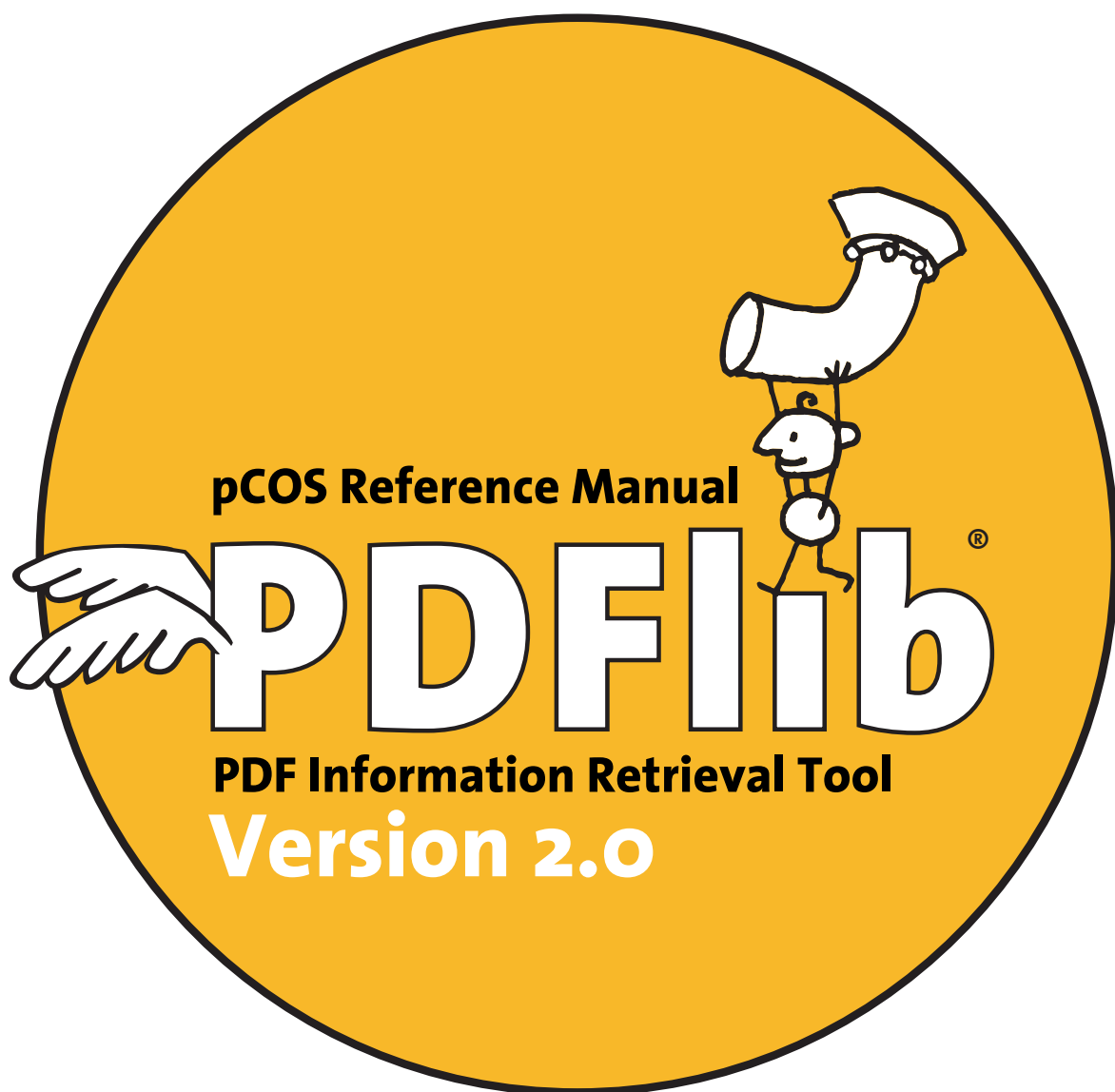


PDFlib GmbH München, Germany

www.pdfliib.com



Copyright © 2005-2007 PDFlib GmbH. All rights reserved.

PDFlib GmbH
Tal 40, 80331 München, Germany
www.pdflib.com

phone +49 • 89 • 29 16 46 87
fax +49 • 89 • 29 16 46 86

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

Licensing contact: sales@pdflib.com
Technical support: support@pdflib.com (please include your license number)

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

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

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

*PDFlib pCOS contains modified parts of the following third-party software:
Zlib compression library, Copyright © 1995-2002 Jean-loup Gailly and Mark Adler
Cryptographic software written by Eric Young, Copyright © 1995-1998 Eric Young (ey@cryptsoft.com)
Cryptographic software, Copyright © 1998-2002 The OpenSSL Project (www.openssl.org)
Independent JPEG Group's JPEG software, Copyright © 1991-1998, Thomas G. Lane*

pCOS contains the RSA Security, Inc. MD5 message digest algorithm.



Contents

o First Steps with pCOS 5

- o.1 Installing the Software 5
- o.2 Applying the pCOS License Key 6
- o.3 What's new in pCOS 2.0? 8

1 pCOS Examples 9

- 1.1 For Starters: simple Mode 9
- 1.2 Extracting Data from PDF 11
- 1.3 For advanced Applications: extended Mode 12
- 1.4 For Experts: raw pCOS Paths 15
- 1.5 For Programmers: pCOS Library Calls 16

2 pCOS Command-Line Reference 19

- 2.1 Option Processing and Exit Codes 19
- 2.2 Input Options 20
- 2.3 Options for Retrieving PDF Elements 21
- 2.4 Advanced Retrieval Options 23
- 2.5 Output Options 25
- 2.6 Unicode Output and Binary Data 27

3 pCOS Path Reference 29

- 3.1 Simple pCOS Examples 29
- 3.2 Handling Basic PDF Data Types 31
- 3.3 Composite Data Structures and IDs 32
- 3.4 Path Syntax 33
- 3.5 Pseudo Objects 35
- 3.6 Encrypted PDF Documents 41

4 pCOS Library Language Bindings 43

- 4.1 Exception Handling 43
- 4.2 C Binding 44
- 4.3 C++ Binding 46
- 4.4 COM Binding 47
- 4.5 Java Binding 48
- 4.6 .NET Binding 49
- 4.7 Perl Binding 50

4.8 PHP Binding 51

5 pCOS Library API Reference 53

5.1 Option Lists 53

5.2 General Functions 54

5.3 Document Functions 55

5.4 Exception Handling 58

5.5 Option Handling 60

5.6 pCOS Query Functions 61

5.7 Unicode Conversion Functions 64

A pCOS Library Quick Reference 67

B Revision History 68

Index 69

o First Steps with pCOS

o.1 Installing the Software

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

- ▶ An introduction to the pCOS features by means of various examples can be found in Chapter 1, »pCOS Examples«, page 9.
- ▶ Users of the pCOS command-line tool can use the executable right away. It can be found in the *bin* subdirectory of the installation directory. The available options are discussed in Chapter 2, »pCOS Command-Line Reference«, page 19, and are also displayed when you execute the pCOS command-line tool without any options.
- ▶ Users of the pCOS library/component should read one of the sections in Chapter 4, »pCOS Library Language Bindings«, page 43, corresponding to their environment of choice, and review the installed examples. On Windows, the pCOS programming examples are accessible via the Start menu.

