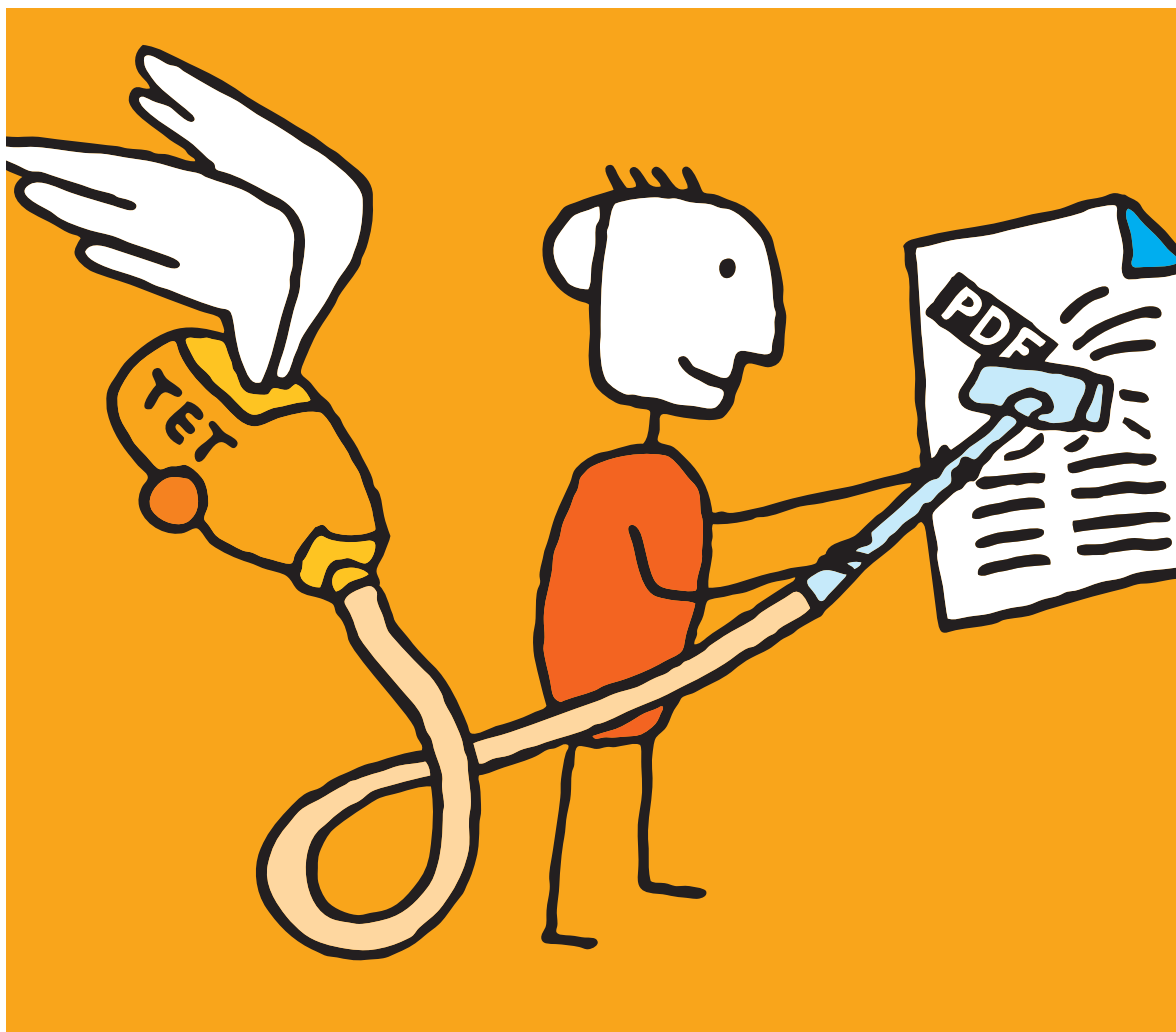


Text Extraction Toolkit (TET)

Version 4.2

**Toolkit for extracting Text, Images,
and Metadata from PDF Documents**



Copyright © 2002–2013 PDFlib GmbH. All rights reserved.
Protected by European and U.S. patents.

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
Support for commercial PDFlib licensees: 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, PostScript, and XMP are trademarks of Adobe Systems Inc. AIX, IBM, OS/390, WebSphere, iSeries, and zSeries are trademarks of International Business Machines Corporation. ActiveX, Microsoft, OpenType, and Windows 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 trademarks of Sun Microsystems, Inc. HKS is a registered trademark of the HKS brand association: Hostmann-Steinberg, K+E Printing Inks, Schmincke. Other company product and service names may be trademarks or service marks of others.

TET contains modified parts of the following third-party software:

Zlib compression library, Copyright © 1995-2012 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

ICU International Components for Unicode, Copyright © 1995-2012 International Business Machines Corporation and others

TET 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 14
- 1.4 What's new? 15

2 TET Command-Line Tool 17

- 2.1 Command-Line Options 17
- 2.2 Constructing TET Command Lines 20
- 2.3 Command-Line Examples 21
 - 2.3.1 Extracting Text 21
 - 2.3.2 Extracting Images 21
 - 2.3.3 Generating TETML 22
 - 2.3.4 Advanced Options 22

3 TET Library Language Bindings 23

- 3.1 Exception Handling 23
- 3.2 C Binding 24
- 3.3 C++ Binding 26
- 3.4 COM Binding 28
- 3.5 Java Binding 29
- 3.6 .NET Binding 31
- 3.7 Objective-C Binding 32
- 3.8 Perl Binding 34
- 3.9 PHP Binding 35
- 3.10 Python Binding 37
- 3.11 REALbasic Binding 38
- 3.12 Ruby Binding 39
- 3.13 RPG Binding 41

4 TET Connectors 43

- 4.1 Free TET Plugin for Adobe Acrobat 43
- 4.2 TET Connector for the Lucene Search Engine 45
- 4.3 TET Connector for the Solr Search Server 48

- 4.4 TET Connector for Oracle 49
- 4.5 TET PDF IFilter for Microsoft Products 52
- 4.6 TET Connector for the Apache TIKI Toolkit 54
- 4.7 TET Connector for MediaWiki 56

5 Configuration 59

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

6 Text Extraction 69

- 6.1 PDF Document Domains 69
- 6.2 Page and Text Geometry 73
- 6.3 Chinese, Japanese, and Korean Text 79
 - 6.3.1 CJK Encodings and CMaps 79
 - 6.3.2 Word Boundaries for CJK Text 79
 - 6.3.3 Vertical Writing Mode 79
 - 6.3.4 CJK Decompositions: Narrow, wide, vertical, etc. 80
- 6.4 Bidirectional Arabic and Hebrew Text 82
 - 6.4.1 General Bidi Topics 82
 - 6.4.2 Postprocessing Arabic Text 82
- 6.5 Content Analysis 84
- 6.6 Layout Analysis 88

7 Advanced Unicode Handling 91

- 7.1 Important Unicode Concepts 91
- 7.2 Unicode Preprocessing (Filtering) 94
 - 7.2.1 Filters for all Granularities 94
 - 7.2.2 Filters for Granularity Word and above 95
- 7.3 Unicode Postprocessing 97
 - 7.3.1 Unicode Folding 97
 - 7.3.2 Unicode Decomposition 100
 - 7.3.3 Unicode Normalization 104
- 7.4 Supplementary Characters and Surrogates 106
- 7.5 Unicode Mapping for Glyphs 107

8 Image Extraction 113

- 8.1 Image Extraction Basics 113
- 8.2 Image Merging and Filtering 115
- 8.3 Placed Images and Image Resources 117
- 8.4 Page-based and Resource-based Image Loops 118
- 8.5 Geometry of Placed Images 119

8.6 Restrictions and Caveats 121

9 TET Markup Language (TETML) 123

- 9.1 Creating TETML 123**
- 9.2 Controlling TETML Details 127**
- 9.3 TETML Elements and the TETML Schema 131**
- 9.4 Transforming TETML with XSLT 134**
- 9.5 XSLT Samples 137**

10 TET Library API Reference 141

- 10.1 Option Lists 141**
- 10.2 Option List Syntax 141**
- 10.3 Basic Types 144**
- 10.4 Geometric Types 147**
- 10.5 Encoding Names 147**
- 10.6 General Functions 148**
 - 10.6.1 Option Handling 148**
 - 10.6.2 Setup 151**
 - 10.6.3 PDFlib Virtual Filesystem (PVF) 152**
 - 10.6.4 Unicode Conversion Function 155**
 - 10.6.5 Exception Handling 157**
 - 10.6.6 Logging 159**
- 10.7 Document Functions 161**
- 10.8 Page Functions 169**
- 10.9 Text and Metrics Retrieval Functions 177**
- 10.10 Image Retrieval Functions 181**
- 10.11 TET Markup Language (TETML) Functions 185**
- 10.12 pCOS Functions 188**

A TET Library Quick Reference 193

B Revision History 195

Index 197

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 17, 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 23, corresponding to their preferred development environment, and review the installed examples. On Windows, the TET programming examples are accessible via the Start menu (for COM and .NET) or in the installation directory (for other language bindings).

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 which is encoded with legacy encodings TET requires the corresponding CMap files for mapping CJK encodings to Unicode. The CMap files are contained in all TET packages, and are installed in the *resource/cmap* directory within the TET installation directory. On 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:

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

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

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

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.

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

Working with a license file. PDFlib products read license keys from a license file, which is a text file according to the format shown below. You can use the template *licensekeys.txt* which is contained in all TET distributions. Lines beginning with a '#' character contain comments and 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 4.2 ...your license key...
```

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

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

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

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

- ▶ Supply the *--teto*pt option of the TET command-line tool and supply the *licensefile* option with the name of a license file:

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

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

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

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

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

On i5/iSeries the license file can be specified as follows (this command can be specified in the startup program *QSTRUP* and will work for all PDFlib GmbH products):

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


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

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

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

```
HKLM\SOFTWARE\PDFlib\TET4\license  
HKLM\SOFTWARE\PDFlib\TET4\4.2\license
```

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

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

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

Default file search paths. On Unix, Linux, OS X and i5/iSeries systems some directories will be searched for files by default even without specifying any path and directory names. Before searching and reading the UPR file (which may contain additional search paths), the following directories will be searched:

```
<rootpath>/PDFlib/TET/4.2/resource/cmap  
<rootpath>/PDFlib/TET/4.2/resource/codelist  
<rootpath>/PDFlib/TET/4.2/resource/glyphlist  
<rootpath>/PDFlib/TET/4.2  
<rootpath>/PDFlib/TET  
<rootpath>/PDFlib
```

On Unix, Linux, and OS X *<rootpath>* will first be replaced with */usr/local* and then with the HOME directory. On i5/iSeries *<rootpath>* is empty.

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

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

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

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

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

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

- In 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 Ruby:

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

- In Perl, Python and PHP:

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

- In RPG:

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

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

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

PDFlib GmbH, Licensing Department
Franziska-Bilek-Weg 9, 80339 München, Germany
www.pdflib.com
phone +49 • 89 • 452 33 84-0
fax +49 • 89 • 452 33 84-99
Licensing contact: sales@pdflib.com
Support for PDFlib licensees: support@pdflib.com

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

1.1 Overview of TET Features

Supported PDF input. TET has been tested against millions of PDF test files from various sources. It accepts PDF 1.0 up to PDF 1.7 extension level 8 and PDF 2.0, corresponding to Acrobat 1-XI including encrypted documents. TET attempts to repair various kinds of malformed and damaged PDF 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.
- ▶ Special character forms (e.g. wide, narrow, prerotated glyphs for vertical text) can optionally be converted (folded) to the corresponding regular forms
- ▶ Horizontal and vertical writing modes are supported.
- ▶ CJK font names are normalized to Unicode.

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

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

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

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

Image extraction. TET extracts raster images from PDF. Adjacent parts of a segmented image 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 are extracted in TIFF, JPEG, JPEG 2000, or JBIG2 format.

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

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

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.

- ▶ Recognize superscript, subscript and dropcaps (large initial characters at the start of a paragraph)

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. The syntax of pCOS query path is described separately in the pCOS Path Reference.

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.

- ▶ The TET Plugin is a free extension for Adobe Acrobat which makes TET available for interactive use (see Section 4.1, »Free TET Plugin for Adobe Acrobat«, page 43, for more information).

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 loop for extracting text from a PDF document.
- ▶ The *extract_images* sample extracts the images on each page and reports about their geometry and other properties.
- ▶ The *image_resources* sample demonstrates the basic loop for extracting images from a PDF document in a resource-oriented way (no geometric information available).
- ▶ The *dumper* sample shows the use of the integrated pCOS interface for querying general information about a PDF document.
- ▶ The *fontfilter* sample shows how to process font-related information, such as font name and font size.
- ▶ The *glyphinfo* sample demonstrates how to retrieve detailed information about glyphs (font, size, position, etc.) as well as text attributes such as *dropcap*, *shadow*, *hyphenation*, etc.
- ▶ 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.

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 a 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 the pCOS Path Reference which is included in the TET package.

1.4 What's new?

What's new in TET 4.0? The following features are new or considerably improved in TET 4.0:

- ▶ performance enhancements: faster for many classes of documents
- ▶ higher speed and smaller memory consumption for very large documents up to hundreds of thousands of pages
- ▶ extract right-to-left and bidirectional text for Arabic, Hebrew, etc.
- ▶ Unicode postprocessing with normalization, folding, and decomposition controls
- ▶ improved shadow removal, word boundary detection, and dehyphenation
- ▶ improved super- and subscript detection
- ▶ workarounds for non-conforming PDF documents to enhance robustness
- ▶ enhanced repair mode for successfully extracting text from damaged PDF
- ▶ More information in XML output (TETML): dehyphenation, dropcap, shadow, and super/subscript; coordinates in topdown system, PDF/A-2, PDF/E, font subsets,
- ▶ improved C++ and Perl language bindings

What's new in TET 4.1? The following features are new or considerably improved in TET 4.1:

- ▶ support for PDF 1.7 extension level 8 (encryption method specified in ISO 32000-2)
- ▶ updated to pCOS interface 8 with more pseudo objects (e.g. font details) and clarified handling of encrypted attachments
- ▶ additional information about PDF documents and fonts in TETML output
- ▶ new language bindings for Objective-C and Ruby
- ▶ word boundary detection for ideographic CJK text improved (suboption *ideographic*)
- ▶ new API functions *TET_convert_to_unicode()* and *TET_info_pvf()*
- ▶ updated connectors for Lucene and Solr

- ▶ new connector for the TIKA toolkit for detecting and extracting metadata and structured text content
- ▶ improved performance, especially for large documents
- ▶ variety of bug fixes and minor improvements

What's new in TET 4.2? The following features are new or considerably improved in TET 4.2:

- ▶ enhanced repair mode for damaged PDF and improved robustness against various kinds of malformed data;
- ▶ improved word boundary detection for ideographic CJK text and implemented the page option *ideographic*;
- ▶ implemented the new page option keyword *docstyle=cad*;
- ▶ extract images in JBIG2 format;
- ▶ improved image merging to cover more flavors of PDF images;
- ▶ made image merging more robust against malformed PDF images;
- ▶ improved the ordering of placed images in TETML;
- ▶ optionally omit ICC profiles from extracted images;
- ▶ optionally use LZW compression for extracted TIFF images as alternative to Flate (also known as »Adobe Flate« compression).

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 141. 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 20. Response files are only recognized before the -- option and before the first filename. Response files can not be used to replace the parameter for another option, but only complete option/parameter combinations.
--docopt	<option list>	Additional option list for <code>TET_open_document()</code> (see Table 10.8, page 162). 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 last specifies the last page, last-1 the page before the last page, etc. Default: 1
--format	utf8 utf16	Specifies the format for text output (default: utf8): utf8 UTF-8 with BOM (byte order mark) utf16 UTF-16 in native byte ordering with BOM This option does not affect TETML output which will always be created in UTF-8.
--help, -? (or no option)		Display help with a summary of available options.
--image ² -i		Extract images from the document. The naming scheme for extracted images depends on the --imageloop option.

Table 2.1 TET command-line options

option	parameters	function
--imageloop	page resource	Specifies the kind of enumeration loop for extracting images with the --image option (default: resource if --tetml is specified, otherwise page): page Enumerate all images on the selected pages. Images which are placed multiply will be extracted multiply. Extracted images will be named according to the following pattern: <filename>_p<pagenumber>_<imagenumber>.[tif jpg jpx jbig2] resource Enumerate all (merged) image resources in the document. Each image resource will be extracted only once, regardless of the number of occurrences in the document. The --firstpage and --lastpage options will be ignored for extracting images. Extracted images will be named according to the following pattern: <filename>_I<imageid>.[tif jpg jpx jbig2] Note: I<imageid> is also used in the TETML attribute Image/@id.
--imageopt	<option list>	Additional option list for TET_write_image_file() (see Table 10.17, page 183)
--lastpage -l	<integer> last	The number of the page where content extraction will finish. The keyword last specifies the last page, last-1 the page before the last page, etc. Default: last
--outfile -o	<filename>	(Not allowed if multiple input file names are supplied) File name for text or TETML output. The file name »-« can be used to designate standard output provided only a single input file has been supplied. Default: name of the input file, with .pdf or .PDF replaced with .txt (for text output) or .tetml (for TETML output).
--pagecount		Print the number of pages in the document, i.e. the value of the pCOS path length: pages, to stdout or the file provided with --outfile .
--pageopt	<option list>	Additional 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 10.10, page 169 and Table 10.18, page 185, for a list of available options. For text output the option granularity will always be set to page.
--password, -p	<password>	User, master or attachment password for encrypted documents. In some situations the shrug feature can be used to index protected documents without supplying a password (see Section 5.1, »Extracting Content from protected PDF«, page 59).
--samedir		Create output files in the same directory as the input file(s).
--searchpath ¹ -s	<path>...	Name of one or more directories where files (e.g. CMaps) will be searched. Default: installation-specific
--targetdir -t	<dirname>	Output directory for generated text, TETML, and image files. The directory must exist. This option is ignored if --samedir is specified. Default: . (i.e. the current working directory)
--tetml -m	glyph 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 will always be created in UTF-8. The supplied parameter selects one of several variants (see Section 9.2, »Controlling TETML Details«, page 127): 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 all glyph 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 10.2, page 148). The option outputformat will be ignored (use --format instead).

Table 2.1 TET command-line options

option	parameters	function
--text²		(Can not be combined with --tetml) Extract text from the document (enabled by default)
--verbose -v	0 1 2 3	verbosity level (default: 1): 0 no output at all 1 emit only errors 2 emit errors and file names 3 detailed reporting
--version, -V		Print the TET version number.

1. This option can be supplied more than once.
2. The option --image disables text extraction, but it can be combined with --text and --tetml.

2.2 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. Wildcards will be evaluated in the current directory, not any searchpath directory.

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

Response files. In addition to options supplied directly on the command-line, options can also be supplied in a response file. The contents of a response file 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.

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

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

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

2.3 Command-Line Examples

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

2.3.1 Extracting Text

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

```
tet file.pdf
```

Exclude the first and last page from text extraction:

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

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

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

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

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

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

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

2.3.2 Extracting Images

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

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

2.3.3 Generating TETML

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

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

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

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

Extract images and generate TETML in a single call:

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

Generate TETML output with topdown coordinates:

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

2.3.4 Advanced Options

Supply the document option *checkglyphlists* to improve Unicode mapping for certain kinds of TeX-generated PDF documents:

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

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

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

Disable punctuation as word boundary:

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

3 TET Library Language Bindings

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, »Extracting Content from protected PDF«, page 59);
- ▶ Trying to open a PDF document with a wrong file name;
- ▶ Trying to open a PDF document which is damaged beyond repair.

TET signals such errors by returning a 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 24).

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 *delete()*, *get_apiname()*, *get_errnum()*, and *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. *open_document()* or *open_page()*, can fail without throwing an exception (they will return `-1` in case of an error). In this situation the functions *get_errnum()*, *get_errmsg()*, and *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

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

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

Exception handling. The TET API provides a mechanism for acting upon exceptions thrown by the library in order to compensate for the lack of native exception handling in the C language. Using the *TET_TRY()* and *TET_CATCH()* macros client code can be set up such that a dedicated piece of code is invoked for error handling and cleanup when an exception occurs. These macros set up two code sections: the try clause with code which may throw an exception, and the catch clause with code which acts upon an exception. If any of the API 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 *auto* 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 is interpreted as UTF-8.
- ▶ If the option starts with an EBCDIC UTF-8 BOM (`\x57\x8B\xAB`) it is interpreted as EBCDIC UTF-8.
- ▶ If no BOM is found, the string is treated as *winansi* (or *ebcdic* on EBCDIC-based platforms).

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

Using TET as a DLL loaded at runtime. While most clients will use TET as a statically bound library or a dynamic library which is bound at link time, you can also load the DLL at runtime and dynamically fetch pointers to all API functions. This is especially useful to load the DLL only on demand, and on MVS where the library is customarily loaded as a DLL at runtime without explicitly linking against TET. TET supports a special mechanism to facilitate this dynamic usage. It works according to the following rules:

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

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

3.3 C++ Binding

Note For applications written in C++ we recommend to access the TET .NET DLL directly instead of via the C++ binding (except for cross-platform applications which should use the C++ binding). The TET distribution contains C++ sample code for use with .NET CLI which demonstrates this combination.

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*. Since *tet.hpp* contains a template-based implementation no corresponding *tet.cpp* module is required. 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.

String handling in C++. TET's template-based string handling approach supports the following usage patterns with respect to string handling:

- ▶ Strings of the C++ standard library type *std::wstring* are used as basic string type. They can hold Unicode characters encoded as UTF-16 or UTF-32. This is the default behavior since TET 4.0 and the recommended approach for new applications unless custom data types (see next item) offer a significant advantage over *wstrings*.
- ▶ Custom (user-defined) data types for string handling can be used as long as the custom data type is an instantiation of the *basic_string* class template and can be converted to and from Unicode via user-supplied converter methods.
- ▶ Plain C++ strings can be used for compatibility with existing C++ applications which have been developed against TET 3.0 or earlier versions. This compatibility variant is only meant for existing applications (see below for notes on source code compatibility).

The new interface assumes that all strings passed to and received from TET methods are native *wstrings*. Depending on the size of the *wchar_t* data type, *wstrings* are assumed to contain Unicode strings encoded as UTF-16 (2-byte characters) or UTF-32 (4-byte characters). Literal strings in the source code must be prefixed with *L* to designate wide strings. Unicode characters in literals can be created with the *\u* and *\U* syntax. Although this syntax is part of standard ISO C++, some compilers don't support it. In this case literal Unicode characters must be created with hex characters.

*Note On EBCDIC-based systems the formatting of option list strings for the wstring-based interface requires additional conversions to avoid a mixture of EBCDIC and UTF-16 wstrings in option lists. Convenience code for this conversion and instructions are available in the auxiliary module *utf16num_ebcdic.hpp*.*

Adjusting applications to the new C++ binding. Existing C++ applications which have been developed against TET 3.0 or earlier versions can be adjusted as follows:

- ▶ Since the TET C++ class now lives in the *pdflib* namespace the class name must be qualified. In order to avoid the *pdflib::TET* construct client applications should add the following before using TET methods:

```
using namespace pdflib;
```

- ▶ Switch the application's string handling to *wstrings*. This includes data from external sources. However, string literals in the source code (including option lists) must also be adjusted by prepending the *L* prefix, e.g.

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

- ▶ Suitable *wstring*-capable methods (*wcerr* etc.) must be used to process TET error messages and exception strings (*get_errmsg()* method in the *TET* and *TET::Exception* classes).
- ▶ The *tet.cpp* module is no longer required for the TET C++ binding. Although the TET distribution contains a dummy implementation of this module, it should be removed from the build process for TET applications.

Full source code compatibility with legacy applications. The new C++ binding has been designed with application-level source code compatibility mind, but client applications must be recompiled. The following aids are available to achieve full source code compatibility for legacy applications:

- ▶ Disable the *wstring*-based interface as follows before including *tet.hpp*:

```
#define TETCPP_TET_WSTRING 0
```

- ▶ Disable the *pdflib* namespace as follows before including *tet.hpp*:

```
#define TETCPP_USE_PDFLIB_NAMESPACE 0
```

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

Native C++ exceptions thrown by TET routines will behave as expected. The following code fragment will catch exceptions thrown by TET:

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

Using TET as a DLL loaded at runtime. Similar to the C language binding the C++ binding allows you to dynamically attach TET to your application at runtime (see »Using TET as a DLL loaded at runtime«, page 25). Dynamic loading can be enabled as follows when compiling the application module which includes *tet.hpp*:

```
#define TETCPP_DL 1
```

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

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 4.2 32-bit\\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 31) 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 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"));
```

Using TET in J2EE application servers and Servlet containers. TET is perfectly suited for server-side Java applications. The TET distribution contains sample code and configuration for using TET in J2EE environments. The following configuration issues must be observed:

- ▶ The directory where the server looks for native libraries varies among vendors. Common candidate locations are system directories, directories specific to the underlying Java VM, and local server directories. Please check the documentation supplied by the server vendor.
- ▶ Application servers and Servlet containers often use a special class loader which may be restricted or uses a dedicated classpath. For some servers it is required to define a special classpath to make sure that the TET package will be found.

