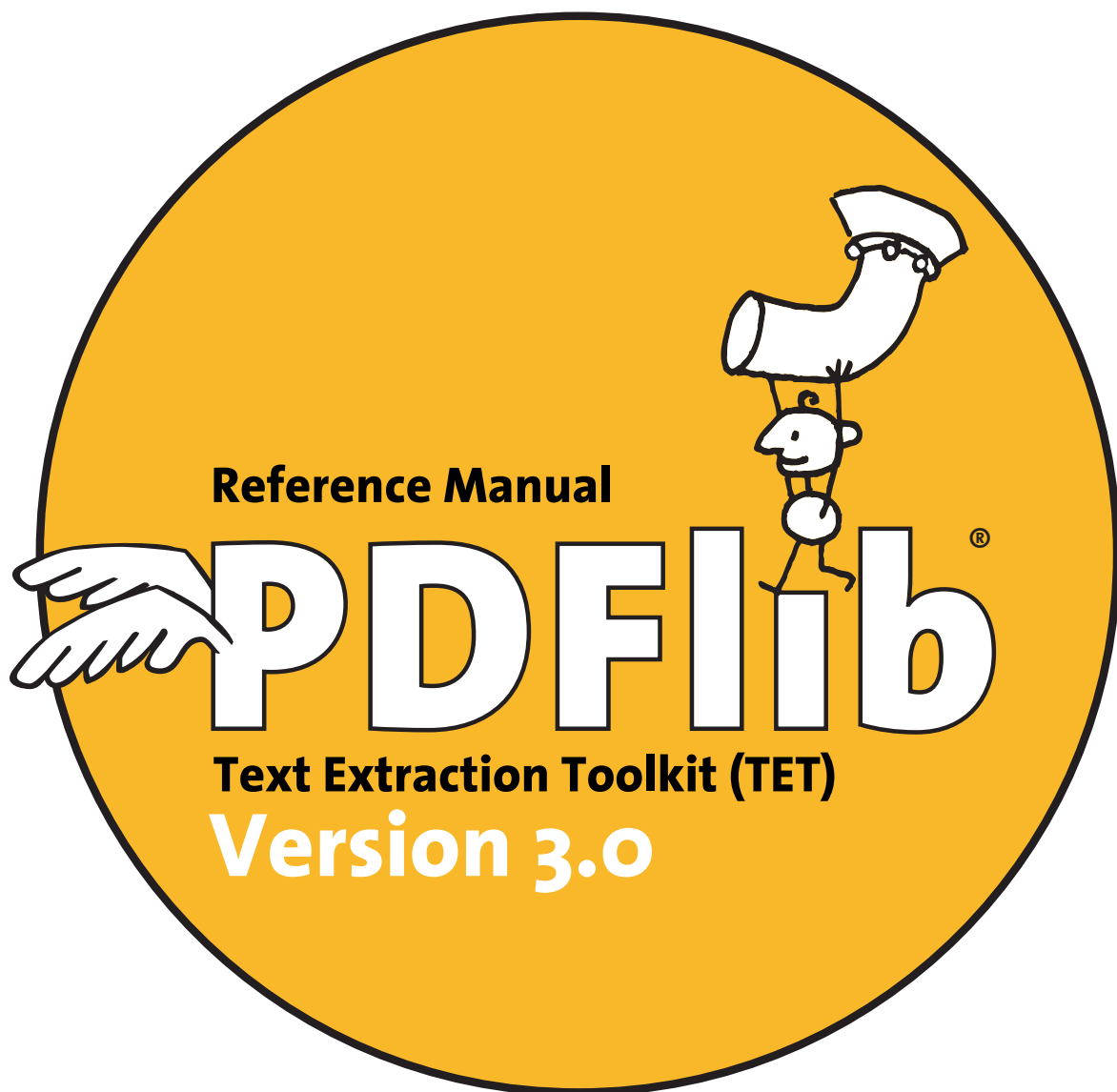


PDFlib GmbH München, Germany

www.pdflib.com



Copyright © 2002-2009 PDFlib GmbH. All rights reserved.
Protected by European patents.
Patents pending in the U.S. and other countries.

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

phone +49 • 89 • 452 33 84-0
fax +49 • 89 • 452 33 84-99

If you have questions check the PDFlib mailing list and archive at tech.groups.yahoo.com/group/pdflib

Licensing contact: sales@pdflib.com
Technical support: support@pdflib.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.

PDFlib and the PDFlib logo are registered trademarks of PDFlib GmbH. PDFlib licensees are granted the right to use the PDFlib name and logo in their product documentation. However, this is not required.

Adobe, Acrobat, and PostScript are trademarks of Adobe Systems Inc. AIX, IBM, OS/390, WebSphere, iSeries, and zSeries are trademarks of International Business Machines Corporation. ActiveX, Microsoft, Windows, and Windows NT are trademarks of Microsoft Corporation. Apple, Macintosh and TrueType are trademarks of Apple Computer, Inc. Unicode and the Unicode logo are trademarks of Unicode, Inc. Unix is a trademark of The Open Group. Java and Solaris are a trademark of Sun Microsystems, Inc. Other company product and service names may be trademarks or service marks of others.

*PDFlib TET contains parts of the following third-party software:
Zlib compression library, Copyright © 1995-2002 Jean-loup Gailly and Mark Adler
TIFFlib image library, Copyright © 1988-1997 Sam Leffler, Copyright © 1991-1997 Silicon Graphics, Inc.
Cryptographic software written by Eric Young, Copyright © 1995-1998 Eric Young (ey@cryptsoft.com)
Independent JPEG Group's JPEG software, Copyright © 1991-1998, Thomas G. Lane
Cryptographic software, Copyright © 1998-2002 The OpenSSL Project (www.openssl.org)
Expat XML parser, Copyright © 1998, 1999, 2000 Thai Open Source Software Center Ltd*

The PDFlib Text Extraction Toolkit contains the RSA Security, Inc. MD5 message digest algorithm.



Contents

o First Steps with TET 7

- o.1 Installing the Software 7
- o.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 13

2 TET Command-Line Tool 15

- 2.1 Command-Line Options 15
- 2.2 Command-line Examples 18

3 TET Library Language Bindings 21

- 3.1 Exception Handling 21
- 3.2 C Binding 22
- 3.3 C++ Binding 24
- 3.4 COM Binding 25
- 3.5 Java Binding 26
- 3.6 .NET Binding 27
- 3.7 Perl Binding 28
- 3.8 PHP Binding 29
- 3.9 Python Binding 31
- 3.10 RPG Binding 32

4 TET Connectors 35

- 4.1 Free TET Plugin for Adobe Acrobat 35
- 4.2 TET Connector for the Lucene Search Engine 37
- 4.3 TET Connector for the Solr Search Server 40
- 4.4 TET Connector for Oracle 41
- 4.5 TET PDF IFilter for Microsoft Products 44
- 4.6 TET Connector for MediaWiki 46

5 Configuration 49

- 5.1 Indexing protected PDF Documents 49
- 5.2 Resource Configuration and File Searching 51
- 5.3 Recommendations for common Scenarios 54

6 Text Extraction 57

- 6.1 Document Domains 57
- 6.2 Unicode Concepts 61
- 6.3 Page and Text Geometry 64
- 6.4 Support for Chinese, Japanese, and Korean Text 67
- 6.5 Unicode Pipeline 68
- 6.6 Content Analysis 71
- 6.7 Layout Analysis 74
- 6.8 Advanced Unicode Mapping Controls 76

7 Image Extraction 81

- 7.1 Image Extraction Basics 81
- 7.2 Image Geometry 83
- 7.3 Image Analysis 85
- 7.4 Restrictions and Caveats 87

8 TET Markup Language (TETML) 89

- 8.1 Creating TETML 89
- 8.2 Controlling TETML Details 92
- 8.3 TETML Elements and the TETML Schema 96
- 8.4 Transforming TETML with XSLT 97
- 8.5 XSLT Samples 101

9 The pCOS Interface 105

- 9.1 Simple pCOS Examples 105
- 9.2 Handling Basic PDF Data Types 107
- 9.3 Composite Data Structures and IDs 109
- 9.4 Path Syntax 110
- 9.5 Pseudo Objects 112
- 9.6 Encrypted PDF Documents 119

10 TET Library API Reference 121

- 10.1 Option Lists 121
- 10.2 General Functions 123
- 10.3 Exception Handling 127
- 10.4 Document Functions 129
- 10.5 Page Functions 134
- 10.6 Text and Metrics Retrieval Functions 142
- 10.7 Image Retrieval Functions 145
- 10.8 TET Markup Language (TETML) Functions 149

10.9 Option Handling 151

10.10 pCOS Functions 153

A TET Library Quick Reference 157

B Revision History 159

Index 161

o First Steps with TET

o.1 Installing the Software

TET is delivered as an MSI installer package for Windows systems, 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 15, 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 21, corresponding to their preferred development environment, and review the installed examples. On Windows, the TET programming examples are accessible via the Start menu.

If you obtained a commercial TET license you must enter your TET license key according to Section o.2, »Applying the TET License Key«, page 8.

CJK configuration. In order to extract Chinese, Japanese, or Korean (CJK) text TET generally requires the corresponding CMap files for mapping CJK encodings to Unicode. The CMap files are contained in all TET packages, and will be installed in the *resource/cmap* directory within the TET installation directory. On Windows systems simply choose the full installation option when installing TET. The CMap files will be found automatically via the registry.

On other 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(tet, "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.

Glyph list configuration for IBM iSeries. On IBM eServer iSeries (but not any other system) the glyph lists in the directory *resource/glyphlst* must be available to TET. Access to these tables is automatically configured if TET is installed in the standard directory.

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.

o.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.

Entering the license key in the Windows installer. Windows users can enter the license key when they install TET using the supplied installer. This is the recommended method on Windows. If you do not have write access to the registry or cannot use the installer, refer to one of the alternate methods below instead.

Entering the license key in a license file. Set an environment (shell) variable which points to a license file before TET functions are called. If you are using the TET library you can alternatively set the path to the license file by setting the *licensefile* parameter with the *TET_set_option()* function. The license file must be a text file according to the sample below; you can use the license file template *licensekeys.txt* which is contained in all TET distributions. Lines beginning with a '#' characters contain comments, and will be ignored; the second line contains version information for the license file itself:

```
# Licensing information for PDFlib GmbH products
PDFlib license file 1.0
TET 3.0 ...your license key...
```

The details of setting environment variables vary across systems, but a typical statement for a Unix shell looks as follows:

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

On IBM eServer iSeries the license file can be specified as follows:

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

This command can be specified in the startup program *QSTRUP* and will work for all PDFlib GmbH products.

Multi-system license files on iSeries and zSeries. License keys for iSeries and zSeries are system-specific and therefore cannot be shared among multiple systems. In order to facilitate resource sharing and work with a single license file which can be shared by multiple systems, the following license file format can be used to hold multiple system-specific keys in a single file:

```
PDFlib license file 2.0
# Licensing information for PDFlib GmbH products
TET      3.0      ...your license key...      ...CPU ID1...
TET      3.0      ...your license key...      ...CPU ID2...
```

Note the changed version number in the first line and the presence of multiple license keys, followed by the corresponding CPU ID.

Set 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 --teto pt "license ...your license key..." ...more options...
```

```
tet --teto pt "licensefile /path/to/your/license/file" ...more options...
```

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

```
tet --teto pt "licensefile {/path/to/license file}" ...more options...
```

Setting the license key with a TET library call. If you use the TET library, add a line to your script or program which sets the license key at runtime:

- In COM/VBScript:

```
oTET.set_option "license=...your license key..."
```

- In C:

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

- In C++, .NET/C#, Java, and PHP 5 with object-oriented interface:

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

- In and PHP with function-based interface:

```
TET_set_option(tet, "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 (in C++, COM, .NET, Java, and PHP).

1 Introduction

The PDFlib Text 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 for repurposing
- ▶ extract metadata and other information from PDF

TET has been designed for standalone 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 thousands of PDF test files from various sources. It accepts PDF 1.0 up to PDF 1.7 extension level 3 (corresponding to Acrobat 1-9) as well as encrypted documents.

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

- ▶ TET converts all text contents to Unicode. In C the text will be returned in UTF-8 or UTF-16 format; in other language bindings as native Unicode strings.
- ▶ Ligatures and other multi-character glyphs will be decomposed into a sequence of their constituent Unicode characters.
- ▶ Vendor-specific Unicode values (Corporate Use Subarea, CUS) are identified, and will be mapped to characters with precisely defined meanings if possible.
- ▶ Glyphs which are lacking Unicode mapping information are identified as such, and will be mapped to a configurable replacement character.
- ▶ UTF-16 surrogate pairs for characters outside the Basic Multilingual Plane (BMP) are properly 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 will be converted to Unicode. The CMap files for CJK encoding conversion are included in the TET distribution.
- ▶ Horizontal and vertical writing modes are supported.
- ▶ CJK font names will be normalized to Unicode.

Image extraction. TET extracts raster images from PDF. Adjacent parts of a segmented image will be recombined 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.

Images will be extracted in the common TIFF, JPEG, or JPEG 2000 formats.

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.

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

pCOS interface for simple access to PDF objects. TET includes pCOS (*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.

TET Markup Language (TETML). The information retrieved from a PDF document can be presented in an XML format called TET Markup Language, or TETML for processing with standard XML tools. TETML contains text, image, and metadata information and can optionally also contain font- and geometry-related details.

What is text? While TET deals with a large class of PDF documents, not all visible text can successfully 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 directly 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 the pCOS interface. 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 or clipped by some other element on the page, e.g. an image.
- ▶ PDF layers are ignored; TET will retrieve the text from all layers regardless of their visibility.

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.

1.3 Roadmap to Documentation and Samples

Mini samples for the TET library. The TET distribution contains programming examples for all supported language bindings. These mini 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 loops for extracting text and images from a PDF document.
- ▶ 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 *tetml* sample contains the prototypical code for generating TETML (TET's XML language for expressing PDF contents) from a PDF document.
- ▶ The *get_attachments* sample (not available for all language bindings) demonstrates how to process PDF file attachments, i.e. PDF documents which are embedded in another PDF document.

Note On Windows Vista the mini samples will be installed in the »Program Files« directory by default. Due to a new protection scheme in Windows Vista the PDF output files created by these samples will only be visible under »compatibility files«. Recommended workaround: copy the examples to a user directory.

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 fields from document-level XMP metadata included in TETML.
- ▶ *solr.xsl*: generate input for the Solr enterprise search server.
- ▶ *table.xsl*: Extract a table to CSV file (comma-separated values).
- ▶ *tetml2html.xsl*: convert TETML to simple 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 Chapter 9, »The pCOS Interface«, page 105.

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 a number of command-line options. The program will insert 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`, `--teto`, `--imageopt`, and `--pageopt` options according to the option list tables in Chapter 10, »TET Library API Reference«, page 121. Table 2.1 lists all TET command-line options (this list will also be displayed if you run the TET program without any options).

Note In order to extract CJK text you must configure access to the CMap files which are shipped with TET according to 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 17. Response files will only be recognized before the -- option and before the first filename, and can not be used to replace the parameter for another option.
--docopt	<option list>	Additional option list for <code>TET_open_document()</code> (see Table 10.3, page 130). The filename suboption of the <code>tetml</code> option can not be used here.
--firstpage -f	<integer> last	The number of the page where content extraction will start. The keyword <code>last</code> specifies the last page, <code>last-1</code> 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
--help, -? (or no option)		Display help with a summary of available options.
--inmemory		Load the input file into memory and process it from there. This can result in a significant performance gain on some systems at the expense of memory usage.
--image ² -i		Extract images from the document. Extracted images will be placed in files according to the following naming scheme: <filename>_p<pagenumber>_<imagenumber>.[tif jpg jpx]
--imageopt	<option list>	Additional option list for <code>TET_write_image_file()</code> (see Table 10.12, page 147)
--lastpage -l	<integer> last	The number of the page where content extraction will finish. The keyword <code>last</code> specifies the last page, <code>last-1</code> the page before the last page, etc. Default: last

Table 2.1 TET command-line options

option	parameters	function
--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).
--pageopt	<option list>	Additional option list which will be supplied to TET_open_page() if text output is generated, or to TET_process_page() if TETML output is generated. See Table 10.5, page 134 and Table 10.13, page 149, for a list of available options. For text output the option granularity will always be set to page.
--password, -p	<password>	User or master 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, »Indexing protected PDF Documents«, page 49).
--searchpath¹ -s	<path>...	Name of one or more directories where files (e.g. CMaps) will be searched. Default: installation-specific
--targetdir -t	<dirname>	Output directory for generated text, TETML, and image files. The directory must exist. Default: . (i.e. the current working directory)
--tetml -m	glyph word wordplus line page	(Can not be combined with --text) Create TETML output according to the TET 3 schema containing text and image information. TETML output will always be created in UTF-8 encoding. The supplied parameter selects one of several variants (see Section 8.2, »Controlling TETML Details«, page 92, for more details): glyph Glyph-based TETML with glyph geometry and font details word Word-based TETML with word boxes wordplus Word-based TETML with word boxes plus glyph geometry and font details line Line-based TETML (text only) page Page-based TETML (text only)
--tetopt	<option list>	Additional option list for TET_set_option() (see Table 10.14, page 151). The option outputformat will be ignored (use --format instead).
--text²		(Can not be combined with --tetml) Extract text from the document (enabled by default)
--unique -u		(Only relevant for --image) Extract each image only once, even if it is placed in the document more than once (e.g. repeated headers).
--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.
--xm³ -x	glyph word word2 line zone page	(Deprecated; new applications should use --tetml instead) Create glyph-, word-, line-, zone-, or page-based XML output according to the deprecated TET 2 DTD containing the text and metrics information. The word2 mode is similar to word mode, but includes details for all the characters in a word.

1. This option can be supplied more than once.
2. The option --image disables text extraction, but it can be combined with --text or --tetml.
3. Deprecated, use --tetml instead.

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

- ▶ Input files will be 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 text 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 will be 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.

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 will be 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 will be 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.

Note On iSeries response files are expected in ASCII format.

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.

2.2 Command-line Examples

The following examples demonstrate some useful combinations of TET command-line options. The samples are shown in two variations; the first uses the long format of all options, while the second uses the equivalent short option format.

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
tet -f 2 -l 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
tet -s /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
tet --format utf16 -o 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
tet -t out in/*.pdf
```

Restrict text extraction to a particular area on the page; this can be achieved by supplying a suitable list of page options:

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

Extract images from *file.pdf* and store them in *file*.tif/file*.jpg* in the directory *out*:

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

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

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

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

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

Extract images and generate TETML in a single call:

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

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

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



3 TET Library Language Bindings

This chapter discusses specifics for the language bindings which are supplied for the TET library. The TET distribution contains full sample code for several small TET applications in all supported language bindings.

3.1 Exception Handling

Errors of a certain kind are called exceptions in many languages for good reasons – they are mere exceptions, and are not expected to occur very often during the lifetime of a program. The general strategy is to use conventional error reporting mechanisms (read: special error return codes) for function calls which may go wrong often times, and use a special 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, »Indexing protected PDF Documents«, page 49);
- ▶ 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 value of `-1` as documented in the API reference. Other events may be considered harmful, but will occur rather infrequently, e.g.

- ▶ running out of virtual memory;
- ▶ supplying wrong function 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 will be thrown instead of passing a special error return value to the caller. In languages which support native exceptions throwing the exception will be done using the standard means supplied by the language or environment. For the C language binding TET supplies a custom exception handling mechanism which must be used by clients (see Section 3.2, »C Binding«, page 22).

It is important to understand that processing a document must be stopped when an exception occurred. The only methods which can safely be called after an exception are `TET_delete()`, `TET_get_apiname()`, `TET_get_errnum()`, and `TET_get_errmsg()`. Calling any other method after an exception may lead to unexpected results. The exception will contain the following information:

- ▶ A unique error number;
- ▶ The name of the API function which caused the exception;
- ▶ A descriptive text containing details of the problem;

Querying the reason of a failed function call. Some TET function calls, e.g. `TET_open_document()` or `TET_open_page()`, can fail without throwing an exception (they will return `-1` in case of an error). In this situation the functions `TET_get_errnum()`, `TET_get_errmsg()`, and `TET_get_apiname()` can be called immediately after a failed function call in order to retrieve details about the nature of the problem.

3.2 C Binding

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 functions 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.
- ▶ As in all TET language bindings 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):

```
volatile int pageno;
...
if ((tet = TET_new()) == (TET *) 0)
{
    printf("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 functions */
}
```

```

TET_CATCH(tet)
{
    printf("Error %d in %s() on page %d: %s\n",
        TET_get_errnum(tet), TET_get_apiname(tet), pageno, TET_get_errmsg(tet));
}
TET_delete(tet);

```

Unicode handling for name strings. The C language does not natively support Unicode. Some string parameters for API functions 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 will be interpreted as UTF-16. If the *length* parameter is 0 the string will be 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 *host* encoding if no BOM is found (or *ebcdic* on all 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 will be 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 will be interpreted as UTF-8.
- ▶ If the option starts with an EBCDIC UTF-8 BOM (`\x57\x8B\xAB`) it will be interpreted as EBCDIC UTF-8.
- ▶ If no BOM is found, the string will be treated as *winansi* (or *ebcdic* on EBCDIC-based platforms).

Note The `TET_utf16_to_utf8()` utility function can be used to create UTF-8 strings from UTF-16 strings, which is useful for creating option lists with Unicode values.

3.3 C++ Binding

In addition to the *tetlib.h* C header file, an object-oriented wrapper for C++ is supplied for TET clients. It requires the *tet.hpp* header file, which in turn includes *tetlib.h*. The corresponding *tet.cpp* module must be linked against the application in addition to the generic TET C library.

Using the C++ object wrapper replaces the functional approach with API functions and *TET_* prefixes in all TET function names with a more object-oriented approach: a *TET* object offers methods, but the method names no longer have the *TET_* prefix.

The TET C++ binding will package Unicode text in standard C++ strings in UTF-16 format. Clients must be prepared to process such strings appropriately.

3.4 COM Binding

Installing the TET COM edition. TET can be deployed in all environments that support COM components. Installing TET is an easy and straight-forward process. Please note the following:

- ▶ If you install on an NTFS partition all TET users must have read permission for the installation directory, and execute permission for
...\\TET 3.0\\bind\\COM\\bin\\tet_com.dll.
- ▶ The installer must have write permission for the system registry. Administrator or Power Users group privileges will usually be sufficient.

Exception Handling. Exception handling for the TET COM component is done according to COM conventions: when a TET exception occurs, a COM exception will be raised and furnished with a clear-text description of the error. In addition the memory allocated by the TET object is released. The COM exception can be caught and handled in the TET client in whichever way the client environment supports for handling COM errors.

Using the TET COM Edition with .NET. As an alternative to the TET.NET edition (see Section 3.6, »*.NET Binding*«, page 27) the COM edition of TET can also be used with .NET. First, you must create a .NET assembly from the TET COM edition using the *tlbimp.exe* utility:

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

You can use this assembly within your .NET application. If you add a reference to *tet_com.dll* from within Visual Studio .NET an assembly will be created automatically. The following code fragment shows how to use the TET COM edition with C#:

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

All other code works as with the .NET edition of TET.

3.5 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 Mac OS X: *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 *pdf_tet.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 and contains a single class called *tet*. 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"));
```

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

```
TET tet = null;

try {
    ...TET method invocations...
} catch (TETException e) {
    System.err.print("TET exception occurred:\n");
    System.err.print "[" + e.get_errnum() + "] " + e.get_apiname() + ": " +
        e.get_errmsg() + "\n";
} catch (Exception e) {
    System.err.println(e.getMessage());
} finally {
    if (tet != null) {
        tet.delete();
    }
} /* delete the TET object */
```

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

3.6 .NET Binding

The .NET edition of TET supports all relevant .NET concepts. In technical terms, the TET.NET edition is a C++ class (with a managed wrapper for the unmanaged TET core library) which runs under control of the .NET framework. It is packaged as a static assembly with a strong name. The TET assembly (*TET_dotnet.dll*) contains the actual library plus meta information.

Note TET.NET requires the .NET Framework 2.0 or above.

Installing the TET Edition for .NET. Install TET with the supplied Windows MSI Installer. The TET.NET MSI installer will install the TET assembly plus auxiliary data files, documentation and samples on the machine interactively. The installer will also register TET so that it can easily be referenced on the .NET tab in the *Add Reference* dialog box of Visual Studio .NET.

Installing TET.NET for ASP.NET. In order to use TET.NET in your ASP.NET scripts you must make the TET.NET assembly available to ASP. This can be achieved by placing *TETlib_dotnet.dll* in the *bin* subdirectory of your IIS installation (if it doesn't exist you must manually create it), or the *bin* directory of your Web application, e.g.

```
C:\inetpub\wwwroot\bin\TET_dotnet.dll           or  
C:\inetpub\wwwroot\WebApplicationX\bin\TET_dotnet.dll
```

Special considerations for ASP.NET. When using external files ASP's *MapPath* facility must be used in order to map path names on the local disk to paths which can be used within ASP.NET scripts. Take a look at the ASP.NET samples supplied with TET, and the ASP.NET documentation if you are not familiar with *MapPath*. Don't use absolute path names in ASP.NET scripts since these may not work without *MapPath*.