If you obtained a commercial pCOS license you must enter your pCOS license key according to the next page.

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

o.2 Applying the pCOS License Key

Using pCOS for production purposes requires a valid license key. Once you purchased a pCOS 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 pCOS license keys are platform-dependent, and can only be used on the platform for which they have been purchased.

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

Entering the license key in a license file. Set an environment (shell) variable which points to a license file before pCOS functions are called. If you are using the pCOS library you can alternatively set the path to the license file by setting the *licensefile* parameter with the *pCOS_set_option()* function. The license file must be a text file with the following structure (you can use the license file template *licensekeys.txt* which is contained in all distributions). Lines beginning with a '#' characters contain comments, and will be ignored:

```
# Licensing information for PDFlib GmbH products
PDFlib license file 1.0
pCOS 2.0 ...your license key...
```

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

```
export PDFLIBLICENSEFILE="/path/to/your/license/file"
```

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

```
pCOS --pcosopt "license ...your license key..." ...more options...
```

```
pCOS --pcosopt "licensefile /path/to/your/license/file" ...more options...
```

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

```
pCOS --pcosopt "licensefile {/path/to/license file}" ...more options...
```

Setting the license key with a pCOS library call. If you use the pCOS library, add a line to your script or program which sets the license key at runtime. The *license* option must be set immediately after instantiating the pCOS object, i.e., after calling *pCOS_new()* (in C) or creating a pCOS object (in C++, COM, .NET, and Java):

- In COM/VBScript:

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

► In C:

```
pCOS_set_option(p, "license=...your license key...");
```

► In C++, .NET/C#, and Java:

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

► In Perl:

```
pCOS_set_option($p, "license=...your license key...");
```

o.3 What's new in pCOS 2.0?

pCOS 2.0 extends the set of pseudo objects which were available in pCOS 1.0. The following PDF objects and properties can be conveniently queried without getting bogged down in the details of the PDF syntax:

- ▶ images, including bit depth, color properties, and compression filter
- ▶ color spaces, including simplified access to details of complex color spaces, e.g. Separation
- ▶ other page resources, such as graphics states, patterns, etc.
- ▶ page labels (e.g. roman numerals instead of decimal numbers)
- ▶ PDF/X and PDF/A status
- ▶ article threads (beads) including the number of the destination page

The PDF parser in pCOS 2.0 includes the following improvements:

- ▶ the new repair mode corrects various problems in PDF input documents.
- ▶ supports the AES encryption algorithm so that AES-encrypted PDF documents are accepted as input
- ▶ implements the CCITTFax, RunLengthDecode, and DCTDecode compression filters for image streams
- ▶ Predictors are supported for Flate and LZW compression (relevant for compressed image data)

Other changes:

- ▶ The pCOS cookbook on www.pdflib.com/pcos-cookbook presents dozens of pCOS programming examples.
- ▶ performance improvements for several special cases
- ▶ *pCOS_get_stream()* now also supports string objects in addition to stream objects. This is useful for rare cases where PDF string objects are used to carry binary information, e.g. the color palette in an Indexed color space.
- ▶ *pCOS_get_stream()* now also supports the option *convert* which treats stream contents as textual data, and applies appropriate Unicode conversion. This is relevant, for example, for JavaScript in stream objects.

1 pCOS Examples

The pCOS command-line tool allows you to query information from one or more PDF documents without the need for any programming. In addition, it can be used as a frontend to the pCOS interface. The pCOS command-line tool is built on top of the pCOS library. In the following sections we will present sample calls of the pCOS tool. We will start with simple examples and proceed to more and more complex applications. A detailed list of all command-line options can be found in Chapter 2, »pCOS Command-Line Reference«, page 19.

Then we will demonstrate several examples for users of the pCOS library. These examples show how the functions *pCOS_get_number()*, *pCOS_get_string()*, and *pCOS_get_stream()* can be used to retrieve information from a PDF using the pCOS path syntax.

1.1 For Starters: simple Mode

The first command does not use any options, which means that general information plus all document info entries will be listed:

```
pcos file.pdf
```

The following command lists all fonts used in the document along with their type and embedding status:

```
pcos --font file.pdf
```

The following command creates a hierarchical list of all form fields in the document along with their field type and the field value:

```
pcos --field file.pdf
```

The following command creates a hierarchical list of all bookmarks in the document:

```
pcos --bookmark file.pdf
```

The following command lists the width and height of all pages as well as relevant Box entries (e.g. CropBox) and rotation:

```
pcos --pagesize file.pdf
```

The following command emits information about the PDF/X and PDF/A status of the document:

```
pcos --pdfx --pdfa file.pdf
```

The following command lists all web links on the first two pages:

```
pcos --firstpage 1 --lastpage 2 --weblink file.pdf
```

The following command lists all digital signature fields along with relevant details:

```
pcos --signature file.pdf
```

Understanding pCOS paths in the generated output. In many cases pCOS creates output which not only includes text and numbers found in the PDF document, but also emits pCOS paths which designate an object within the PDF object hierarchy. While the pCOS path syntax is discussed in detail in Chapter 3, »pCOS Path Reference«, page 29, here are a few important notes based on sample output.

The *--weblink* option creates output similar to the following line. The first column contains the pCOS path, while the second column contains the URL. It is important to note that in pCOS syntax page numbering starts at 0, i.e. the first page is designated as *pages[0]*. Similarly, annotations are numbered starting from 0:

```
pages[0]/annots[0]/A/URI: http://www.pdflib.com
```

In extended mode (see Section 1.3, »For advanced Applications: extended Mode«, page 12) the pCOS path can be created using the *PP* variable in a format string.

1.2 Extracting Data from PDF

Note Our product TET (Text Extraction Toolkit) can be used to extract text contents from PDF pages. Text contents can not be extracted with pCOS.