More detailed notes on using TET with specific Servlet engines and application servers can be found in additional documentation in the J2EE directory of the TET distribution.

Unicode and legacy encoding conversion. For the convenience of TET users we list some useful string conversion methods here. Please refer to the Java documentation for more details. The following constructor creates a Unicode string from a byte array, using the platform's default encoding:

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

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

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

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

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

Javadoc documentation for TET. The TET package contains Javadoc documentation for TET. The Javadoc contains only abbreviated descriptions of all TET API methods; please refer to Section 10, »TET Library API Reference«, page 141, for more details.

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

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

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

Exception handling. The TET 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

Note Detailed information about the various flavors and options for using TET with the .NET Framework can be found in the [PDFlib-in-.NET-HowTo.pdf](#) document which is contained in the distribution packages and also available on the [PDFlib Web site](#).

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.

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.

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*.

Using TET with C++ and CLI. .NET applications written in C++ (based on the *Common Language Infrastructure CLI*) can directly access the TET.NET DLL without using the TET C++ binding. The source code must reference TET as follows:

```
using namespace TET_dotnet;
```

3.7 Objective-C Binding

Although the C and C++ language bindings can be used with Objective-C¹, a genuine language binding for Objective-C is also available. The TET framework is available in the following flavors:

- *TET* for use on OS X
- *TET_ios* for use on iOS

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

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

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

or

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

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

- The value of the first parameter is provided directly after the method name, separated by a colon character.
- For each subsequent parameter the parameter's name and its value (again separated from each other by a colon character) must be provided. The parameter names can be found in Chapter 10, »TET Library API Reference«, page 141, and in *TET_objc.h*.

For example, the following line in the API description:

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

corresponds to the following Objective-C method:

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

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

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

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

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

¹ See developer.apple.com/library/mac/#documentation/Cocoa/Conceptual/ObjectiveC/Introduction/introObjectiveC.html


```

@try {
    ...some TET instructions...
}
@catch (TETException *ex) {
    NSString * errorMessage =
        [NSString stringWithFormat:@"TET error %d in '%@': %@",
        [ex get_errnum], [ex get_apiname], [ex get_errmsg]];
    UIAlertView *alert = [[UIAlertView alloc] init];
    [alert setMessageText: errorMessage];
    [alert runModal];
    [alert release];
}
@catch (NSEException *ex) {
    UIAlertView *alert = [[UIAlertView alloc] init];
    [alert setMessageText: [ex reason]];
    [alert runModal];
    [alert release];
}
@finally {
    [tet release];
}

```

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

3.8 Perl Binding

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

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

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

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

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

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

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

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

Windows. TET supports the ActiveState port of Perl 5 to Windows, also known as ActivePerl. The DLL *tetlib_pl.dll* and the modules *tetlib_pl.pm* and *PDFlib/TET.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\Perl5.10\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 $@;
```

3.9 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=php_tet.dll      ; for Windows
extension=php_tet.so       ; for Unix and OS X
extension=php_tet.sl       ; for HP-UX
```

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("php_tet.dll");        # for Windows
dl("php_tet.so");          # for Unix and OS X
dl("php_tet.sl");          # for HP-UX
```

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;  
}
```

3.10 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 and OS X: *tetlib_py.so*
- ▶ Windows: *tetlib_py.pyd*

Error Handling in Python. The Python binding translates TET exceptions 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 occurred:\n[%d] %s: %s" %
          ((tet.get_errnum()), tet.get_apiname(), tet.get_errmsg()))
```

3.11 REALbasic Binding

Installing the TET edition for REALbasic. TET for REALbasic (*TET.rbxx*) must be copied to a folder called *Plugins* in the same folder where the REALbasic application lives. TET for REALbasic is delivered in a single package and contains the following variants:

- ▶ OS X Carbon (PowerPC and Intel)
- ▶ Windows
- ▶ Linux

This means that you can use the OS X or Windows version to build applications for all supported platforms. When a stand-alone application is generated, REALbasic selects the appropriate parts of TET and includes only the platform-specific portion(s) in the generated application.

Additional REALbasic classes. TET adds two new classes to REALbasic's object hierarchy:

- ▶ The *TET* class contains all TET API methods.
- ▶ The *TETException* class, which is derived from *RuntimeException*, can be used to deal with exceptions thrown by TET (see below).

TET can be used to create GUI applications as well as console applications. Since TET is not a control it does not install a new icon in REALbasic's control palette. However, when TET is available, REALbasic will be aware of the TET class and its associated methods. For example, statement completion and parameter checking fully work for TET API methods.

Error handling in REALbasic. In case of an exception TET will throw a native REALbasic exception of the class *TETException*. TET Exceptions can be handled with standard REALbasic techniques: either use a *try/catch* block (this is recommended, but requires REALbasic 5.5 or above), or handle them in an Exception block. The latter is demonstrated in the following code fragment:

```
Exception err As TETException
    MsgBox("TET exception occurred in extractor sample: [" + _
        Str(err.get_errnum()) + "] " + err.get_apiname() + ": " + err.get_errmsg())
```

As shown in this example, REALbasic developers can access detailed error information by using the *TETException* methods for retrieving error number, error message, and the name of the API function which raised the exception.

3.12 Ruby Binding

Installing the TET Ruby edition. The Ruby¹ extension mechanism works by loading a shared library at runtime. For the TET binding to work, the Ruby interpreter must have access to the TET extension library for Ruby. This library (on Windows and Unix: *TET.so*; on OS X: *TET.bundle*) will usually be installed in the *site_ruby* branch of the local ruby installation directory, i.e. in a directory with a name similar to the following:

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

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

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

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

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

```
begin
    ...some TET instructions...
rescue TETException => pe
    print pe.backtrace.join("\n") + "\n"
    print "Error [" + pe.get_errnum.to_s + "] " + pe.get_apiname + ": " + pe.get_errmsg
    print " on page pageno" if (pageno != 0)
    print "\n"
rescue Exception => e
    print e.backtrace.join("\n") + "\n" + e.to_s + "\n"
ensure
    tet.delete() if tet
end
```

Ruby on Rails. Ruby on Rails² is an open-source framework which facilitates Web development with Ruby. The TET extension for Ruby can be used with Ruby on Rails. Follow these steps to run the TET examples for Ruby on Rails:

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

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

- Edit *config/routes.rb*:

```
...
# remember to delete public/index.html
```

1. See www.ruby-lang.org/en

2. See www.rubyonrails.org

```
root :to => "home#demo"
```

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

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

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

```
$ rails server
```

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

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

3.13 RPG Binding

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

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

Note The `%CHAR` and `%UCS2` functions use the current job's CCSID to convert strings from and to Unicode. The examples provided with TET are based on CCSID 37 (US EBCDIC). 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)
```

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

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

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

4 TET Connectors

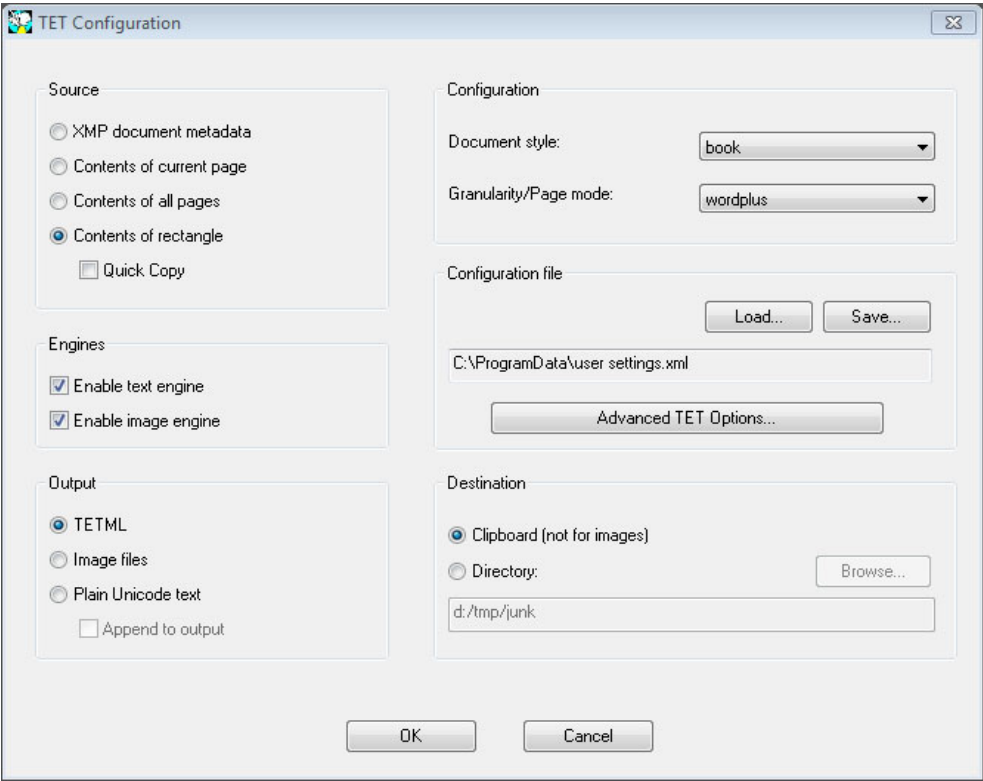
TET connectors provide the necessary glue code for interfacing TET with other software. TET connectors are based on the TET library or the TET command-line tool.

4.1 Free TET Plugin for Adobe Acrobat

This section discusses the TET Plugin, a freely available packaging of TET which can be used for testing in Adobe Acrobat and interactive use of TET with any PDF document. The TET Plugin works with Acrobat 8/9/X/XI 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 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

Fig. 4.1
Configuration panel for the TET Plugin



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

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

- ▶ The output can be customized to match different application requirements.
- ▶ TET is able to correctly interpret the text in many cases where Acrobat copies only garbage to the clipboard.
- ▶ Unknown glyphs (for which proper Unicode mapping cannot be established) 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, »Extracting Content from protected PDF«, page 59, for details). This is prepared in the Connector files, but you must manually enable this option.

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

- ▶ JDK 1.5 or later for Lucene 3.0.x
- ▶ A working installation of the *Ant* build tool
- ▶ The Lucene distribution with the Lucene core JAR file. The Ant build file distributed with TET expects the file *lucene-core-3.0.2.jar*, which is part of the Lucene distribution.
- ▶ An installed TET distribution package for Unix, Linux, OS X, 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-x.x.x.jar* to this directory.
- ▶ Optionally customize the settings by adding global, document-, and page-related TET options in *TetReader.java*. For example, the global option list can be used to supply a suitable search path for resources (e.g. if the CJK CMaps are installed in a directory different from the default installation).

The *PdfDocument.java* module demonstrates how to process PDF documents which are stored either on a disk file or in a memory buffer (e.g. supplied by a Web crawler). In the class *com.pdflib.tet.lucene.IndexPdfFiles* you can customize the target version of the Lucene engine with the *LUCENE_VERSION* variable.

- ▶ 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-x.x.x.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.

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 the pCOS Path Reference for more details on pCOS), e.g.

```
String objType = tet.pcos_get_string(tetHandle, "type:/Info/Subject");
if (!objType.equals("null"))
{
    doc.add(new 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 9.1, »Creating TETML«, page 123).

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

Indexing metadata fields. The TET connector for Solr indexes all standard document info fields. The key of each field 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, »Extracting Content from protected PDF«, page 59, for details). This is prepared in the Connector files, but you must manually enable this option.

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

- ▶ Starting with Oracle Text 11.1.0.7 the database can perform the required character set conversion. Please refer to the section »Using `USER_FILTER` with `Charset` and `Format Columns`« in the Oracle Text 11.1.0.7 documentation, available at 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 141, for option list details). This is especially useful for supplying the TET license key, e.g.:

```
TETOPT="license=aaaaaaa-bbbbbb-ccccc-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.pdflib.com/products/tet-pdf-ifilter.

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

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

- ▶ Supports Western text, Chinese, Japanese, and Korean (CJK) text and right-to-left languages such as Arabic and Hebrew
- ▶ Indexes protected documents and extracts text even from PDFs where Acrobat fails
- ▶ Supports Unicode folding, decomposition, and normalization
- ▶ Deployment: thread-safe, fast and robust, 32- and 64-bit versions
- ▶ Automatic script and language detection for improved search

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 SharePoint Server and FAST server for SharePoint
- ▶ Microsoft Search Server
- ▶ Microsoft SQL Server
- ▶ Microsoft Exchange Server
- ▶ Microsoft Site 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 Search is integrated in Windows Vista/7/8
- ▶ Windows Indexing Service

TET PDF IFilter is free for non-commercial use on desktop operating systems, which provides a convenient basis for test and evaluation.

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

- ▶ All PDF versions up to Acrobat XI, including ISO 32000
- ▶ Protected PDFs which do not require a password for opening the document
- ▶ Damaged PDF documents will be repaired

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

- ▶ Foldings preserve, remove or replace characters, e.g. remove punctuation or characters from irrelevant scripts.
- ▶ Decompositions replace a character with an equivalent sequence of one or more other characters, e.g. replace a Chinese character with its canonically equivalent Unicode character.
- ▶ Text can be converted to all four Unicode normalization forms, e.g. emit NFC form to match the requirements of a database.

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

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

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

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

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

- ▶ Document info entries, Dublin Core fields and other common XMP properties are mapped to 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.
- ▶ 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.

4.6 TET Connector for the Apache Tika Toolkit

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

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

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

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

- JDK 1.5 or later
- A working installation of the *Ant* build tool
- An installed TET distribution package for Unix, Linux, OS X, or Windows.
- A pre-built JAR file for Tika called *tika-app-1.x.jar*. Download information for this file can be found at the following location:

`tika.apache.org/download.html`

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

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

```
ant
```

If your Tika jar file has a name different from *tika-app-1.0.jar* you must supply the name of the jar file on the command line:

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

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

```
ant test
```

This command should produce the contents of `<tet-dir>/bind/data/FontReporter.pdf` as XHTML on the standard output. To test with a PDF file of your choice provide the Ant property *test.inputfile* on the command line as follows:

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

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

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

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

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

On Windows:

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

The following fragment should appear in the generated output:

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

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

On Unix and OS X systems:

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

On Windows:

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

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

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

4.7 TET Connector for MediaWiki

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

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

Requirements and installation. The TET distribution contains a TET connector which can be used to index PDF documents that are uploaded to a MediaWiki site. MediaWiki does not support PDF documents natively, but allows you to upload PDFs as »images«. The TET connector for MediaWiki indexes all PDF documents as they are uploaded. PDF documents which already exist in MediaWiki 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, OS X, 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 3.9, »PHP Binding«, page 35.
- ▶ 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' );
```

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 extracts text and metadata from the PDF document and appends it to the optional user-supplied comment which accompanies the uploaded document. The text is hidden in an HTML comment so that it will not be visible to users when they view the document comment. Since MediaWiki indexes the full contents of the comment (including the hidden full text) the text contents of the PDF 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.

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

- ▶ 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 Extracting Content from protected PDF

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

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

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

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

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

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

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

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

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

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

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

If all of the following conditions are true, the *shrug* feature 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 10.9, page 166. 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 search and the *searchpath* resource category. In addition to relative or absolute path names you can supply file names without any path specification to TET. The *searchpath* resource category can be used to specify a list of path names for directories containing the required data files. When TET must open a file it will first use the file name exactly as supplied, and try to open the file. If this attempt fails, TET will try to open the file in the directories specified in the *searchpath* resource category one after another until it succeeds. Multiple *searchpath* entries can be accumulated, and 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 initializes the *searchpath* resource category with a value read from the following registry keys:

```
HKLM\SOFTWARE\PDFlib\TET4\4.2\SearchPath
HKLM\SOFTWARE\PDFlib\TET4\SearchPath
HKLM\SOFTWARE\PDFlib\SearchPath
```

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

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

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

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

On i5/iSeries the *searchpath* resource category is initialized with the following values:

```
/PDFlib/TET/4.2/resource/cmap  
/PDFlib/TET/4.2/resource/codelist  
/PDFlib/TET/4.2/resource/glyphlst  
/PDFlib/TET/4.2  
/PDFlib/TET  
/PDFlib
```

Default file search paths. On Unix, Linux, OS X and i5/iSeries systems some directories will be searched for files by default even without specifying any path and directory names. Before searching and reading the UPR file (which may contain additional search paths), the following directories will be searched:

```
<rootpath>/PDFlib/TET/4.2/resource/cmap  
<rootpath>/PDFlib/TET/4.2/resource/codelist  
<rootpath>/PDFlib/TET/4.2/resource/glyphlst  
<rootpath>/PDFlib/TET/4.2/resource/fonts  
<rootpath>/PDFlib/TET/4.2/resource/icc  
<rootpath>/PDFlib/TET/4.2  
<rootpath>/PDFlib/TET  
<rootpath>/PDFlib
```

On Unix, Linux, and OS X *<rootpath>* will first be replaced with */usr/local* and then with the HOME directory. On i5/iSeries *<rootpath>* is empty.

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

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

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

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

- ▶ If the environment variable *TETRESOURCEFILE* is defined TET takes its value as the name of the UPR file to be read. If this file cannot be read an exception 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/4.2/tet.upr (on i5/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\TET4\4.2\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:

```
set_option("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:

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

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

Escape sequences for text files. Escape sequences are supported in all text files except UPR files 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 `\000` 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 141, 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=?
```

Use the following document option to remove all unmapped characters from the output:

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

Complex layouts. Some classes of documents often use very elaborate page layouts. For example, with magazines and periodicals TET may not be able to properly determine the relationship of columns on the page. In such situations it is possible to enhance the extracted text at the expense of processing time. Suitable options for this purpose are summarized in Section 6.6, »Layout Analysis«, page 88. See Table 10.12, page 174, for more details on relevant options.

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

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

```
unknown="true"
```

Do not set the `unknownchar` option to any common character since you may be unable to distinguish it from correctly mapped characters without checking the `unknown` field.

- Also to ensure text fidelity you may want to disable text extraction for text which is not visible on the page:

```
ignoreinvisibletext=true
```

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

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

- Start by extracting the text with default settings, and analyze the results. Identify the fonts which do not provide enough information for proper Unicode mapping.
- Write custom encoding tables and glyph name lists to fix problematic fonts. Use the PDFlib FontReporter plugin for analyzing the fonts and preparing Unicode mapping tables.
- Configure the custom mapping tables and extract the text again, using a larger number of documents. If there are still unmappable glyphs 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. Working with `granularity=glyph` may result in conflicts between the visual layout of text and the processed logical text created by TET (e.g. the two characters created by a ligature glyph may not fit into the same space as the ligature).

Corporate fonts with custom-encoded logos. In many cases corporate fonts containing custom logos 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 108) for a PDF containing the font, and locate mismatched 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 109, 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 PDF Document Domains

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

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

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

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

- ▶ How to display with Acrobat: page contents are always visible
- ▶ How to search a single PDF with Acrobat X/XI: *Edit, Find* or *Edit, [Advanced] Search*. TET may be able to process the text in documents where Acrobat does not correctly map glyphs to Unicode values. In this situation you can use the TET Plugin which is based on TET (see Section 4.1, »Free TET Plugin for Adobe Acrobat«, page 43). The TET Plugin offers its own search dialog via *Plug-Ins, PDFlib TET Plugin... TET Find*. However, it is not intended as a full-blown search facility.
- ▶ How to search multiple PDFs with Acrobat X/XI: *Edit, [Advanced] 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. Traditional document info entries are key/value pairs.

- ▶ How to display with Acrobat X/XI: *File, Properties...*
- ▶ How to search a single PDF with Acrobat X/XI: not available
- ▶ How to search multiple PDFs with Acrobat X/XI: click *Edit, [Advanced] Search* and *Show More Options* near the bottom of the dialog. In the *Look In:* pull-down select a folder of PDF documents and in the pull-down menu *Use these additional criteria* select one of *Date Created, Date Modified, Author, Title, Subject, Keywords*.
- ▶ Sample code for the TET library: *dumper* 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 X/XI: *File, Properties..., Custom* (not available in the free Adobe Reader)
- ▶ How to search with Acrobat X/XI: not available

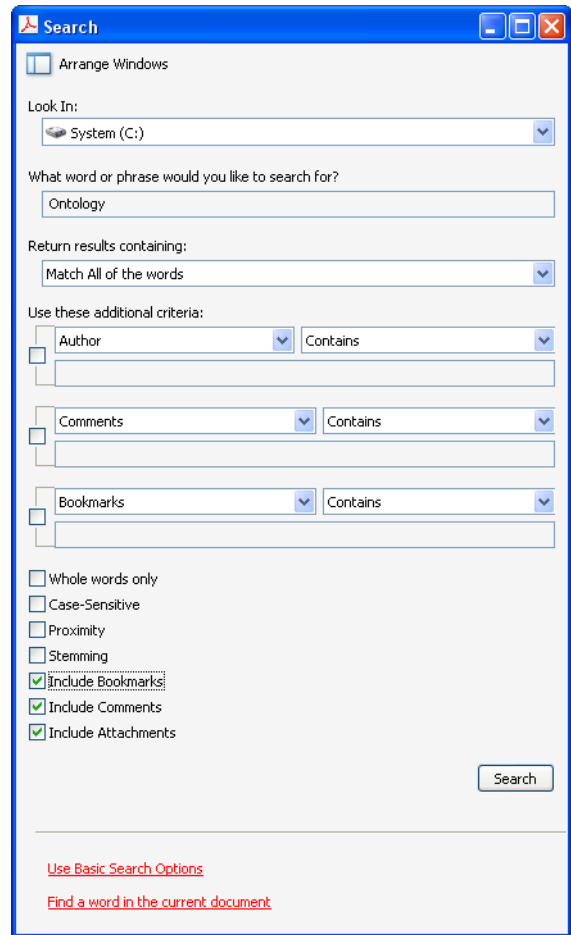


Fig. 6.1
Acrobat's advanced
search dialog

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

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

- ▶ How to display with Acrobat X/XI: *File, Properties..., Additional Metadata..* (not available in the free Adobe Reader)
- ▶ How to search a single PDF with Acrobat X/XI: not available
- ▶ How to search multiple PDFs with Acrobat X/XI: click *Edit, [Advanced] Search* and *Show More Options*. In the *Look In:* pull-down select a folder of PDF documents and in the pull-down menu *Use these additional criteria* select *XMP Metadata* (not available in 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 X: *Tools, Content, Edit Object*, select image, right-click, *Show Metadata...* (not available in the free Adobe Reader)
- ▶ How to display with Acrobat XI: *Tools, Content Editing, Edit Text & Images*, select image, right-click, *Show Metadata...* (not available in the free Adobe Reader)
- ▶ How to search with Acrobat X/XI: 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 X/XI: *Tools, Forms, Edit* (not available in the free Adobe Reader)
- ▶ How to search with Acrobat X/XI: 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 X/XI: *Comment, Comments List*
- ▶ How to search a single PDF with Acrobat X/XI: *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 X/XI: click *Edit, [Advanced] Search* and *Show More Options*. In the *Look In*: pull-down select a folder of PDF documents and in the pull-down menu *Use these additional criteria* select *Comments*.
- ▶ Sample code for the TET library: 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 X/XI: *View, Show/Hide, Navigation Panes, Bookmarks*
- ▶ How to search a single PDF with Acrobat X/XI: *Edit, [Advanced] Search* and check the box *Include Bookmarks*
- ▶ How to search multiple PDFs with Acrobat X/XI: click *Edit, [Advanced] 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 X/XI: *View, Show/Hide, Navigation Panes, Attachments*
- ▶ How to search with Acrobat X/XI: Use *Edit, [Advanced] 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. PDF packages and PDF portfolios are file attachments with additional properties.

- ▶ How to display with Acrobat X/XI: 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 X/XI: *Edit, Search Entire Portfolio*
- ▶ How to search multiple PDF packages with Acrobat X/XI: not available
- ▶ Sample code for the TET library: *get_attachments* mini sample
- ▶ TETML element: */TET/Document/Attachments/Attachment/Document*

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

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

6.2 Page and Text Geometry

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

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

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

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

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

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

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

Area of text extraction. By default, TET will extract all text from the visible page area. Using the *clippingarea* option of *TET_open_page()* (see Table 10.10, page 169) you can change this to any of the PDF page box entries (e.g. *TrimBox*). With the keyword *unlimited* all text regardless of any page boxes can be extracted. The default value *cropbox* instructs TET to extract text within the area which is visible in Acrobat.

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

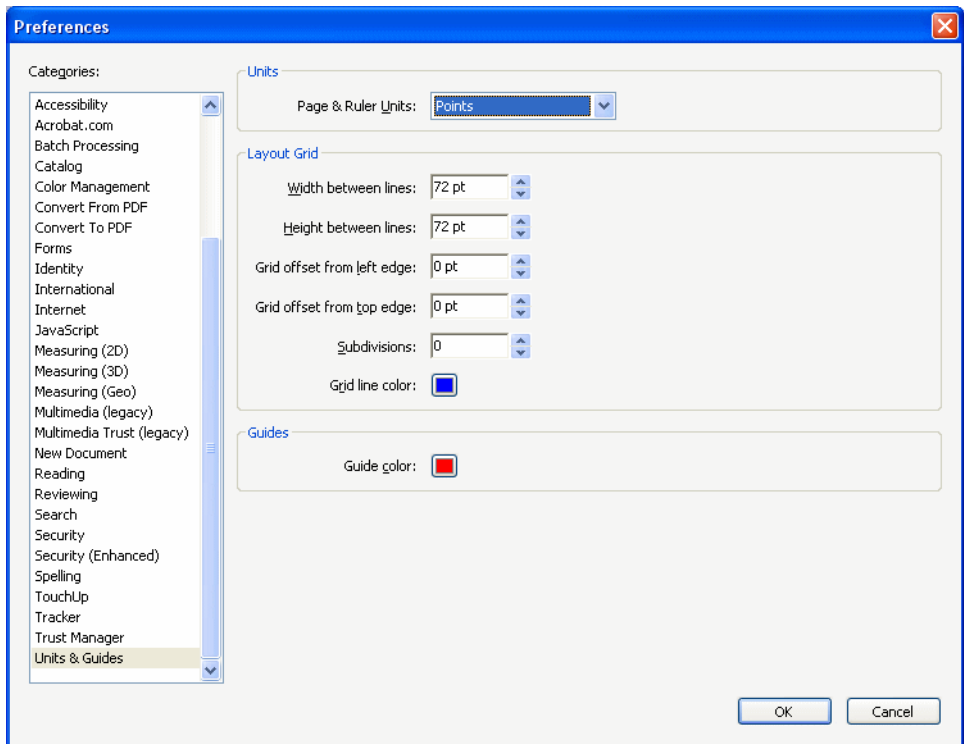


Fig. 6.2
Configuring coordinate display in Acrobat; use View, Cursor Coordinates to display cursor coordinates.

constructed by determining the union of all rectangles specified in the *includebox* option, and subtracting the union of all rectangles specified in the *excludebox* option. A 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.

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 10.15):

- ▶ 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 are reported 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 is treated as an artificial character, and has a *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

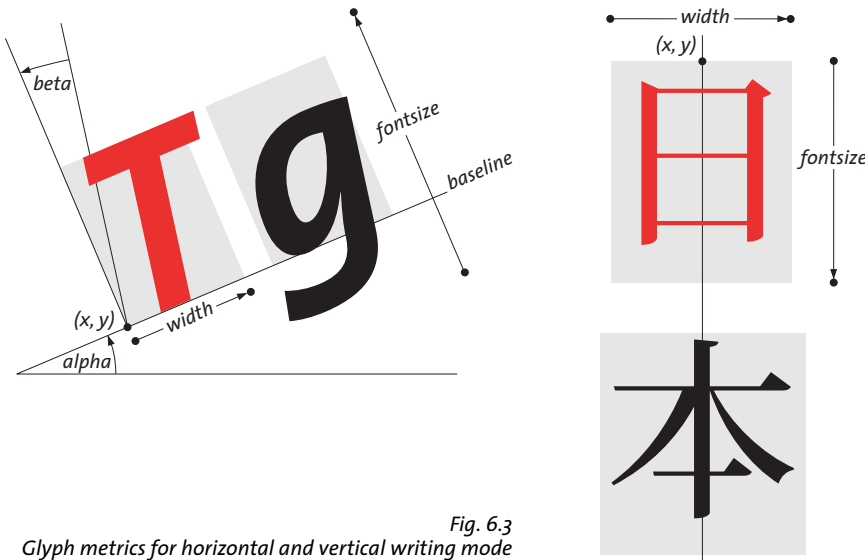


Fig. 6.3
Glyph metrics for horizontal and vertical writing mode
in TET's default coordinate system (topdown=false)

group of artificial characters comprises the continuation of a multi-character sequence (e.g. the second character of a ligature) and inserted separator characters. For artificial characters the position (x, y) will specify the endpoint of the most recent real character, the *width* is 0, and all other fields except *uv* are 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 *attributes* field contains information about the subscript, superscript, dropcap, or shadow status of the glyph as determined by TET's content analysis algorithms.
- ▶ 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 6.3, »Chinese, Japanese, and Korean Text«, page 79 for details on vertical writing mode). For artificial characters, which do not correspond to any glyph on the page, the point (x, y) specifies the end point of the most recent real character. The value of *y* is subject to the *topdown* page option.
- ▶ The *width* field specifies the width of a glyph according to the corresponding font metrics and text output parameters, such as character spacing and horizontal scaling. Since these parameters control the position of the next glyph, the distance between the reference points of two adjacent glyphs may be different from *width*. The *width* may be zero for non-spacing characters. On the other hand, the outline may actually be wider than the glyph's *width* value, e.g. for slanted text. The *width* is 0 for artificial characters.
- ▶ The 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

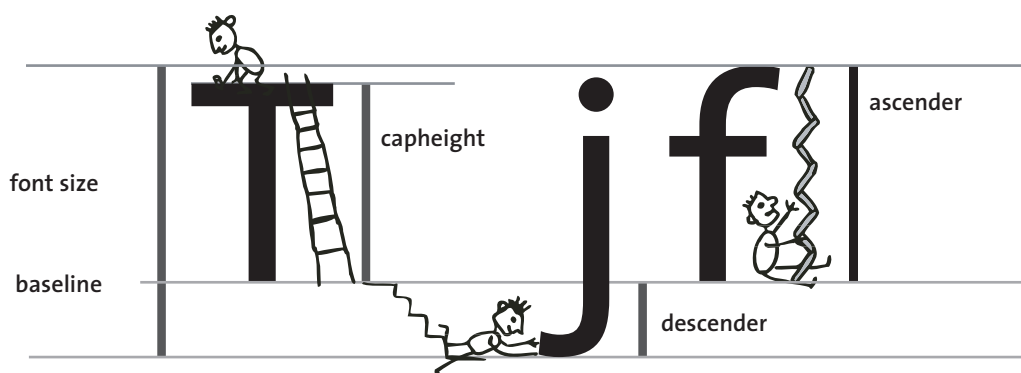


Fig. 6.4 Font-specific metrics

writing mode). Therefore, the angle *alpha* is 0° for standard horizontal text as well as for standard vertical text. The values of *alpha* and *beta* are subject to the *topdown* page option.

- ▶ The angle *beta* specifies any skewing which has been applied to the text, e.g. for slanted (italicized) text. The angle is measured against the perpendicular of *alpha*. It is 0° for standard upright text (for both horizontal and vertical writing mode). If the absolute value of *beta* is greater than 90° the text is mirrored at the baseline.
- ▶ The *fontid* field contains the pCOS ID of the font used for the glyph. It can be used to retrieve detailed font information, such as the font name, embedding status, writing mode (horizontal/vertical), etc. The pCOS Path Reference contains sample code for retrieving font details.
- ▶ The *fontsize* field specifies the size of the text in points. It is normalized and therefore always positive, especially with *topdown*=*{output}*.
- ▶ 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 10.15, page 179). Invisible text is extracted by default, but this can be changed with the *ignoreinvisibletext* option of *TET_open_page()*.

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

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

The *capheight* is the height of capital letters such as *T* or *H* in most Latin fonts. The *xheight* is the height of lowercase letters such as *x* in most Latin fonts. The *ascender* is the height of lowercase letters such as *f* or *d* in most Latin fonts. The *descender* is the distance from the baseline to the bottom of lowercase letters such as *j* or *p* in most Latin fonts. The descender is usually negative. The values of *xheight*, *capheight*, *ascender*, and *descender* are measured in thousands of the font size.

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

```

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

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

```

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

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

```

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

```

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

```

x_end = lrx = x + width
y_end = lry = y

```

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

```

urx = x + width * cos(alpha) - dir * height * sin(alpha)
ury = y + width * sin(alpha) + dir * height * cos(alpha)

```

with *dir=1* in the default case *topdown={output=false}* and *dir=-1* if *topdown={output=true}* (see »Top-down coordinate system«, page 73). The value of *height* depends on the fontsize and the font geometry. The following results in useful values for most common fonts (see »Font-specific metrics«, page 76, for retrieving the *ascender* value):

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

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

```

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

```

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

```

urx = x + width
ury = y + dir * height

```

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

$$x_{\text{end}} = x$$
$$y_{\text{end}} = y - \text{fontsize}$$

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

$$ulx = x - \text{width}/2 * \cos(\alpha)$$
$$uly = y - \text{width}/2 * \sin(\alpha)$$
$$lrx = ulx + \text{width} * \cos(\alpha) + \text{dir} * \text{fontsize} * \sin(\alpha)$$
$$lry = uly + \text{width} * \sin(\alpha) - \text{dir} * \text{fontsize} * \cos(\alpha)$$

with $\text{dir}=1$ in the default case $\text{topdown}=\{\text{output}=\text{false}\}$ and $\text{dir}=-1$ if $\text{topdown}=\{\text{output}=\text{true}\}$ (see »Top-down coordinate system«, page 73).

6.3 Chinese, Japanese, and Korean Text

6.3.1 CJK Encodings and CMaps

TET supports Chinese, Japanese, and Korean (CJK) text, and converts horizontal and vertical CJK text in arbitrary legacy encodings (CMaps) to Unicode. TET supports all of Adobe's CJK character collections:

- ▶ Simplified Chinese: *Adobe-GB1-5*
- ▶ Traditional Chinese: *Adobe-CNS1-5*
- ▶ 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. CJK font names encoded with locale-specific encodings (e.g. Japanese font names encoded in Shift-JIS) are normalized to Unicode.

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

6.3.2 Word Boundaries for CJK Text

Word boundary detection for CJK text can be controlled with the *ideographic* page option:

- ▶ With *ideographic=split* ideographic characters always constitute a word boundary, i.e. single ideographs are returned in *granularity=word*. While ideographic CJK characters are considered as word boundaries, Katakana characters are not treated as word boundaries.
- ▶ With *ideographic=keep* ideographic characters generally don't constitute a word boundary. Punctuation and the transition between ideographic and non-ideographic characters still constitute a word boundary. For *granularity=word* ideographic comma *U+3001* and ideographic full stop *U+3002* also constitute word boundaries. For *granularity=page* no line separator is inserted at the end of a line.

For compatibility reasons the default value is *ideographic=split*, but it is strongly recommended to use *ideographic=keep* to improve text extraction for CJK text.

6.3.3 Vertical Writing Mode

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

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

```

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

```

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

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

- Text line detection for vertical text can often be improved with the following page option:

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

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

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

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












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

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

```
decompose={none}
```

Table 6.1 demonstrates the CJK decompositions along with examples. See Section 7.3.2, »Unicode Decomposition«, page 100, for more information on decompositions.

Table 6.1 CJK compatibility decomposition examples (suboptions for the decompose option)

decomposition name	description	affected Unicode characters	decompositions enabled (default)	decompositions disabled
narrow	Narrow (hankaku) compatibility characters	U+FF61-U+FFDC, U+FFE8-U+FFEE	 U+30F2	 U+FF66
small	Small forms for CNS 11643 compatibility	U+FE50-U+FE6B	 U+002C	 U+FE50
square	CJK squared font variants	U+3250, U+32CC-U+32CF, U+3300-U+3357, U+3371-U+33DF, U+337B-U+337F, U+33FF, U+1F131-U+1F14E, U+1F190, U+1F200, U+1F210-U+1F231	  U+30AD U+30ED	 U+3314
vertical	Vertical layout presentation forms	U+309F, U+30FF, U+FE10-U+FE19 U+FE30-U+FE48	 U+FE37	 U+007B
wide	Wide (zenkaku) compatibility forms	U+3000, U+FF01-U+FF60, U+FFE0-U+FFE6	 U+00A3	 U+FFE1

6.4 Bidirectional Arabic and Hebrew Text

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

6.4.1 General Bidi Topics

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

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

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

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

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

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

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

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

6.4.2 Postprocessing Arabic Text

Normalize Arabic presentation forms and decompose ligatures. Arabic characters exist in up to four different forms for isolated use, at the beginning, in the middle, or at the end of a word. These forms can have different Unicode values although semantically they represent the same character. By default, TET converts all presentation forms to the corresponding canonical forms. As shown in Table 6.2 the *decompose* option can be used to preserve presentation forms (see Section 7.3.2, »Unicode Decomposition«, page 100).

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

Table 6.2 Processing Arabic presentation forms with the decompose option

description and option list	before decomposition	after decomposition (in logical order)
Decompose final, initial, isolated, and medial presentation forms: no decompose option (default) or decompose=none or decompose= {final=_all medial=_all initial=_all isolated=_all}	<div>س</div> <div>U+FEB2</div>	<div>س</div> <div>U+0633</div>
	<div>س</div> <div>U+FEB3</div>	<div>س</div> <div>U+0633</div>
	<div>سر</div> <div>U+FD0E</div>	<div>س</div> <div>ر</div> <div>U+0633 U+0631</div>
	<div>س</div> <div>U+FEB4</div>	<div>س</div> <div>U+0633</div>
Preserve final, initial, isolated, and medial presentation forms: decompose= {final=_none medial=_none initial=_none isolated=_none}	<div>س</div> <div>U+FEB2</div>	<div>س</div> <div>U+FEB2</div>
	<div>س</div> <div>U+FEB3</div>	<div>س</div> <div>U+FEB3</div>
	<div>سر</div> <div>U+FD0E</div>	<div>سر</div> <div>U+FD0E</div>
	<div>س</div> <div>U+FEB4</div>	<div>س</div> <div>U+FEB4</div>

Remove Arabic Tatweel character. The Tatweel character U+o64o (also called kashida) is often used in Arabic text to stretch words so that they completely fill the line. Since the Tatweel doesn’t carry any text information itself it is usually not required in the extracted text. By default, TET removes Tatweel characters from the extracted text. As shown in Table 6.3 the *fold* option can be used to preserve Tatweel characters (see Section 7.3.1, »Unicode Folding«, page 97).

Table 6.3 Processing the Tatweel character U+o64o with the fold option

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

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

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

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

Separator characters are inserted between multiple words, lines, or 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* options of `TET_open_page()` (use `U+0000` to disable a separator), for example:

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

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

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

- ▶ A sophisticated algorithm analyzes the geometric relationship among glyphs to find character groups which together form a word. The algorithm takes into account a variety of properties and special cases in order to accurately identify words even in complicated layouts and for arbitrary text ordering on the page.
- ▶ Some characters, such as space and punctuation characters (e.g. colon, comma, full stop, parentheses) are considered a word boundary, regardless of their width and position. 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 6.5).

Note Word boundary detection for text with ideographic characters works differently; see Section 6.3.2, »Word Boundaries for CJK Text«, page 79, for more information.



Fig. 6.5
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 is removed and the first part of the word is combined with the part on the next line, provided there is at least one more line in the same zone. Dash characters (as opposed to hyphens) are left unmodified. The parts of a hyphenated word will not be modified, only the hyphen is removed. Dehyphenation can be disabled with the following option list for `TET_open_page()`:

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

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

Similarly, word processing applications sometimes support a feature for

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

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

If the Wordfinder is enabled, TET 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}
```

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

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

Introduction

this situation and recombines both characters to form the appropriate combined character.



6.6 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
```

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

The following types are available for the *docstyle* option (Table 6.4 contains typical examples for some document styles):

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

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

Complex layouts. Some classes of documents often use very elaborate page layouts. For example, with magazines and periodicals TET may not be able to properly determine the relationship of columns on the page. In such situations it is possible to en-

Table 6.4 Document styles

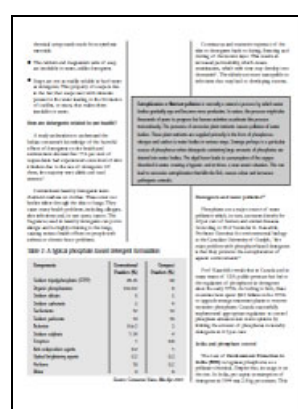
docstyle=book



docstyle=business



docstyle=fancy



docstyle=magazines



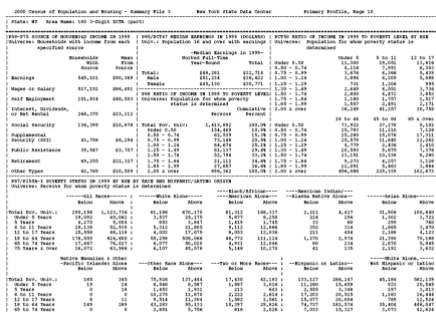
docstyle=papers



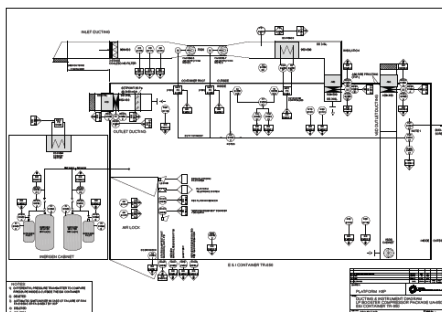
docstyle=science



docstyle=spacegrid



docstyle=cad



hance the extracted text at the expense of processing time. This can be controlled with the *structureanalysis* and *layoutanalysis* page options, e.g.

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

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

7 Advanced Unicode Handling

7.1 Important Unicode Concepts

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

www.unicode.org

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

- ▶ *Characters* are the smallest units which convey information in a language. Common examples are the letters in the Latin alphabet, Chinese ideographs, and Japanese syllables. Characters have a meaning; they are semantic entities.
- ▶ *Glyphs* are 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 7.1).

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

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

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

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 U+0049 I

VIII

Fig. 7.1
Relationship of glyphs
and characters

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

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

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

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

Table 7.1 Byte order marks for various Unicode encoding forms

Encoding form	Byte order mark (hex)	graphical representation in WinAnsi ¹
UTF-8	EF BB BF	ï»¿
UTF-16 big-endian	FE FF	þÿ
UTF-16 little-endian	FF FE	ÿþ
UTF-32 big-endian	00 00 FE FF	■ ■ þÿ
UTF-32 little-endian	FF FE 00 00	ÿþ ■ ■

1. The black square ■ denotes a null byte.

Composite characters and sequences. Some glyphs map to a sequence of multiple characters. For example, ligatures will be mapped to multiple characters according to their constituent characters. However, composite characters (such as the Roman numeral in Figure 7.1) may or may not be split, subject to information in the font and PDF as well as the *decompose* document option (see Section 7.3, »Unicode Postprocessing«, page 97).

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(s) can be obtained via `TET_get_char_info()`, including the information whether the character is the start or continuation of a sequence. Position information will only be returned for the first character of a sequence. Subsequent characters of a sequence will not have any associated position or width information, but must be processed in combination with the first character.

Characters without any corresponding glyph. Although every glyph on the page 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 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 artifacts without any corresponding glyph.

7.2 Unicode Preprocessing (Filtering)

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

7.2.1 Filters for all Granularities

The following filters can be used with all granularities.

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

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

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

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

Invisible text can be identified with the *textrendering* member of the *TET_char_info* structure returned by *TET.get_char_info()* (see Table 10.15, page 179), or with the *Glyph/@textrendering* attribute in TETML.

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