The directory containing your ASP.NET scripts must have execute permission, and also write permission unless the in-core method for generating PDF is used (the supplied ASP samples use in-core PDF generation).

Trust levels in ASP.NET 2.0 and above. ASP.NET 2.0 introduced some restrictions regarding the allowed operations in various trust levels for Web applications. Since TET.NET contains unmanaged code, it requires *Full Trust* level. TET.NET applications cannot be deployed in ASP.NET 2.0 applications with any other trust level, including High or Medium Trust.

Error handling. TET.NET 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_api-name*.

3.7 Perl Binding

Installing the TET Edition for Perl. TET is implemented as a C library which can dynamically be attached to Perl. This requires Perl to be built with support for loading extensions at runtime. The name of the TET Perl extension is *tetlib_pl*.

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 module file *tetlib_pl.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 both *tetlib_pl.so* (on Mac OS X: *tetlib_pl.dylib*) and *tetlib_pl.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/5.8/i686-linux
```

Windows. PDFlib supports the ActiveState port of Perl 5 to Windows, also known as ActivePerl. Both *tetlib_pl.dll* and *tetlib_pl.pm* will be 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\Perl5.8\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 {  
    ...some TET instructions...  
};  
die "Exception caught: $@" if $@;
```

3.8 PHP Binding

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.

Detailed information about the various flavors and options for using TET with PHP, including the question of whether or not to use a loadable TET module for PHP, can be found in the *PDFlib-in-PHP-HowTo* document which is 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.

You must configure PHP so that it knows about the external TET library. You have two choices:

- Add one of the following lines in *php.ini*:

```
extension=libtet_php.dll      ; for Windows
extension=libtet_php.so      ; for Unix
extension=libtet_php.sl      ; for HP-UX
extension=libtet_php.dylib   ; for Mac OS X
```

PHP will search 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.

- Alternatively, you can load TET at runtime with one of the following lines at the start of your script:

```
dl("libtet_php.dll");      # for Windows
dl("libtet_php.so");       # for Unix
dl("libtet_php.sl");       # for HP-UX
dl("libtet_php.dylib");    # for Mac OS X
```

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. Since PHP 5 supports structured exception handling, TET exceptions will be propagated as PHP exceptions. You can use the standard *try/catch* technique to deal with TET exceptions:

```
try {
```

...some TET instructions...

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

3.9 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 will be searched in the directories listed in the PYTHONPATH environment variable. The name of Python wrapper depends on the platform:

- ▶ Unix: *tetlib_py.so*
- ▶ Mac OS X: *tetlib_py.dylib*
- ▶ Windows: *tetlib_py.pyd*

Error Handling in Python. The Python binding installs a special error handler which translates TET errors to native Python exceptions. The Python exceptions can be dealt with by the usual try/catch technique:

```
try:
    ...some TET instructions...
except TETException:
    print 'TET Exception caught!'
```

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 `PDFlib` are based on CCSID 37 (US EBCDIC). Some special 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 various 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 QRPGLSRC,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/QRPGLSRC,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):

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

Exception Handling in RPG. TET clients written in ILE-RPG can use a limited form of TET's try/catch mechanism as follows:

```
c          eval      rtn=tet_try(tet)
c          if        TET_open_document(tet:in_filename:0:optlist)=-1
```



```
c          or tet_catch(tet)=1
c      *
c          callp    TET_delete(tet)
c          eval     error='Couldn't open input file '+
c                  %trim(out_filename)
c          exsr     exit
c          endif
```



4 TET Connectors

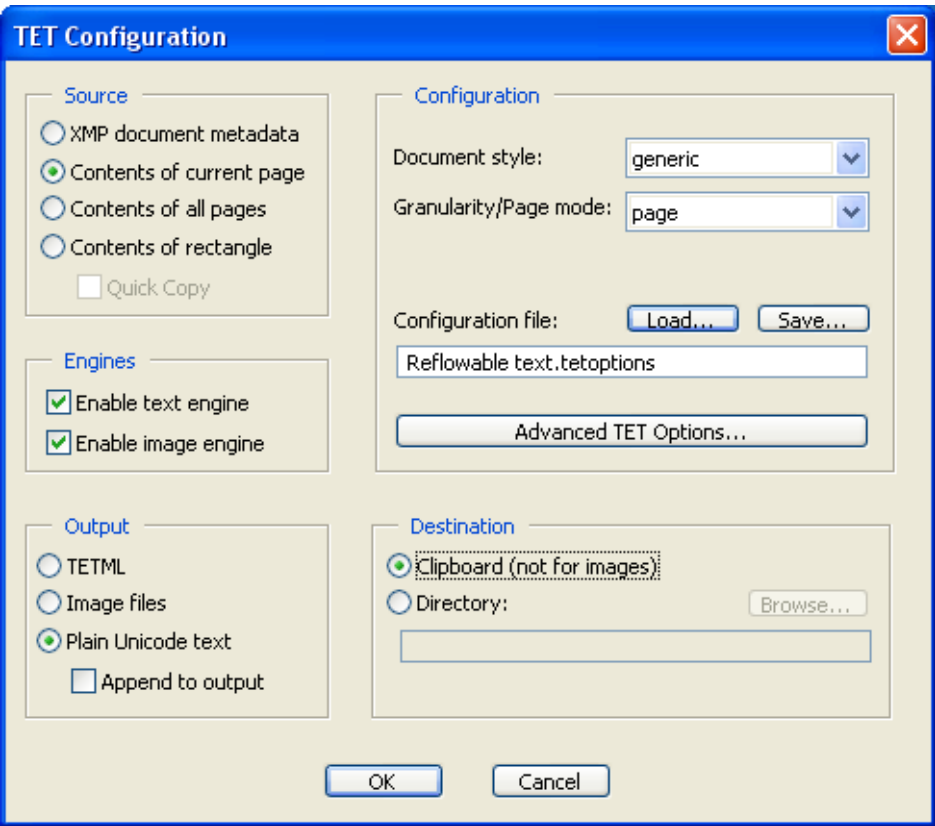
TET connectors provide the necessary glue code for interfacing TET with other software. TET connectors are based on the TET library or the TET command-line tool.

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 7-9 Standard, Pro, and Pro Extended (but not the free Adobe 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 demonstrate 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

Fig. 4.1
Configuration panel for the TET Plugin



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 in plain text to the system clipboard or a disk file. Enhanced clipboard controls facilitate the use of copy/paste.
- ▶ Convert a PDF to TETML and place it in the clipboard or a disk file.
- ▶ Copy XMP document metadata to the clipboard or a disk file.
- ▶ Extract images from the document as TIFF, JPEG, or JPEG 2000 files.
- ▶ Find words in the document.
- ▶ 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) will be 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.

4.2 TET Connector for the Lucene Search Engine

Lucene is an open-source search engine. Lucene is primarily a Java project, but a C version is also available and a version for .NET is under development. 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, »Indexing protected PDF Documents«, page 49, 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:

- ▶ JDK 1.4 or newer
- ▶ 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 file *lucene-core-2.4.0.jar*, which is part of the Lucene 2.4.0 distribution.
- ▶ An installed TET distribution package for Unix, Linux, Mac, or Windows.

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

- ▶ *cd* to the directory *<TET install dir>/connectors/lucene*.
- ▶ Copy the file *lucene-core-2.4.0.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*:

```
devserver (1)$ ant index
Buildfile: build.xml
...
index:
[java] adding ../data/Whitepaper-XMP-metadata-in-PDFlib-products.pdf
[java] adding ../data/Whitepaper-PDFA-with-PDFlib-products.pdf
[java] adding ../data/FontReporter.pdf
[java] adding ../data/TET-PDF-IFilter-datasheet.pdf
[java] adding ../data/PDFlib-datasheet.pdf
[java] 1255 total milliseconds
```

```
BUILD SUCCESSFUL
Total time: 2 seconds
```

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

```
compile:
```

```
search:
```

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

```
PDFlib
```

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

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

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

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

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

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

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

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

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

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

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

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

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

```
q
```

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

```
title:FontReporter
```

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

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

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

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

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

```
q
```

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

```
BUILD SUCCESSFUL
```

```
Total time: 57 seconds
```

Two queries have been performed: one for the word *PDFlib* in the text, and another one for the word *FontReporter* 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.jar=/tmp/lucene-core-2.4.0.jar index
```

Testing TET and Lucene with the demo Web application. The Lucene demo Web application can be deployed on any Java servlet container such as Tomcat or GlassFish. The required steps are described in the HTML documentation that comes with Lucene, also available online at lucene.apache.org/java/2_4_0/demo3.html.

Note the step *Configuration* on that page. Here you must make the location of the index known to the Web application by entering it in the file *configuration.jsp*. The path to add here would be `<TET install dir>/bind/lucene/index` if Ant was run without overriding the property for the location of the Lucene index.

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

- ▶ *path* (tokenized field): the pathname of the document
- ▶ *modified* (DateField): the date of the last modification
- ▶ *contents* (Reader field): 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 Chapter 9, »The pCOS Interface«, page 105, for more details on pCOS), e.g.

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

- ▶ *font*: 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 will be 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 lucene.apache.org/solr).

Solar 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 8.1, »Creating TETML«, page 89).

Note Protected documents can be indexed with the shrug option under certain conditions (see Chapter 5.1, »Indexing protected PDF Documents«, page 49, 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 will be 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 will be 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 will be processed recursively:

```
<?xml version="1.0" encoding="UTF-8"?>
<add>
<doc>
<field name="id">PDFlib-FontReporter-E.pdf</field>
<field name="Author_s">PDFlib GmbH</field>
<field name="CreationDate_s">2008-07-08T15:05:39+00:00</field>
<field name="Creator_s">FrameMaker 7.0</field>
<field name="ModDate_s">2008-07-08T15:05:39+00:00</field>
<field name="Producer_s">Acrobat Distiller 7.0.5 (Windows)</field>
<field name="Subject_s">PDFlib FontReporter</field>
<field name="Title_s">PDFlib FontReporter 1.3 Manual</field>
<field name="text">PDFlib</field>
<field name="text">GmbH</field>
<field name="text">München</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, »Indexing protected PDF Documents«, page 49, 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 download.oracle.com/docs/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 121, for option list details). This is especially useful for supplying the TET license key, e.g.:

```
TETOPT="license=aaaaaaaa-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 will be 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.pdfliib.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), a developer product for reliably 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 the 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:

- Extracts text even from PDFs where Acrobat fails
- Indexes protected documents
- Indexes not only page content, but also document- and image-related metadata, bookmarks, PDF attachments, and PDF packages/portfolios
- Performance: thread-safe, fast and robust, 32- and 64-bit
- Lean stand-alone product without side effects
- Automatic language/script detection
- Actively supported by a dedicated team

Enterprise PDF search. TET PDF IFilter is available in fully thread-safe native 32- and 64-bit versions. You can implement enterprise PDF search solutions with TET PDF IFilter and the following products:

- Microsoft Office SharePoint Server (MOSS)
- Microsoft Search Server 2008 and the free Search Server 2008 Express
- Microsoft SQL Server
- Microsoft Exchange Server

TET PDF IFilter can be used with all other Microsoft and third-party products which support the IFilter interface.

Desktop PDF search. TET PDF IFilter can also be used to implement desktop PDF search, e.g. with the following products:

- Windows Desktop Search (WDS): integrated in Windows Vista; also available as free add-on for Windows XP
- Windows Indexing Service

TET PDF IFilter is freely available for non-commercial desktop use, which provides a convenient basis for test and evaluation.

XMP document metadata and document info entries. The advanced metadata implementation in TET PDF IFilter supports the Windows property system for metadata. It indexes XMP metadata (Adobe's rich XML-based metadata description language) 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 equivalent Windows properties, e.g. *Title, Subject, Author*.
- ▶ TET PDF IFilter adds useful PDF-specific pseudo properties, e.g. page size, PDF/A conformance level, font names.
- ▶ All relevant predefined XMP properties can be searched, e.g. *dc:rights, xmpRights:UsageTerms, xmp:CreatorTool*.
- ▶ User-defined XMP properties can be searched, e.g. company-specific classification properties, PDF/A extension schemas.

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.

XMP image metadata. In addition to document metadata, TET PDF IFilter also supports XMP metadata attached to individual images. In modern workflows metadata travels with the image, e.g. from the digital camera to Photoshop editing up to page layout creation and PDF production. TET PDF IFilter picks up XMP image metadata and makes it available for searches. For example, you can search for documents which contain images from a certain category, images created by a specific photographer, etc.

Internationalization. TET PDF IFilter includes full support for extracting 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.

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
- ▶ Embedded PDFs are processed recursively so that also the text in attached PDF documents can be searched.
- ▶ All documents in a PDF package are indexed. PDF packages are an Acrobat 8 feature for grouping multiple documents in a single PDF file (in Acrobat 9 called portfolios).

4.6 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, »Indexing protected PDF Documents«, page 49, 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 will not be indexed. The following requirements must be met:

- ▶ PHP 5.0 or above
- ▶ MediaWiki 1.11.2 or above (see below for older versions)
- ▶ A TET distribution package for Unix, Linux, Mac, 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 3.8, »PHP Binding«, page 29.
- ▶ Copy `<TET install dir>/connectors/MediaWiki/PDFIndexer.php` to `<MediaWiki install dir>/extensions/PDFIndexer/PDFIndexer.php`.
- ▶ 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
include("extensions/PDFIndexer/PDFIndexer.php");
```

- ▶ In order to avoid warnings when uploading PDF documents it is recommended to add the following lines to `<MediaWiki install dir>/includes/DefaultSettings.php` in order to make `.pdf` a well-known file type extension:

```
/**
 * This is the list of preferred extensions for uploading files. Uploading files
 * with extensions not in this list will trigger a warning.
 */
$wgFileExtensions = array( 'png', 'gif', 'jpg', 'jpeg', 'pdf' );
```

Working with MediaWiki versions older than 1.11.2. The TET connector for MediaWiki does not work properly in MediaWiki versions older before 1.11.2 due to a bug in MediaWiki; a PHP error message occurs instead. In order to use the TET connector with older MediaWiki versions you must apply a simple patch to the file `include/SpecialUpload.php` as detailed here:

[svn.wikimedia.org/viewvc/mediawiki/trunk/phase3/includes/ ↵](http://svn.wikimedia.org/viewvc/mediawiki/trunk/phase3/includes/SpecialUpload.php?sortby=file&r1=30403&r2=30402&pathrev=30403)
`SpecialUpload.php?sortby=file&r1=30403&r2=30402&pathrev=30403`

How the TET connector for MediaWiki works. The TET connector for MediaWiki consists of the PHP module `PfIndexer.php`. Using one of MediaWiki's predefined hooks it is hooked up so that it will be called whenever a new PDF document is uploaded. It ex-

Advanced search

Search in namespaces:

☒ (Main)
 ☐ Talk
 ☐ User
 ☐ User talk
 ☐ Project
 ☐ Project talk
 ☒ Image
 ☐ Image talk
 ☐ MediaWiki
 ☐ MediaWiki talk
 ☐ Template
 ☐ Template talk
 ☒ Help
 ☐ Help talk
 ☐ Category
 ☐ Category talk
 ☐ Manual
 ☐ Manual talk
 ☒ Extension
 ☐ Extension talk

☐ List redirects

Search for

Fig. 4.2 Searching PDF documents in MediaWiki

tracts 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 will also be indexed. The text for the index is constructed as follows:

- ▶ The TET connector feeds the value of all 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 will completely be fed to the index. The advantage of this method is that search results will 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 will be fed to the index. Otherwise it will be truncated, i.e. some text towards the end of the document will not be indexed.

The predefined limit is 512 KB, but this can be changed in *PDFIndexer.php*. If one of the size tests described above hits the limit, a warning message will be written to MediaWiki's *DebugLogFile* if MediaWiki logging is activated.

Searching for PDF documents. Since PDF documents are treated as images by MediaWiki you must search them in the *Image* namespace. This can be achieved by activating the *Image* checkbox in the list of namespaces in the *Advanced search* dialog (see Figure 4.2). The *Image* namespace will not be searched by default. However, this setting can be enabled in the *LocalSettings.php* preferences file as follows:

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

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 will be 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 will be displayed in front of the text. These line numbers are not relevant for PDF documents, and you can safely ignore them.

Indexing metadata fields. The TET connector for MediaWiki indexes all standard document info fields. The value of each field will be fed to the index so that it 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 Indexing protected PDF Documents

PDF documents may use the following means of protection:

- ▶ A user password may be required for opening the document.
- ▶ Permission settings may allow or prevent certain actions, including extracting page contents.
- ▶ A master password may be required for changing permission settings or one of the passwords.

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 conditional can be used to check whether content extraction is allowed:

```
if ((int) tet.pcos_get_number(doc, "pcosmode") == 2 ||
    ((int) tet.pcos_get_number(doc, "pcosmode") == 1 &&
     (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).
- ▶ 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 pro-

tected 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 will be 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* will be set to `true/1`.
- ▶ pCOS runs in full mode (instead of restricted mode), i.e. the *pcosmode* pseudo object will be 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 10.4, page 132. This resource will be evaluated in <code>TET_open_document()</code> , and the result will be appended after the mappings specified in the option <code>glyphmapping</code> of <code>TET_open_document()</code> .
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) will be 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 searching 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 will be searched in reverse order (paths set at a later point in time will be 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 will initialize the *searchpath* resource category with a value read from the following registry key:

```
HKLM\SOFTWARE\PDFlib\TET\3.0\searchpath
```

This registry entry may contain a list of path names separated by a semicolon ';' character. The Windows installer will initialize the *searchpath* registry entry with the following directory names (or similar if you installed TET in a custom directory):

```
C:\Program Files\PDFlib\TET 3.0\resource
C:\Program Files\PDFlib\TET 3.0\resource\cmap
```

On IBM iSeries the *searchpath* resource category will be initialized with the following values:

```
/tet/3.0/resource  
/tet/3.0/resource/cmap
```

On MVS the *searchpath* feature is not supported.

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 will be 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/3.0/tet.upr (on iSeries)  
tet.upr (Windows, Unix, and all other systems)
```

If this file cannot be read no exception will be thrown.

- ▶ On Windows TET will additionally try to read the following registry entry:

```
HKLM\SOFTWARE\PDFlib\TET\3.0\resourcefile
```

The value of this key (which will be created with the value *<installdir>/tet.upr* by the TET installer, but can also be created by other means) will be taken as the name of the resource file to be used. If this file cannot be read an exception will be thrown.

- ▶ The client can force TET to read a resource file at runtime by explicitly setting the *resourcefile* option:

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

This call can be repeated arbitrarily often; the resource entries will be 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:

```
TET_set_option(tet, "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 and CMap 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 *\ooo* will be 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 121, 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()` to maximize text extraction throughput:

- ▶ `docstyle=searchengine`
Several internal parameters will be set to speed up operation by reducing the output quality in a way which does not affect the indexing process for search engines.
- ▶ `skipengines={image}`
If image extraction is not required internal image processing can be skipped in order to speed up operation.
- ▶ `contentanalysis={merge=0}`
This will disable the expensive strip and zone merging step, and reduces processing times for typical files to ca. 60% compared to default settings. However, documents where the contents are scattered across the pages in arbitrary order may result in some text which is not extracted in logical order.
- ▶ `contentanalysis={shadowdetect=false}`
This will disable detection of redundant shadow and fake bold text, which can also reduce processing times.

Words vs. line layout vs. reflowable text. Different applications will prefer different kinds of output (hyphenated words will always be 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 will be separated with a linefeed character to retain the existing line structure.
- ▶ Reflowable text: in order to avoid line breaks and facilitate reflowing of the extracted text use `contentanalysis={lineseparator=U+0020}` and `granularity=page` in `TET_open_page()`. The full page contents can be fetched with a single call to `TET_get_text()`. Zones will be separated with a linefeed character, and a space character will be inserted between the lines in a zone.

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, or other details. If you are not interested in text coordinates calling `TET_get_text()` will be sufficient.
- ▶ If you have advance information about the layout of pages you can use the `include-box` 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.

Unknown characters. If TET is unable to determine the appropriate Unicode mapping for one or more characters it will represent it with the Unicode replacement character U+FFFD. If your application is not concerned about unmappable characters you can simply discard all occurrences of this character. Applications which require more fine-grain results could take the corresponding font into account, and use it to decide on processing of unmappable characters. Use the following document option to replace all unmapped characters with a question mark:

```
unknownchar=?
```

The special character U+0000 means »no character«. Use the following document option to remove all unmapped characters from the output:

```
unknownchar=U+0000
```

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 if content extraction is not allowed in the permission settings.
- ▶ 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 or fonts adjust the mapping tables as appropriate.
- ▶ If you have a large number of documents with unmappable fonts 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.

Corporate fonts with custom-encoded logos. In many cases corporate fonts containing custom logos have 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 76) 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 77, 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 features 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 documents 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 Document Domains

PDF documents may contain text in many other places than only the page contents. While most applications will 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 will summarize how to search these document domains with Acrobat 8/9. 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 8/9: page contents are always visible
- ▶ How to search a single PDF with Acrobat 8/9: *Edit, Find* or *Edit, 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 35). The TET Plugin offers its own search dialog via *Plug-Ins, PDFlib TET Plugin... TET Find*. However, this is not intended as a full-blown search facility.
- ▶ How to search multiple PDFs with Acrobat 8/9: *Edit Search* and in *Where would you like to search?* select *All PDF Documents in*, and browse to a folder with PDF documents.
- ▶ Sample code for the TET library: *extractor* mini sample
- ▶ TETML element: */TET/Document/Pages/Page*

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

- ▶ How to display with Acrobat 8/9: *File, Properties...*
- ▶ How to search a single PDF with Acrobat 8/9: not available
- ▶ How to search multiple PDFs with Acrobat 8/9: click *Edit, Search* and *Use Advanced Search Options*. 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* mini 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 8/9: *File, Properties..., Custom* (not available in the free Adobe Reader)
- ▶ How to search with Acrobat 8/9: not available
- ▶ Sample code for the TET library: *dumper* mini sample
- ▶ TETML element: */TET/Document/DocInfo/Custom*

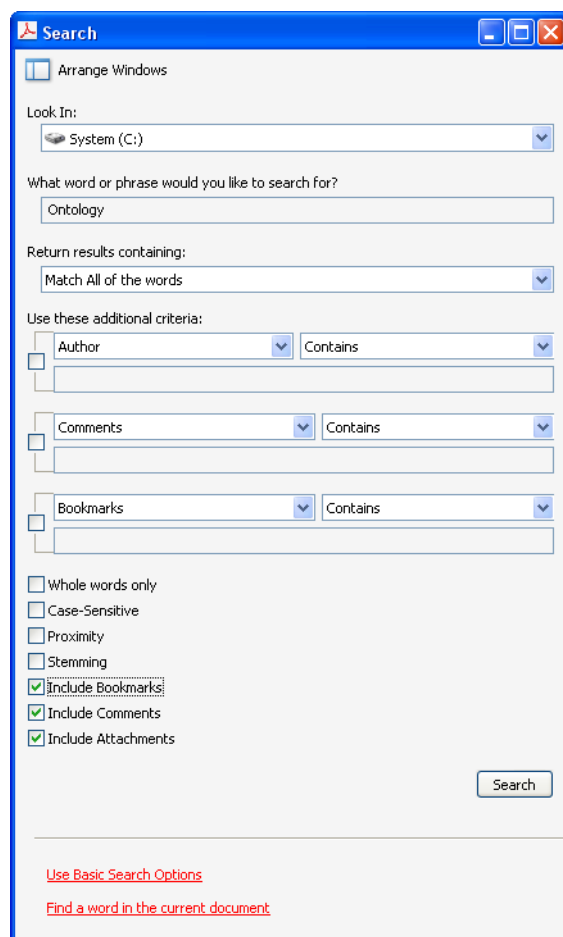


Fig. 6.1
Acrobat's advanced
search dialog

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

- ▶ How to display with Acrobat 8/9: *File, Properties..., Additional Metadata..* (not available in the free Adobe Reader)
- ▶ How to search a single PDF with Acrobat 8/9: not available
- ▶ How to search multiple PDFs with Acrobat 8/9: click *Edit, Search* and *Use Advanced Search 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 the free Adobe Reader).
- ▶ Sample code for the TET library: *dumper* mini 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 8/9: *Tools, Advanced Editing, TouchUp Object Tool*, select image, right-click, *Show Metadata...* (not available in the free Adobe Reader)

- ▶ How to search with Acrobat 8/9: not available
- ▶ Sample code for the TET library: pCOS Cookbook topic *image_metadata*
- ▶ TETML element: */TET/Document/Pages/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 8: *View, Navigation Panels, Fields*
- ▶ How to display with Acrobat 9: *Forms, Add or Edit Fields...*
- ▶ How to search with Acrobat 8/9: not available
- ▶ Sample code for the TET library: pCOS Cookbook topic *fields*
- ▶ TETML element: not available

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 8/9: *View, Navigation Panels, Comments*
- ▶ How to search a single PDF with Acrobat 8/9: *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 8/9: click *Edit, Search* and *Use Advanced Search 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: pCOS Cookbook topic *annotations*
- ▶ TETML element: not available

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 8/9: *View, Navigation Panels, Bookmarks*
- ▶ How to search a single PDF with Acrobat 8/9: *Edit, Search* and check the box *Include Bookmarks*
- ▶ How to search multiple PDFs with Acrobat 8/9: click *Edit, Search* and *Use Advanced Search 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 the free Adobe Reader)
- ▶ Sample code for the TET library: pCOS Cookbook topic *bookmarks*
- ▶ TETML element: not available

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

- ▶ How to display with Acrobat 8/9: *View, Navigation Panels, Attachments*
- ▶ How to search with Acrobat 8/9: Use *Edit, Search* and check the box *Include Attachments* (not available in the free Adobe Reader). Nested attachments will not be searched recursively.
- ▶ Sample code for the TET library: *get_attachments* mini sample
- ▶ TETML element: */TET/Document/Attachments/Attachment/Document*

PDF packages and portfolios. Acrobat 8 (PDF 1.7) introduced the concept of PDF packages which are file attachments with additional properties. Acrobat 9 (PDF 1.7 extension level 3) extends this concept with the introduction of PDF portfolios.

- ▶ How to display with Acrobat 8/9: 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 8/9: *Edit, Search* and in the *Look In:* pull-down select *In the Entire PDF Package*
- ▶ How to search multiple PDF packages with Acrobat 8/9: not available
- ▶ Sample code for the TET library: *get_attachments* mini sample
- ▶ TETML element: */TET/Document/Attachments/Attachment/Document*

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

- ▶ How to display with Acrobat 8: Acrobat 8 does not directly display standards conformance information, but you can find relevant entries in *File, Properties...*, *Custom or File, Properties...*, *Additional Metadata...* You can also use the free PDFlib custom XMP panel¹ for ISO standards to explicitly display conformance information for the PDF/A-1, PDF/X-4, PDF/X-5, and PDF/E-1 standards.
- ▶ Acrobat 9: *View, Navigation Panels, Standards* (only present for standard-conforming PDFs)
- ▶ How to search with Acrobat 8/9: not available
- ▶ Sample code for the TET library: *dumper* mini sample
- ▶ TETML elements and attributes: */TET/Document/@pdfa*, */TET/Document/@pdfx*

1. See www.pdflib.com/developer/xmp-metadata/xmp-panels

6.2 Unicode Concepts

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), i.e. characters in the range U+0000...U+FFFF 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.

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 6.1 lists the representation of the BOM for various encoding forms.

Table 6.1 Byte order marks for various Unicode encoding forms

Encoding form	Byte order mark (hex)	graphical representation
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 ¹
UTF-32 little-endian	FF FE 00 00	ÿþ ? ? ¹

1. There is no standard graphical representation of null bytes.

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 different graphical variants 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 is represented by 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 6.2).

Composite characters and sequences. Some glyphs map to a sequence of multiple characters. For example, some ligatures will be mapped to multiple characters according to their constituent characters. However, composite characters (such as the Roman numeral in Figure 6.2) may or may not be split, subject to information in the font and PDF. See Table 6.2, page 68, for a list of characters which will be post-processed by TET.

If appropriate, TET will split composite characters into a sequence of constituent characters. The corresponding sequence will be part of the text returned by `TET_get_text()`. For each character details of the underlying glyph 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 will be 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

Characters

Glyphs

U+0067 LATIN SMALL LETTER G

g g g g g g

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

fi fi

U+2126 OHM SIGN or
U+03A9 GREEK CAPITAL LETTER OMEGA

Ω

U+2167 ROMAN NUMERAL EIGHT or
U+0056 V U+0049 I U+0049 I

VIII

Fig. 6.2
Relationship of glyphs
and characters

artificial characters which will be 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* options are artefacts without any corresponding glyph.
- ▶ While the leading value of a surrogate pair will be associated with a glyph, the trailing value will be treated as not having a corresponding glyph on the page (see Section 6.5, »Unicode Pipeline«, page 68, section »Characters outside the BMP and surrogate handling«).

6.3 Page and Text Geometry

Coordinate system. TET represents all page and text metrics in the default coordinate system of PDF. However, the origin of the coordinate system (which could be located outside the page) will be adjusted to the lower left corner of the visible page. More precisely, the origin will be located in the lower left corner of the *CropBox* if it is present, or the *MediaBox* otherwise. Page rotation will be 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. All coordinates expected or returned by TET are interpreted in this coordinate system, regardless of their representation in the underlying PDF document. See Section 9.1, »Simple pCOS Examples«, page 105 to see how to determine the size of a PDF page.

Area of text extraction. By default, TET will extract all text from the visible page area. Using the *clippingarea* option of *TET_open_page()* (see Table 10.5, page 134) 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 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 character 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.

In order to determine suitable values for the *includebox* and *excludebox* options it can be helpful to display cursor coordinates in Acrobat. Note, however, that in Acrobat coordinates will increase towards the bottom of the page, while the PDF coordinate system (and therefore TET) use coordinates which increase towards the top of the page.

- ▶ Display cursor coordinates in Acrobat:
Acrobat 7/8: *View, Navigation Panels, Info*;
Acrobat 9: *View, Cursor Coordinates*.
- ▶ Select points as units for the coordinate display:
Acrobat 7/8/9: *Edit, Preferences, [General], Units&Guides, Page&Ruler, Points*;
In Acrobat 7/8 you can also use *Options, Points* in the *Info* panel.

Glyph metrics. 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 10.10, page 144):

- ▶ The *uv* value contains the UTF-32 Unicode value of the current character, i.e. the character for which details are retrieved. This field will always contain UTF-32, even in language bindings that can deal only with UTF-16 strings in their native Unicode strings. Accessing the *uv* field allows applications to deal with characters outside the BMP without having to interpret surrogate pairs. Since surrogate pairs will be report-

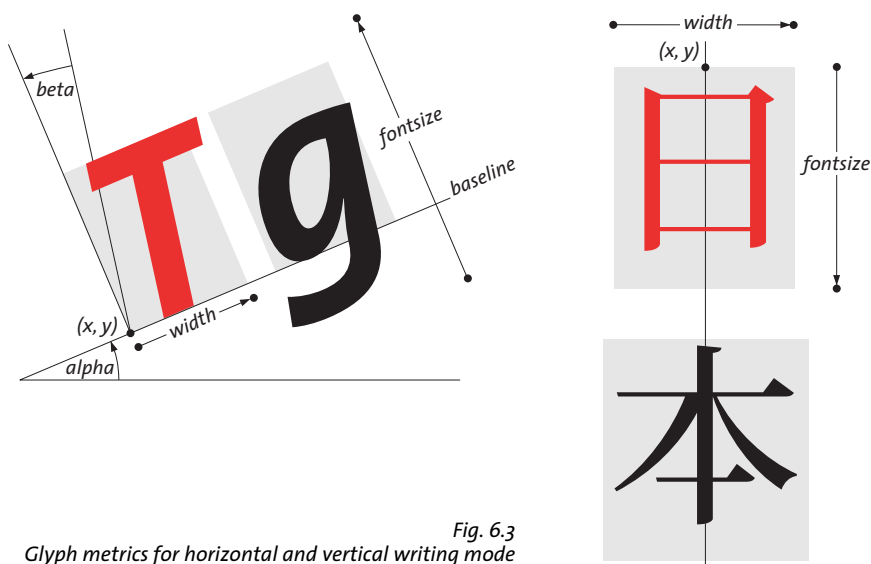


Fig. 6.3
Glyph metrics for horizontal and vertical writing mode

ed as two separate characters, the *uv* field of the leading surrogate value will contain the actual Unicode value (larger than U+FFFF). The *uv* field of the trailing surrogate value will be treated as an artificial character, and will have an *uv* value of 0.

- The *type* field specifies how the character was created. 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), the trailing value of a surrogate pair, and inserted separator characters. For artificial characters the position (x, y) will specify the endpoint of the most recent real character, the *width* will be 0, and all other fields except *uv* will be those of the most recent real character. The endpoint is the point (x, y) plus the *width* added in direction *alpha* (in horizontal writing mode) or plus the *fontsize* in direction -90° (in vertical writing mode).
- The *unknown* field will usually be 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 (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, »Support for Chinese, Japanese, and Korean Text«, page 67 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 *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* will be 0 for artificial characters.

- ▶ The angle *alpha* provides the direction of inline 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* will be 0° for standard horizontal text as well as for standard vertical text.
- ▶ The angle *beta* specifies any skewing which has been applied to the text, e.g. for slanted (italicized) text. The angle will be measured against the perpendicular of *alpha*. It will be 0° for standard upright text (for both horizontal and vertical writing mode). If the absolute value of *beta* is greater than 90° the text will be 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. Section 9.1, »Simple pCOS Examples«, page 105 shows sample code for retrieving font details.
- ▶ The *fontsize* field specifies the size of the text in points. It will be normalized, and therefore always be positive.
- ▶ The *textrendering* field specifies the kind of rendering for a glyph, e.g. stroked, filled, or invisible. It will reflect the numerical text rendering mode as defined for PDF page descriptions (see Table 10.10, page 144). Invisible text will be extracted by default, but this can be changed with the *ignoreinvisibletext* option of `TET_open_page()`.

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:

```
x_end = x + width * cos(alpha)
y_end = y + width * sin(alpha)
```

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

```
x_end = x + width
y_end = y
```

For CJK text with vertical writing mode the end point calculation works as follows:

```
x_end = x
y_end = y - fontsize
```

6.4 Support for Chinese, Japanese, and Korean Text

CJK support. TET supports Chinese, Japanese, and Korean (CJK) text, and will convert horizontal and vertical CJK text in arbitrary encodings (CMaps) to Unicode. TET supports all of Adobe's CJK character collections, which cover all PDF CMaps used in PDF versions up to and including 1.6:

- ▶ Simplified Chinese: *Adobe-GB1-4*
- ▶ Traditional Chinese: *Adobe-CNS1-4*
- ▶ Japanese: *Adobe-Japan1-6*
- ▶ Korean: *Adobe-Korea1-2*

The PDF CMaps in turn cover all of the CJK character encodings which are in use today, such as Shift-JIS, EUC, Big-5, KSC, and many others.

Note In order to extract CJK text you must configure access to the CMap files which are shipped with TET according to Section 0.1, »Installing the Software«, page 7.

Several groups of CJK characters will be modified (see Table 6.2, page 68, for details):

- ▶ Fullwidth ASCII variants and fullwidth symbol variants will be mapped to the corresponding halfwidth characters.
- ▶ CJK compatibility forms (prerotated glyphs for vertical text) and small form variants will be mapped to the corresponding normal variants.

CJK font names which are encoded with locale-specific encodings (e.g. Japanese font names encoded in Shift-JIS) will also be normalized to Unicode. The wordfinder will treat all ideographic CJK characters as individual words, while Katakana characters will not be treated as word boundaries (a sequence of Katakana will be treated as a single word).

CJK text with 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 will advance downwards as determined by the font size and character spacing, regardless of the glyph width (see Figure 6.3).
- ▶ The angle *alpha* will be 0° for standard vertical text. In other words, fonts with vertical writing mode and *alpha=0°* will 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 pCOS code shown in Section 9.1, »Simple pCOS Examples«, page 105 for determining the writing mode of a font. Note that not all text which appears vertically actually uses a font with vertical writing mode.
- ▶ Prerotated glyphs for Latin characters and punctuation will be mapped to the corresponding unrotated Unicode character (see Table 6.2).

6.5 Unicode Pipeline

TET is completely based on the Unicode standard which is only concerned about characters, but not about glyphs. While text in PDF can be represented with a variety of font and encoding schemes, TET will abstract from glyphs and normalize 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.

However, despite all efforts there are still 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. These features are discussed in Section 6.7, »Layout Analysis«, page 74.

Post-processing for certain Unicode values. In some cases the Unicode values which are determined as a result of font and encoding processing will be modified by a post-processing step, e.g. to split ligatures. Table 6.2 lists all Unicode values which are affected by post-processing.

Table 6.2 Post-processing for various Unicode values

UTF-16 values	Processing
U+0000-U+001F, U+007F-U+009F	Control characters will be removed. ¹ Mapping to U+0000 may be useful for eliminating unwanted characters.
U+0020, U+00A0, U+3000, U+2000-U+200B	Space characters will be mapped to U+0020.
U+D800 - U+DBFF (high) U+DC00 - U+DFFF (low) surrogates	Leading (high) surrogates and trailing (low) surrogates will be maintained in the UTF-16 output, and the corresponding UTF-32 value will be available in the uv field (see »Characters outside the BMP and surrogate handling«, page 69).
U+E000-U+F8FF (Private Use Area, PUA)	PUA characters will be kept or replaced according to the keep_pua option (see Section »Unmappable glyphs«).
U+F600-U+F8FF (Adobe CUS)	Will be mapped to the corresponding characters outside the CUS.
U+FB00-U+FB17	Latin and Armenian ligatures will be decomposed into their constituent characters. ²
U+FF01-U+FF5E, U+FF60-U+FF9F	Fullwidth ASCII and symbol variants will be mapped to the corresponding non-fullwidth characters. ³
U+FE30-U+FE6F	CJK compatibility forms (prerotated glyphs for vertical text) and small form variants (U+FE30-U+FE6F) will be mapped to the corresponding normal variants
undefined Unicode values	no modification

1. Characters inserted via the wordseparator, lineseparator options are not subject to this removal.
2. Ligatures in the Arabic and Hebrew presentation forms will not be decomposed.
3. The following characters will be left unchanged: halfwidth CJK punctuation (U+FF61-U+FF64), Katakana variants (U+FF65-U+FF9F), Hangul variants (U+FFA0-U+FFDC), and symbol variants (U+FFE8-U+FFEE).

Options for text filtering. There are several situations where TET will modify the actual character values found on the page in order to make the results more useful. Most of these steps can be controlled via options. The following list gives an overview of all operations which may modify the text:

- ▶ Dehyphenation will remove hyphen characters and combine the parts of a hyphenated word. This can be disabled with the *dehyphenate* suboption of the *contentanalysis* option for *TET_open_page()*.
- ▶ Redundant text which creates only visual artifacts such as shadow effects or artificial bold text will be removed. This can be disabled with the *shadowdetect* suboption of the *contentanalysis* option for *TET_open_page()*.
- ▶ Very small or very large text can be ignored. The limits can be controlled with the *fontsize* option of *TET_open_page()*.
- ▶ Unicode post-processing will replace certain Unicode characters with more familiar ones. For example, Latin ligatures will be replaced with their constituent characters, and fullwidth ASCII variants in CJK fonts will be replaced with the corresponding non-fullwidth characters. For details see Table 6.2, page 68.
- ▶ Invisible text (text with *textrendering=3*) will be extracted by default, but this can be changed with the *ignoreinvisibletext* option of *TET_open_page()*.
- ▶ Glyphs which cannot be mapped to Unicode will be replaced with the Unicode character defined in the *unknownchar* option of *TET_open_document()*. See section »Unmappable glyphs«, page 69.

Characters outside the BMP and surrogate handling. Characters outside Unicode's Basic Multilingual Plane (BMP), i.e. those with Unicode values above 0xFFFF, cannot be expressed as a single UTF-16 value, but require a pair of UTF-16 values called a surrogate pair. Examples of characters outside the BMP include certain mathematical and musical symbols at U+1DXXX as well as thousands of CJK extension characters starting at U+20000.

TET interprets and maintains surrogates, and allows access to the corresponding UTF-32 value even in programming languages where native Unicode strings support only UTF-16. Leading (high) surrogates and trailing (low) surrogates will be maintained. The string returned by *TET_get_text()* will return two UTF-16 values where the leading value will be treated as a real character which carries the glyph properties (size, font, etc.). The trailing value will be treated as an artificial character without any corresponding glyph.

The *uv* field returned by *TET_get_char_info()* for the leading surrogate value will contain the corresponding UTF-32 value. This allows direct access to the UTF-32 value of a non-BMP character even if you are working in an UTF-16 environment without any support for UTF-32. The *uv* field for the trailing surrogate value will be 0.

Unmappable glyphs. There are several reasons why text in a PDF cannot reliably be mapped to Unicode. If the document does not contain a ToUnicode CMap with Unicode mapping information for the codes used on the page, Unicode values can be missing for various reasons:

- ▶ 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. By default, TET will assign the Unicode Replacement Character U+FFFD to these characters. You can select a different replacement character (e.g. the space character

U+0020) for unmappable glyphs with the *unknownchar* option of *TET_open_document()*.

However, if the *keepua* option of *TET_open_document()* is *true*, unknown characters will be mapped to increasing values in the Private Use Area (PUA), starting at U+F200. The same glyph name used in different fonts will end up with the same PUA value, while TrueType or OpenType glyph ids from different fonts will have different PUA values assigned.

- ▶ If the font or PDF provides Unicode values, these may be contained in the Private Use Area (PUA). Since PUA characters are generally not very useful, TET will replace them with *unknownchar* (by default: U+FFFD). However, if the *keepua* option of *TET_open_document()* is *true*, PUA values will be returned without any modification. This may be useful if you can deal with PUA values, e.g. for a specific font, or for all fonts from a specific font vendor.

Since not all glyphs in a document may have proper Unicode values (e.g. custom symbols), TET may have to map some glyphs to *unknownchar*. Your code should be prepared for this character. If you don't care about Unicode mapping problems you can simply ignore it, or use the *unknownchar* option of *TET_open_document()* to set a different character as a replacement for unmappable glyphs (e.g. the *space* character).

In order to check for unmappable glyphs you can use the *unknown* field returned by *TET_get_char_info()*.

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 generally 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 will be 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 that will be 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 will be disabled. TET will return 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 post-processing (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 will group characters into logical words. Each fragment contains a word. Isolated punctuation characters (comma, colon, question mark, quotes, etc.) will be returned as separate fragments by default, while multiple sequential punctuation characters will be 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« below).
- ▶ With *granularity=line* the words identified by the wordfinder will be grouped into lines. If dehyphenation is enabled (which is the default) the parts of hyphenated words at the end of a line will be combined, and the full dehyphenated word will be part of the line.
- ▶ With *granularity=page* all words on the page will be returned in a single fragment.

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

The separator characters can be specified with the *wordseparator*, *lineseparator* sub-options of the *contentanalysis* option in *TET_open_page()* (use U+0000 to disable a separator), for example:

```
contentanalysis={lineseparator=U+000A}
```

By default, all content processing operations will be 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. 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 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.
- ▶ Some characters, such as space and punctuation characters (e.g. colon, comma, full stop, parentheses) will be considered a word boundary, regardless of their width and position. Note that ideographic CJK characters will be considered as word boundaries, while Katakana characters will not be treated as word boundaries. If the *punctuationbreaks* option in *TET_open_page()* is set to *false*, the wordfinder will no longer treat punctuation characters as word boundaries:

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

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 6.4).

Note Currently there is no dedicated support for right-to-left scripts and bidirectional text. Although Unicode values and glyph metrics can be retrieved, the wordfinder does not apply any special handling for right-to-left text.



Fig. 6.4
The default setting *punctuationbreaks=true* will separate the parts of URLs (top), while *punctuationbreaks=false* will keep 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 will therefore dehyphenate, or recombine 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 lower-case character, the hyphen will be removed and the first part of the word will be 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) will be left unmodified. The parts of a hyphenated word will not be modified, only the hyphen will be 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 sup-
port a feature for

Introduction

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 artifacts create severe problems when reusing the extracted text since redundant text contents which contribute only to the visual appearance will be processed although the text does not contribute to the page contents.

If the wordfinder is enabled, TET will identify and remove such redundant visual artifacts by default. Shadow removal can be disabled with the following option list for `TET_open_page()`:

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

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

6.7 Layout Analysis

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 processed 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 list for *TET_open_page()* which looks similar to the following:

```
docstyle=papers
```

The following types are available for the *docstyle* option (Table 6.3 contains typical examples for some document styles):

- ▶ *Book*: typical book layouts with regular pages
- ▶ *Business*: business documents
- ▶ *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 will be disabled to deliver only the raw text and speed up processing. For example, table and page structure recognition will be disabled.

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

Table detection. TET detects tabular structures on the page 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>
<Row>
  <Cell colSpan="5">
    <Para>
      <Word>
        <Text>5</Text>
        <Box llx="317.28" lly="637.14" urx="324.59" ury="650.29"/>
      </Word>
      <Word>
        <Text>.</Text>
        <Box llx="324.60" lly="637.14" urx="328.25" ury="650.29"/>
      </Word>
    </Para>
  </Cell>
</Row>
</Table>
```

Table 6.3 Document styles

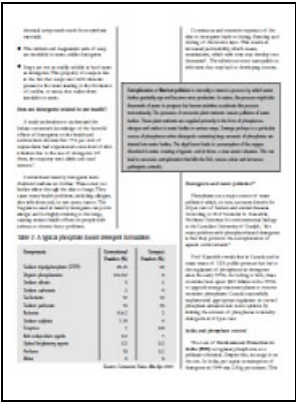
docstyle=book



docstyle=business



docstyle=fancy



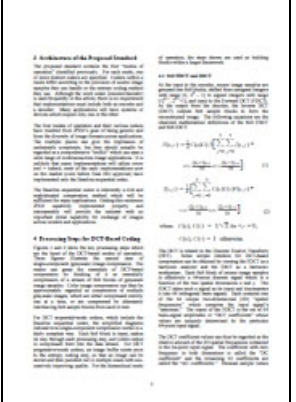
docstyle=magazine



docstyle=papers



docstyle=science



```
<Word>
<Text>REFERENCES</Text>
<Box llx="335.04" lly="637.14" urx="407.64" ury="647.47"/>
</Word>
</Para>
</Cell>
</Row>
...
</Table>
```

6.8 Advanced Unicode Mapping Controls

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.

Summary of Unicode mapping controls. Using the *glyphmapping* option of *TET_open_document()* (see Section 10.4, »Document Functions«, page 129) 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 will be 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 plugin¹. 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 6.5 shows two pages from a sample font report. Font reports created with the 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.

¹ The PDFlib FontReporter plugin is available for free download at www.pdflib.com/products/fontreporter

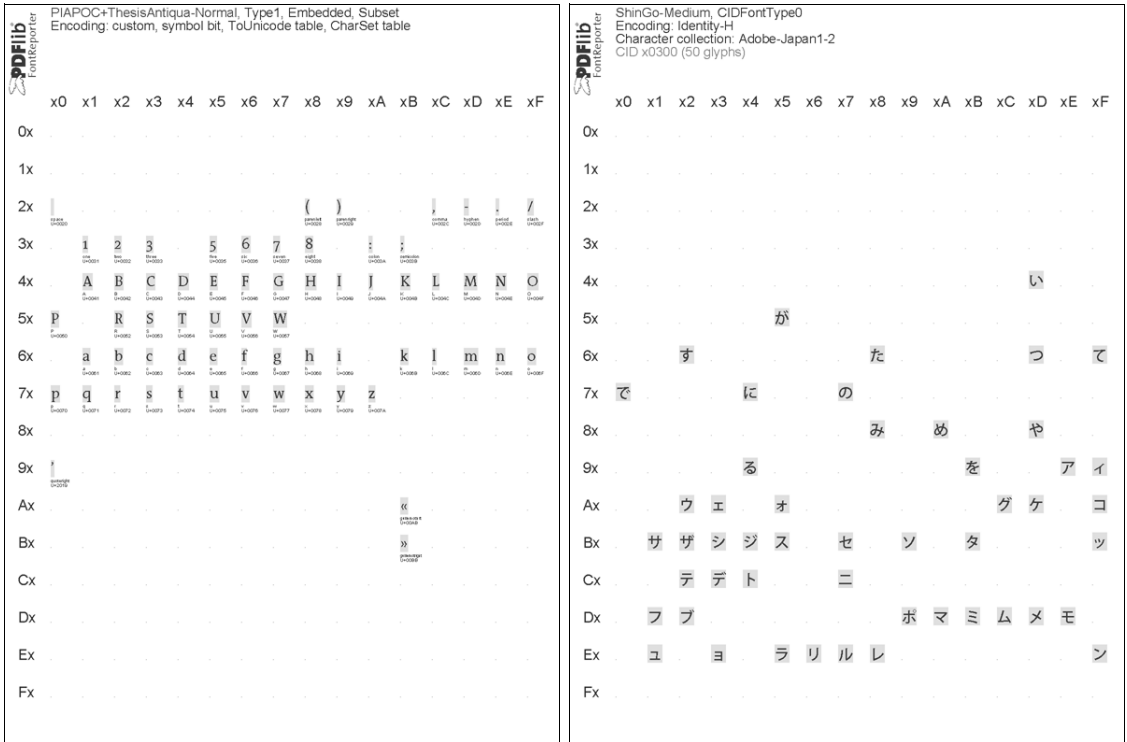


Fig. 6.5
Sample font reports created with the PDFlib FontReporter plugin for Adobe Acrobat

Precedence rules. TET will apply the glyph mapping controls in the following order:

- ▶ Codelist and ToUnicode CMap resources will be consulted first.
- ▶ If the font has an internal ToUnicode CMap it will be considered next.
- ▶ For glyph names TET will apply 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 will be 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 will be 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 '%' will be ignored; this can be used for comments.
- ▶ The first column contains the glyph code in decimal or hexadecimal notation. This must be a value in the range 0-255 for simple fonts, and in the range 0-65535 for CID fonts.
- ▶ The remainder of the line contains up to 7 Unicode code points for the code. The values can be supplied in decimal notation or (with the prefix x or 0x) in hexadecimal notation. UTF-32 is supported, i.e. surrogate pairs can be used.



