the context menu of *packages.config* choose *Migrate packages.config to PackageReference...* to convert the reference (which initially points only to *TET_dotnet.dll)* to a NuGet package reference.
- In the *Configuration Manager* create a *New Solution Platform* with the name *x86* (32-bit) or *x64* (64-bit). Do not use the setting *Any CPU* since it won't work.
- If you plan to use the built-in IIS the selected Bitness must match the platform; set *x86* or *x64* appropriately via *Project, Properties, Web, Servers*.
- If you change the configuration you must rebuild the project to ensure that the correct version of *TET_dotnet_native.dll* is copied to the *bin* directory.

Error handling in .NET. The .NET binding supports .NET exceptions and will throw an exception with a detailed error message when a runtime problem occurs. The client is responsible for catching such an exception and properly reacting on it. Otherwise the .NET framework will catch the exception and usually terminate the application.

In order to convey exception-related information TET defines its own exception class *TET_dotnet.TETException* with the members *get_errnum*, *get_errmsg*, and *get_apiname*. TET implements the *IDisposable* interface which means that clients can call the *Dispose( )* method for cleanup.

Client code can handle .NET exceptions thrown by PDFlib with the usual *try...catch* construct:

```
try {
        p = new TET();
        ...TET instructions...
} catch (TETException e) {
        // caught exception thrown by TET
        Console.WriteLine("TET exception occurred:\n");
        Console.WriteLine("[{0}] {1}: {2}\n",
            e.get_errnum(), e.get_apiname(), e.get_errmsg());
} finally {
        if (p != null) {
            p.Dispose();
        }
}
```

3.5 Objective-C Binding

Although the C and C++ language bindings can be used with Objective-C, a genuine language binding for Objective-C is also available. The TET framework is available in the following flavors:

- *TET* for use on macOS

Both frameworks contain language bindings for C, C++, and Objective-C.

Installing the TET Edition for Objective-C on macOS. In order to use TET in your application you must copy *TET.framework* to the directory */Library/Frameworks*. Installing the TET framework in a different location is possible, but requires use of Apple's *install_name_tool* which is not described here. The *TET_objc.h* header file with TET method declarations must be imported in the application source code:

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

In order to embed the TET framework in an app XCode's code signing expects a framework with the version number *A* while PDFlib products use numeric version numbers. In order to get around this you can create an appropriately named framework folder as follows:

```
cd TET.framework/Versions
mv 5.6 A
rm Current
ln -s A Current
```

Parameter naming conventions. For TET method calls you must supply parameters according to the following conventions:

- The value of the first parameter is provided directly after the method name, separated by a colon character.
- For each subsequent parameter the parameter's name and its value (again separated from each other by a colon character) must be provided. The parameter names can be found in Chapter 10, »TET Library API Reference«, page 159, and in *TET_objc.h*.

For example, the following line in the API description:

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

corresponds to the following Objective-C method:

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

This means your application must make a call similar to the following:

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

Xcode Code Sense for code completion can be used with the TET framework.

Error handling in Objective-C. The Objective-C binding translates TET exceptions to native Objective-C exceptions. In case of a runtime problem TET throws a native Objective-C exception of the class *TETException*. These exceptions can be handled with the usual *try/catch* mechanism:

```
TET *tet = NULL;

@try {
        tet = [[TET alloc] init];
        ...TET instructions...
}
@catch (TETException *ex) {

        NSLog(@"Error %ld in %@(): %@\n",
            [ex get_errnum], [ex get_apiname], [ex get_errmsg]);
}
@catch (NSException *ex) {

        NSLog(@"%@: %@", [ex name], [ex reason]);
}
@finally {
        if (tet)
            [tet release];
}
```

In addition to the *get_errmsg* method you can also use the *reason* field of the exception object to retrieve the error message.

3.6 Perl Binding

The TET wrapper for Perl consists of a C wrapper and two Perl package modules, one for providing a Perl equivalent for each TET API method and another one for the TET object. The C module is used to build a shared library which the Perl interpreter loads at runtime, with some help from the package file. Perl scripts refer to the shared library module via a *use* statement.

Installing the TET Edition for Perl. The Perl extension mechanism loads shared libraries at runtime through the DynaLoader module. The Perl executable must have been compiled with support for shared libraries (this is true for the majority of Perl configurations).

For the TET binding to work, the Perl interpreter must access the TET Perl wrapper and the modules *tetlib_pl.pm* and *PDFlib/TET.pm*. In addition to the platform-specific methods described below you can add a directory to Perl's *@INC* module search path using the *-I* command line option:

```
perl -I/path/to/tet extractor.pl
```

Unix. Perl will search *tetlib_pl.so* (on macOS: *tetlib_pl.bundle)*, *tetlib_pl.pm* and *PDFlib/TET.pm* in the current directory, or the directory printed by the following Perl command:

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

Perl will also search the subdirectory *auto/tetlib_pl*. Typical output of the above command looks like

```
/usr/lib/perl5/site_perl/5.36/i686-linux
```

Windows. The DLL *tetlib_pl.dll* and the modules *tetlib_pl.pm* and *PDFlib/TET.pm* is searched in the current directory, or the directory printed by the following Perl command:

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

Typical output of the above command looks like

```
C:\Program Files\Perl5.36\site\lib
```

Exception Handling in Perl. When a TET exception occurs, a Perl exception is thrown. It can be caught and acted upon using an *eval* sequence:

```
eval {
        my $tet = new PDFlib::TET;
        ...TET instructions...
};
if ($@) {
        die("$0: TET Exception occurred:\n$@");
}
```

3.7 PHP Binding

Note *Detailed information about the various flavors and options for using TET with PHP, can be found in the PDFlib-in-PHP-HowTo document which is included in the distribution packages and available on the PDFlib Web site. Although it is mainly targeted at using PDFlib with PHP the discussion applies equally to using TET with PHP.*

Installing the TET Edition for PHP. TET is implemented as a C library which can dynamically be attached to PHP. TET supports several versions of PHP. Depending on the version of PHP you use you must choose the appropriate TET library from the unpacked TET archive.

You must configure PHP so that it knows about the TET library. Add the following line in *php.ini:*

```
extension=php_tet
```

PHP searches the library in the directory specified in the *extension_dir* variable in *php.ini* on Unix, and additionally in the standard system directories on Windows. You can test which version of the PHP TET binding you have installed with the following one-line PHP script:

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

This will display a long info page about your current PHP configuration. On this page check the section titled *tet*. If this section contains the phrase

```
PDFlib TET Support          enabled
```

(plus the TET version number) you have successfully installed TET for PHP.

File name handling in PHP. Unqualified file names (without any path component) and relative file names are handled differently in Unix and Windows versions of PHP:

- PHP on Unix systems will find files without any path component in the directory where the script is located.
- PHP on Windows will find files without any path component only in the directory where the PHP DLL is located.

Exception handling in PHP. Since PHP supports structured exception handling, TET exceptions are propagated as PHP exceptions. You can use the standard *try/catch* technique to deal with TET exceptions:

```
try {
        $tet = new TET();
        ...TET instructions...

} catch (TETException $e) {
        print "TET exception occurred:\n";
        print "[" . $e->get_errnum() . "] " . $e->get_apiname() . ": "
                $e->get_errmsg() . "\n";
}
catch (Throwable $e) {
        print $e;
}
```

Developing with Eclipse and Zend Studio. The PHP Development Tools (PDT) support PHP development with Eclipse and Zend Studio. PDT can be configured to support context-sensitive help with the steps outlined below.

Add TET to the Eclipse preferences so that it is known to all PHP projects:

- Select *Window, Preferences, PHP, Source Paths, Libraries, New...* to launch a wizard.
- In *User library name* enter *TET*, click *Add External folder...* and select the folder *bind\php\Eclipse PDT*.

In an existing or new PHP project you can add a reference to the TET library as follows:

- In the PHP Explorer right-click on the PHP project and select *Include Path, Configure Include Path...*
- Go to the *Libraries* tab, click *Add Library...*, and select *User Library*, *TET*.

After these steps you can explore the list of TET methods under the *PHP Include Path/TET/TET* node in the PHP Explorer view. When writing new PHP code Eclipse will assist with code completion and context-sensitive help for all TET methods.

3.8 Python Binding

Installing the TET edition for Python. The Python extension mechanism works by loading shared libraries at runtime. For the TET binding to work, the Python interpreter must have access to the TET Python wrapper which is searched in the directories listed in the PYTHONPATH environment variable. The name of Python wrapper depends on the platform:

- Unix and macOS: *tetlib_py.so*
- Windows: *tetlib_py.pyd*

In addition to the TET library the following files must be available in the same directory where the library sits:

- *PDFlib/TET.py*

Error Handling in Python. The Python binding throws a *TETException* in case of an error. The Python exceptions can be dealt with by the usual try/except technique:

```
try:
        tet = TET()
        ...TET instructions...

except TETException as ex:
        print("Error %d in %s(): %s" % (ex.errnum, ex.apiname, ex.errmsg))

except Exception as ex:
        print(ex)

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

3.9 Ruby Binding

Installing the TET Ruby edition. The Ruby extension mechanism works by loading a shared library at runtime. For the TET binding to work, the Ruby interpreter must have access to the TET extension library for Ruby. This library (on Windows and Unix: *TET.so*; on macOS: *TET.bundle)* will usually be installed in the *site_ruby* branch of the local ruby installation directory, i.e. in a directory with a name similar to the following:

```
/usr/local/lib/ruby/site_ruby/<version>/
```

However, Ruby will search other directories for extensions as well. In order to retrieve a list of these directories you can use the following ruby call:

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

This list will usually include the current directory, so for testing purposes you can simply place the TET extension library and the scripts in the same directory.

Error Handling in Ruby. The Ruby binding installs an error handler which translates TET exceptions to native Ruby exceptions. The Ruby exceptions can be dealt with by the usual *rescue* technique:

```
begin
        tet = TET.new
        ...TET instructions...
rescue TETException => pe
    print pe.backtrace.join("\n") + "\n"
    print "Error [" + 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 is an open-source framework which facilitates Web development with Ruby. The TET extension for Ruby can be used with Ruby on Rails. Follow these steps to run the TET examples for Ruby on Rails:

- Install Ruby and Ruby on Rails.
- Set up a new controller from the command line:

```
$ rails new tetdemo
$ cd tetdemo
$ cp <TET dir>/bind/ruby/<version>/TET.so vendor/  # use .so/.dll/.bundle
$ cp <TET dir>/bind/data/TET-datasheet.pdf .
$ rails generate controller home demo
$ rm public/index.html
```

- Edit *config/routes.rb:*

```
...
# remember to delete public/index.html
root :to => "home#demo"
```

- Edit *app/controllers/home_controller.rb* as follows and insert TET code for extracting PDF contents. As a starting point you can use the code in the *extractor-rails.rb* sample:

```
class HomeController < ApplicationController
def demo
require "TET"
begin
     p = TET.new
     doc = tet.open_document(infilename, docoptlist)
     ...TET application code, see extractor-rails.rb...
     ...
     # and finally show the retrieved text
     send_data text, :type => "text/plain", :disposition => "inline"
     rescue TETException => pe
     # error handling
end
end
end
```

- In order to test your installation start the WEBrick server with the command

```
$ rails server
```

  and point your browser to *http://0.0.0.0:3000*. The text extracted from the PDF document is displayed in the browser.

Local TET installation. If you want to use TET only with Ruby on Rails, but cannot install it globally for general use with Ruby, you can install TET locally in the *vendors* directory within the Rails tree. This is particularly useful if you do not have permission to install Ruby extensions for general use, but want to work with TET in Rails nevertheless.

3.10 RPG Binding

TET provides a */copy* module that defines all prototypes and some useful constants needed to compile ILE-RPG programs with embedded TET functions.

Unicode string handling. Since all TET functions use Unicode strings with variable length as parameters, you have to use the *%ucs2* builtin function to convert a single-byte string to a Unicode string. All strings returned by TET functions are Unicode strings with variable length. Use the *%char* builtin function to convert these Unicode strings to single-byte strings.

Note The %CHAR *and* %UCS2 *functions use the current job's CCSID to convert strings from and to Unicode. The examples provided with TET are based on CCSID 37 (US EBCDIC). This codepage can be set with* `CHGJOB CCSID(37)`. *Some characters in option lists (e.g.* `{ [ ] } )` *may not be translated correctly if you run the examples under other codepages.*

Since all strings are passed as variable length strings you must not pass the *length* parameters in those functions which expect explicit string lengths (the length of a variable length string is stored in the first two bytes of the string).

Compiling and binding RPG programs for TET. Using TET functions from RPG requires the compiled TET service program. To include the TET definitions at compile time you have to specify the name in the *D* specs of your ILE-RPG program:

```
d/copy QRPGLESRC,TETLIB
```

If the TET source file library is not on top of your library list you have to specify the library as well:

```
d/copy tetsrclib/QRPGLESRC,TETLIB
```

Before you start compiling your ILE-RPG program you have to create a binding directory that includes the TETLIB service program shipped with TET. The following example assumes that you want to create a binding directory called TETLIB in the library TETLIB:

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

After creating the binding directory you need to add the TETLIB service program to your binding directory. The following example assumes that you want to add the service program TETLIB in the library TETLIB to the binding directory created earlier.

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

Now you can compile your program using the *CRTBNDRPG* command (or option 14 in PDM):

```
ADDLIBLE LIB(TETLIB)
CRTBNDRPG PGM(TETLIB/EXTRACTOR) SRCFILE(TETLIB/QRPGLESRC) SRCMBR(*PGM) DFTACTGRP(*NO)
BNDDIR(TETLIB/TETLIB)
```

Error Handling in RPG. TET clients written in ILE-RPG can use the *monitor/on-error/endmon* error handling mechanism that ILE-RPG provides. Another way to monitor for exceptions is to use the **PSSR* global error handling subroutine in ILE-RPG. If an excep-

tion occurs, the job log shows the error number, the function that failed and the reason for the exception. TET sends an escape message to the calling program.

```
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 Connectors

TET connectors provide the necessary glue code for interfacing TET with other software. TET connectors are based on the TET library or the TET command-line tool.

4.1 Free TET Plugin for Adobe Acrobat

This section discusses the TET Plugin, a freely available packaging of TET which can be used for testing in Adobe Acrobat and interactive use of TET with any PDF document. The TET Plugin works with Acrobat X-DC Standard, Pro, and Pro Extended (but not Acrobat Reader). It can be downloaded for free from the following location: *www.pdflib.com/products/tet-plugin*.

What is the TET Plugin? The TET Plugin provides simple interactive access to TET. Although the TET Plugin runs as an Acrobat plugin, the underlying content extraction features do not use Acrobat functions, but are completely based on TET. The TET Plugin is provided as a free tool which demonstrates the power of PDFlib TET. Since TET is more powerful than Acrobat's built-in text and image extraction tools and offers a number of convenient user interface features, it is useful as a replacement for Acrobat's built-in copy and find features. PDFlib TET can successfully process many documents for which Acrobat provides only garbage when trying to extract the text. The TET Plugin provides the following functions:

- Copy the text from a PDF document to the system clipboard or a disk file.
- Convert a PDF to TETML and place it on the clipboard or a disk file.
- Copy XMP document metadata to the clipboard.
- Find words in the document.
- Highlight all instances of a search term on the page simultaneously.
- Extract images from the document as TIFF, JPEG, JPEG 2000, or JBIG2 files.
- Display color space and position information for images.
- Detailed configuration settings are available to adjust text and image extraction to your requirements. Configuration sets can be saved and reloaded.

Advantages over Acrobat's copy function. The TET Plugin offers several advantages over Acrobat's built-in copy facility:

- The output can be customized to match different application requirements.
- TET is able to correctly interpret the text in many cases where Acrobat copies only garbage to the clipboard.
- Unknown glyphs (for which proper Unicode mapping cannot be established) are highlighted in red color, and can be replaced with a user-selected character (e.g. question mark).
- TET processes documents much faster than Acrobat.
- Images can be selected interactively for export, or all images on the page or in the document can be extracted.
- Tiny image fragments are merged to usable images.
- Artifacts (irrelevant text and images) in Tagged PDF is highlighted with a dedicated color and can easily be recognized.

4.2 TET Connector for the Lucene Search Engine

Lucene is an open-source search engine. Lucene is primarily a Java project, but a version for .NET is also available. For more information on Lucene see *lucene.apache.org*.

Note *Protected documents can be indexed with the* `shrug` *option under certain conditions (see Chapter 5.1, »Extracting Content from protected PDF«, page 59, for details). This is prepared in the Connector files, but you must manually enable this option.*

Requirements and installation. The TET distribution contains a TET connector which can be used to enable PDF indexing in Lucene Java. We describe this connector for Lucene Java in more detail below, assuming the following requirements are met:

- Java 8 or later for Lucene 8.x.x.
- A working installation of the *Ant* build tool
- The Lucene distribution with the Lucene core JAR file. The Ant build file distributed with TET expects the files *lucene-core-x.x.x.jar, lucene-analyzers-common-x.x.x.jar* and *lucene-queryparser-x.x.x.jar*, which are part of the Lucene distribution.
- An installed TET distribution package for Unix, Linux, macOS, or Windows

In order to implement the TET connector for Lucene perform the following steps with a command prompt:

- Change to the directory *<TET install dir>/connectors/lucene*.
- Copy the files *lucene-core-x.x.x.jar, lucene-analyzers-common-x.x.x.jar* and *lucene-queryparser-x.x.x.jar* to this directory.
- Optionally customize the settings by adding global, document-, and page-related TET options in *TetReader.java*. For example, the global option list can be used to supply a suitable search path for resources (e.g. if the CJK CMaps are installed in a directory different from the default installation).
  The *PdfDocument.java* module demonstrates how to process PDF documents which are stored either on a disk file or in a memory buffer (e.g. supplied by a Web crawler).
- Run the command *ant index*. This will compile the source code and run the indexer on the PDF files contained in the directory *<TET install dir>/bind/data*.
- Run the command *ant search* to start the command-line search client where you can enter queries in the Lucene query language.

Testing TET and Lucene with the command-line search client. The following sample session demonstrates the commands and output for indexing with TET and Lucene, and testing the generated index with the Lucene command-line query tool. The process is started by running the command *ant index:*

```
amira (1)$ ant index
Buildfile: build.xml
...
index:
     [echo] Indexing PDF files in directory "../../bind/data"
     [java] adding ../../bind/data/Whitepaper-Technical-Introduction-to-PDFA.pdf
     [java] adding ../../bind/data/Whitepaper-XMP-metadata-in-PDFlib-products.pdf
     [java] adding ../../bind/data/PDFlib-datasheet.pdf
     [java] adding ../../bind/data/TET-datasheet.pdf
     [java] 662 total milliseconds

BUILD SUCCESSFUL
Total time: 1 second
```

```
amira (1)$ ant search
Buildfile: build.xml

compile:

search:
     [java] Enter query:
PDFlib
     [java] Searching for: pdflib
     [java] 4 total matching documents
     [java] 1. ../../bind/data/PDFlib-datasheet.pdf
     [java]    Title: PDFlib, PDFlib+PDI, Personalization Server data sheet
     [java]    Font : PDFlibLogo-Regular
     [java]    Font : TheSans-Plain
     ...
     [java] 2. ../../bind/data/Whitepaper-XMP-metadata-in-PDFlib-products.pdf
     [java]    Title: Whitepaper: XMP Metadata support in PDFlib products
     [java]    Font : PDFlibLogo-Regular
     [java]    Font : TheSansLight-Italic
     ...
     [java] 3. ../../bind/data/Whitepaper-Technical-Introduction-to-PDFA.pdf
     [java]    Title: Whitepaper: A Technical Introduction to PDF/A
     [java]    Font : PDFlibLogo-Regular
     [java]    Font : TheSansLight-Italic
     ...
     [java] 4. ../../bind/data/TET-datasheet.pdf
     [java]    Title: PDFlib TET datasheet
     [java]    Subject: PDFlib TET  extracts text, images, and metadata from PDF
documents
     [java]    Font : TheSans-Plain
     [java]    Font : PDFlibLogo-Regular
     ...
     [java] Press (q)uit or enter number to jump to a page.
q
     [java] Enter query:
title:XMP
     [java] Searching for: title:xmp
     [java] 1 total matching documents
     [java] 1. ../../bind/data/Whitepaper-XMP-metadata-in-PDFlib-products.pdf
     [java]    Title: Whitepaper: XMP Metadata support in PDFlib products
     [java]    Font : PDFlibLogo-Regular
     [java]    Font : TheSansLight-Italic
     ...
```

Two queries have been performed: one for the word *PDFlib* in the text, and another one for the word *XMP* in the *title* field. Note that *q* must be entered to leave the result paging mode before the next query can be started.

All paths and filenames in the Ant *build.xml* file are defined via properties so that the file can be used with different environments, either by providing the properties on the command line or by entering the properties to override in a file *build.properties*, or even platform-specific into the files *windows.properties* or *unix.properties*. For example, to run the sample with a Lucene JAR file which is installed under */tmp* you can invoke Ant as follows:

```
ant -Dlucene-core.jar=/tmp/lucene-core-x.x.x.jar -Dlucene-analyzers-common.jar=/tmp/
lucene-analyzers-common-x.x.x.jar -Dlucene-queryparser.jar=/tmp/lucene-queryparser-
x.x.x.jar index
```

Indexing metadata fields. The TET connector for Lucene indexes the following metadata fields:

- *path (StringField):* the pathname of the document
- *modified (DateLongField):* the date of the last modification (taken from the PDF file's time-stamp, not the PDF metadata)
- *contents (ReaderTextField):* the full text contents of the document
- All predefined and custom PDF document info entries, e.g. Title, Subject, Author, etc. Document info entries can be queried with the pCOS interface which is integrated in TET (see the pCOS Path Reference for more details on pCOS), e.g.

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

- *font:* the names of all fonts in the PDF document

You can customize metadata fields by modifying the set of indexed document info entries or by adding more information based on pCOS paths in *PdfDocument.java*.

PDF file attachments. The Lucene connector for TET recursively processes all PDF file attachments in a document, and feeds the text and metadata of each attachment to the Lucene search engine for indexing. This way search hits are generated even if the searched text is not present in the main document but some attachment. Recursive attachment traversal is especially important for PDF packages and portfolios.

4.3 TET Connector for the Solr Search Server

Solr is a high performance open-source enterprise search server based on the Lucene search library, with XML/HTTP and JSON/Python/Ruby APIs, hit highlighting, faceted search, caching, replication, and a web admin interface. It runs in a Java servlet container (see *solr.apache.org/solr)*.

Solr acts as an additional layer around the Lucene core engine. It expects the indexed data in a simple XML format. Solr input can most easily be generated based on TETML, the XML flavor produced by TET. The TET connector for Solr consists of an XSLT stylesheet which converts TETML to the XML format expected by Solr. The TETML input for this stylesheet can be generated with the TET library or the TET command-line tool (see Section 9.1, »Creating TETML«, page 133).

Note *Protected documents can be indexed with the* `shrug` *option under certain conditions (see Chapter 5.1, »Extracting Content from protected PDF«, page 59, for details). In order to index protected documents you must enable this option in the TET library or the TET command-line tool when generating the TETML input for Solr.*

Indexing metadata fields. The TET connector for Solr indexes all standard document info fields. The key of each field are used as the field name.

PDF file attachments. The TET connector for Solr recursively processes all PDF file attachments in a document, and feeds the text and metadata of each attachment to the search engine for indexing. This way search hits are generated even if the searched text is not present in the main document but some attachment. Recursive attachment traversal is especially important for PDF packages and portfolios.

XSLT stylesheet for converting TETML. The *solr.xsl* stylesheet expects TETML input in any mode except *glyph*. It generates the XML required to supply input data to the search server. Document info entries are supplied as fields which carry the name of the info entry (plus the *_s* suffix to indicate a string value), and the main text is supplied in a number of text fields. PDF attachments (including PDF packages and portfolios) in the document are processed recursively:

```
<?xml version="1.0" encoding="UTF-8"?><add>
<doc>
<field name="id">TET-datasheet.pdf</field>
<field name="Author_s">PDFlib GmbH</field>
<field name="CreationDate_s">2015-08-04T23:45:46+02:00</field>
<field name="Creator_s">Adobe InDesign CS6 (Windows)</field>
<field name="ModDate_s">2015-08-04T23:45:46+02:00</field>
<field name="Producer_s">Adobe PDF Library 10.0.1</field>
<field name="Subject_s">PDFlib TET: Text and Image Extraction Toolkit (TET)</field>
<field name="Title_s">PDFlib TET datasheet</field>
<field name="text">PDFlib</field>
<field name="text">datasheet</field>
<field name="text">PDFlib</field>
<field name="text">TET</field>
<field name="text">5</field>
...
```

4.4 TET Connector for Oracle

The TET connector for Oracle attaches TET to an Oracle database so that PDF documents can be indexed and queried with Oracle Text. The PDF documents can be referenced via their path name in the database, or directly stored in the database as BLOBs.

Note *Protected documents can be indexed with the* shrug *option under certain conditions (see Chapter 5.1, »Extracting Content from protected PDF«, page 59, for details). This is prepared in the Connector files, but you must manually enable this option.*

Requirements and installation. The TET connector has been tested with Oracle 10i and Oracle 11g. In order use the TET connector you must specify the *AL32UTF8* database character set when creating the database. This is always the case for the Universal edition of Oracle Express (but not for the Western European edition). AL32UTF8 is the database character set recommended by Oracle, and also works best with TET for indexing PDF documents. However, it is also possible to connect TET to Oracle Text with other character sets according to one of the following methods:

- Starting with Oracle Text 11.1.0.7 the database can perform the required character set conversion. Please refer to the section »Using USER_FILTER with Charset and Format Columns« in the Oracle Text 11.1.0.7 documentation, available at *docs.oracle.com/cd/B28359_01/text.111/b28304/cdatadic.htm#sthref497*.
- With Oracle Text 11.1.0.6 or earlier the UTF-8 text generated by the TET filter script must be converted to the database character set. This can be achieved by adding a character set conversion command to *tetfilter.sh:*
  Unix: call *iconv* (open-source software) or *uconv* (part of the free ICU Unicode library)
  Windows: call a suitable code page converter in *tetfilter.bat*.

In order to take advantage of the TET Connector for Oracle you must make the TET filter script available to Oracle as follows:

- Copy the TET filter script to a directory where Oracle can find it:
  Unix: copy *connectors/Oracle/tetfilter.sh* to *$ORACLE_HOME/ctx/bin*
  Windows: copy *connectors/Oracle/tetfilter.bat* to *%ORACLE_HOME%\bin*
- Make sure that the *TETDIR* variable in the TET filter script *(tetfilter.sh* or *tetfilter.bat*, respectively) points to the TET installation directory.
- If required you can supply more TET options for the global, document, or page level in the *TETOPT, DOCOPT,* and *PAGEOPT* variables (see Chapter 10, »TET Library API Reference«, page 159, for option list details). This is especially useful for supplying the TET license key, e.g.:

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

  See Section 0.2, »Applying the TET License Key«, page 8, for more options for supplying the TET license key.

Granting privileges to the Oracle user. The examples below assume an Oracle user with appropriate privileges to create and query an index. The following commands grant appropriate privileges to the user *HR* (these commands must be issued as *system* and must be adjusted as appropriate):

```
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;
```

Example A: Store path names of PDF documents in the database. This example stores file name references to the indexed PDF documents in the database. Proceed as follows:

- Change to the following directory in a command prompt:

  ```
  <TET installation directory>/connectors/Oracle
  ```

- Adjust the *tetpath* variable in the *tetsetup_a.sql* script so that it points to the directory where TET is installed.
- Prepare the database: using Oracle's *sqlplus* program create the table *pdftable_a*, fill this table with path names of PDF documents, and create the index *tetindex_a* (note that the contents of the *tetsetup_a.sql* script are slightly platform-dependent because of different path syntax):

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

- Query the database using the index:

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

- Update the index (required after adding more documents):

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

- Optionally clean up the database (remove the index and table):

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

Example B: Store PDF documents as BLOBs in the database and add metadata. This examples stores the actual PDF documents as BLOBs in the database. In addition to the PDF data some metadata is extracted with the pCOS interface and stored in dedicated database columns. The *tet_pdf_loader* Java program stores the PDF documents as BLOBs in the database. In order to demonstrate metadata handling the program uses the pCOS interface to extract the document title (via the pCOS path */Info/Title)* and the number of pages in the document (via the pCOS path *length:pages)*. The document title and the page count are stored in separate columns in the database. Proceed as follows to run this example:

- Change to the following directory in a command prompt:

  ```
  <TET installation directory>/connectors/Oracle
  ```

- Prepare the database: using Oracle's *sqlplus* program create the table *pdftable_b* and the corresponding index *tetindex_b:*

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

- Populate the database: fill the table with PDF documents and metadata via JDBC (note that this is not possible with stored procedures). The ant build file supplied with the TET package expects the *ojdbc14.jar* file for the Oracle JDBC driver in the same directory as the *tet_pdf_loader.java* source code. Specify a suitable JDBC connection string with the *ant* command. The build file contains a description of all properties that can be used to specify options for the Ant build. You can supply values for

these options on the command line. In the following example we use *localhost* as host name, port number 1521, *xe* as database name, and *HR* as user name and password (adjust as appropriate for your database configuration):

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

- Update the index (required initially and after adding more documents):

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

- Query the database using the index:

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

- Optionally clean up the database (remove the index and table):

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

4.5 TET PDF IFilter for Microsoft Products

This section discusses TET PDF IFilter, which is a separate product built on top of PDFlib TET. More information and distribution packages for TET PDF IFilter are available at *www.pdflib.com/products/tet-pdf-ifilter*.

TET PDF IFilter is freely available for non-commercial desktop use; commercial use on desktop systems and deployment on servers requires a commercial license.

What is PDFlib TET PDF IFilter? TET PDF IFilter extracts text and metadata from PDF documents and makes it available to search and retrieval software on Windows. This allows PDF documents to be searched on the local desktop, a corporate server, or the Web. TET PDF IFilter is based on the patented PDFlib Text Extraction Toolkit (TET), an established developer product for extracting text from PDF documents.

TET PDF IFilter is a robust implementation of Microsoft's IFilter indexing interface. It works with all search and retrieval products which support the IFilter interface, e.g. SharePoint and SQL Server. Such products use format-specific filter programs – called IFilters – for particular file formats, e.g. HTML. TET PDF IFilter is such a program, aimed at PDF documents. The user interface for searching documents may be the Windows Explorer, a Web or database frontend, a query script or a custom application. As an alternative to interactive searches, queries can also be submitted programmatically without any user interface.

Unique Advantages. TET PDF IFilter offers the following advantages:

- Supports Western text, Chinese, Japanese, and Korean (CJK) text and right-to-left languages such as Arabic and Hebrew;
- Text from bookmarks, annotations (comments) and form fields;
- Indexes protected documents and extracts text even from PDFs where Acrobat fails;
- Configurable metadata indexing for document properties;
- Automatic script and language detection for improved search.

Enterprise PDF Search. TET PDF IFilter is available in thread-safe 32- and 64-bit versions. You can implement enterprise PDF search solutions with TET PDF IFilter and all all Microsoft or third-party products which support the IFilter interface including the following:

- Microsoft SharePoint Server
- Microsoft Search Server
- Microsoft SQL Server
- Microsoft Exchange Server
- Microsoft Site Server

Desktop PDF Search. TET PDF IFilter can also be used to implement desktop PDF search with Windows Search which is integrated in Windows. TET PDF IFilter is free for non-commercial use on desktop operating systems, which provides a convenient basis for test and evaluation.

Accepted PDF Input. TET PDF IFilter supports all flavors of PDF input:

- All PDF versions up to Acrobat DC, including ISO 32000-1 and ISO 32000-2.
- Protected PDFs which do not require a password for opening the document.
- Damaged PDF documents are repaired.

Internationalization. In addition to Western text TET PDF IFilter fully supports Chinese, Japanese, and Korean (CJK) text. All CJK encodings are recognized; horizontal and vertical writing modes are supported. Automatic detection of the locale ID (language and region identifier) of the text improves the results of Microsoft's word breaking and stemming algorithms, which is especially important for East Asian text.

Right-to-left languages such as Hebrew and Arabic are also supported. Contextual character forms are normalized and the text is delivered in logical order.

PDF is more than just a Bunch of Pages. TET PDF IFilter treats PDF documents as containers which may contain much more information than only plain pages. TET PDF IFilter indexes all relevant items in PDF documents:

- Page contents
- Text in bookmarks, annotations (comments) and form fields
- Metadata (see below)
- Embedded PDFs and PDF packages/portfolios are processed recursively so that the text in all embedded PDF documents can be searched.

XMP Metadata and document info. The advanced metadata implementation in TET PDF IFilter supports the Windows property system for metadata. It indexes XMP metadata as well as standard or custom document info entries. Metadata indexing can be configured on several levels:

- Document info entries, Dublin Core fields and other common XMP properties are mapped to Windows shell properties, e.g. *Title, Subject, Author*.
- TET PDF IFilter adds useful PDF-specific properties, e.g. page size, PDF/A conformance level, font names.
- All predefined XMP properties can be indexed.
- User-defined XMP or PDF-based properties can be searched, e.g. company-specific classification properties, digital signatures or ZUGFeRD conformance.

TET PDF IFilter optionally integrates metadata in the full text index. As a result, even full text search engines without metadata support (e.g. SQL Server) can search for metadata.

Unicode Postprocessing. TET PDF IFilter supports various Unicode postprocessing steps which can be used to improve the search results:

- Foldings preserve, remove or replace characters, e.g. remove punctuation or characters from irrelevant scripts.
- Decompositions replace a character with an equivalent sequence of one or more other characters, e.g. replace a Chinese character with its canonically equivalent Unicode character.

4.6 TET Connector for the Apache TIKA Toolkit

TIKA is an open-source »toolkit for detecting and extracting metadata and structured text content from various documents using existing parser libraries«. For more information about TIKA see *tika.apache.org*. The TET connector for Tika replaces the default PDF parser configured in Tika and hooks up TET as parser for the PDF format. The TET connector supplies the following items to Tika:

- unformatted text contents of all pages
- predefined and custom document info fields
- number of pages in the document

Note *Protected documents can be indexed with the* `shrug` *option under certain conditions (see Chapter 5.1, »Extracting Content from protected PDF«, page 59, for details). This is prepared in the Connector files, but you must manually enable this option.* `TETPDFParser.java` *additionally provides a method for supplying a password in case the* `shrug` *option is not sufficient.*

Requirements and installation. The TET distribution contains a TET connector for the Tika toolkit. In the description below *<tet-dir>* stands for the directory where the TET package was unpacked. The following requirements must be met:

- JDK 1.5 or later
- A working installation of the *Ant* build tool
- An installed TET distribution package for Unix, Linux, macOS, or Windows.
- A pre-built JAR file for Tika called *tika-app-1.x.jar*. Download information for this file can be found at the following location:
  tika.apache.org/download.html
  In general Tika 1.8 or above can be used. However, Tika 1.9 has a bug which prevents overriding the built-in PDF parser. The TET connector can therefore only be used with Tika 1.9 if some tweaks are applied to the Tika source code, or by using a mechanism like the Tika XML configuration file.

Building and testing the TET connector for Tika. Proceed as follows to build and test the TET connector for Tika:

- Copy *tika-app-1.x.jar* to the directory *<tet-dir>/connectors/Tika*.
- Change to *<tet-dir>/connectors/Tika* and build the TET connector for Tika:

  ```
  ant
  ```

  If your Tika jar file has a name different from *tika-app-1.x.jar* you must supply the name of the jar file on the command line:

  ```
  ant -Dtika-app.jar=tika-app-1.x.jar
  ```

- The build file includes a target for running a test with the TET connector for Tika:

  ```
  ant test
  ```

  This command should produce the contents of the test document as XHTML on the standard output. To test with a PDF file of your choice provide the Ant property *test.inputfile* on the command line as follows:

  ```
  ant -Dtest.inputfile=/path/to/your/file.pdf test
  ```

  The ability to supply a password for protected documents can be tested as follows:

```
ant -Dtest.inputfile=<protected file.pdf> -Dtest.outputfile=<output file name> ↵
    -Dtest.password=<password> api-test
```

- To verify that the TET connector for Tika is actually used for the MIME type *application/pdf*, execute the following command in the directory *<tet-dir>/connectors/Tika* on Unix and macOS systems:

```
java -Djava.library.path=<tet-dir>/bind/java -classpath ↵
    <tet-dir>/bind/java/TET.jar:tika-app-1.x.jar:tet-tika.jar ↵
    org.apache.tika.cli.TikaCLI --list-parser-details
```

  On Windows:

```
java -Djava.library.path=<tet-dir>/bind/java -classpath ↵
    <tet-dir>/bind/java/TET.jar;tika-app-1.x.jar;tet-tika.jar ↵
    org.apache.tika.cli.TikaCLI --list-parser-details
```

  The following fragment should appear in the generated output:

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

- For running the Tika GUI application with the TET connector, execute the following command in the directory *<tet-dir>/connectors/Tika:*
  On Unix and macOS systems:

```
java -Djava.library.path=<tet-dir>/bind/java -classpath ↵
    <tet-dir>/bind/java/TET.jar:tika-app-1.x.jar:tet-tika.jar ↵
    org.apache.tika.cli.TikaCLI
```

  On Windows:

```
java -Djava.library.path=<tet-dir>\bind\java -classpath ↵
    <tet-dir>\bind\java\TET.jar;tika-app-1.x.jar;tet-tika.jar ↵
    org.apache.tika.cli.TikaCLI
```

Customizing the TET connector for Tika. You can customize the Tika connector as follows in the *TETPDFParser.java* source module:

- Add document options to the *DOC_OPT_LIST* variable, e.g. the shrug option for processing protected documents;
- Add page options to the *PAGE_OPT_LIST* variable;
- Customize the searchpath for resources such as CJK CMaps in the *SEARCHPATH* variable. Alternatively, the *tet.searchpath* property can be supplied when processing PDF documents.

4.7 TET Connector for MediaWiki

MediaWiki is the free Wiki software which is used to run Wikipedia and many other community Web sites. More details on MediaWiki can be found at *www.mediawiki.org/wiki/MediaWiki*.

Note *Protected documents can be indexed with the* `shrug` *option under certain conditions (see Chapter 5.1, »Extracting Content from protected PDF«, page 59, for details). This is prepared in the Connector files, but you must manually enable this option.*

Requirements and installation. The TET distribution contains a TET connector which can be used to index PDF documents that are uploaded to a MediaWiki site. MediaWiki does not support PDF documents natively, but allows you to upload PDFs as »images«. The TET connector for MediaWiki indexes all PDF documents as they are uploaded. PDF documents which already exist in MediaWiki are not indexed. The following requirements must be met:

- MediaWiki 1.25 or above
- A TET distribution package with the TET binding for PHP on Unix, Linux, macOS or Windows.

In order to implement the TET connector for MediaWiki perform the following steps:

- Install the TET binding for PHP as described in Section 3.7, »PHP Binding«, page 38.
- Copy the files from the directory *<TET install dir>/connectors/MediaWiki* to *<MediaWiki install dir>/extensions/PDFIndexer*.
- If you need support for CJK text copy the CMap files in *<TET install dir>/resource/cmap* to *<MediaWiki install dir>/extensions/PDFIndexer/resource/cmap*.
- Add the following lines to the MediaWiki configuration file *LocalSettings.php:*

```
# Index uploaded PDFs to make them searchable
wfLoadExtension( 'PDFIndexer' );
```

- To allow the upload of PDF files the file name extension ».pdf« must be enabled as an allowed extension. This is achieved by adding the following line to the MediaWiki configuration file *LocalSettings.php:*

```
$wgFileExtensions[] = 'pdf';
```

How the TET connector for MediaWiki works. The TET connector for MediaWiki consists of the PHP module *PDFIndexer.php*. Using one of MediaWiki's predefined hooks it is hooked up so that it is called whenever a new PDF document is uploaded. It extracts text and metadata from the PDF document and appends it to the optional user-supplied comment which accompanies the uploaded document. The text is hidden in an HTML comment so that it will not be visible to users when they view the document comment. Since MediaWiki indexes the full contents of the comment (including the hidden full text) the text contents of the PDF are also indexed. The text for the index is constructed as follows:

- The TET connector feeds the keys and values of all standard and custom document info fields to the index.
- The text contents of all pages are extracted and concatenated.
- If the size of the extracted text is below a limit, it is completely fed to the index. The advantage of this method is that search results display the search term in context.

- If the size of the extracted text exceeds a limit, the text is reduced to unique words (i.e. multiple instances of the same word are reduced to a single instance of the word).

If the size of the reduced text is below a limit, it is fed to the index. Otherwise it is truncated, i.e. some text towards the end of the document is not indexed. The limit is taken from the *$wgMaxArticleSize* configuration variable which defaults to 2048 KB. If one of the size tests described above hits the limit, a warning message is written to MediaWiki's *DebugLogFile* if MediaWiki logging is activated.

Searching for PDF documents. In order to search PDF documents you must activate the *File* checkbox in the list of namespaces in the *Advanced search* dialog.

The search results will display a list of documents which contain the search term. If the full text has been indexed (as opposed to the abbreviated word list for long documents) some additional terms are displayed before and after the search term to provide context. Since the PDF text contents are fed to the MediaWiki index in HTML form, line numbers are displayed in front of the text. These line numbers are not relevant for PDF documents, and you can ignore them.

Indexing metadata fields. The TET connector for MediaWiki indexes all standard and custom document info fields. The keys and values of each field are fed to the index so that they can be used in searches. Since MediaWiki does not support metadata-based searches you cannot directly search for document info entries, but only for info entries as part of the full text.

5 Configuration

5.1 Extracting Content from protected PDF

PDF security features. PDF documents can be protected with password security which offers the following protection features:

- The user password (also referred to as open password) is required to open the file for viewing.
- The master password (also referred to as owner or permissions password) is required to change any security settings, i.e. permissions, user or master password. Files with user and master passwords can be opened for viewing by supplying either password.
- Permission settings restrict certain actions for the PDF document, such as printing or extracting text.
- An attachment password can be specified to encrypt only file attachments, but not the actual contents of the document itself.

If a PDF document uses any of these protection features it is encrypted. In order to display or modify a document's security settings with Acrobat, click *File, Properties..., Security, Show Details...* or *Change Settings...*, respectively.

TET honors PDF permission settings. The password and permission status can be queried with the pCOS paths *encrypt/master*, *encrypt/user*, *encrypt/nocopy*, etc. as demonstrated in the dumper sample. pCOS also offers the *pcosmode* pseudo object which can be used to determine which operations are allowed for a particular document.

Content extraction status. By default, text and image extraction is possible with TET if the document can successfully be opened (this is no longer true if the *requiredmode* option of *TET_open_document( )* was supplied). Depending on the *nocopy* permission setting, content extraction may or may not be allowed in restricted pCOS mode (content extraction is always allowed in full pCOS mode). The following condition can be used to check whether content extraction is allowed:

```
if ((int) tet.pcos_get_number(doc, "encrypt/nocopy") == 0)
{
        /* content extraction allowed */
}
```

The need for processing protected documents. PDF permission settings help document authors to enforce their rights as creators of content, and users of PDF documents must respect the rights of the document author when extracting text or image contents. By default, TET will operate in restricted mode and refuse to extract any contents from such protected documents. However, content extraction does not in all cases automatically constitute a violation of the author's rights. Situations where content extraction may be acceptable include the following:

- Small amounts of content are extracted for quoting (»fair use«).
- Organizations may want to check incoming or outgoing documents for certain keywords (document screening) without any further content repurposing.
- The document author himself may have lost the master password.

- Search engines index protected documents without making the document contents available to the user directly (only indirectly by providing a link to the original PDF).

The last example is particularly important: even if users are not allowed to extract the contents of a protected PDF, they should be able to locate the document in an enterprise or Web-based search. It may be acceptable to extract the contents if the extracted text is not directly made available to the user, but only used to feed the search engine's index so that the document can be found. Since the user only gets access to the original protected PDF (after the search engine indexed the contents and the hit list contained a link to the PDF), the document's internal permission settings will protect the document as usual when accessed by the user.

The »shrug« feature for protected documents. TET offers a feature which can be used to extract text and images from protected documents, assuming the TET user accepts responsibility for respecting the document author's rights. This feature is called *shrug*, and works as follows: by supplying the *shrug* option to *TET_open_document()* the user asserts that he or she will not violate any document authors' rights. PDFlib GmbH's terms and conditions require that TET customers respect PDF permission settings.

If all of the following conditions are true, the *shrug* feature is enabled:
- The *shrug* option has been supplied to *TET_open_document()*.
- The document requires a master password but it has not been supplied to *TET_open_document()*.
- If the document requires a user (open) password, it must have been supplied to *TET_open_document()*.
- Text extraction is not allowed in the document's permission settings, i.e. *nocopy=true*.

The *shrug* feature will have the following effects:
- Extracting content from the document is allowed despite *nocopy=true*. The user is responsible for respecting the document author's rights.
- The pCOS pseudo object *shrug* is set to *true*/1.
- pCOS runs in full mode (instead of restricted mode), i.e. the *pcosmode* pseudo object is set to 2.

The *shrug* pseudo object can be used according to the following idiom to determine whether or not the contents can directly be made available to the user, or should only be used for indexing and similar indirect purposes:

```
int doc = tet.open_document(filename, "shrug");
...
if ((int) tet.pcos_get_number(doc, "shrug") == 1)
{
        /* only indexing allowed */
}
else
{
        /* content may be delivered to the user */
}
```

5.2 Resource Configuration and File Searching

UPR files and resource categories. In some situations TET needs access to resources such as encoding definitions or glyph name mapping tables. In order to make resource handling platform-independent and customizable, a configuration file can be supplied for describing the available resources along with the names of their corresponding disk files. In addition to a static configuration file, dynamic configuration can be accomplished at runtime by adding resources with *TET_set_option( )*. For the configuration file a simple text format called *Unix PostScript Resource* (UPR) is used. The UPR file format as used by TET will be described below. TET supports the resource categories listed in Table 5.1.

Table 5.1 Resource categories (all file names must be specified in UTF-8)

category	***format*[1]**	***explanation***
cmap	*key=value*	*Resource name and file name of a CMap*
codelist	*key=value*	*Resource name and file name of a code list*
encoding	*key=value*	*Resource name and file name of an encoding*
glyphlist	*key=value*	*Resource name and file name of a glyph list*
glyphmapping	*option list*	*An option list describing a glyph mapping method according to Table 10.9, page 185. This resource is evaluated in TET_open_document( ), and the result is appended after the mappings specified in the option* `glyphmapping` *of TET_open_document( ).*
hostfont	*key=value*	*Name of a host font resource (*`key` *is the PDF font name;* `value` *is the UTF-8 encoded host font name) to be used for an unembedded font*
fontoutline	*key=value*	*Font and file name of a TrueType or OpenType font to be used for an unembedded font*
searchpath	*value*	*Relative or absolute path name of directories containing data files*

1. While the UPR syntax requires an equal character '=' between the name and value, this character is neither required nor allowed when specifying resources with TET_set_option( ).

The UPR file format. UPR files are text files with a very simple structure that can easily be written in a text editor or generated automatically. To start with, let's take a look at some syntactical issues:

- Lines can have a maximum of 255 characters.
- A backslash '\' escapes newline characters. This may be used to extend lines.
- An isolated period character ' . ' serves as a section terminator.
- Comment lines may be introduced with a percent '%' character, and terminated by the end of the line.
- Whitespace is ignored everywhere except in resource names and file names.

UPR files consist of the following components:

- A magic line for identifying the file. It has the following form:

```
PS-Resources-1.0
```

- A section listing all resource categories described in the file. Each line describes one resource category. The list is terminated by a line with a single period character.

- A section for each of the resource categories listed at the beginning of the file. Each section starts with a line showing the resource category, followed by an arbitrary number of lines describing available resources. The list is terminated by a line with a single period character. Each resource data line contains the name of the resource (equal signs have to be quoted). If the resource requires a file name, this name has to be added after an equal sign. The *searchpath* (see below) is applied when TET searches for files listed in resource entries.

Sample UPR file. The following listing gives an example of a UPR configuration file:

```
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
.
```

File search and the searchpath resource category. In addition to relative or absolute path names you can supply file names without any path specification to TET. The *searchpath* resource category can be used to specify a list of path names for directories containing the required data files. When TET must open a file it will first use the file name exactly as supplied, and try to open the file. If this attempt fails, TET will try to open the file in the directories specified in the *searchpath* resource category one after another until it succeeds. Multiple *searchpath* entries can be accumulated, and are searched in reverse order (paths set at a later point in time will searched before earlier ones). In order to disable the search you can use a fully specified path name in the TET functions.

On Windows TET initializes the *searchpath* resource category with a value read from the following registry keys:

```
HKLM\SOFTWARE\PDFlib\TET5\5.6\SearchPath
HKLM\SOFTWARE\PDFlib\TET5\SearchPath
HKLM\SOFTWARE\PDFlib\SearchPath
```

These registry entries may contain a list of path names separated by a semicolon ';' character. The Windows installer initializes the *SearchPath* registry entry with the name of the *resource* directory in the TET installation directory.

Note Be careful when manually accessing the registry on 64-bit Windows systems: as usual, 64-bit binaries work with the 64-bit view of the Windows registry, while 32-bit binaries running on a 64-bit system work with the 32-bit view of the registry. If you must add registry keys for a 32-bit

product manually, make sure to use the 32-bit version of the `regedit` *tool. It can be invoked as follows from the* `Start, Run...` *dialog:*

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

Default file search paths. On Unix, Linux, macOS and IBM System i systems some directories are searched for files by default even without specifying any path and directory names. Before searching and reading the UPR file (which may contain additional search paths), the following directories are searched:

```
<rootpath>/PDFlib/TET/5.6/resource/cmap
<rootpath>/PDFlib/TET/5.6/resource/codelist
<rootpath>/PDFlib/TET/5.6/resource/glyphlst
<rootpath>/PDFlib/TET/5.6/resource/fonts
<rootpath>/PDFlib/TET/5.6/resource/icc
<rootpath>/PDFlib/TET/5.6
<rootpath>/PDFlib/TET
<rootpath>/PDFlib
```

On Unix, Linux, and *macOS* <*roothpath*> will first be replaced with */usr/local* and then with the HOME directory. On IBM System i <*roothpath*> is empty.

Default file names for license and resource files. By default, the following file names are searched for in the default search path directories:

```
licensekeys.txt            (license file)
pdflib.upr                 (resource file)
```

This feature can be used to work with a license file without setting any environment variable or runtime option.

Searching for the UPR resource file. If resource files are to be used you can specify them via calls to *TET_set_option()* (see below) or in a UPR resource file. TET reads this file automatically when the first resource is requested. The detailed process is as follows:

- If the environment variable *TETRESOURCEFILE* is defined TET takes its value as the name of the UPR file to be read. If this file cannot be read an exception is thrown.
- If the environment variable *TETRESOURCEFILE* is not defined, TET tries to open a file with the following name:

  ```
  upr (on MVS; a dataset is expected)
  tet.upr (Windows, Unix, and all other systems)
  ```

  If this file cannot be read no exception is thrown.
- On Windows TET will additionally try to read the following registry entry:

  ```
  HKLM\SOFTWARE\PDFlib\TET5\5.6\resourcefile
  ```

  The value of this key (which is created with the value <*installdir*>*/tet.upr* by the TET installer, but can also be set manually) serves as the name of the resource file to be used. If this file cannot be read an exception is thrown.
- The client can force TET to read a resource file at runtime by explicitly setting the *resourcefile* option:

  ```
  set_option("resourcefile=/path/to/tet.upr");
  ```

This call can be repeated arbitrarily often; the resource entries are accumulated.

Configuring resources at runtime. In addition to using a UPR file for the configuration, it is also possible to directly configure individual resources at runtime via *TET_set_option()*. This function takes a resource category name and pairs of corresponding resource names and values as it would appear in the respective section of this category in a UPR resource file, for example:

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

Multiple resource names can be configured in a single option list for a resource category option (but the same resource category option cannot be repeated in a single call to *TET_set_option()*). Alternatively, multiple calls can be used to accumulate resource settings.

Escape sequences for text files. Escape sequences are supported in all text files except UPR files. Special character sequences can be used to include unprintable characters in text files. All sequences start with a backslash '\' character:

- *x* introduces a sequence of two hexadecimal digits *(0-9, A-F, a-f)*, e.g. *x0D*
- *nnn* denotes a sequence of three octal digits *(0-7), e.g. \015*. The sequence *000* is ignored.
- The sequence \\ denotes a single backslash.
- A backslash at the end of a line will cancel the end-of-line character.

5.3 Recommendations for common Scenarios

TET offers a variety of options which you can use to control various aspects of operation. In this section we provide some recommendations for typical TET application scenarios. Please refer to Chapter 10, »TET Library API Reference«, page 159, for details on the functions and options mentioned below.

Optimizing performance. In some situations, particularly when indexing PDF for search engines, text extraction speed is crucial and may play a more important role than optimal output. The default settings of TET have been selected to achieve the best possible output, but can be adjusted to speed up processing. Some tips for choosing options in *TET_open_page( ) and TET_open_document( )* to maximize text extraction throughput:

- *docstyle=searchengine*
  This page option sets up several internal parameters to speed up operation by reducing processing in a way which does not affect the indexing process for search engines.
- *engines={image=false textcolor=false vector=false}*
  If image extraction and text color detection are not required, internal processing steps can be disabled with this document option to speed up operation. The vector engine is required for clipping calculations and improved table detection
- *contentanalysis={merge=0}*
  This page option disables the expensive strip and zone merging step, and significantly reduces processing times for typical files. However, documents where the contents are scattered across the pages in arbitrary order may result in text which is not extracted in logical order.
- *contentanalysis={shadowdetect=false}*
  This page option disables detection of redundant shadow and fake bold text, which can also reduce processing times.
- When creating TETML the following document option can be used to disable creation of TETML elements for various interactive PDF features:

```
tetml={elements={annotations=false bookmarks=false destinations=false fields=false
javascripts=false}}
```

Words vs. line layout vs. reflowable text. Different applications prefer different kinds of output (hyphenated words are always dehyphenated with these settings):

- Individual words (ignore layout): a search engine may not be interested in any layout-related aspects, but only the words comprising the text. In this situation use *granularity=word* in *TET_open_page( )* to retrieve one word per call to *TET_get_text( )*.
- Keep line layout: use *granularity=page* in *TET_open_page( )* for extracting the full text contents of a page in a single call to *TET_get_text( )*. Text lines are separated with a linefeed character U+000A0 to retain the existing line structure.
- Reflowable text: in order to avoid line breaks and facilitate reflowing of the extracted text use the document option *lineseparator=U+0020* and the page option *granularity=page*. The full page contents can be fetched with a single call to *TET_get_text( )*. By default, paragraphs are separated by U+000A. If you want to apply a different paragraph separator use the document option *paraseparator=U+2029* (or another suitable Unicode value).

Writing a search engine or indexer. Indexers are usually not interested in the position of text on the page (unless they provide search term highlighting). In many cases they will tolerate errors which occur in Unicode mapping, and process whatever text contents they can get. Recommendations:

- Use *granularity=word* in *TET_open_page( )*.
- If the application knows how to process punctuation characters you can keep them with the adjacent text by setting the following page option:
  contentanalysis={punctuationbreaks=false}

Geometry. The geometry features may be useful for some applications:

- The *TET_get_char_info( )* interface is only required if you need the position of text on the page, the respective font name, text color or other details. If you are not interested in text coordinates calling *TET_get_text( )* is sufficient.
- If you have advance information about the layout of pages you can use the *includebox* and/or *excludebox* options in *TET_open_page( )* to get rid of headers, footers, or similar items which are not part of the main text.

Complex layouts. Some classes of documents use very elaborate page layouts. For example, with magazines and periodicals TET may not be able to properly determine the relationship of columns on the page. In such situations it is possible to enhance the extracted text at the expense of processing time. Suitable options for this purpose are summarized in Section 6.7, »Layout Analysis and Document Styles«, page 89. See Table 10.12, page 194, for more details on relevant options.

Legal documents. When dealing with legal documents there is usually zero tolerance for wrong Unicode mappings since they might alter the content or interpretation of a document. In many cases the text position is not required, and the text must be extracted word by word. Recommendations:

- Use the *granularity=word* option in *TET_open_page( )*.
- Use the *password* option with the appropriate document password in *TET_open_document( )* if you must process documents which require a password for opening, or the *shrug* option if content extraction is not allowed in the permission settings and you are in a legal position to extract text from the document (see »The »shrug« feature for protected documents«, page 60).
- For absolute text fidelity: stop processing as soon as the *unknown* field in the character info structure returned by *TET_get_char_info( )* is 1, or if the Unicode replacement character U+FFFD is part of the string returned by *TET_get_text( )*. In TETML with one of the text modes *glyph* or *wordplus* you can identify this situation by the following attribute in the *Glyph* element:

  ```
  unknown="true"
  ```

  Do not set the *unknownchar* option to any common character since you may be unable to distinguish it from correctly mapped characters without checking the *unknown* field.
- Also to ensure text fidelity you may want to disable text extraction for text which is not visible on the page:

  ```
  ignoreinvisibletext=true
  ```

Processing documents with PDFlib+PDI. When using PDFlib+PDI to process PDF documents on a per-page basis you can integrate TET for controlling the splitting or merging process. For example, you could split a PDF document based on the contents of a page. If you have control over the creation process you can insert separator pages with suitable processing instructions in the text. The TET Cookbook contains examples for analyzing documents with TET and then processing them with PDFlib+PDI.

Legacy PDF documents with missing Unicode values. In some situations PDF documents created by legacy applications must be processed where the PDF may not contain enough information for proper Unicode mapping. Using the default settings TET may be unable to extract some or all of the text contents. Recommendations:

- Start by extracting the text with default settings, and analyze the results. Identify the fonts which do not provide enough information for proper Unicode mapping.
- Write custom encoding tables and glyph name lists to fix problematic fonts. Use the PDFlib FontReporter plugin for analyzing the fonts and preparing Unicode mapping tables.
- Configure the custom mapping tables and extract the text again, using a larger number of documents. If there are still unmappable glyphs adjust the mapping tables as appropriate.
- If you have a large number of documents with unmappable glyphs PDFlib GmbH may be able to assist you in creating the required mapping tables.

Convert PDF documents to another format. If you want to import the page contents of PDF documents into your application, while retaining as much information as possible you'll need precise character metrics. Recommendations:

- Use *TET_get_char_info( )* to retrieve precise character metrics and font names. Even if you use the *uv* field to retrieve the Unicode values of individual characters, you must also call *TET_get_text( )* since it fills the *char_info* structure.
- Use *granularity=glyph* or *word* in *TET_open_page( )*, depending on what is better suited for your application. Working with *granularity=glyph* may result in conflicts between the visual layout of text and the processed logical text created by TET (e.g. the two characters created by a ligature glyph may not fit into the same space as the ligature).

Corporate fonts with custom-encoded logos. In many cases corporate fonts containing custom logos include missing or wrong Unicode mapping information for the logos. If you have a large number of PDF documents containing such fonts it is recommended to create a custom mapping table with proper Unicode values.

Start by creating a font report (see »Analyzing PDF documents with the PDFlib FontReporter Plugin«, page 114) for a PDF containing the font, and locate mismapped glyphs in the font report. Depending on the font type you can use any of the available configuration tables to provide the missing Unicode mappings. See »Code list resources for all font types«, page 115, for a detailed example of a code list for a logotype font.

TeX documents. PDF documents produced with the TeX documents often contain numerical glyph names, Type 3 fonts and other problematic properties which prevent other products from successfully extracting the text. TET contains many heuristics and workarounds for dealing with such documents. However, a particular flavor of TeX doc-

uments can only be processed with a workaround that requires more processing time, and is disabled by default. You can enable more CPU-intensive font processing for these documents with the following document option:

```
checkglyphlists=true
```

6 Text Extraction

6.1 PDF Document Domains

PDF documents may contain text in many other places than only the page contents. While most applications deal with the page contents only, in many situations other document domains may be relevant as well.

While the page contents can be retrieved with the workhorse functions *TET_get_text()* and *TET_get_image()*, the integrated pCOS interface plays a crucial role for retrieving text from other document domains.

In the remaining section we provide information on domain searching with the TET library and TETML. In addition, we summarize how to search these document domains with Acrobat. This is important to locate search hits in Acrobat.

Text on the page. Page contents are the main source of text in PDF. Text on a page is rendered with fonts and encoded using one of the many encoding techniques available in PDF.

- How to display with Acrobat: page contents are always visible
- How to search a single PDF with Acrobat DC: *Edit, Find* or *Edit, Advanced Search*. TET may be able to process the text in documents where Acrobat does not correctly map glyphs to Unicode values. In this situation you can use the TET Plugin which is based on TET (see Section 4.1, »Free TET Plugin for Adobe Acrobat«, page 45). The TET Plugin offers its own search dialog via *Plug-Ins, PDFlib TET Plugin..., TET Find*. However, it is not intended as a full-blown search facility.
- How to search multiple PDFs with Acrobat DC: *Edit, Advanced Search* and in S*how More Options* under *Look In:* select *All PDF Documents in*, and browse to a folder with PDF documents (see Figure 6.1).
- Sample code for the TET library: *extractor* sample
- TETML element: */TET/Document/Pages/Page/Content*

Predefined document info entries. Standard document info entries are key/value pairs.

- How to display with Acrobat DC: *File, Properties...*
- How to search a single PDF with Acrobat DC: not available
- How to search multiple PDFs with Acrobat DC: click *Edit, Advanced Search* and*c Show More Options* near the bottom of the dialog. In the *Look In:* pull-down select a folder of PDF documents and in the pull-down menu *Use these additional criteria* select one of *Date Created, Date Modified, Author, Title, Subject, Keywords.*
- Sample code for the TET library: *dumper* sample
- TETML element: */TET/Document/DocInfo*

Custom document info entries. Custom document info entries can be defined in addition to the standard entries.

- How to display with Acrobat DC: *File, Properties..., Custom* (not available in Acrobat Reader)
- How to search with Acrobat DC: not available

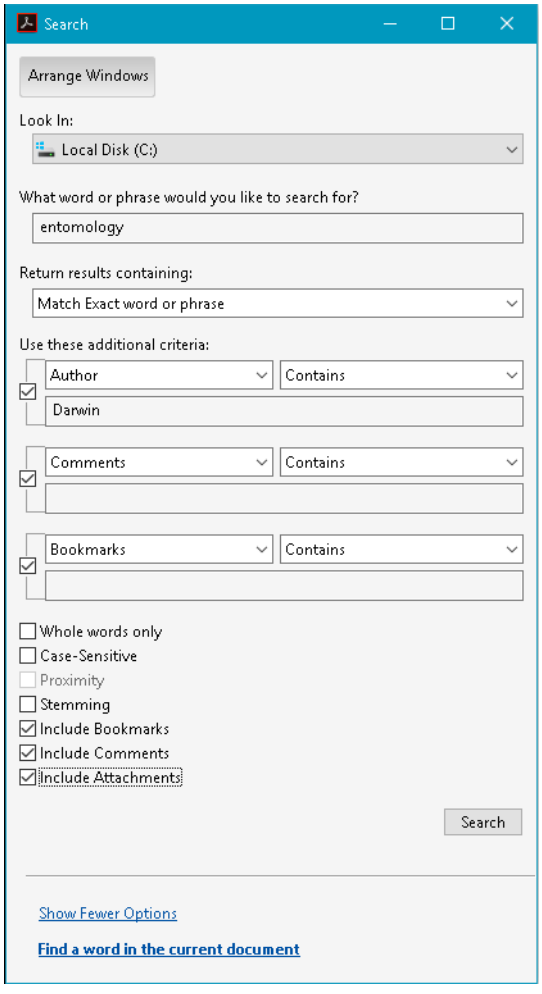


Fig. 6.1
Acrobat's advanced search dialog

- Sample code for the TET library: *dumper* sample
- TETML element: */TET/Document/DocInfo/Custom*

XMP metadata on document level. XMP metadata consists of an XML stream containing extended metadata.

- How to display with Acrobat DC: *File, Properties..., Description, Additional Metadata.*. (not available in Acrobat Reader)
- How to search a single PDF with Acrobat DC: not available
- How to search multiple PDFs with Acrobat DC: click *Edit, Advanced Search* and *Show More Options*. In the *Look In:* pull-down select a folder of PDF documents and in the pull-down menu *Use these additional criteria* select *XMP Metadata* (not available in Acrobat Reader).
- Sample code for the TET library: *dumper* sample
- TETML element: */TET/Document/Metadata*

XMP metadata on image level. XMP metadata can be attached to document components, such as images, pages, fonts, etc. However, XMP is commonly only found on the image level (in addition to document level).

- How to display with Acrobat DC: *View, Show/Hide, Navigation Panes, Content*. Locate the image in the tree structure, right-click on it and select *Show Metadata...* .(not available in Acrobat Reader)
- How to search with Acrobat DC: not available
- Sample code for the TET library: pCOS Cookbook topic *image_metadata*
- TETML element: */TET/Document/Pages/Page/Resources/Images/Image/Metadata*

Text in form fields. Form fields are displayed on top of the page. However, technically they are not part of the page contents, but represented by separate data structures.

- How to display with Acrobat DC: *Tools, Prepare Form* (not available in Acrobat Reader)
- How to search with Acrobat DC: Acrobat searches the visible contents of form fields
- Sample code for the TET library (see Section 6.10, »Annotation Contents«, page 95): annotation appearance streams are processed by default; pCOS Cookbook topic *formfields*
- TETML element: */TET/Document/Pages/Page/Fields/Field/Value* contains the visible value. Additional TETML elements are available for other aspects, e.g. */TET/Document/Pages/Page/Fields/Field/DefaultValue* for the default value, */TET/Document/Pages/Page/Fields/Field/Value* for the interactive tooltip, etc.

Text in comments (annotations). Similar to form fields, annotations (notes, comments, etc.) are layered on top of the page, but are represented by separate data structures. The interesting text contents of an annotation depend on its type. For example, for Web links the interesting part may be the URL, while for other annotation types the visible text contents may be relevant.

- How to display with Acrobat DC: *Tools, Comment, Comments List*
- How to search a single PDF with Acrobat DC: *Edit, Search* and check the box *Include Comments*, or use the *Search Comments* button on the Comments List toolbar
- How to search multiple PDFs with Acrobat DC: click *Edit, [Advanced] Search* and *Show More Options*. In the *Look In:* pull-down select a folder of PDF documents and in the pull-down menu *Use these additional criteria:* select *Comments*.
- Sample code for the TET library (see Section 6.10, »Annotation Contents«, page 95): annotation appearance streams are processed by default; the *Contents* entry of annotations can be extracted with pCOS, see TET Cookbook topic *text_from_annotations*
- TETML element: */TET/Document/Pages/Page/Annotations/Annotation*

Text in bookmarks. Bookmarks are not directly page-related, although they may contain an action which jumps to a particular page. Bookmarks can be nested to form a hierarchical structure.

- How to display with Acrobat DC: *View, Show/Hide, Navigation Panes, Bookmarks*
- How to search a single PDF with Acrobat DC: *Edit, Advanced Search* and check the box *Include Bookmarks*
- How to search multiple PDFs with Acrobat DC: click *Edit, Advanced Search* and *Show More Options*. In the *Look In:* pull-down select a folder of PDF documents and in the pull-down menu *Use these additional criteria* select *Bookmarks* (not available in Acrobat Reader)
- Sample code for the TET library: pCOS Cookbook topic *bookmarks*

- TETML element: */TET/Document/Bookmarks/Bookmark/Title*

File attachments. PDF documents may contain file attachments (on document or page level) which may themselves be PDF documents.

- How to display with Acrobat DC: *View, Show/Hide, Navigation Panes, Attachments*
- How to search with Acrobat DC: Use *Edit, AdvancedSearch* and check the box *Include Attachments* (not available in Acrobat Reader). Nested attachments are not searched recursively.
- Sample code for the TET library: *get_attachments* sample
- TETML element: */TET/Document/Attachments/Attachment/Document*

PDF packages and portfolios. PDF packages and PDF portfolios are file attachments with additional properties.

- How to display with Acrobat DC: Acrobat presents the cover sheet of the package/portfolio and the constituent PDF documents with dedicated user interface elements for PDF packages.
- How to search a single PDF package with Acrobat DC: *Edit, Search Entire Portfolio*
- How to search multiple PDF packages with Acrobat DC: not available
- Sample code for the TET library: *get_attachments* sample
- TETML element: */TET/Document/Attachments/Attachment/Document*

PDF standards and other PDF properties. This domain does not explicitly contain text, but is used as a container which collects various intrinsic properties of a PDF document, e.g. PDF/X and PDF/A status, Tagged PDF status, etc.

- Acrobat DC: *View, Show/Hide, Navigation Panes, Standards* (only present for standard-conforming PDFs)
- How to search with Acrobat DC: not available
- Sample code for the TET library: *dumper* sample
- TETML elements and attributes: */TET/Document/@pdfa, /TET/Document/@pdfe, /TET/Document/@pdfua, /TET/Document/@pdfvt, /TET/Document/@pdfx*

Tagged PDF and Artifacts. TET reconstructs the layout structure and hierarchy directly from the page contents without using the structure tree which is present in Tagged PDF documents. Text and images which are not required to understand the document but rather are generated for layout purposes or as decoration may be marked as Artifacts in Tagged PDF. The most common use of Artifacts is for running headers and footers including page numbers and chapter titles. Depending on the use case it may or may not be desirable to process page contents which are marked as Artifacts:

- How to display with Acrobat DC: *View, Show/Hide, Navigation Panes, Tags;* in the Tags menu click *Find...* and select *Artifacts*. Text, images and vector graphics which are marked as Artifact are highlighted.
  Alternatively, you can activate *Tools, Accessibility, Touch Up Reading Order*. This tool highlights the tagged contents on the page with shaded rectangles. Contents which are not highlighted represents Artifacts.
- How to ignore Artifacts when searching with Acrobat DC: not available
- How to ignore Artifacts with TET: provide the page option *ignoreartifacts*. Text and image artifacts can be identified by the *TET_ATTR_ARTIFACT* flag in the *attributes* field.
- TETML: Text and image Artifacts are identified in TETML with the *artifacts* attribute of the *Glyph*, *Text* and *PlacedImage* elements

Layers. Using layers (technically known as optional content) the page contents can be made visible or invisible. Depending on the use case it may or may not be desirable to process page contents on invisible layers.

- How to display with Acrobat DC: *View*, *Show/Hide*, *Navigation Panes*, *Layers*: layers which are currently visible have an eye symbol in front of the name. Clicking on this symbol controls the visibility of a layer.
- How to search with Acrobat DC: Acrobat searches the contents of all layers. If a search result is found on an invisible layer, Acrobat offers to make the layer visible.
- How to process layers with TET: the page option *layers* can be used to restrict content extraction to either visible or invisible layers. Alternatively, the contents of all layers can be processed which only makes sense if the layers don't overlap.
- TETML: layer contents are processed according to the page option *layers*. Layer names as well as their visibility state and other properties are listed in the TETML element */TET/Document/Pages/Graphics/Layers/Layer*.

6.2 Page and Text Geometry

Default coordinate system. By default TET represents all page and text metrics in the standard coordinate system of PDF. However, the origin of the coordinate system (which could be located outside the page) is adjusted to the lower left corner of the visible page. More precisely, the origin is located in the lower left corner of the *CropBox* if it is present, or the *MediaBox* otherwise. Page rotation is applied if the page has a *Rotate* key. The coordinate system uses the DTP point as unit:

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

The first coordinate increases to the right, the second coordinate increases upwards. By default, all coordinates expected or returned by TET are interpreted in this coordinate system, regardless of their representation in the underlying PDF document. See the pCOS Path Reference to learn how to determine the size of a PDF page.

Top-down coordinate system. Unlike PDF's bottom-up coordinate system some graphics environments use top-down coordinates which may be preferred by some developers. In order to facilitate the use of top-down coordinates TET supports an alternative coordinate system in which all relevant coordinates are interpreted relative to the upper left corner of the page instead of the lower left corner, with *y* coordinates increasing downwards. This *topdown* feature has been designed to make it quite natural for TET users to work in a top-down coordinate system. As an additional advantage, top-down coordinates are identical to the coordinate values displayed in Acrobat (see below). The top-down coordinate system for a page can be activated with the page option *topdown={output}*.

Visualizing coordinates in Acrobat. You can visualize page coordinates in Acrobat as follows (see Figure 6.2):

- To display cursor coordinates in Acrobat DC use *View, Show/Hide, Cursor Coordinates*.
- The coordinates are displayed in the unit which is currently selected in Acrobat. To change the display units to points (as used in TET) in Acrobat DC proceed as follows: go to *Edit, Preferences, Units & Guides, Units* and select *Points*.

Note that the coordinates displayed refer to an origin in the top left corner of the page, and not the default coordinate system of PDF and TET with an origin in the lower left corner. See the previous section for details on selecting a top-down coordinate system which aligns with Acrobat's coordinate display.

Area of text extraction. By default, TET extracts all text from the visible page area. Using the *clippingarea* option of *TET_open_page( )* (see Table 10.10, page 188) you can change this to any of the PDF page box entries (e.g. *TrimBox)*. With the keyword *unlimited* all text regardless of any page boxes can be extracted. The default value *cropbox* instructs TET to extract text within the area which is visible in Acrobat.

The area of text extraction can be specified in more detail by providing an arbitrary number of rectangular areas in the *includebox* and *excludebox* options of *TET_open_page( )*. This is useful for extracting partial page content (e.g. selected columns), or for excluding irrelevant parts (e.g. margins, headers and footers). The final clipping area is

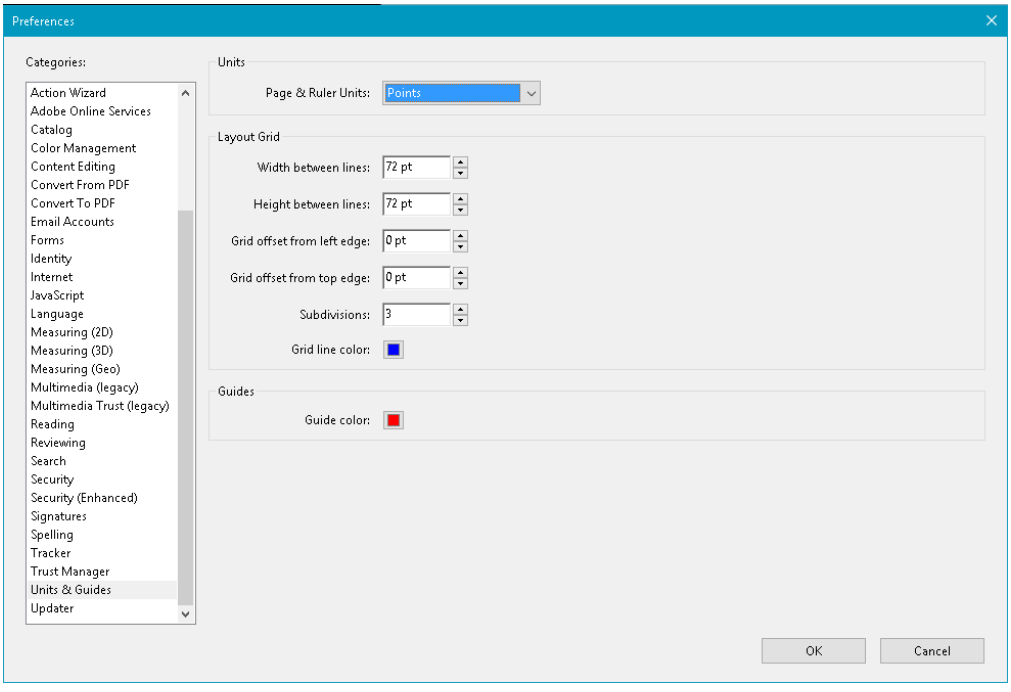


Fig. 6.2
Configuring coordinate display in Acrobat; `View`, `Show/Hide`, `Cursor Coordinates` *displays cursor coordinates.*

constructed by determining the union of all rectangles specified in the *includebox* option, and subtracting the union of all rectangles specified in the *excludebox* option. A glyph is considered inside the clipping area if its reference point is inside the clipping area. This means that a character could be considered inside the clipping area even if parts of it extend beyond the clipping area, or vice versa.

Glyph metrics and other details. Using *TET_get_char_info( )* you can retrieve font and metrics information for the characters which are returned for a particular glyph. The following values are available for each character in the output (see Figure 6.3 and Table 10.16):

- The *uv* value contains the UTF-32 Unicode value of the current character, i.e. the character for which details are retrieved. This field always contains UTF-32, even in language bindings that support only UTF-16 in their native Unicode strings. Accessing the *uv* field allows applications to deal with characters outside the BMP without having to interpret surrogate pairs.
- The *type* field specifies how the character was created; it is filled with the constants *TET_CT_NORMAL* etc. There are two groups: real and artificial characters. The group of real characters comprises normal characters (i.e. the complete result of a single glyph) and characters which start a multi-character sequence that corresponds to a single glyph (e.g. the first character of a ligature). The group of artificial characters comprises the continuation of a multi-character sequence (e.g. the second character of a ligature) and inserted separator characters. For artificial characters the position *(x, y)* specifies the endpoint of the most recent real character, *width* and *height* are 0, and all other fields except *uv* are those of the most recent real character. The end-

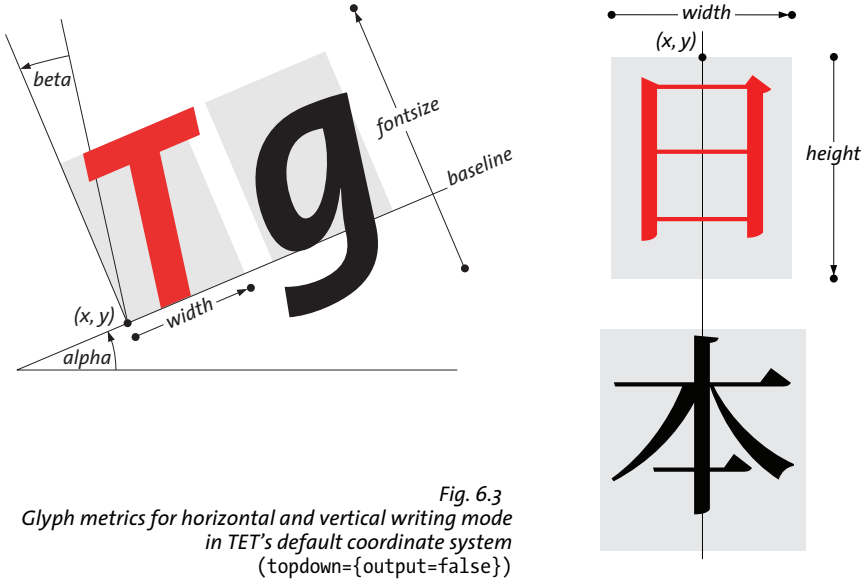


Fig. 6.3
Glyph metrics for horizontal and vertical writing mode in TET's default coordinate system
(`topdown={output=false}`)

point is the point *(x, y)* plus the *width* added in direction *alpha* (in horizontal writing mode) or plus the *height* in direction -90° (in vertical writing mode).

- The *unknown* field is usually *false* (in C and C++: 0), but has a value of *true* (in C and C++: 1) if the original glyph could not be mapped to Unicode and has therefore been replaced with the character specified in the *unknownchar* option. Using this field you can distinguish real document content from replaced characters if you specified a common character as *unknownchar*, such as a question mark or space.
- The *attributes* field contains information about the subscript, superscript, dropcap, or shadow status of the glyph as determined by TET's content analysis algorithms. If the glyph is part of an Artifact (irrelevant content) this is also reported in the *attributes* field. This field is populated with the constants *TET_ATTR_SUB* etc.
- The *(x, y)* fields specify the position of the glyph's reference point, which is the lower left corner of the glyph rectangle in horizontal writing mode, and the top center in vertical writing mode (see Section 6.4, »Chinese, Japanese, and Korean Text«, page 82 for details on vertical writing mode). For artificial characters, which do not correspond to any glyph on the page, the point *(x, y)* specifies the end point of the most recent real character. The value of *y* is subject to the *topdown* page option.
- The *width* field specifies the width of a glyph according to the corresponding font metrics and text output parameters, such as character spacing and horizontal scaling. Since these parameters control the position of the next glyph, the distance between the reference points of two adjacent glyphs may be different from *width*. The *width* may be zero for non-spacing characters. On the other hand, the outline may actually be wider than the glyph's *width* value, e.g. for slanted text.
  The *width* is 0 for artificial characters.
- The *height* field in vertical writing mode specifies the height of the corresponding glyph according to the font metrics and text parameters (e.g. character spacing). The height is positive in the default coordinate system, but negative for *topdown* coordinates. In monospaced vertical fonts all glyphs have *fontsize* as height unless additional character spacing has been applied. Artificial characters (e.g. separators) have a height of 0.

For horizontal writing mode an approximation of the glyph height is supplied. This approximate value is derived from font properties and therefore identical for all glyphs in a font. There is no guarantee that the visible glyph has the exact height value supplied here.

- The angle *alpha* provides the direction of text progression, specified as the deviation from the standard direction. The standard direction is 0° for horizontal writing mode, and -90° for vertical writing mode (see below for more details on vertical writing mode). Therefore, the angle *alpha* is 0° for standard horizontal text as well as for standard vertical text. The values of *alpha* and *beta* are subject to the *topdown* page option.
- The angle *beta* specifies any skewing which has been applied to the text, e.g. for slanted (italicized) text. The angle is measured against the perpendicular of *alpha*. It is 0° for standard upright text (for both horizontal and vertical writing mode). If the absolute value of *beta* is greater than 90° the text is mirrored at the baseline.
- The *fontid* field contains the pCOS ID of the font used for the glyph. It can be used to retrieve detailed font information, such as the font name, embedding status, writing mode (horizontal/vertical), etc. The pCOS Path Reference contains sample code for retrieving font details.
- The *fontsize* field specifies the size of the text in points. It is normalized and therefore always positive, even for *topdown={output}*.
- The *colorid* field contains an index for the text color. It represents the unique combination of fill color, stroke color, and text rendering. All occurrences of the same combination in a document are represented by the same color id. Different combinations are represented by different ids, which means that colors of multiple glyphs can be checked for equality by comparing their color ids. For example, by comparing the *colorid* values of successive glyphs you can identify changes in text color. The exact color space and color components for filling and/or stroking text can be retrieved with *TET_get_color_info( )* (see Section 6.3, »Text Color«, page 80).
- The *textrendering* field specifies the kind of rendering for a glyph, e.g. stroked, filled, or invisible, and possible use of the text as clipping path. It is filled with the constants *TET_TR_FILL* etc. This field contains the numerical text rendering mode as defined in PDF (see Table 10.16, page 200). Invisible text (i.e. *textrendering=3)* is extracted by default, but this can be changed with the *ignoreinvisibletext* option of *TET_open_page( )*.
  Text in Type 3 fonts: *textrendering=3* and *7* result in invisible text; all other values of *textrendering* are irrelevant and are ignored.

Font-specific metrics. TET uses the glyph and font metrics system used by PostScript and PDF which shall be briefly discussed here.

The font size is usually chosen as the minimum distance between adjacent text lines which is required to avoid overlapping character parts. The font size is generally larger than individual characters in a font, since it spans ascender and descender, plus possibly additional space between lines.

The *capheight* is the height of capital letters such as *T* or *H* in most Latin fonts. The *xheight* is the height of lowercase letters such as *x* in most Latin fonts. The *ascender* is the height of lowercase letters such as *t* or *d* in most Latin fonts. The *descender* is the distance from the baseline to the bottom of lowercase letters such as *j* or *p* in most Latin

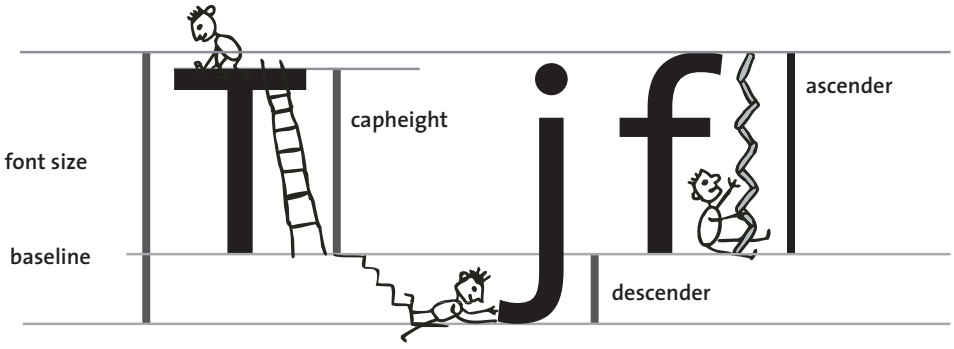


Fig. 6.4 Font-specific metrics

fonts. The descender is usually negative. The values of *xheight, capheight, ascender,* and *descender* are measured in thousands of the font size.

These values vary among fonts, and can be retrieved with the pCOS interface. For example, the following code retrieves the ascender and descender values:

```
/* Query ascender and descender values */
path = "fonts[" + i + "]/ascender";
System.out.println("Ascender=" + p.pcos_get_number(doc, path));

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

Note that *ascender* and other font metrics values should only be queried after calling *TET_get_char_info( )* for a glyph with this font. In order words, using font ids returned by *TET_get_char_info( )* is safe, while enumerating all fonts in the *fonts[ ]* array does not necessarily provide metrics values from embedded font data, but the possibly inaccurate values from the PDF *FontDescriptor* dictionary. For more information refer to the pCOS Path Reference.

End points of glyphs and words. In order to do proper highlighting you need the end position of the last character in a word. Using *x, y, width,* and *alpha* returned by *TET_get_char_info( )* you can determine the end point of a glyph in horizontal writing mode, i.e. the end point of the glyph's advance vector (the lower right corner of the glyph box):

$x_{end} = lr_x = x + width * cos(alpha)$
$y_{end} = lr_y = y + width * sin(alpha)$

In the common case of horizontally oriented text (i.e. *alpha*=0) this reduces to

$x_{end} = lr_x = x + width$
$y_{end} = lr_y = y$

More generally, you can calculate the size of the glyph box by determining the coordinates of the upper right corner (for *beta=0*, i.e. this formula does not take into account glyph skewing):

$ur_x = x + width * cos(alpha) - dir * height * sin(alpha)$
$ur_y = y + width * sin(alpha) + dir * height * cos(alpha)$

with *dir=1* in the default case *topdown={output=false}* and *dir=-1* if *topdown={output=true}* (see »Top-down coordinate system«, page 74). The value of *height* depends on the fontsize and the font geometry. The following results in useful values for most common fonts (see »Font-specific metrics«, page 77, for retrieving the *ascender* value):

```
height = fontsize * ascender / 1000
```

In many graphical development environments the glyph transformations can be expressed as follows:

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

After applying these transformations the upper right corner of the glyph box can be expressed as follows:

$$ur_x = x + width$$
$$ur_y = y + dir * height$$

Glyph calculations for vertical writing mode. For text with vertical writing mode the end point calculation works as follows:

$$x_{end} = x$$
$$y_{end} = y - height$$

The upper left and lower right corners of the glyph box can be calculated as follows (for *beta=0):*

$$ul_x = x - width/2 * cos(alpha)$$
$$ul_y = y - width/2 * sin(alpha)$$

$$lr_x = ul_x + width * cos(alpha) + dir * height * sin(alpha)$$
$$lr_y = ul_y + width * sin(alpha) - dir * height * cos(alpha)$$

with *dir=1* in the default case *topdown={output=false}* and *dir=-1* if *topdown={output=true}* (see »Top-down coordinate system«, page 74).

6.3 Text Color

The text color id returned by *TET_get_char_info( )* describes the fill and/or stroke color of the glyph corresponding to a character. The fill and stroke colors represented by a color id can be achieved with *TET_get_color_info( )* which returns the following values for a color id. These values can be retrieved separately for the fill and stroke color of a glyph:

- The *colorspaceid* field contains the index of the color space in the *colorspaces[ ]* pseudo object (see the pCOS Path Reference), or -1 if no color is applied to the glyph.
- The *patternid* field contains the index of the pattern in the *patterns[ ]* pseudo object (see the pCOS Path Reference), or -1 if no pattern is applied to the glyph.
- The *components* array contains the color values which must be interpreted in the color space reported with *colorspaceid*.
- The *n* field (only available in the C and C++ language bindings) contains the number of relevant entries in the *components* field.

The *glyphinfo* sample demonstrates how to interpret the color values provided by *TET_get_color_info( )* and how to augment this information with general color space attributes retrieved with pCOS. The *colorspaces* and *page_colorspaces* topics in the pCOS Cookbook demonstrate how to retrieve even more color space details, such as *WhitePoint* for calibrated color spaces or the alternate color space of a *Separation* or *DeviceN* color space.

Text stroking, i.e. painting the outline of glyphs (as opposed to filling the interior) is rarely used in PDF documents. Most applications may ignore the stroke color information. Also, patterns are rarely used for text. In some cases glyphs don't carry any fill color nor stroke color information, e.g. invisible text *(textrendering=3)*. This case can be identified by *colorspaceid=-1*.

Text color retrieval can be disabled with the following document option:

```
engines={notextcolor}
```

If the text color engine is disabled, the *colorid* field of *TET_char_info* must not be used since it doesn't contain any meaningful value.

Table 6.1 provides an overview of PDF color spaces. Unless noted otherwise, color values are in the range 0..1.

Table 6.1 Color spaces in PDF

color space	*number of color components*	*notes*
Device color spaces		
DeviceGray DeviceRGB DeviceCMYK	*1* *3* *4*	*The device color spaces are widely known, but are device-dependent and therefore don't represent reliable color information.*
CIE-based (device-independent) color spaces		
ICCBased	*1, 3 or 4*	ICCBased *color spaces are defined by an ICC profile for grayscale, RGB or CMYK color.*
Lab	*3*	Lab *color spaces are defined by an CIE 1976 L*a*b* space. They require a lightness value in the range 0...100 and two color values which are often in the range -128...127.*
CalGray CalRGB	*1* *3*	*Calibrated color spaces define a* WhitePoint *and optional* BlackPoint. *They are rarely used since* ICCBased *color spaces are more flexible.*
Special color spaces		
Pattern	*0* (PaintType=1) N (PaintType=2) *0* (PatternType=2)	Pattern *color spaces are used to apply some graphical pattern instead of a solid color. Tiling patterns* (PatternType=1) *colorize by repeatedly placing some graphical shape, where the shape may be colored with intrinsic colors* (PaintType=1), *or may be uncolored like a stencil mask and require external color* (PaintType=2). *Shading patterns* (PatternType=2) *apply a color gradient instead of solid color.*
Separation[1]	*1*	*A* Separation *color space describes a named spot color and requires an alternate color space which is needed if the named spot color is not directly available for output.*
DeviceN[1]	N	DeviceN *is a generalization of* Separation *color space for more than one named spot color. It is also used to apply a subset of CMYK process colors.*
Indexed	*1, but* N *in the base color space*	Indexed *color spaces allow for efficient storage of a small number of different color values (up to 256) and require an underlying base color space.*

1. The document option glyphcolor=alternate *can be used to report text colors in the alternate color space instead of* Separation *or* DeviceN.

6.4 Chinese, Japanese, and Korean Text

6.4.1 CJK Encodings and CMaps

TET supports Chinese, Japanese, and Korean (CJK) text, and converts horizontal and vertical CJK text in arbitrary legacy encodings (CMaps) to Unicode. TET supports the following CJK character collections and supplements:

- Simplified Chinese: *Adobe-GB1-5*
- Traditional Chinese: *Adobe-CNS1-7*
- Japanese: *Adobe-Japan1-7*
- Korean: *Adobe-Korea1-2* and *Adobe-KR-9*

The PDF CMaps in turn cover a variety of CJK character encodings such as Shift-JIS, EUC, Big-5, KSC, and many others. CJK font names encoded with locale-specific encodings (e.g. Japanese font names encoded in Shift-JIS) are normalized to Unicode.

Note *In order to extract CJK text which is encoded with legacy encodings you must configure access to the CMap files which are shipped with TET according to Section 0.1, »Installing the Software«, page 7. If a required CMap is not available TET_open_page( ) returns an error since the configuration must be corrected.*

6.4.2 Word Boundaries for CJK Text

Ideographic characters don't constitute a word boundary, but punctuation and the transition between ideographic and non-ideographic characters still constitute word boundaries. For *granularity=word* ideographic comma *U+3001* and ideographic full stop *U+3002* also constitute word boundaries. For *granularity=page* no line separator is inserted at the end of a line containing only ideographic characters.

6.4.3 Vertical Writing Mode

TET supports both horizontal and vertical writing modes, and performs all metrics calculations as appropriate for the respective writing mode. Keep the following in mind when dealing with text in vertical writing mode:

- The glyph reference point in vertical writing mode is at the top center of the glyph box. The text position advances downwards as determined by the glyph height, regardless of the glyph width (see Figure 6.3).
- The angle *alpha* is 0° for standard vertical text. In other words, fonts with vertical writing mode and *alpha=0°* progress downwards, i.e. in direction -90°.
- Because of the differences noted above, client code must take the writing mode into account by using the following pCOS code (note that not all text which appears vertically actually uses a font with vertical writing mode):

```
count = p.pcos_get_number(doc, "length:fonts");
for (i=0; i < count; i++)
{
     if (p.pcos_get_number(doc, "fonts[" + id + "]/vertical"))
     {
         /* font uses vertical writing mode */
     }
}
```

- Prerotated glyphs for vertical text and punctuation are mapped to the corresponding unrotated Unicode characters. Use the following document option to preserve prerotated characters:

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

6.4.4 CJK Decompositions: Narrow, wide, vertical, etc.

Unicode and many legacy encodings support the notion of fullwidth and halfwidth characters (sometimes also called double-byte and single-byte characters). By default, TET applies the Unicode decompositions *wide* and *narrow* which replace fullwidth and halfwidth characters with the corresponding standard-width counterparts.

In order to preserve the original fullwidth and halfwidth characters you can use the *decompose* document option and disable the respective decompositions:

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

Similarly, the *small, square,* and *vertical* decompositions also affect CJK characters. Since all these decompositions (including wide and narrow) are enabled by default, the characters are converted to their normal counterparts. Disable the respective decompositions in order to preserve the original characters. The following document option disables all decompositions:

```
decompose={none}
```

Table 6.2 demonstrates the CJK decompositions along with examples. See Section 7.3.2, »Unicode Decomposition«, page 106, for more information on decompositions.