The pCOS command-line tool can be used to extract various data items from PDF documents. The extracted data items will be written to disk files with unique names (based on the name of the input PDF, the data type, and increasing numbers). This section lists several examples for PDF data extraction; see Section 2.3, »Options for Retrieving PDF Elements«, page 21, for more detailed option descriptions.

The following command extracts all file attachments (on page level) in the document:

```
pcos --extract attachment file.pdf
```

The following command extracts all file attachments (on document level) in the document:

```
pcos --extract embeddedfile file.pdf
```

The following command extracts all JavaScripts in the document. Note that a particular script can be used in more than one places (e.g. validation scripts for form fields). In this case the script will be extracted more than once:

```
pcos --extract javascript file.pdf
```

The following command extracts the output intent ICC profile of a PDF/X or PDF/A file:

```
pcos --extract outputintent file.pdf
```

The following command extracts document-level XMP metadata to a file:

```
pcos --extract metadata file.pdf
```

1.3 For advanced Applications: extended Mode

In this section we will present commands which use the extended output mode of pCOS and options for advanced formatting control.

Text output. The following command lists all annotations (links and other types) with their Subtype, destination, the target URL, and the link rectangle coordinates on the page. Double quotes must surround the list of annotation keys since they must be supplied as a single argument to the program:

```
pcos --extended annotation "Subtype destpage A/URI Rect" file.pdf
```

If you have a file with comments from a review process you can list the text in the comments along with the reviewers' name with the following command. The PP variable at the start of the formatting string will create the corresponding pCOS path which includes the page number and the annotation number (both starting at 0). The KEY variable denotes the key (name) of a dictionary entry, which usually is a PDF name object; the VAL variable refers to the corresponding value which may have any type. The parenthesis around the key/value pair mean that this expression will be repeated for all entries in the annotation dictionary.

```
pcos --format "PP (KEY=VAL )\n" --extended annotation "Subtype Contents T" file.pdf
```

The following command lists all file attachments (embedded files):

```
pcos --format "(KEY=VAL )\n" --extended attach "Subtype Contents T Name" file.pdf
```

The following command lists the file name and Author for multiple files. The default headline is disabled since we included the name of the input file (variable IF) in the format string:

```
pcos --headline "" --format "IF:(VAL\n)" --extended docinfo Author *.pdf
```

The following command lists important properties of PDFlib blocks. Double quotes are used to avoid problems with space characters in block names:

```
pcos --bracket dquot --format "(KEY=VAL\n)\n" --extended block "Name Subtype Description" file.pdf
```

The following command creates a table of contents from the bookmark titles and corresponding page numbers; this only works if the bookmarks actually point to a page:

```
pcos --indent 4 --format "(VAL )\n" --extended bookmark "Title destpage" file.pdf
```

The following command lists the names of all named destinations along with the corresponding target page. The pCOS path (variable PP) contains the destination name:

```
pcos --format "PP: page VAL\n" --extended destination destpage file.pdf
```

Tabular output for use in spreadsheet applications. Using the formatting options of pCOS it is easy to create output which can be processed in applications such as Microsoft Excel. The following commands create comma-separated lists of various pieces of information retrieved from an arbitrary number of PDF documents. The required comma and newline characters are created using suitable format strings. The output can be imported in Microsoft Excel and similar spreadsheet applications which support the CSV (comma-separated values) format.

The following command creates a table with the pCOS path (variable PP) containing the page number (starting at 0) in the first column, and the width and height of each page in subsequent columns:

```
pcos --outfile table.csv --format "PP,(VAL,)\n" --extended pagesize "width height"
      file.pdf
```

The following command extends the previous example for use with many files; it creates a table with the file names of all input files (variable IF) along with the pCOS path (variable PP) and the size of all pages. It suppresses the default headline since the input file name is already printed in the first column of each output line:

```
pcos --outfile table.csv --headline "" --format "IF,PP,(VAL,)\n"
      --extended pagesize "width height" file.pdf
```

The following command creates a table of PDFlib block names, types, and position:

```
pcos --outfile table.csv --bracket dquot --format "(VAL,)\n"
      --extended block "Name Subtype fontname Rect[0] Rect[1] Rect[2] Rect[3]" file.pdf
```

The following command creates a table containing the file names (created by the IF variable) and various document info entries:

```
pcos --outfile table.csv --replace missing "" --bracket dquot --headline ""
      --format "IF,(VAL,)\n" --extended docinfo "Title Author Creator Subject" *.pdf
```

The following command creates a table with type, name, and value of form fields. In order to avoid unwanted whitespace we set the indentation to 0. A headline with the names of the extracted field keys is placed at the top. Missing entries are designate with a custom string:

```
pcos --outfile table.csv --indent 0 --headline "FT,fullname,V\n"
      --replace missing "(unavailable)" --format "(VAL,)\n"
      --extended field "FT fullname V" file.pdf
```

The following command creates a table of file names along with all fonts and their embedding status. We place the input file name (variable IF) in the first column of each line, and disable the default heading (which would place the input file name on a separate line) by specifying an empty headline:

```
pcos --outfile table.csv --headline "" --bracket dquot --format "IF,(VAL,)\n"
      --extended font "name type embedded" file.pdf
```

The following command creates a table of all Web links (URL and position). The pCOS path in the first column (variable PP) contains the page and annotation numbers (o-based):

```
pcos --outfile table.csv --format "PP,(VAL,)\n"
      --extended weblink "A/URI Rect[0] Rect[1] Rect[2] Rect[3]" file.pdf
```

Querying all keys in a dictionary object. Using the »xx« special key you can list all keys which are contained in a dictionary without having to know in advance the name of the keys.