```
ignoreinvisibletext=true
```

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

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

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

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

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

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

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

7.2.2 Filters for Granularity Word and above

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

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

Hyphens used for splitting words across lines can be identified with the *attributes* member of the *TET_char_info* structure (see Table 10.15, page 179), or with the *Glyph/@hyphenation* attribute in TETML.

Dehyphenation can be disabled with the following page option:

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

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

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

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

The hyphens can be identified by the *TET_ATTR_DEHYPHENATION_ARTIFACT* flag of the *attributes* member in the *TET_char_info* structure returned by *TET_get_char_info()* (see Table 10.15, page 179), or in TETML with the *Glyph/@dehyphenation* attribute with value *artifact*.

Shadow removal. Redundant text which creates only visual artifacts such as shadow effects or artificial bold text will be removed.

Shadow and artificial bold text can be identified with the *attributes* member of the *TET_char_info* structure (see Table 10.15, page 179), or with the *Glyph/@shadow* attribute in TETML.

Shadow removal can be disabled with the following page option:

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

Unmapped glyphs. Glyphs which cannot be mapped to Unicode are replaced with a character in the Private Use Area (see section »Unmappable glyphs«, page 107). In some cases PDF documents do not contain enough information (or only inconsistent information for assigning a usable Unicode value to a glyph. In such cases the characters specified in the *unknownchar* document option will be assigned.

All PUA characters will be replaced with the Unicode unknown character U+FFFD by default. This behavior can be changed with the *fold* document option. The following option list removes all unknown characters, i.e. PUA characters and characters for which no usable Unicode value could be determined:

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

Unmapped glyphs (i.e. characters which are visible on the page, but cannot be extracted by TET) can be identified with the *unknown* member of the *TET_char_info* structure (see Table 10.15, page 179), or with the *Glyph/@unknown* attribute in TETML.

7.3 Unicode Postprocessing

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

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

7.3.1 Unicode Folding

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

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

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

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

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

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

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

Folding examples. Table 7.2 lists examples for the *fold* option which demonstrate various folding applications. The sample options must be supplied in the option list for *TET_open_document()*. TET can apply foldings to a selected subset of all Unicode characters. These are called Unicode sets; their syntax is discussed in »Unicode sets«, page 145.

Table 7.2 Examples for the fold option



description and option list	before folding	after folding
Remove all characters in a Unicode set		
Keep only characters in ISO 8859-1 (Latin-1) in the output, i.e. remove all characters outside the Basic Latin Block: fold={{[^U+0020-U+00FF] remove}}	A U+0104	n/a
Remove all non-alphabetic characters (e.g. punctuation, numbers): fold={{[:Alphabetic=No:] remove}}	7 U+0037	n/a
	A U+0041	A U+0041
Remove all characters except numbers: fold={{[^[:General_Category=Decimal_Number:]] remove}}	7 U+0037	7 U+0037
	A U+0041	n/a
Remove all unknown characters, i.e. PUA characters and characters for which no usable Unicode value could be determined (the remaining default foldings are re-enabled): fold={{[:Private_Use:] remove} {[U+FFFF] remove} default}	U+FFFF	n/a
Remove all dashed punctuation characters: fold={{[:General_Category=Dash_Punctuation:] remove}}	- U+002D	n/a
Remove all Bidi control characters: fold={{[:Bidi_Control:] remove}}	U+200E	n/a
Replace all characters in a Unicode set with a specific character		
Space folding: map all variants of Unicode space characters to U+0020: fold={{[:blank:] U+0020}}	U+00A0	U+0020
Dashes folding: map all variants of Unicode dash characters to U+002D: fold={{[:Dash:] U+002D}}	- U+2011	- U+002D
Replace all unassigned characters (i.e. Unicode code points to which no character is assigned) with U+FFFD: fold={{[:Unassigned:] U+FFFD}}	U+03A2	U+FFFD
Special handling for individual characters		
Preserve all hyphen characters at line breaks while keeping the remaining default foldings. Since these characters are identified internally in TET (as opposed to having a fixed Unicode property) the keyword <code>_dehyphenation</code> is used to identify the folding's domain: fold={{_dehyphenation preserve}}	- U+002D	- U+002D
Preserve Arabic Tatweel characters (which are removed by default): fold={{[U+0640] preserve}}	- U+0640	- U+0640
Replace various punctuation characters with their ASCII counterparts: fold={{ {[U+2018] U+0027} {[U+2019] U+0027} {[U+201C] U+0022} {[U+201D] U+0022} }}	" U+201C	" U+0022

Default foldings. Except for *granularity=glyph* TET applies all of the foldings listed in Table 7.3 by default. In order to combine custom foldings with the internal default foldings, the keyword *default* must be supplied after the custom folding options, e.g.

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

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

Table 7.3 Default values for the fold option

description and option list	sample input	output
Space folding: map all variants of Unicode space characters to U+0020: fold={{[:blank:] } U+0020}}	U+00A0	U+0020
Map all characters in the Private Use Area (PUA) to the unknown character (by default this is U+FFFD, but it can be changed with unknownchar option): fold={{[:Private_Use:] } unknownchar}}	 U+E001	 U+FFFD
Remove all hyphens in dehyphenated words: fold={{_dehyphenation } remove}}	- U+002D	n/a
Remove the Arabic Tatweel character: fold={{[U+0640] } remove}}	- U+0640	n/a
Remove all control characters as well as characters which are not assigned in Unicode (these foldings will always be performed after all other foldings when creating TETML output): fold={{[:Control:] } remove} {[:Unassigned:] } remove}}	U+000C U+03A2	n/a

7.3.2 Unicode Decomposition

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

www.unicode.org/versions/Unicode5.2.0/ch03.pdf#G729.

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

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

In the Unicode code charts¹ (but not the character tables) canonical mappings are marked with the symbol IDENTICAL TO \equiv _{U+2261}. The decomposition name *<canonical>* is implicitly assumed. Table 7.4 contains several examples.

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

decomposition name	description	before decomposition	after decomposition
canonical ¹	Canonical decomposition		
		Ä U+00C0	A ¨ U+0041 U+0300
		林 U+F9F4	林 U+6797
		Ω U+2126	Ω U+03A9
		ば U+3070	は U+306F U+3099
		ℵ U+FB2F	ℵ U+05D0 U+05B8

1. By default this decomposition is not applied to all characters in order to preserve certain characters; see »Default decompositions«, page 103, for details.

1. See www.unicode.org/Public/5.2.0/charts/

Compatibility decomposition. Characters which are compatibility equivalent represent the same abstract character, but may differ in appearance or behavior. Examples include isolated forms of Arabic characters (e.g. س_{U+0633}) vs. context-specific shaped forms

(e.g. س_{U+FEB2}, س_{U+FEB4}, س_{U+FEB3}). Compatibility equivalent characters differ in formatting. Removing this formatting information implies loss of information, but may simplify processing for certain types of applications (e.g. searching). Compatibility decompositions remove the formatting information.

In the Unicode code charts compatibility mappings are marked with the symbol

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

Note While all entries in Table 7.5 describe compatibility decompositions, the »compat« tag includes only »other« compatibility decompositions, i.e. those without a specific name.

Note Keep in mind that PDF documents may already map glyphs to the decomposed sequence instead of the non-decomposed Unicode value. In this situation the `decompose` option will not affect the output.

Decomposition examples. Decompositions in TET can be controlled with the document option `decompose`. A decomposition can be restricted to operate only on some, but not all Unicode characters. The subset on which a decomposition operates is called its domain. Table 7.5 lists the suboptions for all Unicode decompositions along with examples.

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

Disable all decompositions:

```
decompose={none}
```

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

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

Map all canonical equivalents to their counterparts:

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

The following option list enables the `circle` decomposition, but disables all other decompositions:

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

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

decomposition name	description	before decomposition	after decomposition (in logical order)
circle	Encircled characters	②1 U+3251	2 1 U+0032 U+0031
compat ¹	Other compatibility decompositions, e.g. common ligatures	fi U+FB01	f i U+0066 U+0069
final	Final presentation forms, especially Arabic	س U+FEB2	س U+0633
font	Font variants, e.g. mathematical set letters, Hebrew ligatures	© U+2102	© U+0043
fraction ¹	Vulgar fraction forms	¼ U+00BC	1 / 4 U+0031 U+2044 U+0034
initial	Initial presentation forms, especially Arabic	س U+FEB3	س U+0633
isolated	Isolated presentation forms, especially Arabic	سر U+FD0E	س ر U+0633 U+0631
medial	Medial presentation forms, especially Arabic	س U+FEB4	س U+0633
narrow	Narrow (hankaku) compatibility characters	㍻ U+FF66	㍻ U+30F2
nobreak	Non-breaking characters	U+00A0	U+0020
none	Disable all decompositions which are not explicitly specified in the decompose option list.	(leaves all characters unmodified)	
small	Small forms for CNS 11643 compatibility	’ U+FE50	’ U+002C
square	CJK squared font variants	㏻ ㏼ U+3314	㏻ ㏼ U+30AD U+30ED
sub ¹	Subscript forms	₁ U+2081	₁ U+0031
super ¹	Superscript forms	ₐ U+00AA ™ U+2122	ₐ U+0061 ™ M U+0054 U+004D
vertical	Vertical layout presentation forms	㍿ U+FE37	{ U+007B
wide	Wide (zenkaku) compatibility forms	£ U+FFE1	£ U+00A3

1. By default this decomposition is not applied to all characters in order to preserve certain characters; see »Default decompositions«, page 103, for details.

In contrast, the following option list enables all decompositions (since omitting the other options activates the default):

decompose={circle=_all}

Default decompositions. By default, all decompositions except *fraction* are enabled. While most default decompositions operate on the *_all* domain (i.e. they will be applied to all characters), some operate on smaller default domains according to Table 7.6. A straightforward way of dealing with decompositions is via normalization (see Section 7.3.3, »Unicode Normalization«, page 104). Since Unicode postprocessing is completely disabled for *granularity=glyph* no decompositions are active in this case.

Table 7.6 Default domains for Unicode decompositions (suboptions for the decompose option).

decomposition	default in TET
canonical	<div>canonical={ [U+0374 U+037E U+0387 U+1FBE U+1FEF U+1FFD U+2000 U+2001 U+2126 U+212A U+212B U+2329-U+232A] }</div> <div>The default domain includes canonical duplicates (singletons), but not other canonically equivalent characters. The default is not <i>_all</i> in order to preserve characters like \ddot{A} <small>U+00C4</small>.</div>
compat	<div>compat={ [U+FB00-U+FB17] }</div> <div>The default domain includes Latin and Armenian ligatures, but not other compatibility characters. The default is not <i>_all</i> in order to preserve characters like IJ <small>U+0132</small>.</div>
fraction	<div>fraction=_none</div> <div>Fractions are not decomposed by default because this would lead to undesired sequences of the digits for integer and fractional parts, e.g. client applications would wrongly interpret the sequence $9\frac{1}{2}$ (representing the numerical value 9.5) as $9\frac{1}{2}$ <small>U+0039 U+00BD</small> <small>U+0039 U+0031 U+2044 U+0032</small> which represents the numerical value $(91)/2=45.5$.</div>
sub super	<div>sub={ [U+208A-U+208E] }</div> <div>super={ [U+207A-U+207E] }</div> <div>The default domain includes only mathematical signs. Superscript and subscript digits are not decomposed by default to avoid problems with the numerical interpretation similar to those mentioned above for <i>fraction</i>. Characters such as the trademark sign TM <small>U+2122</small> will not be decomposed to $T M$ <small>U+0054 U+004D</small> by default.</div>
all others	<div>circle=_all final=_all ... vertical=_all wide=_all</div> <div>All other decompositions are enabled for all characters by default.</div>

7.3.3 Unicode Normalization

The Unicode standard defines four normalization forms which are based on the notions of canonical equivalence and compatibility equivalence (these are discussed in Section 7.3.2, »Unicode Decomposition«, page 100). All normalization forms put combining marks in a specific order and apply decomposition and composition in different ways:

- ▶ Normalization Form C (NFC) applies canonical decomposition followed by canonical composition.
- ▶ Normalization Form D (NFD) applies canonical decomposition.
- ▶ Normalization Form KC (NFKC) applies compatibility decomposition followed by canonical composition.
- ▶ Normalization Form KD (NFKD) applies compatibility decomposition.

The normalization forms are specified in Unicode Standard Annex #15 »Unicode Normalization Forms« (see www.unicode.org/versions/Unicode5.2.0/cho3.pdf#G21796 and www.unicode.org/reports/tr15/).

TET supports all four Unicode normalization forms. Unicode normalization can be controlled via the *normalize* document option, e.g.

`normalize=nfc`

TET does not apply normalization by default. Because of the possible interaction between the *decompose* and *normalize* options, setting the *normalize* option to a value different from *none* disables the default decompositions.

The choice of normalization form depends on the application’s requirements. For example, some databases expect text in NFC which also the preferred format for Unicode text on the Web. Table 7.7 demonstrates the effect of Normalization on various characters.

Table 7.7 Unicode normalization forms: examples

<i>before normalization</i>	<i>NFC</i>	<i>NFD</i>	<i>NFKC</i>	<i>NFKD</i>
Ä U+00C4	Ä U+00C4	A ¨ U+0041 U+0308	Ä U+00C4	A ¨ U+0041 U+0308
A ¨ U+0041 U+0308	Ä U+00C4	A ¨ U+0041 U+0308	Ä U+00C4	A ¨ U+0041 U+0308
¨ A U+0308 U+0041	¨ A U+0308 U+0041	¨ A U+0308 U+0041	¨ A U+0308 U+0041	¨ A U+0308 U+0041
fi U+FB01	fi U+FB01	fi U+FB01	f i U+0066 U+0069	f i U+0066 U+0069
3 5 U+0033 U+2075	3 5 U+0033 U+2075	3 5 U+0033 U+2075	3 5 U+0033 U+0035	3 5 U+0033 U+0035
Å U+212B	Å U+00C5	A ° U+0041 U+030A	Å U+00C5	A ° U+0041 U+030A
TM U+2122	TM U+2122	TM U+2122	T M U+0054 U+004D	T M U+0054 U+004D

Table 7.7 Unicode normalization forms: examples

before normalization	NFC	NFD	NFKC	NFKD
<div>Ⅳ</div> <div>U+2163</div>	<div>Ⅳ</div> <div>U+2163</div>	<div>Ⅳ</div> <div>U+2163</div>	<div>ⅠⅤ</div> <div>U+0049 U+0056</div>	<div>ⅠⅤ</div> <div>U+0049 U+0056</div>
<div>ᄃ</div> <div>U+FB48</div>	<div>ᄃ</div> <div>U+05E8 U+05BC</div>	<div>ᄃ</div> <div>U+05E8 U+05BC</div>	<div>ᄃ</div> <div>U+05E8 U+05BC</div>	<div>ᄃ</div> <div>U+05E8 U+05BC</div>
<div>가</div> <div>U+AC00</div>	<div>가</div> <div>U+AC00</div>	<div>ㄱㅏ</div> <div>U+1100 U+1161</div>	<div>가</div> <div>U+AC00</div>	<div>ㄱㅏ</div> <div>U+1100 U+1161</div>
<div>ぢ</div> <div>U+3062</div>	<div>ぢ</div> <div>U+3062</div>	<div>ぢ</div> <div>U+3061 U+3099</div>	<div>ぢ</div> <div>U+3062</div>	<div>ぢ</div> <div>U+3061 U+3099</div>
<div>10月</div> <div>U+32C9</div>	<div>10月</div> <div>U+32C9</div>	<div>10月</div> <div>U+32C9</div>	<div>10月</div> <div>U+0031 U+0030 U+6708</div>	<div>10月</div> <div>U+0031 U+0030 U+6708</div>

7.4 Supplementary Characters and Surrogates

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

TET interprets and maintains supplementary characters and provides access to the corresponding UTF-32 value even in language bindings where native Unicode strings support only UTF-16. The *uv* field returned by `TET_get_char_info()` for the leading surrogate value contains the corresponding UTF-32 value. This allows direct access to the UTF-32 value of a supplementary character even if you are working in a UTF-16 environment without any support for UTF-32.

Leading (high) surrogates and trailing (low) surrogates are maintained. The string returned by `TET_get_text()` contains two UTF-16 values.

7.5 Unicode Mapping for Glyphs

While text in PDF can be represented with a variety of font and encoding schemes, TET abstracts from glyphs and normalizes all text to Unicode characters, regardless of the original text representation in the PDF. Converting the information found in the PDF to the corresponding Unicode values is called *Unicode mapping*, and is crucial for understanding the semantics of the text (as opposed to rendering a visual representation of the text on screen or paper). In order to provide proper Unicode mapping TET consults various data structures which are found in the PDF document, embedded or external font files, as well as builtin and user-supplied tables. In addition, it applies several methods to determine the Unicode mapping for non-standard glyph names.

Despite all efforts there are still PDF documents where some text cannot be mapped to Unicode. In order to deal with these cases TET offers a number of configuration features which can be used to control Unicode mapping for problematic PDF files.

Unmappable glyphs. There are several reasons why text in a PDF cannot reliably be mapped to Unicode. For example, Type 1 fonts may contain unknown glyph names, and TrueType, OpenType, or CID fonts may be addressed with glyph ids without any Unicode values in the font or PDF. TET assigns a code point in the Private Use Area to such unmapped characters. The PUA values can be removed or replaced with the *fold* option. By default, PUA characters will be mapped to U+FFFD, the Unicode unknown character. Your code should be prepared for this character. If you don't care about Unicode mapping problems you can simply ignore U+FFFD, or use the following document option to remove it:

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

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

Summary of Unicode mapping controls. While TET implements many workarounds in order to process PDF documents which actually don't contain Unicode values so that it can successfully extract the text nevertheless. However, there are still documents where the text cannot be extracted since not enough information is available in the PDF and relevant font data structures. TET contains various configuration features which can be used to supply additional Unicode mapping information. These features are detailed in this section.

Using the *glyphmapping* option of *TET_open_document()* (see Section 10.7, »Document Functions«, page 161) 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 al-

ready 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 7.2 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

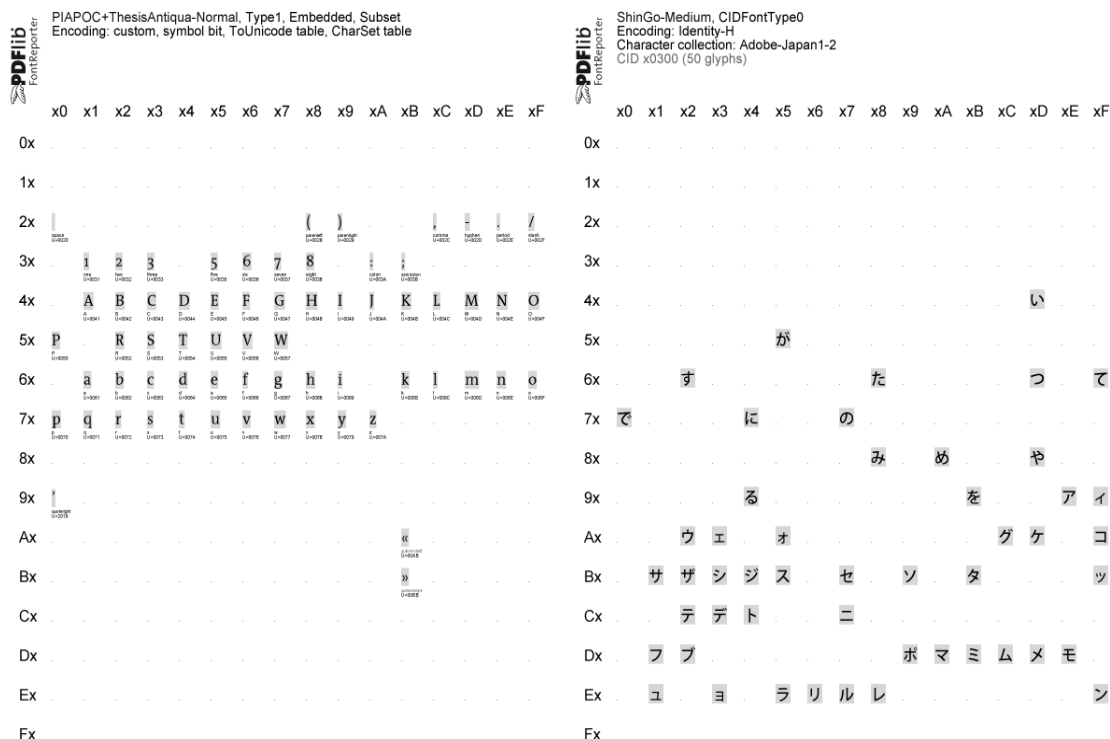


Fig. 7.2
Sample font reports created with the PDFlib FontReporter plugin for Adobe Acrobat

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. 7.3
The font report for a logotype font shows that the font contains wrong Unicode mappings. A custom code list can correct such mappings.

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 61 for details). In other words: if the resource name and the file name (without the `.cl` suffix) are identical you don't have to configure the resource since TET will implicitly do the equivalent of the following call (where *name* is an arbitrary resource name):

```
set_option("codelist {name name.cl}");
```

The following sample demonstrates the use of code lists. Consider the mismapped logotype glyphs in Figure 7.3 where a single glyph of the font actually represents multiple characters, and all characters together create the company logotype. However, the glyphs are wrongly mapped to the characters *a*, *b*, *c*, *d*, and *e*. In order to fix this you could create the following code list:

% Unicode mappings for codes in the GlobeLogosOne font

x61	x0054 x0068 x0065 x0020	% The
x62	x0042 x006F	% Bo
x63	x0073 x0074 x006F x006E x0020	% ston
x64	x0047 x006C x006F	% Glo
x65	x0062 x0065	% be

Then supply the codelist with the following option to `TET_open_document()` (assuming the code list is available in a file called *GlobeLogosOne.cl* and can be found via the search path):

```
glyphmapping {{fontname=GlobeLogosOne codelist=GlobeLogosOne}}
```

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 61). 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 0x) 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 61)

By convention, glyph lists use the file name suffix *.gl*. Glyph lists can be configured with the *glyphlist* resource. If no glyph list resource has been specified explicitly, TET will search for a file named *<myglyphlist>.gl* (where *<myglyphlist>* is the resource name) in the *searchpath* hierarchy (see Section 5.2, »Resource Configuration and File Searching«, page 61, for details). In other words: if the resource name and the file name (without the *.gl* suffix) are identical you don't have to configure the resource since TET will implicitly do the equivalent of the following call (where *name* is an arbitrary resource name):

```
set_option("glyphlist {name name.gl}");
```

Due to the precedence rules for glyph mapping, glyph lists 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 OS X and Windows systems, TET can access fonts which are installed on the host operating system. Access to these host fonts can be disabled with the *usehostfonts* option in `TET_open_document()`.

In order to configure a disk file for the *WarnockPro* font use the following call:

```
set_option("fontoutline {WarnockPro WarnockPro.otf}");
```

See Section 5.2, »Resource Configuration and File Searching«, page 61, for more details on configuring external font files.

8 Image Extraction

8.1 Image Extraction Basics

Image formats. TET extracts raster images from PDF pages and stores the extracted images in one of the following formats:

- ▶ TIFF (*.tif*) images are created in most cases. 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. We regard Adobe Photoshop as benchmark for the validity of TIFF images. Note that the Windows XP image viewer does not support the common Flate compression method in TIFF. In order to work around this viewer restriction you can enable LZW compression with the option `preferredtiffcompression=lzw` in `TET_write_image_file()` or `get_TET_image_data()`.
- ▶ JPEG (*.jpg*) is created for images which are 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*) is created for images which are compressed with the JPEG 2000 algorithm (*JPXDecode* filter) in PDF.
- ▶ JBIG2 (*.jbig2*) is created for images which are compressed with the JBIG2 algorithm (*JBIG2Decode* filter) in PDF. JBIG2 files are created with »sequential organization« according to ISO 14492.

Extracting images to disk or memory. The TET API can deliver the images extracted from PDF documents in two different ways:

- ▶ The `TET_write_image_file()` API function creates an image file on disk. The base file name of this image file must be specified in the `filename` option. TET will automatically add a suitable suffix depending on the image type.
- ▶ The `TET_get_image_data()` API function delivers the image data in memory. This is convenient if you want to pass on the image data to another processing component without having to deal with disk files.

Details depend on your image extraction requirements (see Section 8.4, »Page-based and Resource-based Image Loops«, page 118). In both cases you can determine the type of the extracted image (see next section).

Determine the file type of extracted images. The image file type is reported in the `Image/@extractedAs` attribute in TETML. At the API level you can use the following idiom to determine the type of an extracted image.