Table 6.2 CJK compatibility decomposition examples (suboptions for the `decompose` *option)*

decomposition name	*description*	*affected Unicode characters*	*decompositions enabled (default)*	*decompositions disabled*
narrow	*Narrow (hankaku) compatibility forms*	*U+FF61-U+FFDC, U+FFE8-U+FFEE*	ヲ U+30F2	ｦ U+FF66
small	*Small forms for CNS 11643 compatibility*	*U+FE50-U+FE6B*	, U+002C	﹐ U+FE50
square	*CJK squared font variants*	*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*	キ 口 U+30AD U+30ED	㌔ U+3314
vertical	*Vertical layout presentation forms*	*U+309F, U+30FF, U+FE10-U+FE19 U+FE30-U+FE48*	⁀ U+FE37	｛ U+007B
wide	*Wide (zenkaku) compatibility forms*	*U+3000, U+FF01-U+FF60, U+FFE0-U+FFE6*	£ U+00A3	£ U+FFE1

6.5 Bidirectional Arabic and Hebrew Text

TET applies additional processing to correctly extract text from documents with right-to-left scripts such as Arabic and Hebrew. Since these scripts often contain left-to-right text inserts (e.g. numbers), such documents are called bidirectional. Extracting bidirectional text involves one or more of the processing steps mentioned below.

6.5.1 General Bidi Topics

Reorder right-to-left and bidirectional text. Right-to-left sequences and left-to-right sequences must be reordered to form the correct sequence of logical text. In granularity word or higher TET delivers text in logical order with the following page option (which is the default setting):

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

Bidi processing can explicitly be disabled with the following page option:

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

Determine the dominant text direction of the page. Not only the characters within a word and words within a line are affected by Bidi reordering, but also other aspects of page layout recognition. In some cases mixed Bidi lines cannot safely be reordered without taking into account the fact that the page is an overall right-to-left or left-to-right page. In order to make this decision automatically TET checks the dominant text direction of the page and adjusts its algorithms depending on whether the page must be considered mostly left-to-right or mostly right-to-left.

This decision can be overridden with the *bidilevel* option. For example, the following option list forces right-to-left handling even on pages where the majority of text runs left-to-right:

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

Glyph ordering. The glyph information returned by *TET_get_char_info( )* and the *Glyph* elements in TETML are always ordered according to visual order, i.e. from left to right for horizontal baselines. This left-to-right glyph ordering ensures that client applications receive glyph coordinates in deterministic ordering without having to check the Bidi status of the text. This behavior reflects the fact that the glyphs in Arabic and Hebrew fonts generally have the reference point at the left edge and advance to the right, despite the fact that the actual text direction is right-to-left.

6.5.2 Postprocessing Arabic Text

Normalize Arabic presentation forms and decompose ligatures. Arabic characters exist in up to four different forms for isolated use, at the beginning, in the middle, or at the end of a word. These forms can have different Unicode values although semantically they represent the same character. By default, TET converts all presentation forms to the corresponding canonical forms. As shown in Table 6.3 the *decompose* option can be used to preserve presentation forms (see Section 7.3.2, »Unicode Decomposition«, page 106).

Since the PDF document may map presentation forms either to the isolated Unicode character or one of the presentation forms (e.g. in the document's ToUnicode CMap), TET cannot guarantee that the output contains presentation forms even when decompositions are disabled.

Table 6.3 Processing Arabic presentation forms with the `decompose` *option*

description and option list	*before decomposition*	*after decomposition (in logical order)*
Decompose final, initial, isolated, and medial presentation forms: no `decompose` *option (default) or* `decompose={final=_all medial=_all initial=_all isolated=_all}` *Note that ligatures are only decomposed if they are actually represented by a ligature glyph. If multiple separate glyphs are used these are retained in the output.*	ﺱ U+FEB2	س U+0633
	ﺳ U+FEB3	س U+0633
	ﺭ U+FD0E	س U+0633 ر U+0631
	ﺴ U+FEB4	س U+0633
	ﻵ U+FEFC	ل U+0644 ا U+0627
	ل U+0644 ا U+0627	ل U+0644 ا U+0627

Preserve final, initial, isolated, and medial presentation forms:
`decompose={final=_none medial=_none initial=_none isolated=_none}`
or
`decompose=none`
	ﺱ U+FEB2	ﺱ U+FEB2

	ﺳ U+FEB3	ﺳ U+FEB3
	ﺭ U+FD0E	ﺭ U+FD0E
	ﺴ U+FEB4	ﺴ U+FEB4
	ﻵ U+FEFC	ﻵ U+FEFC

Remove Arabic Tatweel character. The Tatweel character U+0640 (also called *kashida)* is often used in Arabic text to stretch words so that they completely fill the line. Since the Tatweel doesn't carry any text information itself it is usually not required in the extracted text. By default, TET removes Tatweel characters from the extracted text. As shown in Table 6.4 the *fold* option can be used to preserve Tatweel characters (see Section 7.3.1, »Unicode Folding«, page 103).

Table 6.4 Processing the Tatweel character U+0640 with the `fold` *option*

description and option list	*before folding*	*after folding*
Remove Arabic Tatweel characters: no `fold` *option (default) or* `fold={{[U+0640] remove}} or fold={default}`	ـ U+0640	*n/a*
Preserve Arabic Tatweel characters (which are removed by default): `fold={{[U+0640] preserve}}`	ـ U+0640	ـ U+0640

6.6 Content Analysis

PDF documents provide the semantics (Unicode mapping) of individual text characters as well as their position on the page. However, they usually do not convey information about words, lines, columns or other high-level text units. The fragments comprising text on a page may contain individual characters, syllables, words, lines, or an arbitrary mixture thereof, without any explicit marks designating the start or end of a word, line, or column.

To make matters worse, the ordering of text fragments on the page may be different from the logical (reading) order. There are no rules for the order in which portions of text are placed on the page. For example, a page containing two columns of text could be produced by creating the first line in the left column, followed by the first line of the right column, the second line of the left column, the second line of the right column etc. However, logical order requires all text in the left column to be processed before the text in the right column is processed. Extracting text from such documents by simply replaying the instructions on the PDF page generally provides undesirable results since the logical structure of the text is lost.

TET's content analysis engine analyzes the contents, position, and relationship of text fragments in order to achieve the following goals:

- create words from characters, and insert separator characters between words if desired;
- remove redundant text, such as duplicates which are only present to create a shadow effect;
- recombine the parts of hyphenated words which span more than one line;
- identify text columns (zones);
- sort text fragments within a zone, as well as zones within a page.

These operations are discussed in more detail below, as well as options which provide some control over content processing.

Text granularity. The *granularity* option of *TET_open_page( )* specifies the amount of text which is returned by a single call to *TET_get_text( ):*

- With *granularity=glyph* each fragment contains the result of mapping one glyph, which may be more than one character (e.g. for ligatures). In this mode content analysis is disabled. TET returns the original text fragments on the page in their original order. Although this is the fastest mode, it is only useful if the TET client intends to do sophisticated postprocessing (or is only interested in the text position, but not in its logical structure) since the text may be scattered all over the page.
- With *granularity=word* the Wordfinder algorithm groups characters into logical words. Each fragment contains a word. Isolated punctuation characters (comma, colon, question mark, quotes, etc.) are returned as separate fragments by default, while multiple sequential punctuation characters are grouped as a single word (e.g. a series of period characters which simulates a dotted line). However, punctuation treatment can be changed (see »Word boundary detection for Western text«, page 87).
- With *granularity=line* the words identified by the Wordfinder are grouped into lines. If dehyphenation is enabled (which is the default) the parts of hyphenated words at the end of a line are combined, and the full dehyphenated word is part of the line.
- With *granularity=page* all words on the page are returned in a single fragment.

Separator characters are inserted between multiple words, lines, or paragraphs if the chosen granularity is larger than the respective unit. For example, with *granularity=word* there's no need to insert word separators since each call to *TET_get_text( )* returns exactly one word.

The separator characters can be specified with the *wordseparator*, *lineseparator*, *and paraseparator* options of *TET_open_document( )* (use U+0000 to disable a separator), for example:

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

All content processing operations are disabled for *granularity=glyph* and enabled for all other granularity settings. However, more fine-grain control is possible via separate options (see below).

Word boundary detection for Western text. The Wordfinder, which is enabled for all granularity modes except *glyph*, creates logical words from multiple glyphs which may be scattered all over the page in no particular order. Word boundaries for Western text are identified by two criteria:

- A sophisticated algorithm analyzes the geometric relationship among glyphs to find character groups which together form a word. The algorithm takes into account a variety of properties and special cases in order to accurately identify words even in complicated layouts and for arbitrary text ordering on the page.
  The suboption *usemetrics* of the *contentanalysis* page option can be used to disable this algorithm for special situations.
- Some characters, such as space and punctuation characters (e.g. colon, comma, full stop, parentheses) are considered a word boundary, regardless of their width and position. The suboption *useclasses* of the *contentanalysis* page option can be used to disable this algorithm for special situations.

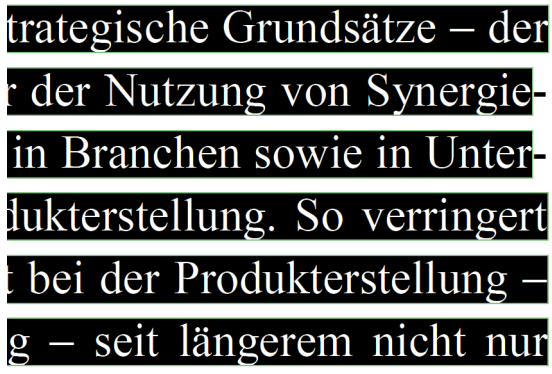
Ignoring punctuation characters for word boundary detection can, for example, be useful for maintaining Web URLs where period and slash characters are usually considered part of a word (see Figure 6.5). If the *punctuationbreaks* page option is set to *false* the Wordfinder no longer treats punctuation characters as word boundaries:

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

Note *Word boundary detection for text with ideographic characters works differently; see Section 6.4.2, »Word Boundaries for CJK Text«, page 82, for more information.*

Fig. 6.5
The default setting `punctuationbreaks=true` *separates the parts of URLs (top), while* `punctuationbreaks=false` *keeps the parts together (bottom).*

Dehyphenation. Hyphenated words at the end of a line are usually not desired for applications which process the extracted text on a logical level. TET therefore dehyphenates or recombines the parts of a hyphenated word. More precisely, if a word at the end of a line ends with a hyphen character and the first word on the next line starts with a lowercase character, the hyphen is removed and the first part of the word is combined with the part on the next line, provided there is at least one more line in the same zone. Dash characters (as opposed to hyphens) are left unmodified. The parts of a hyphenated word are not modified, only the hyphen is removed. Dehyphenation can be disabled with the following option list for *TET_open_page( ):*




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

Shadow and fake bold text removal. PDF documents sometimes include redundant text which does not contribute to the semantics of a page, but creates certain visual effects only. Shadow text effects are usually achieved by placing two or more copies of the actual text on top of each other, where a small displacement is applied. Applying opaque coloring to each layer of text provides a visual appearance where the majority of the text in lower layers is obscured, while the visible portions create a shadow effect.

Similarly, word processing applications sometimes support a feature for creating artificial bold text. In order to create bold text appearance even if a bold font is not available, the text is placed repeatedly on the page in the same color. Using a very small displacement the appearance of bold text is simulated.

Shadow simulation, artificial bold text, and similar visual effects create severe problems when reusing the extracted text since redundant text contents which contribute only to the visual appearance is processed although the text does not contribute to the page contents.

If the Wordfinder is enabled, TET identifies and removes such redundant visual effects by default. Shadow removal can be disabled with the following page option:

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

Accented characters. In many languages accents and other diacritical marks are placed close to other characters to form combined characters. Some typesetting programs, most notably TeX, emit two characters (base character and accent) separately to create a combined character. For example, to create the character *ä* first the letter *a* is placed on the page, and then the dieresis character ¨ is placed on top of it. TET detects this situation and recombines both characters to form the appropriate combined character.

6.7 Layout Analysis and Document Styles

TET analyses the layout of text on the page in order to determine the best possible order of text extraction. This automatic process can be assisted by several options. If you have advance knowledge of the nature of the documents you can improve the text extraction results by supplying suitable options.

Document styles. Several internal parameters are available for processing documents of different layout and style. For example, newspaper pages tend to contain lots of text in multiple columns, while business reports often contain comments in the margins, etc. TET contains predefined settings for several types of document. These settings can be activated with an option for *TET_open_page( )*:

```
docstyle=papers
```

If the type of input documents is known it is strongly recommended to supply suitable values of the *docstyle* page option and (if applicable) also the *layouthint* page option. Supplying the *docstyle* option activates an advanced layout recognition algorithm. However, supplying an unsuitable value for this option may actually create worse results.

The following types are available for the *docstyle* option (Table 6.5 contains typical examples for some document styles):

- *Book:* typical book layouts with regular pages
- *Business:* business documents
- *Cad:* technical or architectural drawings which are typically heavily fragmented
- *Fancy:* fancy pages with complex and sometimes irregular layout
- *Forms:* structured forms
- *Generic:* the most general document class without any further qualification
- *Magazines:* magazine articles, usually with three or more columns and interspersed images and graphics
- *Papers:* newspapers with many columns, large pages and small type
- *Science:* scientific articles, usually with two or more columns and interspersed images, formulae, tables, etc.
- *Search engine:* this class does not refer to a specific type of input document, but rather optimizes TET for the typical requirements of indexers for search engines. Some layout detection features are disabled to deliver only the raw text and speed up processing. For example, table and page structure recognition are disabled.
- *Simple line:* simplistic line-oriented layout; this mode disables dehyphenation and table detection and attempts to retrieve the text as close as possible to the original line layout. This may be useful if many similar documents are processed and the application has advance knowledge of the layout.
- *Space grid:* this class is targeted at list-oriented reports which are often generated on mainframe systems. The characteristic of this document class is that the visual layout is generated with space characters instead of explicit positioning of text. When processing this kind of document text extraction can be accelerated since some processing steps (e.g. shadow detection) can be skipped.

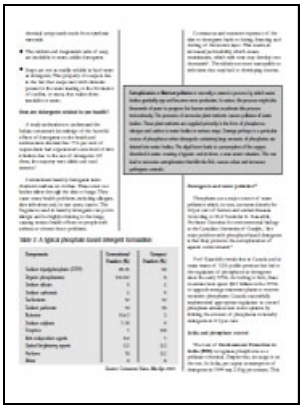
Choosing the most appropriate document style can speed up processing and enhance text extraction results.

Table 6.5 Document styles

docstyle=book

docstyle=business

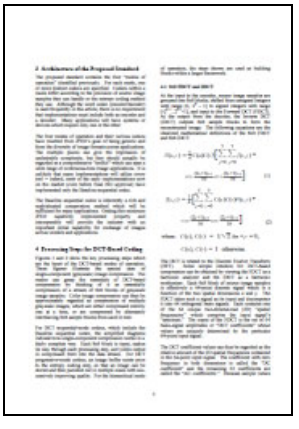
docstyle=fancy



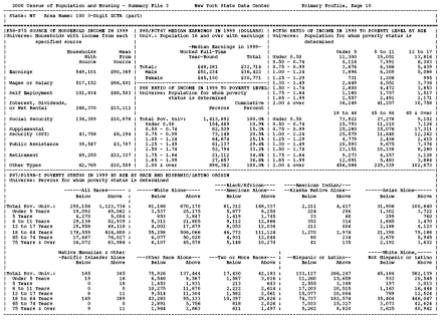
docstyle=magazines

docstyle=papers

docstyle=science



docstyle=spacegrid



docstyle=cad

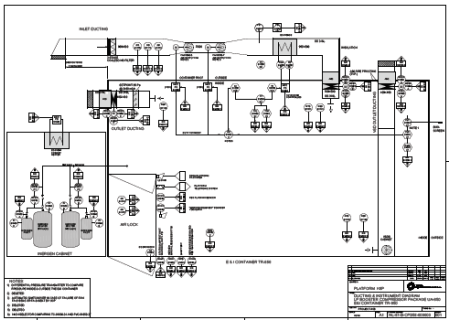


Table 6.5 Document styles

docstyle=simpleline

Complex layouts. Some classes of documents often use very elaborate page layouts. For example, with magazines and periodicals TET may not be able to properly determine the relationship of columns on the page. In such situations it is possible to enhance the extracted text at the expense of processing time. This can be controlled with the *structureanalysis* and *layoutanalysis* page options, e.g.

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

6.8 Table and List Detection

Table detection. TET detects tabular layouts and structures the table contents in rows, columns and cells. Information about tables detected on the page is not provided directly by the API, but is only available in TETML output as in the following example:

```
<Table llx="302.14" lly="639.72" urx="525.50" ury="731.50">
 <Row>
  <Cell colSpan="3" llx="306.14" lly="641.52" urx="516.67" ury="650.52">
   <Para>
    <Box llx="306.14" lly="641.52" urx="516.67" ury="650.52">
     <Word>
      <Text>TET</Text>
      <Box llx="306.14" lly="641.52" urx="319.70" ury="650.52"/>
     </Word>
     <Word>
      <Text>processes</Text>
      <Box llx="321.67" lly="641.52" urx="356.89" ury="650.52"/>
     </Word>
     <Word>
      <Text>all</Text>
      <Box llx="358.85" lly="641.52" urx="368.15" ury="650.52"/>
     </Word>
     ...
    </Box>
   </Para>
  </Cell>
 </Row>
</Table>
```

TET can optionally analyze the horizontal and vertical lines or colored boxes which are often used to enhance the table layout. This vector graphics analysis is disabled by default. It improves the results of table and layout detection if such graphical elements are present. Vector graphics analysis can be enabled with the page option *vectoranalysis*, for example

```
vectoranalysis={structures=tables}
```

If the table cells are completely framed with vector graphics you can instruct TET to identify cells solely based on the cell border lines (instead of analyzing the text positions). The following page option improves table detection results, but works only for tables where the cells are completely framed and which don't contain any row or column spans:

```
vectoranalysis={structures=usevectoronly}
```

By default, table detection ignores lines which are almost as long as the page height or width. If you know that large tables with long lines are present you can instruct TET to take these lines into account:

```
vectoranalysis={structures=usevectoronly pagesizelines=true}
```

The following page option enables row and column span detection in complex table layouts:

```
vectoranalysis={structures=vectoriterative}
```

List detection. TET detects list structures on the page, i.e. one or more list items where each items consists of a list label, e.g. a bullet, number or character, and a body section consisting of one or more paragraphs. Only list labels at the beginning of a line can be detected. Lists may be nested, i.e. a list item's body may itself contain another list. Information about lists detected on the page is not provided directly by the API, but is only available in TETML output as in the following example:

```
<List>
 <Item>
  <Label>
   <Word>
    <Text>•</Text>
    <Box llx="35.00" lly="737.00" urx="45.50" ury="767.00"/>
   </Word>
  </Label>
  <Body>
   <Para>
    <Box llx="35.00" lly="737.00" urx="169.15" ury="767.00">
     <Word>
      <Text>four</Text>
      <Box llx="70.00" lly="737.00" urx="85.00" ury="767.00"/>
     </Word>
     <Word>
      <Text>sorts</Text>
      <Box llx="92.50" lly="737.00" urx="169.15" ury="767.00"/>
     </Word>
    </Box>
   </Para>
    ...
  </Body>
 </Item>

</List>
```

For compatibility reasons list detection is disabled by default, and must be enabled with the following page option:

```
structureanalysis={list=true}
```

The following additional options can be used to control list detection:

- The suboption *bullets* of the page option *structureanalysis* can be used to define Unicode values and font names which are used for bullet characters. The default list contains a variety of Unicode characters such as asterisks, dashes, bullets which are commonly used as list labels.
- The *numericentities* suboption of the page option *contentanalysis* can be used to control the relationship of punctuation and digits which together comprise numeric entities such as composite numbers, fractions or time, e.g. the section number »10.4« in a heading is treated as follows:
  with the default *contentanalysis={numericentities=keep}:* as single word »10.4«
  with *contentanalysis={numericentities=split}:* as separate words »10«, ».« and »4«.

6.9 Check whether an Area is empty

TET can also be used to check whether a particular area on the page is empty, i.e. contains any text, image, or vector graphics objects which may be useful for postprocessing applications. For example, consider that you need to place a stamp, page number, barcode or other item somewhere on a page. If the page contents are variable it may be difficult to specify a location on the page where the stamp or barcode can be placed without obscuring some existing contents. TET can check whether the target area is actually empty. This feature works as follows with the TET API:

- The *emptycheck* page option activates the feature and disables any page content retrieval.
- The coordinates of the rectangular area which is checked are supplied in the *includebox* page option. Double braces are required since this option usually accepts multiple boxes (but only a single box makes sense for *emptycheck):*

  ```
  includebox={{100 20 500 100}}
  ```

  If the *includebox* option is not supplied the whole clipping area is checked. This can be used to identify empty pages.
- Instead of retrieving any page contents, *TET_get_text( )* returns one of the strings *empty* or *notempty* as result of the check.

The *emptycheck* feature can be used in the TET command-line tool as follows:

```
tet --pageopt "emptycheck includebox={{300 760 450 820}}" input.pdf
box on page 1: empty
box on page 2: empty
box on page 3: notempty
```

The *emptycheck* topic in the TET Cookbook demonstrates how to check whether a rectangle on the page is empty.

6.10 Annotation Contents

PDF documents may contain text and images in annotations (comments). There are two fundamentally different ways how text can be stored in annotations (see Table 6.6):

- The annotation may have an *appearance stream* which (like a page) can use the full PDF graphics model with fonts and colors. Text in an appearance stream is styled and has a position. Appearance streams may also contain images. The appearance stream is rendered on the page and therefore visible. TET processes annotation appearance streams by default, but this can be changed with the document option *engines={noannotation}*.
- The annotation may contain an entry called *Contents* which consists of a plain Unicode string without any styling or position. The PDF standard describes this entry as follows: »*It is the text that shall be displayed for the annotation or, if the annotation does not display text, an alternative description of the annotation's contents in human-readable form.*« Applications must decide whether they are interested in this text. The *Contents* entry is processed by default when creating TETML output, but must be retrieved explicitly with pCOS methods when using the API.

Table 6.6 Annotation text in appearance streams vs. `Contents` *entry*

	appearance stream of annotations	***Contents entry of annotations***
General		
may contain	*text and images*	*text*
text is styled and has geometry	*yes*	*no*
page option `includebox` *is honored*	*yes*	*no*
page option `granularity` *and document options* `wordseparator` *etc. are honored*	*yes*	*no*
Extraction with API methods		
relevant API methods	*TET_get_text()* *TET_get_image_info()* *TET_write_image_file()*	*TET_pcos_get_string()* *TET_pcos_get_number()*
extraction controlled by	*document option* `engines={annotation}`	*application code*
Where does the text appear?	*page contents*	*separately*
sample code	*extractor and* `images_per_page` *samples*	`text_from_annotations` *topic in the TET Cookbook*
other notes	*annotations are integrated in page processing, e.g. layout analysis and image merging*	*application should replace paragraph separator U+000D with U+000A*
Extraction with TETML		
extraction controlled by	*document option* `engines={annotation}`	*page option* `tetml={elements={annotations}}`
Where does the text appear?	`Content` *element of* `Page`	`Annotations` *element*
contributes fonts, images and color spaces to `Resources` *section*	*yes*	*no*

Form fields in PDF are also coded as annotations. They are therefore also subject to TET's annotation engine. The active contents of a form field can be retrieved from the /V (value) entry; see pCOS Cookbook topic *formfields*.

A related topic is text which is marked up with a highlight annotation. Proceed as follows to extract only highlighted text:

- Retrieve the rectangle coordinates of each Highlight annotation with pCOS.
- Use the *includebox* option of *TET_get_text( )* to restrict text extraction to a particular annotation, and process all highlight annotations this way.

This method is demonstrated in the topic *extract_highlighted_text* in the TET Cookbook.

7 Advanced Unicode Handling

7.1 Important Unicode Concepts

This section provides basic information about Unicode since text handling in TET heavily relies on the Unicode standard. The Unicode Web site provides a wealth of additional information:

```
www.unicode.org
```

Characters and glyphs. When dealing with text it is important to clearly distinguish the following concepts:

- *Characters* are the smallest units which convey information in a language. Common examples are the letters in the Latin alphabet, Chinese ideographs, and Japanese syllables. Characters have a meaning: they are semantic entities.
- *Glyphs* are graphical shapes which represent one or more particular characters. Glyphs have an appearance: they are representational entities.

There is no one-to-one relationship between characters and glyphs. For example, a ligature is a single glyph which represents two or more separate characters. On the other hand, a specific glyph may be used to represent different characters depending on the context (some characters look identical, see Figure 7.1).

Unicode postprocessing in TET can change the relationship of glyphs and resulting characters even more. For example, decompositions may convert a single character into multiple characters, and foldings may remove characters. For these reasons you must not assume any specific relationship of characters and glyphs.

BMP and PUA. The following terms occur frequently in Unicode-based environments:

- The *Basic Multilingual Plane (BMP)* comprises the code points in the Unicode range U+0000...U+FFFF. The Unicode standard contains many more code points in the supplementary planes, i.e. in the range U+10000...U+10FFFF.

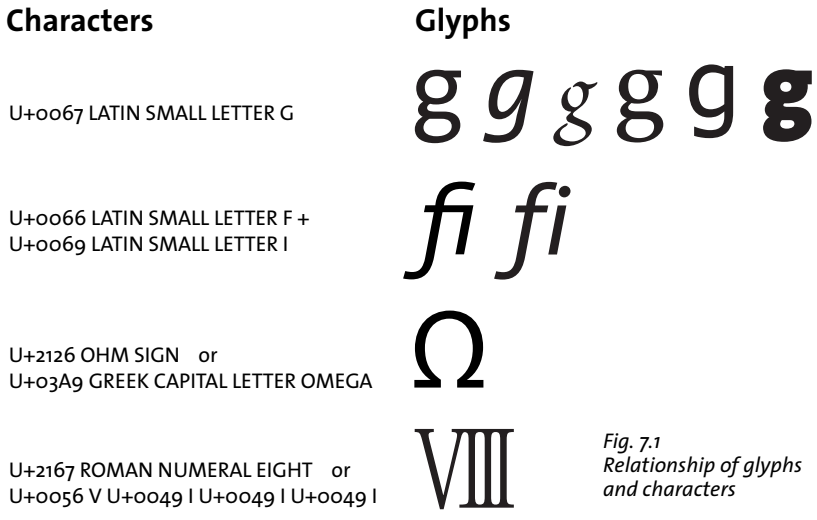


Fig. 7.1
Relationship of glyphs and characters

- A *Private Use Area (PUA)* is one of several ranges which are reserved for private use. PUA code points cannot be used for general interchange since the Unicode standard does not specify any characters in this range. The Basic Multilingual Plane includes a PUA in the range U+E000...U+F8FF. Plane fifteen (U+F0000... U+FFFFD) and plane sixteen (U+100000...U+10FFFD) are completely reserved for private use.

Unicode encoding forms (UTF formats). The Unicode standard assigns a number (code point) to each character. In order to use these numbers in computing, they must be represented in some way. In the Unicode standard this is called an encoding form (formerly: transformation format); this term should not be confused with font encodings. Unicode defines the following encoding forms:

- *UTF-8:* This is a variable-width format where code points are represented by 1-4 bytes. ASCII characters in the range U+0000...U+007F are represented by a single byte in the range 00...7F. Latin-1 characters in the range U+00A0...U+00FF are represented by two bytes, where the first byte is always 0xC2 or 0xC3 (these values represent *Â* and *Ã* in Latin-1).
- *UTF-16:* Code points in the Basic Multilingual Plane (BMP) are represented by a single 16-bit value. Code points in the supplementary planes, i.e. in the range U+10000... U+10FFFF, are represented by a pair of 16-bit values. Such pairs are called surrogate pairs. A surrogate pair consists of a high-surrogate value in the range D800...DBFF and a low-surrogate value in the range DC00...DFFF. High- and low-surrogate values can only appear as parts of surrogate pairs, but not in any other context.
- *UTF-32:* Each code point is represented by a single 32-bit value.

Unicode encoding schemes and the Byte Order Mark (BOM). Computer architectures differ in the ordering of bytes, i.e. whether the bytes constituting a larger value (16- or 32-bit) are stored with the most significant byte first (big-endian) or the least significant byte first (little-endian). A common example for big-endian architectures is PowerPC, while the x86 architecture is little-endian. Since UTF-8 and UTF-16 are based on values which are larger than a single byte, the byte-ordering issue comes into play here. An encoding scheme (note the difference to encoding form above) specifies the encoding form plus the byte ordering. For example, UTF-16BE stands for UTF-16 with big-endian byte ordering. If the byte ordering is not known in advance it can be specified by means of the code point U+FEFF, which is called Byte Order Mark (BOM). Although a BOM is not required in UTF-8, it may be present as well, and can be used to identify a stream of bytes as UTF-8. Table 7.1 lists the representation of the BOM for various encoding forms.

Table 7.1 Byte order marks for various Unicode encoding forms

Encoding form	*Byte order mark (hex)*	*graphical representation in WinAnsi*[1]
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. The square ■ denotes a null byte.

Composite characters and sequences. Some glyphs map to a sequence of multiple characters. For example, ligatures are mapped to multiple characters according to their constituent characters. However, composite characters (such as the Roman numeral in Figure 7.1) may or may not be split, subject to information in the font and PDF as well as the *decompose* document option (see Section 7.3, »Unicode Postprocessing«, page 103).

If appropriate, TET will split composite characters into a sequence of constituent characters. The corresponding sequence is part of the text returned by *TET_get_text()*. For each character, details of the underlying glyph(s) can be obtained via *TET_get_char_info()*, including the information whether the character is the start or continuation of a sequence. Position information will only be returned for the first character of a sequence. Subsequent characters of a sequence will not have any associated position or width information, but must be processed in combination with the first character.

Characters without any corresponding glyph. Although every glyph on the page is mapped to one or more corresponding Unicode characters, not all characters delivered by TET actually correspond to a glyph. Characters which correspond to a glyph are called real characters, others are called artificial characters. There are several classes of artificial characters which are delivered although a directly corresponding glyph is not available:

- A composite character (see above) will map to a sequence of multiple Unicode characters. While the first character in the sequence corresponds to the actual glyph, the remaining characters do not correspond to any glyph.
- Separator characters inserted via the *lineseparator*, *wordseparator*, and *paraseparator* options don't correspond to any visible glyph.

7.2 Text Preprocessing (Filtering)

TET applies several filters to remove text which is unlikely to be useful. These filters modify the text before applying any Unicode postprocessing steps. While some filters are always active, others require the Wordfinder and are therefore active only for *granularity=word* or above.

7.2.1 Filters for all Granularities

The following filters can be used with all granularities.

Text in unwieldy font sizes. Very small or very large text can optionally be ignored, e.g. large characters in the background of the page. The limits can be controlled with the *fontsizerange* page option. By default text in all font sizes is extracted.

The following page option limits the range of font sizes for extracted text from 10 to 50 points; text in other font sizes is ignored:

```
fontsizerange={10 50}
```

Invisible text. Invisible text (i.e. text with *textrendering=3)* is extracted by default. Note that text in PDF may be invisible for various other reasons than the textrendering property, e.g. the text color is identical to the background color, the text may be obscured by other objects on the page, etc. The behavior described here relates only to text with *textrendering=3*. This PDF technique is commonly used for the results of OCR where the text sits invisibly »on top of« the scanned raster image.

Invisible text can be identified with the *textrendering* member of the *TET_char_info* structure returned by *TET_get_char_info()* (see Table 10.16, page 200), or with the *Glyph/@textrendering* attribute in TETML.

Use the following page option if you want to ignore invisible text:

```
ignoreinvisibletext=true
```

Completely ignore text with certain font names or font types. In some situations it may be useful to completely ignore text in one ore more fonts specified by name, e.g. a symbolic font which does not contribute any meaningful text. As an alternative, the problematic fonts can also be specified by font type. This is mainly useful for Type 3 fonts which are sometimes used for ornaments. This filter can be controlled via the *remove* suboption of the *glyphmapping* document option.

E.g. ignore all text in Type 3 fonts:

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

Ignore all text in the Webdings, Wingdings, Wingdings 2, and Wingdings 3 fonts:

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

The conditions for font name and font type can also be combined, e.g. ignore text in all Type 3 fonts starting with the letter A:

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

Artifacts. Artifacts in Tagged PDF designate irrelevant text or images. Typical examples are running headers and footers, e.g. page numbers. In Tagged PDF documents such Artifacts can be marked. By default TET extracts Artifacts like regular content. However, Artifacts can be skipped with the page option *ignoreartifacts*. This option affects both the API and TETML.

Alternatively, Artifacts can be identified with the *attributes* member of the *TET_char_info* structure (see Table 10.16, page 200) or the *Glyph/@artifact* and *Text/@artifact* attributes in TETML:

```
<Para>
 <Box llx="76.58" lly="765.12" urx="82.58" ury="777.12">
  <Word>
   <Text artifact="true">6</Text>
   <Box llx="76.58" lly="765.12" urx="82.58" ury="777.12"/>
  </Word>
 </Box>
</Para>
```

7.2.2 Filters for Granularity Word and above

The following filters can be used only for *granularity=word, line*, and *page*.

Dehyphenation. Dehyphenation removes hyphen characters and combines the parts of a hyphenated word.

Hyphens used for splitting words across lines can be identified with the *attributes* member of the *TET_char_info* structure (see Table 10.16, page 200), or with the *Glyph/@hyphenation* attribute in TETML.

Dehyphenation can be disabled with the following page option:

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

Hyphen reporting. If dehyphenation is enabled you can decide whether or not the hyphen characters between the parts of hyphenated words are reported in the generated glyph lists or not, i.e. the list of glyphs returned by *TET_get_char_info( )* and the *Glyph* elements in TETML. By default, hyphens are removed.

However, some applications may need to know the exact location of the hyphen on the page. For example, the *highlight_search_terms* and *search_and_replace_text* topics in the TET Cookbook take the hyphen glyph into account when placing an annotation or replacement text on top of the original word. In this situation you can instruct TET to include all hyphens which have been detected by the dehyphenation process with the following page option:

```
contentanalysis={keephyphenglyphs=true}
```

Hyphens can be identified with the *TET_ATTR_DEHYPHENATION_ARTIFACT* flag of the attributes member in the *TET_char_info* structure returned by *TET_get_char_info( )* (see Table 10.16, page 200), or in TETML with the *Glyph/@dehyphenation* attribute with value *artifact (unrelated to Artifacts in Tagged PDF)*.

Shadow removal. Redundant text which creates only visual effects such as shadowed or artificially bolded text is removed.

Shadow and artificial bold text can be identified with the *attributes* member of the *TET_char_info* structure (see Table 10.16, page 200), or with the *Glyph/@shadow* attribute in TETML.

Shadow removal can be disabled with the following page option:

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

7.3 Unicode Postprocessing

TET offers various controls for fine-tuning the Unicode characters comprising the extracted text. The postprocessing steps discussed in this chapter are defined in the Unicode standard. They are available in TET and are processed in the following order:

- Foldings are controlled by the *fold* document option and preserve, remove, or replace certain characters. Examples: remove hyphens which are used to split words, remove Arabic Tatweel characters.
- Decomposition is controlled by the *decompose* document option and replaces a character with one ore more equivalent characters. Examples: split ligatures, map fullwidth ASCII and symbol variants to the corresponding non-fullwidth characters.
- Normalization is controlled by the *normalize* document option and converts the text to one of the normalized Unicode forms. Examples: combine base character and diacritical character to a common character; map Ohm sign to Greek Omega.

Note *Unicode postprocessing does not apply to* `granularity=glyph` *and the* `Glyph` *element in TETML. Foldings, decomposition and normalization are not performed in these cases.*

7.3.1 Unicode Folding

Foldings process one or more Unicode characters and apply a certain action on each of the characters. The following actions are available:

- preserve the character;
- remove the character;
- replace it with a another (fixed) character.

Foldings are not chained: the output of a folding will not be processed again by the available foldings. Foldings affect only the Unicode text output, but not the set of glyphs reported in the *TET_char_info* structure or the <*Glyph*> elements in TETML. For example, if a folding removes certain Unicode characters, the corresponding glyphs which created the initial characters will still be reported.

In order to improve readability the examples in the tables below list isolated suboptions of the *fold* document option list. Keep in mind that these suboptions must be combined to a single large *fold* option list if you want to apply multiple foldings; do not supply the *fold* option more than once. For example, the following is wrong:

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

The following document option list shows the correct syntax for multiple foldings:

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

Folding examples. Table 7.2 lists examples for the *fold* option which demonstrate various folding applications. The options must be supplied in the option list for *TET_open_document()*. TET can apply foldings to a selected subset of all Unicode characters. These are called Unicode sets; their syntax is discussed in »Unicode sets«, page 163.

Table 7.2 Examples for the `fold` *option*

description and option list	*before folding*	*after folding*
Remove all characters in a Unicode set		
Keep only characters in ISO 8859-1 (Latin-1) in the output, i.e. remove all characters outside the Basic Latin Block: `fold={{[^U+0020-U+00FF] remove} default}`	Ą U+0104	*n/a*
Remove all non-alphabetic characters (e.g. punctuation, numbers): `fold={{[:Alphabetic=No:] remove} default}`	7 U+0037	*n/a*
Remove all characters except numbers: `fold={{[^[:General_Category=Decimal_Number:]] remove} default}`	7 U+0037 A U+0041	7 U+0037 *n/a*
Remove all dashes and punctuation characters: `fold={{[:General_Category=Dash_Punctuation:] remove} default}`	- U+002D	*n/a*
Remove all Bidi control characters: `fold={{[:Bidi_Control:] remove} default}`	U+200E	*n/a*
Remove all variation selectors for Standard or Ideographic Variation Sequences (IVS): `fold={{[[\uFE00-\uFE0F][\U000E0100-\U000E01EF]] remove} default}`	≦ U+2268 FE 00 U+FE00	≦ U+2268
Replace all characters in a Unicode set with another character		
Space folding: map all variants of Unicode space characters to U+0020: `fold={{[:blank:] U+0020} default}`	U+00A0	U+0020
Dashes folding: map all variants of Unicode dash characters to U+002D: `fold={{[:Dash:] U+002D} default}`	‐ U+2011	- U+002D
Replace all unassigned characters (i.e. Unicode code points to which no character is assigned) with U+FFFD: `fold={{[:Unassigned:] U+FFFD} default}`	U+03A2	? U+FFFD
Special handling for individual characters		
Preserve all hyphen characters at line breaks while keeping the remaining default foldings. Since these characters are identified internally in TET (as opposed to having a fixed Unicode property) the keyword `_dehyphenation` *is used to identify the folding's domain:* `fold={{_dehyphenation preserve} default}`	‐ U+002D	- U+002D
Preserve Arabic Tatweel characters (which are removed by default): `fold={{[U+0640] preserve} default}`	ـ U+0640	ـ U+0640
Replace various punctuation characters with their ASCII counterparts: `fold={ {[U+2018] U+0027} {[U+2019] U+0027} {[U+201C] U+0022} {[U+201D] U+0022} default}`	“ U+201C	" U+0022
Handle font-specific PUA characters, e.g. Japanese EUDC or logo font		
Default behavior: replace PUA characters with the Unicode replacement character U+FFFD: `fold={{[:Private_Use:] U+FFFD} default}`		? U+FFFD
Preserve PUA characters: `fold={{[:Private_Use:] preserve} default}`		
Remove PUA characters: `fold={{[:Private_Use:] remove} default}`		*n/a*
Remove TET PUA values for unmappable glyphs, but preserve PUA characters from fonts: `fold={{_tet_pua remove} {[:Private_Use:] preserve} default}`		

Default foldings. Except for *granularity=glyph* TET applies the following default foldings which are explained in Table 7.3:

```
{[:blank:] U+0020}
{_tetpua unknownchar}
{[:Private_Use:] U+FFFD}
{_dehyphenation remove}
{[[\u0640][:Control:][:Unassigned:]] remove}
```

In order to combine custom foldings with the default foldings, the keyword *default* must be supplied after the custom folding options (this is shown in all examples in Table 7.2).For example, the following *fold* document option list preserves hyphens in dehyphenated words and then applies the default foldings:

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

Adding the keyword *default* to the *fold* document option list is recommended in most cases unless you want to explicitly disable all default foldings.

Table 7.3 Default foldings

folding and description	***sample input***	***output***
Space folding: map all variants of Unicode space characters to U+0020: `{[:blank:] U+0020}`	U+00A0	U+0020
Map TET PUA values for unmappable glyphs to the character specified in the `unknownchar` *option (or apply the specified action* `preserve`/`remove`*):* `{_tetpua unknownchar}`		U+FFFD
Map PUA characters to the Unicode replacement character U+FFFD: `{[:Private_Use:] U+FFFD}`		U+FFFD
Remove hyphens in dehyphenated words: `{_dehyphenation remove}`	- U+002D	*n/a*
Remove Arabic Tatweel characters, control characters and characters which are not assigned in Unicode (these foldings are always performed after all other foldings when creating TETML output): `{[\u0640][:Control:][:Unassigned:] remove}`	- U+0640	*n/a*
	U+000C U+03A2	

7.3.2 Unicode Decomposition

Decompositions replace a character with an equivalent sequence of one or more other characters. A Unicode character is called (either compatibility or canonical) equivalent to another character or a sequence of characters if they actually mean the same, but for historical reasons (mostly related to round tripping with legacy encodings) are encoded separately in Unicode. Decompositions destroy information. This is useful if you are not interested in the difference between the original character and its equivalent. If you *are* interested in the difference, however, the respective decomposition should not be applied.

Note The term »decomposition« is used here as defined in the Unicode standard, although many decompositions do not actually split a character into multiple parts, but convert a single character to another character.

Canonical decomposition. Characters or character sequences which are canonically equivalent represent the same abstract character and should therefore always have the same appearance and behavior. Common examples include precomposed characters (e.g. Ä U+00C4) vs. combining sequences (e.g. A ¨ U+0041 U+0308): both representations are canonically equivalent. Switching from one representation to the other does not remove information. Canonical decompositions replace one representation with another which is considered the canonical representation.

In the Unicode code charts (but not the character tables) canonical mappings are marked with the symbol IDENTICAL TO ≡ U+2261. The decomposition name <*canonical*> is implicitly assumed. Table 7.4 contains several examples.

Compatibility decomposition. Characters which are compatibility equivalent represent the same abstract character, but may differ in appearance or behavior. Examples include isolated forms of Arabic characters (e.g. س U+0633) vs. context-specific shaped forms (e.g. ﺱ U+FEB1, ﺳ U+FEB4, ﺲ U+FEB2). Compatibility equivalent characters differ in formatting. Removing this formatting information implies loss of information, but may simplify processing for certain types of applications (e.g. searching).

In the Unicode code charts compatibility mappings are marked with the symbol ALMOST EQUAL TO ≈ U+2248, followed by the decomposition name (or »tag«) in angle brackets, e.g. <*noBreak*>. If no tag name is provided, <*compat*> is assumed. The tag names are identical to the option names in Table 7.5. As can be seen in some of the examples, the result of a decomposition may convert a single character to a sequence of multiple characters.

Note PDF documents may map glyphs to the decomposed sequence instead of to the non-decomposed Unicode value. In this situation the `decompose` *option doesn't affect the output.*

Decomposition examples. Decompositions in TET can be controlled with the document option *decompose*. A decomposition can be restricted to operate only on some, but not all Unicode characters. The subset on which a decomposition operates is called its

Table 7.4 Canonical decomposition: suboption for the decompose *option (canonically equivalent characters are marked with the symbol IDENTICAL TO* ≡ U+2261 *in the Unicode code charts)*

decomposition name	***description***	***before decomposition***	***after decomposition***
canonical[1]	*Canonical decomposition*	À U+00C0	A ◌̀ U+0041 U+0300
		林 U+F9F4	林 U+6797
		Ω U+2126	Ω U+03A9
		ば U+3070	は ◌゙ U+306F U+3099
		אַ U+FB2F	א ◌ַ U+05D0 U+05B8

1. By default this decomposition is not applied to all characters in order to preserve certain characters; see »Default decompositions«, page 109, for details.

domain. Table 7.5 lists the suboptions for all Unicode decompositions along with examples.

The following examples for the *decompose* option must be supplied in the option list for *TET_open_document( )*. The decomposition names in the *decompose* option list are taken from Table 7.5.

Disable all decompositions:

```
decompose={none}
```

Preserve wide (double-byte or *zenkaku)* and narrow *(hankaku)* characters:

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

Map all canonical equivalents to their counterparts:

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

Table 7.5 Compatibility decomposition: suboptions for the `decompose` *option (canonically equivalent characters are marked with the symbol ALMOST EQUAL TO* ≈ U+2248 *in the Unicode code charts)*

decomposition name	*description*	*before decomposition*	*after decomposition (in logical order)*
circle	*Encircled characters*	㉑ U+3251	2 1 U+0032 U+0031
compat	*Other compatibility decompositions, e.g. common ligatures*	ﬁ U+FB01	f i U+0066 U+0069
final	*Final presentation forms, especially Arabic*	ﺱ U+FEB2	س U+0633
font	*Font variants, e.g. mathematical set letters, Hebrew ligatures*	ℂ U+2102	C U+0043
fraction	*Vulgar fraction forms*	¼ U+00BC	1 ⁄ 4 U+0031 U+2044 U+0034
initial	*Initial presentation forms, especially Arabic*	ﺳ U+FEB3	س U+0633
isolated	*Isolated presentation forms, especially Arabic*	ﺴﺮ U+FD0E	س ر U+0633 U+0631
medial	*Medial presentation forms, especially Arabic*	ﺴ U+FEB4	س U+0633
narrow	*Narrow (hankaku) compatibility characters*	ｦ U+FF66	ヲ U+30F2
nobreak	*Non-breaking characters*	U+00A0	U+0020
small	*Small forms for CNS 11643 compatibility*	﹐ U+FE50	, U+002C
square	*CJK squared font variants*	㌔ U+3314	キ 口 U+30AD U+30ED
sub	*Subscript forms*	₁ U+2081	1 U+0031
super	*Superscript forms*	ª U+00AA	a U+0061
		™ U+2122	T M U+0054 U+004D
vertical	*Vertical layout presentation forms*	﹇ U+FE37	{ U+007B
wide	*Wide (zenkaku) compatibility forms*	￡ U+FFE1	£ U+00A3

Default decompositions. By default, all decompositions except *fraction* are enabled. While most default decompositions operate on the *_all* domain (i.e. they are applied to all characters), some operate on smaller default domains according to Table 7.6. A straightforward way of dealing with decompositions is via normalization (see Section 7.3.3, »Unicode Normalization«, page 110). Since Unicode postprocessing is disabled for *granularity=glyph* no decompositions are active in this case.