The following command lists all entries in the PDFlib block dictionaries (generally this will be all required entries and those with a non-default value, since the PDFlib Block plugin omits properties which have their default value):

```
pcos --format "(KEY=VAL\n)\n" --extended block xx file.pdf
```

The following command lists all entries in all font dictionaries:

```
pcos --bracket round --format "(KEY=VAL\n)\n" --extended font xx file.pdf
```

1.4 For Experts: raw pCOS Paths

The following command prints the total number of fonts in the document; using the pCOS paths *length:bookmarks*, *length:pages*, or *length:fields* you can check the number of bookmarks, pages, or form fields, respectively:

```
pcos --pcospath "length:fonts" file.pdf
```

The following command extracts an embedded Distiller job options file:

```
pcos --outfile embedded.joboptions --pcospath "names/EmbeddedFiles[0]/EF/F" file.pdf
```

The following command dumps information about the version of PDFlib blocks on the first page, and the version of the Block plugin used to create the blocks:

```
pcos --format "PP=VAL\n" --pcospath "pages[0]/PieceInfo/PDFlib/Private/Version"  
      --pcospath "pages[0]/PieceInfo/PDFlib/Private/PluginVersion" file.pdf
```

The following command prints the number of annotations on the first page:

```
pcos --pcospath "length:pages[0]/Annots" file.pdf
```

The following command extracts the first file attachment on the first page (see Section 1.5, »For Programmers: pCOS Library Calls«, page 16, for determining the total number of file attachments on all pages):

```
pcos --outfile attachment.txt --pcospath "pages[0]/Annots[0]/FS/EF" file.pdf
```

1.5 For Programmers: pCOS Library Calls

The pCOS Cookbook. The pCOS Cookbook, available on the Web, is a collection of programming examples which demonstrate how to write PDF querying applications based on the pCOS programming interface. The Cookbook contains stand-alone Java programming examples which can be used as a starting point for your own programming. Since the pCOS API is identical for all language bindings the basic logic can be applied to other programming languages as well. The following is a partial list of programming samples for which full source code is available in the pCOS Cookbook:

- ▶ retrieve all annotations, articles, attachments, bookmarks, form fields, named destinations, etc.
- ▶ create a list of layer names
- ▶ print information about font, images, or colorspace in the document
- ▶ retrieve page size, separation names, page labels
- ▶ retrieve XMP metadata or XFA form data
- ▶ query PDF/X or PDF/A status
- ▶ list digital signatures
- ▶ extract output intent ICC profiles, embedded files

It is strongly recommended to browse the pCOS Cookbook on the Web or download the full pCOS Cookbook package from the following location:

www.pdflib.com/pcos-cookbook

Simple programming examples. In the following code fragments we focus on the crucial path processing. Standard programming items, such as try/catch handling and document open/close calls are not included in the samples. See the pCOS distribution and the pCOS Cookbook for complete samples which contain the general pCOS programming framework in various programming languages.

Assuming a valid pCOS object (called *p* in the samples below) and PDF document handle (called *doc*) are available, the pCOS functions *pcos_get_number()*, *pcos_get_string()*, and *pcos_get_stream()* can be used to retrieve information from a PDF using the pCOS path syntax. Table 1.1 lists some common pCOS paths and their meaning (a numerical array index is indicated by ...).

Table 1.1 pCOS paths for commonly used PDF objects

pCOS path	type	explanation
length:pages	number	number of pages
encrypt/description	string	encryption algorithm
/Info/Title	string	document info field Title
fields[...]	array	all form fields
/Root/Metadata	stream	XMP stream with the document's metadata
fonts[...]/name	string	name of a font; the number of entries can be retrieved with length:fonts
fonts[...]/embedded	boolean	embedding status of a font
pages[...]/width	number	width of the visible area of the page

Number of pages. The following fragment queries the total number of pages:

```
pagecount = (int) p.pCOS_get_number(doc, "length:pages");
```

Document info fields. The following fragment retrieves the *Title* document information entry:

```
String objtype = p.pCOS_get_string(doc, "type:/Info/Title");

if (objtype.equals("string"))
{
    /* Document info key found */
    System.out.println(p.pCOS_get_string(doc, "/Info/Title"));
}
```

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

```
double width = p.pCOS_get_number(doc, "pages[" + 2 + "]/width");
double height = p.pCOS_get_number(doc, "pages[" + 2 + "]/height");
```

Retrieve XMP metadata. The following fragments checks for the existence of document-level metadata, and fetches the XMP stream contents if available:

```
String objtype = p.pCOS_get_string(doc, "type:/Root/Metadata");
if (objtype.equals("stream"))
{
    /* XMP meta data found */
    byte[] metadata = p.pCOS_get_stream(doc, "", "/Root/Metadata");
}
```


2 pCOS Command-Line Reference

2.1 Option Processing and Exit Codes

The pCOS program can be controlled via a number of command-line options. It is called as follows for one or more input PDF files:

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

Constructing pCOS command lines. The following rules must be observed for constructing pCOS 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 (e.g. *--last* instead of *--lastpage*)
- ▶ Depending on encryption status of the input file, a user or master password may be required. This can be supplied with the *--password* option. pCOS will check whether this password is sufficient for the requested operation.

pCOS 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 pCOS. 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.

Exit codes. The pCOS 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 (parser warning): the parser detected a problem in the command-line options, but continued after issuing a warning (e.g. wrong verbosity number)
- ▶ Exit code 2 (parser error): the parser detected a fatal problem in the command-line options, and stopped.
- ▶ Exit code 3: a warning was issued while processing the input, but processing continues.
- ▶ Exit code 4: an error was found while processing the input, processing stopped.

Encrypted PDF. All objects can be queried if the proper master password has been supplied with the *--password* option. If no password or only the user password has been supplied some objects are available, while others are not. See Section 3.6, »Encrypted PDF Documents«, page 41 for details on PDF security and pCOS modes.

2.2 Input Options

Table 2.1 lists options related to the input or general processing.

Table 2.1 *pCOS* command-line options related to input or general processing

<i>option</i>	<i>parameters</i>	<i>function</i>
--docopt	<option list>	Additional option list for <code>pCOS_open_document()</code> (see Table 5.1, page 56)
--firstpage	1, 2, ..., last	The number of the page where page-related processing will start. The keyword <code>last</code> can be used to specify the last page. Default: 1
--inmemory		Load the input file(s) into memory and process it from there. This can result in a significant performance gain on some systems at the expense of memory usage.
--lastpage	1, 2, ..., last	The number of the page where page-related processing will finish. The keyword <code>last</code> can be used to specify the last page. Default: last
--password, -p	<password>	User or master password for encrypted documents
--pcosopt	<option list>	Additional option list for <code>pCOS_set_option()</code> (see Table 5.2, page 60). This can be used to pass the license or licensefile options.

2.3 Options for Retrieving PDF Elements

Table 2.2 lists options for simple output retrieval (there are no short option forms nor parameters in this group). Multiple retrieval options can be provided in a single call. In this case output will be created in the following order: first, the `--general` and `--docinfo` options will be processed (if supplied), and then all other retrieval options in Table 2.2 and Table 2.3 in the order in which they have been specified on the command line. If no retrieval option has been provided, the default `--general --docinfo` will be used.

All options in Table 2.2 except `--general` require full pCOS mode, i.e. the master password must be provided for encrypted files.