Fig. 6.6
The font report for a logotype font shows that the font contains wrong Unicode mappings.
A custom code list can correct such mappings.

By convention, code lists use the file name suffix *.cl*. Code lists can be configured with the *codelist* resource. If no code list resource has been specified explicitly, TET will search for a file named *<mycodelist>.gl* (where *<mycodelist>* is the resource name) in the *search-path* hierarchy (see Section 5.2, »Resource Configuration and File Searching«, page 51 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):

```
TET_set_option(tet, "codelist {name name.cl}");
```

The following sample demonstrates the use of code lists. Consider the mismapped logotype glyphs in Figure 6.6 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}}
```

ToUnicode CMap resources for all font types. PDF supports a data structure called ToUnicode CMap which can be used to provide Unicode values for the glyphs of a font. If this data structure is present in a PDF file TET will use it. Alternatively, a ToUnicode CMap can be supplied in an external file. This is useful when a ToUnicode CMap in the PDF is incomplete, contains wrong entries, or is missing. A ToUnicode CMap will take precedence over a code list. However, code lists use an easier format the ToUnicode CMaps so they are the preferred format.

By convention, CMaps don't use any file name suffix. ToUnicode CMaps can be configured with the *cmap* resource (see Section 5.2, »Resource Configuration and File Searching«, page 51). The contents of a *cmap* resource must adhere to the standard CMap syntax.¹ In order to apply a ToUnicode CMap to all fonts in the *Warnock* family use the following option to *TET_open_document()*:

1. See partners.adobe.com/public/developer/en/acrobat/5411.ToUnicode.pdf

```
glyphmapping {{fontname=Warnock* tounicodecmap=warnock}}
```

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 '%' will be 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 will be 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 decimal notation or (with the prefix x or ox) in hexadecimal notation. 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 51)

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 51, 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):

```
TET_set_option(tet, "glyphlist {name name.gl}");
```

Due to the precedence rules for glyph mapping, glyph lists will not be 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 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 *Goo*, *Go1*, *Go2*, 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 will be 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 will be 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 *co0*, *co1*, *co2*, ..., *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 configured with the *fontoutline* resource category. As an alternative on Mac 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:

```
TET_set_option(tet, "fontoutline {WarnockPro WarnockPro.otf}");
```

See Section 5.2, »Resource Configuration and File Searching«, page 51, for more details on configuring external font files.

7 Image Extraction

7.1 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 will be created in the majority of cases. Most TIFF images created by TET can be used in the majority of TIFF viewers and consumers. However, some advanced TIFF features are not supported by all image viewers. Note that the Windows XP image viewer does not support the common Flate compression method in TIFF. We regard Adobe Photoshop as benchmark for the validity of TIFF images.
- ▶ JPEG (*.jpg*) will be created for images which are already compressed with the JPEG algorithm (*DCTDecode* filter) in PDF. However, in some cases DCT-compressed images must be extracted as TIFF since not all aspects of PDF color handling can be expressed in JPEG.
- ▶ JPEG 2000 (*.jpx*) will be created for images which are already compressed with the JPEG 2000 algorithm (*JPXDecode* filter) in PDF.

Placed images and image objects. TET distinguishes between placed images and image objects. 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). The image is usually 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.

An *image object* is a resource which represents the actual pixel data, colorspace and number of components, number of bits per component, etc. Unlike placed images, image objects don't have any intrinsic geometry. However, they do have width and height properties (measured in pixels). Each image object has a unique associated ID which can be used to extract its pixel data. Image objects are represented by the *Image* element in TETML.

A placed image always corresponds to exactly one image object, but there is no one-to-one relationship. 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 associated with the same image 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 object. This would result in the same visual appearance, but a more bloated PDF document.

Page-oriented and document-oriented image extraction. The distinction between placed images and image objects gives rise to two fundamentally different approaches to image extraction:

- ▶ Page-oriented image extraction: the application is interested in the exact page layout, but doesn't care about duplicated images. Extracting images in a page-oriented way may result in the same data for more than one extracted image. However, the application can avoid image duplication by checking for duplicate image IDs.

- Document-oriented image extraction: the application is interested in all images in the document, but doesn't care which image is used on which page. Images which are placed more than once should be extracted only once.

In order to extract an image, the corresponding image ID is required. The image ID is used as an index in the pCOS *images[]* array, and can be obtained in the following ways which correspond to page-oriented and document-oriented image extraction, respectively:

- *TET_get_image_info()* retrieves geometric information about a placed image as well as the pCOS image ID (in the *imageid* field of *TET_image_info*) 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()* will not touch the actual pixel data of the image. If the same image is referenced multiply on one or more pages, the corresponding IDs will be the same. This method for page-oriented image extraction is demonstrated in the *extractor* mini sample and the *image_extractor* topic in the TET Cookbook.
- Enumerate all values from 0 to the highest image ID, which is queried with *TET_pcos_get_number()* as the value of the pCOS path *length:images*. Note that the relationship of images and pages will be lost in this case.

Once an image ID has been retrieved, the functions *TET_write_image_file()* or *TET_get_image_data()* can be called to write the image data to a disk file or fetch the pixel data in memory, respectively.

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 and its use in PDF at the following location: www.pdflib.com/developer/xmp-metadata.

An image object may have XMP metadata associated with it in the PDF document. If XMP metadata is present, TET will by default embed 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 will ignore image metadata when generating the image output file.

The *image_metadata* topic in the pCOS Cookbook shows how to extract image metadata with the pCOS interface directly, without generating any image file.

7.2 Image Geometry

Using `TET_get_image_info()` you can retrieve geometric information for a placed image. The following values are available for each image in the `TET_image_info` structure (see Figure 7.1):

- ▶ 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 *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 will be in the range $-180^\circ < \alpha \leq +180^\circ$. The angle *alpha* rotates the image at its reference point. For upright images *alpha* will be 0° .
- ▶ The angle *beta* describes the direction of the pixel columns, relative to the perpendicular of *alpha*. This angle will be in the range $-180^\circ < \beta \leq +180^\circ$, but different from $\pm 90^\circ$. The angle *beta* skews the image, and *beta*= 180° mirrors the image at the *x* axis. For upright images *beta* will be in the range $-90^\circ < \beta < +90^\circ$. If $\text{abs}(\beta) > 90^\circ$ 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 actual image pixel data with `TET_write_image_file()` or `TET_get_image_data()`.

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 object in the PDF. Any rotation or mirror transformations applied to the placed image on the PDF page will not be applied to the pixel data, but the original pixel data will be extracted.

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;  
}
```

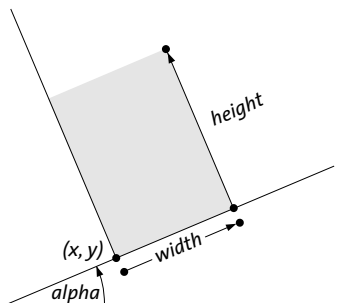


Fig. 7.1
Image geometry

```
        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.

7.3 Image Analysis

Image merging. 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 close to each other.

There are some common reasons for this 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.

TET's image merging engine detects this situation and recombines the image parts to form a larger and more useful image. Several conditions must be met in order for images to be considered as candidates for merging:

- ▶ The image fragments are oriented horizontally or vertically (but not at arbitrary angles), and form a rectangular grid of sub-images.
- ▶ The number of bits per component must be the same.
- ▶ The colorspace must be the same or compatible.
- ▶ Some combinations of colorspace and compression scheme (in particular, JPEG 2000 compression) prevent image merging.

If the merging candidates can be combined to a larger image, they will be merged.

Merged images can be identified as such by the *images[]/mergetype* pCOS pseudo object: it will have 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.

In order to completely disable image merging use the following page option:

```
imageanalysis={merge={disable}}
```



*Fig. 7.2
Although this
image consists of
many little strips,
TET will extract it
as a single reus-
able image.*

Small image filtering. TET ignores very small images if any of those is present on the page. Since the image merging process often combines many small images to a larger image, small image removal is performed after image merging. Only images which can not be merged to form a larger image will be candidates for small image removal. In addition, they must satisfy the conditions for size and count which can be specified in the *maxarea* and *maxcount* suboptions of the *smallimages* option of *TET_open_page()* and *TET_process_page()*.

In order to completely disable small image removal use the following page option:

```
imageanalysis={smallimages={disable}}
```

7.4 Restrictions and Caveats

Image color fidelity. TET does not degrade image quality when extracting images:

- ▶ Raster images are never downsampled.
- ▶ The color space of an image will be retained in the output. TET never applies any CMYK-to-RGB or similar color conversion.
- ▶ The number of color components will always be unchanged. For example, RGB images will not be changed to grayscale if they contain only gray colors.

Image workarounds. In some situations the color appearance of the extracted image may be different from the visual appearance of the PDF page. While the image shape is preserved, the colors may appear different because of the following reasons:

- ▶ Image masks are applied.
- ▶ Colorized grayscale images are extracted without the color, but as grayscale images.
- ▶ Since DeviceN color is not supported in TIFF, images with the DeviceN colorspace are extracted as grayscale, RGB, or CMYK images for N=1, 3, and 4, respectively. For N>4 CMYK TIFF images with one or more alpha channels are generated.
- ▶ Images with Separation colorspace are extracted as grayscale images. The spot color used to colorize the image will be lost.
- ▶ Images with Indexed ICCBased colorspace: the ICC profile will be ignored.

Unexpected results when extracting images. In some cases the shape of extracted images may appear different from 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.
- ▶ Since image masks are ignored, masking effects will not be reflected in the extracted image.

Unsupported image types. The following types of PDF images can not be extracted, i.e.

TET_write_image_file() will return -1 in these cases:

- ▶ PDF inline images
- ▶ Images with JBIG2 compression
- ▶ Images with Indexed Lab colorspace.



8 TET Markup Language (TETML)

8.1 Creating TETML

As an alternative to supplying the contents of a PDF document via a programming interface, TET can create XML output which represents the same information. We refer to the XML output created by TET as TET Markup Language (TETML). TETML contains the text content 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 will be 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 colorspace.

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 will create 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 15, 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 program demonstrates the canonical sequence for programmatically generating TETML. This sample program is available in all supported language bindings.

TETML output can be generated on a disk file or in memory. 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.

What's included in TETML? TETML output is encoded in UTF-8 (on zSeries with USS or MVS: EBCDIC-UTF-8, see www.unicode.org/reports/tr16), and includes the following information:

- ▶ general document information and metadata
- ▶ text contents of each page (words or paragraph)
- ▶ glyph information (font name, size, coordinates)
- ▶ structure information, e.g. tables
- ▶ information about placed images on the page
- ▶ resource information, i.e. fonts, colorspace, and images
- ▶ error messages if an exception occurred during PDF processing

Various elements and attributes in TETML are optional. See Section 8.2, »Controlling TETML Details«, page 92, for details.

TETML examples. The following shortened document shows the most important parts of a TETML document:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Created by the PDFlib Text Extraction Toolkit TET (www.pdflib.com) -->
<TET xmlns="http://www.pdflib.com/XML/TET3/TET-3.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.pdflib.com/XML/TET3/TET-3.0
    http://www.pdflib.com/XML/TET3/TET-3.0.xsd"
  version="3">
  <Creation platform="Win32" tetVersion="3.0" date="2008-10-15T22:04:36+02:00" />
  <Document filename="..\data\FontReporter.pdf" pageCount="9" filesize="128109"
    linearized="true" pdfVersion="1.6">
  <DocInfo>
    <Author>PDFlib GmbH</Author>
    <CreationDate>2008-07-08T15:05:39+00:00</CreationDate>
    <Creator>FrameMaker 7.0</Creator>
    <ModDate>2008-09-30T23:15:19+02:00</ModDate>
    <Producer>Acrobat Distiller 7.0.5 (Windows)</Producer>
    <Subject>PDFlib FontReporter</Subject>
    <Title>PDFlib FontReporter 1.3 Manual</Title>
  </DocInfo>
  <Metadata>
    <x:xmpmeta xmlns:x="adobe:ns:meta/" x:xmptk="Adobe XMP Core 4.2.1-c041 52.342996, 2008/
05/07-20:48:00"
      <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        ...XMP metadata...
      </rdf:RDF>
    </x:xmpmeta>
  </Metadata>
  <Options>engines={text=true image=true} tetml={} </Options>
  <Pages>
    <Page number="1" width="485" height="714">
    <Options>tetml={} granularity=word </Options>
    <Content granularity="word" font="false" geometry="false">
    <Para>
      <Word>
        <Text>PDFlib</Text>
        <Box llx="111.48" lly="636.33" urx="161.14" ury="654.33"/>
      </Word>
      <Word>
        <Text>GmbH</Text>
        <Box llx="165.06" lly="636.33" urx="214.84" ury="654.33"/>
      </Word>
      <Word>
        <Text>München</Text>
        <Box llx="218.75" lly="636.33" urx="292.23" ury="654.33"/>
      </Word>
      ...more page content...
    </Content>
  </Page>
  <Resources>
  <Fonts>
    <Font id="F0" name="Gen_TheSans-Plain" type="Type 1 CFF" embedded="true" />
    <Font id="F1" name="TheSansExtraBold-Plain" type="Type 1 CFF" embedded="true" />
    <Font id="F2" name="TheSans-Plain" type="Type 1 CFF" embedded="true" />
    <Font id="F3" name="TheSans-Italic" type="Type 1 CFF" embedded="true" />
    ...more fonts...
```

```

</Fonts>
<Images>
  <Image id="I0" width="595" height="792" colorspace="CS3" bitsPerComponent="8"/>
  <Image id="I1" width="595" height="792" colorspace="CS3" bitsPerComponent="8"/>
</Images>
<ColorSpaces>
  <ColorSpace id="CS0" name="DeviceGray" components="1" />
  <ColorSpace id="CS1" name="DeviceRGB" components="3" />
  <ColorSpace id="CS2" name="DeviceCMYK" components="4" />
  <ColorSpace id="CS3" name="Indexed" components="1" base="CS1" />
</ColorSpaces>
</Resources>
</Pages>
</Document>
</TET>

```

Depending on the selected TETML mode more details can be expressed in TETML. TETML modes are discussed in more detail in »Selecting the text mode«, page 92; here is a variation of the sample above with more glyph details. The *Glyph* element contains font and position information:

```

<Para>
  <Word>
    <Text>PDFlib</Text>
    <Box llx="111.48" lly="636.33" urx="161.14" ury="654.33">
      <Glyph font="F1" size="18" x="111.48" y="636.33" width="9.65">P</Glyph>
      <Glyph font="F1" size="18" x="121.12" y="636.33" width="11.88">D</Glyph>
      <Glyph font="F1" size="18" x="133.00" y="636.33" width="8.33">F</Glyph>
      <Glyph font="F1" size="18" x="141.33" y="636.33" width="4.88">l</Glyph>
      <Glyph font="F1" size="18" x="146.21" y="636.33" width="4.88">i</Glyph>
      <Glyph font="F1" size="18" x="151.08" y="636.33" width="10.06">b</Glyph>
    </Box>
  </Word>
  <Word>
    <Text>GmbH</Text>
    <Box llx="165.06" lly="636.33" urx="214.84" ury="654.33">
      <Glyph font="F1" size="18" x="165.06" y="636.33" width="12.06">G</Glyph>
      <Glyph font="F1" size="18" x="177.12" y="636.33" width="15.44">m</Glyph>
      <Glyph font="F1" size="18" x="192.56" y="636.33" width="10.06">b</Glyph>
      <Glyph font="F1" size="18" x="202.61" y="636.33" width="12.22">H</Glyph>
    </Box>
  </Word>
  <Word>
    <Text>München</Text>
    <Box llx="218.75" lly="636.33" urx="292.23" ury="654.33">
      <Glyph font="F1" size="18" x="218.75" y="636.33" width="15.77">M</Glyph>
      <Glyph font="F1" size="18" x="234.52" y="636.33" width="10.19">ü</Glyph>
      <Glyph font="F1" size="18" x="244.70" y="636.33" width="10.22">n</Glyph>
      <Glyph font="F1" size="18" x="254.92" y="636.33" width="7.52">c</Glyph>
      <Glyph font="F1" size="18" x="262.44" y="636.33" width="10.22">h</Glyph>
      <Glyph font="F1" size="18" x="272.66" y="636.33" width="9.34">e</Glyph>
      <Glyph font="F1" size="18" x="282.00" y="636.33" width="10.22">n</Glyph>
    </Box>
  </Word>

```

8.2 Controlling TETML Details

TETML text modes. TETML can be generated in various text modes which include different amounts of font and geometry information, and differ regarding the grouping of text into larger units (granularity). The text mode can be specified individually for each page. However, in most situations TETML files will contain the data for all pages in the same mode. The following text modes are available:

- ▶ *Glyph* mode is a low-level flavor which includes the text, font, and coordinates 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«, page 72).
- ▶ *Wordplus* mode is similar to *word* mode, but adds font and coordinate details for all glyphs in a word. This makes it possible to analyze font usage and track changes of font, font size, etc. within a word. Since *wordplus* is the only text 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 the 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 page content is known to be grouped into lines, or 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.

Table 8.1 lists the TETML elements which are present in the text modes.

Table 8.1 TETML elements in various text mode

text mode	structure	tables	text position	text details
<i>glyph</i>	–	–	–	Glyph
<i>word</i>	Para, Word	Table, Row, Cell	Box	–
<i>wordplus</i>	Para, Word	Table, Row, Cell	Box	Glyph
<i>line</i>	Para, Line	–	–	–
<i>page</i>	Para	Table, Row, Cell	–	–

Selecting the text mode. With the TET command-line tool (see Section 2.1, »Command-Line Options«, page 15) you can specify the desired page 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 text 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()*.
- ▶ For *granularity=glyph* or *word* you can additionally specify the amount of glyph details. Using the *geometry* and *font* suboptions you can omit some parts of the glyph information if you don't need it.

The following page option list generates TETML output in *wordplus* mode with all glyph details:

```
granularity=word glyphdetails={geometry=true font=true}
```

Table 8.2 summarizes the options for creating page modes.

Table 8.2 Creating TETML text modes with the TET library

text mode	granularity option of <i>TET_process_page()</i>	tetml option of <i>TET_process_page()</i>
<i>glyph</i>	granularity=glyph	tetml={glyphdetails={geometry=true font=true}}
<i>word</i>	granularity=word	–
<i>wordplus</i>	granularity=word	tetml={glyphdetails={geometry=true font=true}}
<i>line</i>	granularity=line	–
<i>page</i>	granularity=page	–

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 10.3.

Document-related options must be supplied to the *--docopt* command-line option or to the *TET_open_document()* function.

The *tetml* option 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={nodocxmp} }
```

The *engines* option enables or disables the text and image extraction engines. The following option list will process text contents, but disable image processing:

```
engines={noimage}
```

All document options which have been supplied when creating TETML will be recorded in the */TET/Document/Options* element unless disabled with the following document option:

```
tetml={ elements={nooptions} }
```

Page options for controlling TETML output. The complete description of page options can be found in Table 10.5. Page-related options must be supplied to the *--pageopt* command-line option or to the *TET_process_page()* function.

The *tetml* 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 coordinate details:

```
tetml={ glyphdetails={nogeometry font} }
```

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}
```

The following page option makes sense only for page mode. It changes the default separator character from linefeed to space:

```
contentanalysis={lineseparator=U+0020}
```

All page options which have been supplied when creating TETML will be 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 will generally try to repair or ignore the problem if possible, or throw an exception otherwise. However, when generating TETML output with TET PDF parsing problems will usually be reported as an *Exception* element in the 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 output.

Problems which prevent the generation of the TETML output file (e.g. no write permission for the output file) will still trigger an exception, and no valid TETML output will be created.

Table 8.3 TETML elements

TETML element	description
Attachment	For PDF attachments describes the contents in a nested Document element. For non-PDF attachments only the name will be listed, but no contents.
Attachments	Container of Attachment elements
Box	Describes the coordinates of a Word. A word may contain multiple Box elements, e.g. a hyphenated word which spans multiple lines of text, or a word which starts with a large character.
Cell	Describes the contents of a single table cell.
ColorSpace	Describes a PDF colorspace.
ColorSpaces	Container of ColorSpace elements
Content	Describes the page contents as a hierarchical structure.
Creation	Describes the date and operating system platform for the TET execution, plus the version number of TET.
DocInfo	Predefined and custom document info entries
Document	Describes general document information including PDF file name and size, PDF version number.
Encryption	Describes various security settings.
Exception	Contains the error message and number associated with an exception which was thrown by TET. The Exception element may replace other elements if not enough information can be extracted from the input because of malformed PDF data structures.
Font	Describes a font resource.
Fonts	Container of Font elements
Glyph	Describes font and geometry details for a single glyph. The element content holds the Unicode characters produced by this glyph. This can be more than one character, e.g. for ligatures. The Glyph elements for a word are grouped within one or more Box elements.
Image	Describes an image resource, i.e. the actual pixel array comprising the image.
Images	Container of Image elements
Line	Contains the text for a single line.
Metadata	Contains XMP metadata and can be associated to the document, a font, or an image.
Options	Contains the document or page options used for generating the TETML
Page	Contains the contents of a single page
Pages	Container of Page elements
Para	Contains the text comprising a single paragraph.
PlacedImage	Describes an instance of an image placed on the page.
Resources	Contains colorspace, font, and image resources
Row	Contains one or more table cells
Table	Contains one or more table rows.
TET	The root element
Text	The actual text for a word or other element.
Word	A single word

8.3 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:

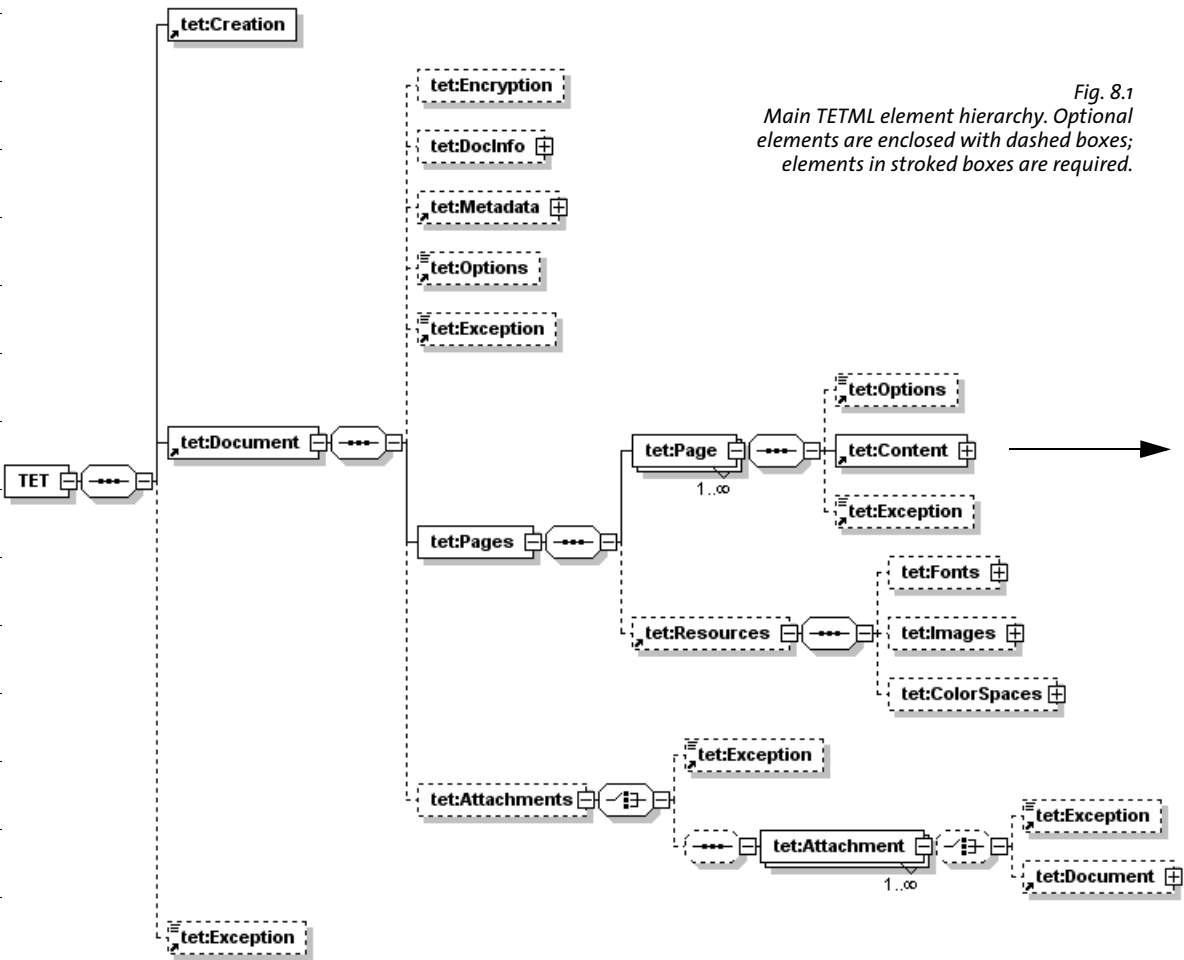
<http://www.pdflib.com/XML/TET3/TET-3.0>

The schema can be downloaded from the following URL on the Web:

<http://www.pdflib.com/XML/TET3/TET-3.0.xsd>

Both TETML namespace and schema location are present in the root element of each TETML document.