```
int imageType = tet.write_image_file(doc, tet.imageid, "typeonly");
```

```
/* Map the numerical image type to a format */
String imageFormat;
switch (imageType) {
case 10:
    imageFormat = "TIFF";
    break;
```

```
case 20:
```

```

        imageFormat = "JPEG";
        break;

case 30:
    imageFormat = "JPEG2000";
    break;

case 40:
    imageFormat = "RAW";
    break;

case 50:
    imageFormat = "JBIG2";
    break;

default:
    System.err.println("write_image_file() returned unknown value "
        + imageType + ", skipping image, error: "
        + tet.get_errmsg());
}

```

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/knowledge-base/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.

ICC profiles. An image in PDF may have an associated ICC profile which allows precise color reproduction. By default, TET processes attached ICC profiles and embeds them in the generated TIFF or JPEG image files. You can disable ICC profile embedding with the option *keepiccprofile=false* in *TET_write_image_file()* and *TET_get_image_data()*. This will reduce the size of the image files at the expense of color fidelity. Disabling ICC profile embedding is therefore not recommended for workflows which require precise color representation.

8.2 Image Merging and Filtering

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 and JBIG2 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 disable image merging use the following page option:

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



*Fig. 8.1
Although this
image consists of
many little strips,
TET extracts it as
a single reusable
image.*

When are images merged? Analyzing and merging images on a page are triggered by the corresponding call to `TET_open_page()`. This leads to the following important consequences:

- ▶ The number of entries in the pCOS `images[]` array, i.e. the value of the `length:images` pseudo object, may increase: as more pages are processed, artificial images which result from image merging are added to the array. In order to extract all merged images you must therefore open all pages in the document before querying `length:images` and extracting image data. Artificial (merged) images are marked with the corresponding flag `artificial` (numerical value 1) in the `images[]/mergetype` pseudo object.
- ▶ On the other hand, elements in the `images[]` array may only be used as parts of merged images. However, entries are never removed from the `images[]` array, but the consumed entries are marked with the corresponding flag `consumed` (numerical value 2) in the `images[]/mergetype` pseudo object.

How many images are in a document? Surprisingly, there is no simple answer to this simple question. The answer depends on the following decisions:

- ▶ Do you want to count image resources or placed images?
- ▶ Do you want to take images into account which are only used as parts of merged images, but are never placed isolated?

Using TET and pCOS pseudo objects you can determine all variants of the image count answer. The `image_count` topic in the TET Cookbook demonstrates various possibilities of image counting. It generates output like the following:

```
No of raw image resources before merging: 82
No of placed images: 12
No of images after merging (all types): 83
  normal images: 1
  artificial (merged) images: 1
  consumed images: 81
No of relevant (normal or artificial) image resources: 2
```

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` suboption of the `imageanalysis` page option. In order to completely disable small image removal use the following page option:

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

8.3 Placed Images and Image Resources

TET distinguishes between placed images and image resources.

- ▶ A *placed image* corresponds to an image on a page. A placed image has geometric properties: it is placed at a certain location and has a size (measured in points, millimeters, or some other absolute unit). In most cases the image is visible on the page, but in some cases it may be invisible because it is obscured by other objects on the page, is placed outside the visible page area, is fully or partially clipped, etc. Placed images are represented by the *PlacedImage* element in TETML.
- ▶ An *image resource* is a resource which represents the actual pixel data, colorspace and number of components, number of bits per component, etc. Unlike placed images, image resources don't have any intrinsic geometry. However, they do have width and height properties (measured in pixels). Each image resource has a unique ID which can be used to extract its pixel data. Image resources are represented by the *Image* element in TETML.

An image resource may be used as the basis for an arbitrary number of placed images in the document. Commonly each image resource will be placed exactly once, but it could also be placed repeatedly on the same page or on multiple pages. For example, consider an image for a company logo which is used repeatedly on the header of each page in the document. Each logo on a page constitutes a placed image, but all those placed images may be associated with the same image resource in an optimized PDF. On the other hand, in a non-optimized PDF each placed logo could be based on its own copy of the same image resource. This would result in the same visual appearance, but a larger PDF document. Non-optimized PDF documents may even contain image resources which are not even referenced on any page (i.e. unused resources).

Table 8.1 compares various properties of placed images and image resources.

Table 8.1 Comparison of placed images and image resources

property	placed images	image resources
TETML element	PlacedImage	Image
affected by image merging	yes	yes
associated with a page	yes	–
width and height in pixels	yes	yes
width and height in points	yes	–
position on the page	yes	–
number of appearances	1	0, 1, or more
unique ID	no: the <code>imageid</code> member returned by <code>TET_get_image_info()</code> and the <code>PlacedImage/@image</code> attribute in TETML identify only the underlying image resource	yes: <code>imageid</code> member returned by <code>TET_get_image_info()</code> <code>Image/@id</code> attribute in TETML
file naming convention in the TET command-line tool	<code><filename>_p<pagenumber>_<imagenumber>.[tif jpg jpx jbig2]</code>	<code><filename>_I<imageid>.[tif jpg jpx jbig2]</code>

8.4 Page-based and Resource-based Image Loops

The distinction between placed images and image resources gives rise to two fundamentally different approaches to image extraction: page-based and resource-based image extraction loops. Both methods can be used to extract images to a disk file or to memory.

Page-based image extraction loop. In this case the application is interested in the exact page layout and placed images, but doesn't care about duplicated image data. Extracting images with a page-based loop creates an image file for each placed image, and may result in the same image data for more than one extracted placed image. The application can avoid image duplication by checking for duplicate image IDs. However, unique image resource can more easily be extracted with the resource-based image extraction loop (see below).

The page-based image extraction loop can be activated in the TET command-line tool with the option `--imageloop page`. Code for page-based image extraction at the API level is demonstrated in the *images_per_page* and *images_in_memory* topics in the TET Cookbook. The *images_per_page* Cookbook topic and the *image_extractor* mini sample in the TET packages also show how to retrieve the image geometry.

Details of the page-based image extraction loop (please refer to the sample code mentioned above): `TET_get_image_info()` retrieves geometric information about a placed image as well as the pCOS image ID (in the *imageid* field) of the underlying image data. This ID can be used to retrieve more image details with `TET_pcos_get_number()`, such as the color space, width and height in pixels, etc., as well as the actual pixel data with `TET_write_image_file()` or `TET_get_image_data()`. `TET_get_image_info()` does not touch the actual pixel data of the image. If the same image is referenced multiply on one or more pages, the corresponding IDs will be the same.

Resource-based image extraction loop. In this case the application is interested in the image resources of the document, but doesn't care which image is used on which page. Image resources which are placed more than once (on one or more pages) are extracted only once. On the other hand, image resources which are not placed at all on any page will also be extracted.

The resource-based image extraction loop can be activated in the TET command-line tool with the option `--imageloop resource`. Code for resource-based image extraction at the API level is demonstrated in the *image_resources* mini sample and Cookbook topic. The pCOS Path Reference contains more information regarding the pCOS interface.

Details of the resource-based image extraction loop (please refer to the sample code mentioned above): all pages are opened before extracting image resources to make sure that image merging is activated; if image merging is not relevant this step can be skipped. In order to extract an image, the corresponding image ID is required. The code enumerates all values from 0 to the highest image ID, which is queried with `TET_pcos_get_number()` as the value of the pCOS path *length:images*. In order to skip the consumed parts of merged images (e.g. the strips of a multi-strip image), the type of each image resource is examined with the *mergetype* pCOS pseudo object. This allows us to skip image parts which have been consumed by the image merging process (since we are only interested in the resulting merged image). Once an image ID has been determined, one of the functions `TET_write_image_file()` or `TET_get_image_data()` can be called to write the image data to a disk file or pass the pixel data in memory, respectively.

8.5 Geometry of Placed Images

Using `TET_get_image_info()` you can retrieve geometric information for a placed image. The following values are available for each image in the `image_info` structure (see Figure 8.2):

- ▶ The `x` and `y` fields are the coordinates of the image reference point. The reference point is usually the lower left corner of the image. However, coordinate system transformations on the page may result in a different reference point. For example, the image may be mirrored horizontally with the result that the reference point becomes the upper left corner of the image. The value of `y` is subject to the *topdown* page option.
- ▶ The `width` and `height` fields correspond to the physical dimensions of the placed image on the page. They are provided in points (i.e. 1/72 inch).
- ▶ The angle `alpha` describes the direction of the pixel rows. This angle 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 values of `alpha` and `beta` are subject to the *topdown* page option.
- ▶ The angle `beta` describes the direction of the pixel columns, relative to the perpendicular of `alpha`. This angle 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");
```

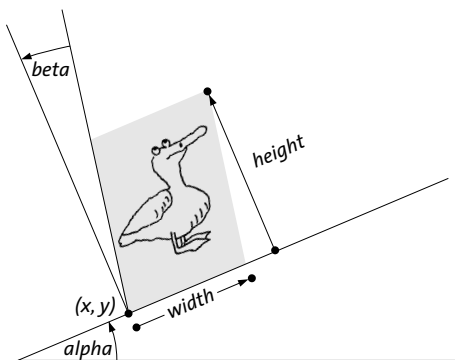


Fig. 8.2
Image geometry

```
double xDpi = 72 * width / tet.width;
double yDpi = 72 * height / tet.height;
...
}
```

Note that dpi values for rotated or skewed images may be meaningless. Full code for image dpi calculations can be found in the *determine_image_resolution* topic in the TET Cookbook.

8.6 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() returns -1 in these cases:

- ▶ PDF inline images: this is a rare flavor of PDF images which is sometimes used for small raster images.
- ▶ Images with Indexed Lab colorspace.

9 TET Markup Language (TETML)

9.1 Creating TETML

As an alternative to supplying the contents of a PDF document via a programming interface, TET can create XML output which represents the same information. We refer to the XML output created by TET as TET Markup Language (TETML). TETML contains the text contents of the PDF pages plus optional information such as text position, font, font size, etc. If TET detects table-like structures on the page the tables 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 17, 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 (some of these items are optional):

- ▶ general document information; encryption status, PDF standards, Tagged PDF etc.
- ▶ document info fields and XMP metadata
- ▶ text contents of each page (words or paragraphs; optionally lines)
- ▶ font and geometry for the glyph (font name, size, coordinates)
- ▶ layout attributes for the glyph (sub/superscript, dropcap, shadow)
- ▶ hyphenation attributes
- ▶ 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 9.2, »Controlling TETML Details«, page 127, 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="4.2">
<Creation platform="Linux-x86_64" tetVersion="4.2" date="2013-01-27T11:16:43+01:00" />
<Document filename="FontReporter.pdf" pageCount="9" fileSize="132437" linearized="true"
  pdfVersion="1.6">
<DocInfo>
<Author>PDFlib GmbH</Author>
<CreationDate>2010-07-06T22:51:50+00:00</CreationDate>
<Creator>FrameMaker 7.0</Creator>
<ModDate>2010-07-06T23:07:59+02:00</ModDate>
<Producer>Acrobat Distiller 9.3.3 (Windows)</Producer>
<Subject>PDFlib FontReporter</Subject>
<Title>PDFlib FontReporter Manual</Title>
</DocInfo>
<Metadata>
<x:xmpmeta xmlns:x="adobe:ns:meta/" x:xmptk="Adobe XMP Core 4.2.1-c043 52.372728, 2009/
01/18-15:08:04" >
  <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
    ...XMP metadata...
  </rdf:RDF>
</x:xmpmeta>
</Metadata>
<Options>tetml={} </Options>
<Pages>
<Page number="1" width="485" height="714">
<Options>tetml={} granularity=word </Options>
<Content granularity="word" dehyphenation="false" dropcap="false" font="false"
geometry="false" shadow="false" sub="false" sup="false">
<Para>
  <Word>
    <Text>FontReporter</Text>
    <Box llx="28.32" lly="613.53" urx="214.98" ury="643.53"/>
  </Word>
</Para>
<Para>
  <Word>
    <Text>Version</Text>
    <Box llx="28.32" lly="582.87" urx="100.24" ury="604.83"/>
  </Word>
  <Word>
    <Text>1.4</Text>
    <Box llx="105.05" lly="582.87" urx="128.79" ury="604.83"/>
  </Word>
</Para>
  ...more page content...
</Content>
</Page>
```

```

...more pages...
<Resources>
<Fonts>
  <Font id="F0" name="PDFlibLogo-Regular" fullname="MMOHKN+PDFlibLogo-Regular"
    type="TrueType" embedded="true" ascender="1000" capheight="700" italicangle="0"
    descender="0" weight="400" xheight="500"/>
  <Font id="F1" name="ThesisAntiqua-Bold" fullname="MMOHKO+ThesisAntiqua-Bold"
    type="Type 1 CFF" embedded="true" ascender="741" capheight="679" italicangle="0"
    descender="-250" weight="606" xheight="505"/>
...more fonts...
</Fonts>
<Images>
  <Image id="I0" extractedAs=".tif" width="595" height="750" colorspace="CS3"
    bitsPerComponent="8"/>
  <Image id="I1" extractedAs=".tif" width="595" height="750" colorspace="CS3"
    bitsPerComponent="8"/>
</Images>
<ColorSpaces>
  <ColorSpace id="CS0" name="DeviceGray" components="1"/>
  <ColorSpace id="CS1" name="DeviceCMYK" components="4"/>
...more color spaces...
</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 127; here is a variation of the sample above with more glyph details. The *Glyph* element contains font and position information:

```

<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>
  </Box>
</Word>

```

```
<Glyph font="F1" size="18" x="254.92" y="636.33" width="7.52">c</Glyph>
<Glyph font="F1" size="18" x="262.44" y="636.33" width="10.22">h</Glyph>
<Glyph font="F1" size="18" x="272.66" y="636.33" width="9.34">e</Glyph>
<Glyph font="F1" size="18" x="282.00" y="636.33" width="10.22">n</Glyph>
</Box>
</Word>
```

9.2 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 for Western text«, page 85). Lines of text can optionally be identified with the *Line* element; this is controlled via the *tetml* page option.
- ▶ *Wordplus* mode is similar to *word* mode, but adds font and coordinate details for all glyphs in a word. The coordinates are expressed relative to the lower left or upper left corner subject to the *topdown* page option. *Wordplus* mode makes it possible to analyze font usage and track changes of font, font size, etc. within a word. Since *wordplus* is the only 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 9.1 lists the TETML elements which are present in the text modes.

Table 9.1 TETML elements in various text mode

text mode	structure	tables	text position	text details
glyph	–	–	–	Glyph
word	Para, Word optionally: Line	Table, Row, Cell	Box	–
wordplus	Para, Word optionally: Line	Table, Row, Cell	Box	Glyph
line	Para, Line	–	–	–
page	Para	Table, Row, Cell	–	–

Selecting the text mode. With the TET command-line tool (see Section 2.1, »Command-Line Options«, page 17) 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. With the *glyphdetails* suboption of the *tetml* option 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 tetml={ glyphdetails={all} }
```

Table 9.2 summarizes the options for creating page modes.

Table 9.2 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={all}}
<i>word</i>	granularity=word	–
<i>wordplus</i>	granularity=word	tetml={glyphdetails={all}}
<i>word with Line elements</i>	granularity=word	tetml={elements={line}}
<i>wordplus with Line elements</i>	granularity=word	tetml={glyphdetails={all} elements={line}}
<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 10.8.

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 *skipengines* option can be used to disable the text and image extraction engines. The following option list instructs TET to process text contents, but disable image processing:

```
skipengines={image}
```


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 10.10. 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 other glyph attributes:

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

The following page option list adds *Line* elements to the TETML output:

```
tetml={ glyphdetails={font} elements={line} }
```

The following page option adds *sub* and *sup* attributes to the *Glyph* element to designate subscripts and superscripts:

```
tetml={ glyphdetails={sub sup} }
```

The following page option uses *all* to generate all possible attributes to the *Glyph* element:

```
tetml={ glyphdetails={all} }
```

The following page option requests topdown coordinates instead of the default bottom-up coordinates:

```
topdown={output}
```

The following page option list instructs TET to combine punctuation characters with the adjacent words, i.e. punctuation characters are no longer treated as individual words:

```
contentanalysis={nopunctuationbreaks}
```

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.

9.3 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 9.3 describes the role of all TETML elements. Attributes which have been introduced with TET 4.1 and TET 4.0 are marked. Figure 9.1 visualizes the XML hierarchy of TETML elements.

Table 9.3 TETML elements and attributes

TETML element	description and attributes
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. Attributes: name, level, pageNumber
Attachments	Container of Attachment elements
Box	Describes the coordinates of a Word. The attributes llx and lly describe the lower left corner, urx and ury describe the upper right corner of the Box in standard PDF coordinates. If the Box represents a rectangle with edges parallel to the page edges, the four values llx, lly, urx, ury describe the lower left and upper right corners; otherwise the coordinates of all four corners are present. 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. Attributes: llx, lly ¹ , urx, ury ¹ , ulx, uly ¹ , lrx, lry ¹
Cell	Describes the contents of a single table cell. Attribute: colSpan
ColorSpace	Describes a PDF colorspace. Attributes: alternate, base, components, id, name
ColorSpaces	Container of ColorSpace elements
Content	Describes the page contents as a hierarchical structure. Attributes: granularity, dehyphenation (TET 4.0), dropcap (TET 4.0), font, geometry, shadow (TET 4.0), sub (TET 4.0), sup (TET 4.0)
Creation	Describes the date and operating system platform for the TET execution, plus the version number of TET. Attributes: platform, tetVersion, date
DocInfo	Predefined and custom document info entries
Document	Describes general document information including PDF file name and size, PDF version number. Attributes: filename, pageCount, filesize, linearized, pdfVersion, pdfa (TET 4.0: new values for PDF/A-2; TET 4.1: new values for PDF/A-3), pdfe (TET 4.0; TET 4.1: new values for PDF/E-2), pdfua (TET 4.1), pdfvt (TET 4.1), pdfx (TET 4.1: enumerated values), tagged

Table 9.3 TETML elements and attributes

TETML element	description and attributes
Encryption	Describes various security settings. Attributes: keylength, algorithm (TET 4.1: new values 8-11), attachment (TET 4.1), description (TET 4.1: new values for algorithms 8-11), masterpassword, userpassword, noprint, nomodify, nocopy, noannots, noassemble, noforms, noaccessible, nohiresprint, plainmetadata
Exception	Error message and number associated with an exception which was thrown by TET. The Exception element may replace other elements if not enough information can be extracted from the input because of malformed PDF data structures. Attribute: errnum
Font	Describes a font resource. The required name attribute contains the canonical font name, while the optional fullname attribute contains the font name including subset prefix. Attributes: ascender (TET 4.1), capheight (TET 4.1), descender (TET 4.1), embedded, fullname (TET 4.0), id, italicangle (TET 4.1), type, name, vertical, weight (TET 4.1), xheight (TET 4.1)
Fonts	Container of Font elements
Glyph	Describes font and geometry details for a single glyph. The element content holds the Unicode character(s) produced by this glyph. A single glyph may produce more than one character, e.g. for ligatures. The Glyph elements for a word are grouped within one or more Box elements. Attributes: x, y ¹ , width, alpha ¹ , beta ¹ , shadow (TET 4.0), dropcap (TET 4.0), font, size, sub (TET 4.0), sup (TET 4.0), textrendering, unknown, dehyphenation (TET 4.0)
Image	Describes an image resource, i.e. the actual pixel array comprising the image. Attributes: bitsPerComponent, colorspace, extractedAs (TET 4.0, additional value introduced with TET 4.2), height, id, mask, maskonly, mergetype, width
Images	Container of Image elements
Line	Text for a single line. TET 4.0: Line may also contain Word elements.
Metadata	XMP metadata which can be associated with the document, a font, or an image
Options	Document or page options used for generating the TETML
Page	Contents of a single page. Attributes: number, height, width, topdown (TET 4.0)
Pages	Container of Page elements
Para	Text comprising a single paragraph
PlacedImage	Describes an instance of an image placed on the page. Attributes: alpha ¹ , beta ¹ , height, image, width, x, y ¹
Resources	Colorspace, font, and image resources
Row	One or more table cells
Table	One or more table rows
TET	Root element Attribute: version (TET 4.2 creates 4.2; TET 4.1 creates 4.1; TET 4.0 creates 4.0, TET 3 creates 3)
Text	Text contents of a word or other element
Word	Single word

1. All vertical coordinates and angles are expressed relative to the lower left or upper left corner subject to the topdown page option.

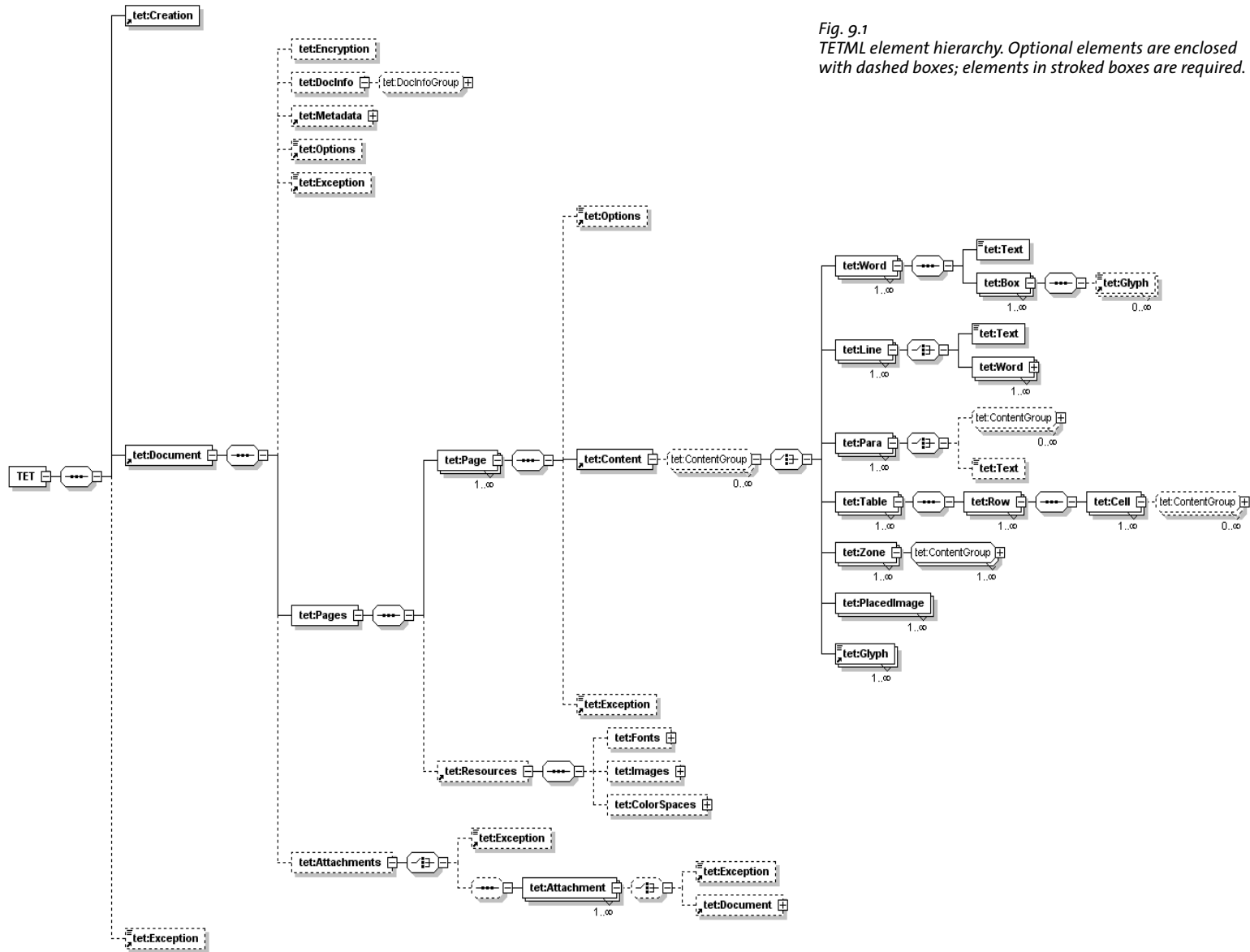


Fig. 9.1
TETML element hierarchy. Optional elements are enclosed with dashed boxes; elements in stroked boxes are required.

9.4 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 9.1, »Creating TETML«, page 123.

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 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
- ▶ 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 to 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/en-us/download/details.aspx?displaylang=en&id=21714

Run the program as follows:

```
msxsl.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.XsltTransform* 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 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"
...
```


9.5 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 9.4, »Transforming TETML with XSLT«, page 134 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 127, 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 TETML to HTML which can be displayed in a browser. The converter does not attempt to generate an identical visual representation of the PDF document, but demonstrates the following aspects:

- ▶ Create heading elements (*H1*, *H2*, etc.) based on configurable font sizes.
- ▶ Map table elements in TETML to the corresponding HTML table constructs to visualize tables in the browser.
- ▶ Create 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.
- ▶ Create a list of images for each page where the images are linked to the corresponding image file, using the image file names created by the TET command-line tool in the *resource* imageloop mode (e.g. *tet --image --tetml file.pdf*).

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.

10 TET Library API Reference

10.1 Option Lists

Option lists are a powerful yet easy method for controlling API function calls. Instead of requiring a multitude of function parameters, many API methods support option lists, or *optlists* for short. These are strings which can contain an arbitrary number of options. Option lists support various data types and composite data like lists. In most language bindings optlists can easily be constructed by concatenating the required keywords and values.

- Bindings* C language binding: you may want to use the *sprintf()* function for constructing optlists.
- Bindings* .NET language binding: C# programmers should keep in mind that the *AppendFormat()* *StringBuilder* method uses the { and } braces to represent format items which will be replaced by the string representation of arguments. On the other hand, the *Append()* method does not impose any special meaning on the brace characters. Since the option list syntax makes use of the brace characters, care must be taken in selecting the *AppendFormat()* or *Append()* method appropriately.

10.2 Option List Syntax

- Formal option list syntax definition.** Option lists must be constructed according to following rules:
- ▶ All elements (keys and values) in an option list must be separated by one or more of the following separator characters: space, tab, carriage return, newline, equal sign '='.
 - ▶ An outermost pair of enclosing braces is not part of the element. The sequence {} designates an empty element.
 - ▶ Separators within the outermost pair of braces no longer split elements, but are part of the element. Therefore, an element which contains separators must be enclosed with braces.
 - ▶ An element which contains braces at the beginning or end must be enclosed with braces.
 - ▶ If an element contains unbalanced braces, these must be protected with a preceding backslash character. A backslash in front of the closing brace of an element must also be preceded by a backslash character.
 - ▶ Option lists must not contain binary zero values.

An option may have a list value according to its documentation in this PDFlib Reference. List values contain one or more elements (which may themselves be lists). They are separated according to the rules above, with the only difference that the equal sign is no longer treated as a separator.

Simple option lists. In many cases option lists will contain one or more key/value pairs. Keys and values, as well as multiple key/value pairs must be separated by one or

more whitespace characters (space, tab, carriage return, newline). Alternatively, keys can be separated from values by an equal sign '=':