Table 2.2 pCOS command-line options for simple output retrieval

option	function
<code>--annotation¹</code>	Contents and type of annotations. This option queries the keys Contents and Subtype in pages[...]/annots for all pages, using the format PP/KEY: VAL\n.
<code>--attachment¹</code>	Description and file name of file attachments on the pages (see also <code>--embeddedfile</code>). This option queries the keys Contents, FS/F, and FS/UF in pages[...]/annots for all pages (if FS is present), using the format PP/KEY: VAL\n. The actual contents of a file attachment can be retrieved via <code>--extract attachment</code> .
<code>--block¹</code>	Name and subtype of PDFlib blocks. This option queries the keys Name and Subtype in pages[...]/PieceInfo/PDFlib/Private/Blocks for all pages, using the format KEY: VAL\n.
<code>--bookmark</code>	Names of bookmarks. This option queries the key Title in bookmarks[...], using the format »VAL\n«, and bookmarks[...]/level for indentation. The target page of a bookmark can be retrieved via bookmarks[...]/destpage.
<code>--destination</code>	Names and destination pages of named destinations. This option queries all keys in names[...]/Dest (i.e. all named destinations) and the value of the destpage subkey, using the format PP/KEY: VAL\n.
<code>--docinfo</code>	Key and value of document info entries. This option queries all keys in /Info, using the format KEY: VAL\n.
<code>--embedded-file</code>	File name and description of named embedded files. This option queries document-level file attachments, while <code>--attachment</code> will retrieve file attachments on the page level. This option queries the keys F, UF, and Desc in names/EmbeddedFiles/*, using the format PP/KEY: VAL\n. The actual contents of an embedded file can be retrieved via <code>--extract embeddedfile</code> .
<code>--field</code>	Names, types, and values of form fields. This option queries the keys fullname, FT, and V in fields[...], using the format PP/KEY: VAL\n, and fields[...]/level for indentation.
<code>--font</code>	Names, types, and embedding status of fonts. This option queries the keys name, type, and embedded in fonts[...], using the format PP/KEY: VAL\n.
<code>--general</code>	File name and size, PDF version, encryption status, master/user password, linearization status, PDF/X, PDF/A, XFA, tagged status, signature details, Reader-enabled status, PDF package (portable collection) status, number of pages, number of fonts (page and font count are only available in full pCOS mode). This option queries various real and pseudo objects.

Table 2.2 pCOS command-line options for simple output retrieval

option	function
--javascript	<p>JavaScript at various locations in the document. For each script its length (in Unicode characters) will be printed, as well as the total number of scripts found. Depending on the location of the JavaScript in the document, additional information will be printed:</p> <p>Document open actions: JavaScript which will activated when the document is opened.</p> <p>Bookmarks: JavaScript for bookmark activation.</p> <p>Document-level JavaScript: additional information for the trigger event (didprint, didsave, willclose, willprint, willsave)</p> <p>Page-level JavaScript: additional information for the trigger event (open, close)</p> <p>JavaScript for annotation activation. Additional information: page number, annotation type</p> <p>Field-level JavaScript. Additional information: form field name, trigger (activate, keystroke, format, validate, calculate, enter, exit, down, up, focus, blur)</p>
--layer	<p>Names of all layers in the document. This may include unused layers and layers which are not visible in Acrobat's user interface (e.g. layers which do not require any interaction because they are controlled by JavaScript). This option queries the key Name in /Root/OCProperties/OCGs, using the format VAL\n.</p>
--layer-default	<p>Names of layers which are presented by default in Acrobat's layer pane (not related to the visibility of layer contents on the page). Only layers which are presented to the user will be shown, using indentation to visualize the layer hierarchy. Text labels for grouping (which do not directly resemble a layer) will also be printed. Use --layer to catch all layers, regardless of their presence in the user interface. This option queries the key Name in /Root/OCProperties/D/Order, using the format VAL\n.</p>
--outputintent	<p>Properties of one or more output intent ICC profiles, mostly used for PDF/X and PDF/A documents. This option queries various keys in the /Root/OutputIntents[...] dictionary, using the format PP/KEY: VAL\n.</p>
--pagesize¹	<p>Width, height, and various boxes describing the page dimensions. This option queries the keys width, height, MediaBox, CropBox, and Rotate in pages[...] for all pages, using the format PP/KEY: VAL\n.</p>
--pdfa	<p>PDF/A version and output intent name (no validation). This option queries the part, conformance, and amd (amendment) keys in the pdfaid section of the document's XMP metadata (/Root/Metadata) if present. If the file conforms to any of the PDF/A-1 standards, the corresponding keys /Root/OutputIntents[...] /OutputConditionIdentifier and /Root/OutputIntents[...] /Info are queried as well.</p>
--pdfx	<p>PDF/X version and output intent name (no validation). This option first queries the key /Info/GTS_PDFXVersion. If the file conforms to any of the PDF/X standards, the corresponding keys /Root/OutputIntents[...] /OutputConditionIdentifier and /Root/OutputIntents[...] /Info are queried as well.</p>
--signature	<p>Signature information: name and visibility of all signature fields, signed/unsigned status, and signature details for signed fields. This option queries the key fullname and various entries in the V dictionary in fields[...] (if FT=Sig).</p>
--weblink¹	<p>Contents and URL of web links. This option queries the keys Contents and A/URI in pages[...] /annots for all pages (if A/URI is present), using the format PP/KEY: VAL\n.</p>
--xfa	<p>Checks whether the documents contains any XFA information (eXtensible Forms Architecture). This option queries the key /Root/AcroForm/XFA.</p>

1. This option is subject to the --firstpage and --lastpage options.

2.4 Advanced Retrieval Options

Table 2.3 lists options for advanced output retrieval. If pCOS runs in minimum or restricted mode, i.e. the master password has not been provided for an encrypted file, not all objects may be available (see Section 3.6, »Encrypted PDF Documents«, page 41 for details). If the path designates a simple object, its value will be printed, dictionary objects will be enumerated recursively up to the level specified with `--depth`, and array objects will be completely enumerated recursively.