Table 8.3 describes the role of all TETML elements. Figure 8.1 visualizes the XML hierarchy of the top-level TETML elements. The hierarchy for the *Content* element is shown in Figure 8.2.



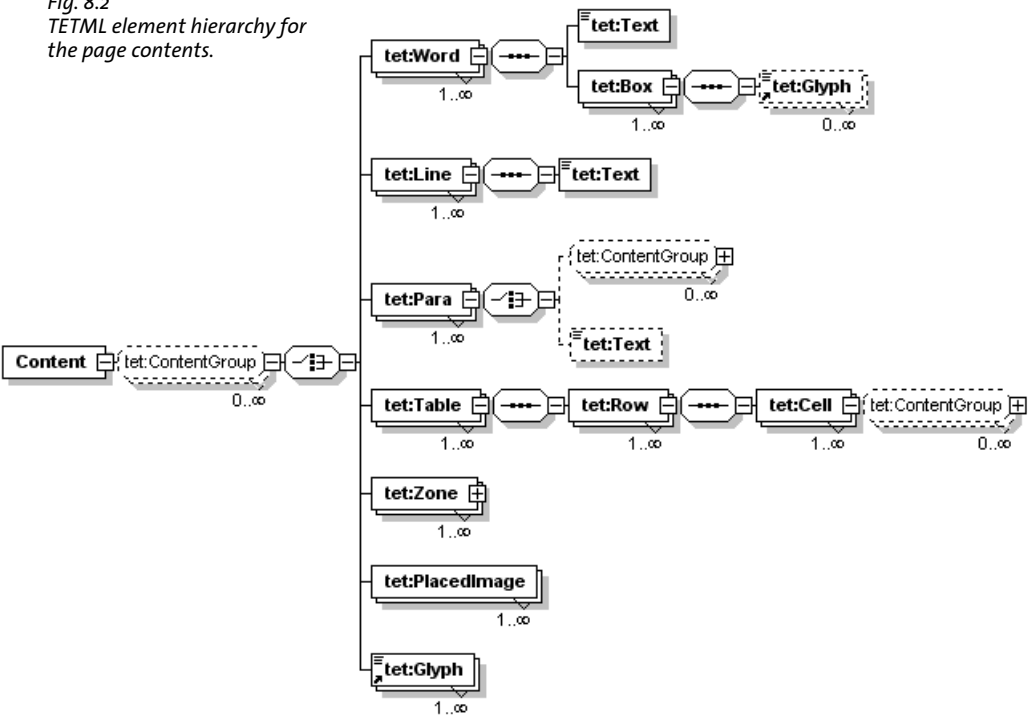
8.4 Transforming TETML with XSLT

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 (a TETML document 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 will use XSLT stylesheets to process TETML input in order to generate a new dataset (provided in text, XML, CSV, or HTML format) based on the input which in turn 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 8.1, »Creating TETML«, page 89.

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 wide variety of printed and Web resources on this topic. We restrict our samples to XSLT 1.0. Although XSLT 2.0 implementations are available, they are not yet in widespread use compared to XSLT 1.0. The XSLT 1.0 specification can be found at www.w3.org/TR/xslt.

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

Fig. 8.2
TETML element hierarchy for
the page contents.



running XSLT stylesheets, and lists common software for this purpose. In order 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 will 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 ships with the operating system since Windows 2000 SP4
- ▶ Microsoft's .NET Framework 2.0 XSLT implementation
- ▶ Saxon, which is available in free and commercial versions
- ▶ Xalan, an open-source project (available in C++ and Java implementations) hosted by the Apache foundation
- ▶ The open-source *libxslt* library of the GNOME project
- ▶ Sablotron, an open-source XSLT toolkit

XSLT on the command line. Applying XSLT stylesheets from the command-line provides a convenient development and testing environment. The examples below show how apply XSLT stylesheets on the command-line. All samples process the input file *FontReporter.tetml* with the stylesheet *tetml2html.xsl* while setting the XSLT parameter *toc-generate* (which is used in the stylesheet) to the value *0*, and send the generated output to *FontReporter.html*:

- ▶ The Java-based Saxon processor (see www.saxonica.com) can be used as follows:

```
java -jar saxon9.jar -o FontReporter.html FontReporter.tetml tetml2html.xsl 
toc-generate=0
```

- ▶ The *xsltproc* tool is included in most Linux distributions, see xmlsoft.org/XSLT. Use the following command to apply a stylesheet to a TETML document:

```
xsltproc --output FontReporter.html --param toc-generate 0 tetml2html.xsl 
FontReporter.tetml
```

- ▶ Xalan C++ provides a command-line tool which can be invoked as follows:

```
Xalan -o FontReporter.html -p toc-generate 0 FontReporter.tetml tetml2html.xsl
```

- ▶ On Windows systems with the MSXML parser you can use the free *msxsl.exe* program provided by Microsoft. The program (including source code) is available at the following location:

www.microsoft.com/downloads/details.aspx?familyid=2FB55371-C94E-4373-B0E9-DB4816552E41

Run the program as follows:

```
msxsl.exe FontReporter.tetml tetml2html.xsl -o FontReporter.html toc-generate=0
```

- ▶ On Windows systems with the .NET Framework 2.0 XSLT implementation you can use the free *nxslt.exe* program which is available from the following location:

www.xmllab.net/Products/nxslt/tabid/62/Default.aspx

Run the program as follows:

```
nxslt3.exe FontReporter.tetml tetml2html.xsl -o FontReporter.html toc-generate=0
```

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 will 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 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. You can also execute Java-based XSLT in the *ant* build tool without any coding. The *build.xml* file in the TET distribution contains XSLT tasks for all samples.
- ▶ .NET 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.
- ▶ All Windows-based programming languages which support COM automation can use the methods of the *MSXML2.DOMDocument* automation class supplied by the MSXML parser. This is demonstrated in the *runxslt.vbs* sample. Similar code can be used with other COM-enabled languages.

XSLT extensions are available for many other modern 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 or ASP.NET can make use of the COM or .NET interfaces mentioned above.
- ▶ Java-based Web servers can make use of the *javax.xml.transform* package.
- ▶ PHP-based Web servers can make use of the Sablotron processor, see www.php.net/manual/en/intro.xsl.php.

XSLT in the Web browser. XSLT transformations are also supported by most modern browsers. In order to instruct the browser to apply an XSLT stylesheet to a TETML document add a line with a suitable processing instruction after the first line of the TETML document containing the *xml* processing instruction and before the root element. You can then load it in the browser which will apply the stylesheet and display the resulting output (note that Internet Explorer requires the file name suffix *.xml* when processing files from the local disk):

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="tetml2html.xsl" version="1.0"?>
<TET xmlns="http://www.pdflib.com/XML/TET3/TET-3.0"
...
```

The browser will apply the XSLT stylesheet to the TETML document and then display the resulting text, HTML, or XML output. As an alternative, XSLT processing in the browser can also be initiated from JavaScript code.

With Firefox 2 and above you can supply parameters to the XSLT stylesheet with the *xslt-param* processing instruction:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="tetml2html.xsl" version="1.0"?>
<?xslt-param name="toc-generate" value="0"?>
<TET xmlns="http://www.pdflib.com/XML/TET3/TET-3.0"
...
```

8.5 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 8.4, »Transforming TETML with XSLT«, page 97 discusses many 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 be 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 text mode (e.g. *word* mode, see »TETML text modes«, page 92, 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 interesting elements within the *Attachments* element; the relevant *xsl:template* elements themselves don't have to be modified.
- ▶ All XSLT samples work with XSLT 1. While some samples could be simplified using features from XSLT 2, we wanted to stick to XSLT 1 for better usability.

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 207
font 107
of 100
a 92
in 83
and 75
fonts 64
PDF 60
FontReporter 58
...
```

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:

```
[TheSansBold-Plain/24] Contents
```

```
[TheSansBold-Plain/13.98] 1
[TheSansBold-Plain/13.98] Installing
[TheSansBold-Plain/13.98] PDFlib
[TheSansBold-Plain/13.98] FontReporter
[TheSansBold-Plain/13.98] 2
[TheSansBold-Plain/13.98] Working
[TheSansBold-Plain/13.98] with
[TheSansBold-Plain/13.98] FontReporter
[TheSansBold-Plain/13.98] A
[TheSansBold-Plain/13.98] Revision
[TheSansBold-Plain/13.98] History
[TheSansBold-Plain/24] 1
[TheSansBold-Plain/24] Installing
[TheSansBold-Plain/24] PDFlib
[TheSansBold-Plain/24] FontReporter
...
```

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.

TheSansExtraBold-Plain used on:

page 1:

(111, 636), (165, 636), (219, 636), (292, 636), (301, 636), (178, 603), (221, 603), (226, 603),
(272, 603), (277, 603), (102, 375), (252, 375), (261, 375), (267, 375)

TheSans-Plain used on:

page 1:

(102, 266), (119, 266), (179, 266), (208, 266), (296, 266), (346, 266), (367, 266)
...

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.

19894 total glyphs in the document; breakdown by font:

68.71% ThesisAntiqua-Normal: 13669 glyphs
22.89% TheSans-Italic: 4553 glyphs
6.38% TheSansBold-Plain: 1269 glyphs
0.9% TheSansMonoCondensed-Plain: 179 glyphs
0.49% TheSansBold-Italic: 98 glyphs
0.27% TheSansExtraBold-Plain: 54 glyphs
0.21% TheSerif-Caps: 42 glyphs
0.15% TheSans-Plain: 29 glyphs
0.01% Gen_TheSans-Plain: 1 glyphs

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 will be ignored.

Alphabetical list of words in the document along with their page number:

A
about 2 7 8
access 8 12
accessible 11
achieving 9 12
Acrobat 2 5 7 8 9 10 11 14 15 17
ActiveX 2
actual 9 12
actually 11 12 14
addition 9
Additional 12
additions 17
address 9 12
addressed 9
addressing 9
Adobe 2 5 8 12 14
...

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 will be processed recursively:

```
dc:creator = PDFlib GmbH  
xmp:CreatorTool = FrameMaker 7.0
```

Extract table contents in CSV format. The *table.xsl* stylesheet expects TETML input in *word*, *wordplus*, or *page* mode. It extracts the contents of a specified 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 the 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 creates heading elements (H1, H2, etc.) based on configurable font sizes. It also maps table elements in TETML to the corresponding HTML table constructs to visualize tables in the browser. The converter also creates a table of contents at the beginning of the HTML page, where each entry is based on some heading in the document, and contains an active link which jumps to the corresponding heading.

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 will be processed recursively.



9 The pCOS Interface

The pCOS (*PDFlib Comprehensive Object Syntax*) interface provides a simple and elegant facility for retrieving arbitrary information from all sections of a PDF document which do not describe page contents, such as page dimensions, metadata, interactive elements, etc. pCOS users are assumed to have some basic knowledge of internal PDF structures and dictionary keys, but do not have to deal with PDF syntax and parsing details.

We strongly recommend that pCOS users obtain a copy of the *PDF Reference*, which is available as follows:

Adobe Systems Incorporated: PDF Reference, Sixth Edition: Version 1.7. Downloadable PDF from www.adobe.com/devnet/pdf/pdf_reference.html.

9.1 Simple pCOS Examples

Cookbook A collection of pCOS coding fragments for solving specific problems can be found in the pCOS Cookbook.

Assuming a valid PDF document handle is available, the pCOS functions `TET_pcos_get_number()`, `TET_pcos_get_string()`, and `TET_pcos_get_stream()` can be used to retrieve information from a PDF using the pCOS path syntax. Table 9.1 lists some common pCOS paths and their meaning.

Table 9.1 pCOS paths for commonly used PDF objects

pCOS path	type	explanation
<code>length:pages</code>	<i>number</i>	<i>number of pages in the document</i>
<code>/Info/Title</code>	<i>string</i>	<i>document info field Title</i>
<code>/Root/Metadata</code>	<i>stream</i>	<i>XMP stream with the document's metadata</i>
<code>fonts[...]/name</code>	<i>string</i>	<i>name of a font; the number of entries can be retrieved with <code>length:fonts</code></i>
<code>fonts[...]/vertical</code>	<i>boolean</i>	<i>check a font for vertical writing mode</i>
<code>fonts[...]/embedded</code>	<i>boolean</i>	<i>embedding status of a font</i>
<code>pages[...]/width</code>	<i>number</i>	<i>width of the visible area of the page</i>

Number of pages. The total number of pages in a document can be queried as follows:

```
pagecount = tet.pcos_get_number(doc, "length:pages");
```

Document info fields. Document information fields can be retrieved with the following code sequence:

```
objtype = tet.pcos_get_string(doc, "type:/Info/Title");
if (objtype.equals("string"))
{
    /* Document info key found */
    title = tet.pcos_get_string(doc, "/Info/Title");
}
```

Page size. Although the *MediaBox*, *CropBox*, and *Rotate* entries of a page can directly be obtained via pCOS, they must be evaluated in combination in order to find the actual size of a page. Determining the page size is much easier with the *width* and *height* keys of the *pages* pseudo object. The following code retrieves the width and height of page 3 (note that indices for the *pages* pseudo object start at 0):

```
pagenum = 2
width = tet.pcos_get_number(doc, "pages[" + pagenum + "]/width");
height = tet.pcos_get_number(doc, "pages[" + pagenum + "]/height");
```

Listing all fonts in a document. The following sequence creates a list of all fonts in a document along with their embedding status:

```
fontcount = tet.pcos_get_number(doc, "length:fonts");

for (i=0; i < fontcount; i++)
{
    fontname = tet.pcos_get_string(doc, "fonts[" + i + "]/name");
    embedded = tet.pcos_get_number(doc, "fonts[" + i + "]/embedded");
}
```

Writing mode. Using pCOS and the *fontid* value provided in the *char_info* structure you can easily check whether a font uses vertical writing mode:

```
if (tet.pcos_get_number(doc, "fonts[" + ci->fontid + "]/vertical"))
{
    /* font uses vertical writing mode */
}
```

Encryption status. You can query the *pcosmode* pseudo object to determine the pCOS mode for the document:

```
if (tet.pcos_get_number(doc, "pcosmode") == 2)
{
    /* full pCOS mode */
}
```

XMP meta data. A stream containing XMP meta data can be retrieved with the following code sequence:

```
objtype = tet.pcos_get_string(doc, "type:/Root/Metadata");
if (objtype.equals("stream"))
{
    /* XMP meta data found */
    metadata = tet.pcos_get_stream(doc, "", "/Root/Metadata");
}
```

9.2 Handling Basic PDF Data Types

pCOS offers the three functions `TET_pcos_get_number()`, `TET_pcos_get_string()`, and `TET_pcos_get_stream()`. These can be used to retrieve all basic data types which may appear in PDF documents.

Numbers. Objects of type *integer* and *real* can be queried with `TET_pcos_get_number()`. pCOS doesn't make any distinction between integer and floating point numbers.

Names and strings. Objects of type *name* and *string* can be queried with `TET_pcos_get_string()`. Name objects in PDF may contain non-ASCII characters and the # syntax (decoration) to include certain special characters. pCOS deals with PDF names as follows:

- ▶ Name objects will be undecorated (i.e. the # syntax will be resolved) before they are returned.
- ▶ Name objects will be returned as Unicode strings in most language bindings. However, in the C and C++ language bindings they will be returned as UTF-8 without BOM.

Since the majority of strings in PDF are text strings `TET_pcos_get_string()` will treat them as such. However, in rare situations strings in PDF are used to carry binary information. In this case strings should be retrieved with the function `TET_pcos_get_stream()` which preserves binary strings and does not modify the contents in any way.

Booleans. Objects of type *boolean* can be queried with `TET_pcos_get_number()` and will be returned as 1 (true) or 0 (false). `TET_pcos_get_string()` can also be used to query boolean objects; in this case they will be returned as one of the strings *true* and *false*.

Streams. Objects of type *stream* can be queried with `TET_pcos_get_stream()`. Depending on the pCOS data type (*stream* or *fstream*) the contents will be compressed or uncompressed. Using the *keepfilter* option of `TET_pcos_get_stream()` the client can retrieve compressed data even for type *stream*.

Stream data in PDF may be preprocessed with one or more filters. The list of filters present at the stream can be queried from the stream dictionary; for images this information is much easier accessible in the image's *filterinfo* dictionary. If a stream's filter chain contains only supported filters its type will be *stream*. When retrieving the contents of a *stream* object, `TET_pcos_get_stream()` will remove all filters and return the resulting unfiltered data.

Note pCOS does not support the following stream filters: JBIG2 and JPX.

If there is at least one unsupported filter in a stream's filter chain, the object type will be reported as *fstream* (filtered stream). When retrieving the contents of an *fstream* object, `TET_pcos_get_stream()` will remove the supported filters at the beginning of a filter chain, but will keep the remaining unsupported filters and return the stream data with the remaining unsupported filters still applied. The list of applied filters can be queried from the stream dictionary, and the filtered stream contents can be retrieved with `TET_pcos_get_stream()`. Note that the names of supported filters will not be removed when querying the names of the stream's filters, so the client should ignore the names of supported filters.

Streams in PDF generally contain binary data. However, in rare cases (text streams) they may contain textual data instead (e.g. JavaScript streams). In order to trigger the appropriate text conversion, use the *convert=unicode* option in *TET_pcos_get_stream()*.

9.3 Composite Data Structures and IDs

Objects with one of the basic data types can be arranged in two kinds of composite data structures: arrays and dictionaries. pCOS does not offer specific functions for retrieving composite objects. Instead, the objects which are contained in a dictionary or array can be addressed and retrieved individually.

Arrays. Arrays are one-dimensional collections of any number of objects, where each object may have arbitrary type.

The contents of an array can be enumerated by querying the number *N* of elements it contains (using the *length* prefix in front of the array's path, see Table 9.2), and then iterating over all elements from index 0 to *N*-1.

Dictionaries. Dictionaries (also called associative arrays) contain an arbitrary number of object pairs. The first object in each pair has the type *name* and is called the key. The second object is called the value, and may have an arbitrary type except *null*.

The contents of a dictionary can be enumerated by querying the number *N* of elements it contains (using the *length* prefix in front of the dictionary's path, see Table 9.2), and then iterating over all elements from index 0 to *N*-1. Enumerating dictionaries will provide all dictionary keys in the order in which they are stored in the PDF using the *.key* suffix at the end of the dictionary's path. Similarly, the corresponding values can be enumerated with the *.val* suffix. Inherited values (see below) and pseudo objects will not be visible when enumerating dictionary keys, and will not be included in the *length* count.

Some page-related dictionary entries in PDF can be inherited across a tree-like data structure, which makes it difficult to retrieve them. For example the *MediaBox* for a page is not guaranteed to be contained in the page dictionary, but may be inherited from an arbitrarily complex page tree. pCOS eliminates this problem by transparently inserting all inherited keys and values into the final dictionary. In other words, pCOS users can assume that all inheritable entries are available directly in a dictionary, and don't have to search all relevant parent entries in the tree. This merging of inherited entries is only available when accessing the pages tree via the *pages[]* pseudo object; accessing the */Pages* tree, the *objects[]* pseudo object, or enumerating the keys via *pages[][]* will return the actual entries which are present in the respective dictionary, without any inheritance applied.

pCOS IDs for dictionaries and arrays. Unlike PDF object IDs, pCOS IDs are guaranteed to provide a unique identifier for an element addressed via a pCOS path (since arrays and dictionaries can be nested an object can have the same PDF object ID as its parent array or dictionary). pCOS IDs can be retrieved with the *pcosid* prefix in front of the dictionary's or array's path (see Table 9.2).

The pCOS ID can therefore be used as a shortcut for repeatedly accessing elements without the need for explicit path addressing. For example, this will improve performance when looping over all elements of a large array. Use the *objects[]* pseudo object to retrieve the contents of an element identified by a particular ID.

9.4 Path Syntax

The backbone of the pCOS interface is a simple path syntax for addressing and retrieving any object contained in a PDF document. In addition to the object data itself pCOS can retrieve information about an object, e.g. its type or length. Depending on the object's type (which itself can be queried) one of the functions *TET_pcos_get_number()*, *TET_pcos_get_string()*, and *TET_pcos_get_stream()* can be used to obtain the value of an object. The general syntax for pCOS paths is as follows:

```
[<prefix>:][pseudoname[<index>]]/<name>[<index>]/<name>[<index>] ... [.key|.val]
```

The meaning of the various path components is as follows:

- ▶ The optional *prefix* can attain the values listed in Table 9.2.
- ▶ The optional *pseudo object name* may contain one of the values described in Section 9.5, »Pseudo Objects«, page 112.
- ▶ The *name* components are dictionary keys found in the document. Multiple names are separated with a / character. An empty path, i.e. a single / denotes the document's Trailer dictionary. Each name must be a dictionary key present in the preceding dictionary. Full paths describe the chain of dictionary keys from the initial dictionary (which may be the Trailer or a pseudo object) to the target object.
- ▶ Paths or path components specifying an array or dictionary can have a numerical index which must be specified in decimal format between brackets. Nested arrays or dictionaries can be addressed with multiple index entries. The first entry in an array or dictionary has index 0.
- ▶ Paths or path components specifying a dictionary can have an index qualifier plus one of the suffixes *.key* or *.val*. This can be used to retrieve a particular dictionary key or the corresponding value of the indexed dictionary entry, respectively. If a path for a dictionary has an index qualifier it must be followed by one of these suffixes.

Encoding for pCOS paths. In most cases pCOS paths will contain only plain ASCII characters. However, in a few cases (e.g. PDFlib Block names) non-ASCII characters may be required. pCOS paths must be encoded according to the following rules:

- ▶ When a path component contains any of the characters */*, *[*, *]*, or *#*, these must be expressed by a number sign *#* followed by a two-digit hexadecimal number.
- ▶ In Unicode-aware language bindings the path consists of a regular Unicode string which may contain ASCII and non-ASCII characters.
- ▶ In non-Unicode-aware language bindings the path must be supplied in UTF-8. The string may or may not contain a BOM, but this doesn't make any difference. A BOM may be placed at the start of the path, or at the start of individual path components (i.e. after a slash character).

On EBCDIC systems the path must generally be supplied in *ebcdic* encoding. Characters outside the ASCII character set must be supplied as EBCDIC-UTF-8 (with or without BOM).

Path prefixes. Prefixes can be used to query various attributes of an object (as opposed to its actual value). Table 9.2 lists all supported prefixes.

The *length* prefix and content enumeration via indices are only applicable to plain PDF objects and pseudo objects of type *array*, but not any other pseudo objects. The *pcosid* prefix cannot be applied to pseudo objects. The *type* prefix is supported for all pseudo objects.

Table 9.2 *pCOS* path prefixes

prefix	explanation
length	(Number) Length of an object, which depends on the object's type: array Number of elements in the array dict Number of key/value pairs in the dictionary stream Number of key/value pairs in the stream dict (not the stream length; use the Length key to determine the length of stream data in bytes) fstream Same as stream other 0
pcosid	(Number) Unique <i>pCOS</i> ID for an object of type dictionary or array. If the path describes an object which doesn't exist in the PDF the result will be -1. This can be used to check for the existence of an object, and at the same time obtaining an ID if it exists.
type	(String or number) Type of the object as number or string: 0, null Null object or object not present (use to check existence of an object) 1, boolean Boolean object 2, number Integer or real number 3, name Name object 4, string String object 5, array Array object 6, dict Dictionary object (but not stream) 7, stream Stream object which uses only supported filters 8, fstream Stream object which uses one or more unsupported filters Enums for these types are available for the convenience of C and C++ developers.

9.5 Pseudo Objects

Pseudo objects extend the set of pCOS paths by introducing some useful elements which can be used as an abbreviation for information which is present in the PDF, but cannot easily be accessed by reading a single value. The following sections list all supported pseudo objects. Pseudo objects of type *dict* can not be enumerated.

Universal pseudo objects. Universal pseudo objects are always available, regardless of encryption and passwords. This assumes that a valid document handle is available, which may require setting the option *requiredmode* suitably when opening the document. Table 9.3 lists all universal pseudo objects.

Table 9.3 Universal pseudo objects

object name	explanation
encrypt	(Dict) Dictionary with keys describing the encryption status of the document:
length	(Number) Length of the encryption key in bits
algorithm	(Number)
description	(String) Encryption algorithm number or description:
	-1 Unknown encryption
	0 No encryption
	1 40-bit RC4 (Acrobat 2-4)
	2 128-bit RC4 (Acrobat 5)
	3 128-bit RC4 (Acrobat 6)
	4 128-bit AES (Acrobat 7)
	5 Public key on top of 128-bit RC4 (Acrobat 5) (unsupported)
	6 Public key on top of 128-bit AES (Acrobat 7) (unsupported)
	7 Adobe Policy Server (Acrobat 7) (unsupported)
	8 Adobe Digital Editions (EBX) (unsupported)
master	(Boolean) True if the PDF requires a master password to change security settings (permissions, user or master password), false otherwise
user	(Boolean) True if the PDF requires a user password for opening, false otherwise
noaccessible, noannots, noassemble, nocopy, noforms, nohiresprint, nomodify, noprint	(Boolean) True if the respective access protection is set, false otherwise
plainmetadata	(Boolean) True if the PDF contains unencrypted meta data, false otherwise
extension-level	(String) Adobe Extension Level based on ISO 32000, or 0 if no extension level is present. Acrobat 9 creates documents with extension level 3.
filename	(String) Name of the PDF file.
filesize	(Number) Size of the PDF file in bytes
fullpdf-version	(Number) Numerical value for the PDF version number. The numbers increase monotonically for each PDF/Acrobat version. The value $100 * \text{BaseVersion} + \text{ExtensionLevel}$ will be returned, e.g.
	150 PDF 1.5 (Acrobat 6)
	160 PDF 1.6 (Acrobat 7)
	170 PDF 1.7 (Acrobat 8)
	173 PDF 1.7 Adobe Extension Level 3 (Acrobat 9)
linearized	(Boolean) True if the PDF document is linearized, false otherwise
major	(Number) Major, minor, or revision number of the library, respectively.
minor	
revision	

Table 9.3 Universal pseudo objects

object name	explanation
pcosinterface	(Number) Interface number of the underlying pCOS implementation. This specification describes interface number 3. The following table details which product versions implement various pCOS interface numbers:
	1 TET 2.0, 2.1
	2 pCOS 1.0
	3 PDFlib+PDI 7, PPS 7, TET 2.2, pCOS 2.0, PLOP 3.0, TET 2.3
	4 PLOP 4.0, TET 3.0
pcosmode	(Number/string) pCOS mode as number or string:
pcos-modename	0 minimum
	1 restricted
	2 full
pdfversion	(Number) PDF version number multiplied by 10, e.g. 16 for PDF 1.6
pdfversion-string	(String) Full PDF version string in the form expected by various API functions for setting the PDF output compatibility, e.g. 1.5, 1.6, 1.7, 1.7ext3
shrug	(Boolean) True if and only if security settings were ignored when opening the PDF document; the client must take care of honoring the document author's intentions. For TET the value will be true, and content extraction will be allowed, if all of the following conditions are true: <ul style="list-style-type: none">► Shrug mode has been enabled with the shrug option.► The document has a master password but this has not been supplied.► The user password (if required for the document) has been supplied.► Content extraction is not allowed in the document's permission settings.
version	(String) Full library version string in the format <major>.<minor>.<revision>, possibly suffixed with additional qualifiers such as beta, rc, etc.

Pseudo objects for PDF objects, pages, and interactive elements. Table 9.4 lists pseudo objects which can be used for retrieving object or page information, or serve as short-cuts for various interactive elements.

Table 9.4 Pseudo objects for PDF objects, pages, and interactive elements

object name	explanation
articles	(Array of dicts) Array containing the article thread dictionaries for the document. The array will have length 0 if the document does not contain any article threads. In addition to the standard PDF keys pCOS supports the following pseudo key for dictionaries in the articles array: beads (Array of dicts) Bead directory with the standard PDF keys, plus the following: destpage (Number) Number of the target page (first page is 1)
bookmarks	(Array of dicts) Array containing the bookmark (outlines) dictionaries for the document. In addition to the standard PDF keys pCOS supports the following pseudo keys for dictionaries in the bookmarks array: level (Number) Indentation level in the bookmark hierarchy destpage (Number) Number of the target page (first page is 1) if the bookmark points to a page in the same document, -1 otherwise.
fields	(Array of dicts) Array containing the form fields dictionaries for the document. In addition to the standard PDF keys in the field dictionary and the entries in the associated Widget annotation dictionary pCOS supports the following pseudo keys for dictionaries in the fields array: level (Number) Level in the field hierarchy (determined by « as separator) fullname (String) Complete name of the form field. The same naming conventions as in Acrobat 7 will be applied.
names	(Dict) A dictionary where each entry provides simple access to a name tree. The following name trees are supported: AP, AlternatePresentations, Dests, EmbeddedFiles, IDS, JavaScript, Pages, Renditions, Templates, URLS. Each name tree can be accessed by using the name as a key to retrieve the corresponding value, e.g.: names/Dests[0].key retrieves the name of a destination names/Dests[0].val retrieves the corresponding destination dictionary In addition to standard PDF dictionary entries the following pseudo keys for dictionaries in the Dests names tree are supported: destpage (number) Number of the target page (first page is 1) if the destination points to a page in the same document, -1 otherwise. In order to retrieve other name tree entries these must be queried directly via /Root/Names/Dests etc. since they are not present in the name tree pseudo objects.
objects	(Array) Address an element for which a pCOS ID has been retrieved earlier using the pcoid prefix. The ID must be supplied as array index in decimal form; as a result, the PDF object with the supplied ID will be addressed. The length prefix cannot be used with this array.

Table 9.4 Pseudo objects for PDF objects, pages, and interactive elements

object name	explanation
pages	(Array of dicts) Each array element addresses a page of the document. Indexing it with the decimal representation of the page number minus one addresses that page (the first page has index 0). Using the length prefix the number of pages in the document can be determined. A page object addressed this way will incorporate all attributes which are inherited via the /Pages tree. The entries /MediaBox and /Rotate are guaranteed to be present. In addition to standard PDF dictionary entries the following pseudo entries are available for each page: colorspaces, extgstates, fonts, images, patterns, properties, shadings, templates (Arrays of dicts) Page resources according to Table 9.5. annots (Array of dicts) In addition to the standard PDF keys in the Annots array pCOS supports the following pseudo key for dictionaries in the annots array: destpage (Number; only for Subtype=Link and if a Dest entry is present) Number of the target page (first page is 1) blocks (Array of dicts) Shorthand for pages[]/PieceInfo/PDFlib/Private/Blocks[], i.e. the page's block dictionary. In addition to the existing PDF keys pCOS supports the following pseudo key for dictionaries in the blocks array: rect (Rectangle) Similar to Rect, except that it takes into account any relevant CropBox/MediaBox and Rotate entries and normalizes coordinate ordering. height (Number) Height of the page. The MediaBox or the CropBox (if present) will be used to determine the height. Rotate entries will also be applied. isempty (Boolean) True if the page is empty, and false if the page is not empty label (String) The page label of the page (including any prefix which may be present). Labels will be displayed as in Acrobat. If no label is present (or the PageLabel dictionary is malformed), the string will contain the decimal page number. Roman numbers will be created in Acrobat's style (e.g. VI), not in classical style which is different (e.g. XLV). If /Root/PageLabels doesn't exist, the document doesn't contain any page labels. width (Number) Width of the page (same rules as for height) The following entries will be inherited: CropBox, MediaBox, Resources, Rotate.
pdfa	(String) PDF/A conformance level of the document (e.g. PDF/A-1a:2005) or none
pdfx	(String) PDF/X conformance level of the document (e.g. PDF/X-1a:2001) or none
tagged	(Boolean) True if the PDF document is tagged, false otherwise

Pseudo objects for simplified resource handling. Resources are a key concept for managing various kinds of data which are required for completely describing the contents of a page. The resource concept in PDF is very powerful and efficient, but complicates access with various technical concepts, such as recursion and resource inheritance. pCOS greatly simplifies resource retrieval and supplies several groups of pseudo objects which can be used to directly query resources. Some of these pseudo resource dictionaries contain entries in addition to the standard PDF keys in order to further simplify resource information retrieval. pCOS pseudo resources reflect resources from the user's point of view, and differ from native PDF resources:

- ▶ Some entries may have been added (e.g. inline images, simple color spaces) or deleted (e.g. listed fonts which are not used on any page).
- ▶ In addition to the original PDF dictionary keys resource dictionaries may contain some user-friendly keys for auxiliary information (e.g. embedding status of a font, number of components of a color space).

pCOS supports two groups of pseudo objects for resource retrieval. Global resource arrays contain all resources of a given type in a PDF document, while page-based resources contain only the resources used by a particular page. The corresponding pseudo arrays are available for all resource types listed in Table 9.5:

- ▶ A list of all resources in the document is available in the global resource array (e.g. *images[]*). Retrieving the length of one of the global resource pseudo arrays results in a resource scan (see below) for all pages.
- ▶ A list of resources on each page is available in the page-based resource array (e.g. *pages[]/images[]*). Accessing the length of one of a page's resource pseudo arrays results in a resource scan for that page (to collect all resources which are actually used on the page, and to merge images on that page).

A *resource scan* is a full scan of the page including Unicode mapping and image merging, but excluding Wordfinder operation. Applications which require a full resource listing and all page contents are recommended to process all pages before querying resource pseudo objects in order to avoid the resource scans in addition to the regular page scans.

Table 9.5 Pseudo objects for resources; each resource category *P* creates two resource arrays *P*[] and pages[]/*P*[].

object name	explanation
colorspaces	(Array of dicts) Array containing dictionaries for all color spaces on the page or in the document. In addition to the standard PDF keys in color space and ICC profile stream dictionaries the following pseudo keys are supported: alternateid (Integer; only for name=Separation and DeviceN) Index of the underlying alternate color space in the colorspaces[] pseudo object. baseid (Integer; only for name=Indexed) Index of the underlying base color space in the colorspaces[] pseudo object. colorantname (Name; only for name=Separation) Name of the colorant. Non-ASCII CJK color names will be converted to Unicode. colorantnames (Array of names; only for name=DeviceN) Names of the colorants components (Integer) Number of components of the color space name (String) Name of the color space: CalGray, CalRGB, DeviceCMYK, DeviceGray, DeviceN, DeviceRGB, ICCBased, Indexed, Lab, Separation csarray (Array; not for name=DeviceGray/RGB/CMYK) Array describing the underlying native color space, i.e. the original color space object in the PDF. Color space resources will include all color spaces which are referenced from any type of object, including the color spaces which do not require native PDF resources (i.e. DeviceGray, DeviceRGB, and DeviceCMYK).
extgstates	(Array of dicts) Array containing the dictionaries for all extended graphics states (ExtGStates) on the page or in the document
fonts	(Array of dicts) Array containing dictionaries for all fonts on the page or in the document. In addition to the standard PDF keys in font dictionaries, the following pseudo keys are supported: name (String) PDF name of the font without any subset prefix. Non-ASCII CJK font names will be converted to Unicode. embedded (Boolean) Embedding status of the font type (String) Font type: (unknown), Composite, Multiple Master, OpenType, TrueType, TrueType (CID), Type 1, Type 1 (CID), Type 1 CFF, Type 1 CFF (CID), Type 3 vertical (Boolean) true for fonts with vertical writing mode, false otherwise

Table 9.5 Pseudo objects for resources; each resource category *P* creates two resource arrays *P*[] and pages[]/*P*[].

object name	explanation
images	<p>(Array of dicts) Array containing dictionaries for all images on the page or in the document. The TET product will add merged (artificial) images to the images[] array.</p> <p>In addition to the standard PDF keys the following pseudo keys are supported:</p> <p>bpc (Integer) The number of bits per component. This entry is usually the same as BitsPerComponent, but unlike this it is guaranteed to be available. For JPEG2000 images it may be -1 since the number of bits per component may not be available in the PDF structures.</p> <p>colorspaceid (Integer) Index of the image's color space in the colorspace[] pseudo object. This can be used to retrieve detailed color space properties. For JPEG 2000 images the color space id may be -1 since the color space may not be encoded in the PDF structures.</p> <p>filterinfo (Dict) Describes the remaining filter for streams with unsupported filters or when retrieving stream data with the keepfilter option set to true. If there is no such filter no filterinfo dictionary will be available. The dictionary contains the following entries:</p> <p>name (Name) Name of the filter</p> <p>supported (Boolean) True if the filter is supported</p> <p>decodeparms (Dict) The DecodeParms dictionary if one is present for the filter</p> <p>mergetype (Integer) The following types describe the status of the image:</p> <p>0 (normal) The image corresponds to an image in the PDF.</p> <p>1 (artificial) The image is the result of merging multiple consumed images (i.e. images with mergetype=2) into a single image. The resulting artificial image does not exist in the PDF data structures as an object.</p> <p>2 (consumed) The image should be ignored since it has been merged into a larger image. Although the image exists in the PDF, it usually should not be extracted because it is part of an artificial image (i.e. an image with mergetype=1).</p> <p>This entry reflects information regarding all pages processed so far. It may change its value while processing other pages in the document. If final (constant) information is required, all pages in the document must have been processed, or the value of the pCOS path length:images must have been retrieved.</p>
patterns	(Array of dicts) Array containing dictionaries for all patterns on the page or in the document
properties	(Array of dicts) Array containing dictionaries for all properties on the page or in the document
shadings	<p>(Array of dicts) Array containing dictionaries for all shadings on the page or in the document. In addition to the standard PDF keys in shading dictionaries the following pseudo key is supported:</p> <p>colorspaceid (Integer) Index of the underlying color space in the colorspace[] pseudo object.</p>
templates	(Array of dicts) Array containing dictionaries for all templates (Form XObjects) on the page or in the document

9.6 Encrypted PDF Documents

pCOS supports encrypted and unencrypted PDF documents as input. However, full object retrieval for encrypted documents requires the appropriate master password to be supplied when opening the document. Depending on the availability of user and master password, encrypted documents can be processed in one of the pCOS modes described below.

See Section 5.1, »Indexing protected PDF Documents«, page 49, for information on indexing protected documents without supplying the master password.

Full pCOS mode (mode 2). Encrypted PDFs can be processed without any restriction provided the master password has been supplied upon opening the file. All objects will be returned unencrypted. Unencrypted documents will always be opened in full pCOS mode.

Restricted pCOS mode (mode 1). If the document has been opened without the appropriate master password and does not require a user password (or the user password has been supplied) pCOS operations are subject to the following restriction: The contents of objects with type *string*, *stream*, or *fstream* can not be retrieved with the following exceptions:

- ▶ The objects */Root/Metadata* and */Info/** (document info keys) can be retrieved if *nocopy=false* or *plainmetadata=true*.
- ▶ The objects *bookmarks[...]/Title* and *pages[...]/annots/Contents* (bookmark and annotation contents) can be retrieved if *nocopy=false*, i.e. if text extraction is allowed for the main text on the pages.

Note See Section 5.1, »Indexing protected PDF Documents«, page 49, for information on retrieving text in restricted pCOS mode under certain conditions.

Minimum pCOS mode (mode 0). Regardless of the encryption status and the availability of passwords, the universal pCOS pseudo objects listed in Table 9.3 are always available. For example, the *encrypt* pseudo object can be used to query a document's encryption status. Encrypted objects can not be retrieved in minimum pCOS mode.

Table 9.6 lists the resulting pCOS modes for various password combinations. Depending on the document's encryption status and the password supplied when opening the file, PDF object paths may be available in minimum, restricted, or full pCOS mode. Trying to retrieve a pCOS path which is inappropriate for the respective mode will raise an exception.

Table 9.6 Resulting pCOS modes for various password combinations

If you know...	...pCOS will run in...
none of the passwords	restricted pCOS mode if no user password is set, minimum pCOS mode otherwise
only the user password	restricted pCOS mode
the master password	full pCOS mode



10 TET Library API Reference

10.1 Option Lists

Option lists are a powerful yet easy method to control TET operations. Instead of requiring a multitude of function parameters, many API methods support option lists, or optlists for short. Options lists are strings which may contain an arbitrary number of options. Since option lists will be 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. Optlists support various data types and composite data like arrays. In most languages optlists can easily be constructed by concatenating the required keywords and values. C programmers may want to use the *sprintf()* function in order to construct optlists.

An optlist is a string containing one or more pairs of the form

```
name value
```

Names and values, as well as multiple name/value pairs can be separated by arbitrary whitespace characters (space, tab, carriage return, newline). The value may consist of a list of multiple values. You can also use an equal sign '=' between name and value:

```
name=value
```

Simple values. Simple values may use any of the following data types:

- ▶ Boolean: *true* or *false*; if the value of a boolean option is omitted, the value *true* is assumed. As a shorthand notation *nofoo* can be used instead of *foo=false* to disable option *foo*.
- ▶ String: these are plain ASCII strings which are generally used for non-localizable keywords. Strings containing whitespace or '=' characters must be bracketed with { and }. An empty string can be constructed with {}. The characters {, }, and \ must be preceded by an additional \ character if they are supposed to be part of the string.
- ▶ Strings and name strings: these can hold Unicode content in various formats; see Section 3.2, »C Binding«, page 22 for C- and C++-specific details regarding name strings.
- ▶ Unichar: these are single Unicode characters, where several syntax variants are supported: decimal values (e.g. 173), hexadecimal values prefixed with x, X, ox, oX, or U+ (xAD, oxAD, U+oAAD), numerical or character references (see below), but without the '&' and ';' decoration (shy, #xAD, #173). Alternatively, literal characters can be supplied. Unichars must be in the range 0-65535 (0-xFFFF).
- ▶ Keyword: one of a predefined list of fixed keywords
- ▶ Float and integer: decimal floating point or integer numbers; point and comma can be used as decimal separators for floating point values. Integer values can start with x, X, ox, or oX to specify hexadecimal values. Some options (this is stated in the respective function description) support percentages by adding a % character directly after the value.
- ▶ Handle: several internal object handles, e.g., document or page handles. Technically these are integer values.

Depending on the type and interpretation of an option additional restrictions may apply. For example, integer or float options may be restricted to a certain range of values;

handles must be valid for the corresponding type of object, etc. Some examples for simple values (the first line shows a password string containing a blank character):

TET `open_document()`: password {secret string}
TET `open_document()`: lineseparator={ CRLF }

List values. List values consist of multiple values, which may be simple values or list values in turn. Lists are bracketed with { and }. Example:

TET `set_option()`: searchpath={/usr/lib/tet d:\tet}

Note The backslash \ character requires special handling in many programming languages

Rectangles. A rectangle is a list of four float values specifying the coordinates of the lower left and upper right corners of a rectangle. Rectangle coordinates will be interpreted in the standard or user coordinate system (see Section 6.3, »Page and Text Geometry«, page 64). Example:

TET `open_page()`: includebox = {{0 0 500 100} {0 500 500 600}}

Character references in option lists. Some environments require the programmer to write source code in 8-bit encodings. This makes it cumbersome to include isolated Unicode characters in 8-bit encoded text without changing all characters in the text to multi-byte encoding. In order to aid developers in this situation, TET supports character references, a method known from markup languages such as SGML and HTML.

TET supports all numeric character references and character entity references defined in HTML 4.0, but in option lists they must be used without the '&' and ';' decoration. Numeric character references can be supplied in decimal or hexadecimal notation for the character's Unicode value. The following are examples for valid character references along with a description of the resulting character:

#173	soft hyphen
#xAD	soft hyphen
shy	soft hyphen

In addition to the HTML-style references above TET supports the custom character entity names for control characters (see Table 10.1).

Table 10.1 Custom character entity names for control characters

Unicode character (VB equivalents)	custom entity name	Unicode character (VB equivalents)	custom entity name
U+0020	SP, space	U+00AD	SHY, shy
U+00A0	NBSP, nbsp	U+000B	VT, verctab
		U+2028	LS, linesep
U+0009 (VbTab)	HT, hortab	U+000A (VbLf)	LF, linefeed
		U+000D (VbCr)	CR, return
		U+000D and U+000A (VbCrLf)	CRLF
		U+0085	NEL, newline
		U+2029	PS, parasep
U+002D	HY, hyphen	U+000C (VbFormFeed)	FF, formfeed

10.2 General Functions

<i>Perl PHP</i>	<i>resource TET_new()</i>
<i>C</i>	<i>TET *TET_new(void)</i>
	Create a new TET object.
<i>Returns</i>	A handle to a TET object to be used in subsequent calls. If this function doesn't succeed due to unavailable memory it will return NULL.
<i>Bindings</i>	This function is not available in object-oriented language bindings since it is hidden in the TET constructor.
<i>Java</i>	<i>void delete()</i>
<i>C#</i>	<i>void Dispose()</i>
<i>Perl PHP</i>	<i>resource TET_delete(resource tet)</i>
<i>C</i>	<i>void TET_delete(TET *tet)</i>
	Delete a TET object and release all related internal resources.
<i>Details</i>	Deleting a TET object automatically closes all of its open documents. The TET object must no longer be used in any function after it has been deleted.
<i>Bindings</i>	In object-oriented language bindings this function 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 <i>Dispose()</i> should be called at the end of processing to clean up unmanaged resources.
<i>C++</i>	<i>string utf8_to_utf16(string utf8string, string ordering)</i>
<i>Perl PHP</i>	<i>string TET_utf8_to_utf16(resource tet, string utf8string, string ordering)</i>
<i>C</i>	<i>const char *TET_utf8_to_utf16(TET *tet, const char *utf8string, const char *ordering, int *size)</i>
	Convert a string from UTF-8 format to UTF-16.
	utf8string String to be converted. It must contain a valid UTF-8 sequence (on EBCDIC platforms it must be encoded in EBCDIC). If a Byte Order Mark (BOM) is present, it will be removed.
	ordering Specifies the byte ordering of the result string: <ul style="list-style-type: none">▶ <i>utf16</i> or an empty string: The converted string will not have a BOM, and will be stored in the platform's native byte order.▶ <i>utf16le</i>: The converted string will be formatted in little endian format, and will be prefixed with the LE BOM (<i>\xFF\xFE</i>).▶ <i>utf16be</i>: The converted string will be formatted in big endian format, and will be prefixed with the BE BOM (<i>\xFE\xFF</i>).
	size (C language binding only) Pointer to a memory location where the length of the returned string (in bytes, but excluding the terminating two null bytes) will be stored.
<i>Returns</i>	The converted UTF-16 string. In C it will be terminated by two null bytes. The returned string is valid until the next call to any function other than <i>TET_utf16_to_utf8()</i> , <i>TET_</i>

`utf8_to_utf16()`, or `TET_utf32_to_utf16()` or until an exception is thrown. Clients must copy the string if they need it longer.

Bindings This function is not available in Unicode-capable language bindings. The memory used for the converted string will be managed by TET, and must not be freed by the client.

C++ `string utf16_to_utf8(string utf16string)`

Perl PHP `string TET_utf16_to_utf8(resource tet, string utf16string)`

C `const char *TET_utf16_to_utf8(TET *tet, const char *utf16string, int len, int *size)`

Convert a string from UTF-16 format to UTF-8.

utf16string The string to be converted. A Byte Order Mark (BOM) in the string will be interpreted. If it is missing the platform's native byte ordering is assumed.

len (C language binding only) Length of `utf16string` in bytes.

size (C language binding only) C-style pointer to a memory location where the length of the returned string (in bytes) will be stored. If the pointer is NULL it will be ignored.

Returns The converted UTF-8 string. The generated UTF-8 string will start with the UTF-8 BOM (`\xEF\xBB\xBF`). On EBCDIC platforms the conversion result including the BOM will finally be converted to EBCDIC. The returned string is valid until the next call to any function other than `TET_utf16_to_utf8()`, `TET_utf8_to_utf16()`, or `TET_utf32_to_utf16()` or until an exception is thrown. Clients must copy the string if they need it longer.

Bindings This function is not available in Unicode-capable language bindings.

C++ `string utf32_to_utf16(string utf32string, string ordering)`

Perl PHP `string TET_utf32_to_utf16(resource p, string utf32string, string ordering)`

C `const char *TET_utf32_to_utf16(TET *tet, const char *utf32string, int len, const char *ordering, int *size)`

Convert a string from UTF-32 format to UTF-16.

utf32string The string to be converted, which must contain a valid UTF-32 sequence. If a Byte Order Mark (BOM) is present, it will be interpreted

len (C language binding only) Length of `utf32string` in bytes.

ordering Specifies the byte ordering of the result string:

- ▶ `utf16` or an empty string: the converted string will not have any BOM, and will be stored in the platform's native byte order.
- ▶ `utf16le`: the converted string will be formatted in little endian format, and will be prefixed with the little-endian BOM (`\xFF\xFE`).
- ▶ `utf16be`: the converted string will be formatted in big endian format, and will be prefixed with the big-endian BOM (`\xFE\xFF`).

size (C language binding only) C-style pointer to a memory location where the length of the returned string (in bytes) will be stored.

Returns The converted UTF-16 string. The returned string is valid until the next call to any function other than `TET_utf16_to_utf8()`, `TET_utf8_to_utf16()`, or `TET_utf32_to_utf16()` or until an exception is thrown. Clients must copy the string if they need it longer.

Scope any

Bindings This function is not available in Unicode-capable language bindings.

C++	<code>void create_pvf(string filename, const void *data, size_t size, string optlist)</code>
C# Java	<code>void create_pvf(String filename, byte[] data, String optlist)</code>
Perl PHP	<code>TET_create_pvf(resource tet, string filename, string data, string optlist)</code>
VB	<code>Sub create_pvf(filename As String, data, optlist As String)</code>
C	<code>void TET_create_pvf(TET *tet, const char *filename, int len, const void *data, size_t size, const char *optlist)</code>

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 language binding 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 COM this is a variant of byte containing the data comprising the virtual file. In C and C++ this is a pointer to a memory location. In Java this is a byte array. In Perl and PHP this is a string.

size (C and C++ only) The length in bytes of the memory block containing the data.

optlist An option list according to Table 10.2. The following option can be used: *copy*

Details The virtual file name can be supplied to any API function which uses input files. Some of these functions may set a lock on the virtual file until the data is no longer needed. Virtual files will be 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 will be thrown. This function 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 10.2 Options for `TET_create_pvf()`

option	description
copy	(Boolean) TET will immediately create an internal copy of the supplied data. In this case the caller may dispose of the supplied data immediately after this call. The copy option will automatically be set to true in the COM, .NET, and Java bindings (default for other bindings: false). In other language bindings the data will not be copied unless the copy option is supplied.

C++ `int delete_pvf(string filename)`

C# Java `int delete_pvf(String filename)`

Perl PHP `int TET_delete_pvf(resource tet, string filename)`

VB `Function delete_pvf(filename As String) As Long`

C `int TET_delete_pvf(TET *tet, const char *filename, int len)`

Delete a named virtual file and free its data structures (but not the contents).

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 function will silently do nothing. After successfully calling this function *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) will be freed; otherwise, the contents will not be freed, since the client is supposed to do so.

10.3 Exception Handling

C++ *string* **get_apiname()**
C# Java *String* **get_apiname()**
Perl PHP *string* **TET_get_apiname(resource tet)**
VB *Function* **get_apiname()** *As String*
C *const char ****TET_get_apiname(TET *tet)**

Get the name of the API function which caused an exception or failed.

Returns The name of the function which threw an exception, or the name of the most recently called function which failed with an error code. An empty string will be returned if there was no error.

C++ *string* **get_errmsg()**
C# Java *String* **get_errmsg()**
Perl PHP *string* **TET_get_errmsg(resource tet)**
VB *Function* **get_errmsg()** *As String*
C *const char ****TET_get_errmsg(TET *tet)**

Get the text of the last thrown exception or the reason for a failed function call.

Returns Text containing the description of the last exception thrown, or the reason why the most recently called function failed with an error code. An empty string will be returned if there was no error.

C++ *int* **get_errno()**
C# Java *int* **get_errno()**
Perl PHP *long* **TET_get_errno(resource tet)**
VB *Function* **get_errno()** *As Long*
C *int* **TET_get_errno(TET *tet)**

Get the number of the last thrown exception or the reason for a failed function call.

Returns The number of an exception, or the error code of the most recently called function which failed with an error code. This function will return 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 will be left without entering the corresponding

TET_CATCH() block. *TET_RETHROW()* can be used to throw an exception again to a higher-level function after catching it.

Details (C language binding only) See Section 3.2, »C Binding«, page 22.

10.4 Document Functions

C++	<code>int open_document(string filename, string optlist)</code>
C# Java	<code>int open_document(String filename, String optlist)</code>
Perl PHP	<code>long TET_open_document(resource tet, string filename, string optlist)</code>
VB	<code>Function open_document(filename As String, optlist As String) As Long</code>
C	<code>int TET_open_document(TET *tet, const char *filename, int len, const char *optlist)</code>

Open a disk-based or virtual PDF document for content extraction.

filename (Name string, but Unicode file names are only supported on Windows) Absolute or relative name of the PDF input file to be processed. The file will be searched in all directories specified in the *searchpath* resource category. On Windows it is OK to use UNC paths or mapped network drives. In PHP Unicode filenames must be UTF-8.

In non-Unicode language bindings file names with *len* = 0 will be interpreted in the current system codepage unless they are preceded by a UTF-8 BOM, in which case they will be interpreted as UTF-8 or EBCDIC-UTF-8.

len (Only C language binding) Length of *filename* (in bytes) for UTF-16 strings. If *len* = 0 a null-terminated string must be provided.

optlist An option list specifying document options according to Table 10.3. The following options can be used: *encodinghint*, *glyphmapping*, *keepppua*, *inmemory*, *password*, *repair*, *requiredmode*, *shrug*, *tetml*, *unknownchar*, *usehostfonts*.

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, »Indexing protected PDF Documents«, page 49).

Supported file systems on iSeries: 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 10.3 Document options for `TET_open_document()` and `TET_open_document_callback()`

option	description
check-glyphlists	(Boolean) If <code>true</code> , TET will check all builtin glyphmapping rules with <code>condition=allfonts</code> before text extraction starts. Otherwise the global glyphmapping rules will not be 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: <code>false</code>
encoding-hint	(String ¹) The name of an encoding which will be 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 <code>none</code> can be used to disable all predefined rules. Default: <code>winansi</code>
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 reliably be mapped with standard methods. The mappings will be used in least-recently-set order. If the last option list contains the fontname wildcard <code>»*</code> , preceding mappings will no longer be used. Each rule consists of an option list according to Table 10.4. All glyph mappings which match a particular font name will be applied to this font. (default: predefined internal glyph rules will be 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 51).
infomode	(Boolean) Deprecated, use <code>requiredmode</code>
keeppua	(Boolean) If <code>true</code> , PUA (Private Use Area) values will be returned as such; otherwise they will be mapped to the Unicode replacement character (see option <code>unknownchar</code>). Default: <code>false</code>
inmemory	(Boolean; Only for <code>TET_open_document()</code>) If <code>true</code> , 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 <code>false</code> , individual parts of the document will be read from disk as needed. Default: <code>false</code>
password	(String; Maximum string length: 32 characters) The user or master 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 Section 9.6, »Encrypted PDF Documents«, page 119, 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 <code>shrug</code> option can be used to enable content extraction from protected documents under certain conditions (see Section 5.1, »Indexing protected PDF Documents«, page 49).
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: <code>auto</code>): 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 will be made at repairing the document. If there are problems in the PDF the function call will fail.
requiredmode	(Keyword) The minimum <code>pcosmode</code> (minimum/restricted/full) which is acceptable when opening the document. The call will fail if the resulting <code>pcosmode</code> (see Section 9.6, »Encrypted PDF Documents«, page 119) would be lower than the required mode. If the call succeeds it is guaranteed that the resulting <code>pcosmode</code> is at least the one specified in this option. However, it may be higher; e.g. <code>requiredmode=minimum</code> for an unencrypted document will result in <code>full</code> mode. Default: <code>full</code>
shrug	(Boolean) If <code>true</code> , the <code>shrug</code> feature will be activated to enable content extraction from protected documents under certain conditions (see Chapter 5.1, »Indexing protected PDF Documents«, page 49). By using the <code>shrug</code> option you assert that you will honor the PDF document author's rights. Default: <code>false</code>

Table 10.3 Document options for `TET_open_document()` and `TET_open_document_callback()`

option	description
tetml	(Option list) TETML output will be initiated, and can be created page by page with <code>TET_process_page()</code> . The following suboptions are supported: elements (List of Boolean) Specify whether certain TETML elements will be included in the output (default: all true): docinfo The /TET/Document/DocInfo element docxmp The /TET/Document/Metadata element options The elements /TET/Document/Options and /TET/Document/Pages/Page/Options encodingname (Keyword) The name to use in the XML encoding declaration of the text declaration of the generated TETML. The output will always be created in UTF-8 (default: UTF-8): _none No encoding declaration will be created; the output will still be in UTF-8 format. UTF-8 The declaration encoding="UTF-8" will be created. Any other encoding name will be 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) The name of the TETML file. If no filename is supplied, output will be created in memory, and can be retrieved with <code>TET_get_xml_data()</code> . If the function call fails (i.e. the PDF input document could not successfully be opened), no TETML output will be created. version (Integer) Version number of the DTD or schema for the generated TETML output (default: 3): 2 Use the DTD for TET 2.x (which uses version 1 internally) 3 Use the schema for TET 3.0
unknown-char	(Unichar) The character to be used as a replacement for unknown characters which cannot be mapped to Unicode (see Section 6.5, »Unicode Pipeline«, page 68) . Default: U+FFFD (Replacement Character)
usehostfonts	(Boolean) If true, data for fonts which are not embedded, but are required for determining Unicode mappings will be searched on the Mac or Windows host operating system. Default: true

1. See footnote 2 in Table 10.4

Table 10.4 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.
fontname	(Name string) Partial or full name of the font(s) which will be selected for the rule. If a subset prefix has been supplied only the specified subset will be selected. If no subset prefix has been supplied, all fonts where the name (without any subset prefix) matches will be selected. Limited wildcards ¹ are supported. Default: *
fonttypes	(List of keywords) The glyphmapping will only be applied to the specified font types: * (designates all font types), Type1, MMTYPE1, TrueType, CIDFontType2, CIDFontType0, Type3. Default: *
force-encoding	(List with one or two strings ² , If there are two names, the first must be winansi, macroman, or Custom) Replace the first encoding with the encoding resource specified by the second name. If only one entry is supplied, the specified encoding will be used to replace all instances of MacRoman, WinAnsi, and MacExpert encoding. If this option matches a font no other glyph mappings will be applied to the same font.
forcettsymbol-encoding	(Keyword or string ²) The name of an encoding which will be used to determine Unicode mappings for embedded pseudo TrueType symbol fonts which are actually text fonts, or one of the following keywords (default: auto): <div>auto If the font's builtin encoding (see below) contains at least one Unicode character in the symbolic range U+FO000-U+FOFF, the encoding specified in the encodinghint option will be used to map the pseudo symbol characters to real text characters. Otherwise encodinghint will not be used, and the characters will be 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. The well-known TrueType fonts Wingdings* and Webdings* will always be treated as symbol fonts.</div>
globalglyphlist	(Boolean) If true, the specified glyph list will be 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: <div>prefix (String; may be empty) Prefix of the glyph names to which the rule will be applied. base (Keyword) Specifies the interpretation of glyph names: <div>ascii Single-byte glyphnames will be interpreted as the corresponding literal ASCII character (e.g. 1 will be mapped to U+0031). auto Automatically determine whether glyph names represent decimal or hexadecimal values. If the result is not unique, decimal will be assumed. dec The glyphnames will be interpreted as a decimal representation of a code. hex The glyphnames will be interpreted as a hexadecimal representation of a code.</div> encoding (String) Name of an encoding resource which will be used for this rule, or the keyword none to disable the rule.</div>
ignoreto-unicodemap	(Boolean) If true, a ToUnicode CMap for the font will be ignored. Default: false
override	(Boolean; only reasonable together with the glyphlist or glyphrule option) If true, the glyphmapping rule will be applied before the standard (builtin) glyph name mappings (i.e. the new mappings will have priority over the builtin ones), otherwise before. Default: true
tounicode-cmap	(String) Name of a ToUnicode CMap resource to be applied to the font; it will have higher priority than an embedded ToUnicode CMap or encoding entry.

1. Limited wildcards: The standalone character »*« denotes all fonts; Using »*« after a prefix (e.g. »MSTT*«) denotes all fonts starting with the specified prefix.
2. The following predefined encoding names can be used without additional configuration: winansi, macroman, macroman_apple, macroman_euro, ebcdic, ebcdic_37, iso8859-X, cpXXXX, and U+XXXX. Custom encodings can be defined as resources.

```

C++ int open_document_callback(void *opaque, size_t filesize,
                               size_t (*readproc)(void *opaque, void *buffer, size_t size),
                               int (*seekproc)(void *opaque, long offset),
                               string optlist)

C int TET_open_document_callback(TET *tet, void *opaque, size_t filesize,
                                  size_t (*readproc)(void *opaque, void *buffer, size_t size),
                                  int (*seekproc)(void *opaque, long offset),
                                  const char *optlist)

```

Open a PDF document from a custom data source for content extraction.

opaque A pointer to some user data that might be associated with the input PDF document. This pointer will be passed as the first parameter of the callback functions, and can be used in any way. TET will not use the opaque pointer in any other way.

filesize The size of the complete PDF document in bytes.

readproc A 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 A 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 An option list specifying document options according to Table 10.3.

Returns See [TET_open_document\(\)](#).

Details See [TET_open_document\(\)](#).

Bindings This function is only available in the C and C++ language bindings.

```

C++ void close_document(int doc)
C# Java void close_document(int doc)
Perl PHP TET_close_document(resource tet, long doc)
VB Sub close_document(doc As Long)
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 will be 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 function call.

10.5 Page Functions

C++ `int open_page(int doc, int pagenumber, string optlist)`
C# Java `int open_page(int doc, int pagenumber, String optlist)`
Perl PHP `long TET_open_page(resource tet, long pagenumber, string optlist)`
VB `Function open_page(doc As Long, pagenumber As Long, optlist As String) As Long`
C `int TET_open_page(TET *tet, int doc, int pagenumber, const char *optlist)`

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 An option list specifying page options according to Table 10.5. The following options can be used: *clippingarea*, *contentanalysis*, *docstyle*, *excludebox*, *fontsize*, *range*, *granularity*, *ignoreinvisibletext*, *imageanalysis*, *includebox*, *layoutanalysis*, *layouteffort*, *skipengines*, *structureanalysis*.

Returns A handle for the page, or -1 in case of an 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) which may be present on the page are not taken into account: all text on all layers of the page will be extracted, regardless of the visibility of layers.

Table 10.5 Page options for `TET_open_page()` and `TET_process_page()`

option	description
clippingarea	(Keyword) Specifies the clipping area (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
content-analysis	(Option list; not for granularity=glyph) List of suboptions according to Table 10.6 for controlling high-level text processing.

Table 10.5 Page options for `TET_open_page()` and `TET_process_page()`

option	description
docstyle	(Keyword) A hint which will be used by the layout detection engine to select various parameters. These parameters will optimize layout detection for situations where the input document belongs to one of the following classes: book Typical book business Business documents fancy Fancy pages with complex layout forms Structured forms generic The most general document class without any further qualification. magazines Magazine articles papers Newspaper science Scientific article searchengine The application is a search engine or similar, and mainly interested in retrieving the word list for the page as fast as possible. Table and page structure recognition will be disabled.
excludebox	(List of rectangles) Exclude the combined area of the specified rectangles from text extraction. Default: empty
fontsize-range	(List of two floats) Two numbers specifying the minimum and maximum font size of text. Text with a size outside of this interval will be ignored. The maximum can be specified with the keyword unlimited, which means that no upper limit will be active. Default: { 0 unlimited }
granularity	(Keyword) The granularity of the text fragments returned by <code>TET_get_text()</code> ; all modes except glyph will enable the Wordfinder. See »Text granularity«, page 71, for more details (default: word). glyph A fragment contains the result of mapping 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 will be inserted between two consecutive words. page A fragment contains the contents of a single page. Word, line, and zone separators will be inserted as appropriate.
ignore-invisibletext	(Boolean) If true, text with rendering mode 3 (invisible) will be 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 10.8 for controlling high-level image processing.
includebox	(List of rectangles) Restrict text extraction to the combined area of the specified rectangles. Default: the complete clipping area
layout-analysis	(Option list; not for granularity=glyph) List of suboptions according to Table 10.7 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
skipengines	(List of keywords) Skip some of the available parsers for the page contents. A skipped engine never returns any data for this page. Skipping an engine which is not required will improve performance for applications which don't need the data delivered by this engine (default: all engines are active): text (Keyword) Skip the text extraction engine. image (Keyword) Skip the image extraction engine.
structure-analysis	(Option list; not for granularity=glyph) List of suboptions according to Table 10.9 for controlling page structure analysis.



Table 10.6 Suboptions for the contentanalysis option of `TET_open_page()` and `TET_process_page()`

option	description
dehyphenate	(Boolean) If true, hard hyphens (U+002D and U+2010) and soft hyphens (U+00AD) at the end of a line will be removed, and the text fragments surrounding the hyphen will be combined. Default: true
dropcapsize	(Float) The minimum size at which large glyphs will be 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 will be 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 neighbouring text. Large characters will be 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) When multiple include boxes have been supplied (see option includebox), this option controls how the order of boxes affects the wordfinder (default: 0): <div><div>0</div><div>Ignore include box ordering when analyzing the page contents. The result will be 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.</div></div> <div><div>1</div><div>Take include box ordering into account when creating words and zones, but not for zone ordering. A word will never belong to more than one box. The resulting zones will be sorted in logical order. In case of overlapping boxes the text will belong 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.</div></div> <div><div>2</div><div>Consider include box ordering for all operations. The contents of each include box will be treated independently from other boxes, and the resulting text will be 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.</div></div>
lineseparator	(Unichar; Only for granularity=page) Character to be inserted between lines ¹ . Default: U+000A
linespacing	(Keyword) Specify the typical vertical distance between text lines within a paragraph: small, medium, or large (default: medium)
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 will be active) the detected zones on the page will be merged appropriately and sorted. If the number of words on the page is greater than the specified number, no zones will be built, and words will be retrieved in page content reading order. Processing will be 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: 5000
merge	(Integer) Controls strip and zone merging (default: 2): <div><div>0</div><div>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.</div></div> <div><div>1</div><div>Simple strip-into-zone merging: strips will be 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).</div></div> <div><div>2</div><div>Advanced zone merging for out-of-sequence text: in addition to merge=1, multiple overlapping zones will be combined into a single zone, provided the text contents of both zones do not overlap.</div></div>
shadow-detect	(Boolean) If true, redundant instances of overlapping text fragments which create a shadow or fake bold text will be detected and removed. Default: true
punctuation breaks	(Boolean; only for granularity=word) If true, punctuation characters which are placed close to a letter will be treated as word boundaries, otherwise they will be included in the adjacent word. Default: true

Table 10.6 Suboptions for the contentanalysis option of `TET_open_page()` and `TET_process_page()`

option	description
<code>wordseparator</code>	(Unichar; Only for granularity=line and page) Character to be inserted between words ¹ . Default: U+0020

1. Use U+0000 to disable the separator.

Table 10.7 Suboptions for the layoutanalysis option of `TET_open_page()` and `TET_process_page()`

option	description
layout-astable	(Boolean) If true, the layout recognition engine 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 will be 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 will be sorted column by column. none No hint available; zone ordering will be determined by page content order. singlecolumn The page contains single-column text; zones will be sorted row by row.
layoutdetect	(Integer) Specifies the depth of recursive layout recognition (default: 1): 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.
layoutrowhint	(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 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 will be 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 merge. 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 will be 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 10.7 Suboptions for the layoutanalysis option of `TET_open_page()` and `TET_process_page()`

option	description
supertable-columns	(Integer; only if <code>layoutastable=true</code>) Minium 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: 1): 0 No table recognition. 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 10.8 Suboptions for the imageanalysis option of `TET_open_page()` and `TET_process_page()`

option	description
smallimages	(Option list) Control small image removal. Small images must often be ignored since they are artifacts and not real images. Supported options: disable (Boolean) If true, small image removal will be disabled. Default: false maxarea (Float) Maximum area (=width x height) in pixels of an image to be considered as a small image. Default: 500 maxcount (Integer) Maximum allowed number of small images. If more small images are found all of them will be removed. Default: 50
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 will be disabled. Default: false gap (Float) Maximum gap in points between two images to be considered for merging. Default: 1.0 (not 0.0 because of unavoidable inaccuracies in the position calculations)

C++	<code>void close_page(int page)</code>
C# Java	<code>void close_page(int page)</code>
Perl PHP	<code>TET_close_page(resource tet, long page)</code>
VB	<code>Sub close_page(page As Long)</code>
C	<code>void TET_close_page(TET *tet, int page)</code>

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 will be 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 function call.

Table 10.9 Suboptions for the structureanalysis option of `TET_open_page()` and `TET_process_page()`

option	description
bullets	<p>(List of option lists; only if <code>list=true</code>) Specifies combinations of Unicode characters and font names which are used as bullet characters in lists. Supported suboptions:</p> <p>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 <code>fontname</code> will be treated as bullet characters.</p> <p>fontname (String) Name of the font from which bullet characters are drawn. If this suboption is not supplied, the characters specified in the <code>bulletchars</code> suboption will always be treated as bullet characters.</p> <p>Examples: <code>bullets={{fontname=ZapfDingbats}}</code> <code>bullets={{bulletchars={U+2022}}}</code> <code>bullets={{fontname=KozGoPro-Medium bulletchars={U+2460 U+2461 U+2462 U+2463 U+2464}}}</code></p>
list	(Boolean) Enable list recognition (default: false). If false, no information about list structure will be determined.
paragraph	(Boolean) Enable paragraph recognition (default: true). If false, no information about paragraph structure will be determined.
table	(Boolean) Enable table recognition (default: true). If false, the table recognition engine will be disabled.

10.6 Text and Metrics Retrieval Functions

C++ `string get_text(int page)`

C# Java `String get_text(int page)`

Perl PHP `string TET_get_text(resource tet, long page)`

VB `Function get_text(page As Long) As String`

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 language binding only) A pointer to a variable which will hold the length of the returned string in UTF-16 values (not bytes!). To determine the number of bytes this value must be multiplied by 2 if `outputformat=utf16`; the string length of the returned null-terminated string must be used if `outputformat=utf8`.

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 6.2, »Unicode Concepts«, page 61).

If all text on the page has been retrieved an empty string or null object will be returned (see below). In this case `TET_get_errnum()` should 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.

Bindings C language binding: the result will be provided as null-terminated UTF-8 (default) or UTF-16 string according to the *outputformat* option of `TET_set_option()`. On iSeries and zSeries 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 function. If no more text is available a NULL pointer and `*len=0` will be returned.

C++ and COM: the result will be provided as Unicode string in UTF-16 format. If no more text is available an empty string will be returned.

Java and .NET: the result will be provided as Unicode string. If no more text is available a null object will be returned.

Perl, PHP, and Python language bindings: the result will be provided as UTF-8 string. If no more text is available a null object will be returned.

RPG language binding: the result will be provided as Unicode string. If no more text is available a NULL pointer will be returned.

C++	<code>const TET_char_info *get_char_info(int page)</code>
C# Java	<code>int get_char_info(int page)</code>
Perl PHP	<code>object TET_get_char_info(resource tet, long page)</code>
VB	<code>Function get_char_info(int page) As Long</code>
C	<code>const TET_char_info *TET_get_char_info(TET *tet, int page)</code>

Get detailed information for the next character in the most recent text fragment.

page A valid page handle obtained with `TET_open_page()`.

Returns If no more characters are available for the most recent text fragment returned by `TET_get_text()`, a binding-specific value will be returned. See section *Bindings* below for more details.

Details This function can be called after `TET_get_text()`. It will advance to the next character for the current text fragment associated with the supplied page handle (or return 0 or NULL if there are no more characters), and provide detailed information for this character. There will be *N* successful calls to this function where *N* is the number of UTF-16 characters in the text fragment returned by the most recent call to `TET_get_text()`.

For granularities other than *glyph* this function will advance to the next character of the string returned by the most recent call to `TET_get_text()`. This way it is possible to retrieve character metrics when the wordfinder is active and a text fragment may contain more than one character. In order to retrieve all character details for the current text fragment this function must be called repeatedly until it returns NULL or 0.

The character details in the structure or properties/fields are valid until the next call to `TET_get_char_info()` or `TET_close_page()` with the same page handle (whichever occurs first). Since there is only a single set of character info properties/fields per TET object, clients must retrieve all character 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 characters are available for the most recent text fragment returned by `TET_get_text()`, a NULL pointer will be returned. Otherwise, a pointer to a `TET_char_info` structure containing information about a single character will be returned. The members of the data structure are detailed in Table 10.10.

COM, Java and .NET language bindings: -1 will be returned if no more characters are available for the most recent text fragment returned by `TET_get_text()`, otherwise 1. Individual character info can be retrieved from the TET properties/public fields according to Table 10.10. All properties/fields will contain a value of -1 (the *unknown* field will contain *false*) if they are accessed although the function returned -1.

Perl language binding: 0 will be returned if no more characters are available for the most recent text fragment returned by `TET_get_text()`, otherwise a hash containing the keys listed in Table 10.10. Individual character info can be retrieved with the keys in this hash.

PHP language binding: an empty (null) object will be returned if no more characters are available for the most recent text fragment returned by `TET_get_text()`, otherwise an object containing the fields listed in Table 10.10. Individual character info can be retrieved from the member fields of this object. Integer fields in the character info object are implemented as *long* in the PHP language binding.

Table 10.10 Members of the **TET_char_info** structure (C and C++), equivalent public fields (Java, PHP), keys (Perl) or properties (COM and .NET) with their type and meaning. See »Glyph metrics«, page 64, for more details.

property/ field name	explanation
uv	(Integer) UTF-32 Unicode value of the current character. It will be 0 if the corresponding UTF-16 value is the trailing value of a surrogate pair (i.e. if type=11).
type	(Integer) Type of the character. The following types describe real characters which correspond to a glyph on the page. The values of all other properties/fields are determined by the corresponding glyph: 0 Normal character which corresponds to exactly one glyph 1 Start of a sequence (e.g. ligature) The following types describe artificial characters which do not correspond to a glyph on the page. The x and y fields will specify the most recent real character's endpoint, the width field will be 0, and all other fields except uv will contain the values corresponding to the most recent real character: 10 Continuation of a sequence (e.g. ligature) 11 Trailing value of a surrogate pair; the leading value has type=0, 1, or 10. 12 Inserted word, line, or zone separator
unknown	(Boolean, in C and C++: integer) Usually false (0), but will be 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 will be 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 the width will be 0.
alpha	(Double) Direction of inline text progression in degrees measured counter-clockwise. For horizontal writing mode this is the direction of the text baseline; for vertical writing mode it is the digression from the standard -90° direction. The angle will be in the range -180° < alpha ≤ +180°. For standard horizontal text as well as for standard text in vertical writing mode the angle will be 0°.
beta	(Double) Text slanting angle in degrees (counter-clockwise), relative to the perpendicular of alpha. The angle will be 0° for upright text, and negative for italicized (slanted) text. The angle will be 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 Table 9.5). 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.
textrendering	(Integer) Text rendering mode: 0 fill text 1 stroke text (outline) 2 fill and stroke text 3 invisible text (often used for OCR results) 4 fill text and add it to the clipping path 5 stroke text and add it to the clipping path 6 fill and stroke text and add it to the clipping path 7 add text to the clipping path

10.7 Image Retrieval Functions

C++	<code>const TET_image_info *get_image_info(int page)</code>
C# Java	<code>int get_image_info(int page)</code>
Perl PHP	<code>object TET_image_info TET_get_image_info(resource tet, long page)</code>
VB	<code>Function get_image_info(int page) As Long</code>
C	<code>const TET_image_info *TET_get_image_info(TET *tet, int page)</code>

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 will be returned, otherwise image details are available in a binding-specific manner. See section *Bindings* below for more details.