Table 7.6 Default domains for Unicode decompositions (suboptions for the `decompose` *option)*

decomposition	*default in TET*
canonical	`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]}` *The default domain includes canonical duplicates (singletons), but not other canonically equivalent characters. The default is not* `_all` *in order to preserve characters like* Ä U+00C4 .
compat	`compat={[U+FB00-U+FB17]}` *The default domain includes Latin and Armenian ligatures, but not other compatibility characters. The default is not* `_all` *in order to preserve characters like* IJ U+0132 .
fraction	`fraction=_none` *Fractions are not decomposed by default because this would lead to undesired sequences of the digits for integer and fractional parts, e.g. client applications would wrongly interpret the sequence* 9 U+0039 ½ U+00BD *(representing the numerical value 9.5) as* 9 U+0039 1 U+0031 ⁄ U+2044 2 U+0032 *which represents the numerical value (91)/2=45.5.*

sub
super
	`sub={[U+208A-U+208E]}`
	`super={[U+207A-U+207E]}`
	The default domain includes only mathematical signs. Superscript and subscript digits are not decomposed by default to avoid problems with the numerical interpretation similar to those mentioned above for `fraction`**.** *Characters such as the trademark sign* TM U+2122 *will not be decomposed to* T U+0054 M U+004D *by default.*

all others	*All decompositions not mentioned above are enabled for all characters by default:* `circle=_all final=_all ... vertical=_all wide=_all`

7.3.3 Unicode Normalization

The Unicode standard defines four normalization forms which are based on the notions of canonical equivalence and compatibility equivalence.[1] All normalization forms put combining marks in a specific order and apply decomposition and composition in different ways:

- Normalization Form C (NFC) applies canonical decomposition followed by canonical composition. For example, the composed form C stores *Ä* as a single character Ä (U+00C4). NFC is the preferred format for Unicode text in Windows, on the Web and in most databases.
- Normalization Form D (NFD) applies canonical decomposition. For example, the decomposed form D stores *Ä* as a sequence A ¨ (U+0041 U+0308) of base character and combining diacritical character.
- Normalization Form KC (NFKC) applies compatibility decomposition followed by canonical composition. In other words, some characters are mapped to compatible basic forms, e.g. the ligature fi (U+FB01) is mapped to the sequence f i (U+0066 U+0069).
- Normalization Form KD (NFKD) applies compatibility decomposition. This is similar to form KC, but does not apply canonical composition.

The choice of normalization form depends on the application's requirements. Table 7.7 demonstrates the effect of Normalization on various characters.

TET supports all four Unicode normalization forms. Unicode normalization can be controlled via the *normalize* document option, e.g.

```
normalize=nfc
```

TET does not apply normalization by default. Because of the possible interaction between the *decompose* and *normalize* options, setting the *normalize* option to a value different from *none* disables the default decompositions.

Table 7.7 Examples for Unicode normalization forms

before normalization	***NFC***	***NFD***	***NFKC***	***NFKD***
Ä U+00C4	Ä U+00C4	A ¨ U+0041 U+0308	Ä U+00C4	A ¨ U+0041 U+0308
A ¨ U+0041 U+0308	Ä U+00C4	A ¨ U+0041 U+0308	Ä U+00C4	A ¨ U+0041 U+0308
¨ A U+0308 U+0041	¨ A U+0308 U+0041	¨ A U+0308 U+0041	¨ A U+0308 U+0041	¨ A 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

1. The normalization forms are specified in Unicode Standard Annex #15 »Unicode Normalization Forms« (see www.unicode.org/versions/Unicode8.0.0/ch03.pdf#G21796 and www.unicode.org/reports/tr15/).

Table 7.7 Examples for Unicode normalization forms

before normalization	***NFC***	***NFD***	***NFKC***	***NFKD***
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	A ◌̊ U+0041 U+030A	Å U+00C5	A ◌̊ U+0041 U+030A
™ U+2122	™ U+2122	™ 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
רּ U+FB48	רּ U+05E8 U+05BC	רּ U+05E8 U+05BC	רּ U+05E8 U+05BC	רּ 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	ち ◌゙ U+3061 U+3099	ぢ U+3062	ち ◌゙ U+3061 U+3099
㉙ U+32C9	㉙ U+32C9	㉙ U+32C9	1 0 月 U+0031 U+0030 U+6708	1 0 月 U+0031 U+0030 U+6708

7.4 Supplementary Characters and Surrogates

Supplementary characters outside Unicode's Basic Multilingual Plane (BMP), i.e. those with Unicode values above *U+FFFF*, cannot be expressed as a single UTF-16 value, but require a pair of UTF-16 values called a surrogate pair. Examples of supplementary characters include various mathematical and musical symbols at *U+1DXXX* as well as thousands of CJK extension characters starting at *U+20000*. TET also uses the Supplementary Private Use Area to assign Unicode values to glyphs for which no Unicode mapping was found in the PDF document. By default, these characters are replaced with the Unicode replacement character U+FFFD. However, with the option *unknownchar=preserve* they can occur in the output as Unicode values outside the BMP, i.e. values above U+FFFF (see »Unmappable glyphs and the TET PUA«, page 113).

TET interprets and maintains supplementary characters and provides access to the corresponding UTF-32 value even in language bindings where native Unicode strings support only UTF-16. The *uv* field returned by *TET_get_char_info( )* for the leading surrogate value contains the corresponding UTF-32 value. This allows direct access to the UTF-32 value of a supplementary character even if you are working in a UTF-16 environment without any support for UTF-32.

Leading (high) surrogates and trailing (low) surrogates are maintained. The string returned by *TET_get_text( )* contains two UTF-16 values.

7.5 Unicode Mapping for Glyphs

While text in PDF can be represented with a variety of font and encoding schemes, TET abstracts from glyphs and normalizes all text to Unicode characters, regardless of the original text representation in the PDF. Converting the information found in the PDF to the corresponding Unicode values is called *Unicode mapping*, and is crucial for understanding the semantics of the text (as opposed to rendering a visual representation of the text on screen or paper). In order to provide proper Unicode mapping TET consults various data structures which are found in the PDF document, embedded or external font files, as well as builtin and user-supplied tables. In addition, it applies several methods to determine the Unicode mapping for non-standard glyph names.

Despite all efforts there are still a few PDF documents where some text cannot be mapped to Unicode. In order to deal with these cases TET offers a number of configuration features which can be used to control Unicode mapping for problematic PDF files.

Unmappable glyphs and the TET PUA. There are several reasons why text in a PDF cannot be mapped to Unicode. For example, Type 1 fonts may contain unknown glyph names, and TrueType, OpenType, or CID fonts may be addressed with glyph ids without any Unicode values in the font or PDF. If TET cannot determine a Unicode value after examining the information in the PDF document, embedded and external fonts, configured tables and internal tables the glyph is considered as unmappable.

Unmapped glyphs can be identified with the *unknown* member of the *TET_char_info* structure (see Table 10.16, page 200) or the *Glyph/@unknown* attribute in TETML.

TET assigns decreasing values in the TET Private Use Area (TET PUA) to all unmappable glyphs. The TET PUA is located in the *Supplementary Private Use Area*, i.e. outside the BMP, to avoid conflicts with PUA values assigned in fonts. The TET PUA can be addressed with the keyword *_tetpua* as source in the *fold* option.

By default, TET PUA values for unmappable glyphs are replaced with the Unicode replacement character U+FFFD. This behavior can be modified with the *unknownchar* document option which can be set to an arbitrary Unicode character, or to specify that TET PUA values for unmappable glyphs are preserved or removed. Table 7.2 explains various combinations of the *fold* and *unknownchar* options for different use cases.

Table 7.8 Specifying treatment of TET PUA values for unmappable glyphs with the `unknownchar` *document option*

description and option list	*raw input*	*result*
Default behavior: replace TET PUA values for unmappable glyphs with the Unicode replacement character U+FFFD: `unknownchar=U+FFFD`		U+FFFD
Replace TET PUA values for unmappable glyphs with a question mark (or any other suitable Unicode character); this may be useful for visually identifying problematic glyphs in the text: `unknownchar=?`		? U+003F
Remove TET PUA values for unmappable glyphs: `unknownchar=remove`		*n/a*
Preserve TET PUA values for unmappable glyphs; this may be useful for debugging and analysis: `unknownchar=preserve`		*(TET PUA value)*

Characters in the Private Use Area (PUA). A font or PDF document may map a glyph to a Unicode character in the Private Use Area. This is commonly used for symbols without any global standardized meaning, such as fonts for Japanese end-user defined characters (EUDC) or logo fonts. Since PUA characters cannot meaningfully be used in generic Unicode workflows they are replaced with the Unicode replacement character U+FFFD by default. See Table 7.2 for preserving PUA values in situations where the application can handle PUA values.

Summary of Unicode mapping controls. While TET implements many workarounds in order to process PDF documents which actually don't contain Unicode values so that it can successfully extract the text nevertheless. However, there are still documents where the text cannot be extracted since not enough information is available in the PDF and relevant font data structures. TET contains various configuration features which can be used to supply additional Unicode mapping information. These features are detailed in this section.

Using the *glyphmapping* option of *TET_open_document( )* (see Section 10.3, »Document Methods«, page 179) you can control Unicode mapping for glyphs in several ways. The following list gives an overview of available methods (which can be combined). These controls can be applied on a per-font basis or globally for all fonts in a document:

- The suboption *forceencoding* can be used to completely override all occurrences of the predefined PDF encodings *WinAnsiEncoding* or *MacRomanEncoding*.
- The suboptions *codelist* and *tounicodecmap* can be used to supply Unicode values in a simple text format (a *codelist* resource).
- The suboption *glyphlist* can be used to supply Unicode values for non-standard glyph names.
- The suboption *glyphrule* can be used to define a rule which is used to derive Unicode values from numerical glyph names in an algorithmic way. Several rules are already built into TET. The option *encodinghint* can be used to control the internal rules.
- In addition to dozens of predefined encodings, custom encodings can be defined for use with the *encodinghint* option or the *encoding* suboption of the *glyphrule* option.
- External fonts can be configured to provide Unicode mapping information if the PDF does not provide enough information and the font is not embedded in the PDF.

Analyzing PDF documents with the PDFlib FontReporter Plugin[1]. In order to obtain the information required to create appropriate Unicode mapping tables you must analyze the problematic PDF documents.

PDFlib GmbH provides a free companion product to TET which assists in this situation: PDFlib FontReporter is an Adobe Acrobat plugin for easily collecting font, encoding, and glyph information. The plugin creates detailed font reports containing the actual glyphs along with the following information:

- The corresponding code: the first hex digit is given in the left-most column, the second hex digit is given in the top row. For CID fonts the offset printed in the header must be added to obtain the code corresponding to the glyph.
- The glyph name if present.
- The Unicode value(s) corresponding to the glyph (if Acrobat can determine them).

These pieces of information play an important role for TET's glyph mapping controls. Figure 7.2 shows two pages from a sample font report. Font reports created with the

1. The PDFlib FontReporter plugin is available for free download at www.pdflib.com/products/fontreporter

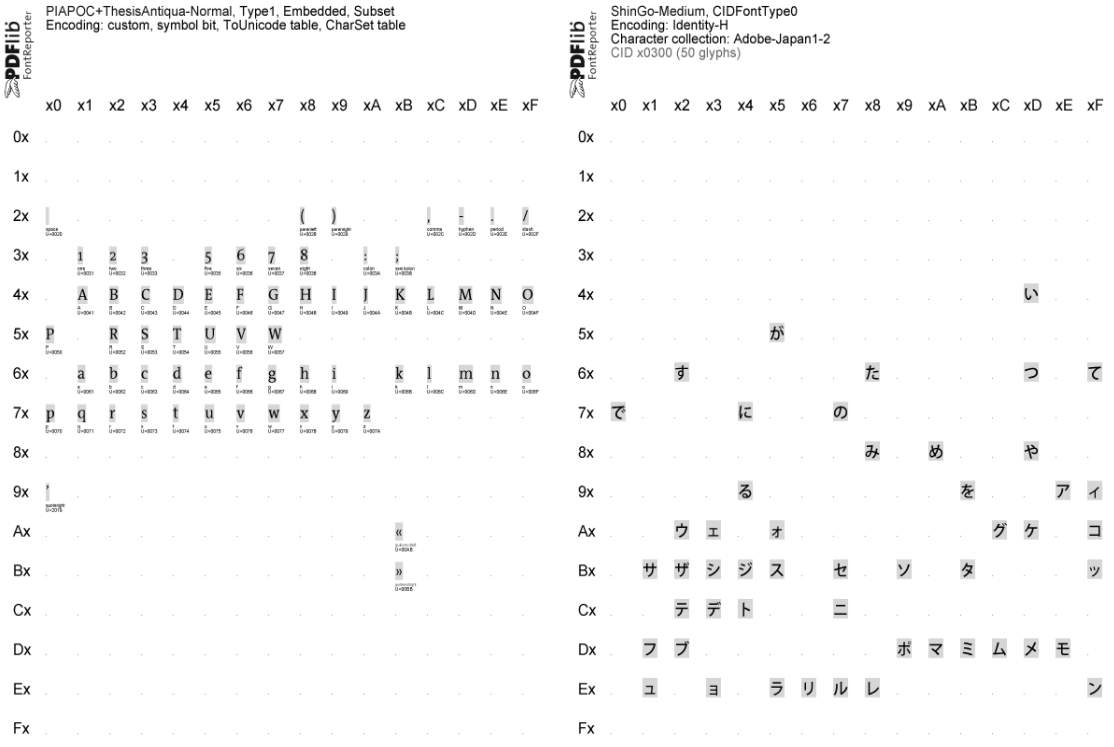


Fig. 7.2
Sample font reports created with the PDFlib FontReporter plugin for Adobe Acrobat

FontReporter plugin can be used to analyze PDF fonts and create mapping tables for successfully extracting the text with TET. It is highly recommended to take a look at the corresponding font report if you want to write Unicode mapping tables or glyph name heuristics to control text extraction with TET.

Precedence rules. TET will apply the glyph mapping controls in the following order:

- Codelist resources are consulted first.
- If the font has an internal ToUnicode CMap it is considered next.
- For glyph names TET applies an external or internal glyph name mapping rule if one is available which matches the font and glyph name.
- Lastly, a user-supplied glyph list is applied.

Code list resources for all font types. Code lists are similar to glyph lists except that they specify Unicode values for individual codes instead of glyph names. Although multiple fonts from the same foundry may use identical code assignments, codes (also called glyph ids) are generally font-specific. As a consequence, separate code lists are required for individual fonts. A code list is a text file where each line describes a Unicode mapping for a single code according to the following rules:

- Text after a percent sign '%' is ignored; this can be used for comments.
- The first column contains the glyph code in hexadecimal notation. This must be a value in the range 0-255 for simple fonts, and in the range 0-65535 for CID fonts.

Fig. 7.3
The font report for a logotype font shows that the font contains wrong Unicode mappings. A custom code list can correct such mappings.

- The remainder of the line contains up to 7 Unicode code points for the code. The values can be supplied in hexadecimal notation with the prefix *x* or *ox*. UTF-32 is supported, i.e. surrogate pairs can be used.

By convention, code lists use the file name suffix *.cl*. Code lists can be configured with the *codelist* resource category. If no code list resource has been specified explicitly, TET will search for a file named <*mycodelist*>*.cl* (where <*mycodelist*> is the resource name) in the *searchpath* hierarchy (see Section 5.2, »Resource Configuration and File Searching«, page 61 for details). In other words: if the resource name and the file name (without the *.cl* suffix) are identical you don't have to configure the resource since TET will implicitly do the equivalent of the following call (where *name* is an arbitrary resource name):

```
set_option("codelist {name name.cl}");
```

The following sample demonstrates the use of code lists. Consider the mismapped logotype glyphs in Figure 7.3 where a single glyph of the font actually represents multiple characters, and all characters together create the company logotype. However, the glyphs are wrongly mapped to the characters *a, b, c, d,* and *e*. In order to fix this you could create the following code list:

```
% Unicode mappings for codes in the GlobeLogosOne font

x61     x0054 x0068 x0065 x0020         % The
x62     x0042 x006F                     % Bo
x63     x0073 x0074 x006F x006E x0020   % ston
x64     x0047 x006C x006F               % Glo
x65     x0062 x0065                     % be
```

Then supply the codelist with the following option to *TET_open_document()* (assuming the code list is available in a file called *GlobeLogosOne.cl* and can be found via the search path):

```
glyphmapping {{fontname=GlobeLogosOne codelist=GlobeLogosOne}}
```

Glyph list resources for simple fonts. Glyph lists (short for: glyph name lists) can be used to provide custom Unicode values for non-standard glyph names, or override the existing values for standard glyph names. A glyph list is a text file where each line describes a Unicode mapping for a single glyph name according to the following rules:

- Text after a percent sign '%' is ignored; this can be used for comments.
- The first column contains the glyph name. Any glyph name used in a font can be used (i.e. even the Unicode values of standard glyph names can be overridden). In order to use the percent sign as part of a glyph name the sequence `\%` must be used (since the percent sign serves as the comment introducer).
- At most one mapping for a particular glyph name is allowed; multiple mappings for the same glyph name is treated as an error.

- The remainder of the line contains up to 7 Unicode code points for the glyph name. The values can be supplied in hexadecimal notation with the prefix *x* or *ox*. UTF-32 is supported, i.e. surrogate pairs can be used.
- Unprintable characters in glyph names can be inserted by using escape sequences for text files (see Section 5.2, »Resource Configuration and File Searching«, page 61).

By convention, glyph lists use the file name suffix *.gl*. Glyph lists can be configured with the *glyphlist* resource. If no glyph list resource has been specified explicitly, TET will search for a file named <*myglyphlist*>*.gl* (where <*myglyphlist*> is the resource name) in the *searchpath* hierarchy (see Section 5.2, »Resource Configuration and File Searching«, page 61, for details). In other words: if the resource name and the file name (without the *.gl* suffix) are identical you don't have to configure the resource since TET will implicitly do the equivalent of the following call (where *name* is an arbitrary resource name):

```
set_option("glyphlist {name name.gl}");
```

Due to the precedence rules for glyph mapping, glyph lists are not consulted if the font contains a ToUnicode CMap. The following sample demonstrates the use of glyph lists:

```
% Unicode values for glyph names used in TeX documents

precedesequal   0x227C
similarequal    0x2243
negationslash   0x2044
union           0x222A
prime           0x2032
```

In order to apply a glyph list called *tarski.gl* to all font names starting with *CMSY* use the following option for *TET_open_document( ):*

```
glyphmapping {{fontname=CMSY* glyphlist=tarski}}
```

Rules for interpreting numerical glyph names in simple fonts. Sometimes PDF documents contain glyphs with names which are not taken from some predefined list, but are generated algorithmically. This can be a »feature« of the application generating the PDF, or may be caused by a printer driver which converts fonts to another format: sometimes the original glyph names get lost in the process, and are replaced with schematic names such as *G00, G01, G02,* etc. TET contains builtin glyph name rules for processing numerical glyph names created by various common applications and drivers. Since the same glyph names may be created for different encodings you can provide the *encodinghint* option to *TET_open_document( )* in order to specify the target encoding for schematic glyph names encountered in the document. For example, if you know that the document contains Russian text, but the text cannot successfully be extracted for lack of information in the PDF, you can supply the option *encodinghint= cp1250* to specify a Cyrillic codepage.

In addition to the builtin rules for interpreting numerical glyph names you can define custom rules with the *fontname* and *glyphrule* suboptions of the *glyphmapping* option of *TET_open_document( )*. You must supply the following pieces of information:

- The full or abbreviated name of the font to which the rule is applied *(fontname* option)
- A prefix for the glyph names, i.e. the characters before the numerical part *(prefix* suboption)

- The base (decimal or hexadecimal) in which the numbers are interpreted *(base* suboption)
- The encoding in which to interpret the resulting numerical codes *(encoding* suboption)

For example, if you determined (e.g. using PDFlib FontReporter) that the glyphs in the fonts *T1*, *T2*, *T3*, etc. are named *c00*, *c01*, *c02*, ..., *cFF* where each glyph name corresponds to the WinAnsi character at the respective hexadecimal position *(00, ..., FF)* use the following option for *TET_open_document( )*:

```
glyphmapping {{fontname=T* glyphrule={prefix=c base=hex encoding=winansi} }}
```

External font files and system fonts. If a PDF does not contain sufficient information for Unicode mapping and the font is not embedded, you can configure additional font data which TET will use to derive Unicode mappings. Font data may come from a TrueType or OpenType font file on disk, which can be configure with the *fontoutline* resource category. As an alternative on macOS and Windows systems, TET can access fonts which are installed on the host operating system. Access to these host fonts can be disabled with the *usehostfonts* option in *TET_open_document( )*.

In order to configure a disk file for the *WarnockPro* font use the following call:

```
set_option("fontoutline {WarnockPro WarnockPro.otf}");
```

See Section 5.2, »Resource Configuration and File Searching«, page 61, for more details on configuring external font files.

8 Image Extraction

8.1 Image Extraction Basics

Image formats. TET extracts raster images from PDF pages and stores the extracted images in one of the following formats:

- TIFF *(.tif)* images are created in most cases. The majority of TIFF images created by TET are compatible with all TIFF viewers and consumers. However, some advanced TIFF features are not supported by all image viewers, especially additional spot color channels (see »Spot colors«, page 130). We regard Adobe Photoshop as benchmark for the validity of TIFF images.
- JPEG *(.jpg)* is created for images which are compressed with the JPEG algorithm *(DCTDecode* filter) in PDF. JPEG-compressed image data in the PDF document is validated unless the validation has been disabled with the option *validatejpeg=false* in *TET_write_image_file( )* or *TET_get_image_data( ),* which may slightly speed up processing. In some cases DCT-compressed images are extracted as TIFF since not all PDF color spaces can be expressed in JPEG (e.g. spot colors).
- JPEG 2000 is created for images which are compressed with the JPEG 2000 algorithm *(JPXDecode* filter) in PDF. JPEG 2000 images come in different flavors. The main flavor with MIME type *image/jp2* and file name suffix *.jp2* is encoded according to ISO 15444-1 (Annex I). The extended flavor with MIME type *image/jpx* is encoded according to ISO 15444-2 (Annex M). It supports additional features such as CMYK and Lab color and uses *.jpf* as file name suffix (note the difference between MIME type and recommended suffix). Finally, raw JPEG 2000 code streams contain only the bare pixel data without any additional properties such as color space information. They are extracted with file name suffix *.j2k*.
  Applications which cannot handle JPEG 2000 output can avoid this extraction format with the document option *allowjpeg2000=false*. In this case 8-bit or 16-bit TIFF images are created instead of JPEG 2000, which may result in larger output. TIFF images for JPX-compressed data are also created if spot color information must be preserved or if image merging is involved. If a JPX-compressed image is extracted as TIFF, implicit internal ICC profiles in the JPX stream are ignored. For example, sRGB JPEG 2000 images are extracted as plain RGB TIFF.
- JBIG2 *(.jbig2)* is created for images which are compressed with the JBIG2 algorithm *(JBIG2Decode* filter) in PDF. JBIG2 files are created with »sequential organization« according to ISO 14492.

Extracting images to disk or memory. The TET API can deliver the images extracted from PDF documents in two different ways:

- The *TET_write_image_file( )* API method creates an image file on disk. The base file name of this image file must be specified in the *filename* option. TET will automatically add a suitable suffix depending on the image format.
- The *TET_get_image_data( )* API method delivers the image data in memory. This is convenient if you want to pass on the image data to another processing component without having to deal with disk files.

Details depend on your image extraction requirements (see Section 8.2.2, »Page-based and Resource-based Image Retrieval«, page 123). In both cases you can determine the type of the extracted image (see next section).

Determine the file format and name of extracted images. The image file type is reported in the *Image/@extractedAs* attribute in TETML. At the API level you can use the following code to determine the type of an extracted image:

```
int imageType = tet.write_image_file(doc, tet.imageid, "typeonly");

/* Map the numerical image type to a format suffix */
String imageSuffix;
switch (imageType) {
case TET.IF_TIFF:
        imageSuffix = ".tif";
        break;

case TET.IF_JPEG:
        imageSuffix = ".jpg";
        break;

case TET.IF_JP2:
        imageSuffix = ".jp2";
        break;

case TET.IF_JPF:
        imageSuffix = ".jpf";
        break;

case TET.IF_J2K:
        imageSuffix = ".j2k";
        break;

case TET.IF_JBIG2:
        imageSuffix = ".jbig2";
        break;

default:
        System.err.println("write_image_file() returned unknown value "
            + imageType + ", skipping image, error: "
            + tet.get_errmsg());
}
```

The image file name is reported in the *Image/@filename* attribute in TETML. At the API level you can supply the image file name to *TET_write_image_file()*.

The structure of the image file names produced by the TET command-line tool is documented in Section 2.1, »Command-Line Options«, page 19.

XMP metadata for images. PDF uses the XMP format to attach metadata to the whole document or parts of it. You can find more information about XMP on *www.pdflib.com*.

An image object may have XMP metadata associated with it in the PDF document. You can check the presence of image XMP in Acrobat XI/DC as follows:

- Click *View, Show/Hide, Navigation Panes, Content*.
- Locate the image in the tree structure, right-click on it and select *Show Metadata...* .

- The image is highlighted and the XMP panel pops up which displays XMP metadata for the selected image.

If XMP metadata is present, TET by default embeds it in the extracted image for the output formats JPEG and TIFF. This behavior can be controlled with the *keepxmp* option of *TET_write_image_file()* and *TET_get_image_data()*. If this option has been set to *false*, TET ignores image metadata when generating the image output file.

If image metadata is available, TET attaches a *Metadata* element to the image in the TETML output. This behavior can be controlled with the *tetml={elements={metadata}}* image option.

The *image_metadata* topic in the pCOS Cookbook demonstrates how to extract image metadata with the pCOS interface directly, without generating any image file.

TET implements a special heuristic for XMP image metadata which bypasses the usual PDF method for attaching XMP to an image object, but uses an alternate method based on marked content properties. This construct is typically generated by Adobe InDesign. Note that this kind of image XMP is not available via pCOS, but only in TETML and the extracted image files.

Artifact identification or removal. Artifacts in Tagged PDF designate irrelevant text or images. By default Artifacts are extracted like regular content. However, Artifacts can be removed with the page option *ignoreartifacts*.

Alternatively, Artifact images can be identified with the *attributes* member of the *TET_image_info* structure (see Table 10.19, page 205) or the *PlacedImage/@artifact* attribute in TETML. The Artifact status is also reflected in the names of the generated image files when extracting images with the TET command-line tool with *--imageloop page* (see »Image file names«, page 21).

Restrictions. In some cases the shape of extracted images may appear different from its rendering on the PDF page:

- Images may appear mirrored horizontally (upside down) or vertically. This is caused by the fact that TET extracts the original pixel data of the image, without respect to any transformation which may have been applied to the image on the PDF page.
- Masking effects achieved by applying a soft mask to another image are not visible in the extracted image. However, you can extract the mask as a separate image.

8.2 Extracting Images

8.2.1 Placed Images and Image Resources

TET distinguishes between placed images and image resources:

- A *placed image* corresponds to an image on a page. A placed image has geometric properties: it is placed at a certain location and has a size (measured in points, millimeters, or some other absolute unit). In most cases the image is visible on the page, but in some cases it may be invisible because it is obscured by other objects on the page, is placed outside the visible page area, is fully or partially clipped, etc. Placed images are represented by the *PlacedImage* element in TETML. Processing of placed images is subject to the *clippingarea*, *excludebox*, and *includebox* options.
- An *image resource* is a resource which represents the actual pixel data, color space and number of components, number of bits per component, etc. Unlike placed images, image resources don't have any intrinsic geometry. However, they do have width and height properties (measured in pixels). Each image resource has a unique ID which can be used to extract its pixel data. Image resources are represented by the *Image* element in TETML. Processing of image resources is not subject to the *clippingarea*, *excludebox*, and *includebox* options.

An image resource may be used as the basis for an arbitrary number of placed images in the document. Commonly each image resource is placed exactly once, but it could also be placed repeatedly on the same page or on multiple pages. For example, consider an image for a company logo which is used repeatedly on the header of each page in the document. Each logo on a page constitutes a placed image, but all those placed images may be created by the same image resource in an optimized PDF. On the other hand, in a non-optimized PDF each placed logo could be based on its own copy of the same image resource. This would result in the same visual appearance, but a larger PDF document. Non-optimized PDF documents may even contain image resources which are not even referenced on any page (i.e. unused resources).

Table 8.1 compares various aspects of placed images and image resources.

How many images are in a document? Surprisingly, there is no simple answer to this question. The answer depends on the following decisions:

- Do you want to count image resources or placed images?
- Do you want to count images which are only used as parts of merged images, but are never placed isolated?
- Do you want to count images which are only used as a mask for another image?
- Do you want to count irrelevant images (Artifacts)?
- Do you want to count images in annotations, patterns and soft masks (see »Sources of images in PDF«, page 124)?

Using TET and pCOS pseudo objects you can determine all variants of the image count answer. The *image_count* topic in the TET Cookbook demonstrates various possibilities of image counting. It generates output like the following:

```
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
```

Table 8.1 Comparison of placed images and image resources

property	*placed images*	*image resources*
TETML element	`PlacedImage`	`Image`
associated with a page	*yes*	–
width and height in pixels	*yes*	*yes*
geometry: width and height in points and position on the page	*yes*	–
may be marked as Artifact	*yes: the* TET_ATTR_ARTIFACT *bit in the* `attributes` *member returned by* TET_get_image_info( ) *and* `PlacedImage/@artifact` *attribute in TETML*	–
number of appearances	*1*	*0, 1, or more*
unique ID	*no: the* `imageid` *member returned by* TET_get_image_info( ) *and the* `PlacedImage/@image` *attribute in TETML identify the underlying image resource*	*yes:* `imageid` *member returned by* TET_get_image_info( ) *and* `Image/@id` *attribute in TETML*
file name convention in the TET command-line tool	`<filename>_p<pagenumber>_<imagenumber>.[tif|jpg|jp2|jpf|j2k|jbig2]`	`<filename>_I<imageid>.[tif|jpg|jp2|jpf|j2k|jbig2]`
handling of image masks in the TET command-line tool	*masks are extracted as* `<filename>_p<pagenumber>_<imagenumber>_mask.[tif|jpg|jp2|jpf|j2k|jbig2]`	*masks are extracted according to their own image ID without additional labels in the file name*

```
  consumed images: 81
No of relevant (normal or artificial) image resources: 2
```

8.2.2 Page-based and Resource-based Image Retrieval

The distinction between placed images and image resources gives rise to two fundamentally different approaches to image extraction: page-based and resource-based image extraction loops. Both methods can be used to extract images to a disk file or to memory.

Page-based image extraction. In this case the application is interested in the exact page layout and placed images, but doesn't care about duplicated image data. Extracting images with a page-based loop creates an image file for each placed image, and may result in the same image data for more than one extracted placed image. The application could avoid image duplication by checking for duplicate image IDs. However, unique image resources can more easily be extracted with the resource-based image extraction loop (see below).

The page-based image extraction loop can be activated in the TET command-line tool with the option *--imageloop page*. Code for page-based image extraction at the API level is demonstrated in the *images_per_page* Cookbook topic and sample. These samples also show how to retrieve the image geometry.

Details of the page-based image extraction loop (please refer to the sample code mentioned above): *TET_get_image_info( )* retrieves geometric information about a placed image as well as the pCOS image ID (in the *imageid* field) of the underlying image data. This ID can be used to retrieve more image details with *TET_pcos_get_number( )*, such as the color space, width and height in pixels, etc., as well as the actual pixel data

with *TET_write_image_file()* or *TET_get_image_data()*. *TET_get_image_info()* does not touch the actual pixel data of the image. If the same image is referenced multiply on one or more pages, the corresponding IDs are the same.

Resource-based image extraction. In this case the application is interested in the image resources of the document, but doesn't care which image is used on which page. Image resources which are placed more than once (on one or more pages) are extracted only once. On the other hand, image resources which are not placed at all on any page are also extracted.

The resource-based image extraction loop can be activated in the TET command-line tool with the option *--imageloop resource*. Code for resource-based image extraction at the API level is demonstrated in the *image_resources* sample and Cookbook topic.

Details of the resource-based image extraction loop (please refer to the sample code mentioned above): all pages must be opened before extracting image resources to make sure that image merging has been performed; if image merging is not relevant this step can be skipped. In order to extract an image, the corresponding image ID is required. The code enumerates all values from 0 to the highest image ID which is queried *as follows:*

```
n_images = (int) tet.pcos_get_number(doc, "length:images");
```

In order to skip the consumed parts of merged images (e.g. the strips of a multi-strip image) the type of each image resource is examined with the *mergetype* pCOS pseudo object. This allows us to skip images which have been consumed by the image merging process (since we are only interested in the resulting merged image). Once an image ID has been determined, *TET_write_image_file()* or *TET_get_image_data()* can be called to write the image data to a disk file or pass the pixel data in memory.

Sources of images in PDF. A PDF document may contain images in various places, some of which are not necessarily expected. TET searches all of the following locations for images:

- The most common source is the page description itself.
- An annotation or form field may contain an appearance stream with arbitrary graphics including raster images. Such images can be identified with the *TET_ATTR_ANNOTATION* flag in the attributes field returned by *TET_get_image_info()* and the *source="annoation"* attribute of the *PlacedImage* element in TETML. Annotation processing can be disabled with the document option *engines={annotation=false}*.
- A tiling pattern may contain arbitrary text and graphics including raster images. Such images can be identified with the *TET_ATTR_PATTERN* flag in the attributes field returned by *TET_get_image_info()* and the *source="pattern"* attribute of the *PlacedImage* element in TETML. Since patterns are rendered repeatedly to tile a filled object, the image position may not match the visible position on the page. TET places the pattern at the lower left corner of the page before extracting contents.
- A graphics state may contain a Transparency group XObject describing a soft mask with arbitrary graphics including raster images (this technique is often used for drop shadow effects). Such images can be identified with the *TET_ATTR_SOFTMASK* flag in the attributes field returned by *TET_get_image_info()* and the *source="softmask"* attribute of the *PlacedImage* element in TETML.

In TETML the source location of an image can be identified with the attribute *PlacedImage/@source*.

8.2.3 Geometry of Placed Images

You can retrieve geometric information for each placed image. The following values are available for each image in the information returned by *TET_get_image_info( )* (see Figure 8.1):

- The *x* and *y* fields are the coordinates of the image reference point. The reference point is usually the lower left corner of the image. However, coordinate system transformations on the page may result in a different reference point. For example, the image may be mirrored horizontally with the result that the reference point becomes the upper left corner of the image. The value of *y* is subject to the *topdown* page option.
- The *width* and *height* fields correspond to the physical dimensions of the placed image on the page. They are provided in points (i.e. 1/72 inch).
- The angle *alpha* describes the direction of the pixel rows. This angle is in the range $-180° < alpha \leq +180°$. The angle *alpha* rotates the image at its reference point. For upright images *alpha* is 0°. The values of *alpha* and *beta* are subject to the *topdown* page option.
- The angle *beta* describes the direction of the pixel columns, relative to the perpendicular of *alpha*. This angle is in the range $-180° < beta \leq +180°$, but different from ±90°. The angle *beta* skews the image, and *beta*=180° mirrors the image at the *x* axis. For upright images *beta* is in the range $-90° < beta < +90°$. If *abs(beta)* > 90° the image is mirrored at the baseline.
- The *imageid* field contains the pCOS ID of the image. It can be used to retrieve detailed image information with pCOS functions and the image pixel data with *TET_write_image_file( )* or *TET_get_image_data( )*.
- The *attributes* field contains the bit *TET_ATTR_ARTIFACT* which is set if the image is marked as an Artifact (irrelevant content). This field can also contain the bit values *TET_ATTR_ANNOTATION, TET_ATTR_PATTERN and TET_ATTR_SOFTMASK* which describe where the image was found (see »Sources of images in PDF«, page 124).

As a result of image transformations, the orientation of the extracted images may appear wrong since the extracted image data is based on the image resource in the PDF. Any rotation or mirror transformations applied to the placed image on the PDF page are not applied to the extracted pixel data. Instead, the original pixel data is extracted.

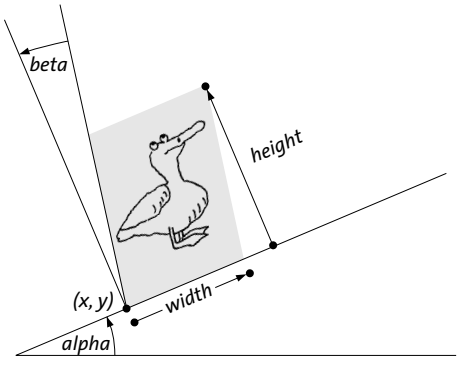


Fig. 8.1
Image geometry

Calculating the coordinates of all image corners. The *x* and *y* fields retrieved with *TET_get_image_info()* provide the coordinates of the image's reference point. Using the image's *x/y, width/height* and *alpha/beta* values you can calculate the coordinates of all image corners as follows:

ll_x = x
ll_y = y

lr_x = x + width * cos(alpha)
lr_y = y + width * sin(alpha)

ul_x = x + dir * height * (tan(beta)*cos(alpha) - sin(alpha))
ul_y = y + dir * height * (tan(beta)*sin(alpha) + cos(alpha))

ur_x = x + width * cos(alpha) + dir * height * (tan(beta)*cos(alpha) - sin(alpha))
ur_y = y + width * sin(alpha) + dir * height * (tan(beta)*sin(alpha) + cos(alpha))

with *dir=1* in the default case *topdown={output=false}*. In topdown coordinates, i.e. if *topdown={output= true}* (see »Top-down coordinate system«, page 74), you must set *dir=-1* and the corners are swapped, i.e. *ll* must be swapped with *ul*, and *lr* with *ur*.

While the initial reference point in PDF is the lower left corner of the image, this may no longer be true depending on the transformations applied when the PDF page was generated. In order to determine the lower left (or other) corner of the image in the general case you must calculate all corners as shown above and determine the corner with the smallest *x/y* coordinates.

Image resolution. In order to calculate the image resolution in dpi (dots per inch) you must divide the image width in pixels by the image width in points and multiply by 72:

```
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;
        double yDpi = 72 * height / tet.height;
        ...
}
```

Note that dpi values for rotated or skewed images may be meaningless. Full code for image dpi calculations can be found in the *determine_image_resolution* topic in the TET Cookbook.

TET by default records a dummy resolution value of 72 dpi in generated TIFF images to satisfy the TIFF specification. The *dpi* option of *TET_write_image_file()* can be used to embed calculated resolution values instead. TET cannot embed calculated resolution values automatically since a particular image may have been placed more than once, each time with different size and therefore different resolution. The value *dpi=0* can be used to suppress the dummy resolution values.

The TET command-line tool embeds calculated resolution values when operating in the page-based image loop.

8.3 Merging Fragmented Images

Sometimes it is not desirable to extract images exactly as they are represented in the PDF document: in many situations what appears to be a single image is actually a collection of several smaller images which are placed adjacent to each other. There are some common reasons for such image fragmentation:

- Some applications and drivers convert multi-strip TIFF images to fragmented PDF images. The number of strips can range from dozens to hundreds.
- Some scanning software divides scanned pages in smaller fragments (strips or tiles). The number of fragments is usually not more than a few dozen.
- Some applications break images into small pieces when generating print or PDF output. In extreme cases, especially documents created with Microsoft Office applications, a page may contain thousands of small image fragments.
- Some page layout programs, e.g. Adobe InDesign, cut images into smaller and sometimes irregular fragments when creating PDF output (see Figure 8.2).

TET's image merging engine detects this situation and recombines the image parts to form a larger and more useful image. If the merging candidates can be combined to a larger image, they are merged. Transparency flattening may lead to irregular image layouts which are impossible to merge to a rectangular result.

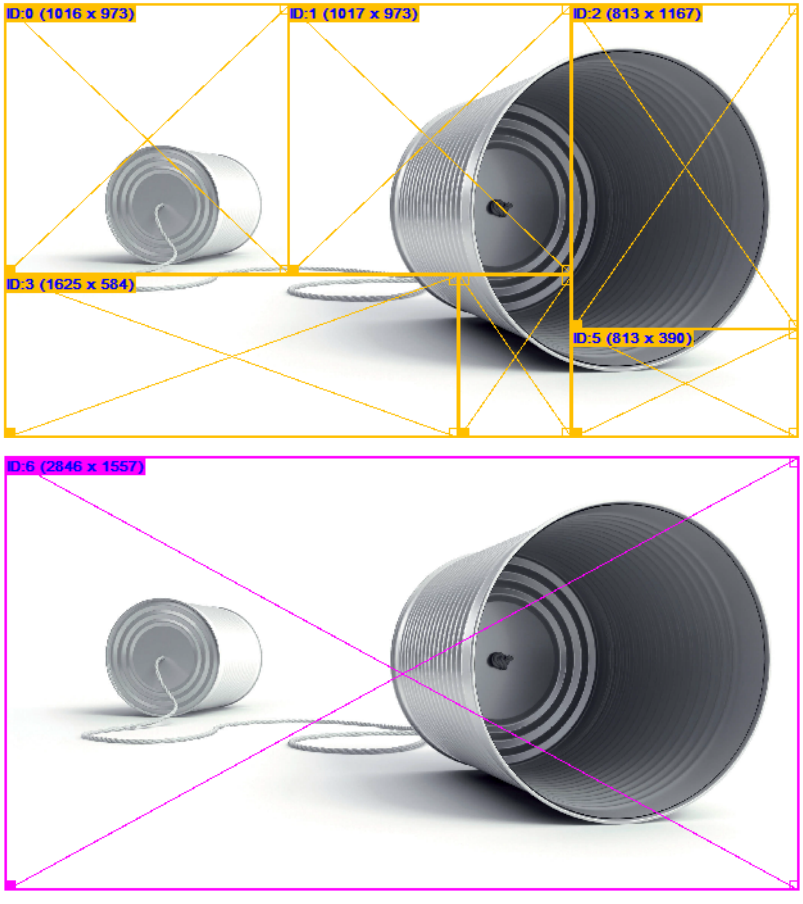


Fig. 8.2
Although this image is segmented into smaller parts (top), TET extracts it as a single reusable image (bottom).

In order to disable image merging use the following page option:

```
imageanalysis={merge={disable}}
```

Merged images in pCOS. Merged images can be identified by the pCOS pseudo object *images[ ]/mergetype*: it has the value 1 *(artificial)* for merged images and 2 *(consumed)* for images which have been consumed by the merging process. Consumed images should generally be ignored by the receiving application.

Gaps and overlap between images. In order to compensate for inaccuracies in the image locations some amount of gap or overlap is allowed between adjacent images. By default, images are merged if the gap or overlap is smaller than one point. This value can be modified with the following page option:

```
imageanalysis={merge={gap=2}}
```

Larger gap/overlap values are often required when extracting images from newspapers or magazines.

When are images merged? Analyzing and merging images on a page are triggered by the corresponding call to *TET_open_page( )*. This leads to the following consequences:

- The number of entries in the pCOS *images[ ]* array, i.e. the value of the *length:images* pseudo object, may increase: as more pages are processed, artificial images which result from image merging are added to the array. In order to extract all merged images via the *images[ ]* array you must therefore open all pages in the document before querying *length:images* and extracting image data. Artificial (merged) images are marked with the corresponding flag *artificial* (numerical value 1) in the *images[ ]/mergetype* pseudo object.
- On the other hand, some elements in the *images[ ]* array may only be consumed as parts of merged images, but are not used as images in their own right. Such entries are never removed from the *images[ ]* array, but the consumed entries are marked with the corresponding flag *consumed* (numerical value 2) in the *images[ ]/mergetype* pseudo object.

8.4 Small and Large Image Filtering

By default, TET ignores small images since these are often irrelevant or useless. Since the image merging process often combines small image fragments to a larger image, small images are removed after image merging. Only images which cannot be merged to form a larger image are candidates for small image removal. In addition, they must satisfy the size conditions which are specified in the *heightrange/sizerange/widthrange* suboptions of the *imageanalysis* page option. In order to completely disable image filtering use the following page option:

```
imageanalysis={sizerange={1 unlimited}}
```

By default, images with a width or height below 20 pixels are ignored. If an image itself is small, but has a larger mask (i.e. the mask dimensions are within the specified range) attached to it, the image and its mask are not ignored.

Alternatively to the page option above, the same option list can be applied as document option. While size filtering is applied in the same way, small images are removed before attempting image merging, i.e. TET will no longer be able to merge such small image fragments. Disabling small images on the document level can improve performance in situations with very large numbers of small images at the expense of losing image merging.

Small and large images in pCOS. Images which have been removed according to the *heightrange/sizerange/widthrange* options are ignored by *TET_write_image_file()* and *TET_get_image_data()*, but are still present in the pCOS *images[ ]* array. They can be identified with the pCOS pseudo object *images[ ]/small* (the keyword *small* is used regardless of whether the images were filtered because they are too small or too large).

8.5 Image Colors and Masking

8.5.1 Color Spaces

Image color fidelity. Table 6.1 provides an overview of PDF color spaces. All color spaces are supported for images. TET does not degrade image quality when extracting images:

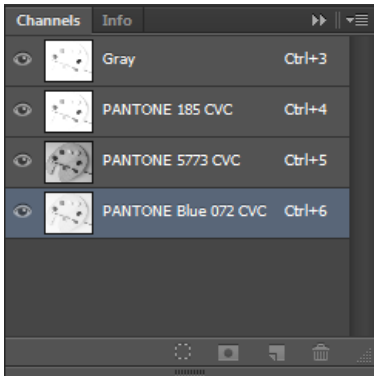
- Raster images are never downsampled.
- The color space of an image is retained in the output. TET never applies any CMYK-to-RGB or similar color conversion.

ICC profiles. An image in PDF may have an ICC profile assigned which allows precise color reproduction. By default, TET processes attached ICC profiles and embeds them in the generated TIFF or JPEG image files. You can disable ICC profile embedding with the option *keepiccprofile=false* in *TET_write_image_file( )* and *TET_get_image_data( )*. This reduces the size of the image files at the expense of color fidelity. Disabling ICC profile embedding is not recommended for workflows which need precise color representation.

Spot colors. Images in PDF may be colorized with a named color. Usually named colors are used to specify custom spot colors, but the same mechanism can also be used to apply a subset of CMYK process colors to an image (e.g. only the Cyan and Magenta channels). The *Separation* color space in PDF holds a single named color, while the *DeviceN* color space can be used to assign multiple named colors. Separation colors are accompanied by a so-called alternate color which makes it possible to represent the color even if the spot color is not available (e.g. on a monitor). For example, if a Separation color is called *Company Red* it is useful to have an alternate representation in a well-known color space such as RGB or CMYK to display the spot color on devices where *Company Red* is not available as named color.

TET extracts images with *Separation* or *DeviceN* colors as follows: CMYK process color names are identified: if a named color is called *Black* it is treated as process color and the image is extracted as grayscale image. The color names *Cyan*, *Magenta* and *Yellow* are also identified and the image is extracted as CMYK image. Custom spot color names, i.e. names different from *Cyan*, *Magenta*, *Yellow* and *Black* can be handled in different ways subject to the document option *spotcolor:*

- With *spotcolor=convert* (which is default) spot colors are converted to the corresponding alternate color space if possible. If such a conversion is not possible this method behaves like *spotcolor=ignore* (for a single custom spot color) or *spotcolor= preserve* (for two or more custom spot colors).
- The option *spotcolor=ignore* is similar to *spotcolor=convert* except that images with exactly one custom spot color are extracted as grayscale image and the spot color name is lost.
- With *spotcolor=preserve* spot color names are preserved, and the image is extracted as grayscale or CMYK image with one or more extra spot color channels. This requires TIFF output; the generated TIFF flavor can be viewed with Adobe Photoshop and compatible programs (see Figure 8.3). Simple TIFF viewers often ignore the extra spot color channels.



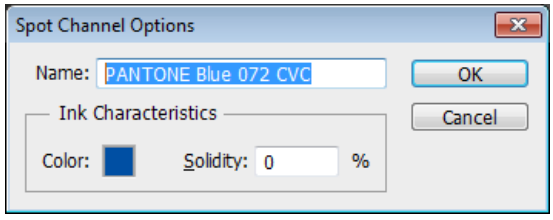


Fig. 8.3
Adobe Photoshop displays spot color channels of TIFF images which have been extracted with `spotcolor=preserve` *in the* `Channels` *window (left). Double-clicking one of the icons reveals the alternate color (top).*

Table 8.2 summarizes the output formats for different combinations of spot color names and settings of the document option *spotcolor*.