Table 2.3 pCOS command-line options for advanced output retrieval

long option	parameters	function
<code>--binary</code>		Retrieved string objects will be treated as binary data, i.e. will not be subject to Unicode and EBCDIC conversions. This option is useful for binary string data, e.g. Contents of a signature dictionary; it is not required for stream data which is always treated in binary mode.
<code>--extended¹</code>	<code><type> <keys></code>	Extended object retrieval for one of the following types: annotation, attachment, block, bookmark, destination, docinfo, font, layer, pagesize, signature, weblink <code><keys></code> contains a list of keys to be retrieved from the respective object(s). Use <code>xx</code> to query all existing keys (excluding pseudo keys if they exist for an object, e.g. a font dictionary, and some low-level bookkeeping keys for maintaining tree structures). The list of keys must be provided as a single command-line argument (in some environments this requires surrounding double quotes).
<code>--extract¹</code>	<code><type></code>	Extract the binary data associated with one of the following types and print general information about the items: attachment All file attachments on page level (takes into account the <code>--firstpage</code> and <code>--lastpage</code> options) embeddedfile All file attachments on document level javascript All JavaScripts for document open action, bookmarks, document-level scripts, page-level scripts, annotation activation, and fields. metadata XMP document metadata (without any format conversion) outputintent All output intent ICC profiles signature All certificate values, i.e. the Contents entry of signature fields. It contains a PKCS#1 (rare) or PKCS#7 object (common). Each data item will be written to a separate disk file. Starting at the directory specified with the <code>--targetdir</code> option, a directory will be created using the name of the input PDF (without any <code>.pdf</code> or <code>.PDF</code> suffix, and with critical characters replaced with <code>"_"</code>). Within this directory various subdirectories for the data items will be created. The <code>--outfile</code> option will be ignored. In addition to the generated data files a description of all extracted data items will be created on standard output.

Table 2.3 pCOS command-line options for advanced output retrieval

long option	parameters	function
--format -f	<string>	(Affects only --extended and --pcospath) Output format for recursion level o. Expressions within (...) will iterate over all existing keys. Format examples can be found in Table 2.2. The following placeholders can be used in addition to regular characters: IF input file name PP pCOS path of the object KEY name of the object VAL value of the object \n carriage return plus linefeed on Windows; single linefeed on all other systems \r carriage return \t horizontal tab Default: PP/KEY: VAL\n for --extended, VAL\n for --pcospath
--pcospath ¹	<path>...	pCOS path of an object that will be queried. Examples for object paths can be found in Table 2.2, and a full description in Chapter 3, »pCOS Path Reference«, page 29.

1. This option can be supplied more than once.

2.5 Output Options

Table 2.4 lists options for controlling details of the generated output.

Table 2.4 pCOS command-line options for controlling output details

option	parameters	function
--bracket -b	<keyword>	Bracketing of strings, arrays, names, dictionaries, and empty values (default: none): none no brackets angle < > curly {} round () squared [] dquot " " squot ' '
--depth -d	1, 2, ...	Recursion depth for resolving dictionaries. For higher recursion levels the string supplied with --replace dictionary will be printed. Default: 2
--headline -h	<string>	Header line for each file. The following placeholders can be used in addition to regular characters (default: no header when a single file is processed, and \nIF:\n when multiple files are processed): IF input file name OF output file name \n carriage return plus linefeed on Windows; single linefeed on all other systems \r carriage return \t horizontal tab
--help -?		Display help with a summary of available options.
--indent	0, 1, 2, ...	Indentation for hierarchical output of --bookmark, --field, and --layerdefault. Default: 3 (use --indent 0 for creating tabular output)
--outfile -o	<filename>	Output file name (will be ignored for --extract). The following special names are recognized (default: -): - standard output + base name of the input file with .pdf replaced with .txt
--replace ¹ -r	<keyword> <string>	Replacement strings. The following keywords are supported: missing String for non-existing objects. Default: <not found> dictionary String for unresolved dictionaries. Default: <dictionary> control Replacement of control characters (U+0000-U+001F and U+007F-U+009F). A C-style formatting expression (e.g. %03o) will be replaced with the formatted value of the character. The replacement will be performed in textual and stream data. Default: no replacement
--separator -s	<string>	Separator string between keys and values of type dictionary for recursion levels 1 and above. Default: =
--targetdir -t	<dirname>	Output directory name; the directory must exist. Default: .
--utf16 -u		(Ignored when writing to standard output) Convert the output to UTF-16 with BOM. Without this option the text will be output in UTF-8 format, and stream contents will be output without any modification.

Table 2.4 pCOS command-line options for controlling output details

option	parameters	function
--verbose	0, 1, 2, 3	Verbosity level (default: 1):
-v		0 no output at all
		1 emit only warnings, errors, and banner
		2 like 2, but also emit file names
		3 detailed reporting

1. This option can be supplied more than once.

2.6 Unicode Output and Binary Data

Conversion rules. Subject to the PDF objects retrieved, the output created by pCOS can be plain ASCII text (e.g. most font names), Unicode text (e.g. Japanese document info entries, or binary data (e.g. ICC profiles). pCOS creates output according to the following rules:

- ▶ Name and string objects will be output in UTF-8 without BOM. This means that ASCII text will result in plain ASCII output, but Latin-1 special characters (e.g. umlauts or accented characters) will result in two-byte UTF-8 sequences. Users must be prepared for UTF-8 output, and must convert to other formats (e.g. WinAnsi) if required. Lines will be terminated with `\r\n` (carriage return plus linefeed) on Windows, and with `\n` (single linefeed) on all other systems.
- ▶ If the `--utf16` option has been supplied and the output channel is not *stdout* the complete output will be converted from UTF-8 to native UTF-16 with BOM (byte order mark). This only makes sense if all output items are UTF-8 (without any binary stream objects). pCOS emits a warning at the end of the output for some critical combinations, or if the output couldn't be converted from UTF-8 to UTF-16 (the most likely reason for this is that binary stream data was included in the output).
- ▶ Stream objects will be output in binary format without any modification. This includes XMP metadata streams, but these are usually stored in the PDF as UTF-8 anyway.

3 pCOS Path Reference

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

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

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

3.1 Simple pCOS Examples

Note More pCOS programming examples can be found on the PDFlib Web site and in the respective chapters of the pCOS product manual.

Assuming a valid PDF document handle is available, the pCOS functions *pCOS_get_number()*, *pCOS_get_string()*, and *pCOS_get_stream()* can be used to retrieve information from a PDF using the pCOS path syntax. Table 3.1 lists some common pCOS paths and their meaning.

Table 3.1 pCOS paths for commonly used PDF objects

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

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

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

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

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

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

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

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

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

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

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

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

Text extraction status. By default, content extraction is possible with TET if the document can successfully be opened. However, with *infomode=true* this is not necessarily true. 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 expression can be used to check whether extraction is allowed:

```
if ((int) p.pcos_get_number(doc, "pcosmode") == 2 ||
    ((int) p.pcos_get_number(doc, "pcosmode") == 1 &&
     (int) p.pcos_get_number(doc, "encrypt/nocopy") == 0))
{
    /* text extraction allowed */
}
```

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

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

3.2 Handling Basic PDF Data Types

pCOS offers the three functions *pCOS_get_number()*, *pCOS_get_string()*, and *pCOS_get_stream()*. These can be used to retrieve all basic data types which may appear in PDF documents.