Details This function advances to the next image associated with the supplied page handle (or return 0 or NULL if there are no more images), and provide detailed information for this image. This function will also return artificial images created by the image merging mechanism. However, the consumed images used to create artificial images will not be returned.

The image details in the structure or properties/fields are valid until the next call to `TET_get_image_info()` or `TET_close_page()` with the same page handle (whichever occurs first). 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 or document.

Bindings C and C++ language bindings: If no more images are available on the page a NULL pointer will be 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 10.11.

COM, Java and .NET language bindings: -1 will be 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 10.11. All properties/fields will contain a value of -1 if they are accessed although the function returned -1.

Perl language binding: 0 will be returned if no more images are available on the page, otherwise a hash containing the keys listed in Table 10.11. Individual character info can be retrieved with the keys in this hash.

PHP language binding: an empty (null) object will be 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 10.11. Integer fields in the image info object are implemented as *long* in the PHP language binding.

Table 10.11 Members of the `TET_image_info` structure (C and C++), equivalent public fields (Java and PHP), and properties (COM and .NET) with their type and meaning. See »Image Extraction Basics«, page 81, for more 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 will be in the range $-180^{\circ} < \alpha \leq +180^{\circ}$. For upright images alpha will be 0° .
beta	(Double) Direction of the pixel columns, relative to the perpendicular of alpha. The angle will be in the range $-180^{\circ} < \beta \leq +180^{\circ}$, but different from $\pm 90^{\circ}$. For upright images beta will be in the range $-90^{\circ} < \beta < +90^{\circ}$. If $\text{abs}(\beta) > 90^{\circ}$ the image will be mirrored at the baseline.
imageid	(Integer) Index of the image in the pCOS pseudo object images[]. Detailed image properties can be retrieved via the entries in this pseudo object (see Table 9.5).

```

C++  int write_image_file(int doc, int imageid, string optlist)
C# Java  int write_image_file(int doc, int imageid, String optlist)
Perl PHP  long TET_write_image_file(resource tet, long doc, long imageid, string optlist)
VB  Function write_image_file(doc As Long, imageid As Long, optlist As String) As Long
C  int TET_write_image_file(TET *tet, int doc, int imageid, const char *optlist)

```

Write image data to disk.

- doc** A valid document handle obtained with `TET_open_document*()`.
- 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` pseudo object (there are `length:images` entries in this array).
- optlist** An option list specifying page options according to Table 10.12. The following options can be used: `compression`, `filename`, `keepxmp`, `typeonly`.

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 will be created in case of an error. The rare case of images in an unsupported format will also be reported as 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:

- ▶ -1: an error occurred; no image will be extracted
- ▶ 10: image extracted as TIFF (.tif)
- ▶ 20: image extracted as JPEG (.jpg)
- ▶ 30: image extracted as JPEG 2000 (.jpx)

Details This function will convert the pixel data for the image with the specified pCOS ID to one of several image formats, and write the result to a disk file. If the `typeonly` option has been supplied, only the image type will be returned, but no image file will be generated.

Bindings C/C++: macros for the return values are available in `tetlib.h`.

Table 10.12 Options for `TET_write_image_file()` and `TET_get_image_data()`

option	description
compression	(Keyword) The algorithm for compressing the pixel data (default: auto): <ul style="list-style-type: none"> auto select a suitable compression algorithm automatically none (Only relevant for TIFF images) Write the pixel data without any compression if possible.
filename ¹	(String; required unless <code>typeonly</code> is also supplied) The name of the image file on disk. A suffix will be added to the filename to indicate the image file format.
keepxmp	(Boolean) If true and the image has associated XMP metadata in the PDF, the metadata will be embedded in extracted TIFF and JPEG images. Default: true
typeonly ¹	(Boolean) The image type will be determined according to the supplied options, but no image file will be written. This is useful for determining the type of image returned by <code>TET_get_image_data()</code> , which does not return the image type itself. Default: false

1. Only for `TET_write_image_file()`

```

C++  const char *get_image_data(int doc, size_t *length, int imageid, string optlist)
C#   final byte[] get_image_data(int doc, int imageid, String optlist)
Perl PHP  string TET_get_image_data(resource tet, long doc, long imageid, string optlist)
VB      Function get_image_data(doc As Long, imageid As Long, optlist As String)
C       const char *TET_get_image_data(TET *tet, int doc, size_t *length, int imageid, const char *optlist)

```

Retrieve image data from memory.

doc A valid document handle obtained with *TET_open_document**().

length (C and C++ language bindings only) C-style pointer to a memory location where the length of the returned data in bytes will be 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 An option list specifying image-related options according to Table 10.12. The following options can be used: *compression*, *keepxmp*

Returns The data representing the image according to the specified options. In case of an error (including images which cannot be extracted) a NULL pointer will be returned in C and C++, and empty data in other language bindings. If an error happens it is recommended to call *TET_get_errmsg*() to find out more details about the error.

Details This function will convert the pixel data for the image with the specified pCOS ID to one of several image formats, and make the data available in memory.

Bindings COM: Most client programs will use the Variant type to hold the image data.

C and C++ language bindings: The returned data buffer can be used until the next call to this function.

10.8 TET Markup Language (TETML) Functions

C++	<code>int process_page(int doc, int pagenumber, string optlist)</code>
C# Java	<code>int process_page(int doc, int pagenumber, String optlist)</code>
Perl PHP	<code>long TET_process_page(resource tet, long doc, long pagenumber, string optlist)</code>
VB	<code>Function process_page(doc As Long, pagenumber As Long, optlist As String) As Int</code>
C	<code>int TET_process_page(TET *tet, int doc, int pagenumber, const char *optlist)</code>

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** An option list specifying options from the following groups:
- General page-related options according to Table 10.5 (these will be ignored if `pagenumber=0`): `clippingarea`, `contentanalysis`, `excludebox`, `fontsize`, `granularity`, `ignoreinvisibletext`, `imageanalysis`, `includebox`, `layoutanalysis`, `skipengines`
 - Option specifying processing details according to Table 10.13: `tetml`

Table 10.13 Additional options for `TET_process_page()`

option	description
tetml	(Option list) Controls details of TETML. The following options are available: glyphdetails (Option list; only for <code>granularity=glyph</code> and <code>word</code>) Specify which members of the <code>TET_char_info</code> structure will be reported for each Glyph element: geometry (Boolean) Emit attributes <code>x</code> , <code>y</code> , <code>width</code> , <code>alpha</code> , <code>beta</code> . Default: false font (Boolean) Emit attributes <code>font</code> , <code>fontsize</code> , <code>textrendering</code> , <code>unknown</code> . Default: false trailer (Boolean) If true, document trailer data, i.e. data after the last page, will be emitted (it must be appended to the page-specific data emitted earlier). This option is required in the last call to this function in order to emit trailer data. If <code>pagenumber=0</code> only trailer data (without any page-specific data) will be emitted. Once <code>trailer=true</code> has been supplied, no more calls to <code>TET_process_page()</code> are allowed for the same document. Default: false

Returns -1 on error, or 1 otherwise. However, in TETML mode this function will always succeed since problems related to the input document will be reported in a TETML element, and all other problems will raise an exception.

Details This function will open a page, create output according to the format-related options supplied to `TET_open_document*`(), and close the page. The generated data can be retrieved with `TET_get_xml_data()`.

This function 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, will be created by `TET_open_document*`() before the first page data. It can be retrieved separately by calling `TET_get_xml_data()` 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 function 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_xml_data(int doc, size_t *length, string optlist)*

C# Java *final byte[] get_xml_data(int doc, String optlist)*

Perl PHP *string TET_get_xml_data(resource tet, long doc, string optlist)*

VB *Function get_xml_data(doc As Long, optlist As String)*

C *const char *TET_get_xml_data(TET *tet, int doc, size_t *length, const char *optlist)*

Retrieve TETML data from memory.

doc A valid document handle obtained with *TET_open_document*()*.

length (C and C++ language binding only) A pointer to a variable which will hold the length of the returned string in bytes. *length* does not count the terminating null byte.

optlist (Currently there are no supported options.)

Returns A byte array containing the next chunk of data according to the specified options. If the buffer is empty an empty string will be returned (in C: a NULL pointer and **len=0*).

Details This functions retrieves TETML data which has been created by *TET_open_document*()* and one or more calls to *TET_process_page()*. The TETML data will always be encoded in UTF-8, regardless of the *outputformat* option. The internal buffer will be cleared by this call. It is not required to call *TET_get_xml_data()* 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 function must be called at least once before *TET_close_document()* since otherwise the data would no longer be accessible. If *TET_get_xml_data()* 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 function 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 will be provided as null-terminated UTF-8. On iSeries and zSeries EBCDIC-encoded UTF-8 will be returned. The returned data buffer can be used until the next call to *TET_get_xml_data()*.

Java and .NET language bindings: the result will be provided as a byte array containing UTF-8 data.

COM: Most client programs will use the Variant type to hold the UTF-8 data.

PHP language binding: the result will be provided as UTF-8 string.

RPG language binding: the result will be provided as null-terminated EBCDIC-encoded UTF-8.

10.9 Option Handling

C++
C# Java
Perl PHP
VB
C

void set_option(string optlist)
void set_option(String optlist)
TET_set_option(resource tet, string optlist)
Sub set_option(optlist As String)
void TET_set_option(TET *tet, const char *optlist)

Set one or more global options for TET.

optlist An option list specifying global options according to Table 10.14. If an option is provided more than once the last instance will override all previous ones. In order to supply multiple values for a single option (e.g. *searchpath*) supply all values in a list argument to this option.

The following options can be used: *asciifile*, *cmap*, *codelist*, *encoding*, *fontoutline*, *glyphlist*, *license*, *licensefile*, *outputformat*, *resourcefile*, *searchpath*

Details Multiple calls to this function can be used to accumulate values for those options marked in Table 10.14. For unmarked options the new value will override the old one.

Table 10.14 Global options for *TET_set_option()*

option	description
<i>asciifile</i>	(Boolean; Only supported on iSeries and zSeries). Expect text files (e.g. UPR configuration files, glyph lists, code lists) in ASCII encoding. Default: true on iSeries; false on zSeries
<i>cmap</i> ^{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 51).
<i>codelist</i> ^{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 51).
<i>encoding</i> ^{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 51).
<i>fontoutline</i> ^{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 51).
<i>glyphlist</i> ^{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 51).
<i>hostfont</i> ^{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.
<i>license</i>	(String) Set the license key. It must be set before the first call to <i>TET_open_document*()</i> .
<i>licensefile</i>	(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 <i>TET_open_document*()</i> . Alternatively, the name of the license file can be supplied in an environment variable called PDFLIBLICENSEFILE or (on Windows) via the registry.

Table 10.14 Global options for `TET_set_option()`

option	description
output-format	(Keyword; Only for the C and RPG language bindings) Specifies the format of the text returned by <code>TET_get_text()</code> (default on zSeries with USS or MVS: ebcdicutf8; on all other systems: utf8):
	utf8 Strings will be returned in null-terminated UTF-8 format (on both ASCII- and EBCDIC-based systems).
	ebcdicutf8 (Only available on EBCDIC-based systems) Strings will be returned in null-terminated EBCDIC-encoded UTF-8 format. Code page 37 will be used on iSeries, code page 1047 on zSeries.
	utf16 Strings will be returned in UTF-16 format in the machine's native byte ordering (on Intel x86 architectures the native byte order is little-endian, while on Sparc and PowerPC systems it is big-endian).
resourcefile	<p>(Name string) Relative or absolute file name of the UPR resource file. The resource file will be loaded immediately. Existing resources will be kept; their values will be overridden by new ones if they are set again. Explicit resource options will be evaluated after entries in the resource file.</p> <p>The resource file name can also be supplied in the environment variable <code>TETRESOURCEFILE</code> or with a Windows registry key (see Section 5.2, »Resource Configuration and File Searching«, page 51). Default: <code>tet.upr</code> (on MVS: <code>upr</code>)</p>
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 will be accumulated and used in least-recently-set order (see Section 5.2, »Resource Configuration and File Searching«, page 51). An empty string deletes all existing search path entries. On Windows the search path can also be set via a registry entry. Default: empty

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.

10.10 pCOS Functions

The full pCOS syntax for retrieving object data from a PDF is supported; see Chapter 9, »The pCOS Interface«, page 105 for a detailed description.

```
C++ double pcos_get_number(int doc, string path)
C# Java double pcos_get_number(int doc, String path)
Perl PHP float TET_pcos_get_number(resource tet, int doc, String path)
VB Function pcos_get_number(doc as Long, path As String) As Double
C double TET_pcos_get_number(TET *tet, int doc, const char *path, ...)
```

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 will be returned if they are *true*, and 0 otherwise.

```
C++ string pcos_get_string(int doc, string path)
C# Java String pcos_get_string(int doc, String path)
Perl PHP String TET_pcos_get_string(resource tet, int doc, String path)
VB Function pcos_get_string(doc as Long, path As String) As String
C const char *TET_pcos_get_string(TET *tet, int doc, const char *path, ...)
```

Get the value of a pCOS path with type *name*, *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* will be returned.

Details This function will raise an exception if pCOS does not run in full mode and the type of the object is *string* (see Section 9.6, »Encrypted PDF Documents«, page 119). As an exception, the objects */Info/** (document info keys) can also be retrieved in restricted pCOS

mode if *nocopy=false* or *plainmetadata=true*, and *bookmarks[...]/Title* and *pages[...]/Annots/Contents* can be retrieved in restricted pCOS mode if *nocopy=false*.

This function 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 binding: The returned strings will be stored in a ring buffer with up to 10 entries. If more than 10 strings are queried, buffers will be 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 function 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 and C++ language bindings: The string will be returned in UTF-8 format without BOM.

C++ and COM: the result will be provided as Unicode string in UTF-16 format. If no more text is available an empty string will be returned.

Java and .NET: the result will be provided as Unicode string. If no more text is available a null object will be returned.

Perl, PHP, and Python language bindings: the result will be provided as UTF-8 string. If no more text is available a null object will be returned.

RPG language binding: the result will be provided as UTF-8 string.

C++	<i>const unsigned char *pcos_get_stream(int doc, int *length, string optlist, string path)</i>
C# Java	<i>final byte[] pcos_get_stream(int doc, String optlist, String path)</i>
Perl PHP	<i>String TET_pcos_get_stream(resource tet, int doc, String optlist, String path)</i>
VB	<i>Function pcos_get_stream(doc as Long, optlist As String, path As String)</i>
C	<i>const unsigned char *TET_pcos_get_stream(TET *tet, int doc, int *length, const char *optlist, const char *path, ...)</i>

Get the contents of a pCOS path with type *stream*, *fstream*, or *string*.

doc A valid document handle obtained with *TET_open_document*()*.

length (C and C++ language bindings only) A pointer to a variable which will receive the length of the returned stream data in bytes.

optlist An option list specifying stream retrieval options according to Table 10.15.

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 will be empty (in C and C++: NULL) if the stream or string is empty.

If the object has type *stream* all filters will be removed from the stream contents (i.e. the actual raw data will be returned). If the object has type *fstream* or *string* the data will be delivered exactly as found in the PDF file, with the exception of ASCII85 and ASCII-Hex filters which will be removed.

In addition to decompressing the data and removing ASCII filters, text conversion may be applied according to the *convert* option.

Details This function will throw an exception if pCOS does not run in full mode (see Section 9.6, »Encrypted PDF Documents«, page 119). 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 function can also be used to retrieve objects of type *string*. Unlike *TET_pcos_get_string()*, which treats the object as a text string, this function will not modify the returned data in any way. Binary string data is rarely used in PDF, and cannot be reliably detected automatically. The user is therefore responsible for selecting the appropriate function for retrieving string objects as binary data or text.

Bindings COM: Most client programs will use the Variant type to hold the stream contents. JavaScript with COM does not allow to retrieve the length of the returned variant array (but it does work with other languages and COM).

C and C++ language bindings: The returned data buffer can be used until the next call to this function.

Note *This function can be used to retrieve embedded font data from a PDF. Users are reminded of the fact that fonts are subject to the respective font vendor's license agreement, and must not be reused without the explicit permission of the respective intellectual property owners. Please contact your font vendor to discuss the relevant license agreement.*

Table 10.15 Options for *TET_pcos_get_stream()*

option	description
convert	(Keyword; will be ignored for streams which are compressed with unsupported filters) Controls whether or not the string or stream contents will be 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 <i>TET_pcos_get_string()</i>), and normalize it to Unicode. In non-Unicode-aware language bindings this means the data will be 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; will be ignored for streams which are compressed with unsupported filters) If true, the stream data will be compressed with the filter which is specified in the image's filterinfo pseudo object (see Table 9.5, page 117). If false, the stream data will be 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.

General Functions

Function prototype	page
(C) TET *TET_new(void)	123
void delete()	123
(C) const char *TET_utf8_to_utf16(TET *tet, const char *utf8string, const char *ordering, int *size)	123
(C) const char *TET_utf16_to_utf8(TET *tet, const char *utf16string, int len, int *size)	124
(C) const char *TET_utf32_to_utf16(TET *tet, const char *utf32string, int len, const char *ordering, int *size)	124
void create_pvf(String filename, byte[] data, String optlist)	125
int delete_pvf(String filename)	126

Exception Handling Functions

Function prototype	page
String get_apiname()	127
String get_errmsg()	127
int get_errnum()	127

Document Functions

Function prototype	page
int open_document(String filename, String optlist)	129
(C) int TET_open_document_callback(TET *tet, void *opaque, size_t filesize, size_t (*readproc)(void *opaque, void *buffer, size_t size), int (*seekproc)(void *opaque, long offset), const char *optlist)	133
void close_document(int doc)	133

Page Functions

Function prototype	page
int open_page(int doc, int pagenumber, String optlist)	134
void close_page(int page)	140

Text and Metrics Retrieval Functions

Function prototype	page
String get_text(int page)	142
int get_char_info(int page)	143

Image Retrieval Functions

<i>Function prototype</i>	<i>page</i>
<i>int get_image_info(int page)</i>	145
<i>int write_image_file(int doc, int imageid, String optlist)</i>	147
<i>final byte[] get_image_data(int doc, int imageid, String optlist)</i>	148

TET Markup Language (TETML) Functions

<i>Function prototype</i>	<i>page</i>
<i>int process_page(int doc, int pagenumber, String optlist)</i>	149
<i>final byte[] get_xml_data(int doc, String optlist)</i>	150

Option Handling

<i>Function prototype</i>	<i>page</i>
<i>void set_option(String optlist)</i>	151

pCOS Functions

<i>Function prototype</i>	<i>page</i>
<i>double pcos_get_number(int doc, String path)</i>	153
<i>String pcos_get_string(int doc, String path)</i>	153
<i>final byte[] pcos_get_stream(int doc, String optlist, String path)</i>	154

B Revision History

Revision history of this manual

Date	Changes
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 <code>TET_open_doc_callback()</code> 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* 59
- API reference* 121
- area of text extraction* 64
- arrays* 109

B

- bookmarks* 59
- Byte Order Mark (BOM)* 61, 123

C

- C binding* 22
- C++ binding* 24
- categories of resources* 51
- character references* 122
- characters* 61
- CJK (Chinese, Japanese, Korean)* 67
 - compatibility forms* 67
 - configuration* 7
- CJK support* 12
- codelist* 77
- COM binding* 25
- command-line tool* 15
- comments* 59
- composite characters* 62
- concordance (XSLT sample)* 101
- connector* 35
- content analysis* 71
- coordinate system* 64
- CSV format* 103
- CUS (Corporate Use Subarea)* 68

D

- dehyphenation* 73
- dictionaries* 109
- Dispose()* 123
- document and page functions* 129
- document domains* 57
- document info entries* 57
- document info fields* 105
- document styles* 74

E

- EBCDIC-based systems* 152
- encrypted PDF documents* 119
- encryption status* 106
- end points of glyphs and words* 66

- evaluation version* 7

- examples*

- document info fields* 105
 - encryption status* 106
 - fonts in a document* 106
 - number of pages* 105
 - page size* 106
 - pCOS paths* 105
 - text extraction status* 49
 - writing mode* 106
 - XSLT* 101

- exception handling* 21
 - in C* 22

F

- fake bold removal* 73
- file attachments* 59
- file searching* 52
- font filtering (XSLT sample)* 101
- font statistics (XSLT sample)* 102
- FontReporter plugin* 11, 76
- fonts in a document* 106
- form fields* 59
- fullwidth variants* 67

G

- glyph metrics* 64
- glyph rules* 79
- glyphlist* 79
- glyphs* 61
- granularity* 71

H

- halfwidth variants* 67
- highlighting* 66
- HTML converter (XSLT sample)* 103

I

- IFilter for Microsoft products* 44
- image*
 - small image removal* 86
- image extraction* 81
- image objects* 81
- images*
 - color fidelity* 87
 - geometry* 83
 - merging* 85
 - resolution* 83

- unsupported types* 87
 - XMP metadata* 82
- inch* 64
- index (XSLT sample)* 102
- installing TET* 7

J

- Java binding* 26

L

- license key* 8
- ligatures* 62
- list values in option lists* 122
- Lucene search engine* 37

M

- MediaWiki* 46
- millimeters* 64
- mini samples* 13

N

- .NET binding* 27
- number of pages* 105

O

- optimizing performance* 54
- option lists* 121
- Oracle Text* 41

P

- packages* 60
- page boxes* 64
- page size* 106
- pCOS* 105
 - API functions* 153
 - data types* 107
 - encryption* 119
 - path syntax* 110
 - pseudo objects* 112
- pCOS Cookbook* 14
- pCOS interface* 105
- PDF Reference Manual* 105
- PDF versions* 11
- performance optimization* 54
- Perl binding* 28
- PHP binding* 29
- placed images* 81
- points* 64
- portfolios* 60
- post-processing for Unicode values* 68
- prerotated glyphs* 67
- protected documents* 49
- PUA (Private Use Area)* 68
- Python Binding* 31

R

- raw text extraction (XSLT sample)* 103
- rectangles in option lists* 122
- replacement character* 69
- resource configuration* 51
- resourcefile parameter* 53
- response file* 17
- roadmap to documentation and samples* 13
- RPG binding* 32

S

- schema* 96
- searching for font usage (XSLT sample)* 102
- searchpath* 52
- sequences* 62
- shadow removal* 73
- shrug feature* 49
- small image removal* 86
- Solr search server* 40
- surrogates* 63, 65

T

- table detection* 74
- table extraction (XSLT sample)* 103
- TET command-line tool* 15
- TET connector* 35
 - for Lucene* 37
 - for MediaWiki* 46
 - for Microsoft products* 44
 - for Oracle* 41
 - for Solr* 40
- TET Cookbook* 14
- TET features* 11
- TET Markup Language (TETML)* 89
- TET plugin for Adobe Acrobat* 35
- tet.upr* 53
- TET_CATCH()* 127
- TET_close_document()* 133
- TET_close_page()* 140
- TET_create_pvf()* 125
- TET_delete()* 123
- TET_delete_pvf()* 126
- TET_EXIT_TRY()* 22, 127
- TET_get_apiname()* 127
- TET_get_char_info()* 143
- TET_get_errmsg()* 127
- TET_get_errnum()* 127
- TET_get_image_data()* 148
- TET_get_image_info()* 145
- TET_get_text()* 142
- TET_get_xml_data()* 150
- TET_new()* 123
- TET_open_document()* 129
- TET_open_document_callback()* 133
- TET_open_page()* 134
- TET_pcos_get_number()* 153

TET_pcos_get_stream() 154
TET_pcos_get_string() 153
TET_RETHROW() 127
TET_set_option() 151
TET_TRY() 127
TET_utf16_to_utf8() 124
TET_utf32_to_utf16() 124
TET_utf8_to_utf16() 123
TET_write_image_file() 147
TETML 89
TETML schema 96
TETRESOURCEFILE environment variable 53
TeX documents 56
text extraction status 49
text filtering 69
ToUnicode CMap 78

U

Unicode mapping 68
Unicode pipeline 68
units 64
unmappable glyphs 69

UPR file format 51
UTF formats 61
UTF-32 69
UTF-8 and UTF-16 123

V

vertical writing mode 67

W

word boundary detection 72
wordfinder 72
writing mode 106

X

XMP metadata 58, 106
 for images 82
 XSLT sample 103
XSD schema for TETML 96
XSLT 97
XSLT samples 14, 101