Table 8.2 Output formats for images with `Separation` *and* `DeviceN` *colors*

Separation or DeviceN color names	***spotcolor=ignore***	***spotcolor=convert***	***spotcolor=preserve***
only `Black`	*grayscale*		
one or more of `Cyan`, `Magenta`, `Yellow`, `Black`	*CMYK (unused channels are empty)*		
exactly one custom spot color (i.e. different from `Cyan`, `Magenta`, `Yellow`, `Black`*)*	*grayscale*	*alternate color space if possible*[1]	*empty grayscale channel plus a named extra channel*
two or more color names and all are different from `Cyan`, `Magenta`, `Yellow`	*alternate color space if possible*[1]		*grayscale channel plus one or more named extra channels*
two or more color names including one or more of `Cyan`, `Magenta`, `Yellow`	*alternate color space if possible*[1]		*CMYK plus one or more named extra channels*

1. Behaves like `spotcolor=ignore` *(for a single custom spot color) or* `spotcolor=preserve` *(for two or more custom spot colors) if conversion to the alternate color space is not possible.*

8.5.2 Image Masks and Soft Masks

Masking information and the actual image data used for masking another image can be retrieved with TET. PDF supports the following types of image masking:

- A stencil mask is a 1-bit image with the PDF key *ImageMask*. The image is used as a stencil which is partly opaque and partly transparent: by default, color is applied where the image has pixel value 0, and the background shines through unchanged where the image has pixel value 1.
- A mask is a 1-bit grayscale image which is applied to another image (PDF key *Mask)*. It specifies which image areas shall be painted and which shall be masked out (left unchanged).
- A soft mask is a grayscale image of arbitrary bit depth which is applied to another image (PDF key *SMask)*. It provides a smooth transition between the masked image and its background, creating a real transparency effect.

Since hard and soft masks differ only in bit depth, they are treated uniformly in TET.

Image masks in TETML. Image masking is handled as follows in TETML:

- Stencil masks: the TETML attribute *Image/@stencilmask* signals that a 1-bit image itself is used as a stencil mask.
- Masks: the TETML attribute *Image/@maskid* references an image mask *(Mask* or *SMask)* which may be attached to an image. Details of the mask image can be retrieved in the mask image's entry in the *images[ ]* array.

Image masks in the TET command-line tool. Image masks are handled as follows in the TET command-line tool (information about stencil masking is not available):

- Extracting images with *--imageloop page* extracts all plain images as usual. Images used as mask for one of the extracted plain images are also extracted using the suffix *_mask* in the image file name.
- Extracting images with *--imageloop resource* extracts all plain images and mask images. The generated file names include the *image/@id* TETML attribute of the mask image (which is identical to the *image/@maskid* attribute of the masked image) so that applications can locate the corresponding files for images referenced in TETML.

Image masks in pCOS. Image masking is handled in the pCOS pseudo object*images[ ]* and *TET_pcos_get_number( )* as follows:

- Images which are used as stencil mask can be identified by the *images[ ]/stencilmask* pseudo object.
- If an image has a soft mask assigned the corresponding *images[ ]/maskid* pseudo object has a value different from -1. The value designates the image ID of the mask and can be used to query further details of the mask using the corresponding entry in the *images[ ]* array.

Image masks in the API. Image masking is handled as follows in the TET API:

- *TET_get_image_info( )* enumerates only plain images which are placed on the page, and skips masks. The *imageid* field in the *image_info* structure can be used to obtain the image's pCOS id, which in turn can be used to query mask and stencil mask information via pCOS as described above.
- *TET_write_image_file( )* and *TET_get_image_data( )* can be used to retrieve the pixel data of the mask, using the image id retrieved with the *maskid* pCOS object of the masked image. This is demonstrated in the *images_per_page* sample. Alternatively, you can iterate over all entries in the pCOS *images[ ]* array to create image files for all plain images and mask images. This is demonstrated in the *image_resources* sample.

9 TET Markup Language (TETML)

9.1 Creating TETML

As an alternative to supplying the contents of a PDF document via a programming interface, TET can create XML output. We refer to the XML output created by TET as TET Markup Language (TETML). TETML contains the text contents of the PDF pages plus optional information such as text position, font, font size, etc. If TET detects table-like structures on the page the tables are expressed in TETML as a hierarchy of table, row, and cell elements. Note that table information is not available via the TET programming interface, but only through TETML. TETML also contains information about images and color spaces as well as annotations, form fields, bookmarks and other interactive elements.

You can convert PDF documents to TETML with the TET command-line tool or the TET library. In both cases there are various options available for controlling details of TETML generation.

Creating TETML with the TET command-line tool. Using the TET command-line tool you can generate TETML output with the *--tetml* option. The following command creates a TETML output document *file.tetml:*

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

You can use various options to convert only some pages of the document, supply processing options, etc. Refer to Section 2.1, »Command-Line Options«, page 19, for more details.

Creating TETML with the TET library. Using a simple sequence of API calls you can generate TETML output with the TET library. The *tetml* sample demonstrates the canonical code sequence for generating TETML. This sample program is available in all supported language bindings.

TETML is created page by page, which means that the client may choose to process only a subset of pages. The TETML trailer must be created after processing the last page:

```
final int n_pages = (int) tet.pcos_get_number(doc, "length:pages");

/* Loop over all pages in the document */
for (int pageno = 1; pageno <= n_pages; ++pageno)
{
        tet.process_page(doc, pageno, pageoptlist);
}

/* This could be combined with the last page-related call */
tet.process_page(doc, 0, "tetml={trailer}");
```

If the *filename* option has been supplied to *TET_open_document()* the TETML output is written to the specified disk file. Otherwise TETML is accumulated in memory and can be fetched with *TET_get_tetml()*. This can be done for the full TETML stream in a single call (only recommended for small documents), or with multiple calls where each call retrieves a smaller chunk of the full TETML stream.

The generated TETML stream can be parsed into a XML tree using the XML support provided by most modern programming languages. Processing the TETML tree is also demonstrated in the *tetml* sample programs for language bindings with integrated XML support.

What's included in TETML? TETML output is encoded in UTF-8 (on IBM Z with USS or MVS: EBCDIC-UTF-8) and includes the following information (some of these items are optional):

- general document information, encryption status, PDF standards, Tagged PDF etc.
- document info fields and XMP metadata
- text contents of each page (words or paragraphs; optionally lines)
- font, geometry and color of the glyphs
- layout attributes for the glyph (sub/superscript, dropcap, shadow)
- hyphenation attributes
- artifact (irrelevant content) status of text and images
- structure information, e.g. tables and lists
- information about placed images on the page
- resource information, i.e. fonts, color spaces, images, ICC profiles
- output intent information
- layer information (also known as optional content)
- interactive elements: bookmarks, named destinations, annotations, form fields, actions, and JavaScript
- anchors are provided in the text stream for easy reference of links, form fields, and bookmark targets
- digital signatures
- error messages if an exception occurred during PDF processing

Various elements and attributes in TETML are optional. See Section 9.3, »Controlling TETML Details«, page 139, for details.

9.2 TETML Examples

The TETML samples below demonstrate some important features. The full list of TETML elements along with descriptions can be found in Section 9.4, »TETML Elements and the TETML Schema«, page 143.

Document header and text output. The following fragment shows the most important parts of a TETML document:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Created by the PDFlib Text and Image Extraction Toolkit TET (www.pdflib.com) -->
<TET xmlns="http://www.pdflib.com/XML/TET5/TET-5.0"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://www.pdflib.com/XML/TET5/TET-5.0
 http://www.pdflib.com/XML/TET5/TET-5.0.xsd"
 version="5.2">
<Creation platform="Win64" tetVersion="5.2" date="2019-07-05T18:26:02+02:00" />
<Document filename="TET-datasheet.pdf" pageCount="6" filesize="508093" linearized="true"
pdfVersion="1.7">
<DocInfo>
<Author>PDFlib GmbH</Author>
<CreationDate>2015-08-05T17:43:14+02:00</CreationDate>
<Creator>Adobe InDesign CS6 (Windows)</Creator>
<ModDate>2015-08-05T17:43:15+02:00</ModDate>
<Producer>Adobe PDF Library 10.0.1</Producer>
<Subject>PDFlib TET: Text and Image Extraction Toolkit (TET)</Subject>
<Title>PDFlib TET datasheet</Title>
</DocInfo>
<Metadata>
<x:xmpmeta xmlns:x="adobe:ns:meta/" x:xmptk="Adobe XMP Core 5.3-c011 66.145661, 2012/02/
06-14:56:27        ">
   <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
   ...XMP metadata...
   </rdf:RDF>
</x:xmpmeta>
</Metadata>
<Options> tetml={filename={TET-datasheet.word.tetml}}</Options>
<Pages>
<Page number="1" width="595.28" height="841.89">
<Options> granularity=word tetml={}</Options>
<Content granularity="word" dehyphenation="false" dropcap="false" font="false"
geometry="false" shadow="false" sub="false" sup="false">
<Para>
 <Box llx="235.80" lly="796.02" urx="397.67" ury="816.72">
  <Word>
   <Text>PDFlib</Text>
   <Box llx="235.80" lly="796.02" urx="291.91" ury="814.02"/>
  </Word>
  <Word>
   <Text>datasheet</Text>
   <Box llx="306.14" lly="796.22" urx="397.67" ury="816.72"/>
  </Word>
 </Box>

...more page contents...

</Content>
```

```
</Page>
 ...more pages...
<Resources>
<Fonts>
 <Font id="F0" name="TheSans-Plain" fullname="FXLUMY+TheSans-Plain" type="Type 1 CFF"
embedded="true" ascender="1170" capheight="675" italicangle="0" descender="-433"
weight="400" xheight="497"/>
 <Font id="F1" name="PDFlibLogo2-Regular" fullname="DUMIKC+PDFlibLogo2-Regular"
type="Type 1 CFF" embedded="true" ascender="800" capheight="700" italicangle="0"
descender="-9" weight="400" xheight="537"/>
 ...more fonts...
</Fonts>
<Images>
 <Image id="I0" filename="TET-datasheet_I0.tif" extractedAs=".tif" width="885"
height="565" colorspace="CS3" bitsPerComponent="8"/>
 <Image id="I1" filename="TET-datasheet_I1.tif" extractedAs=".tif" width="1253"
height="379" colorspace="CS4" bitsPerComponent="8"/>
 ...more images...
</Images>
<ColorSpaces>
 <ColorSpace id="CS0" name="DeviceCMYK" components="4"/>
 <ColorSpace id="CS1" name="DeviceGray" components="1"/>
 ...more colorspaces...
</ColorSpaces>
</Resources>
...
</Pages>
</Document>
</TET>
```

Glyph coordinates and color. Depending on the selected TETML mode more glyph details can be expressed in TETML. TETML modes are discussed in »Selecting the TETML mode«, page 139. Here is a variation of the sample above with more glyph details. The *Glyph* element contains font, position and color information (see below for more details on color representation):

```
<Word>
 <Text>datasheet</Text>
 <Box llx="306.14" lly="796.22" urx="397.67" ury="816.72">
  <Glyph font="F0" size="20.5000" x="306.14" y="796.22" width="11.42" fill="C0">d</Glyph>
  <Glyph font="F0" size="20.5000" x="317.87" y="796.22" width="10.68" fill="C0">a</Glyph>
  <Glyph font="F0" size="20.5000" x="328.61" y="796.22" width="7.61" fill="C0">t</Glyph>
  <Glyph font="F0" size="20.5000" x="336.52" y="796.22" width="10.68" fill="C0">a</Glyph>
  <Glyph font="F0" size="20.5000" x="347.51" y="796.22" width="8.71" fill="C0">s</Glyph>
  <Glyph font="F0" size="20.5000" x="356.53" y="796.22" width="11.79" fill="C0">h</Glyph>
  <Glyph font="F0" size="20.5000" x="368.63" y="796.22" width="10.41" fill="C0">e</Glyph>
  <Glyph font="F0" size="20.5000" x="379.35" y="796.22" width="10.41" fill="C0">e</Glyph>
  <Glyph font="F0" size="20.5000" x="390.07" y="796.22" width="7.61" fill="C0">t</Glyph>
 </Box>
</Word>
```

Color values and color spaces. Colors are represented by a color space (e.g. D*eviceRGB)* and a color value. Color values for text are available for the fill and stroke colors. Since stroked glyphs are quite rare in PDF you will see the fill attribute more often. The color values for images come from the actual pixel data. Color spaces for text, vector graphics and images are listed in the *ColorSpaces* element in the *Resources* section. Each *ColorSpace*

element contains details depending on the type of color space. Some color spaces refer to others, e.g. an *Indexed* color space requires an underlying base color space, and *Separation* and *DeviceN* require an alternate color space:

```
<Resources>
<ColorSpaces>
 <ColorSpace id="CS0" name="DeviceCMYK" components="4"/>
 <ColorSpace id="CS1" name="DeviceGray" components="1"/>
 <ColorSpace id="CS2" name="Indexed" components="1" base="CS0" hival="255">
  <Lookup>0000000007050003493400297453004l6F50003E775600...</Lookup>
 </ColorSpace>
 <ColorSpace id="CS0" name="Separation" components="1" alternate="CS0">
  <Colorant name="PANTONE 294 CVC"/>
  <Function type="interpolate">
   ...
   <C1>
    <Value>0.93</Value>
    <Value>0.62</Value>
    <Value>0.00</Value>
    <Value>0.00</Value>
   </C1>
   <Exponent>1</Exponent>
  </Function>
 </ColorSpace>
 ...
</ColorSpaces>
</Resources>
```

Tables in TETML. Tables identified by TET are expressed with table, row and cell structure in TETML. Cells which span multiple columns are labeled with a *colSpan* attribute:

```
<Table>
 <Box llx="302.14" lly="639.72" urx="525.50" ury="731.50">
  <Row>
   <Box llx="311.64" lly="721.10" urx="521.50" ury="730.70"/>
   <Cell>
    <Box llx="311.64" lly="721.10" urx="375.22" ury="730.70"/>
    <Para>
     <Word>
      <Text>Device-dependent</Text>
      <Box llx="311.64" lly="721.90" urx="375.22" ury="729.90"/>
     </Word>
    </Para>
   </Cell>
   <Cell>
    <Box llx="397.91" lly="721.10" urx="431.99" ury="730.70"/>
    <Para>
     <Word>
      <Text>CIE-based</Text>
      <Box llx="397.91" lly="721.90" urx="431.99" ury="729.90"/>
     </Word>
    </Para>
   </Cell>
...
  <Row>
   <Box llx="306.14" lly="641.52" urx="516.67" ury="650.52"/>
   <Cell colSpan="3">
    <Box llx="306.14" lly="706.42" urx="516.67" ury="716.02"/>
```

```
    <Para>
     <Word>
      <Text>TET</Text>
      <Box llx="306.14" lly="641.52" urx="319.70" ury="650.52"/>
     </Word>
     <Word>
      <Text>.</Text>
      <Box llx="514.83" lly="641.52" urx="516.67" ury="650.52"/>
     </Word>
    </Para>
   </Cell>
  </Row>
 </Box>
</Table>
```

Interactive elements. Links, bookmarks, form fields etc. are also available in TETML as shown in the following example:

```
<Page number="6" width="595.27600" height="841.89000">
<Annotations>
<Annotation id="ANN0" type="Link" anchor="A0">
 <Box llx="327.14" lly="64.89" urx="395.08" ury="79.18"/>
 <Action type="URI" trigger="activate" URI="mailto:sales%40pdflib.com"/>
</Annotation>
<Annotation id="ANN1" type="Link" anchor="A1">
 <Box llx="327.14" lly="52.89" urx="391.05" ury="67.18"/>
 <Action type="URI" trigger="activate" URI="http://www.pdflib.com"/>
</Annotation>
</Annotations>
```

The text inside a link is wrapped with *A* (anchor) elements which provide the relationship between the geometrically defined PDF annotation and the corresponding page contents, i.e. the text which activates the link. Keep in mind that the active content does not need to correspond to complete semantic entities. For example, a link may span some fraction of a word or paragraph. Since anchors don't necessarily span complete TETML elements separate *start/stop* anchor elements are required instead of enclosing the link contents with a single *A* element:

```
<A id="A1" type="start"/>
<Word>
 <Text>www.pdflib.com</Text>
 <Box llx="327.14" lly="56.71" urx="391.05" ury="65.71"/>
</Word>
<A id="A1" type="stop"/>
```

9.3 Controlling TETML Details

TETML modes. TETML can be generated in various modes which include different amounts of font and geometry information, and differ regarding the grouping of text into larger units (granularity). The TETML mode can be specified individually for each page. Usually TETML files contain the data for all pages in the same mode. The following TETML modes include text and image information as well as interactive elements:

- *Glyph* mode is a low-level flavor which includes the text, font, coordinates, and color for each glyph, without any word grouping or structure information. It is intended for debugging and analysis purposes since it represents the original text information on the page.
- *Word* mode groups text into words and adds *Box* elements with the coordinates of each word. No font information is available. This mode is suitable for applications which operate on word basis. Punctuation characters will by default be treated as individual words, but this behavior can be changed with a page option (see »Word boundary detection for Western text«, page 87). Lines of text can optionally be identified with the *Line* element; this is controlled via the *tetml* page option.
- *Wordplus* mode is similar to *word* mode, but adds font and coordinate details plus color information for all glyphs in a word. The coordinates are expressed relative to the lower left or upper left corner subject to the *topdown* page option. *Wordplus* mode makes it possible to analyze font usage and track changes of font, font size, etc. within a word. Since *wordplus* is the only TETML mode which contains all relevant TETML elements it is suited for all kinds of processing tasks. On the other hand, it creates the largest amount of output due to the wealth of information contained in TETML.
- *Line* mode includes all text which comprises a line in a separate *Line* element. In addition, multiple lines may be grouped in a *Para* element. Line mode is recommended only in situations where the receiving application can only deal with line-based text input.
- *Page* mode includes structure information starting at the paragraph level, but does not include any font or coordinate details. Note that the layout detection results in page mode may be slightly different from word mode since anchors for images and destinations are treated differently.

If you are only interested in image information you can also skip other types of output in TETML:

- *Image* mode includes information about placed images and image resources, but not any text- or font-related elements nor information about interactive elements.

Table 9.1 lists the TETML elements which are present in the TETML modes.

Selecting the TETML mode. With the TET command-line tool (see Section 2.1, »Command-Line Options«, page 19) you can specify the desired mode as a parameter for the *--tetml* option. The following command generates TETML output in *wordplus* mode:

```
tet --tetml wordplus file.pdf
```

With the TET library the TETML mode cannot be specified directly, but as a combination of options:

- You can specify the amount of text in the smallest element with the *granularity* option of *TET_process_page()*.

Table 9.1 Text-related elements in various TETML modes; `PlacedImage` *and* `Image` *are always present.*

TETML mode	*structure*	*tables*	*text position*	*glyph details*
glyph	–	–	–	`Glyph`, `Color`
word	`Para`, `Word` *optionally:* `Line`, `List`	`Table`, `Row`, `Cell`	`Box` *inside* `Word` `Box` *inside* `Para`	–
wordplus	`Para`, `Word` *optionally:* `Line`, `List`	`Table`, `Row`, `Cell`	`Box` *inside* `Word` `Box` *inside* `Para`	`Glyph`, `Color`
line	`Para`, `Line`	–	`Box` *inside* `Para`	–
page	`Para`	`Table`, `Row`, `Cell`	`Box` *inside* `Para`	–
image	–	–	–	–

- For *granularity=glyph* or *word* you can additionally specify the amount of glyph details. With the *glyphdetails* suboption of the *tetml* option you can omit some parts of the glyph information if you don't need it.
- In order to suppress all text output (i.e. image mode) you can disable the text engine with the following document option:

```
engines={notext}
```

The following page option list generates TETML output in *wordplus* mode with all glyph details:

```
granularity=word tetml={ glyphdetails={all} }
```

Table 9.2 summarizes the options for creating TETML modes.

Table 9.2 Creating TETML modes with the TET library

TETML mode	*document options*	*options of TET_process_page()*
glyph	`tetml={glyphdetails={all}}`	`granularity=glyph`
word	–	`granularity=word`
wordplus	`tetml={glyphdetails={all}}`	`granularity=word`
word with Line elements	`tetml={elements={line}}`	`granularity=word`
wordplus with Line elements	`tetml={glyphdetails={all} elements={line}}`	`granularity=word`
line	–	`granularity=line`
page	–	`granularity=page`
image	`engines={notext novector}` `tetml={elements={annotations=false docinfo=true bookmarks=false destinations=false fields=false javascripts=false metadata=true options=true}}`	

Document options for controlling TETML output. In this section we will summarize the effect of various options which directly control the generated TETML output. All

other document options can be used to control processing details. The complete description of document options can be found in Table 10.8.

Document-related options must be supplied to the *--docopt* command-line option or to the *TET_open_document( )* function.

The *tetml* option[1] controls general aspects of TETML. The *elements* suboption can be used to suppress certain TETML elements if they are not required. The following document option list will suppress document-level XMP metadata in the generated TETML output:

```
tetml={ elements={nometadata} }
```

The *engines* option can be used to disable some of the TET kernel's processing engines. The following option list instructs TET to process text contents, but disable text color retrieval and image processing:

```
engines={notextcolor noimage}
```

The following document option makes sense only for *granularity=page*. It changes the default line separator character from linefeed to space:

```
lineseparator=U+0020
```

All document options which have been supplied when creating TETML are recorded in the */TET/Document/Options* element unless disabled with the following document option:

```
tetml={ elements={nooptions} }
```

Document options for controlling TETML output for interactive elements. TETML can also include information about interactive elements in the PDF document. The document option *tetml* with the suboption *elements* can be used to enable or disable TETML output for various aspects, e.g.

```
elements={annotations=true bookmarks=true destinations=true fields=true javascripts=true}
```

Page options for controlling TETML output. The complete description of page options can be found in Table 10.10. Page-related options must be supplied to the *--pageopt* command-line option or to *TET_process_page( )*.

The *tetml* page option enables or disables coordinate- and font-related information in the *Glyph* element. The following page option list enables font details in the *Glyph* element, but suppresses other glyph attributes:

```
tetml={ glyphdetails={font} }
```

The following page option list adds *Line* elements to the TETML output:

```
tetml={ glyphdetails={font} elements={line} }
```

The following page option adds *sub* and *sup* attributes to the *Glyph* element to designate subscripts and superscripts:

```
tetml={ glyphdetails={sub sup} }
```

1. Keep in mind that there are two different `tetml` *options: one on document level and one on page level.*

The following page option uses *all* to generate all possible attributes to the *Glyph* element:

```
tetml={ glyphdetails={all} }
```

The following page option requests topdown coordinates instead of the default bottom-up coordinates:

```
topdown={output}
```

The following page option list instructs TET to combine punctuation characters with the adjacent words, i.e. punctuation characters are no longer treated as individual words:

```
contentanalysis={nopunctuationbreaks}
```

All page options which have been supplied when creating TETML are recorded in the */TET/Document/Pages/Page/Options* elements (individually for each page) unless disabled with the following document option:

```
tetml={ elements={nooptions} }
```

Exception handling. If an error happens during PDF parsing TET generally tries to repair or ignore the problem if possible, or throws an exception otherwise. However, when generating TETML output with TET PDF parsing problems are usually reported as an *Exception* element in TETML:

```
<Exception errnum="4506">Object 'objects[49]/Subtype' does not exist</Exception>
```

Applications should be prepared to deal with *Exception* elements instead of the expected elements when processing TETML.

Problems which prevent the generation of the TETML output file (e.g. invalid options, no write permission for the output file) still trigger a runtime exception and no valid TETML output is created.

9.4 TETML Elements and the TETML Schema

A formal XML schema description (XSD) for all TETML elements and attributes as well as their relationships is contained in the TET distribution. The TETML namespace is the following:

```
http://www.pdflib.com/XML/TET5/TET-5.0
```

The schema can be downloaded from the following URL on the Web:

```
http://www.pdflib.com/XML/TET5/TET-5.0.xsd
```

Both TETML namespace and schema location are present in the root element of each TETML document.

Table 9.3 describes the role of all TETML elements. Elements and attributes which have been introduced after TET 5.0 are marked. Figure 9.1 and Figure 9.2 visualize the XML hierarchy of TETML elements.

Table 9.3 TETML elements and attributes

TETML element	*description and attributes*
A	*(Only for* `granularity=glyph` *and* `word`*) Anchor for an annotation, destination or field within the page content* *Attributes:* `id`, `type` *(the types* `start` *and* `stop` *enclose text, type* `rect` *abbreviates anchors without any content)*
Action	*Describes a PDF action.* *Attributes:* `filename`, `name`, `javascript`, `URI`, `trigger`, `type`
Annotation	*Describes a PDF annotation (excluding form fields which are described in* `Field` *elements). If the annotation has a corresponding popup annotation, the popup is expressed as a nested* `Annotation` *element.* *Attributes:* `alignment`, `anchor`, `color`, `creationdate`, `destination`, `hidden`, `icon`, `id`, `intent`, `interiorcolor`, `invisible`, `moddate`, `name`, `onscreen`, `opacity`, `open`, `print`, `readonly`, `rotate`, `subject`, `symbol`, `type` *Child elements:* `Action`, `Box`, `Annotation`, `Contents`, `Title`
Annotations	*Container of* `Annotation` *elements* *Attribute:* `xml:space`
Attachment	*For PDF attachments describes the contents in a nested* `Document` *element. For non-PDF attachments only the name is listed, but no contents.* *Attributes:* `name`, `level`, `pagenumber`
Attachments	*Container of* `Attachment` *elements*
BitPerSample	*Number of bits per sample for sampled functions, i.e.* `Function/@type="sampled"`
BlackPoint	*Tristimulus value of the black point for CalGray, CalRGB and Lab color spaces* *Attributes:* `x`, `y`, `z`
Body	*(TET 5.1) List body* *Child elements:* `Para`, `List`, `Table`, `PlacedImage`, `A`
Bookmark	*Contains* `Bookmark` *and* `Title` *elements to describe text, properties and nested bookmarks of a PDF bookmark (also called outline entry)* *Attributes:* `color`, `destination`, `fontstyle`, `open` *Child elements:* `Action`, `Bookmark`, `Title`

Table 9.3 TETML elements and attributes

TETML element	description and attributes
Bookmarks	*Container of* Bookmark *elements* *Attribute:* xml:space
Bounds	*Intervals for stitched functions, i.e.* Function/@type="stitching" *Child element:* Value
Box	*Describes the coordinates of a word, paragraph, annotation or form field. The attributes* llx *and* lly *describe the lower left corner,* urx *and* ury *describe the upper right corner of the Box*[1]*. A word or paragraph may contain multiple* Box *elements, e.g. a hyphenated word which spans several lines of text or a word which starts with a large dropcap character.* *Attributes:* llx, lly, urx, ury[2], ulx[2], uly[2], lrx, lry[2] *Child elements:* A, Glyph, Line, Para, PlacedImage, Table, Text, Word *Parent elements:* Para, Word
Co	*Initial color value for interpolation functions, i.e.* Function/@type="interpolate" *Child element:* Value
C1	*Terminal color value for interpolation functions, i.e.* Function/@type="interpolate"*. This element describes the alternate color of a spot color. As a convenience feature this element is also created for sampled functions, i.e.* Function/@type="sampled"*, although it is not present in PDF for such functions.* *Child element:* Value
Calculator	*Operators for PostScript functions, i.e.* Function/@type="PostScript"
Cell	*Describes the contents of a single table cell.* *Attributes:* colSpan, llx, lly[2], urx, ury[2], ulx[1], uly[2], lrx, lry[2], rowSpan *(TET 5.2)*
Color	*Describes a PDF color.* *Attributes:* colorspace, id, svgname, pattern
Colorant	*Colorant of a Separation or DeviceN color space* *Attributes:* name, colorspace
Colors	*Container of* Color *elements*
ColorSpace	*Describes a PDF color space.* *Attributes:* alternate, base, components, hival, iccprofile, id, name, pattern, subtype *Child elements:* BlackPoint, Colorant, Exception, Function, Gamma, Lookup, Matrix, Process, Range, WhitePoint
ColorSpaces	*Container of* ColorSpace *elements*
Content	*Describes the page contents as a hierarchical structure.* *Attributes:* granularity, dehyphenation, dropcap, font, geometry, shadow, sub, sup
Contents	*As child of* Annotation*: contents of an annotation* *As child of* Field*: contents of a form field*
Creation	*Describes the date and operating system platform for the TET execution, plus the version number of TET.* *Attributes:* date, platform, tetVersion
Decode	*Mapping of sample values for sampled functions, i.e.* Function/@type="sampled" *Child element:* Value
Destination	*Describes a PDF destination in the document.* *Attributes:* anchor, bottom[2], id, left, name, page, right, top[2], type, zoom
Destinations	*Container of* Destination *elements*

Table 9.3 TETML elements and attributes

TETML element	description and attributes
DefaultValue	*Default value of a form field*
DocInfo	*Predefined and custom document info entries* *Child elements:* Author, CreationDate, Creator, GTS_PDFXConformance, GTS_PDFXVersion, GTS_PPMLVDXConformance, GTS_PPMLVDXVersion, ISO_PDFEVersion, Keywords, ModDate, Producer, Subject, Title, Trapped, Custom *(attribute:* key), CustomBinary *(attribute:* key)
Document	*Describes general document information including PDF file name and size, PDF version number.* *Attributes:* filename, destination, pageCount, filesize, linearized, pdfVersion, pdfa *(TET 5.2: new values for PDF/A-4),* pdfe, pdfua *(TET 5.2: new value for PDF/UA-2),* pdfvcr *(TET 5.2),* pdfvt *(TET 5.2: new value for PDF/VT-3),* pdfx *(TET 5.2: new values for PDF/X-6),* revisions *(TET 5.0),* tagged, usagerights *(TET 5.0)* *Child elements:* Action, Attachments, Bookmarks, Destinations, DocInfo, Encryption, Exception, JavaScripts, Metadata, Options, OutputIntents, Pages, SignatureFields, XFA
Domain	*Input value interval(s) for functions* *Child element:* Value
Encode	*Mapping of input values for stitched functions, i.e.* Function/@type="stitching" *Child element:* Value
Encryption	*Describes various security settings.* *Attributes:* keylength, algorithm, attachment, description, masterpassword, userpassword, noprint, nomodify, nocopy, noannots, noassemble, noforms, noaccessible, nohiresprint, plainmetadata
Exception	*Error message and number associated with an exception which was thrown by TET and translated to TETML. The* Exception *element may replace other elements if not enough information can be extracted from the input because of malformed PDF data structures.* *The following elements may have an* Exception *element as child:* Annotation, Annotations, Attachment, Attachments, Bookmark, Bookmarks, Color, ColorSpace, ColorSpaces, Document, Field, Fields, Font, Fonts, ICCProfile, Image, Images, Metadata, Page, Pattern, Patterns, SignatureField, SignatureFields *Attribute:* errnum
Exponent	*Interpolation exponent for interpolated functions, i.e.* Function/@type="interpolate"
Field	*Describes a PDF form field.* *Attributes:* alignment, anchor, backgroundcolor, bordercolor, caption, captiondown, captionrollover, destination, export, exportvalue *(only for* type=radiobutton *and* checkbox), hidden, id, mappingname, name, onscreen, print, readonly, required, rotate, sort, state, type, visible *Child elements:* Action, Box, Contents, Field *(for the buttons comprising a field with* type=radiogroup), DefaultValue, OptionalValue, Tooltip, Value
Fields	*Container of* Field *elements* *Attribute:* xml:space
Font	*Describes a font resource. The required* name *attribute contains the canonical font name, while the optional* fullname *attribute contains the font name including subset prefix.* *Attributes:* ascender, capheight, descender, embedded, fullname, id, italicangle, type, name, vertical, weight, xheight
Fonts	*Container of* Font *elements*

Table 9.3 TETML elements and attributes

TETML element	description and attributes
Function	*Tint transform function for a* Separation *or* DeviceN *color space* *Attribute:* type *Child elements:* BitsPerSample, Bounds, Calculator, C0, C1, Decode, Domain, Encode, Functions, Exponent, Order, Range, Samples, Size
Functions	*Container of sub-functions for stitched functions, i.e.* Function/@type="stitching" *Child element:* Function
Gamma	*Gamma values for CalGray or CalRGB color space* *Child element:* Value
Glyph	*Describes font and geometry details for a single glyph. The element content holds the Unicode character(s) produced by the glyph. A single glyph may produce more than one character, e.g. for ligatures. The* Glyph *elements for a word are grouped within one or more* Box *elements.* *Attributes:* x, y[2], width, height *(only for vertical writing mode and if the glyph height is different from the font size),* alpha[2], beta[2], shadow, dropcap, fill, font, size, stroke, sub, sup, textrendering, unknown, dehyphenation, artifact *(TET 5.2)*
Graphics	*Container of the* Colors, ICCProfiles, *and* Layers *elements*
ICCProfiles	*Container of* ICCProfile *elements*
ICCProfile	*Describes an ICC color profile.* *Attributes:* checksum, iccversion, id, deviceclass, embedded, fromCIE, profilecs, profile-name, toCIE
Image	*Describes an image resource, i.e. the actual pixel array comprising the image.* *Attributes:* bitsPerComponent, colorspace, extractedAs, filename, height, id, maskid, mergetype, stencilmask, width *The attribute* Image/@filename *is identical to the image file names created by the TET command-line tool with the option* --imageloop resource *(see »Image file names«, page 21).*
Images	*Container of* Image *elements*
Item	*(TET 5.1) List item containing list label and body, plus optional* A *and* PlacedImage *elements* *Child elements:* Label, Body, A, PlacedImage
JavaScript	*Describes a sequence of JavaScript code* *Attributes:* id, name
JavaScripts	*Container of* JavaScript *elements*
Label	*(TET 5.1) List label* *Child elements:* Word
Layer	*Describes an optional content group (OCG), commonly called layer* *Attributes:* name, visible, label, locked *Child element:* Layer
Layers	*Container of* Layer *elements*
Line	*Text for a single line.* Line *may also contain* Word *elements.* *Attributes:* llx, lly[2], urx, ury[2], ulx[1], uly[2], lrx, lry[2] *(all introduced in TET 5.2)* *Child elements:* Text, Word
List	*(TET 5.1) Container of List items, plus optional* A *and* PlacedImage *elements. This element is only emitted if list detection is enabled.* *Attributes:* id *Child elements:* Item, A, PlacedImage

Table 9.3 TETML elements and attributes

TETML element	***description and attributes***
Lookup	*Lookup table for Indexed color spaces, i.e.* `ColorSpace/@name="Indexed"`. *It contains a hexadecimal sequence of values which must be interpreted in the Indexed color space's base color space.*
Matrix	*Transformation matrix of a CalRGB color space* *Child element:* `Value`
Metadata	*XMP metadata which can be associated with the document, a font, or an image*
OptionalValue	*Optional value of a form field*
Options	*Document or page options used for generating the TETML*
Order	*Order of sample interpolation for sampled functions, i.e.* `Function/@type="sampled"`
OutputIntent	*Describes the output intent of a document or page* *Attributes:* `iccprofile`, `subtype` *Child elements:* `OutputCondition`, `OutputConditionIdentifier`, `RegistryName`, `Info`
OutputIntents	*Container* `OutputIntent` *elements*
Page	*Contents of a single page.* *Attributes:* `hasdefaultcmyk`, `hasdefaultgray`, `hasdefaultrgb`, `height`, `label`, `number`, `topdown`, `width` *Child elements:* `Action`, `Annotations`, `Content`, `Exception`, `Fields`, `Options`, `OutputIntents`
Pages	*Container of* `Page` *elements*
Para	*Text comprising a single paragraph* *Child elements:* `A`, `Box`, `Word`
Pattern	*Describes a PDF pattern* *Attributes:* `id`, `patterntype`, `painttype`, `tilingtype`
Patterns	*Container of* `Pattern` *elements*
PlacedImage	*Describes an instance of an image placed on the page.* *Attributes:* `alpha`^{2}, `artifact` *(TET 5.2)*, `beta`^{2}, `height`, `image`, `source` *(TET 5.2)*, `width`, `x`, `y`^{2}
Process	*Process color space description of a* `DeviceN` *color space with subtype* `NChannel` *Attribute:* `colorspace` *Child element:* `Component`
Range	*As child of* `ColorSpace`: *Range of an Lab color space* *As child of* `Function`: *Range of output values for functions* *Child element:* `Value`
Resources	*Container of* `ColorSpaces`, `Fonts`, `Images`, *and* `Patterns` *resource containers*
Row	*Container of one or more table cells* *Child element:* `Cell`
Samples	*Hexadecimal sequence of samples for sampled functions, i.e.* `Function/@type="sampled"`
SignatureField	*Describes a signed or unsigned signature field* *Attributes:* `cades`, `field`, `fillablefields`, `permissions`, `preventchanges`, `sigtype`, `visible`
SignatureFields	*Container of* `SignatureField` *elements*
Size	*Number of samples in each input dimension for sampled functions, i.e.* `Function/@type="sampled"` *Child element:* `Value`

Table 9.3 TETML elements and attributes

TETML element	***description and attributes***
Table	*Container of one or more table rows* *Attributes:* `llx`, `lly`[2], `urx`, `ury`[2], `ulx`[1], `uly`[2], `lrx`, `lry`[2] *Child element:* `Row`
TET	*TETML root element. Attribute:* `version`
Text	*Text contents of a word or other element* *Attribute:* `artifact` *(TET 5.2)*
Title	*As child of* `Annotation`: *Title of an annotation* *As child of* `Bookmark`: *Title of a bookmark* *As child of* `DocInfo`: *document info entry* `Title`
Tooltip	*Tooltip of a form field*
Value	*Value of a form field*
WhitePoint	*Tristimulus value of the white point for CalGray, CalRGB and Lab color spaces* *Attributes:* `x`, `y`, `z`
Word	*Single word*
XFA	*The document contains XFA form information* *Attribute:* `type` *(always* `static` *since TET refuses to process dynamic XFA forms)*

1. If the Box represents a rectangle with edges parallel to the page edges, the four values `llx`, `lly`, `urx`, `ury` *describe the lower left and upper right corners; otherwise the coordinates of the upper left and lower right corners are additionally present as* `ulx`, `uly`, `lrx`, `lry`.
2. All vertical coordinates and angles are expressed relative to the lower left or upper left corner subject to the `topdown` *page option.*

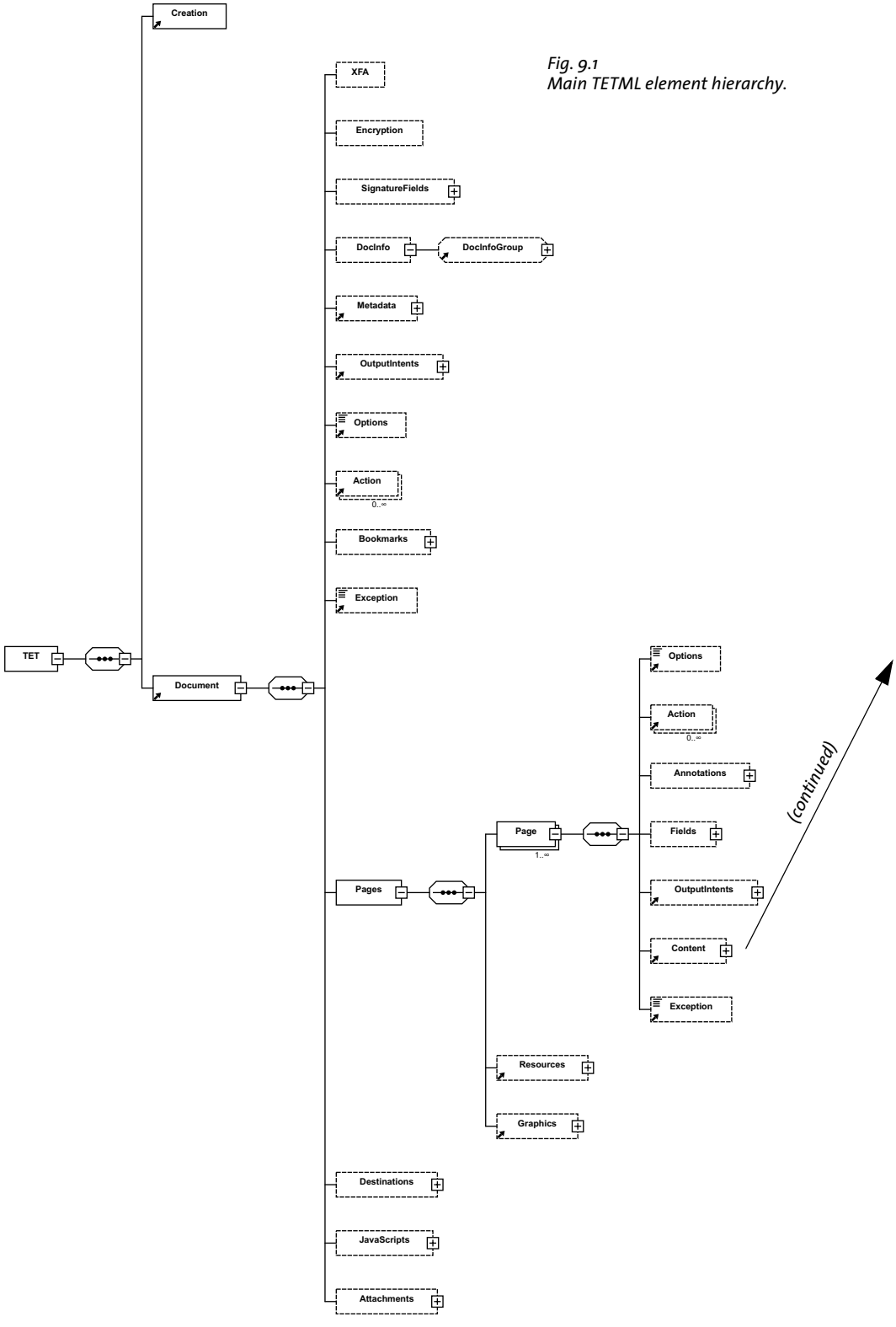
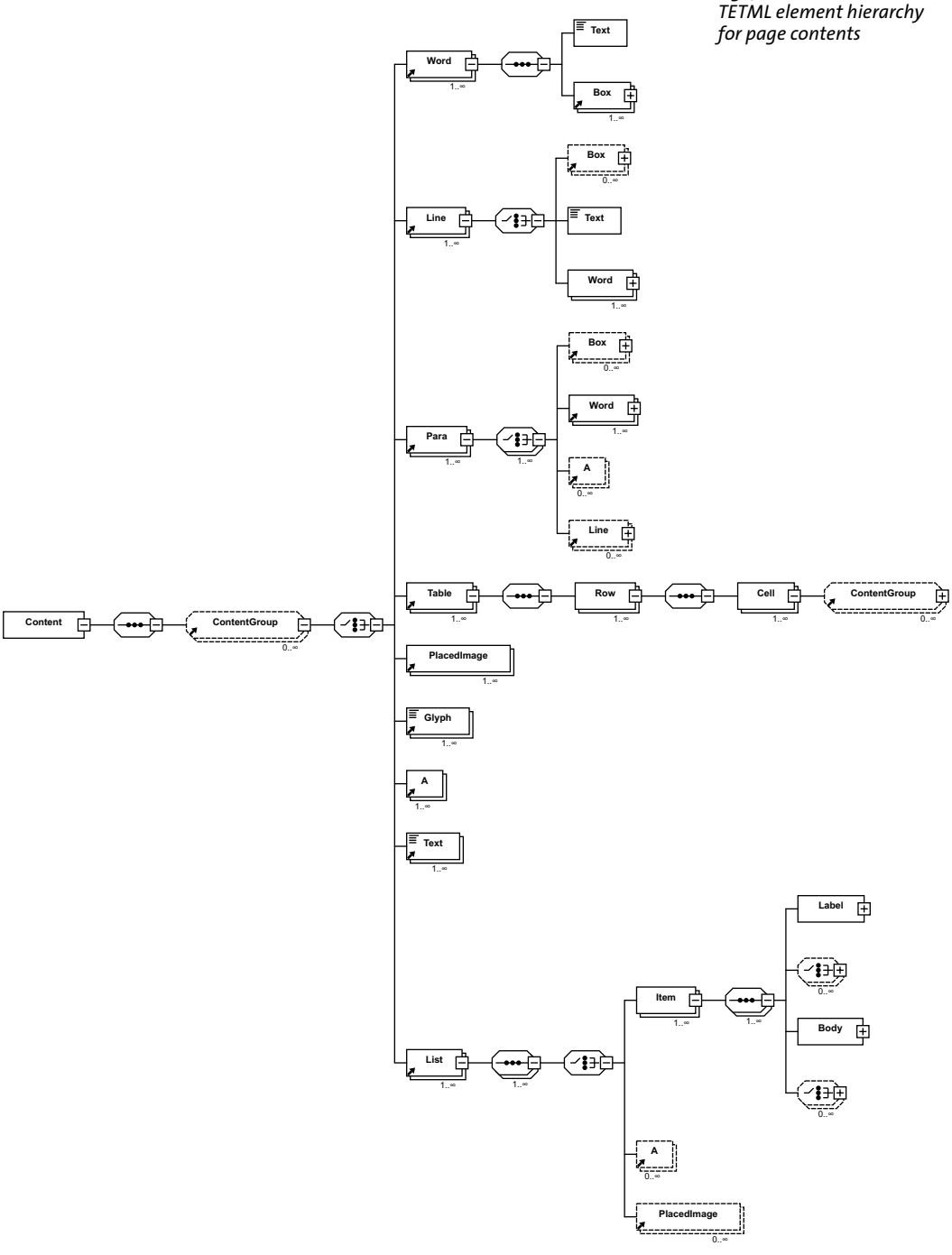


Fig. 9.1
Main TETML element hierarchy.

Fig. 9.2
TETML element hierarchy for page contents



9.5 Transforming TETML with XSLT

Very short overview of XSLT. XSLT (which stands for *eXtensible Stylesheet Language Transformations)* is a language for transforming XML documents to other documents. While the input is always an XML document (TETML in our case), the output does not necessarily have to be XML. XSLT can also perform arbitrary calculations and produce plain text or HTML output. We use XSLT stylesheets to process TETML input to generate a new dataset in text, XML, CSV, or HTML format based on the input which reflects the contents of a PDF document. The TETML document has been created with the TET command-line tool or the TET library as explained in Section 9.1, »Creating TETML«, page 133.

While XSLT is very powerful, it is considerably different from conventional programming languages. We do not attempt to provide an introduction to XSLT programming in this section; please refer to the available resources on this topic.

However, we do want to assist you in getting XSLT processing of TETML documents up and running quickly. This section describes the most important environments for running XSLT stylesheets and lists common software for this purpose. To apply XSLT stylesheets to XML documents you need an XSLT processor. There are various free and commercial XSLT processors available which can be used either in a stand-alone manner or in your own programs with the help of a programming language.

XSLT stylesheets can make use of parameters which are passed from the environment to the stylesheet in order to control processing details. Since some of our XSLT samples make use of stylesheet parameters we also supply information about passing parameters to stylesheets in various environments.

Common XSLT processors which can be used in various packagings include the following:

- Microsoft's XML implementation called MSXML. A free program called *msxsl.exe* is provided by Microsoft.
- Microsoft's .NET Framework XSLT implementation
- The .NET Core XSLT implementation
- Saxon *(www.saxonica.com)* is available in free and commercial versions. It is based on Java, but editions for .NET, C/C++ and PHP are also available.
- Xalan *(xalan.apache.org)*, an open-source project (available in C++ and Java implementations) hosted by the Apache foundation
- The open-source *libxslt* library of the GNOME project (*xmlsoft.org/XSLT)* and the corresponding *xsltproc* command-line tool are available in most Linux distributions.

XSLT on the command line. Applying XSLT stylesheets from the command-line provides a convenient development and testing environment. The examples below show how to apply XSLT stylesheets on the command-line. The samples below process the input file *TET-datasheet.tetml* with the stylesheet *tetml2html.xsl* while setting the XSLT parameter *toc-generate* (which is used in the stylesheet) to the value *0* and write the generated output to *TET-datasheet.html:*

- The Java-based Saxon processor can be used as follows:

```
java -jar saxon9.jar -o TET-datasheet.html TET-datasheet.tetml tetml2html.xsl
```

- You can apply XSLT scripts with the *ant* build tool. A minimal build file for applying XSLT looks as follows:

```
<project name="tetml2html" default="tetml2html">
```

```
   <target name="tetml2html">
      <xslt in="TET-datasheet.tetml" style="tetml2html.xsl" out="TET-datasheet.html"/>
   </target>
</project>
```

The *build.xml* file in the TET distribution contains XSLT tasks for all samples. The command *ant* applies all XSLT samples and converts the input document *TET-datasheet.pdf* to TETML. The following command processes another PDF input document:

```
ant -Dinput.pdf=myfile.pdf
```

- The *xsltproc* tool is included in most Linux distributions. Use the following command to apply a stylesheet to a TETML document:

```
xsltproc --output TET-datasheet.html --param toc-generate 0 tetml2html.xsl ↵
    TET-datasheet.tetml
```

The *runxslt.sh* shell script in the TET distribution can be used to run all XSLT samples with *xsltproc* (run *ant* once to create the TETML input files).

- Xalan C++ provides a command-line tool which can be invoked as follows:

```
Xalan -o TET-datasheet.html -p toc-generate 0 TET-datasheet.tetml tetml2html.xsl
```

On Windows systems with the MSXML parser you can use the *msxsl.exe* program as follows:

```
msxsl.exe TET-datasheet.tetml tetml2html.xsl -o TET-datasheet.html toc-generate=0
```

The *runxslt.ps1* and *runxslt.vbs* scripts in the TET distribution can be used to run all XSLT samples with *msxml* (run *ant* once to create the TETML input files).

XSLT within your own application. If you want to integrate XSLT processing in your application, the choice of XSLT processor obviously depends on your programming language and environment. The TET distribution contains sample code for various important environments. The *runxslt* samples demonstrate how to load a TETML document, apply an XSLT stylesheet with parameters, and write the generated output to a file. If the programs are executed without any arguments they exercise all XSLT samples supplied with the TET distribution. Alternatively, you can supply parameters for the TETML input file name, XSLT stylesheet name, output file name and additional parameter/value pairs. You can use the *runxslt* samples as a starting point for integrating XSLT processing into your application:

- Java developers can use the methods in the *javax.xml.transform* package. This is demonstrated in the *runxslt.java* sample.
- .NET Classic developers can use the methods in the *System.Xml.Xsl.XslTransform* namespace. This is demonstrated in the *runxslt.ps1* PowerShell script. Similar code can be used with C# and other .NET languages.
- .NET Core developers can use the *XslCompiledTransform* class. This is demonstrated in the *runxslt.cs* sample.

XSLT extensions are available for many other programming languages as well, e.g. Perl.

XSLT on the Web server. Since XML-to-HTML conversion is a common XSLT use case, XSLT stylesheets are often run on a Web server. Some important scenarios:

- Windows-based Web servers with ASP.NET can use the .NET interfaces mentioned above.
- Java-based Web servers can use the *javax.xml.transform* package.
- PHP-based Web servers can use the *XSLTProcessor* class.

9.6 XSLT Samples

The TET distribution includes several XSLT stylesheets which demonstrate the power of XSLT applied to TETML and can be used as a starting point for TETML applications. This section provides an overview of the XSLT samples and presents sample output. Section 9.5, »Transforming TETML with XSLT«, page 152 discusses options for deploying the XSLT stylesheets. More details regarding the functionality and inner workings of the stylesheets can be found in comments in the XSLT code. Some general aspects of the stylesheet samples:

- Most XSLT samples support parameters which can used to control various processing details. These parameters can be set within the XSLT code or overridden from the environment (e.g. *ant)*.
- Most XSLT samples require TETML input in a certain TETML mode (e.g. *word* mode, see »TETML modes«, page 139, for details). In order to protect themselves from wrong input, they check whether the supplied TETML input conforms to the requirement, and report an error otherwise.
- Some XSLT samples recursively process PDF attachments in the document (this is mentioned in the descriptions below). Most samples ignore PDF attachments, though. They are written in a way so that they can easily be expanded to process attachments as well. It is sufficient to select the relevant elements within the *Attachments* element; the relevant *xsl:template* elements themselves don't have to be modified.

Create a concordance. The *concordance.xsl* stylesheet expects TETML input in *word* or *wordplus* mode. It creates a concordance, i.e. a list of unique words in a document sorted by descending frequency. This may be useful to create a concordance for linguistic analysis, cross-references for translators, consistency checks, etc.