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

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

- ▶ Name objects will be undecorated (i.e. the # syntax will be resolved) before they are returned.
- ▶ Name objects will be returned as Unicode strings in most language bindings. However, in the C and C++ language bindings they will be returned as UTF-8.

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

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

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

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

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

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

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

3.3 Composite Data Structures and IDs

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

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

The contents of an array can be enumerated by querying the number *N* of elements it contains (using the *length* prefix in front of the array's path, see Table 3.2), and then iterating over all elements from index 0 to *N*-1.

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

The contents of a dictionary can be enumerated by querying the number *N* of elements it contains (using the *length* prefix in front of the dictionary's path, see Table 3.2), and then iterating over all elements from index 0 to *N*-1. Enumerating dictionaries will provide all dictionary keys in the order in which they are stored in the PDF using the *.key* suffix at the end of the dictionary's path. Similarly, the corresponding values can be enumerated with the *.val* suffix. Inherited values (see below) and pseudo objects will not be visible when enumerating dictionary keys, and will not be included in the *length* count.

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

pCOS IDs for dictionaries and arrays. Unlike PDF object IDs, pCOS IDs are guaranteed to provide a unique identifier for an element addressed via a pCOS path (since arrays and dictionaries can be nested an object can have the same PDF object ID as its parent array or dictionary). pCOS IDs can be retrieved with the *pcosid* prefix in front of the dictionary's or array's path (see Table 3.2).

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

3.4 Path Syntax

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

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

The meaning of the various path components is as follows:

- ▶ The optional *prefix* can attain the values listed in Table 3.2.
- ▶ The optional *pseudo object name* may contain one of the values described in Section 3.5, »Pseudo Objects«, page 35.
- ▶ The *name* components are dictionary keys found in the document. Multiple names are separated with a / character. An empty path, i.e. a single / denotes the document's Trailer dictionary. Each name must be a dictionary key present in the preceding dictionary. Full paths describe the chain of dictionary keys from the initial dictionary (which may be the Trailer or a pseudo object) to the target object.
- ▶ Paths or path components specifying an array or dictionary can have a numerical index which must be specified in decimal format between brackets. Nested arrays or dictionaries can be addressed with multiple index entries. The first entry in an array or dictionary has index 0.
- ▶ Paths or path components specifying a dictionary can have an index qualifier plus one of the suffixes *.key* or *.val*. This can be used to retrieve a particular dictionary key or the corresponding value of the indexed dictionary entry, respectively. If a path for a dictionary has an index qualifier it must be followed by one of these suffixes.

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

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

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

Path prefixes. Prefixes can be used to query various attributes of an object (as opposed to its actual value). Table 3.2 lists all supported prefixes.

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

Table 3.2 pCOS path prefixes

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

3.5 Pseudo Objects

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

Universal pseudo objects. Universal pseudo objects are always available, regardless of encryption and passwords. This assumes that a valid document handle is available, which may require setting the option *requiredmode* suitably when opening the document. Table 3.3 lists all universal pseudo objects.

Table 3.3 Universal pseudo objects

object name	explanation
encrypt	(Dict) Dictionary with keys describing the encryption status of the document:
length	(Number) Length of the encryption key in bits
algorithm	(Number)
description	(String) Encryption algorithm number or description:
-1	Unknown encryption
0	No encryption
1	40-bit RC4 (Acrobat 2-4)
2	128-bit RC4 (Acrobat 5)
3	128-bit RC4 (Acrobat 6)
4	128-bit AES (Acrobat 7)
5	Public key on top of 128-bit RC4 (Acrobat 5) (unsupported)
6	Public key on top of 128-bit AES (Acrobat 7) (unsupported)
7	Adobe Policy Server (Acrobat 7) (unsupported)
8	Adobe Digital Editions (EBX) (unsupported)
master	(Boolean) True if the PDF requires a master password to change security settings (permissions, user or master password), false otherwise
user	(Boolean) True if the PDF requires a user password for opening, false otherwise
noaccessible, noannots, noassemble, nocopy, noforms, nohiresprint, nomodify, noprint	(Boolean) True if the respective access protection is set, false otherwise
plainmetadata	(Boolean) True if the PDF contains unencrypted meta data, false otherwise
filename	(String) Name of the PDF file.
filesize	(Number) Size of the PDF file in bytes
linearized	(Boolean) True if the PDF document is linearized, false otherwise
major minor revision	(Number) Major, minor, or revision number of the library, respectively.
pcosinterface	(Number) Interface number of the underlying pCOS implementation. This specification describes interface number 3. The following table details which product versions implement various pCOS interface numbers:
1	TET 2.0, 2.1
2	pCOS 1.0
3	PDFlib+PDI 7, PPS 7, TET 2.2, pCOS 2.0, PLOP 3.0

Table 3.3 Universal pseudo objects

object name	explanation	
<i>pcosmode</i>	(Number/string) pCOS mode as number or string:	
<i>pcos-</i>	0	minimum
<i>modename</i>	1	restricted
	2	full
<i>pdfversion</i>	(Number) PDF version number multiplied by 10, e.g. 16 for PDF 1.6	
<i>version</i>	(String) Full library version string in the format <major>.<minor>.<revision>, possibly suffixed with additional qualifiers such as beta, rc, etc.	

Pseudo objects for PDF objects, pages, and interactive elements. Table 3.4 lists pseudo objects which can be used for retrieving object or page information, or serve as short-cuts for various interactive elements.