```
key=value
key = value
key value
key1 = value1 key2 = value2
```

To increase readability we recommend to use equal signs between key and value and whitespace between adjacent key/value pairs.

Since option lists 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. In the following example the first option assignment will be overridden by the second, and *key* will have the value *value2* after processing the option list:

```
key=value1 key=value2
```

List values. Lists contain one or more separated values, which may be simple values or list values in turn. Lists are bracketed with { and } braces, and the values in the list must be separated by whitespace characters. Examples:

```
searchpath={/usr/lib/tet d:\tet} (list containing two directory names)
```

A list may also contain nested lists. In this case the lists must also be separated from each other by whitespace. While a separator must be inserted between adjacent } and { characters, it can be omitted between braces of the same kind:

```
fold={ {[Private_Use:] remove} {[U+FFFD] remove} } (list containing two lists)
```

If the list contains exactly one list the braces for the nested list must not be omitted:

```
fold={ {[Private_Use:] remove} } (list containing one nested list)
```

Nested option lists and list values. Some options accept the type *option list* or *list of option lists*. Options of type *option list* contain one or more subordinate options. Options of type *list of option lists* contain one or more nested option lists. When dealing with nested option lists it is important to specify the proper number of enclosing braces. Several examples are listed below.

The value of the option *contentanalysis* is an option list which itself contains a single option *punctuationbreaks*:

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

The value of the option *glyphmapping* in the following example is a list of option lists containing a single option list:

```
glyphmapping={ {fontname=GlobeLogosOne codelist=GlobeLogosOne} }
```

The value of the option *glyphmapping* in the following example is a list of option lists containing two option lists:

```
glyphmapping { {fontname=CMSY* glyphlist=tarski} {fontname=ZEH* glyphlist=zeh}}
```

List containing one option list with a *fontname* value that includes spaces and therefore requires an additional pair of braces:

```
glyphmapping={ {fontname={Globe Logos One} codelist=GlobeLogosOne} }
```

List containing two keywords:

```
fonttype={Type1 TrueType}
```

List containing different types – the inner lists contain a Unicode set and a keyword, the outer list contains two option lists and the keyword *default*:

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

List containing one rectangle:

```
includeboxes={{10 20 30 40}}
```

Common traps and pitfalls. This paragraph lists some common errors regarding option list syntax.

Braces are not separators; the following is wrong:

```
key1 {value1}key2 {value2}                WRONG!
```

This will trigger the error message *Unknown option 'value2'*. Similarly, the following are wrong since the separators are missing:

```
key{value}                                WRONG!  
key={{value1}{value2}}                   WRONG!
```

Braces must be balanced; the following is wrong:

```
key={open brace {}                        WRONG!
```

This will trigger the error message *Braces aren't balanced in option list 'key={open brace {}'*. A single brace as part of a string must be preceded by an additional backslash character:

```
key={closing brace \} and open brace \{}  CORRECT!
```

A backslash at the end of a string value must be preceded by another backslash if it is followed by a closing brace character:

```
key={\value\}                             WRONG!  
key={\value\\}                             CORRECT!
```

10.3 Basic Types

String. Strings are plain ASCII strings (or EBCDIC strings on EBCDIC platforms) which are generally used for non-localizable keywords. Strings containing whitespace or '=' characters must be bracketed with { and }:

```
password={ secret string }           (string value contains three blanks)
contents={length=3mm}                (string value containing one equal sign)
```

The characters { and } must be preceded by an additional \ character if they are supposed to be part of the string:

```
password={weird\}string}             (string value contains a right brace)
```

A backslash in front of the closing brace of an element must be preceded by a backslash character:

```
filename={C:\path\name\\}            (string ends with a single backslash)
```

An empty string can be constructed with a pair of braces:

```
{}
```

Content strings, hypertext strings and name strings: these can hold Unicode content in various formats. Single bytes can be expressed by an escape sequence if the parameter *escapesequence* is set. For details on these string types and encoding choices for string options see the *PDFlib Tutorial*.

Non-Unicode capable language bindings: if an option list starts with a [EBCDIC-]UTF-8 BOM, each content, hypertext or name string of the option list will be interpreted as a [EBCDIC-]UTF-8 string.

Unichar. A Unichar is a single Unicode value where several syntax variants are supported: decimal values ≥ 10 (e.g. 173), hexadecimal values prefixed with x, X, 0x, 0X, or U+ (xAD, 0xAD, U+00AD), numerical references, character references, and glyph name references but without the '&' and ';' decoration (*shy*, #xAD, #173). Example:

```
unknownchar=?                        (literal)
unknownchar=63                       (decimal)
unknownchar=x3F                      (hexadecimal)
unknownchar=0x3F                     (hexadecimal)
unknownchar=U+003F                   (Unicode notation)
lineseparator={CRLF}                 (standard glyph name reference)
```

Single characters which happen to be a number are treated literally, not as decimal Unicode values:

```
replacementchar=3                    (U+0033 THREE, not U+0003!)
```

Unichars must be in the hexadecimal range 0-0xFFFF (decimal 0-1114111).

Unicode sets. Unicode sets can be constructed with the following building blocks:

- ▶ Patterns are a series of characters bounded by square brackets that contain lists of Unicode characters and Unicode property sets.
- ▶ Lists are a sequence of Unicode characters that may have ranges indicated by a '-' between two characters, as in `U+FB00-U+FB17`. The sequence specifies the range of all characters from the left to the right, in Unicode order. Multiple Unicode characters must not be separated by whitespace, but must directly follow each other, e.g. `U+0048U+006C`.
- ▶ Unicode characters in lists can be specified as follows:
 - ASCII characters can be specified as literals
 - Exactly 4 hex digits: `\uhhhh` or `U+hhhh`
 - Exactly 5 hex digits: `U+hhhhh`
 - 1-6 hex digits: `\x{hhhhhh}`
 - Exactly 8 hex digits: `\Uhhhhhhhh`
 - escaped backslash: `\\`
- ▶ Unicode property sets are specified by a Unicode property. The syntax for specifying the property names is an extension of POSIX and Perl syntax, where *type* represents the name of a Unicode property (see www.unicode.org/Public/UNIDATA/PropertyAliases.txt) and *value* the corresponding value (see www.unicode.org/Public/UNIDATA/PropertyValueAliases.txt):
 - POSIX-style syntax: `[type=value:]`
 - POSIX-style syntax with negation: `[^type=value:]`
 - Perl-style syntax: `[p{type=value}]`
 - Perl-style syntax with negation: `[P{type=value}]`
 - The *type=* can be omitted for the Category and Script properties, but is required for other properties.
- ▶ Set operations can be applied to patterns:
 - To build the union of two sets, simply concatenate them: `[[:letter:]][:number:]`
 - To intersect two sets, use the '&' operator: `[[:letter:]] & [U+0061-U+007A]`
 - To take the set difference of two sets, use the '-' operator: `[[:letter:]]-[U+0061-U+007A]`
 - To invert a set, place a '^' immediately after the opening '[': `[^U+0061-U+007A]`. In any other location, the '^' does not have a special meaning.

See Table 10.1 for examples of Unicode sets. You can use the following Web site for interactively testing Unicode set expressions:

unicode.org/cldr/utility/list-unicodeset.jsp

Boolean. Booleans have the values *true* or *false*; if the value of a Boolean option is omitted, the value *true* is assumed. As a shorthand notation *noname* can be used instead of *name=false*:

<code>usehostfonts</code>	(equivalent to <code>usehostfonts=true</code>)
<code>nousehostfonts</code>	(equivalent to <code>usehostfonts=false</code>)

Keyword. An option of type keyword can hold one of a predefined list of fixed keywords. Example:

`clippingarea=cropbox`

For some options the value holds either a number or a keyword.

Table 10.1 Unicode set examples

<i>specification of Unicode set</i>	<i>characters in the Unicode set</i>
[U+0061-U+007A]	lower case letters a through z
[U+0640]	single character Arabic Tatweel
[\x{0640}]	single character Arabic Tatweel
[U+FB00-U+FB17]	Latin and Armenian ligatures
[^U+0061-U+007A]	all characters except a through z
[:Lu:] [:UppercaseLetter:]	all uppercase letters (short and long forms of the Unicode set)
[:L:] [:Letter:]	all Unicode categories starting with L (short and long forms of the Unicode set)
[:General_Category=Dash_Punctuation:]	all characters in the general category Dash_Punctuation
[:Alphabetic=No:]	all non-alphabetic characters
[:Private_Use:]	all characters in the Private Use Area (PUA)

Number. Option list support several numerical types.

Integer types can hold decimal and hexadecimal integers. Positive integers starting with x, X, ox, or oX specify hexadecimal values:

-12345
0
0xFF

Floats can hold decimal floating point or integer numbers; period and comma can be used as decimal separators for floating point values. Exponential notation is also supported. The following values are all equivalent:

size = -123.45
size = -123,45
size = -1.2345E2
size = -1.2345e+2

10.4 Geometric Types

Rectangle. A rectangle is a list of four float values specifying the *x* and *y* coordinates of the lower left and upper right corners of a rectangle. The coordinate system for interpreting the coordinates (default or user coordinate system) varies depending on the option, and is documented separately. Example:

```
includebox = {{0 0 500 100} {0 500 500 600}}^
```

10.5 Encoding Names

Various options and parameters support the names of encodings, e.g. the *filename-handling* option of *TET_set_option()*, the *forceencoding* option of *TET_open_document()*, and the *inputformat* parameter of *TET_convert_to_unicode()*. The following keywords can be supplied as encoding names:

- ▶ The keyword *auto* specifies the most natural encoding for certain environments:
 - ▶ On Windows: the current system code page
 - ▶ On Unix and OS X: *iso8859-1*
 - ▶ On i5/iSeries: the current job's encoding (*IBMCCSID000000000000*)
 - ▶ On zSeries: *ebcdic*
- ▶ *winansi* (=cp1252)
- ▶ *iso8859-1* - *iso8859-10*, *iso8859-13* - *iso8859-14*
- ▶ *cp1250* - *cp1258*
- ▶ *macroman*, *macroman_euro* (replaces currency with Euro), *macroman_apple*, (replaces currency with Euro and includes additional mathematical/greek symbols)
- ▶ *U+XXXX* (256 characters starting at the specified value)
- ▶ *ebcdic* (=code page 1047), *ebcdic_37* (=code page 037)
- ▶ CJK encodings *cp932*, *cp936*, *cp949*, *cp950*
- ▶ on the following systems all encodings available on the host system can be used:
 - ▶ *cpXXXX* on Windows
 - ▶ any *Coded Character Set Identifier* without the *CCSID* prefix on i5/iSeries
 - ▶ any *Coded Character Set Identifier* (*CCSID*) on zSeries
- ▶ custom encodings can be defined as resources and referenced by their resource name

10.6 General Functions

10.6.1 Option Handling

C++

void set_option(wstring optlist)

C# Java

void set_option(String optlist)

Perl PHP

set_option(string optlist)

VB RB

Sub set_option(optlist As String)

C

void TET_set_option(TET *tet, const char *optlist)

Set one or more global options for TET.

optlist An option list specifying global options according to Table 10.2. 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*, *filenamehandling*, *fontoutline*, *glyphlist*, *license*, *licensefile*, *logging*, *userlog*, *outputformat*, *resourcefile*, *searchpath*

Details Multiple calls to this function can be used to accumulate values for those options marked in Table 10.2. For unmarked options the new value will override the old one.

Table 10.2 Global options for *TET_set_option()*

option	description
<i>asciifile</i>	(Boolean; Only supported on i5/iSeries and zSeries). Expect text files (e.g. UPR configuration files, glyph lists, code lists) in ASCII encoding. Default: true on i5/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 61).
<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 61).
<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 61).

Table 10.2 Global options for `TET_set_option()`

option	description
filename-handling	(Keyword; not required on Windows) Target encoding for file names. On Windows this option will be applied to supplied file names, but not to the names of generated files (default: unicode on OS X, auto on <i>is/iSeries</i> , otherwise <i>honorlang</i>): ascii 7-bit ASCII baseibcdic Basic EBCDIC according to code page 1047, but only Unicode values $\leq U+007E$ baseibcdic_37 Basic EBCDIC according to code page 0037, but only Unicode values $\leq U+007E$ honorlang (Not supported on <i>is/iSeries</i>) The environment variables <code>LC_ALL</code> , <code>LC_CTYPE</code> and <code>LANG</code> will be interpreted and applied to file names if it specifies <code>utf8</code> , <code>UTF-8</code> , <code>cpXXXX</code> , <code>CPXXXX</code> , <code>iso8859-x</code> , or <code>ISO-8859-x</code> . legacy Use auto encoding (i.e. the current system encoding) to interpret the file name and interpret the <code>LANG</code> variable if the <code>honorlang</code> parameter is set. unicode Unicode encoding in (EBCDIC-) UTF-8 format all names of 8-bit and CJK encodings Encoding name according to Section 10.5, »Encoding Names«, page 147 File names supplied in non-Unicode aware language bindings without a UTF-8 BOM and with <code>length=0</code> are interpreted according to the <code>filenamehandling</code> option.
fontoutline ^{1, 2}	(List of name strings) A list of string pairs, where each pair contains the name and value of a <code>FontOutline</code> resource (see Section 5.2, »Resource Configuration and File Searching«, page 61).
glyphlist ^{1, 2}	(List of name strings) A list of string pairs, where each pair contains the name and value of a <code>glyphlist</code> resource (see Section 5.2, »Resource Configuration and File Searching«, page 61).
hostfont ^{1, 2}	(List of name strings) A list of string pairs, where each pair contains a PDF font name and the UTF-8 encoded name of a host font to be used for an unembedded font.
license	(String) Set the license key. It must be set before the first call to <code>TET_open_document*()</code> .
licensefile	(String) Set the name of a file containing the license key(s). The license file can be set only once before the first call to <code>TET_open_document*()</code> . Alternatively, the name of the license file can be supplied in an environment variable called <code>PDFLIBLICENSEFILE</code> or (on Windows) via the registry.
logging ¹	(Option list; unsupported) An option list specifying logging output according to Table 10.7. Alternatively, logging options can be supplied in an environment variable called <code>TETLOGGING</code> or on Windows via the registry. An empty option list will enable logging with the options set in previous calls. If the environment variable is set logging will start immediately after the first call to <code>TET_new()</code> .
userlog	(Name string; unsupported) Arbitrary string which will be written to the log file if logging is enabled.
output-format	(Keyword; only for the C, Ruby, Perl, Python, and PHP language bindings) Specifies the format of the text returned by <code>TET_get_text()</code> : utf8 Strings are returned in (in C: null-terminated) UTF-8 format . utf16 Strings are returned in UTF-16 format in the machine's native byte ordering. utf32 Strings are returned in UTF-32 format in the machine's native byte ordering. ebcdicutf8 (Only available on EBCDIC-based systems) Strings are returned in null-terminated EBCDIC-encoded UTF-8 format. Code page 37 is used on <i>is/iSeries</i> , code page 1047 on <i>zSeries</i> . Default: <code>utf8</code> for C, Ruby, Perl, Python, PHP, and <code>ebcdicutf8</code> for C on <i>is/iSeries</i> and <i>zSeries</i>
resourcefile	(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. 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 61). Default: <code>tet.upr</code> (on MVS: <code>upr</code>)

Table 10.2 Global options for `TET_set_option()`

option	description
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 61). An empty string deletes all existing search path entries. On Windows the search path can also be set via a registry entry. Default: platform-specific, see »File search and the searchpath resource category«, page 62.
shutdown-strategy	(Integer) Strategy for releasing global resources which are allocated once for all TET objects. Each global resource is initialized on demand when it is first needed. This option must be set to the same value for all TET objects in a process; otherwise the behavior is undefined (default: 0): <ul style="list-style-type: none">0 A reference counter keeps track of how many PLOP objects use the resource. When the last TET object is deleted and the reference counter drops to zero, the resource is released.1 The resource is kept until the end of the process. This may slightly improve performance, but requires more memory after the last TET object is deleted.

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.

10.6.2 Setup

C	<i>TET *TET_new(void)</i>
----------	----------------------------------

Create a new TET object.

Returns 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.

Bindings This function is not available in object-oriented language bindings since it is hidden in the TET constructor.

Java	<i>void delete()</i>
-------------	-----------------------------

C#	<i>void Dispose()</i>
-----------	------------------------------

C	<i>void TET_delete(TET *tet)</i>
----------	---

Delete a TET object and release all related internal resources.

Details Deleting a TET object automatically closes all of its open documents. The TET object must no longer be used in any function after it has been deleted.

Bindings 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 *Dispose()* should be called at the end of processing to clean up unmanaged resources.

10.6.3 PDFlib Virtual Filesystem (PVF)

C++	<code>void create_pvf(wstring filename, const void *data, size_t size, wstring optlist)</code>
C# Java	<code>void create_pvf(String filename, byte[] data, String optlist)</code>
Perl PHP	<code>create_pvf(string filename, string data, string optlist)</code>
VB RB	<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 10.3. 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 10.3 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++	<i>int delete_pvf(wstring filename)</i>
C# Java	<i>int delete_pvf(String filename)</i>
Perl PHP	<i>int delete_pvf(string filename)</i>
VB RB	<i>Function delete_pvf(filename As String) As Long</i>
C	<i>int TET_delete_pvf(TET *tet, const char *filename, int len)</i>

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.

C++	<i>int info_pvf(wstring filename, wstring keyword)</i>
C# Java	<i>int info_pvf(String filename, String keyword)</i>
Perl PHP	<i>int info_pvf(string filename, string keyword)</i>
VB RB	<i>Function info_pvf(filename As String, keyword As String) As Long</i>
C	<i>int TET_info_pvf(TET *tet, const char *filename, int len, const char *keyword)</i>

Query properties of a virtual file or the PDFlib Virtual File system (PVF).

filename (Name string) The name of the virtual file. The filename may be empty if *keyword=filecount*.

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

keyword A keyword according to Table 10.4.

Details This function returns various properties of a virtual file or the PDFlib Virtual File system (PVF). The property is specified by *keyword*.

Table 10.4 Keywords for **TET_info_pvf()**

option	description
filecount	Total number of files in the PDFlib Virtual File system maintained for the current TET object. The filename parameter will be ignored.
exists	1 if the file exists in the PDFlib Virtual File system (and has not been deleted), otherwise 0
size	(Only for existing virtual files) Size of the specified virtual file in bytes.

Table 10.4 Keywords for `TET_info_pvf()`

option	description
<i>iscopy</i>	(Only for existing virtual files) 1 if the copy option was supplied when the specified virtual file was created, otherwise 0
<i>lockcount</i>	(Only for existing virtual files) Number of locks for the specified virtual file set internally by TET functions. The file can only be deleted if the lock count is 0.

10.6.4 Unicode Conversion Function

C++	<code>string convert_to_unicode(wstring inputformat, string input, wstring optlist)</code>
C# Java	<code>String convert_to_unicode(String inputformat, byte[] input, String optlist)</code>
Perl PHP	<code>string convert_to_unicode(string inputformat, string input, string optlist)</code>
VB RB	<code>Function convert_to_unicode(inputformat As String, input As String, optlist As String) As String</code>
C	<code>const char *TET_convert_to_unicode(TET *tet, const char *inputformat, const char *input, int inputlen, int *outputlen, const char *optlist))</code>

Convert a string in an arbitrary encoding to a Unicode string in various formats.

inputformat Unicode text format or encoding name specifying interpretation of the input string:

- ▶ Unicode text formats: *utf8*, *ebcdicutf8*, *utf16*, *utf16le*, *utf16be*, *utf32*
- ▶ An encoding name according to Section 10.5, »Encoding Names«, page 147
- ▶ The keyword *auto* specifies the following behavior: if the input string contains a UTF-8 or UTF-16 BOM it will be used to determine the appropriate format, otherwise the current system codepage is assumed.

input String to be converted to Unicode.

inputlen (C language binding only) Length of the input string in bytes. If *inputlen=0* a null-terminated string must be provided.

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

optlist An option list specifying options according to Table 10.5:

- ▶ Input filter options: *charref*, *escapesequence*
- ▶ Unicode conversion options: *bom*, *errorpolicy*, *inflate*, *outputformat*

Returns A Unicode string created from the input string according to the specified parameters and options. If the input string does not conform to the specified input format (e.g. invalid UTF-8 string) an empty output string will be returned if *errorpolicy=return*, and an exception will be thrown if *errorpolicy=exception*.

Details This function may be useful for general Unicode string conversion. It is provided for the benefit of users working in environments which do not provide suitable Unicode converters.

Bindings C binding: the returned strings will be stored in a ring buffer with up to 10 entries. If more than 10 strings are converted, the 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++ binding: The parameters *inputformat* and *optlist* must be passed as *wstrings* as usual, while *input* and returned data must have type *string*.

Python binding: UTF-8 results will be returned as a string, Python 3: non-UTF-8 results will be returned as bytes.

Table 10.5 Options for `TET_convert_to_unicode()`

option	description
charref	(Boolean) If true, enable substitution of numeric and character entity references and glyph name references. Default: false
bom	(Keyword; ignored for <code>outputformat=utf32</code> ; for Unicode-aware language bindings only none is allowed) Policy for adding a byte order mark (BOM) to the output string. Supported keywords (default: none): add Add a BOM. keep Add a BOM if the input string has a BOM. none Don't add a BOM. optimize Add a BOM except if <code>outputformat=utf8</code> or <code>ebcdicutf8</code> and the output string contains only characters in the range <code>< U+007F</code> .
errorpolicy	(Keyword) Behavior in case of conversion errors (default: exception): return The replacement character U+FFFD will be used if a character reference cannot be resolved or a builtin code or glyph ID doesn't exist in the specified font. An empty string will be returned in case of conversion errors. exception An exception will be thrown in case of conversion errors.
escape-sequence	(Boolean) If true, enable substitution of escape sequences in strings. Default: false
inflate	(Boolean; only for <code>inputformat=utf8</code> ; will be ignored if <code>outputformat=utf8</code>) If true, an invalid UTF-8 input string will not trigger an exception, but rather an inflated byte string in the specified output format will be generated. This may be useful for debugging. Default: false
output-format	(Keyword) Unicode text format of the generated string: <code>utf8</code> , <code>ebcdicutf8</code> , <code>utf16</code> , <code>utf16le</code> , <code>utf16be</code> , <code>utf32</code> . An empty string is equivalent to <code>utf16</code> . Default: <code>utf16</code> Unicode-aware language bindings: the output format will be forced to <code>utf16</code> . C++ language binding: only the following output formats are allowed: <code>ebcdicutf8</code> , <code>utf8</code> , <code>utf16</code> , <code>utf32</code> .

10.6.5 Exception Handling

C++
C# Java
Perl PHP
VB RB
C

wstring get_apiname()
String get_apiname()
string get_apiname()
Function get_apiname() As String
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++
C# Java
Perl PHP
VB RB
C

wstring get_errmsg()
String get_errmsg()
string get_errmsg()
Function get_errmsg() As String
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++
C# Java
Perl PHP
VB RB
C

int get_errnum()
int get_errnum()
long get_errnum()
Function get_errnum() As Long
int TET_get_errnum(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
C
C
C

TET_TRY(tet)
TET_CATCH(tet)
TET_RETHROW(tet)
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 24.

10.6.6 Logging

The logging feature can be used to trace API calls. The contents of the log file may be useful for debugging purposes, or may be requested by PDFlib GmbH support. Table 10.6 lists the options for activating the logging feature with `TET_set_option()` (see Section 10.6.1, »Option Handling«, page 148).

Table 10.6 Logging-related keys for `TET_set_option()`

key	explanation
logging	Option list with logging options according to Table 10.7
userlog	String which will be copied to the log file

The logging options can be supplied in the following ways:

- ▶ As an option list for the `logging` option of `TET_set_option()`, e.g.:

```
tet.set_option("logging={filename={debug.log} remove}")
```
- ▶ In an environment variable called `TETLOGGING`. Doing so will activate the logging output starting with the very first call to one of the API functions.

Table 10.7 Suboptions for the logging option of `TET_set_option()`

key	explanation
(empty list)	Enable log output after it has been disabled with <code>disable</code> .
disable	(Boolean) Disable logging output. Default: false
enable	(Boolean) Enable logging output
filename	(String) Name of the log file (stdout and stderr are also acceptable). Output will be appended to any existing contents. The log file name can alternatively be supplied in an environment variable called <code>TET_LOGFILENAME</code> (in this case the option <code>filename</code> will always be ignored). Default: <code>tet.log</code> (on Windows and OS X in the <code>/</code> directory, on Unix in <code>/tmp</code>)
flush	(Boolean) If true, the log file will be closed after each output, and reopened for the next output to make sure that the output will actually be flushed. This may be useful when chasing program crashes where the log file is truncated, but significantly slows down processing. If false, the log file will be opened only once. Default: false
remove	(Boolean) If true, an existing log file will be deleted before writing new output. Default: false
stringlimit	(Integer) Limit for the number of characters in text strings, or 0 for unlimited. Default: 0

Table 10.7 Suboptions for the logging option of `TET_set_option()`

key	explanation
classes	(Option list) List containing options of type integer, where each option describes a logging class and the corresponding value describes the granularity level. Level 0 disables a logging class, positive numbers enable a class. Increasing levels provide more and more detailed output. The following options are supported (default: {api=1 warning=1}): api Log all API calls with their function parameters and results. If api=2 a timestamp will be created in front of all API trace lines, and deprecated functions and options will be marked. If api=3 try/catch calls will be logged (useful for debugging problems with nested exception handling). filesearch Log all attempts related to locating files via SearchPath or PVE. resource Log all attempts at locating resources via Windows registry, UPR definitions as well as the results of the resource search. user User-specified logging output supplied with the userlog option. warning Log all warnings, i.e. error conditions which can be ignored or fixed internally. If warning=2 messages from functions which do not throw any exception, but hook up the message text for retrieval via <code>TET_get_errmsg()</code> , and the reason for all failed attempts at opening a file (searching for a file in searchpath) will also be logged.

10.7 Document Functions

C++	<code>int open_document(wstring filename, wstring optlist)</code>
C# Java	<code>int open_document(String filename, String optlist)</code>
Perl PHP	<code>long open_document(string filename, string optlist)</code>
VB RB	<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) 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 file-names 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 10.8. The following options can be used: *checkglyphlists*, *decompose*, *encodinghint*, *fold*, *glyphmapping*, *lineseparator*, *normalize*, *inmemory*, *password*, *repair*, *requiredmode*, *shrug*, *tetml*, *usehostfonts*, *wordseparator*,

Returns -1 on error, or a document handle otherwise. For example, it is an error if the input document or the TETML output file cannot be opened. If -1 is returned it is recommended to call *TET_get_errmsg()* to find out more details about the error.

Details Within a single TET object an arbitrary number of documents may be kept open at the same time. However, a single TET object must not be used in multiple threads simultaneously without any locking mechanism for synchronizing the access.

Encryption: if the document is encrypted its user password must be supplied in the *password* option if the permission settings allow content extraction. The document's master password must be supplied if the permission settings do not allow content extraction. If the *requiredmode* option has been specified, documents can be opened even without the appropriate password, but operations are restricted. The *shrug* option can be used to enable content extraction from protected documents under certain conditions (see Section 5.1, »Extracting Content from protected PDF«, page 59).

Supported file systems on i5/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 10.8 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>
decompose	<p>(Keyword or option list) Unicode decompositions which will be applied to all characters which have a specified Unicode decomposition tag and are part of the specified Unicode set. These conditions are provided in the suboption name and value. Decompositions can be used to either remove or preserve the distinction between equivalent Unicode characters (see Section 7.3, »Unicode Postprocessing«, page 97). Default: see »Default decompositions«, page 103. However, if the <code>normalize</code> option has a value other than <code>none</code>, all default decompositions are disabled, i.e. setting the <code>normalize</code> option sets the default to <code>decompose=none</code>. However, user-specified decompositions can still be applied.</p> <p>The following keywords can be supplied instead of a list:</p> <p>none No decompositions will be applied.</p> <p>default The default decompositions (see »Default decompositions«, page 103) will be applied before other specified decompositions.</p> <p>The following suboptions for decompositions are supported:</p> <p>canonical, circle, compat, final, font, fraction, initial, isolated, medial, narrow, nobreak, small, square, sub, super, vertical, wide</p> <p>Each of these suboptions accepts a string or keyword which specifies the decomposition's domain, i.e. the set of Unicode characters to which the decomposition will be applied. A string specifies a Unicode set for the domain. This can be used to restrict decompositions to subsets of the characters with the specified decomposition tag. Characters outside the domain will not be modified.</p> <p>As an alternative to a string for a Unicode set the following keywords can be supplied:</p> <p>_all The set of all Unicode characters, i.e. the decomposition will be applied to all characters with the specified decomposition tag.</p> <p>_none The empty set, i.e. the decomposition will not be applied at all.</p>
encodinghint	(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>

Table 10.8 Document options for `TET_open_document()` and `TET_open_document_callback()`

option	description
fold	<p>(Keyword or list of lists; the first element of each inner list is a Unicode set or keyword, the second element is a <code>Unichar</code> or a keyword) Apply a folding (equivalence mapping) to all characters in a folding domain specified as a Unicode set. The foldings will be applied to all text except separator characters added by the <code>lineseparator</code> or <code>wordseparator</code> options (see see Section 7.3, »Unicode Postprocessing«, page 97). Default: see Table 7.3, page 99.</p> <p>The following keyword can be supplied instead of a list:</p> <p>none No foldings will be applied.</p> <p>The following keyword can be supplied instead of a sublist:</p> <p>default The default foldings will be applied before other specified foldings.</p> <p>The first element of each list specifies the folding's domain, i.e. the set of Unicode characters to which the folding will be applied. A string specifies a Unicode set for the domain. If a character is included in multiple sets specified within the <code>fold</code> option, the first matching set definition has priority over all others. In order to avoid problems it is recommended to use disjoint sets.</p> <p>As an alternative to a Unicode set the following keyword can be supplied:</p> <p>_dehyphenation</p> <p>The folding will be applied to hyphen characters which have been found within hyphenated words at line breaks. These characters will be flagged in the <code>attributes</code> member returned by <code>TET_get_char_info()</code> and the <code>@dehyphenation</code> attribute in TETML.</p> <p>The second element in each list contains the target character or action for the folding. It is specified with one of the following variants:</p> <p>(Unichar) Replace all characters in the domain with the specified Unicode character.</p> <p>remove All characters in the domain will be removed.</p> <p>preserve The characters in the domain will not be modified.</p> <p>unknownchar</p> <p>Replace all characters in the domain with the character specified in the <code>unknownchar</code> option.</p>
glyphmapping	<p>(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 font name wildcard <code>»*«</code>, preceding mappings will no longer be used. Each rule consists of an option list according to Table 10.9. All glyph mappings which match a particular font name will be applied to this font. (default: pre-defined internal glyph rules will be applied).</p> <p>Note that glyph mapping rules can also be specified as an external resource in the UPR file (see Section 5.2, »Resource Configuration and File Searching«, page 61).</p>
lineseparator	<p>(<code>Unichar</code>; Only for <code>granularity=page</code>) Character to be inserted between lines². Default: <code>U+000A</code></p>
normalize	<p>(Keyword) Normalize the text output to one of the Unicode normalization forms (default: none):</p> <p>none Do not apply any normalization.</p> <p>nfc Normalization Form C (NFC): canonical decomposition followed by canonical composition</p> <p>nfd Normalization Form D (NFD): canonical decomposition</p> <p>nfkfc Normalization Form KC (NFKC): compatibility decomposition followed by canonical composition</p> <p>nfkfd Normalization Form KD (NFKD): compatibility decomposition</p> <p>Since the Unicode normalization forms involve canonical and compatibility decompositions, combinations of the options <code>decompose</code> and <code>normalize</code> must be constructed carefully. Setting the <code>normalize</code> option to a value different from <code>none</code> sets the <code>decompose</code> default to <code>decompose=none</code>. The <code>normalize</code> option is processed after the <code>decompose</code> option.</p>
inmemory	<p>(Boolean; Only for <code>TET_open_document()</code>) If true, TET will load the complete file into memory and process it from there. This can result in a tremendous performance gain on some systems (especially MVS) at the expense of memory usage. If false, individual parts of the document will be read from disk as needed. Default: false</p>

Table 10.8 Document options for `TET_open_document()` and `TET_open_document_callback()`

option	description
password	<p>(String) The user, master or attachment password for encrypted documents. If the document's permission settings allow text copying then the user password is sufficient, otherwise the master password must be supplied.</p> <p>See the pCOS Path Reference to find out how to query a document's encryption status, and pCOS operations which can be applied even without knowing the user or master password.</p> <p>The shrug option can be used to enable content extraction from protected documents under certain conditions (see Section 5.1, »Extracting Content from protected PDF«, page 59).</p>
repair	<p>(Keyword) Specifies how to treat damaged PDF documents. Repairing a document takes more time than normal parsing, but may allow processing of certain damaged PDFs. Note that some documents may be damaged beyond repair (default: auto):</p> <p>force Unconditionally try to repair the document, regardless of whether or not it has problems.</p> <p>auto Repair the document only if problems are detected while opening the PDF.</p> <p>none No attempt will be made at repairing the document. If there are problems in the PDF the function call will fail.</p>
requiredmode	<p>(Keyword) The minimum pcosmode (minimum/restricted/full) which is acceptable when opening the document. The call will fail if the resulting pcosmode (see the pCOS Path Reference) would be lower than the required mode. If the call succeeds it is guaranteed that the resulting pcosmode is at least the one specified in this option. However, it may be higher; e.g. requiredmode=minimum for an unencrypted document will result in full mode. Default: full</p>
shrug	<p>(Boolean) If true, the shrug feature will be activated to enable content extraction from protected documents under certain conditions (see Chapter 5.1, »Extracting Content from protected PDF«, page 59). By using the shrug option you assert that you will honor the PDF document author's rights. Default: false</p>
tetml	<p>(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:</p> <p>elements (List of Boolean) Specify whether certain TETML elements will be included in the output (default: all true):</p> <p> docinfo The /TET/Document/DocInfo element</p> <p> docxmp The /TET/Document/Metadata element</p> <p> options The elements /TET/Document/Options and /TET/Document/Pages/Page/Options</p> <p>encodingname</p> <p> (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):</p> <p> _none No encoding declaration will be created; the output will still be in UTF-8 format.</p> <p> UTF-8 The declaration encoding="UTF-8" will be created.</p> <p> 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.</p> <p>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.</p>
unknown-char	<p>(Unichar) Character to be used as a replacement for characters which cannot be mapped to Unicode because of inconsistent or missing information in the PDF document. U+0000 means that unknown characters will be removed. Default: U+FFFD (Replacement Character)</p> <p>Related options: use fold={{[:Private_Use:] unknownchar}} to also replace unknown (PUA) characters with the specified unknownchar, or fold={{[:Private_Use:] remove}} to remove them.</p>

Table 10.8 Document options for `TET_open_document()` and `TET_open_document_callback()`

option	description
usehostfonts	(Boolean) If true, data for fonts which are not embedded, but are required for determining Unicode mappings will be searched on the OS X or Windows host operating system. Default: true
wordseparator	(Unichar; Only for granularity=line and page) Character to be inserted between words ² . Default: U+0020

1. See footnote 2 in Table 10.9
2. Use U+0000 to disable the separator.

Table 10.9 Suboptions for the glyphmapping option of `TET_open_document()` and `TET_open_document_callback()`

option	description
codelist	(String) Name of a codelist resource to be applied to the font. It will have higher priority than an embedded ToUnicode CMap or encoding entry.
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: *
fonttype	(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) Fonts with an 8-bit encoding: Replace the first encoding with the encoding resource specified by the second name. If only one entry is supplied, the specified encoding 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. CID fonts: Only the single value unicode is supported. It interprets CID values as Unicode values.
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
remove	(Boolean) If true, all text which uses the specified font name(s) and/or font type(s) will be removed from the retrieved text.

Table 10.9 Suboptions for the glyphmapping option of `TET_open_document()` and `TET_open_document_callback()`

option	description
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 stand-alone character »*« denotes all fonts; Using »*« after a prefix (e.g. »MSTT*«) denotes all fonts starting with the specified prefix.
- 2. Encoding name according to Section 10.5, »Encoding Names«, page 147

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

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 10.8.

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 close_document(long doc)
VB RB 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.

10.8 Page Functions

C++	<code>int open_page(int doc, int pagenumber, wstring optlist)</code>
C# Java	<code>int open_page(int doc, int pagenumber, String optlist)</code>
Perl PHP	<code>long open_page(long pagenumber, string optlist)</code>
VB RB	<code>Function open_page(doc As Long, pagenumber As Long, optlist As String) As Long</code>
C	<code>int TET_open_page(TET *tet, int doc, int pagenumber, const char *optlist)</code>

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 10.10. The following options can be used: `clippingarea`, `contentanalysis`, `docstyle`, `excludebox`, `fontsize`, `granularity`, `ignoreinvisibletext`, `imageanalysis`, `includebox`, `layoutanalysis`, `layouteffort`, `skipengines`, `structureanalysis`, `topdown`.

Returns A handle for the page, or -1 in case of an error. If -1 is returned it is recommended to call `TET_get_errmsg()` to find out more details about the error.

Details Within a single document an arbitrary number of pages may be kept open at the same time. The same page may be opened multiply with different options. However, options can not be changed while processing a page.

Layer definitions (optional content groups) 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 10.10 Page options for `TET_open_page()` and `TET_process_page()`

option	description
clippingarea	(Keyword; will be ignored if <code>includebox</code> is specified) Specifies the area from which text will be extracted (default: <code>cropbox</code>): mediabox Use the <code>MediaBox</code> (which is always present) cropbox Use the <code>CropBox</code> (the area visible in Acrobat) if present, else <code>MediaBox</code> bleedbox Use the <code>BleedBox</code> if present, else use <code>cropbox</code> trimbox Use the <code>TrimBox</code> if present, else use <code>cropbox</code> artbox Use the <code>ArtBox</code> if present, else use <code>cropbox</code> unlimited Consider all text, regardless of its location
content-analysis	(Option list; not for <code>granularity=glyph</code>) List of suboptions according to Table 10.11 for controlling high-level content analysis and text processing.

Table 10.10 Page options for `TET_open_page()` and `TET_process_page()`

option	description
docstyle	<p>(Keyword) A hint which will be used by the layout detection engine to select various parameters. These parameters optimize layout detection for situations where the document belongs to one of the classes below. If the document is known to fall into one of these classes layout detection results can be improved significantly by supplying a suitable value for this option. This option activates advanced layout recognition (default: none):</p> <p>book Typical book</p> <p>business Business documents</p> <p>cad Technical or architectural drawings which are typically heavily fragmented</p> <p>fancy Fancy pages with complex layout</p> <p>forms Structured forms</p> <p>generic The most general document class without any further qualification.</p> <p>magazines Magazine articles</p> <p>none No specific document style is known and advanced layout recognition will be disabled.</p> <p>papers Newspaper</p> <p>science Scientific article</p> <p>searchengine The application is a search engine indexer or similar application, and mainly interested in retrieving the word list for the page as fast as possible. Table and page structure recognition are disabled.</p> <p>spacegrid List-oriented report (often generated on mainframe systems) where the visual layout is generated using space characters. Since many heuristics like shadow detection and sophisticated word boundary detection are not required for this class of documents text extraction can be accelerated with this option.</p>
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	<p>(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 84, for more details (default: word).</p> <p>glyph A fragment contains the result of mapping one glyph, but may contain more than one character (e.g. for ligatures).</p> <p>word A fragment contains a word as determined by the Wordfinder.</p> <p>line A fragment contains a line of text, or the closest approximation thereof. Word separators will be inserted between two consecutive words.</p> <p>page A fragment contains the contents of a single page. Word, line, and zone separators will be inserted as appropriate.</p>
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 10.13 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 10.12 for controlling layout detection features.

Table 10.10 Page options for `TET_open_page()` and `TET_process_page()`

option	description
layouteffort	(Keyword) Controls the quality/performance trade-off of layout recognition. Layout recognition can be improved by spending more effort, but this may slow down operation. The layout recognition effort can be controlled with the keywords none, low, medium, high, and extra. Default: low
layouthint	(Option list) Inform the layout recognition engine about the presence of certain page layout elements: subsummary (Keyword) Informs the engine about the presence of subsummaries (marginalia) and possibly also their position. Supported keywords (default: none) : auto No subsummary detection left Try to detect subsummaries on the left side of the page. none Try to detect subsummaries automatically. right Try to detect subsummaries on the right side of the page. header (Boolean) If true, the engine tries to detect page headers (default: false). footer (Boolean) If true, the engine tries to detect page footers (default: false).
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 10.14 for controlling page structure analysis.
topdown	(Option list) Specify a coordinate system with the origin in the top left corner of the visible page, and y coordinates which increase downwards; otherwise the default coordinate system with the origin in the lower left corner will be used. Enabling topdown coordinates enables the same coordinate system which is displayed in Acrobat. Supported suboptions: input (Boolean) If true, enable coordinates for the following items (default: false): page options includebox, excludebox output (Boolean) If true, enable coordinates for the following items (default: false): TET_char_info: y, alpha, beta TET_image_info: y, alpha, beta TETML: Glyph/@y, Glyph/@alpha, Glyph/@beta, Box/@lly, Box/@ury, PlacedImage/@y, PlacedImage/@alpha, PlacedImage/@beta

Table 10.11 Suboptions for the contentanalysis option of `TET_open_page()` and `TET_process_page()`

option	description
bidirectional	(Keyword) Will be ignored for granularity=glyph; has an effect only if right-to-left characters are present on the page) Control the inverse Bidi algorithm which reorders right-to-left and left-to-right text in a chunk (default: logical): visual Keep RTL and LTR characters in a chunk in visual order, i.e. do not apply the inverse Bidi algorithm logical Apply the inverse Bidi algorithm to bring the characters in a chunk in logical order.
bidirectionlevel	(Keyword) Specify the page's base level (i.e. the main direction of text progression) for the inverse Bidi algorithm (default: auto): auto Determine the main direction of text progression heuristically based on the content. ltr Assume left-to-right as main direction of text progression (e.g. Latin documents) rtl Assume right-to-left as main direction of text progression (e.g. Hebrew or Arabic documents)
dehyphenate	(Boolean) If true, hyphenated words will be identified and the text fragments surrounding the hyphen will be combined. The hyphen itself will be treated according to the keep hyphens option. 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 neighboring 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)
ideographic	(Keyword) Control word boundary detection for ideographic characters. It is recommended to set this option to keep although the default is split for compatibility reasons (default: split): keep Ideographic characters generally don't constitute a word boundary. Punctuation and the transition between ideographic and non-ideographic characters still constitute a word boundary. For granularity=word ideographic comma U+3001 and ideographic full stop U+3002 also constitute word boundaries. For granularity=page no line separator will be inserted at the end of a line. split Ideographic characters always constitute a word boundary.
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): 0 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. 1 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. 2 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.

Table 10.11 Suboptions for the contentanalysis option of `TET_open_page()` and `TET_process_page()`

option	description
keephyphen-glyphs	(Boolean) If true and dehyphenate=true the hyphen glyph between parts of dehyphenated words will be kept in the list of glyphs returned by <code>TET_get_char_info()</code> and the Glyph element in TETML. This is useful for applications which need detailed information about the position of hyphens, e.g. exactly replacing text on the page. Note that this is different from fold={{ dehyphenation remove} which only removes hyphens from the logical text returned by <code>TET_get_text()</code> , but does not affect glyphs. Default: false
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>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>1</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>2</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>
numeric-entities	(Keyword) Control word boundary detection for numeric entities such as numbers, fractions, and time (default: keep): <div><div>split</div>Split the entity according to the punctuationbreaks suboption.<div>keep</div>Keep the entity as a whole word.</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. For example, this option affects treatment of URLs and mail addresses Default: true
superscript	(Integer) Controls subscript and superscript detection (default: 2): <div><div>0</div>No subscript and superscript detection<div>1</div>Simple subscript and superscript detection<div>2</div>Advanced algorithm for subscript and superscript detection</div>

Table 10.12 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. This keyword should be used with layouteffort=low.
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 of larger layout rows. This is recommended for complex layouts, such as magazines and papers where paragraphs within columns are separated from each other by more than the font size, and for layouts with several multi-column articles one under the other. thin Try to separate neighboring zones and place them in different layout rows. This results in a larger number of smaller layout rows. Example: layoutanalysis = {layoutrowhint={full separation=thick}}
mergetables	(Integer) Tables with a single row 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 10.12 Suboptions for the layoutanalysis option of `TET_open_page()` and `TET_process_page()`

option	description
supertable-columns	(Integer; only if layoutastable=true) Minimum number of columns in a layout row to consider the sequence of zones as a supertable. When a table is created from paragraphs, these columns are recognized as separate zones instead of being combined. As a consequence of this, layout recognition can identify these zone sequences as a table (default: 4).
tabledetect	(Integer) Specifies the depth of recursive table recognition (default: 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 10.13 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 is 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 are 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++
C# Java
Perl PHP
VB RB
C

void close_page(int page)
void close_page(int page)
close_page(long page)
Sub close_page(page As Long)
void TET_close_page(TET *tet, int page)

Release a page handle and all related resources.

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

Details All open pages of the document 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 10.14 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:</p> <pre>bullets={{fontname=ZapfDingbats}} bullets={{bulletchars={U+2022}}} bullets={{fontname=KozGoPro-Medium bulletchars={U+2460 U+2461 U+2462 U+2463 U+2464}}}</pre>
list	(Boolean) Enable list recognition (default: <code>false</code>). If <code>false</code> , no information about list structure will be determined.
paragraph	(Boolean) Enable paragraph recognition (default: <code>true</code>). If <code>false</code> , no information about paragraph structure will be determined.
table	(Boolean) Enable table recognition (default: <code>true</code>). If <code>false</code> , the table recognition engine will be disabled.

10.9 Text and Metrics Retrieval Functions

C++	<code>wstring get_text(int page)</code>
C# Java	<code>String get_text(int page)</code>
Perl PHP	<code>string get_text(long page)</code>
VB RB	<code>Function get_text(page As Long) As String</code>
C	<code>const char *TET_get_text(TET *tet, int page, int *len)</code>

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 depending on the *outputformat* option of `TET_set_option()`:

If *outputformat=utf8* the length is reported as number of Unicode characters. The number of bytes in the null-terminated string (which is identical to the number of 8-bit code units) can be determined with the *strlen()* function.

If *outputformat=utf16* the length is reported as number of 16-bit code units; surrogate pairs are counted as two code units. The number of bytes in the string is *2*len*.

If *outputformat=utf32* the length is reported as number of 32-bit code units (which is identical to the number of Unicode characters). The number of bytes in the string is *4*len*.

Returns A string containing the next text fragment on the page. The length of the fragment is determined by the *granularity* option of `TET_open_page()`. Even for *granularity=glyph* the string may contain more than one character (see Section 7.1, »Important Unicode Concepts«, page 91).

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 is provided as null-terminated UTF-8 (default) or UTF-16/UTF-32 string according to the *outputformat* option of `TET_set_option()`. On i5/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 is provided as Unicode string in UTF-16 format (*wstring* in C++). If no more text is available an empty string will be returned.

Java, .NET and Objective-C: the result is provided as Unicode string. If no more text is available a null (*nil* in Objective-C) object will be returned.

Perl, PHP, Python and Ruby language bindings: the result is provided as UTF-8 (default) or UTF-16/UTF-32 string according to the *outputformat* option of `TET_set_option()`. In Python 3 UTF-16/UTF-32 results are returned as bytes. If no more text is available a null object will be returned.

REALbasic: the result is provided as Unicode string. If no more text is available an empty string will be returned.

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