```
List of words in the document along with the number of occurrences:

the 138
and 91
TET 87
to 63
of 59
for 57
PDF 53
text 51
in 50
a 44
is 37
be 36
as 34
are 34
PDFlib 32
...
```

Font filtering. The *fontfilter.xsl* stylesheet expects TETML input in *glyph* or *wordplus* mode. It lists all words in a document which use a particular font in a size larger than a specified value. This may be useful to detect certain font/size combinations or for quality control. The same concept can be used to create a table of contents based on text portions which use a large font size.

```
Text containing font 'TheSansBold-Plain' with size greater than 10:

[ThesisAntiqua-Bold/32.0000] PDFlib
[ThesisAntiqua-Bold/32.0000] TET
[ThesisAntiqua-Bold/32.0000] 5
[ThesisAntiqua-Bold/14.0000] What
[ThesisAntiqua-Bold/14.0000] is
[ThesisAntiqua-Bold/14.0000] PDFlib
[ThesisAntiqua-Bold/14.0000] TET
[ThesisAntiqua-Bold/14.0000] ?
[ThesisAntiqua-Bold/14.0000] PDFlib
[ThesisAntiqua-Bold/14.0000] TET
[ThesisAntiqua-Bold/14.0000] Features
[ThesisAntiqua-Bold/14.0000] Challenges
[ThesisAntiqua-Bold/14.0000] with
[ThesisAntiqua-Bold/14.0000] PDF
[ThesisAntiqua-Bold/14.0000] Text
[ThesisAntiqua-Bold/14.0000] Extraction
[ThesisAntiqua-Bold/14.0000] Challenges
...
```

Searching for font usage. The *fontfinder.xsl* stylesheet expects TETML input in *glyph* or *wordplus* mode. For all fonts in a document, it lists all occurrences of text using this particular font along with page number and the position on the page. This may be useful for detecting unwanted fonts and checking consistency, locating use of a particular bad font size, etc.

```
TheSans-Plain used on:
page 1:
(306, 796)

ThesisAntiqua-Bold used on:
page 1:
(306, 757), (412, 757), (474, 757), (28, 514), (67, 514), (81, 514), (128, 514), (152,
514),
...
```

Font statistics. The *fontstat.xsl* stylesheet expects TETML input in *glyph* or *wordplus* mode. It generates font and glyph statistics. This may be useful for quality control and even accessibility testing since unmapped glyphs (i.e. glyphs which cannot be mapped to any Unicode character) will also be reported for each font.

```
17048 total glyphs in the document; breakdown by font:

85.21% TheSansLight-Plain: 14527 glyphs
5.19% TheSansLight-Italic: 885 glyphs
4.83% ThesisAntiqua-Bold: 823 glyphs, 3 uses of ligatures: fi
2.87% TheSansMonoCondensed-Plain: 489 glyphs
0.33% TheSansSemiLight-Caps: 57 glyphs
0.33% TheSansLight-Plain: 56 glyphs
0.25% TheSansLight-Italic: 42 glyphs
0.17% TheSansExtraLight-Italic: 29 glyphs
0.16% TheSansLight-Plain: 28 glyphs
0.16% TheSansLight-Plain: 28 glyphs
0.16% TheSansLight-Italic: 28 glyphs
0.16% TheSansLight-Plain: 28 glyphs
0.06% TheSansBold-Plain: 10 glyphs
```

```
0.05% TheSans-Plain: 9 glyphs
0.04% WarnockPro-It: 7 glyphs, 7 uses of ligatures: fi fl ffi Th sp ct st
0.01% PDFlibLogo2-Regular: 1 glyphs, 1 uses of ligatures: PDFlib
0.01% WarnockPro-Regular: 1 glyphs
```

Create an index. The *index.xsl* stylesheet expects TETML input in *word* or *wordplus* mode. It generates a back-of-the-book index, i.e. an alphabetically sorted list of words in the document and the corresponding page numbers. Numbers and punctuation characters are ignored.

```
Alphabetical list of words in the document along with their page number:

A
able 5
about 2
About 6
accent 3
Accented 3
accents 3
accept 5
Accepted 1
access 6
accessible 6
achieved 3
Acrobat 1 2 4 6
actual 2
actually 5
added 5
adding 6
addition 1 2 5
additional 2 4 5
Adobe 2 5 6
advanced 1
algorithm 3 4
...
```

Extract XMP metadata. The *metadata.xsl* stylesheet expects TETML input in any mode. It targets XMP metadata on the document level, and extracts some metadata properties from the XMP. PDF attachments (including PDF packages and portfolios) in the document are processed recursively:

```
dc:creator = PDFlib GmbH
xmp:CreatorTool = Adobe InDesign CS6 (Windows)
```

Extract table of contents in CSV format. The *table.xsl* stylesheet expects TETML input in *word, wordplus,* or *page* mode. It extracts the contents of a selected table and creates a CSV file (comma-separated values) which contains the table contents. CSV files can be opened with all spreadsheet applications. This may be useful to repurpose the contents of tables in PDF documents.

Convert TETML to HTML. The *tetml2html.xsl* stylesheet expects TETML input in *wordplus* mode. It converts TETML to HTML which can be displayed in a browser. The converter does not attempt to generate an identical visual representation of the PDF document, but demonstrates the following aspects:

- Create a linked table of contents at the beginning of the HTML page, where the entries are based on PDF bookmarks or headings in the document.
- Create heading elements *(H1, H2,* etc.) based on configurable font sizes and font names.
- Convert link annotations of type URI to HTML links.
- Map table elements in TETML to HTML table constructs to visualize tables in the browser.
- Map list elements in TETML to unordered or ordered HTML lists.
- Create a list of images for each page where the images are linked to the corresponding image file.
- Create links from PDF annotations.

Extract raw text from TETML. The *textonly.xsl* stylesheet expects TETML input in any mode. It extracts the raw text contents by fetching all *Text* elements while ignoring all other elements. PDF attachments (including PDF packages and portfolios) in the document are processed recursively.

10 TET Library API Reference

10.1 Option Lists and other general Topics

Option lists are a powerful yet easy method for controlling API method calls. Instead of requiring a multitude of method parameters, many API methods support option lists, or *optlists* for short. These are strings which can contain an arbitrary number of options. Option lists support various data types and composite data like lists. In most language bindings optlists can easily be constructed by concatenating the required keywords and values.

Bindings C language binding: you may want to use the *sprintf( )* function for constructing optlists.

Bindings .NET language binding: C# programmers should keep in mind that the *AppendFormat( )* StringBuilder method uses the *{* and *}* braces to represent format items which are replaced by the string representation of arguments. On the other hand, the *Append( )* method does not impose any special meaning on the brace characters. Since the option list syntax makes use of the brace characters, care must be taken in selecting the *AppendFormat( )* or *Append( )* method appropriately.

10.1.1 Option List Syntax

Formal option list syntax definition. Option lists must be constructed according to following rules:

- All elements (keys and values) in an option list must be separated by one or more of the following separator characters: space, tab, carriage return, newline, equal sign '='.
- An outermost pair of enclosing braces is not part of the element. The sequence *{ }* designates an empty element.
- Separators within the outermost pair of braces no longer split elements, but are part of the element. Therefore, an element which contains separators must be enclosed with braces.
- An element which contains braces at the beginning or end must be enclosed with braces.
- If an element contains unbalanced braces, these must be protected with a preceding backslash character. A backslash in front of the closing brace of an element must also be preceded by a backslash character.
- Option lists must not contain binary zero values.

An option may have a list value according to its documentation in this reference. List values contain one or more elements (which may themselves be lists). They are separated according to the rules above, with the only difference that the equal sign is no longer treated as a separator.

Simple option lists. In many cases option lists will contain one or more key/value pairs. Keys and values, as well as multiple key/value pairs must be separated by one or

more whitespace characters (space, tab, carriage return, newline). Alternatively, keys can be separated from values by an equal sign '=':

```
key=value
key = value
key value
key1 = value1  key2 = value2
```

To increase readability we recommend to use equal signs between key and value and whitespace between adjacent key/value pairs.

Since option lists are evaluated from left to right an option can be supplied multiply within the same list. In this case the last occurrence will overwrite earlier ones. In the following example the first option assignment is overridden by the second, and *key* will have the value *value2* after processing the option list:

```
key=value1 key=value2
```

List values. Lists contain one or more separated values, which may be simple values or list values in turn. Lists are bracketed with *{* and *}* braces, and the values in the list must be separated by whitespace characters. Examples:

```
searchpath={/usr/lib/tet d:\tet}              (list containing two directory names)
```

A list may also contain nested lists. In this case the lists must also be separated from each other by whitespace. While a separator must be inserted between adjacent *}* and *{* characters, it can be omitted between braces of the same kind:

```
fold={ {[:Private_Use:] remove} {[U+FFFD] remove} }    (list containing two lists)
```

If the list contains exactly one list the braces for the nested list must not be omitted:

```
fold={ {[:Private_Use:] remove} }                      (list containing one nested list)
```

Nested option lists and list values. Some options accept the type *option list* or *list of option lists*. Options of type *option list* contain one or more subordinate options. Options of type *list of option lists* contain one or more nested option lists. When dealing with nested option lists it is important to specify the proper number of enclosing braces. Several examples are listed below.

The value of the option *contentanalysis* is an option list which itself contains the single option *punctuationbreaks:*

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

The value of the option *glyphmapping* in the following example is a list of option lists containing a single option list:

```
glyphmapping={ {fontname=GlobeLogosOne codelist=GlobeLogosOne} }
```

The value of the option *glyphmapping* in the following example is a list of option lists containing two option lists:

```
glyphmapping { {fontname=CMSY* glyphlist=tarski} {fontname=ZEH* glyphlist=zeh}}
```

List containing one option list with a *fontname* value that includes spaces and therefore requires an additional pair of braces:

```
glyphmapping={ {fontname={Globe Logos One} codelist=GlobeLogosOne} }
```

List containing two keywords:

```
fonttype={Type1 TrueType}
```

List containing different types – the inner lists contain a Unicode set and a keyword, the outer list contains two option lists and the keyword *default:*

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

List containing one rectangle:

```
includebox={{10 20 30 40}}
```

Common traps and pitfalls. This paragraph lists some common errors regarding option list syntax.

Braces are not separators; the following is wrong:

```
key1 {value1}key2 {value2}                    WRONG!
```

This will trigger the error message *Unknown option 'value2'*. Similarly, the following are wrong since the separators are missing:

```
key{value}                                    WRONG!
key={{value1}{value2}}                        WRONG!
```

Braces must be balanced; the following is wrong:

```
key={open brace {}                            WRONG!
```

This will trigger the error message *Braces aren't balanced in option list 'key={open brace {}'*. A single brace as part of a string must be preceded by an additional backslash character:

```
key={closing brace \} and open brace \{}      CORRECT!
```

A backslash at the end of a string value must be preceded by another backslash if it is followed by a closing brace character:

```
key={\value\}                                 WRONG!
key={\value\\}                                CORRECT!
```

10.1.2 Basic Types

String. Strings are plain ASCII strings (or EBCDIC strings on EBCDIC platforms) which are generally used for non-localizable keywords. Strings containing whitespace or '=' characters must be bracketed with *{* and *}:*

```
password={ secret string }                    (string value contains three blanks)
contents={length=3mm}                         (string value containing one equal sign)
```

The characters *{* and *}* must be preceded by an additional `\` character if they are supposed to be part of the string:

```
password={weird\}string}                    (string value contains a right brace)
```

A backslash in front of the closing brace of an element must be preceded by a backslash character:

```
filename={C:\path\name\\}                   (string ends with a single backslash)
```

An empty string can be constructed with a pair or braces:

```
{}
```

Non-Unicode capable language bindings: if an option list starts with a [EBCDIC-]UTF-8 BOM, each content, hypertext or name string of the option list is interpreted as a [EBCDIC-]UTF-8 string.

Unquoted string values. In the following situations the actual characters in an option value may conflict with optlist syntax characters:

- Passwords or file names may contain unbalanced braces, backslashes and other special characters
- Japanese SJIS filenames in option lists (reasonable only in non-Unicode-capable language bindings)

In order to provide a simple mechanism for supplying arbitrary text or binary data which does not interfere with option list syntax elements, unquoted option values can be supplied along with a length specifier in the following syntax variants:

```
key[n]=value
key[n]={value}
```

The decimal number *n* represents the following:

- in Unicode-capable language bindings: the number of UTF-16 code units
- in non-Unicode aware language bindings: the number of bytes comprising the string

The braces around the string value are optional, but strongly recommended. They are required for strings starting with a space or other separator character. Braces, separators and backslashes within the string value are taken literally without any special interpretation.

Example for specifying a 7-character password containing space and brace characters. The whole string is surrounded by braces which are not part of the option value:

```
password[7]={ ab}c d}
```

Unichar. A Unichar is a single Unicode value where several syntax variants are supported: decimal values ≥ 10 (e.g. 173), hexadecimal values prefixed with *x, X, ox, oX*, or U+ *(xAD, oxAD, U+ooAD)*, numerical references, character references, and glyph name references but without the '&' and ';' decoration *(shy, #xAD, #173)*. Examples:

```
unknownchar=?                   (literal)
unknownchar=63                  (decimal)
unknownchar=x3F                 (hexadecimal)
unknownchar=0x3F                (hexadecimal)
unknownchar=U+003F              (Unicode notation)
lineseparator={CRLF}            (standard glyph name reference)
```

Single characters which happen to be a digit are treated literally, not as decimal Unicode values:

```
replacementchar=3                    (U+0033 THREE, not U+0003!)
```

Unichars must be in the hexadecimal range *0-0x10FFFF* (decimal *0-1114111)*.

Unicode sets. Unicode sets and can be constructed with the following building blocks:

- Patterns are a series of characters bounded by square brackets that contain lists of Unicode characters and Unicode property sets.
- Lists are a sequence of Unicode characters that may have ranges indicated by a '-' between two characters, as in *U+FB00-U+FB17*. The sequence specifies the range of all characters from the left to the right, in Unicode order. Multiple Unicode characters must not be separated by whitespace, but must directly follow each other, e.g. *U+0048U+006C*.
- Unicode characters in lists can be specified as follows:
  ASCII characters can be specified as literals
  Exactly 4 hex digits: *uhhhh* or *U+hhhh*
  Exactly 5 hex digits: *U+hhhhh*
  1-6 hex digits: *x{hhhhhh}*
  Exactly 8 hex digits: *Uhhhhhhhh*
  escaped backslash: \\
- Unicode property sets are specified by a Unicode property. The syntax for specifying the property names is an extension of POSIX and Perl syntax, where *type* represents the name of a Unicode property (see *www.unicode.org/Public/UNIDATA/PropertyAliases.txt)* and *value* the corresponding value (see *www.unicode.org/Public/UNIDATA/PropertyValueAliases.txt):*
  POSIX-style syntax: *[:type=value:]*
  POSIX-style syntax with negation: *[:^type=value:]*
  Perl-style syntax: *p{type=value}*
  Perl-style syntax with negation: *P{type=value}*
  The *type=* can be omitted for the *Category* and *Script* properties, but is required for other properties.
- Set operations can be applied to patterns:
  To build the union of two sets, simply concatenate them: *[[:letter:] [:number:]]*
  To intersect two sets, use the '&' operator: *[[:letter:] & [U+0061-U+007A]]*
  To take the set difference of two sets, use the '-' operator: *[[:letter:]-[U+0061-U+007A]]*
  To invert a set, place a '^' immediately after the opening '[':
  `[^U+0061-U+007A]`
  In any other location, the '^' does not have a special meaning.

See Table 10.1 for examples of Unicode sets. You can use the following Web site for interactively testing Unicode set expressions:

```
util.unicode.org/UnicodeJsps/list-unicodeset.jsp
```

Boolean. Booleans have the values *true* or *false*; if the value of a Boolean option is omitted, the value *true* is assumed. As a shorthand notation *noname* can be used instead of *name=false*:

Table 10.1 Unicode set examples

specification of Unicode set	*characters in the Unicode set*
[U+0061-U+007A]	*lower case letters* a *through* z
[U+0640]	*single character Arabic Tatweel*
[\x{0640}]	*single character Arabic Tatweel*
[U+FB00-U+FB17]	*Latin and Armenian ligatures*
[^U+0061-U+007A]	*all characters except* a *through* z
[:Lu:] [:UppercaseLetter:]	*all uppercase letters (short and long forms of the Unicode set)*
[:L:] [:Letter:]	*all Unicode categories starting with L (short and long forms of the Unicode set)*
[:General_Category=Dash_Punctuation:]	*all characters in the general category* Dash_Punctuation
[:Alphabetic=No:]	*all non-alphabetic characters*
[:Private_Use:]	*all characters in the Private Use Area (PUA)*

```
usehostfonts            (equivalent to usehostfonts=true)
nousehostfonts          (equivalent to usehostfonts=false)
```

Keyword. An option of type keyword can hold one of a predefined list of fixed keywords. Example:

```
clippingarea=cropbox
```

For some options the value hold either a number or a keyword.

Number. Option lists support several numerical types.

Integer types can hold decimal and hexadecimal integers. Positive integers starting with x, X, 0x, or 0X specify hexadecimal values:

```
-12345
0
0xFF
```

Floats can hold decimal floating point or integer numbers; period and comma can be used as decimal separators for floating point values. Exponential notation is also supported. The following values are all equivalent:

```
size = -123.45
size = -123,45
size = -1.2345E2
size = -1.2345e+2
```

10.1.3 Geometric Types

Rectangle. A rectangle is a list of four float values specifying the *x* and *y* coordinates of the lower left and upper right corners of a rectangle. The coordinate system for interpreting the coordinates (default or user coordinate system) varies depending on the option, and is documented separately. Example:

```
includebox = {{0 0 500 100} {0 500 500 600}}
```

10.1.4 Unicode Support in Language Bindings

If a programming language or environment supports Unicode strings natively we call the binding Unicode-capable. The following language bindings are Unicode-capable:

- C++
- .NET and .NET Core
- Java
- Objective-C
- Python
- RPG

String handling in these environments is straightforward: all strings are supplied as Unicode strings in native UTF-16 format. The language wrappers correctly deal with Unicode strings provided by the client and automatically set certain options.

The following language bindings are not Unicode-capable by default:

- C
- Perl
- PHP
- Ruby

The use of UTF-8 is recommended for non-Unicode-capable language bindings. Some aspects of the API differ between Unicode-capable and non-Unicode-capable language bindings. Such differences are mentioned in the corresponding API descriptions in this chapter.

The *TET_convert_to_unicode( )* method can be used to convert between UTF-8, UTF-16 and UTF-32 strings or from arbitrary encodings to Unicode with an optional BOM.

10.1.5 Encoding Names

Various options and parameters support the names of encodings, e.g. the *filename-handling* option of *TET_set_option( )*, the *forceencoding* option of *TET_open_document( )*, and the *inputformat* parameter of *TET_convert_to_unicode( )*. The following keywords can be supplied as encoding names:

- The keyword *auto* specifies the most natural encoding for certain environments:
  - On Windows: the current system code page
  - On Unix, Windows and macOS: *iso8859-1*
  - On IBM System i: the current job's encoding *(IBMCCSIDoooooooooooo)*
  - On IBM Z: *ebcdic*
- *winansi (=cp1252)*
- *ebcdic* (=code page 1047), *ebcdic_37 (=code page 037)*
- on the following systems all encodings available on the host system can be used:
  - Windows: *cpXXXX*
  - Linux: all codesets known to the *iconv* facility
  - IBM System i: any *Coded Character Set Identifier* without the *CCSID* prefix
  - IBM Z: any *Coded Character Set Identifier* (CCSID)
- custom encodings can be defined as resources and referenced by their resource name

The TET package contains a collection of code page files under the path */resource/codepage* that can be loaded as encoding resources.

10.1.6 Multi-threaded Programming

While TET itself is single-threaded, it can safely be used in multi-threaded applications. In the common situation that a TET object is only used within one thread, no particular multi-threading precautions are necessary. If the same object is used in multiple threads the application must synchronize the threads to make sure that the TET object is not accessed simultaneously by more than one thread. A typical scenario would involve a pool of TET objects where each thread fetches an existing PLOP object from the pool instead of creating a new one, and returns it to the pool after creating a document if the object is no longer needed. Using the same TET object in another thread before the output document is finished will rarely provide any advantage and is not recommended.

10.2 General Methods

API methods in this section:

- *TET_set_option( )*
- *TET_new)*
- *TET_delete( )*
- *TET_create_pvf( )*
- *TET_delete_pvf( )*
- *TET_info_pvf( )*
- *TET_convert_to_unicode( )*
- *TET_get_apiname)*
- *TET_get_errmsg( )*
- *TET_get_errnum( )*

10.2.1 Option Handling

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

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

C ***void TET_set_option(TET *tet, const char *optlist)***

Set one or more global options for TET.

optlist Option list specifying global options according to Table 10.2. The following options can be used:
asciifile, cmap, codelist, encoding, filenamehandling, fontoutline, glyphlist, hostfont, license, licensefile, logging, mmiolimit, userlog, outputformat, resourcefile, searchpath, shutdownstrategy, timeout

Details Multiple calls to this method can be used to accumulate values for those options marked in Table 10.2. For unmarked options the new value overrides the old one. In order to supply multiple values for an option (e.g. *searchpath)* in a single call supply all values in a list argument to this option.

Table 10.2 Global options for TET_set_option( )

option	***description***
asciifile	*(Boolean; Only supported on IBM System i and IBM Z). Expect text files (e.g. UPR configuration files, glyph lists, code lists) in ASCII encoding. Default:* `true` *on IBM System i;* `false` *on IBM Z*
cmap[1, 2]	*(List of name strings) A list of string pairs, where each pair contains the name and value of a CMap resource (see Section 5.2, »Resource Configuration and File Searching«, page 61).*
codelist[1, 2]	*(List of name strings) A list of string pairs, where each pair contains the name and value of a codelist resource (see Section 5.2, »Resource Configuration and File Searching«, page 61).*
encoding[1, 2]	*(List of name strings) A list of string pairs, where each pair contains the name and value of an encoding resource (see Section 5.2, »Resource Configuration and File Searching«, page 61).*

Table 10.2 Global options for TET_set_option( )

option	description
filename-handling	*(Keyword) Indicates the encoding of file names. File names supplied as method parameters without UTF-8 BOM in non-Unicode aware language bindings are interpreted according to this option to guard against characters which would be illegal in the file system and to create a Unicode version of the file name. An error occurs if the file name contains characters outside the specified encoding. Default:* `unicode` *on Windows and macOS,* `auto` *on IBM System i, otherwise* `honorlang`**:** **ascii** *7-bit ASCII* **basicebcdic** *Basic EBCDIC according to code page 1047, but only Unicode values <= U+007E* **basicebcdic_37** *Basic EBCDIC according to code page 0037, but only Unicode values <= U+007E* **honorlang** *(Not supported on IBM System i) The environment variables LC_ALL, LC_CTYPE and LANG are interpreted. The codeset specified in* `LANG` *is applied to file names if it is available.* **unicode** *Unicode encoding in (EBCDIC-)UTF-8 format* **all names of 8-bit and CJK encodings** *Encoding name according to Section 10.1.5, »Encoding Names«, page 165*
fontoutline[1, 2]	*(List of name strings) A list of string pairs, where each pair contains the name and value of a FontOutline resource (see Section 5.2, »Resource Configuration and File Searching«, page 61).*
glyphlist[1, 2]	*(List of name strings) A list of string pairs, where each pair contains the name and value of a glyphlist resource (see Section 5.2, »Resource Configuration and File Searching«, page 61).*
hostfont[1, 2]	*(List of name strings) A list of string pairs, where each pair contains a PDF font name and the UTF-8 encoded name of a host font to be used for an unembedded font.*
license	*(String) Set the license key. It must be set before the first call to TET_open_document*( ).*
licensefile	*(String) Set the name of a file containing the license key(s). The license file can be set only once before the first call to TET_open_document*( ). Alternatively, the name of the license file can be supplied in an environment variable called PDFLIBLICENSEFILE or (on Windows) via the registry.*
logging[1]	*(Option list; unsupported) Option list specifying logging output according to Table 10.7. Alternatively, logging options can be supplied in an environment variable called TETLOGGING. An empty option list enables logging with the options set in previous calls. If the environment variable is set, logging starts immediately after the first call to TET_new( ).*
mmiolimit	*(Integer) Upper limit for the size of input files in MB (=1024*1024 bytes) which are memory-mapped. Setting this option to 0 (zero) disables memory mapping. Disabling memory mapping can be used on non-Windows systems to avoid problems when remote files suddenly become unavailable while being used. Default: 50 on 32-bit platforms and IBM System i, 2048 otherwise*
userlog	*(Name string; unsupported) Arbitrary string which is written to the log file if logging is enabled.*
output-format	*(Keyword; only for the C, Ruby, RPG, Perl, Python, and PHP language bindings) Specifies the format of the text returned by TET_get_text( ):* **utf8** *Strings are returned in (in C: null-terminated) UTF-8 format .* **utf16** *Strings are returned in UTF-16 format in the machine's native byte ordering.* **utf32** *Strings are returned in UTF-32 format in the machine's native byte ordering.* **ebcdicutf8** *(Only available on EBCDIC-based systems) Strings are returned in null-terminated EBCDIC-encoded UTF-8 format. Code page 37 is used on IBM System i, code page 1047 on IBM Z.* *Default:* `utf8` *for C, Ruby, RPG, Perl, Python, PHP, and* `ebcdicutf8` *for C on IBM System i and IBM Z*

Table 10.2 Global options for TET_set_option( )

option	description
resourcefile	*(Name string) Relative or absolute file name of the UPR resource file. The resource file is loaded immediately. Existing resources are kept; their values are overridden by new ones if they are set again. Explicit resource options are evaluated after entries in the resource file.* *The resource file name can also be supplied in the environment variable TETRESOURCEFILE or with a Windows registry key (see Section 5.2, »Resource Configuration and File Searching«, page 61). Default: tet.upr (on MVS: upr)*
searchpath[1]	*(List of name strings) Relative or absolute path name(s) of a directory containing files to be read. The search path can be set multiply; the entries are accumulated and used in least-recently-set order (see Section 5.2, »Resource Configuration and File Searching«, page 61). It is recommended to use double braces even for a single entry to avoid problems with directory names containing space characters. An empty string list (i.e. {{}}) deletes all existing search path entries including the default entries. On Windows the search path can also be set via a registry entry. Default: platform-specific, see »File search and the searchpath resource category«, page 62.*
shutdown-strategy	*(Integer) Strategy for releasing global resources which are allocated once for all TET objects. Each global resource is initialized on demand when it is first needed. This option must be set to the same value for all TET objects in a process; otherwise the behavior is undefined (default: 0):* **0** *A reference counter keeps track of how many TET objects use the resource. When the last TET object is deleted and the reference counter drops to zero, the resource is released.* **1** *The resource is kept until the end of the process. This may slightly improve performance, but requires more memory after the last TET object is deleted.*
timeout	*(Integer) Reset the timer and set the maximum processing time in seconds (more precisely: a limit for the accumulated processing time spent in all TET API methods); zero means no limit. This can be used to end processing after the specified time. If the timeout is reached TET throws an exception. The timeout is only changed when this option is supplied (but not automatically for each document). Default: 0*

1. Option values can be accumulated with multiple calls.
2. Unlike the UPR syntax an equal character '=' between the name and value is neither required nor allowed.

10.2.2 Setup

C ***TET *TET_new(void)***

Create a new TET object.

Returns A handle to a TET object to be used in subsequent calls. If this method doesn't succeed due to unavailable memory it returns NULL.

Bindings This method is not available in object-oriented language bindings since it is hidden in the TET constructor.

Java ***void delete( )***

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

C ***void TET_delete(TET *tet)***

Delete a TET object and release all related internal resources.

Details Deleting a TET object automatically closes all of its open documents. The TET object must no longer be used in any API method after it has been deleted.

Bindings In object-oriented language bindings this method is generally not required since it is hidden in the TET destructor. However, in Java it is available nevertheless to allow explicit cleanup in addition to automatic garbage collection. In .NET *Dispose( )* should be called at the end of processing to clean up unmanaged resources.

10.2.3 PDFlib Virtual Filesystem (PVF)

C++ ***void create_pvf(String filename, const void *data, size_t size, String optlist)***

C# Java ***void create_pvf(String filename, byte[] data, String optlist)***

Perl PHP ***create_pvf(string filename, string data, string optlist)***

C ***void TET_create_pvf(TET *tet,***
const char *filename, int len, const void *data, size_t size, const char *optlist)

Create a named virtual read-only file from data provided in memory.

filename (Name string) The name of the virtual file. This is an arbitrary string which can later be used to refer to the virtual file in other TET calls.

len (C and RPG language bindings only) Length of *filename* (in bytes) for UTF-16 strings. If *len=0* a null-terminated string must be provided.

data A reference to the data for the virtual file. In C and C++ it is a pointer to a memory location. In Java it is a byte array. In Perl and PHP it is a string.

size (C and C++ only) The length in bytes of the memory block containing the data.

optlist Option list according to Table 10.3. The following option can be used: *copy*

Details The virtual file name can be supplied to any API method which uses input files. Some of these methods may set a lock on the virtual file until the data is no longer needed. Virtual files are kept in memory until they are deleted explicitly with *TET_delete_pvf( )*, or automatically in *TET_delete( )*.

Each TET object will maintain its own set of PVF files. Virtual files cannot be shared among different TET objects. Multiple threads working with separate TET objects do not need to synchronize PVF use. If *filename* refers to an existing virtual file an exception is thrown. This method does not check whether *filename* is already in use for a regular disk file.

Unless the *copy* option has been supplied, the caller must not modify or free (delete) the supplied data before a corresponding successful call to *TET_delete_pvf( )*. Not obeying to this rule will most likely result in a crash.

Table 10.3 Options for TET_create_pvf( )

option	description
copy	*(Boolean) If* `true`*, PDFlib, creates an internal copy of the supplied data. In this case the caller may dispose of the supplied data immediately after this call. Default:* `false` *for C and C++, but* `true` *for all other language bindings*

C++ Java C# ***int delete_pvf(String filename)***

Perl PHP ***int delete_pvf(string filename)***

C ***int TET_delete_pvf(TET *tet, const char *filename, int len)***

Delete a named virtual file and free its data structures.

filename (Name string) The name of the virtual file as supplied to *TET_create_pvf( )*.

len (C language binding only) Length of *filename* (in bytes) for UTF-16 strings. If *len=0* a null-terminated string must be provided.

Returns -1 if the corresponding virtual file exists but is locked, and 1 otherwise.

Details If the file isn't locked, TET will immediately delete the data structures associated with *filename*. If *filename* does not refer to a valid virtual file this method will silently do nothing. After successfully calling this method *filename* may be reused. All virtual files will automatically be deleted in *TET_delete( )*.

The detailed semantics depend on whether or not the *copy* option has been supplied to the corresponding call to *TET_create_pvf( ):* If the *copy* option has been supplied, both the administrative data structures for the file and the actual file contents (data) are freed; otherwise, the contents will not be freed, since the client is supposed to do so.

C++ Java C# ***int info_pvf(String filename, String keyword)***

Perl PHP ***int info_pvf(string filename, string keyword)***

C ***int TET_info_pvf(TET *tet, const char *filename, int len, const char *keyword)***

Query properties of a virtual file or the PDFlib Virtual File system (PVF).

filename (Name string) The name of the virtual file. The filename may be empty if *keyword=filecount*.

len (C language binding only) Length of *filename* (in bytes) for UTF-16 strings. If *len=0* a null-terminated string must be provided.

keyword Keyword according to Table 10.4.

Returns The value of some file parameter as requested by *keyword*.

Details This method returns various properties of a virtual file or the PDFlib Virtual File system (PVF). The property is selected by *keyword*.

Table 10.4 Keywords for TET_info_pvf( )

option	description
filecount	*Total number of files in the PDFlib Virtual File system maintained for the current TET object. The* `filename` *parameter is ignored.*
exists	*1 if the file exists in the PDFlib Virtual File system (and has not been deleted), otherwise 0*
size	*(Only for existing virtual files) Size of the specified virtual file in bytes.*
iscopy	*(Only for existing virtual files) 1 if the* `copy` *option was supplied when the specified virtual file was created, otherwise 0*
lockcount	*(Only for existing virtual files) Number of locks for the specified virtual file set internally by TET API methods. The file can only be deleted if the lock count is 0.*

10.2.4 Unicode Conversion

C++ ***string convert_to_unicode(String inputformat, string input, String optlist)***
C# Java ***String convert_to_unicode(String inputformat, byte[ ] input, String optlist)***
Perl PHP ***string convert_to_unicode(string inputformat, string input, string optlist)***
C ***const char *TET_convert_to_unicode(TET *tet,***
const char *inputformat, const char *input, int inputlen, int *outputlen, const char *optlist)

Convert a string in an arbitrary encoding to a Unicode string in various formats.

inputformat Unicode text format or encoding name specifying interpretation of the input string:
- Unicode text formats: *utf8, ebcdicutf8 (on EBCDIC platforms), utf16, utf16le, utf16be, utf32*
- An encoding name according to Section 10.1.5, »Encoding Names«, page 165
- The keyword *auto* specifies the following behavior: if the input string contains a UTF-8 or UTF-16 BOM it is used to determine the appropriate format, otherwise the current system codepage is assumed.

input String to be converted to Unicode.

inputlen (C and RPG language bindings only) Length of the input string in bytes. If *inputlen=0* a null-terminated string must be provided.

outputlen (C and RPG language bindings only) C-style pointer to a memory location where the length of the returned string (in bytes) is stored.

optlist Option list specifying options according to Table 10.5:
- Input filter options: *charref, escapesequence*
- Unicode conversion options: *bom, errorpolicy, inflate, outputformat*

Returns A Unicode string created from the input string according to the specified parameters and options. If the input string does not conform to the specified input format (e.g. invalid UTF-8 string) an empty output string is returned if *errorpolicy=return*, and an exception is thrown if *errorpolicy=exception*.

Details This method may be useful for general Unicode string conversion. It is provided for the benefit of users working in environments which do not provide suitable Unicode converters.

Bindings C binding: the returned strings are stored in a ring buffer with up to 10 entries. If more than 10 strings are converted, the buffers are reused, which means that clients must copy the strings if they want to access more than 10 strings in parallel. For example, up to 10 calls to this method can be used as parameters for a *printf( )* statement since the return strings are guaranteed to be independent if no more than 10 strings are used at the same time.

C++ binding: The parameters *inputformat* and *optlist* must have the selected string data type (see Section 3.2, »C++ Binding«, page 28), while *input* and returned data always have type *string*.

Python binding: UTF-8 results are returned as a string, Python 3: non-UTF-8 results are returned as bytes.

Table 10.5 Options for TET_convert_to_unicode( )

option	description
charref	*(Boolean) If* `true`*, enable substitution of numeric and character entity references and glyph name references. Default:* `false`
bom	*(Keyword; ignored for* `outputformat=utf32`*; in .NET, Java, Objective-C and Python only* `none` *is allowed) Policy for adding a byte order mark (BOM) to the output string. Supported keywords (default:* `none`*):* **add** *Add a BOM.* **keep** *Add a BOM if the input string has a BOM.* **none** *Don't add a BOM.* **optimize** *Add a BOM except if* `outputformat=utf8` *or* `ebcdicutf8` *and the output string contains only characters in the range < U+007F.*
errorpolicy	*(Keyword) Behavior in case of conversion errors (default:* `exception`*):* **return** *The replacement character U+FFFD is used if a character reference cannot be resolved. An empty string is returned in case of conversion errors.* **exception** *An exception is thrown in case of conversion errors.*
escapesequence	*(Boolean) If* `true`*, enable substitution of escape sequences in strings. Default:* `false`
inflate	*(Boolean; only for* `inputformat=utf8`*; ignored if* `outputformat=utf8`*) If* `true`*, an invalid UTF-8 input string will not trigger an exception, but rather an inflated byte string in the specified output format is generated. This may be useful for debugging. Default:* `false`
outputformat	*(Keyword) Unicode text format of the generated string:* `utf8`*,* `ebcdicutf8` *(on EBCDIC platforms),* `utf16`*,* `utf16le`*,* `utf16be`*,* `utf32`*. An empty string is equivalent to* `utf16` *(default:* `utf16`*)* *Unicode-aware language bindings: the output format is forced to* `utf16`*.* *C++ and RPG language bindings: only the following output formats are allowed:* `ebcdicutf8`*,* `utf8`*,* `utf16`*,* `utf32`*.*

10.2.5 Exception Handling

Exceptions vs. error return values. Errors of a certain kind are called exceptions ifor good reasons – they are mere exceptions, and are not expected to occur very often. The general strategy is to use conventional mechanisms with error return codes) for method calls which may fail often, and use an exception mechanism for those rare occasions which don't justify cluttering the code with conditionals. This is exactly the path that TET goes: some operations can be expected to go wrong rather frequently, for example:

- Trying to open a PDF document for which one doesn't have the proper password (but see also the *shrug* feature described in Section 5.1, »Extracting Content from protected PDF«, page 59);
- Trying to open a PDF document with a wrong file name;
- Trying to open a PDF document which is damaged beyond repair.

TET signals such errors by returning a special value (usually –1, but 0 in the PHP binding) documented in the API reference. Other events are considered harmful, but occur rather infrequently, e.g.

- running out of virtual memory;
- supplying wrong method parameters (e.g. an invalid document handle);
- supplying malformed option lists;
- a required resource (e.g. a CMap file for CJK text extract) cannot be found.

When TET detects such a situation, an exception is thrown instead of passing a special error return value to the caller.

It is important that processing a document must be stopped when an exception occurred. The only methods which can safely be called after an exception are *delete( )*, *get_apiname( )*, *get_errnum( )*, and *get_errmsg( )*. Calling any other method after an exception may lead to unexpected results. An exception delivers the following information:

- A unique error number;
- The name of the API method which caused the exception;
- A descriptive text containing details of the problem;

Querying the reason of a failed method call. Some TET method calls, e.g. *open_document( )* or *open_page( )*, can fail without throwing an exception (they return an error code). In this situation the methods *get_errnum( )*, *get_errmsg( )*, and *get_apiname( )* can be called immediately after a failed method call to retrieve details about the problem.

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

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

C ***const char *TET_get_apiname(TET *tet)***

Get the name of the API method which caused an exception or failed.

Returns The name of the method which threw an exception, or the name of the most recently called method which failed with an error code. An empty string is returned if there was no error.

C++ Java C# ***String get_errmsg( )***

Perl PHP ***string get_errmsg( )***

C ***const char *TET_get_errmsg(TET *tet)***

Get the text of the last thrown exception or the reason for a failed method call.

Returns Text containing the description of the last exception thrown, or the reason why the most recently called method failed with an error code. An empty string is returned if there was no error.

C++ Java C# ***int get_errnum( )***

Perl PHP ***long get_errnum( )***

C ***int TET_get_errnum(TET *tet)***

Get the number of the last thrown exception or the reason for a failed method call.

Returns The number of an exception, or the error code of the most recently called method which failed with an error code. This method returns 0 if there was no error.

C ***TET_TRY(tet)***

C ***TET_CATCH(tet)***

C ***TET_RETHROW(tet)***

C ***TET_EXIT_TRY(tet)***

Set up an exception handling block; catch or rethrow an exception; or inform the exception machinery that a *TET_TRY( )* block is left without entering the corresponding *TET_CATCH( )* block. *TET_RETHROW( )* can be used to throw an exception again to a higher-level method after catching it.

Details (C language binding only) See Section 3.1, »C Binding«, page 25.

10.2.6 Logging

The logging feature can be used to trace API calls. The contents of the log file may be useful for debugging purposes or may be requested by PDFlib GmbH support. Table 10.6 lists the options for activating the logging feature with *TET_set_option( )* (see Section 10.2.1, »Option Handling«, page 167).

Table 10.6 Logging-related keys for TET_set_option( )

key	explanation
logging	*Option list with logging options according to Table 10.7*
userlog	*String which is copied to the log file*

The logging options can be supplied in the following ways:

- As an option list for the *logging* option of *TET_set_option( )*, e.g.:

```
tet.set_option("logging={filename={debug.log} remove}");
```

- In an environment variable called *TETLOGGING*. Doing so will activate the logging output starting with the very first call to one of the API methods.

Table 10.7 Suboptions for the `logging` *option of TET_set_option( )*

key	explanation
(empty list)	*Enable log output after it has been disabled with* `disable`.
disable	*(Boolean) Disable logging output. Default:* `false`
enable	*(Boolean) Enable logging output*
filename	*(String) Name of the log file (*`stdout` *and* `stderr` *are also acceptable). Output is appended to any existing contents. The log file name can alternatively be supplied in an environment variable called TETLOGFILENAME (in this case the option* `filename` *will always be ignored). Default:* `tet.log` *(on Windows and macOS in the / directory, on Unix in* `/tmp`*)*
flush	*(Boolean) If* `true`*, the log file will be closed after each output, and reopened for the next output to make sure that the output will actually be flushed. This may be useful when chasing program crashes where the log file is truncated, but significantly slows down processing. If false, the log file is opened only once. Default:* `false`
includepid	*(Boolean; not on MVS) Include the process id in the log file name. This should be enabled if multiple processes use the same log file name. Default:* `false`
includetid	*(Boolean; not on MVS) Include the thread id in the log file name. This should be enabled if multiple threads in the same process use the same log file name. Default:* `false`
includeoid	*(Boolean; not on MVS) Include the object id in the log file name. This should be enabled if multiple TET objects in the same thread use the same log file name. Default:* `false`
remove	*(Boolean) If* `true`*, an existing log file is deleted before writing new output. Default:* `false`
removeonsuccess	*(Boolean) Remove the generated log file in TET_delete( ) unless an exception occurred. This may be useful for analyzing occasional problems in multi-threaded applications or problems which occur only sporadically. It is recommended to combine this option with* `includepid`/`includetid`/`includeoid` *as appropriate.*
stringlimit	*(Integer) Limit for the number of characters in text strings, or 0 for unlimited. Default: 0*

Table 10.7 Suboptions for the `logging` *option of TET_set_option( )*

key	***explanation***
classes	*(Option list) List containing options of type integer, where each option describes a logging class and the corresponding value describes the logging level. Level 0 disables a logging class, positive numbers enable a class. Increasing levels provide more and more detailed output. The following options are supported (default:* `{api=1 warning=1}`*):*
	api *Log all API calls with their method parameters and results. If* `api=2` *a timestamp is created in front of all API trace lines, and deprecated methods and options are marked.*
	convert *String conversion.*
	filesearch *Log all attempts related to locating files via* `SearchPath` *or PVF.*
	resource *Log all attempts at locating resources via Windows registry, UPR definitions as well as the results of the resource search.*
	user *User-specified logging output supplied with the* `userlog` *option.*
	warning *Log all warnings, i.e. error conditions which can be ignored or fixed internally. If* `warning=2` *messages from methods which do not throw any exception, but hook up the message text for retrieval via TET_get_errmsg( ), and the reason for all failed attempts at opening a file (searching for a file in searchpath) is also logged.*
	.

10.3 Document Methods

API methods in this section:

- *TET_open_document( )*
- *TET_open_document_callback( )*
- *TET_close_document( )*

C++ Java C# ***int open_document(String filename, String optlist)***

Perl PHP ***long open_document(string filename, string optlist)***

C ***int TET_open_document(TET *tet, const char *filename, int len, const char *optlist)***

Open a disk-based or virtual PDF document for content extraction.

filename Full path name of the PDF file to be opened. The file is searched by means of the *SearchPath* resource.

In non-Unicode language bindings the file name is converted to UTF-8 according to the *filenamehandling* option (unless *filenamehandling=unicode* or the supplied file name starts with a UTF-8 BOM). If *len* is different from 0 (C language binding only) the file name is converted from UTF-16 to UTF-8 regardless of the option *filenamehandling*. An error occurs if the file name cannot be converted or if the file name does not constitute valid UTF-8 or UTF-16.

On Windows it is OK to use UNC paths or mapped network drives as long as you have the necessary permissions (which may not be the case when running in ASP).

len (C language binding only) Length of *filename* (in bytes) for UTF-16 strings. If *len* = 0 a null-terminated string must be provided.

optlist Option list specifying document options according to Table 10.8. The following options can be used:
acceptdynamicxfa, allowjpeg2000, checkglyphlists, decompose, encodinghint, engines, fold, glyphcolor, glyphmapping, ignoreactualtext, imageanalysis, lineseparator, normalize, inmemory, paraseparator, password, pcosengines, repair, requiredmode, shrug, spotcolor, tetml, unknownchar, usehostfonts, wordseparator

Returns -1 on error, or a document handle otherwise. For example, it is an error if the input document or the TETML output file cannot be opened. If -1 is returned it is recommended to call *TET_get_errmsg( )* to find out more details about the error.

Details Within a single TET object an arbitrary number of documents may be kept open at the same time. However, a single TET object must not be used in multiple threads simultaneously without any locking mechanism for synchronizing the access.

Encryption: if the document is encrypted its user password must be supplied in the *password* option if the permission settings allow content extraction. The document's master password must be supplied if the permission settings do not allow content extraction. If the *requiredmode* option has been specified, documents can be opened even without the appropriate password, but operations are restricted. The *shrug* option can be used to enable content extraction from protected documents under certain conditions (see Section 5.1, »Extracting Content from protected PDF«, page 59).

Supported file systems on IBM System i: TET has been tested with PC type file systems only. Therefore input and output files should reside in PC type files in the IFS (Integrated File System). The *QSYS.lib* file system for input files has not been tested and is

not supported. Since *QSYS.lib* files are mostly used for record-based or database objects, unpredictable behavior may be the result if you use TET with *QSYS.lib* objects. TET file I/O is always stream-based, not record-based.

Table 10.8 Document options for TET_open_document( ) and TET_open_document_callback( )

option	description
accept-dynamicxfa	*(Boolean) If* `true`*, dynamic XFA forms can successfully be opened. Querying pCOS paths is the only reasonable activity. Calling TET_open_page( ) will fail since no meaningful text or images can be extracted. Default:* `false`
allowjpeg-2000	*(Boolean) If* `true`*, JPEG 2000 (*.jp2, *.jpf or *.j2k) is allowed as output format for TET_write_image_file( ) and TET_get_image_data( ). Otherwise JPEG 2000 is avoided in favor of TIFF which may result in larger image files. Default:* `true`
check-glyphlists	*(Boolean) If* `true`*, TET checks all builtin glyphmapping rules with* `condition=allfonts` *before text extraction starts. Otherwise the global glyphmapping rules are not applied. This option slows down processing, but is useful for certain kinds of TeX documents with glyph names which cannot be mapped to Unicode by default. Default:* `false`
decompose	*(Keyword or option list; not relevant for* `granularity=glyph` *and the* `Glyph` *element in TETML) Unicode decompositions which are applied to all characters which have a specified Unicode decomposition tag and are part of the specified Unicode set. These conditions are provided in the suboption name and value. Decompositions can be used to either remove or preserve the distinction between equivalent Unicode characters (see Section 7.3, »Unicode Postprocessing«, page 103).* *Default: see »Default decompositions«, page 109. However, if the* `normalize` *option has a value other than* `none`*, all default decompositions are disabled, i.e. setting the* `normalize` *option sets the default to* `decompose=none`*. User-specified decompositions can still be applied.* *The following keywords can be supplied instead of a list:* ***none*** *No decompositions are applied.* ***default*** *The default decompositions (see »Default decompositions«, page 109) are applied before other specified decompositions.* *The following suboptions for decompositions are supported:* ***canonical, circle, compat, final, font, fraction, initial, isolated, medial, narrow, nobreak, small, square, sub, super, vertical, wide*** *Each of these suboptions accepts a string or keyword which specifies the decomposition's domain, i.e. the set of Unicode characters to which the decomposition is applied. A string specifies a Unicode set for the domain. This can be used to restrict decompositions to subsets of the characters with the specified decomposition tag. Characters outside the domain will not be modified.* *As an alternative to a string for a Unicode set the following keywords can be supplied:* ***_all*** *The set of all Unicode characters, i.e. the decomposition is applied to all characters with the specified decomposition tag.* ***_none*** *The empty set, i.e. the decomposition will not be applied at all.*
encodinghint	*(String[1]) Name of an encoding which is used to determine Unicode mappings for glyph names which cannot be mapped by standard rules, but only by a predefined internal glyph mapping rule. The keyword* `none` *can be used to disable all predefined rules. Default:* `winansi`

Table 10.8 Document options for TET_open_document( ) and TET_open_document_callback( )

option	description
engines	*(Option list) Enable or disable TET engines for page parsing. Disabled engines never provide any information. Disabling engines which are not required improves performance (default: all engines are active):* **annotation** *(Boolean) Enable the annotation engine. Annotation contents are processed subject to the* image *and* text *suboptions.* **image** *(Boolean) Enable image extraction from pages and annotations. Implies* pcosengines={image=false}. **text** *(Boolean) Enable text extraction from pages and annotations.* **textcolor** *(Boolean) Enable the text color engine.* **vector** *(Boolean) Enable the vector graphics engine which is relevant for clipping and improved table detection.*
fold	*(Keyword or list of lists; the first element of each inner list is a Unicode set or keyword, the second element is a Unichar or a keyword; not relevant for* granularity=glyph *and the* Glyph *element in TETML) Apply a post-folding (equivalence mapping) to all characters in a folding domain specified as a Unicode set or keyword. The foldings are applied to all text except separator characters added by the* lineseparator, paraseparator, *or* wordseparator *options (see Section 7.3, »Unicode Postprocessing«, page 103). Default: see Table 7.3, page 105.* *The following keyword can be supplied instead of a list:* **none** *No foldings are applied.* *The following keyword can be supplied instead of a sublist:* **default** *The default foldings are applied. It is strongly recommended to append this keyword to user-supplied foldings because disabling the default foldings can lead to unwanted effects.* *The first element of each list specifies the folding's domain, i.e. the set of Unicode characters to which the folding is applied. A string specifies a Unicode set for the domain. If a character is included in multiple sets specified within the* fold *option, the first matching set definition has priority over all others. In order to avoid unexpected results it is recommended to use disjoint sets.* *As an alternative to specifying the domain as a Unicode set the following keywords can be used:* **_dehyphenation** *The folding is applied to hyphen characters which have been found within hyphenated words at line breaks. These characters are flagged as TET_ATTR_DEHYPHENATION_ARTIFACT in the* attributes *member returned by TET_get_char_info( ) and the Glyph/@*dehyphenation *attribute in TETML.* **_tetpua** *The folding is applied to the TET PUA values which are assigned to unmappable glyphs. These characters are flagged with the* unknown *member returned by TET_get_char_info( ) and the* Glyph/@unknown *attribute in TETML.* *The second element in each list contains the target character or action for the folding. It is specified with one of the following variants:* **(Unichar)** *Replace all characters in the domain with the specified Unicode character.* **preserve** *The characters in the domain are not modified.* **remove** *The characters in the domain are removed.* **shift** *Shift all characters in the domain by the specified value (which may be negative).* **unknownchar** *Replace all characters in the domain with the character specified in the* unknownchar *option, or apply the action specified in the* unknownchar *option.*
glyphcolor	*(Keyword) Select the color space in which fill and stroke colors for text are reported. This option is only relevant for text using* Separation *or* DeviceN *color. It affects the color information reported in both TET_char_info( )/TET_get_color_info( ) and TETML. Supported keywords (default:* native*):* **native** *Use the native PDF color space for all text colors.* **alternate** *Use the alternate color space for Separation and DeviceN text colors.*

Table 10.8 Document options for TET_open_document( ) and TET_open_document_callback( )

option	description
glyphmapping	*(List of option lists) A list of option lists where each option list describes a glyph mapping method for one or more font/encoding combinations which cannot be mapped with standard methods. The mappings are used in least-recently-set order. If the last option list contains the font name wildcard »*«, preceding mappings will no longer be used. Each rule consists of an option list according to Table 10.9. All glyph mappings which match a particular font name are applied to this font (default: predefined internal glyph mappings are applied).* *Note that glyph mapping rules can also be specified as an external resource in the UPR file (see Section 5.2, »Resource Configuration and File Searching«, page 61).*
ignore-actualtext	*(Boolean) If* `true`*, all* `ActualText` *mappings in the document are ignored. Default:* `false`
image-analysis	*(Option list) List of suboptions for controlling processing. The following suboption is supported:* **sizerange** *(List of two positive integers, or positive integer and keyword) Minimum and maximum edge length of extracted images in pixels. Images for which at least one edge lies outside the specified interval are ignored. The maximum can be specified with the keyword* `unlimited` *which means that no upper limit is active. Small image fragments will be removed before attempting image merging. In contrast, when size filtering is applied via the same-named page option, small images are removed after image merging. Default:* `{1 unlimited}`
lineseparator	*(Unichar; Only for* `granularity=page`*) Character to be inserted between lines (but not after the last line on a page)[2]. No line separator is inserted in CJK text. Default: U+000A*
normalize	*(Keyword; not relevant for* `granularity=glyph` *and the* `Glyph` *element in TETML) Normalize the text output to one of the Unicode normalization forms (default:* `none`*):* **none** *Do not apply any normalization.* **nfc** *Normalization Form C (NFC): canonical decomposition followed by canonical composition* **nfd** *Normalization Form D (NFD): canonical decomposition* **nfkc** *Normalization Form KC (NFKC): compatibility decomposition followed by canonical composition* **nfkd** *Normalization Form KD (NFKD): compatibility decomposition* *Since the Unicode normalization forms involve canonical and compatibility decompositions, combinations of the options* `decompose` *and* `normalize` *must be constructed carefully. Setting the* `normalize` *option to a value different from* `none` *sets the decomposition default to* `decompose=none`**.**
inmemory	*(Boolean; Only for TET_open_document( )) If* `true`*, TET will load the complete file into memory and process it from there. This can result in a tremendous performance gain on some systems (especially MVS) at the expense of memory usage. If* `false`*, individual parts of the document are read from disk as needed. Default:* `false`
paraseparator	*(Unichar; Only for* `granularity=page`*) Character to be inserted between paragraphs[2]. Default: U+000A*
password	*(String) The user, master or attachment password for encrypted documents. If the document's permission settings allow text copying then the user password is sufficient, otherwise the master password must be supplied.* *See the pCOS Path Reference to find out how to query a document's encryption status, and pCOS operations which can be applied even without knowing the user or master password.* *The* `shrug` *option can be used to enable content extraction from protected documents under certain conditions (see Section 5.1, »Extracting Content from protected PDF«, page 59).*
pcosengines	*(Option list) Enable or disable pCOS engines for document-wide resource collection. Turning off an engine means that the corresponding per-document and per-page pseudo object arrays will be empty. Disabling a pCOS engine does have effects on TET's corresponding engine, e.g. disabling the pCOS image engine will prevent image extraction by TET. Supported keywords (default: all pCOS engines are active):* `colorspace`, `extgstate`, `font`, `image`, `pattern`, `property`, `shading`, `template`

Table 10.8 Document options for TET_open_document( ) and TET_open_document_callback( )

option	description
repair	*(Keyword) Specifies how to treat damaged PDF documents. Repairing a document takes more time than normal parsing, but may allow processing of certain damaged PDFs. Note that some documents may be damaged beyond repair (default:* `auto`*):* ***force*** *Unconditionally try to repair the document, regardless of whether or not it has problems.* ***auto*** *Repair the document only if problems are detected while opening the PDF.* ***none*** *No attempt is made at repairing the document. If there are problems in the PDF the call fails.*
requiredmode	*(Keyword) The minimum pcosmode* (`minimum`/`restricted`/`full`) *which is acceptable when opening the document. The call will fail if the resulting* `pcosmode` *(see pCOS Path Reference) would be lower than the required mode. If the call succeeds the resulting* `pcosmode` *is at least the one specified in this option. However, it may be higher; e.g.* `requiredmode=minimum` *for an unencrypted document results in* `full` *mode. Default:* `full`
shrug	*(Boolean) If* `true`*, the shrug feature is activated to enable content extraction from protected documents under certain conditions (see Chapter 5.1, »Extracting Content from protected PDF«, page 59). By using the* `shrug` *option you assert that you will honor the PDF document author's rights. Default:* `false`
spotcolor	*(Keyword) Control treatment of spot color images in TET_write_image_file( ) and TET_get_image_data( ). Images with a* `Separation` *or* `DeviceN` *color space, i.e. one or more named process or spot colors are extracted as follows (default:* `ignore`*):* ***convert*** *Emit a grayscale or CMYK image if no custom spot colors are used. Otherwise convert spot colors to the corresponding alternate color space. For some images conversion to the alternate color space is not possible. In this case this method behaves like* `spotcolor=ignore` *(for a single custom spot color) or* `spotcolor=preserve` *(for two or more custom spot colors).* ***ignore*** *Like* `convert` *except that images with exactly one custom spot color are extracted as grayscale image and the spot color name is lost.* ***preserve*** *(Forces TIFF output) Emit a grayscale or CMYK image with one or more extra spot color channels if required for custom spot color names. TIFF images with preserved spot colors in extra channels work only in Adobe Photoshop and compatible programs, but not in some simple TIFF viewers.*

Table 10.8 Document options for TET_open_document( ) and TET_open_document_callback( )

option	description
tetml	*(Option list) TETML output is initiated and can be created page by page with TET_process_page( ). The following suboptions are supported:*
	elements *(Option list) Specify whether certain TETML elements are included in the output:*
	annotations *(Boolean) Emit* `/TET/Document/Pages[]/Page/Annotations` *if the document contains annotations. Default:* `true`
	attachments *(Boolean) Emit* `/TET/Document/Attachments` *if the document contains attachments. Default:* `true`
	bookmarks *(Boolean) Emit* `/TET/Document/Bookmarks` *if the document contains bookmarks. Default:* `true`
	destinations *(Boolean) Emit* `/TET/Document/Destinations` *if the document contains destinations. Default:* `true`
	docinfo *(Boolean) Emit* `/TET/Document/DocInfo` *element if the document contains document info entries. Default:* `true`
	fields *(Boolean) Emit* `/TET/Document/Pages[]/Page/Fields` *and* `TET/Document/SignatureFields` *if the document contains* `AcroForm` *fields or digital signatures. Default:* `true`
	javascripts *(Boolean) Emit* `/TET/Document/JavaScripts` *if the document contains JavaScript. Default:* `true`
	metadata *(Boolean) Emit* `/TET/Document/Metadata` *and/or* `/TET/Document/Images[]/Image/Metadata` *if the document contains XMP metadata on the document or image level. Default:* `true`
	options *(Boolean) The elements* `/TET/Document/Options` *and* `/TET/Document/Pages[]/Page/Options`. *Default:* `true`
	encodingname *(Keyword) Name to use in the XML encoding declaration of the text declaration of the generated TETML. The output is always created in UTF-8 (default:* `UTF-8`*):*
	_none *No encoding declaration is created; the output is still in UTF-8 format.*
	UTF-8 *The declaration* `encoding="UTF-8"` *is created.*
	Any other encoding name is used literally in the encoding declaration. The client is responsible for supplying a suitable encoding name and converting the generated TETML (which is UTF-8) to the specified encoding after TET finished TETML output.
	filename *(String) Name of the TETML file. If no filename is supplied, output is created in memory and can be retrieved with TET_get_tetml( ). If the call fails (i.e. the PDF input document could not successfully be opened), no TETML output is created.*
unknownchar	*(Unichar or keyword) Character or action to be applied to TET PUA characters for unmappable glyphs (see »Unmappable glyphs and the TET PUA«, page 113). The following keywords are supported (default: Unicode replacement character U+FFFD):*
	remove *Unmappable glyphs are removed. The value U+0000 is equivalent to* `remove`.
	preserve *Unmappable glyphs are represented by TET PUA values.*
usehostfonts	*(Boolean) If* `true`*, data for fonts which are not embedded, but are required for determining Unicode mappings is searched on the macOS or Windows host operating system. Default:* `true`
wordseparator	*(Unichar; Only for* `granularity=line` *and* `page`*) Character to be inserted between words[2]. Default: U+0020*

1. Encoding name according to Section 10.1.5, »Encoding Names«, page 165
2. Use U+0000 to disable the separator.

Table 10.9 Suboptions for the glyphmapping *option of TET_open_document( ) and TET_open_document_callback( )*

option	*description*
codelist	*(String) Name of a codelist resource to be applied to the font. It will have higher priority than an embedded ToUnicode CMap or encoding entry.*
fold	*Apply a pre-folding (equivalence mapping) to all characters in a folding domain specified as a Unicode set; see description of option* fold *in Table 10.8. The keywords* remove, preserve *and* unknownchar *can not be used. Font-specific foldings with the* shift *keyword can be used to correct systematic errors in a font's ToUnicode CMap.*
fontname	*(Name string) Partial or full name of the font(s) which are selected for the rule. If a subset prefix has been supplied only the specified subset is selected. If no subset prefix has been supplied, all fonts where the name (without any subset prefix) matches are selected. The wildcard character »*« can be used to specify multiple similar font names. Default: **
fonttype	*(List of keywords) The glyphmapping will only be applied to the specified font types: * (designates all font types),* Type1, TrueType, CIDFontType2, CIDFontType0, Type3. *Default: **
force-encoding	*(List with one or two strings[1]; if there are two names, the first must be* winansi, macroman, *or* custom, *where* custom *matches any encoding)* *Fonts with an 8-bit encoding: Replace the first encoding with the encoding resource specified by the second name. If only one entry is supplied, the specified encoding is used to replace all instances of* MacRoman, WinAnsi, *and* MacExpert *encoding. If this option matches a font no other glyph mappings are applied to the same font.* *CID fonts: Only the single value* unicode *is supported. It interprets CID values as Unicode values.*
forcettsymbol-encoding	*(Keyword or string[1]) The name of an encoding which are used to determine Unicode mappings for embedded pseudo TrueType symbol fonts which are actually text fonts, or one of the following keywords (default:* none*):* ***auto*** *If the font's builtin encoding (see below) contains at least one Unicode character in the symbolic range U+F000-U+F0FF, the encoding specified in the* encodinghint *option is used to map the pseudo symbol characters to real text characters. Otherwise* encodinghint *will not be used, and the characters are mapped according to the* builtin *keyword.* ***builtin*** *Use the font's builtin encoding, which results from the Unicode mappings of the glyph names in the font's* post *table.* ***none*** *No encoding is enforced.* *The well-known TrueType fonts Wingdings* and Webdings* are always treated as symbol fonts.*
globalglyphlist	*(Boolean) If* true*, the specified glyph list is kept in memory until the end of the TET object, i.e. it can be applied to more than one document. Default:* false
glyphlist	*(String) Name of a glyphlist resource to be applied*
glyphrule	*(Option list) Mapping rule for numerical glyph names (in addition to the predefined rules). The option list must contain the following suboptions:* ***prefix*** *(String; may be empty) Prefix of the glyph names to which the rule is applied. The wildcard character »?« can be used. It matches exactly one character provided this character is different from 0-9.* ***base*** *(Keyword) Specifies the interpretation of glyph names:* ***ascii*** *Single-byte glyphnames are interpreted as the corresponding literal ASCII character (e.g.* 1 *is mapped to U+0031).* ***auto*** *Automatically determine whether glyph names represent decimal or hexadecimal values. If the result is not unique, decimal is assumed.* ***dec*** *The glyphnames are interpreted as a decimal representation of a code.* ***hex*** *The glyphnames are interpreted as a hexadecimal representation of a code.* ***encoding*** *(String) Name of an encoding resource which is used for this rule, or the keyword* none *to disable the rule.*

Table 10.9 Suboptions for the glyphmapping *option of TET_open_document( ) and TET_open_document_callback( )*

option	description
ignoreto-unicodecmap	*(Boolean) If* true*, a ToUnicode CMap for the font is ignored. Default:* false
override	*(Boolean; only reasonable together with the* glyphlist *or* glyphrule *option) If* true*, the glyphmapping rule is applied before the standard (builtin) glyph name mappings (i.e. the new mappings have priority over the builtin ones), otherwise the rule is applied after the builtin mappings. Default:* true
remove	*(Boolean) If* true*, all text which uses the specified font name(s) and/or font type(s) is removed from the retrieved text.*

1. Encoding name according to Section 10.1.5, »Encoding Names«, page 165

C++ ***int open_document_callback(void *opaque, tet_off_t filesize,***
size_t (*readproc)(void *opaque, void *buffer, size_t size),
int (*seekproc)(void *opaque, tet_off_t offset),
String optlist)

C ***int TET_open_document_callback(TET *tet, void *opaque, tet_off_t filesize,***
size_t (*readproc)(void *opaque, void *buffer, size_t size),
int (*seekproc)(void *opaque, tet_off_t offset),
const char *optlist)

Open a PDF document from a custom data source for content extraction.

opaque Pointer to some user data that might be associated with the input PDF document. This pointer is passed as the first parameter to the callback functions and can be used in any way. TET does not use the opaque pointer.

filesize Size of the PDF document in bytes. The *tet_off_t* type is defined conditionally in *tetlib.h*. It usually holds 64-bit values as offset type for large files beyond 2GB. The application must be built with Large File Support (LFS).

readproc C callback function which copies *size* bytes to the memory pointed to by *buffer*. If the end of the document is reached it may copy less data than requested. The function must return the number of bytes copied.

seekproc C callback function which sets the current read position in the document. *offset* denotes the position from the beginning of the document (0 meaning the first byte). If successful, this function must return 0, otherwise -1.

optlist Option list specifying document options according to Table 10.8.

Returns See *TET_open_document( )*.

Details See *TET_open_document( )*. The callback functions must not call any API method or macro with the same context pointer.

Bindings This method is only available in the C and C++ language bindings.

C++ Java C# **STET_**Cthe value 1parameterthe *opaque* pointer passed to *interrupt***void close_document(int doc)**

Perl PHP **close_document(long doc)**

C **void TET_close_document(TET *tet, int doc)**

Release a document handle and all internal resources related to that document.

doc A valid document handle obtained with *TET_open_document*()*.

Details Closing a document automatically closes all of its open pages. All open documents and pages are closed automatically when *TET_delete()* is called. It is good programming practice, however, to close documents explicitly when they are no longer needed. Closed document handles must no longer be used in any method call.

10.4 Page Methods

API methods in this section:

- *TET_open_page( )*
- *TET_close_page( )*

C++ Java C# ***int open_page(int doc, int pagenumber, String optlist)***

Perl PHP ***long open_page(long pagenumber, string optlist)***

C ***int TET_open_page(TET *tet, int doc, int pagenumber, const char *optlist)***

Open a page for content extraction.

doc A valid document handle obtained with *TET_open_document*( )*.

pagenumber The physical number of the page to be opened. The first page has page number 1. The total number of pages can be retrieved with *TET_pcos_get_number( )* and the pCOS path *length:pages*.

optlist Option list specifying page options according to Table 10.10. The following options can be used:

clippingarea, contentanalysis, docstyle, emptycheck, excludebox, fontsizerange, granularity, ignoreartifacts, ignoreinvisibletext, imageanalysis, includebox, layers, layoutanalysis, layouteffort, structureanalysis, topdown, vectoranalysis.

Returns A handle for the page, or -1 in case of an error. If -1 is returned it is recommended to call *TET_get_errmsg( )* to find out more details about the error.

Details Within a single document an arbitrary number of pages may be kept open at the same time. The same page may be opened multiply with different options. However, options can not be changed while processing a page.

Layer definitions (optional content groups): the contents of all visible layers on the page are extracted by default. This behavior can be modified with the *layers* option.

Table 10.10 Page options for TET_open_page( ) and TET_process_page( )

option	description
clippingarea	*(Keyword; ignored if* `includebox` *is specified) Specifies the area from which text and images are extracted (default:* `cropbox`*):* ***mediabox*** *Use the MediaBox (which is always present)* ***cropbox*** *Use the CropBox (the area visible in Acrobat) if present, else MediaBox* ***bleedbox*** *Use the BleedBox if present, else use cropbox* ***trimbox*** *Use the TrimBox if present, else use cropbox* ***artbox*** *Use the ArtBox if present, else use cropbox* ***unlimited*** *Consider all text, regardless of its location*
contentanalysis	*(Option list; not for* `granularity=glyph`*) List of suboptions according to Table 10.11 for controlling high-level content analysis and text processing.*

Table 10.10 Page options for TET_open_page( ) and TET_process_page( )

option	description
docstyle	*(Keyword) A hint which is used by the layout detection engine to select various parameters. These parameters optimize layout detection for situations where the document belongs to one of the classes below. If the document is known to fall into one of these classes layout detection results can be improved significantly by supplying a suitable value for this option. This option activates advanced layout recognition (default:* `generic`*):* ***book*** *Typical book* ***business*** *Business documents* ***cad*** *Technical or architectural drawings which are typically heavily fragmented* ***fancy*** *Fancy pages with complex layout* ***forms*** *Structured forms* ***generic*** *The most general document class without any further qualification.* ***magazines*** *Magazine articles* ***none*** *No specific document style is known and advanced layout recognition is disabled.* ***native*** *Disable layout recognition and return the contents in native page content ordering. This may be useful for layouts such as forms where text is placed all over the page and column detection is not desired, but rather row-by-row text retrieval.* ***papers*** *Newspaper* ***science*** *Scientific article* ***searchengine*** *The application is a search engine indexer or similar application, and mainly interested in retrieving the word list for the page as fast as possible. Table and page structure recognition are disabled.* ***simpleline*** *Assume a simplistic single-column page layout and process text as individual lines. Table detection and dehyphenation are disabled. This may be useful for invoices and other documents with a simple line-oriented layout. This* `docstyle` *is unrelated to* `granularity`*.* ***spacegrid*** *List-oriented report (often generated on mainframe systems) where the visual layout is generated using space characters. Since many heuristics like shadow detection and sophisticated word boundary detection are not required for this class of documents text extraction can be accelerated with this option.*
emptycheck	*(Boolean) If* `true` *normal content extraction is disabled. Instead, the box provided in the* `includebox` *option is used to check whether the box contains any text, image, or vector graphics (only a single includebox is supported). If the* `includebox` *option is not supplied the whole clipping area is checked. This can be used to identify empty pages. The following options are ignored:* `granularity`, `engines`, `fontsizerange`*. Clipping operators are ignored.* *The result of the check can be retrieved with a call to TET_get_text( ) which will return one of the strings* `empty` *or* `notempty` *instead of any page contents. Default:* `false`
excludebox	*(List of rectangles) Exclude the combined area of the specified rectangles from text and image extraction. Default: empty*
fontsize-range	*(List of two floats, or float and keyword) Two numbers specifying the minimum and maximum font size of text. Text with a size outside of this interval is ignored. The maximum can be specified with the keyword* `unlimited`*, which means that no upper limit is active. Default:* `{ 0 unlimited }`

Table 10.10 Page options for TET_open_page( ) and TET_process_page( )

option	description
granularity	*(Keyword) The granularity of the text fragments returned by TET_get_text( ); all modes except glyph will enable the Wordfinder. See »Text granularity«, page 86, for more details (default:* `word`*).* **glyph** *A fragment contains the result of one glyph, but may contain more than one character (e.g. for ligatures).* **word** *A fragment contains a word as determined by the Wordfinder.* **line** *A fragment contains a line of text, or the closest approximation thereof. Word separators are inserted between two consecutive words.* **page** *A fragment contains the contents of a single page. Word, line,and paragraph separators are inserted as appropriate.*
ignore-artifacts	*(Boolean) Ignore text and images which are marked as Artifact. This can be used to skip irrelevant page contents which are marked as artifact. Content may be marked as Artifact even in documents without tags. Default:* `false`
ignore-invisibletext	*(Boolean) If* `true`*, text with rendering mode 3 (invisible) is ignored. Default:* `false` *(since invisible text is mainly used for image+text PDFs containing scanned pages and the corresponding OCR text)*
image-analysis	*(Option list) List of suboptions according to Table 10.13 for controlling high-level image processing.*
includebox	*(List of rectangles) Restrict text and image extraction to the combined area of the specified rectangles. Default: the complete clipping area*
layers	*(Keyword) Treatment of page contents within layers (also known as optional content). Supported keywords (default:* `visible`*):* **all** *Extract all page contents regardless of layers. Text may be garbled and image merging may be spoiled if the contents of multiple layers overlap on the page.* **invisible** *Extract contents of all layers which are invisible by default and ignore all other layers.* **visible** *Extract contents of all layers which are visible by default and ignore all other layers.*
layout-analysis	*(Option list; not for* `granularity=glyph`*) List of suboptions according to Table 10.12 for controlling layout detection features.*
layouteffort	*(Keyword) Controls the quality/performance trade-off of layout recognition. Layout recognition can be improved by spending more effort, but this may slow down operation. The layout recognition effort can be controlled with the keywords* `none`*,* `low`*,* `medium`*,* `high`*, and* `extra`*. Default:* `low`
layouthint	*(Option list) Inform the layout recognition engine about the presence of certain page layout elements:* **subsummary** *(Keyword) Informs the engine about the presence of subsummaries (marginalia) and possibly also their position. Supported keywords (default:* `none`*):* **auto** *No subsummary detection* **left** *Try to detect subsummaries on the left side of the page.* **none** *Try to detect subsummaries automatically.* **right** *Try to detect subsummaries on the right side of the page.* **header** *(Boolean) If* `true`*, the engine tries to detect page headers (default:* `false`*).* **footer** *(Boolean) If* `true`*, the engine tries to detect page footers (default:* `false`*).*
maxvector-count	*(Float) Maximum number of vector objects to be taken into account by the vector graphics engine. Default: 500*
minvectorsize	*(Float) Minimum size of a vector object to be taken into account by the vector graphics engine. The size of a vector object is the length of the diagonal of its bounding box in points. Default: 5*
structure-analysis	*(Option list; not for* `granularity=glyph`*) List of suboptions according to Table 10.14 for controlling page structure analysis.*

Table 10.10 Page options for TET_open_page( ) and TET_process_page( )

option	description
topdown	*(Option list) Specify a coordinate system with the origin in the top left corner of the visible page, and* y *coordinates which increase downwards; otherwise the default coordinate system with the origin in the lower left corner is used. Enabling topdown coordinates enables the same coordinate system which is displayed in Acrobat. Supported suboptions:* **input** *(Boolean) If* `true`*, enable topdown coordinates for the following items (default:* `false`*):* *page options* `includebox`, `excludebox` **output** *(Boolean) If* `true`*, enable topdown coordinates for the following items (default:* `false`*):* *TET_char_info:* `y`, `alpha`, `beta` *TET_image_info:* `y`, `alpha`, `beta` *TETML:* `Destination/@bottom`, `Destination/@top`, *attributes* `@lly`, `@ury`, `@uly`, `@lry` *of the elements* `Cell`, `Box`, `Line`, `Table`, *attributes* `@y`, `@alpha`, `@beta` *of the elements* `Glyph`, `PlacedImage`.
vector-analysis	*(Option list; not for* `granularity=glyph`*) Suboptions according to Table 10.15 for controlling analysis of vector graphics for table and layout detection. If this option is present, vector graphics is taken into account for table and layout detection.*

Table 10.11 Suboptions for the `contentanalysis` *option of TET_open_page( ) and TET_process_page( )*

option	*description*
bidi	*(Keyword; ignored for* `granularity=glyph`*; has an effect only if right-to-left characters are present on the page) Control the inverse Bidi algorithm which reorders right-to-left and left-to-right text in a chunk (default:* `logical`*):* ***visual*** *Keep RTL and LTR characters in a chunk in visual order, i.e. do not apply the inverse Bidi algorithm* ***logical*** *Apply the inverse Bidi algorithm to bring the characters in a chunk in logical order.*
bidilevel	*(Keyword) The page's base level (i.e. the main direction of text progression) for the inverse Bidi algorithm and layout recognition (default:* `auto`*):* ***auto*** *Determine the main direction of text progression based on the content.* ***ltr*** *Assume left-to-right as main direction of text progression (e.g. Latin documents)* ***rtl*** *Assume right-to-left as main direction of text progression (e.g. Hebrew or Arabic documents)*
dehyphenate	*(Boolean) If* `true`*, hyphenated words are identified and the text fragments surrounding the hyphen are combined. The hyphen itself is treated according to the* `keephyphens` *option. Default:* `true`
dropcapsize	*(Float) The minimum size at which large glyphs are recognized as a drop cap. Drop caps are large characters at the beginning of a zone that are enlarged to »drop« down several lines. They are merged with the remainder of the zone and form part of the first word in the zone. Default: 35*
dropcapratio	*(Float) The minimum ratio of the font size of drop caps and neighboring text. Large characters are recognized as drop caps if their size exceeds* `dropcapsize` *and the font size quotient exceeds* `dropcapratio`*. In other words, this is the number of text lines spanned by drop caps. Default: 4 (drop caps spanning three lines are very common, but additional line spacing must be taken into account)*
includebox-order	*(Integer) If multiple include boxes have been supplied (see option* `includebox`*), this option controls how the order of boxes affects the Wordfinder (default: 0):* ***0*** *Ignore include box ordering when analyzing the page contents. The result is the same as if all the text outside the include boxes was deleted. This is useful for eliminating unwanted text (e.g. headers and footers) while not affecting the Wordfinder in any way.* ***1*** *Take include box ordering into account when creating words and zones, but not for zone ordering. A word never belongs to more than one box. The resulting zones are sorted in logical order. In case of overlapping boxes the text belongs to the box which is earlier in the list. Other than that, the ordering of include boxes in the option list doesn't matter. This setting is useful for extracting text from forms, extracting text from tables, or when include boxes overlap for complicated layouts.* ***2*** *Consider include box ordering for all operations. The contents of each include box are treated independently from other boxes, and the resulting text is concatenated according to the order of the include boxes. This is useful for extracting text from forms in a particular ordering, or extracting article columns in a magazine layout in a predefined order. In these cases advance knowledge about the page layout is required in order to specify the include boxes in appropriate order.*
keephyphen-glyphs	*(Boolean) If* `true` *and* `dehyphenate=true` *the hyphen glyph between parts of dehyphenated words are kept in the list of glyphs returned by TET_get_char_info( ) and the* `Glyph` *element in TETML. This is useful for applications which need detailed information about the position of hyphens, e.g. exactly replacing text on the page. Note that this is different from* `fold={{_dehyphenation remove}` *which only removes hyphens from the logical text returned by TET_get_text( ), but does not affect glyphs. Default:* `false`
linespacing	*(Keyword) Specify the typical vertical distance between text lines within a paragraph:* `small`*,* `medium`*,* `large` *or* `extra` *(default:* `medium`*)*

Table 10.11 Suboptions for the `contentanalysis` *option of TET_open_page( ) and TET_process_page( )*

option	description
maxwords	*(Integer or keyword) If the number of words on the page is not greater than the specified number (the keyword* `unlimited` *means that no limit is active) the detected zones on the page are merged appropriately and sorted. If the number of words on the page is greater than the specified number, no zones are built, and words are retrieved in page content reading order. Processing is faster in the latter case, but the ordering of the retrieved words may not be optimal. Setting this option to* `unlimited` *is recommended for large pages with many words, such as newspapers. Default: 10000 for* `docstyle=paper` *and* `fancy`*, otherwise 8000*
merge	*(Integer) Controls strip and zone merging (default: 2):* **0** *No merging after strip creation. This can significantly increase processing speed, but may create less than optimal output, and prevent some shadows from being detected properly.* **1** *Simple strip-into-zone merging: strips are merged into a zone if they overlap this particular zone, but don't overlap strips other than the next one (to avoid zone overlapping for non-shadow cases).* **2** *Advanced zone merging for out-of-sequence text: in addition to* `merge=1`*, multiple overlapping zones are combined into a single zone, provided the text contents of both zones do not overlap.*
numeric-entities	*(Keyword) Control word boundary detection for numeric entities such as numbers, fractions, and time (default:* `keep`*):* **split** *Split the entity according to the* `punctuationbreaks` *suboption.* **keep** *Keep the entity as a whole word.*
shadow-detect	*(Boolean) If* `true`*, redundant instances of overlapping text fragments which create a shadow or fake bold text are detected and removed. Default:* `true`
punctuation-breaks	*(Boolean)* *For* `granularity=word`*: if* `true`*, punctuation characters which are placed close to a letter are treated as word boundaries, otherwise they are included in the adjacent word. For example, this option may be useful for the treatment of URLs and mail addresses.* *For* `granularity=line`*,* `page` *: if* `true`*, punctuation characters which are placed close in between two letters, which are not capital letters or figures are treated as word boundaries, otherwise they are included in the adjacent word. For example, this option may be useful for the treatment of abbreviations.* *Default:* `false` *for* `docstyle=simpleline`*,* `spacegrid` *and* `forms`*, otherwise* `true`
remove-dummy-spaces	*(Keyword; unsupported) Remove space characters below glyphs (»dummy spaces«) which are created by some applications instead of soft hyphens. Such dummy spaces should generally be ignored when determining word boundaries since they appear in the middle of a word. Default:* `auto` **none** *Never remove space characters below glyphs.* **auto** *Conditionally remove space characters below glyphs.* **force** *Always remove space characters below glyphs.*
superscript	*(Integer) Controls subscript and superscript detection (default: 2):* **0** *No subscript and superscript detection* **1** *Simple subscript and superscript detection* **2** *Advanced algorithm for subscript and superscript detection*
useclasses	*(Boolean) If* `true`*, Unicode classification is considered to determine word boundaries. Default:* `true`
usemetrics	*(Boolean) If* `true`*, the distance between glyphs is compared with the width of the space glyph to determine word boundaries. Default:* `true`

Table 10.12 Suboptions for the `layoutanalysis` *option of TET_open_page() and TET_process_page()*

option	*description*
layout-astable	*(Boolean) If* `true`*, layout recognition will treat the zones on the page as one or more tables. The minimum number of columns which is required to consider the sequence as a table depends on the document style. If* `false`*, supertable recognition is disabled (default:* `true`*).*
layout-columnhint	*(Keyword) This option may improve zone reading order detection for complex layouts. Supported keywords (default:* `multicolumn`*):* ***multicolumn*** *The page contains multi-column text; zones are sorted column by column.* ***none*** *No hint available; zone ordering is determined by page content order.* ***singlecolumn*** *The page contains single-column text; zones are sorted row by row. This keyword should be used with* `layouteffort=low`*.*
layoutdetect	*(Integer) Specifies the depth of recursive layout recognition (default: depends on* `docstyle`*):* ***0*** *No layout recognition.* ***1*** *Layout recognition for the whole page. This is sufficient for the vast majority of documents.* ***2*** *Layout recognition for the results of level 1. This is required for layouts with different multi-column sublayouts and titles on different parts of the page as well as multi-paragraph tables.* ***3*** *Layout recognition for the results of level 2. This is required only for very complex layouts.*
layoutrow-hint	*(Option list) Control layout row processing. Supported options (default:* `none`*):* ***full*** *Enable layout row processing.* ***none*** *Disable layout row processing.* ***separation*** *(Keyword) Enable layout row processing, but disable it if layout recognition suspects a supertable. The following suboptions can be supplied:* ***preservecolumns*** *Try to keep vertical columns based on the geometric relationship between zones. This is recommended if zones within columns are separated by large gaps (e.g. caused by images).* ***thick*** *Try to combine neighboring zones and place them in the same layout row. This results in a smaller number of larger layout rows. This is recommended for complex layouts, such as magazines and papers where paragraphs within columns are separated from each other by more than the font size, and for layouts with several multi-column articles one under the other.* ***thin*** *Try to separate neighboring zones and place them in different layout rows. This results in a larger number of smaller layout rows.* *Example:* `layoutanalysis = {layoutrowhint={full separation=thick}}`
mergetables	*(Integer) Tables with a single row are skipped during table recognition, and treated as regular zones. If two sequential zones are tables (even with only a single row) they can be combined. (default:* `none`*):* ***down*** *Combine downstairs only.* ***none*** *Don't combine.* ***up*** *Combine upstairs only.* ***updown*** *Combine in both directions.*
splithint	*(Keyword or option list) Activate special treatment of double-page spreads (or even pages consisting of more spreads). The page may be divided vertically or horizontally in two or more sections. The keyword* `includebox` *means that the split areas are defined by the* `includebox` *option. Alternatively the following options can be supplied:* ***x*** *(Float) Divider for the x axis, e.g. 0.5 for a double-page spread, 0.33 for a three-page spread.* ***y*** *(Float) Divider for y axis.*
standalone-fontsize	*(Float) Minimum font size for huge glyphs. Huge glyphs form single-glyph strips, and will not be combined with other zones (default: 70).*

Table 10.12 Suboptions for the `layoutanalysis` *option of TET_open_page( ) and TET_process_page( )*

option	*description*
supertable-columns	*(Integer; only if* `layoutastable=true`*) Minimum number of columns in a layout row to consider the sequence of zones as a supertable. When a table is created from paragraphs, these columns are recognized as separate zones instead of being combined. As a consequence of this, layout recognition can identify these zone sequences as a table (default: 4).*
tabledetect	*(Integer) Specifies the depth of recursive table recognition (default: depends on* `docstyle`*):* ***0*** *No position-based table recognition; tables with cell borders are still detected if the suboption* `structures=usevectoronly` *of the* `vectoranalysis` *option is set.* ***1*** *Table recognition for each zone.* ***2*** *Table recognition for each table cell detected in level 1. This is required for nested tables and resolving row spans.*

Table 10.13 Suboptions for the `imageanalysis` *option of TET_open_page( ) and TET_process_page( )*

option	*description*
heightrange	*(List of two positive integers, or positive integer and keyword) Minimum and maximum height of extracted images in pixels. Images for which its height and the height of its mask (if present) lies outside the specified interval are ignored. The maximum can be specified with the keyword* `unlimited` *which means that no upper limit is active. Default: the value of* `sizerange`
merge	*(Option list) Control image merging. This process combines adjacent images which together may form a single larger image. This is useful for multi-strip images where the individual strips have been preserved in the PDF, and for background images which are broken into a large number of very small images. Supported options:* ***disable*** *(Boolean) If* `true`*, image merging is disabled. Default:* `false` ***gap*** *(Float) Maximum gap or overlap between two images to be considered for merging.The value is interpreted as absolute distance in points, and also as number of pixels. Default: 1.0*
sizerange	*(List of two positive integers, or positive integer and keyword) Minimum and maximum edge length of extracted images in pixels. Images for which at least one edge lies outside the specified interval are ignored. The maximum can be specified with the keyword* `unlimited` *which means that no upper limit is active. The* `sizerange` *option is a shortcut for setting both* `heightrange` *and* `widthrange` *to the same values. Default:* `{20 unlimited}`
smallimages	*Deprecated, use* `heightrange/sizerange/widthrange`
widthrange	*(List of two positive integers, or positive integer and keyword) Minimum and maximum width of extracted images in pixels. Images for which the width and the width of its mask (if present) lies outside the specified interval are ignored. The maximum can be specified with the keyword* `unlimited` *which means that no upper limit is active. Default: the value of* `sizerange`

Table 10.14 Suboptions for the `structureanalysis` *option of TET_open_page( ) and TET_process_page( )*

option	*description*

bullets	*(List of option lists; relevant for* `list=true`*) Combinations of Unicode characters and font names which are used as bullet characters in lists. Supported suboptions:*
	bulletchars *(List of Unicode values) One or more Unicode values for the bullet characters. If this suboption is not supplied, all characters using the specified* `fontname` *are treated as bullet characters. Default: a list containing dozens of characters which are commonly used as bullets*