```
C++  const TET_char_info *get_char_info(int page)
C# Java  int get_char_info(int page)
Perl PHP  object get_char_info(long page)
VB RB  Function get_char_info(int page) As Long
C  const TET_char_info *TET_get_char_info(TET *tet, int page)
```

Get detailed information for the next glyph in the most recent text chunk.

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

Note The name of this function is a misnomer. It should better be called `TET_get_glyph_info()` since it reports information about visual glyphs on the page, not the corresponding Unicode characters.

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

Details This function can be called one or more times after `TET_get_text()`. It will advance to the next glyph for the current text chunk associated with the supplied page handle (or return nothing if there are no more glyphs), and provide detailed information for this glyph. There will be $N > 0$ successful calls to this function (corresponding to N glyphs) for a text chunk with M logical characters. The relationship between N and M depends on the granularity:

- ▶ For *granularity=glyph* each text chunk corresponds to a single glyph, i.e. $N = 1$. One glyph corresponds to one character in many cases, i.e. $M = 1$. However, for ligature glyphs multiple characters correspond to a single glyph, i.e. $M > 1$ and `TET_get_char_info()` must be called more than once.
- ▶ For granularities other than *glyph* a sequence of glyphs results in a sequence of characters, where each glyph may contribute to 0, 1, or more characters. The sequence of glyphs serves as raw material for the sequence of Unicode characters. In other words, there is no known relationship between N and M . The relationship between N and M may be influenced by content analysis (e.g. hyphens are removed by the dehyphenation process) or Unicode postprocessing (e.g. characters are added or deleted because of a folding).

For granularities other than *glyph* this function advances to the next glyph which contributes to the chunk returned by the most recent call to `TET_get_text()`. This way it is possible to retrieve glyph metrics when the Wordfinder is active and a text chunk may contain more than one character. In order to retrieve all glyph details for the current text chunk this function must be called repeatedly until it returns no more info.

The glyph details in the structure or properties/fields are valid until the next call to `TET_get_char_info()` or `TET_close_page()` with the same page handle (whichever occurs first). Since there is only a single set of glyph info properties/fields per TET object, clients must retrieve all glyph info before they call `TET_get_char_info()` again for the same or another page or document.

Bindings C and C++ language bindings: If no more glyphs are available for the most recent text chunk returned by `TET_get_text()`, a NULL pointer will be returned. Otherwise, a pointer to a `TET_char_info` structure containing information about a single glyph will be returned. The members of the data structure are detailed in Table 10.15.

COM, Java, .NET, and Objective-C language bindings: -1 will be returned if no more glyphs are available for the most recent text chunk returned by `TET_get_text()`, otherwise 1. Individual glyph info can be retrieved from the TET properties/public fields according to Table 10.15. All properties/fields contain the value -1 (the *unknown* field contains *false*) if they are accessed although the function returned -1.

Perl and Python language bindings: 0 will be returned if no more glyphs are available for the most recent text chunk returned by `TET_get_text()`, otherwise a hash containing the keys listed in Table 10.15. Individual glyph info can be retrieved with the keys in this hash.

PHP language binding: an empty (null) object will be returned if no more glyphs are available for the most recent text chunk returned by `TET_get_text()`, otherwise an object containing the fields listed in Table 10.15. Individual glyph info can be retrieved from the member fields of this object. Integer fields in the glyph info object are implemented as *long* in the PHP language binding.

REALbasic binding: *nil* will be returned if no more glyphs are available for the most recent text chunk returned by `TET_get_text()`, otherwise a `TET_char_info` object containing the members listed in Table 10.15. Individual glyph info can be retrieved with the keys in this object. The *attributes* field is called *attrs* in the REALbasic binding to work around a REALbasic interface problem.

Ruby binding: *nil* (null object) will be returned if no more glyphs are available, and a `TET_char_info` object otherwise.

Table 10.15 Members of the `TET_char_info` structure (C, C++, Ruby), equivalent public fields (Java, PHP, Objective-C), keys (Perl) or properties (COM and .NET) with their type and meaning. See »Glyph metrics«, page 74, and Figure 6.3 for more details.

property/ field name	explanation
uv	(Integer) UTF-32 Unicode value for the current glyph. For granularities other than glyph this may be an artificial or intermediate value which has no relationship to the final text chunk. For granularity=glyph the sequence of Unicode values for the glyphs is identical to the logical text, but for other granularities it may be modified by various processing steps.
type	(Integer) Type of the character. The following types describe real characters which correspond to a glyph on the page. The values of all other properties/fields are determined by the corresponding glyph: <div><div>0</div>Normal character which corresponds to exactly one glyph</div> <div><div>1</div>Start of a sequence (e.g. ligature)</div> <div>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: <div><div>10</div>Continuation of a sequence (e.g. ligature)</div><div><div>11</div>(Deprecated and unused)</div><div><div>12</div>Inserted word, line, or zone separator</div></div>

Table 10.15 Members of the `TET_char_info` structure (C, C++, Ruby), equivalent public fields (Java, PHP, Objective-C), keys (Perl) or properties (COM and .NET) with their type and meaning. See »Glyph metrics«, page 74, and Figure 6.3 for more details.

property/ field name	explanation
attributes¹	(Integer) Glyph attributes expressed as bits which can be combined: bit 0 Geometric or semantic subscript bit 1 Geometric or semantic superscript bit 2 Drop cap character (initial large character at the start of a paragraph) bit 3 Glyph- or word-based shadow duplicate of this glyph has been removed bit 4 Glyph represents last character before hyphenation point bit 5 Hyphenation artifact (i.e. the hyphen character) which was removed unless <code>contentanalysis={keeplyphenglyphs=true}</code> was specified. bit 6 Glyph represents the character after hyphenation point
unknown	(Boolean, in C, C++ and Perl: 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 <code>unknownchar</code> .
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 (i.e. inserted separators with <code>type=12</code> and hyphenation artifacts with attribute bit 5 set) the width is 0.
alpha	(Double) Direction of inline text progression in degrees measured counter-clockwise (or clockwise for top-down coordinates). For horizontal writing mode this is the direction of the text baseline; for vertical writing mode it is the digression from the standard vertical direction. The angle will be in the range $-180^{\circ} < \alpha \leq +180^{\circ}$. 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 measured counter-clockwise (or clockwise for topdown coordinates), relative to the perpendicular of alpha. The angle will be 0° for upright text, and negative for italicized (slanted) text (positive for topdown coordinates). The angle will be in the range $-180^{\circ} < \beta \leq 180^{\circ}$, but different from $\pm 90^{\circ}$. If $\text{abs}(\beta) > 90^{\circ}$ the text is mirrored at the baseline.
fontid	(Integer) Index of the font in the <code>fonts[]</code> pseudo object (see the pCOS Path Reference). <code>fontid</code> 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

1. In the REALbasic binding this field is called `attrs`.

10.10 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 image_info get_image_info(long page)</code>
VB RB	<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 10.16.

COM, Java, .NET, and Objective-C 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 10.16. All properties/fields contain the value -1 if they are accessed although the function returned -1.

Perl and Python language bindings: 0 will be returned if no more images are available on the page, otherwise a hash containing the keys listed in Table 10.16. Individual image 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 10.16. Integer fields in the image info object are implemented as *long* in the PHP language binding.

REALbasic binding: *nil* will be returned if no more images are available on the page, otherwise a `TET_image_info` object containing the members listed in Table 10.16. Individual image info can be retrieved with the member of this object.

Ruby binding: *nil* (null object) will be returned if no more images are available, and a `TET_image_info` object otherwise.

Table 10.16 Members of the **TET_image_info** structure (C, C++, Ruby), equivalent public fields (Java, PHP, Objective-C), and properties (COM and .NET) with their type and meaning. See »Image Extraction Basics«, page 113, 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 the pCOS Path Reference).

```

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

```

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 10.17. The following options can be used: `compression`, `filename`, `keepiccpfile`, `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`)
- ▶ 40: image extracted as raw pixel data (`.raw`)
- ▶ 50: image extracted as JBIG2 (`.jbig2`)

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 10.17 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.
filename¹	(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. The following file name pattern is recommended to match the <code>Image/@id</code> attribute in TETML: I<imageid> Here <code>imageid</code> is the decimal representation of the <code>imageid</code> parameter.
keepiccpfile	(Boolean) If true and an ICC profile is assigned to the image, the ICC profile is embedded in extracted TIFF and JPEG images. Setting this option to false may result in smaller image files, but sacrifices color management. Default: true
keepxmp	(Boolean) If true and the image has associated XMP metadata in the PDF, the metadata will be embedded in extracted TIFF and JPEG images. Default: true

Table 10.17 Options for `TET_write_image_file()` and `TET_get_image_data()`

option	description
preferredtiff-compression	(Keyword) Compression scheme used for most extracted TIFF images (default: flate):
	lzw LZW compression (TIFF compression scheme 5)
	flate Flate compression, also called Adobe Deflate or zlib compression (TIFF compression scheme 8)
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++	<code>const char *get_image_data(int doc, size_t *length, int imageid, wstring optlist)</code>
C# Java	<code>final byte[] get_image_data(int doc, int imageid, String optlist)</code>
Perl PHP	<code>string get_image_data(long doc, long imageid, string optlist)</code>
VB RB	<code>Function get_image_data(doc As Long, imageid As Long, optlist As String)</code>
C	<code>const char *TET_get_image_data(TET *tet, int doc, size_t *length, int imageid, const char *optlist)</code>

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 10.17. 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.

REALbasic: the result will be provided as REALbasic string with encoding -1 (binary data). If no more text is available an empty string will be returned.

10.11 TET Markup Language (TETML) Functions

C++	<code>int process_page(int doc, int pagenumber, wstring optlist)</code>
C# Java	<code>int process_page(int doc, int pagenumber, String optlist)</code>
Perl PHP	<code>long process_page(long doc, long pagenumber, string optlist)</code>
VB RB	<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 10.10 (these will be ignored if `pagenumber=0`): `clippingarea`, `contentanalysis`, `excludebox`, `fontsize`, `granularity`, `ignoreinvisibletext`, `imageanalysis`, `includebox`, `layoutanalysis`, `skipengines`
 - Option specifying processing details according to Table 10.18: `tetml`

Table 10.18 Additional options for `TET_process_page()`

option	description
tetml	(Option list) Controls details of TETML. The following options are available: <ul style="list-style-type: none">elements (Option list) Specify optional TETML elements:<ul style="list-style-type: none">line (Only for granularity=word) If true, TETML output includes Line elements between Para and Word levels. Default: falseglyphdetails (Option list; only for granularity=glyph and word) Specify which glyph attributes will be reported for each Glyph element (default for all suboptions: false):<ul style="list-style-type: none">all (Boolean) Enable all attribute suboptionsdehyphenation (Boolean) Emit attribute dehyphenation to indicate hyphenated words.dropcap (Boolean) Emit attribute dropcap to indicate large initial characters at the start of a paragraph.geometry (Boolean) Emit attributes x, y, width, alpha, beta.font (Boolean) Emit attributes font, fontsize, textrendering, unknown.sub (Boolean) Emit attribute sub to indicate subscripts.sup (Boolean) Emit attribute sup to indicate superscripts.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 pagenumber=0 only trailer data (without any page-specific data) will be emitted. Once trailer=true 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 will be reported in a TETML *Exception* element.

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++	<code>const char *get_xml_data(int doc, size_t *length, wstring optlist)</code>
C# Java	<code>final byte[] get_xml_data(int doc, String optlist)</code>
Perl PHP	<code>string get_xml_data(long doc, string optlist)</code>
VB RB	<code>Function get_xml_data(doc As Long, optlist As String)</code>
C	<code>const char *TET_get_xml_data(TET *tet, int doc, size_t *length, const char *optlist)</code>

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 i5/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.

REALbasic: The result will be returned as REALbasic String with encoding UTF-8.

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

Python: the result will be returned as 8-bit string (Python 3: *bytes*).

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

10.12 pCOS Functions

The full pCOS syntax for retrieving object data from a PDF is supported. For a detailed description please refer to the pCOS Path Reference which is available as a separate document.

C++
C# Java
Perl PHP
VB RB
C

double pcos_get_number(int doc, wstring path)
double pcos_get_number(int doc, String path)
float pcos_get_number(int doc, string path)
Function pcos_get_number(doc as Long, path As String) As Double
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++
C# Java
Perl PHP
VB RB
C

wstring pcos_get_string(int doc, wstring path)
String pcos_get_string(int doc, String path)
string pcos_get_string(int doc, string path)
Function pcos_get_string(doc as Long, path As String) As String
const char *TET_pcos_get_string(TET *tet, int doc, const char *path, ...)

Get the value of a pCOS path with type *name*, *number*, *string*, or *boolean*.

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

path A full pCOS path for a string, name, or boolean object.

Additional parameters (C language binding only) A variable number of additional parameters can be supplied if the *key* parameter contains corresponding placeholders (%s for strings or %d for integers; use %% for a single percent sign). Using these parameters will save you from explicitly formatting complex paths containing variable numerical or string values. The client is responsible for making sure that the number and type of the placeholders matches the supplied additional parameters.

Returns A string with the value of the object identified by the pCOS path. For Boolean values the strings *true* or *false* will be returned.

Details This function raises an exception if pCOS does not run in full mode and the type of the object is *string*. However, the objects */Info** (document info keys) can also be retrieved in

restricted pCOS mode if *nocopy=false* or *plainmetadata=true*, and *bookmarks[...]/Title* as well as all paths starting with *pages[...]/annots[...]/* can be retrieved in restricted pCOS mode if *nocopy=false*.

This 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 language binding: The string will be returned in UTF-8 format (on zSeries and i5/iSeries: EBCDIC-UTF-8) without BOM. 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++ language binding: The string will be returned as *wstring* in the default *wstring* configuration of the C++ wrapper. In *string* compatibility mode on zSeries and i5/iSeries the result will be returned in EBCDIC-UTF-8 without BOM.

Java and .NET bindings: 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 EBCDIC-UTF-8 string.

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

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 10.19.

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, or if the contents of encrypted attachments in an unencrypted document are queried and the attachment password has not been supplied.

If the object has type *stream* all filters will be removed from the stream contents (i.e. the actual raw data will be returned) unless *keepfilter=true*. 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 ASCIIHex 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 the pCOS Path Reference). As an exception, the object */Root/Metadata* can also be retrieved in restricted pCOS mode if *nocopy=false* or *plainmetadata=true*. An exception will also be thrown if *path* does not point to an object of type *stream*, *fstream*, or *string*.

Despite its name this 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).

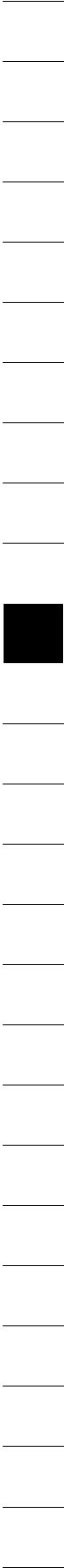
C and C++ language bindings: The returned data buffer can be used until the next call to this function.

Python: the result will be returned as 8-bit string (Python 3: *bytes*).

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 10.19 Options for *TET_pcos_get_stream()*

option	description
convert	(Keyword; ignored for streams which are compressed with unsupported filters) Controls whether or not the string or stream contents 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 the pCOS Path Reference). 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.

Setup Functions

Function prototype	page
(C) TET *TET_new(void)	151
void delete()	151

PVF Functions

Function prototype	page
void create_pvf(String filename, byte[] data, String optlist)	152
int delete_pvf(String filename)	153
int info_pvf(String filename, String keyword)	153

Unicode Conversion Function

Function prototype	page
String convert_to_unicode(String inputformat, byte[] input, String optlist)	155

Exception Handling Functions

Function prototype	page
String get_apiname()	157
String get_errmsg()	157
int get_errnum()	157

Document Functions

Function prototype	page
int open_document(String filename, String optlist)	161
(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)	167
void close_document(int doc)	167

Page Functions

Function prototype	page
int open_page(int doc, int pagenumber, String optlist)	169
void close_page(int page)	175

Text and Metrics Retrieval Functions

Function prototype	page
String get_text(int page)	177
int get_char_info(int page)	178

Image Retrieval Functions

<i>Function prototype</i>	<i>page</i>
<i>int get_image_info(int page)</i>	181
<i>int write_image_file(int doc, int imageid, String optlist)</i>	183
<i>final byte[] get_image_data(int doc, int imageid, String optlist)</i>	184

TET Markup Language (TETML) Functions

<i>Function prototype</i>	<i>page</i>
<i>int process_page(int doc, int pagenumber, String optlist)</i>	185
<i>final byte[] get_xml_data(int doc, String optlist)</i>	186

Option Handling

<i>Function prototype</i>	<i>page</i>
<i>void set_option(String optlist)</i>	148

pCOS Functions

<i>Function prototype</i>	<i>page</i>
<i>double pcos_get_number(int doc, String path)</i>	188
<i>String pcos_get_string(int doc, String path)</i>	188
<i>final byte[] pcos_get_stream(int doc, String optlist, String path)</i>	189

B Revision History

Revision history of this manual

Date	Changes
May 17, 2013	► Updates for TET 4.2
April 04, 2012	► Updates for TET 4.1p1
February 20, 2012	► Updates for TET 4.1
September 22, 2010	► Updates for TET 4.0p2
July 27, 2010	► Updates for TET 4.0
February 01, 2009	► Updates for TET 3.0
January 16, 2008	► Updated the manual for TET 2.3
January 23, 2007	► Minor additions for TET 2.2
December 14, 2005	► Additions and corrections for TET 2.1.0; added descriptions for the PHP and RPG language bindings
June 20, 2005	► Expanded and reorganized the manual for TET 2.0.0
October 14, 2003	► Updated the manual for TET 1.1
November 23, 2002	► Added the description of <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* 71
- API reference* 141
- Arabic* 82
- area of text extraction* 73
- ascender* 76
- attachment password* 59

B

- Basic Multilingual Plane* 91
- bidirectional text* 82
- BMP* 91
- bookmarks* 71
- Boolean values in option lists* 145
- Byte Order Mark (BOM)* 92

C

- C binding* 24
- C++ and .NET* 31
- C++ binding* 26
- canonical decomposition* 100
- capheight* 76
- categories of resources* 61
- characters and glyphs* 91
- CJK (Chinese, Japanese, Korean)* 12, 79
 - compatibility forms* 80
 - configuration* 7
 - word boundaries* 79
- CLI* 26
- codelist* 109
- COM binding* 28
- command-line tool* 17
- comments* 71
- commercial license* 10
- compatibility decomposition* 101
- composite characters* 93
- concordance (XSLT sample)* 137
- connector* 43
- content analysis* 84
- coordinate system* 73
- CSV format* 139

D

- decomposition* 100
- dehyphenation* 86
- descender* 76
- Dispose()* 151
- document and page functions* 161

- document domains* 69
- document info entries* 69
- document styles* 88
- double-byte variants* 80

E

- end points of glyphs and words* 77
- evaluation version* 7
- examples*
 - text extraction status* 59
 - XSLT* 137
- exception handling* 23
 - in C* 24

F

- fake bold removal* 86
- file attachments* 71
- file search* 62
- float and integer values in option lists* 146
- folding* 97
- font filtering (XSLT sample)* 137
- font statistics (XSLT sample)* 138
- FontReporter plugin* 11, 108
- form fields* 71
- fullwidth variants* 80

G

- geometry of images* 119
- glyph metrics* 74
- glyph rules* 111
- glyphlist* 111
- glyphs* 91
- granularity* 84

H

- halfwidth variants* 80
- Hebrew* 82
- highlighting* 77
- HTML converter (XSLT sample)* 139

I

- ideographic text*
 - word boundaries* 79
- IFilter for Microsoft products* 52
- images*
 - color fidelity* 121
 - determining type* 113

- extract to disk or memory* 113
- extracting* 113
- formats* 113
- geometry* 119
- merging* 115
- number of images in a document* 116
- page-based extraction loop* 118
- placed images* 117
- resolution* 119
- resource-based extraction loop* 118
- resources* 117
- small image removal* 116
- unsupported types* 121
- XMP metadata* 114
- inch* 73
- index (XSLT sample)* 138
- installing TET* 7

J

- JzEE application servers* 29
- Java binding* 29
- Javadoc* 30
- JBIG2* 113
- JPEG* 113
- JPEG 2000* 113

K

- keywords in option lists* 145

L

- license key* 8
- ligatures* 93
- list values in option lists* 142
- logging* 159
- Lucene search engine* 45

M

- master password* 59
- MediaWiki* 56
- millimeters* 73
- mini samples* 14

N

- nested option lists* 142
- .NET binding* 31
- normalization* 104
- numbers in option lists* 146

O

- Objective-C binding* 32
- optimizing performance* 65
- option list syntax* 141
- option lists* 141
- Oracle Text* 49

- owner password* 59

P

- packages* 72
- page boxes* 73
- page-based image extraction loop* 118
- passwords* 59
- pCOS*
 - API functions* 188
 - Cookbook* 15
- PDF versions* 11
- performance optimization* 65
- Perl binding* 34
- permissions password* 59
- PHP binding* 35
- placed images* 117
- points* 73
- portfolios* 72
- postprocessing* 94
- preprocessing* 94
- prerotated glyphs* 80
- Private Use Area* 92
- protected documents* 59
- PUA* 92
- Python Binding* 37

R

- raw text extraction (XSLT sample)* 139
- REALbasic binding* 38
- rectangles in option lists* 147
- resource configuration* 61
- resource-based image extraction loop* 118
- resourcefile parameter* 64
- response file* 20
- roadmap to documentation and samples* 14
- RPG binding* 41
- Ruby binding* 39

S

- schema* 131
- searching for font usage (XSLT sample)* 138
- searchpath* 62
- sequences* 93
- servlets* 29
- shadow removal* 86
- shrug feature* 59
- single-byte variants* 80
- small image removal* 116
- Solr search server* 48
- strings in option lists* 144
- surrogates* 92
- syntax of option lists* 141

T

- table detection* 90
- table extraction (XSLT sample)* 139

- TET command-line tool 17
- TET connector 43
 - for Lucene 45
 - for MediaWiki 56
 - for Microsoft products 52
 - for Oracle 49
 - for Solr 48
 - for Tika 54
- TET Cookbook 14
- TET features 11
- TET Markup Language (TETML) 123
- TET plugin for Adobe Acrobat 43
- TET_CATCH() 157
- TET_close_document() 167
- TET_close_page() 175
- TET_convert_to_unicode() 155
- TET_create_pvf() 152
- TET_delete_pvf() 153
- TET_delete() 151
- TET_EXIT_TRY() 24, 157
- TET_get_apiname() 157
- TET_get_char_info() 178
- TET_get_errmsg() 157
- TET_get_errnum() 157
- TET_get_image_data() 184
- TET_get_image_info() 181
- TET_get_text() 177
- TET_get_xml_data() 186
- TET_info_pvf() 153
- TET_new() 151
- TET_open_document_callback() 167
- TET_open_document() 161
- TET_open_page() 169
- TET_pcos_get_number() 188
- TET_pcos_get_stream() 189
- TET_pcos_get_string() 188
- TET_RETHROW() 157
- TET_set_option() 148
- TET_TRY() 157
- TET_write_image_file() 183
- tet.upr 63
- TETML 123
 - schema 131
- TETRESOURCEFILE environment variable 63
- TeX documents 68
- text extraction status 59

- text filtering 94
- TIFF 113
- TIKA toolkit 54
- ToUnicode CMap 110

U

- Unichar values in option lists 144
- Unicode
 - BOM 92
 - concepts 91
 - decomposition 100
 - encoding forms 92
 - encoding schemes 92
 - folding 97
 - in option lists 144
 - normalization 104
 - postprocessing 97
 - pre- and postprocessing 94
 - preprocessing 94
 - sets 145
- units 73
- unmappable glyphs 107
- UPR file format 61
- user password 59
- UTF formats 92
- UTF-32 106

V

- vertical writing mode 79

W

- word boundary detection 85
- Wordfinder 85

X

- xheight 76
- XMP metadata 70
 - for images 114
 - XSLT sample 139
- XSD schema for TETML 131
- XSLT 134
 - samples 14, 137

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

Support

support@pdflib.com (*please include your license number*)