	fontname *(String) Name of the font from which bullet characters are drawn. If this suboption is not supplied, the characters specified in the* `bulletchars` *suboption are always treated as bullet characters.*
	Examples:
	`bullets={{fontname=ZapfDingbats}}`
	`bullets={{bulletchars={U+2022}}}`
	`bullets={{fontname=KozGoPro-Medium bulletchars={U+2460 U+2461 U+2462 U+2463 U+2464}}}`

list	*(Boolean) Enable list recognition for TETML output (default:* `true`*). If* `false`*, no information about list structure is determined.*
overpagelists	*(Keyword) Treatment of potential list fragments which span pages. Supported keywords (default:* `ascending`*):* ***ascending*** *Store list information across pages only if the pages are processed in ascending order without gaps.* ***always*** *Always store list information across pages regardless of the order of page processing.* ***never*** *Never store list information across pages, i.e. detected lists are limited to the same page and numbered list don't necessarily start at 1.*

Table 10.15 Suboptions for the `vectoranalysis` *option of TET_open_page( ) and TET_process_page( )*

option	*description*
closetablearea	*(Boolean) If* `true`*, create table border for analysis even if none is present. Default:* `false`
ignorelines	*(Keyword) Specify which lines to exclude from the analysis. Supported keywords (default:* `none`*):* ***horizontal*** *Ignore horizontal lines.* ***none*** *Don't ignore any lines.* ***vertical*** *Ignore vertical lines.*
pagesizelines	*(Boolean) If* `true`*, take into account large lines which are almost as long as the page size. Default:* `false`
splitsequence	*(Boolean) If* `true`*, vertical lines are allowed to split text sequences (or horizontal lines for rotated text). Default:* `false`
structures	*(Keyword) Specify the algorithm for interpreting vector graphics for table detection. Supported keywords (default:* `unions`*):* ***tables*** *Extended* `unions` *mode: the algorithm additionally checks whether unions form a table net. If so, the result is treated as a single table zone.* ***unions*** *The algorithm tries to build sub-layout unions from lines. If such a union is built, it is treated as a complete sub-layout entity, i.e. all enclosed text zones belong to the sub-layout.* ***usevectoronly*** *The algorithm ignores text positions for table detection and uses only vector graphics. This algorithm is fast and works well for simple tables, but doesn't handle row and column spans. It is recommended for tables with complete cell borders, i.e. tables where vector graphics can reliably be used for identifying each table cell. Such tables are detected even with the suboption* `tabledetect=0` *of the* `layoutanalysis` *option. Default:* `false` ***vectoriterative*** *This algorithm uses only vector graphics, and operates iteratively to identify row and column spans. It requires more time and is recommended for complex table layouts including table cells which span multiple rows or columns.*

C++ Java C# ***void close_page(int page)***

Perl PHP ***close_page(long page)***

C ***void TET_close_page(TET *tet, int page)***

Release a page handle and all related resources.

page A valid page handle obtained with *TET_open_page( )*.

Details All open pages of the document are closed automatically when *TET_close_document( )* is called. It is good programming practice, however, to close pages explicitly when they are no longer needed. Closed page handles must no longer be used in any method call.

10.5 Text and Glyph Details Retrieval Methods

API methods in this section:

- *TET_get_text( )*
- *TET_get_char_info( )*
- *TET_get_color_info( )*

C++ Java C# ***String get_text(int page)***

Perl PHP ***string get_text(long page)***

C ***const char *TET_get_text(TET *tet, int page, int *len)***

Get the next text fragment from a page's content.

page A valid page handle obtained with *TET_open_page( )*.

len (C and RPG language bindings only) A pointer to a variable which will hold the length of the returned string depending on the *outputformat* option of *TET_set_option( )*:

If *outputformat=utf8* the length is reported as number of Unicode characters. The number of bytes in the null-terminated string (which is identical to the number of 8-bit code units) can be determined with the *strlen( )* function.

If *outputformat=utf16* the length is reported as number of 16-bit code units; surrogate pairs are counted as two code units. The number of bytes in the string is *2*len*.

If *outputformat=utf32* the length is reported as number of 32-bit code units (which is identical to the number of Unicode characters). The number of bytes in the string is *4*len*.

Returns A string containing the next text fragment on the page. The length of the fragment is determined by the *granularity* option of *TET_open_page( )*. Even for *granularity=glyph* the string may contain more than one character (see Section 7.1, »Important Unicode Concepts«, page 97).

If all text on the page has been retrieved an empty string or null object is returned (see below). In this case *TET_get_errnum( )* can be called to find out whether there is no more text because of an error on the page, or because the end of the page has been reached.

Details If the page option *ignoreartifacts* has been supplied, text marked as Artifact is ignored. If the page option *ignoreinvisibletext* has been supplied, text with rendering mode 3 (invisible) is ignored.

Bindings C language binding: the result is provided as null-terminated UTF-8 (default) or UTF-16/UTF-32 string according to the *outputformat* option of *TET_set_option( )*. On IBM System i and IBM Z EBCDIC-encoded UTF-8 can also be selected, and is enabled by default. The returned data buffer can be used until the next call to this method. If no more text is available a NULL pointer and **len=0* is returned.

C++: the result is provided as Unicode in the selected string format (see Section 3.2, »C++ Binding«, page 28). If no more text is available an empty string is returned.

Java, .NET and Objective-C: the result is provided as Unicode string. If no more text is available a null *(nil* in Objective-C) object is returned.

Perl and PHP: the result is provided as UTF-8 (default) or UTF-16/UTF-32 string according to the *outputformat* option of *TET_set_option( )*. If no more text is available *undef* is returned in Perl and an empty string in PHP.

Python: the result is provided as UTF-8 (default) or UTF-16/UTF-32 string according to the *outputformat* option of *TET_set_option( )*. In Python 3 UTF-16/UTF-32 results are returned as bytes. If no more text is available, *None* is returned.

Ruby: the result is provided as UTF-8 (default) or UTF-16/UTF-32 string according to the *outputformat* option of *TET_set_option( )*. If no more text is available a nil object is returned.

RPG language binding: the result is provided as Unicode string according to the *outputformat* option of *TET_set_option( )*. If no more text is available *null* is returned.

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)***

C ***const TET_char_info *TET_get_char_info(TET *tet, int page)***

Get detailed information for the next glyph in the most recent text chunk.

page A valid page handle obtained with *TET_open_page( )*.

Note *The name of this method is a misnomer. It should better be called TET_get_glyph_info( ) since it reports information about visual glyphs on the page, not the corresponding Unicode characters.*

Returns If no more glyphs are available for the most recent text fragment returned by *TET_get_text( )*, a binding-specific value is returned. See section *Bindings* below for more details.

Details This method can be called one or more times after *TET_get_text( )*. It will advance to the next glyph for the current text chunk associated with the supplied page handle (or return nothing if there are no more glyphs), and provide detailed information for this glyph. There will be *one* or more successful calls to this method for a text chunk with *N* glyphs and *M* logical characters. The relationship between *N* and *M* depends on the granularity:

- For *granularity=glyph* each text chunk corresponds to a single glyph, i.e. *N = 1*. One glyph corresponds to one character in many cases, i.e. *M = 1*. However, for ligature glyphs a single glyph creates multiple characters, i.e. *M > 1* and *TET_get_char_info( )* must be called more than once.
- For granularities other than *glyph* a sequence of glyphs creates a sequence of characters, where each glyph may contribute to 0, 1, or more characters. The sequence of glyphs serves as raw material for the sequence of Unicode characters. In other words, there is no fixed relationship between *N* and *M*. The relationship between *N* and *M* may be influenced by content analysis (e.g. hyphens are removed by the dehyphenation process) or Unicode postprocessing (e.g. characters are added or deleted because of a folding).

For granularities other than *glyph* this method advances to the next glyph which contributes to the chunk returned by the most recent call to *TET_get_text( )*. This way it is possible to retrieve glyph details when the Wordfinder is active and a text chunk may

contain more than one character. In order to retrieve all glyph details for the current text chunk this method must be called repeatedly until it returns no more info.

The glyph details in the structure or properties/fields are valid until the next call to *TET_get_char_info( )*, *TET_get_image_info( )* or *TET_close_page( )* with the same page handle. Since there is only a single set of glyph info properties/fields per TET object, clients must retrieve all glyph info before they call *TET_get_char_info( )* again for the same or another page or document.

Bindings C and C++ language bindings: If no more glyphs are available for the most recent text chunk returned by *TET_get_text( )*, a NULL pointer is returned. Otherwise, a pointer to a *TET_char_info* structure containing information about a single glyph is returned. The members of the data structure are detailed in Table 10.16.

ava, .NET, and Objective-C language bindings: -1 is returned if no more glyphs are available for the most recent text chunk returned by *TET_get_text( )*, otherwise 1. Individual glyph info can be retrieved from the TET properties/public fields according to Table 10.16. All properties/fields contain the value -1 (the *unknown* field contains *false)* if they are accessed although the method returned -1.

Perl and Python language bindings: 0 is returned if no more glyphs are available for the most recent text chunk returned by *TET_get_text( )*, otherwise a hash containing the keys listed in Table 10.16. Individual glyph info can be retrieved with the keys in this hash.

PHP language binding: an empty (null) object is returned if no more glyphs are available for the most recent text chunk returned by *TET_get_text( )*, otherwise an object containing the fields listed in Table 10.16. Individual glyph info can be retrieved from the member fields of this object. Integer fields in the glyph info object are implemented as *long* in the PHP language binding.

Ruby binding: *nil* (null object) is returned if no more glyphs are available, and a *TET_char_info* object otherwise.

*Table 10.16 Members of the TET_*`char_info` *structure (C, C++, Ruby), equivalent public fields (Java, PHP, Objective-C), keys (Perl) or properties (.NET) with their type and meaning. See Section 6.2, »Page and Text Geometry«, page 74 and Section 6.3, »Text Color«, page 80 for more details.*

property/ field name	***explanation***
uv	*(Integer) Unicode value of the current glyph in UTF-32 format (regardless of the* `outputformat` *option). For granularities other than* `glyph` *this may be an artificial value or an inserted separator character which has no relationship to the final text chunk. For* `granularity=glyph` *the sequence of Unicode values for the glyphs is identical to the logical text, but for other granularities it may be modified by various processing steps.*
type	*(Integer) Type of the character. The following types describe a real glyph on the page. The values of all other properties/fields are determined by the corresponding glyph:* ***0*** *(TET_CT_NORMAL) Normal character which corresponds to exactly one glyph* ***1*** *(TET_CT_SEQ_START) Start of a sequence (e.g. ligature)* *The following types describe artificial characters which do not correspond to any glyph on the page. The* x *and* y *fields specify the most recent real glyph's endpoint, the width field is 0, and all other fields except* uv *contain the values corresponding to the most recent real glyph:* ***10*** *(TET_CT_SEQ_CONT) Continuation of a sequence, e.g. ligature* ***12*** *(TET_CT_INSERTED) Inserted word, line, or paragraph separator*

Table 10.16 Members of the TET_char_info structure (C, C++, Ruby), equivalent public fields (Java, PHP, Objective-C), keys (Perl) or properties (.NET) with their type and meaning. See Section 6.2, »Page and Text Geometry«, page 74 and Section 6.3, »Text Color«, page 80 for more details.

property/ field name	explanation
attributes	*(Integer) Glyph attributes expressed as bits which can be combined:*
	(none) *(TET_ATTR_NONE) If no bits are set no special glyph attribute has been detected.*
	bit 0 *(TET_ATTR_SUB) Geometric or semantic subscript*
	bit 1 *(TET_ATTR_SUP) Geometric or semantic superscript*
	bit 2 *(TET_ATTR_DROPCAP) Drop cap character (initial large character at the start of a paragraph)*
	bit 3 *(TET_ATTR_SHADOW) Glyph- or word-based shadow duplicate of this glyph has been removed*
	bit 4 *(TET_ATTR_DEHYPHENATION_PRE) Glyph represents last character before hyphenation point.*
	bit 5 *(TET_ATTR_DEHYPHENATION_ARTIFACT) Hyphenation character which was removed unless* `contentanalysis={keephyphenglyphs=true}` *was specified.*
	bit 6 *(TET_ATTR_DEHYPHENATION_POST) Glyph represents the character after hyphenation point.*
	bit 8 *(TET_ATTR_ARTIFACT) Glyph represents an Artifact (irrelevant content).*
unknown	*(Boolean; in C, C++ and Perl: integer) Usually* `false` *(0), but* `true` *(1) if the original glyph could not be mapped to Unicode and has been replaced with the character specified as* `unknownchar`.
x, y	*(Double) Position of the glyph's reference point. The reference point is the lower left corner of the glyph box for horizontal writing mode, and the top center point for vertical writing mode. For artificial characters the* x, y *coordinates are those of the end point of the most recent real character.*
width	*(Double) Width of the corresponding glyph (for both horizontal and vertical writing mode). For artificial characters (i.e. inserted separators with* `type=12` *and hyphenation characters with attribute bit 5 set) the width is 0.*
height	*(Double) For vertical writing mode: height of the corresponding glyph according to the font metrics and text output parameters (e.g. character spacing). The height is positive in the default coordinate system, but negative for* `topdown` *coordinates. In monospaced vertical fonts all glyphs have* `fontsize` *as height unless character spacing has been applied. Artificial characters (e.g. separators) have a height of 0.* *For horizontal writing mode an approximation of the glyph height is supplied. This approximate value is derived from font properties and therefore identical for all glyphs in a font. There is no guarantee that the visible glyph has the exact height value supplied here.*
alpha	*(Double) Direction of text progression in degrees measured counter-clockwise (or clockwise for topdown coordinates). For horizontal writing mode this is the direction of the text baseline; for vertical writing mode it is the digression from the standard vertical direction. The angle is in the range -180° < alpha ≤ +180°. For standard horizontal text as well as for standard text in vertical writing mode the angle is 0°.*
beta	*(Double) Text slanting angle in degrees measured counter-clockwise (or clockwise for topdown coordinates), relative to the perpendicular of alpha. The angle is 0° for upright text, and negative for italicized (slanted) text (positive for topdown coordinates). The angle is in the range -180° <* `beta` *≤ 180°, but different from ±90°. If* `abs(beta)` *> 90° the text is mirrored at the baseline.*
fontid	*(Integer) Index of the font in the* `fonts[]` *pseudo object (see the pCOS Path Reference).* `fontid` *is never negative.*
fontsize	*(Double) Size of the font (always positive); the relation of this value to the actual height of glyphs is not fixed, but may vary with the font design. For most fonts the font size is chosen such that it encompasses all ascenders (including accented characters) and descenders.*

Table 10.16 Members of the TET_char_info structure (C, C++, Ruby), equivalent public fields (Java, PHP, Objective-C), keys (Perl) or properties (.NET) with their type and meaning. See Section 6.2, »Page and Text Geometry«, page 74 and Section 6.3, »Text Color«, page 80 for more details.

property/ field name	explanation
textrendering	*(Integer) Text rendering mode (plain integers, not bit positions):*
	0 *(TET_TR_FILL) Fill text*
	1 *(TET_TR_STROKE) Stroke text (outline)*
	2 *(TET_TR_FILLSTROKE) Fill and stroke text*
	3 *(TET_TR_INVISIBLE) Invisible text (e.g. for OCR text)*
	4 *(TET_TR_FILL_CLIP) Fill text and add it to the clipping path*
	5 *(TET_TR_STROKE_CLIP) Stroke text and add it to the clipping path*
	6 *(TET_TR_FILLSTROKE_CLIP) Fill and stroke text and add it to the clipping path*
	7 *(TET_TR_CLIP) Add text to the clipping path*
	Text in Type 3 fonts: `textrendering=3` *and 7 result in invisible text; all other values of* `textrendering` *are irrelevant and are ignored.*
colorid	*(Integer) Index of the text color which represents the combination of fill color, stroke color, and text rendering. All occurrences of the same combination in a document are represented by the same color id. Different combinations are represented by different ids, which means that colors of multiple glyphs can be checked for equality by comparing their color ids. For example, by comparing the* `colorid` *values of successive glyphs you can identify changes in text color. The exact color space and color components for filling and/or stroking text can be retrieved with TET_get_color_info( ).* *For* `Separation` *and* `DeviceN` *colors the alternate colors of a* `Separation` *or* `DeviceN` *color spacecan be requested with the document option* `glyphcolor`**.**

C++ ***const TET_color_info *get_color_info(int doc, int colorid, String optlist)***

C# Java ***int get_color_info(int doc, int colorid, String optlist)***

Perl PHP ***object get_color_info(long doc, long colorid, string optlist)***

C ***const TET_color_info *TET_get_color_info(TET *tet, int doc, int colorid, const char *optlist)***

Retrieve color details for a color id which has been retrieved with *TET_get_char_info*.

doc Valid document handle obtained with *TET_open_document*( )*.*

colorid Valid color id obtained from the *colorid* member of *TET_get_char_info( ).*

optlist Option list according to Table 10.17 specifying the kind of color to retrieve.

Table 10.17 Option for TET_get_color_info( )

option	***description***
usage	*(Keyword) Usage of the color (default:* `fill`*)*
	fill *Color used for filling*
	stroke *Color used for stroking*

Returns C/C++: A pointer to a structure with details about the requested color space and color. Perl, Python, PHP, Ruby: a hash or object containing the requested color information. Java, .NET, Objective-C: the fixed value 1.

Details This method returns details about the color specified by *colorid*. Depending on the *usage* option the fill or stroke color associated with *colorid* is reported.

Bindings C and C++ language bindings: A pointer to a *TET_color_info* structure containing information about the requested color is returned. The members of the data structure are detailed in Table 10.18.

Java, .NET, and Objective-C language bindings: color information can be retrieved from the TET properties/public fields according to Table 10.18.

Perl, Python, PHP and Ruby language bindings: color information can be retrieved from a hash or object containing the keys/fields/members listed in Table 10.18.

*Table 10.18 Members of the TET_*`color_info` *structure (C, C++, Ruby), equivalent public fields (Java, PHP, Objective-C), keys (Perl) or properties (.NET) with their type and meaning. See Section 6.3, »Text Color«, page 80 for more details.*

property/ field name	***explanation***
colorspaceid	*(Integer) Index of the color space in the* `colorspaces[]` *pseudo object (see the pCOS Path Reference), or -1 if no color is applied to the glyph, e.g. for invisible text* (`textrendering=3`).
patternid	*(Integer) Index of the pattern in the* `patterns[]` *pseudo object (see the pCOS Path Reference), or -1 if no pattern is applied to the glyph.*
components[1]	*(Array of double values) Color component values which must be interpreted in the color space reported with* `colorspaceid`*.* *C and C++ language bindings: The number of relevant array entries is available in the* `n` *member.*
n[1]	*(Integer; C and C++ language bindings only) Number of array entries in the* `components` *member*

1. This member must only be accessed if the `colorspaceid` *member is not -1.*

10.6 Image Retrieval

API methods in this section:

- *TET_get_image_info( )*
- *TET_write_image_file( )*
- *TET_get_image_data( )*

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)***

C ***const TET_image_info *TET_get_image_info(TET *tet, int page)***

Retrieve information about the next image on the page (but not the actual pixel data).

page A valid page handle obtained with *TET_open_page( )*.

Returns If no more images are available on the page, a binding-specific value is returned, otherwise image details are available in a binding-specific manner. See section *Bindings* below for more details.

Details This method advances to the next image associated with the supplied page handle and provides detailed information for the image. If there are no more images on the page the method returns 0, -1 or NULL. The following types of images are ignored:

- Images used as mask are ignored. They can be retrieved via pCOS and the *maskid* pseudo object (see Section 8.5.2, »Image Masks and Soft Masks«, page 131).
- Images which have been consumed by the merging process and merged to form a larger image (i.e. *mergetype=consumed)* are ignored.
- Images which have been eliminated by the small image filter (see Section 8.4, »Small and Large Image Filtering«, page 129) are ignored.
- Images which are located completely outside the extraction area specified by the *clippingarea, excludebox,* and *includebox* options are ignored.
- Images which are marked as Artifact are skipped if the *ignoreartifacts* page option has been supplied.

The image details in the structure or properties/fields are valid until the next call to *TET_get_image_info( )* *TET_get_text_info( )* or *TET_close_page( )* with the same page handle. Since there is only a single set of image info properties/fields per TET object, clients must retrieve all image info before they call *TET_get_image_info( )* again for the same or another page.

Bindings C and C++ language bindings: If no more images are available on the page a NULL pointer is returned, otherwise a pointer to a *TET_image_info* structure containing information about the image. The members of the data structure are detailed in Table 10.19.

Java, .NET, and Objective-C language bindings: -1 is returned if no more images are available on the page, otherwise 1. Individual image info can be retrieved from the TET properties/fields according to Table 10.19. All properties/fields contain the value -1 if they are accessed although the method returned -1.

Perl and Python language bindings: 0 is returned if no more images are available on the page, otherwise a hash containing the keys listed in Table 10.19. Individual image info can be retrieved with the keys in this hash.

PHP language binding: an empty (null) object is returned if no more images are available on the page, otherwise an object of type *TET_image_info*. Individual image info can be retrieved from its fields according to Table 10.19. Integer fields in the image info object are implemented as *long* in the PHP language binding.

Ruby binding: nil (null object) is returned if no more images are available, and a *TET_image_info* object otherwise.

Table 10.19 Members of the TET_image_info structure (C, C++, Ruby), equivalent public fields (Java, PHP, Objective-C), and properties (.NET) with their type and meaning. See Section 8.1, »Image Extraction Basics«, page 119, for details.

property/ field name	explanation
x, y	(Double) Position of the image's reference point. The reference point is the lower left corner of the image.
width, height	(Double) Width and height of the image on the page in points, measured along the image's edges
alpha	(Double) Direction of the pixel rows. The angle is in the range -180° < alpha ≤ +180°. For upright images alpha is 0°.
beta	(Double) Direction of the pixel columns, relative to the perpendicular of alpha. The angle is in the range -180° < beta ≤ +180°, but different from ±90°. For upright images beta is in the range -90° < beta < +90°. If abs(beta) > 90° the image is mirrored at the baseline.
imageid	(Integer) Index of the image in the pCOS pseudo object images[ ]. Detailed image and mask properties can be retrieved via the entries in this pseudo object (see the pCOS Path Reference).
attributes	(Integer) Image attributes expressed as bits which can be combined. The following bits are defined (the least significant bit is bit 0): **bit 8** (TET_ATTR_ARTIFACT) Image represents an Artifact (irrelevant content). Artificial images which have been merged from Artifact images are also marked as Artifact. **bit 9** (TET_ATTR_ANNOTATION) Image extracted from an annotation (appearance stream). **bit 10** (TET_ATTR_PATTERN) Image extracted from a pattern. **bit 11** (TET_ATTR_SOFTMASK) Image extracted from a soft mask in a graphics state (defined in a Transparency Group XObject)

C++ Java C# **int write_image_file(int doc, int imageid, String optlist)**

Perl PHP **long write_image_file(long doc, long imageid, string optlist)**

C **int TET_write_image_file(TET *tet, int doc, int imageid, const char *optlist)**

Write image data to disk.

doc A valid document handle obtained with *TET_open_document*( )*.

imageid pCOS ID of the image. This ID can be retrieved from the *imageid* field after a successful call to *TET_get_image_info( )*, or by looping over all entries in the *images* pseudo object (there are *length:images* entries in this array).

optlist Option list specifying page options according to Table 10.20. The following options can be used:

dpi, filename, keepiccprofile, keepxmp, typeonly, validatejpeg.

The following options of other method also affect the generated image output:

- *TET_open_document*( ): allowjpeg2000, spotcolor* (see Table 10.8)
- *TET_open_page/TET_process_page( ): imageanalysis* (see Table 10.10 and Table 10.13)

Returns -1 on error, or a value greater than 0 otherwise. If -1 is returned it is recommended to call *TET_get_errmsg( )* to find out more details about the error. No image output is created in case of an error. If the return value is different from -1 it indicates that the image can be extracted in the file format indicated by the return value:

- 10 *(TET_IF_TIFF):* image extracted as TIFF *(.tif)*
- 20 *(TET_IF_JPEG):* image extracted as JPEG *(.jpg)*
- 31 *(TET_IF_JP2):* image extracted as plain JPEG 2000 *(.jp2)*
- 32 *(TET_IF_JPF):* image extracted as extended JPEG 2000 *(.jpf)*
- 33 *(TET_IF_J2K):* image extracted as raw JPEG 2000 *code stream (.j2k)*
- 50 *(TET_IF_JBIG2):* image extracted as JBIG2 *(.jbig2)*

The formats *TET_IF_JP2*, *TET_IF_JPF* and *TET_IF_J2K* are only created if *allowjpeg2000=true;* otherwise *TET_IF_TIFF* is created.

Details This method converts the pixel data for the image with the specified pCOS ID to one of several image formats and writes the result to a disk file. If the *typeonly* option has been supplied, only the image type is returned, but no image file is generated.

Bindings C/C++: macros for the return values are available in *tetlib.h*.

Table 10.20 Options for TET_write_image_file( ) and TET_get_image_data( )

option	description
compression	*(Keyword; deprecated)*
dpi	*(List of one or two non-negative float values) One or two values specifying the image resolution in pixels per inch in horizontal and vertical direction. If a single value is supplied it is used for both dimensions. The supplied values are recorded in generated TIFF images. They don't change the number of pixels in the image (i.e. no downsampling). See »Image resolution«, page 126, for details about determining image resolution. If one or two values are zero no resolution entry is written. Default: 72*
filename[1]	*(String; required unless* `typeonly` *is supplied) The name of the image file on disk. A suffix is added to the filename to indicate the image file format.* *The file name conventions used by the TET command-line tool with the option* `--imageloop resource` *match those in TETML (see »Image file names«, page 21). It is recommended to use the same file name patterns if the extracted images are used together with TETML.*
keepiccprofile	*(Boolean) If* `true` *and an ICC profile is assigned to the image, the ICC profile is embedded in extracted TIFF and JPEG images. Setting this option to* `false` *may result in smaller image files, but sacrifices color management. Default:* `true`
keepxmp	*(Boolean) If* `true` *and the image has associated XMP metadata in the PDF, the metadata is embedded in extracted TIFF and JPEG images. Default:* `true`
preferredtiff-compression	*(Keyword; deprecated)*
typeonly[1]	*(Boolean) The image type is determined according to the supplied options, but no image file is written. This is useful for determining the type of images returned by TET_get_image_data( ), which does not return the image type itself. Default:* `false`
validatejpeg	*(Boolean) If* `true`*, extracted JPEG images are validated to ensure correct image output. If* `false`*, processing is slightly faster, but invalid JPEG data is copied unmodified to the generated image file. Default:* `true`

1. Only for TET_write_image_file( )

C++ ***const char *get_image_data(int doc, size_t *length, int imageid, String 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)***

C ***const char * TET_get_image_data(TET *tet, int doc, size_t *length, int imageid, const char *optlist)***

Write image data to memory.

doc A valid document handle obtained with *TET_open_document*( )*.

length (C, C++ and RPG language bindings only) C-style pointer to a memory location where the length of the returned data in bytes is stored.

imageid The pCOS ID of the image. This ID can be retrieved from the *imageid* field after a successful call to *TET_get_image_info( )*, or by looping over all entries in the *images* pCOS array (there are *length:images* entries in this array).

optlist Option list specifying image-related options according to Table 10.20. The following options can be used: *keepxmp, dpi*

Returns The data representing the image according to the specified options. In case of an error a NULL pointer is returned in C and C++, and empty data in other language bindings. If an error occurs it is recommended to call *TET_get_errmsg( )* to find out more details about the error.

Details This method converts the pixel data for the image with the specified pCOS ID to one of several image formats, and makes the data available in memory.

Bindings C and C++ language bindings: The returned data buffer can be used until the next call to this method.

10.7 TET Markup Language (TETML)

API methods in this section:

- *TET_process_page( )*
- *TET_get_tetml( )*

C++ Java C# ***int process_page(int doc, int pagenumber, String optlist)***

Perl PHP ***long process_page(long doc, long pagenumber, string optlist)***

C ***int TET_process_page(TET *tet, int doc, int pagenumber, const char *optlist)***

Process a page and create TETML output.

doc A valid document handle obtained with *TET_open_document*( )*.

pagenumber The physical number of the page to be processed. The first page has page number 1. The total number of pages can be retrieved with *TET_pcos_get_number( )* and the pCOS path *length:pages*. The *pagenumber* parameter may be 0 if *trailer=true*.

optlist Option list specifying options from the following groups:

- General page-related options according to Table 10.10 (these are ignored if *pagenumber=0*):
  clippingarea, contentanalysis, excludebox, fontsizerange, granularity, ignoreinvisibletext, imageanalysis, includebox, layoutanalysis
- Option specifying TETML details according to Table 10.21: *tetml*

Table 10.21 Additional options for TET_process_page( )

option	*description*
tetml	*(Option list) Controls details of TETML. The following options are available:*
	elements *(Option list) Specify optional TETML elements:*
	line *(Boolean; only for* `granularity=word`*) If* `true`*, TETML output includes* `Line` *elements between* `Para` *and* `Word` *levels. Default:* `false`
	glyphdetails *(Option list; only for granularity=*`glyph` *and* `word`*) Specify which attributes are reported for each* `Glyph` *element (default for all suboptions:* `false`*):*
	all *(Boolean) Enable all attribute suboptions.*
	dehyphenation *(Boolean) Emit attribute* `dehyphenation` *to indicate hyphenated words.*
	dropcap *(Boolean) Emit attribute* `dropcap` *to indicate large initial characters at the start of a paragraph.*
	font *(Boolean) Emit attributes* `font`, `size`, `textrendering`, `unknown`.
	geometry *(Boolean) Emit attributes* `x`, `y`, `width`, `alpha`, `beta`.
	sub *(Boolean) Emit attribute* `sub` *to indicate subscripts.*
	sup *(Boolean) Emit attribute* `sup` *to indicate superscripts.*
	shadow *(Boolean) Emit attribute* `shadow` *to indicate shadow or simulated bold text.*
	textcolor *(Boolean) Emit attributes* `fill` *and* `stroke` *for the glyph colors (subject to* `textrendering`*) and corresponding* `Color` *elements.*
	trailer *(Boolean) If* `true`*, document trailer data, i.e. data after the last page, is emitted (it must be appended to the page-specific data emitted earlier). This option is required in the last call to this method in order to emit trailer data. If* `pagenumber=0` *only trailer data (without any page-specific data) is emitted. Once* `trailer=true` *has been supplied, no more calls to TET_process_page( ) are allowed for the same document. Default:* `false`

Returns This method always returns 1. PDF problems are reported in a TETML *Exception* element. Problems related to option list parsing trigger an exception.

Details This method opens a page, creates TETML output according to the format-related options supplied to *TET_open_document*()*, and closes the page. The generated data can be retrieved with *TET_get_tetml()*.

This method must only be called if the option *tetml* has been supplied in the corresponding call to *TET_open_document*()*. Header data, i.e. document-specific data before the first page, is created by *TET_open_document*()* before the first page data. It can be retrieved separately by calling *TET_get_tetml()* before the first call to *TET_process_page()*, or in combination with page-related data.

Trailer data, i.e. document-specific data after the last page, must be requested with the *trailer* suboption when this method is called for the last time for a document. Trailer data can be created with a separate call after the last page *(pagenumber=0)*, or together with the last page *(pagenumber* is different from 0). Pages can be retrieved in any order, and any subset of the document's pages can be retrieved.

It is an error to call *TET_close_document()* without retrieving the trailer, or to call *TET_process_page()* again after retrieving the trailer.

C++ ***const char *get_tetml(int doc, size_t *length, String optlist)***

C# Java ***final byte[ ] get_tetml(int doc, String optlist)***

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

C ***const char * TET_get_tetml(TET *tet, int doc, size_t *len, const char *optlist)***

Retrieve TETML data from memory.

doc A valid document handle obtained with *TET_open_document*()*.

len (C, C++ and RPG language bindings only) A pointer to a variable which will hold the length of the returned string in bytes. *len* does not count the terminating null byte.

optlist (Currently there are no supported options.)

Returns A byte array containing the next chunk of data. If the buffer is empty an empty string or *null* is returned (in C: a NULL pointer and **len=0*).

Details This method retrieves TETML data which has been created by *TET_open_document*()* and one or more calls to *TET_process_page()*. The TETML data is encoded in UTF-8, regardless of the *outputformat* option. The internal buffer is cleared by this call. It is not required to call *TET_get_tetml()* after each call to *TET_process_page()*. The client may accumulate the data for one or more pages or for the whole document in the buffer.

In TETML mode this method must be called at least once before *TET_close_document()* since otherwise the data would no longer be accessible. If *TET_get_tetml()* is called exactly once (such a single call must happen between the last call to *TET_process_page()* and *TET_close_document()*) the buffer is guaranteed to contain well-formed TETML output for the whole document.

This method must not be called if the *filename* suboption has been supplied to the *tetml* option of *TET_open_document*()*.

Bindings C and C++ language bindings: the result is provided as null-terminated UTF-8. On IBM System i and IBM Z EBCDIC-encoded UTF-8 is returned. The returned data buffer can be used until the next call to *TET_get_tetml( )*.

Java and .NET language bindings: the result is provided as a byte array containing UTF-8 data.

PHP language binding: the result is provided as UTF-8 string.

Python: the result is returned as 8-bit string (Python 3: *bytes)*.

RPG language binding: the result is returned as null-terminated UTF-8.

10.8 pCOS Methods

API methods in this section:

- *TET_pcos_get_number( )*
- *TET_pcos_get_string( )*
- *TET_pcos_get_stream( )*

The full pCOS syntax for retrieving object data from a PDF is supported. For a detailed description please refer to the pCOS Path Reference which is available as a separate document.

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

Perl PHP ***float pcos_get_number(int doc, string path)***

C ***double TET_pcos_get_number(TET *tet, int doc, const char *path, ...)***

Get the value of a pCOS path with type *number* or *boolean*.

doc A valid document handle obtained with *TET_open_document*( )*.

path A full pCOS path for a numerical or boolean object.

Additional parameters (C language binding only) A variable number of additional parameters can be supplied if the *key* parameter contains corresponding placeholders *(%s* for strings or *%d* for integers; use *%%* for a single percent sign). Using these parameters will save you from explicitly formatting complex paths containing variable numerical or string values. The client is responsible for making sure that the number and type of the placeholders matches the supplied additional parameters.

Returns The numerical value of the object identified by the pCOS path. For Boolean values 1 is returned if they are *true*, and 0 otherwise.

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

Perl PHP ***string pcos_get_string(int doc, string path)***

C ***const char *TET_pcos_get_string(TET *tet, int doc, const char *path, ...)***

Get the value of a pCOS path with type *name, number, string,* or *boolean*.

doc A valid document handle obtained with *TET_open_document*( )*.

path A full pCOS path for a string, name, or boolean object.

Additional parameters (C language binding only) A variable number of additional parameters can be supplied if the *key* parameter contains corresponding placeholders *(%s* for strings or *%d* for integers; use *%%* for a single percent sign). Using these parameters will save you from explicitly formatting complex paths containing variable numerical or string values. The client is responsible for making sure that the number and type of the placeholders matches the supplied additional parameters.

Returns A string with the value of the object identified by the pCOS path. For Boolean values the strings *true* or *false* is returned.

Details This method raises an exception if pCOS does not run in full mode and the type of the object is *string*. However, the objects */Info/** (document info keys) can also be retrieved in

restricted pCOS mode if *nocopy=false* or *plainmetadata=true*, and *bookmarks[...]/Title* as well as all paths starting with *pages[...]/annots[...]/* can be retrieved in restricted pCOS mode if *nocopy=false*.

This method assumes that strings retrieved from the PDF document are text strings. String objects which contain binary data should be retrieved with *TET_pcos_get_stream( )* instead which does not modify the data in any way.

Bindings C language binding: The string is returned in UTF-8 format (on IBM System i and IBM Z: EBCDIC-UTF-8) without BOM. The returned strings are stored in a ring buffer with up to 10 entries. If more than 10 strings are queried, buffers are reused, which means that clients must copy the strings if they want to access more than 10 strings in parallel. For example, up to 10 calls to this method can be used as parameters for a *printf( )* statement since the return strings are guaranteed to be independent if no more than 10 strings are used at the same time.

C++ language binding: the result is provided as Unicode in the selected string format (see Section 3.2, »C++ Binding«, page 28).

Java and .NET bindings: the result are provided as Unicode string. If no more text is available a null object is returned.
Perl, PHP, Python and Ruby language bindings: the result is provided as UTF-8 string. If no more text is available a null object is returned.

C++ ***const unsigned char *pcos_get_stream(int doc, int *len, string optlist, String 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)***

C ***const unsigned char *TET_pcos_get_stream(TET *tet, int doc, int *len, const char *optlist, const char *path, ...)***

Get the contents of a pCOS path with type *stream*, *fstream*, or *string*.

doc A valid document handle obtained with *TET_open_document*( )*.

len (C, C++ and RPG language bindings only) A pointer to a variable which will receive the length of the returned stream data in bytes.

optlist Option list specifying stream retrieval options according to Table 10.22.

path A full pCOS path for a stream or string object.

Additional parameters (C language binding only) A variable number of additional parameters can be supplied if the *key* parameter contains corresponding placeholders *(%s* for strings or *%d* for integers; use *%%* for a single percent sign). Using these parameters will save you from explicitly formatting complex paths containing variable numerical or string values. The client is responsible for making sure that the number and type of the placeholders matches the supplied additional parameters.

Returns The unencrypted data contained in the stream or string. The returned data is empty (in C and C++: NULL) if the stream or string is empty, or if the contents of encrypted attachments in an unencrypted document are queried and the attachment password has not been supplied.

If the object has type *stream* all filters are removed from the stream contents (i.e. the actual raw data is returned) unless *keepfilter=true*. If the object has type *fstream* or *string* the data is delivered exactly as found in the PDF file, with the exception of ASCII85 and ASCIIHex filters which are removed.

In addition to decompressing the data and removing ASCII filters, text conversion may be applied according to the *convert* option.

JPX-compressed streams are treated as follows: image data with 1..8 bits per component is returned with 8 bits per component; Image data with 9..16 bits per component is returned with 16 bits per component. If no PDF color space is present and the JPX-compressed stream contains an internal color palette, the palette is applied before returning the uncompressed stream data to ensure that the pixel data matches the reported color space and number of components. Note that the palette is not applied if the PDF color space *Indexed* is present.

Details This method throws an exception if pCOS does not run in full mode (see the pCOS Path Reference). As an exception, the object */Root/Metadata* can also be retrieved in restricted pCOS mode if *nocopy=false* or *plainmetadata=true*. An exception will also be thrown if *path* does not point to an object of type *stream*, *fstream*, or *string*.

Despite its name this method can also be used to retrieve objects of type *string*. Unlike *TET_pcos_get_string( )*, which treats the object as a text string, this method does not modify the returned data Binary string data is rarely used in PDF, and cannot be detected automatically. The user is therefore responsible for selecting the appropriate method for retrieving string objects as binary data or text.

Bindings

Bindings C language binding: If *convert=unicode* is supplied, the string is returned in UTF-8 format without BOM (on IBM System i and IBM Z: EBCDIC-UTF8).

C and C++ language bindings: The returned data buffer can be used until the next call to this method.

Note Python: the result is returned as 8-bit string (Python 3: *bytes)*.

Table 10.22 Options for TET_pcos_get_stream( )

option	*description*	
convert	*(Keyword; ignored for streams which are compressed with unsupported filters) Controls whether or not the string or stream contents are converted (default:* none*)* **:**	
	none	*Treat the contents as binary data without any conversion.*
	unicode	*Treat the contents as textual data (i.e. exactly as in TET_pcos_get_string( )), and normalize it to Unicode. In non-Unicode-aware language bindings this means the data is converted to UTF-8 format without BOM.* *This option is required for the data type »text stream« in PDF which is rarely used (e.g. it can be used for JavaScript, although the majority of JavaScripts is contained in string objects, not stream objects).*
keepfilter	*(Boolean; recommended only for image data streams; ignored for streams which are compressed with unsupported filters) If* `true`*, the stream data remains compressed with the filter which is specified in the image's* `filterinfo` *pseudo object (see the pCOS Path Reference). If* `false`*, the stream data is uncompressed. Default:* `true` *for all unsupported filters,* `false` *otherwise*	

A TET Library Quick Reference

The following tables contain an overview of all TET API functions. The prefix *(C)* denotes C prototypes of functions which are not available in the Java language binding.

Setup

Function prototype	*page*
*(C) TET *TET_new(void)*	*170*
void delete( )	*170*

Option Handling

Function prototype	*page*
void set_option(String optlist)	*167*

PDFlib Virtual Filesystem (PVF)

Function prototype	*page*
void create_pvf(String filename, byte[] data, String optlist)	*171*
int delete_pvf(String filename)	*171*
int info_pvf(String filename, String keyword)	*172*

Unicode Conversion

Function prototype	*page*
String convert_to_unicode(String inputformat, byte[ ] input, String optlist)	*173*

Exception Handling

Function prototype	*page*
String get_apiname( )	*175*
String get_errmsg( )	*176*
int get_errnum( )	*176*

Document

Function prototype	*page*
int open_document(String filename, String optlist)	*179*
*(C/C++) int TET_open_document_callback(TET *tet, void *opaque, tet_off_t filesize, size_t (*readproc)(void *opaque, void *buffer, size_t size), int (*seekproc)(void *opaque, tet_off_t offset), const char *optlist)*	*186*
STET_Cthe value 1parameterthe opaque pointer passed to interruptvoid close_document(int doc)	*187*

Page

Function prototype	*page*
int open_page(int doc, int pagenumber, String optlist)	*188*
void close_page(int page)	*197*

Text and Glyph Details Retrieval

Function prototype	page
String get_text(int page)	198
int get_char_info(int page)	199
int get_color_info(int doc, int colorid, String optlist)	203

Image Retrieval

Function prototype	page
int get_image_info(int page)	204
int write_image_file(int doc, int imageid, String optlist)	205
final byte[ ] get_image_data(int doc, int imageid, String optlist)	207

TET Markup Language (TETML)

Function prototype	page
int process_page(int doc, int pagenumber, String optlist)	208
final byte[ ] get_tetml(int doc, String optlist)	209

pCOS

Function prototype	page
double pcos_get_number(int doc, String path)	211
String pcos_get_string(int doc, String path)	211
final byte[ ] pcos_get_stream(int doc, String optlist, String path)	212

B Revision History

Revision history of this manual

Date	Changes
December 10, 2024	► Updates for TET 5.6
December 14, 2023	► Updates for TET 5.5
December 6, 2022	► Updates for TET 5.4
April 26, 2021	► Updates for TET 5.3
October 17, 2019	► i5-related clarifications for TET 5.2p1
July 18, 2019	► Updates for TET 5.2
May 24, 2017	► Updates for TET 5.1
November 03, 2015	► Updates for TET 5.0
January 27, 2015	► Updates for TET 4.4
May 26, 2014	► Updates for TET 4.3
May 17, 2013	► Updates for TET 4.2
April 04, 2012	► Updates for TET 4.1p1
February 20, 2012	► Updates for TET 4.1
September 22, 2010	► Updates for TET 4.0p2
July 27, 2010	► Updates for TET 4.0
February 01, 2009	► Updates for TET 3.0
January 16, 2008	► Updated the manual for TET 2.3
January 23, 2007	► Minor additions for TET 2.2
December 14, 2005	► Additions and corrections for TET 2.1.0; added descriptions for the PHP and RPG language bindings
June 20, 2005	► Expanded and reorganized the manual for TET 2.0.0
October 14, 2003	► Updated the manual for TET 1.1
November 23, 2002	► Added the description of TET_open_doc_callback( ) and a code sample for determining the page size for TET 1.0.2
April 4, 2002	► First edition for TET 1

Index

A

annotations 71
API reference 159
Arabic 84
area of text extraction 74
Artifacts 101
Artifacts in Tagged PDF 190
    TET_char_info structure 201
    TET_image_info structure 205
ascender 77
attachment password 59

B

bidirectional text 84
BMP (Basic Multilingual Plane) 97, 112
bookmarks 71
Boolean values in option lists 163
Byte Order Mark (BOM) 98

C

C binding 25
C++ binding 28
canonical decomposition 106
capheight 77
categories of resources 61
characters and glyphs 97
CJK (Chinese, Japanese, Korean) 12, 82
    compatibility forms 83
    configuration 7
    word boundaries 82
codelist 115
color of text 80
color spaces 130
command-line tool 19
comments 71
commercial license 10
compatibility decomposition 106
composite characters 99
concordance (XSLT sample) 155
connector 45
content analysis 86
coordinate system 74
CSV format 157

D

decomposition 106
dehyphenation 88
descender 77
DeviceN color space 130
Dispose( ) 170
document and page functions 179
document domains 69
document info entries 69
document styles 89
double-byte variants 83

E

end points of glyphs and words 78
EUDC fonts 114
evaluation version 7
examples
    text extraction status 59
    XSLT 155
exception handling
    in C 25

F

fake bold removal 88
file attachments 72
file search 62
fill color of text 80
float and integer values in option lists 164
folding 103
font filtering (XSLT sample) 155
font statistics (XSLT sample) 156
FontReporter plugin 12, 114
form fields 71
fullwidth variants 83

G

geometry of images 125
glyph metrics 75
glyph rules 117
glyphlist 116
glyphs 97
granularity 86

H

halfwidth variants 83
Hebrew 84
highlighting 78
HTML converter (XSLT sample) 157

I

ICC profiles 130

ideographic text: word boundaries 82
IFilter for Microsoft products 53
images
  color fidelity 130
  determining type 120
  extract to disk or memory 119
  extracting 119
  formats 119
  geometry 125
  merging 127
  number of images in a document 122
  page-based extraction loop 123
  placed images 122
  resolution 126
  resource-based extraction loop 124
  resources 122
  small image removal 129
  XMP metadata 120
inch 74
index (XSLT sample) 157
installing TET 7
invisible text 202

J

Java binding 30
Javadoc 30
JBIG2 119
JPEG 119
JPEG 2000 119

K

keywords in option lists 164

L

layers 73, 190
license key 8
ligatures 99
list detection 93
list values in option lists 160
logging 177
logo fonts 114
Lucene search engine 46

M

master password 59
MediaWiki 57
millimeters 74

N

nested option lists 160
.NET binding 32
normalization 110
numbers in option lists 164

O

Objective-C binding 35
optimizing performance 65
option list syntax 159
option lists 159
Oracle Text 50
outline text 202
owner password 59

P

packages 72
page boxes 74
page-based image extraction loop 123
passwords 59
pCOS
  API methods 211
  Cookbook 15
PDF versions 11
performance optimization 65
Perl binding 37
permissions password 59
PHP binding 38
placed images 122
points 74
portfolios 72
preprocessing text 100
prerotated glyphs 83
programming samples 14
protected documents 59
PUA (Private Use Area) 98, 104, 114
Python Binding 40

R

raw text extraction (XSLT sample) 158
rectangles in option lists 164
resource configuration 61
resource-based image extraction loop 124
resourcefile parameter 63
response file 22
roadmap to documentation and samples 14
RPG binding 43
Ruby binding 41

S

sample code 14
schema 143
searching for font usage (XSLT sample) 156
searchpath 62
separation color space 130
sequences 99
shadow removal 88
shrug feature 59
single-byte variants 83
small image removal 129
Solr search server 49
spot color 130

strings in option lists 161
stroke color of text 80
supplementary characters 112
surrogates 98, 112
syntax of option lists 159

T

table detection 92
table extraction (XSLT sample) 157
Tagged PDF 72, 101
TET command-line tool 19
TET connector 45
  for Lucene 46
  for MediaWiki 57
  for Microsoft products 53
  for Oracle 50
  for Solr 49
  for TIKA 55
TET Cookbook 15
TET features 11
TET Markup Language (TETML) 133
TET plugin for Adobe Acrobat 45
TET_CATCH( ) 176
TET_close_document( ) 187
TET_close_page( ) 197
TET_convert_to_unicode( ) 173
TET_create_pvf() 171
TET_delete_pvf() 171
TET_delete( ) 170
TET_EXIT_TRY( ) 25, 176
TET_get_apiname() 175
TET_get_char_info( ) 199
TET_get_color_info( ) 203
TET_get_errmsg( ) 176
TET_get_errnum( ) 176
TET_get_image_data( ) 207
TET_get_image_info( ) 204
TET_get_tetml( ) 209
TET_get_text( ) 198
TET_info_pvf() 172
TET_new( ) 170
TET_open_document_callback( ) 186
TET_open_document( ) 179
TET_open_page( ) 188
TET_pcos_get_number( ) 211
TET_pcos_get_stream( ) 212
TET_pcos_get_string( ) 211
TET_RETHROW( ) 176
TET_set_option( ) 167
TET_TRY( ) 176
TET_write_image_file( ) 205
tet.upr 63
TETML 133
  schema 143
TETRESOURCEFILE environment variable 63
TeX documents 67
text color 80
text extraction status 59
text filtering 100
TIFF 119
TIKA toolkit 55

U

Unichar values in option lists 162
Unicode
  BOM 98
  concepts 97
  decomposition 106
  encoding forms 98
  encoding schemes 98
  folding 103
  in option lists 162
  normalization 110
  postprocessing 103
  sets 163
Unicode-capable language bindings 165
units 74
unmappable glyphs 113
Unquoted string values in option lists 162
UPR file format 61
user password 59
UTF formats 98
UTF-32 112

V

vertical writing mode 82

W

word boundary detection 87
Wordfinder 87

X

XFA forms 11, 148
xheight 77
XMP metadata 70
  for images 120
  XSLT sample 157
XSD schema for TETML 143
XSLT 152
  samples 14, 155

PDFlib GmbH

PDFlib GmbH
Franziska-Bilek-Weg 9
80339 München, Germany
www.pdflib.com

Licensing contact
sales@pdflib.com

Support
PDFlib_support@pdftron.com *(please include your license number)*