Table 3.4 Pseudo objects for PDF objects, pages, and interactive elements

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

Table 3.4 Pseudo objects for PDF objects, pages, and interactive elements

object name	explanation
pages	(Array of dicts) Each array element addresses a page of the document. Indexing it with the decimal representation of the page number minus one addresses that page (the first page has index 0). Using the length prefix the number of pages in the document can be determined. A page object addressed this way will incorporate all attributes which are inherited via the /Pages tree. The entries /MediaBox and /Rotate are guaranteed to be present. In addition to standard PDF dictionary entries the following pseudo entries are available for each page: colorspaces, extgstates, fonts, images, patterns, properties, shadings, templates (Arrays of dicts) Page resources according to Table 3.5. annots (Array of dicts) In addition to the standard PDF keys in the Annots array pCOS supports the following pseudo key for dictionaries in the annots array: destpage (Number; only for Subtype=Link and if a Dest entry is present) Number of the target page (first page is 1) blocks (Array of dicts) Shorthand for pages[]/PieceInfo/PDFLib/Private/Blocks[], i.e. the page's block dictionary. In addition to the existing PDF keys pCOS supports the following pseudo key for dictionaries in the blocks array: rect (Rectangle) Similar to Rect, except that it takes into account any relevant CropBox/MediaBox and Rotate entries and normalizes coordinate ordering. height (Number) Height of the page. The MediaBox or the CropBox (if present) will be used to determine the height. Rotate entries will also be applied. isempty (Boolean) True if the page is empty, and false if the page is not empty label (String) The page label of the page (including any prefix which may be present). Labels will be displayed as in Acrobat. If no label is present (or the PageLabel dictionary is malformed), the string will contain the decimal page number. Roman numbers will be created in Acrobat's style (e.g. VL), not in classical style which is different (e.g. XLV). If /Root/PageLabels doesn't exist, the document doesn't contain any page labels. width (Number) Width of the page (same rules as for height) The following entries will be inherited: CropBox, MediaBox, Resources, Rotate.
pdfa	(String) PDF/A conformance level of the document (e.g. PDF/A-1a:2005) or none
pdfx	(String) PDF/X conformance level of the document (e.g. PDF/X-1a:2001) or none
tagged	(Boolean) True if the PDF document is tagged, false otherwise

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

- ▶ Some entries may have been added (e.g. inline images, simple color spaces) or deleted (e.g. the parts of multi-strip images).
- ▶ In addition to the original PDF dictionary keys resource dictionaries may contain some user-friendly keys for auxiliary information (e.g. embedding status of a font, number of components of a color space).

pCOS supports two groups of pseudo objects for resource retrieval. Global resource arrays contain all resources of a given type in a PDF document, while page-based resources contain only the resources used by a particular page. The corresponding pseudo arrays are available for all resource types listed in Table 3.5:

- ▶ A list of all resources in the document is available in the global resource array (e.g. `images[]`). Accessing one of the global resource pseudo arrays results in a scan over all pages' contents.
- ▶ A list of resources on each page is available in the page-based resource array (e.g. `pages[]/images[]`). Accessing one of a page's resource pseudo arrays results in a scan over that page's contents (to collect all resources which are actually used on the page).

Table 3.5 Pseudo objects for resources; each resource category *P* creates two resource arrays `P[]` and `pages[]/P[]`.

object name	explanation
colorspaces	(Array of dicts) Array containing dictionaries for all color spaces on the page or in the document. In addition to the standard PDF keys in color space and ICC profile stream dictionaries the following pseudo keys are supported:
alternateid	(Integer; only for name=Separation and DeviceN) Index of the underlying alternate color space in the <code>colorspaces[]</code> pseudo object.
alternateonly	(Boolean) If true, the colorspace is only used as the alternate color space for (one or more) Separation or DeviceN color spaces, but not directly.
baseid	(Integer; only for name=Indexed) Index of the underlying base color space in the <code>colorspaces[]</code> pseudo object.
colorantname	(Name; only for name=Separation) Name of the colorant
colorantnames	(Array of names; only for name=DeviceN) Names of the colorants
components	(Integer) Number of components of the color space
name	(String) Name of the color space
csarray	(Array; not for name=DeviceGray/RGB/CMYK) Array describing the underlying native color space.

Color space resources will include all color spaces which are referenced from any type of object, including the color spaces which do not require native PDF resources (i.e. DeviceGray, DeviceRGB, and DeviceCMYK).

Table 3.5 Pseudo objects for resources; each resource category *P* creates two resource arrays *P*[] and pages[]/P[].

object name	explanation
extgstates	(Array of dicts) Array containing the dictionaries for all extended graphics states (ExtGStates) on the page or in the document
fonts	<p>(Array of dicts) Array containing dictionaries for all fonts on the page or in the document. In addition to the standard PDF keys in font dictionaries, the following pseudo keys are supported:</p> <p>name (String) PDF name of the font without any subset prefix. Non-ASCII CJK font names will be converted to Unicode.</p> <p>embedded (Boolean) Embedding status of the font</p> <p>type (String) Font type</p> <p>vertical (Boolean) true for fonts with vertical writing mode, false otherwise</p>
images	<p>(Array of dicts) Array containing dictionaries for all images on the page or in the document. The TET product will add merged (artificial) images to the images[] array, and remove the strips comprising merged images from this array.</p> <p>In addition to the standard PDF keys the following pseudo keys are supported:</p> <p>bpc (Integer) The number of bits per component. This entry is usually the same as BitsPerComponent, but unlike this it is guaranteed to be available.</p> <p>colorspaceid (Integer) Index of the image's color space in the colorspaces[] pseudo object. This can be used to retrieve detailed color space properties.</p> <p>filterinfo (Dict) Describes the remaining filter for streams with unsupported filters or when retrieving stream data with the keepfilter option set to true. If there is no such filter no filterinfo dictionary will be available. The dictionary contains the following entries:</p> <p>name (Name) Name of the filter</p> <p>supported (Boolean) True if the filter is supported</p> <p>decodeparms (Dict) The DecodeParms dictionary if one is present for the filter</p> <p>maskid (Integer) Index of the image's mask in the images[] pseudo object if the image is masked, otherwise -1</p> <p>maskonly (Boolean) If true, the image is only used as a mask for (one or more) other images, but not directly</p>
patterns	(Array of dicts) Array containing dictionaries for all patterns on the page or in the document
properties	(Array of dicts) Array containing dictionaries for all properties on the page or in the document
shadings	<p>(Array of dicts) Array containing dictionaries for all shadings on the page or in the document. In addition to the standard PDF keys in shading dictionaries the following pseudo key is supported:</p> <p>colorspaceid (Integer) Index of the underlying color space in the colorspaces[] pseudo object.</p>
templates	(Array of dicts) Array containing dictionaries for all templates (Form XObjects) on the page or in the document

3.6 Encrypted PDF Documents

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

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

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

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

Minimum pCOS mode (mode 0). Regardless of the encryption status and the availability of passwords, the universal pCOS pseudo objects listed in Table 3.3 are always available. For example, the *encrypt* pseudo object can be used to query a document's encryption status. Encrypted objects can not be retrieved in minimum pCOS mode.

Table 3.6 lists the resulting pCOS modes for various password combinations. Depending on the document's encryption status and the password supplied when opening the file, PDF object paths may be available in minimum, restricted, or full pCOS mode. Trying to retrieve a pCOS path which is inappropriate for the respective mode will raise an exception.

Table 3.6 Resulting pCOS modes for various password combinations

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

4 pCOS Library Language Bindings

This chapter discusses specifics for the language bindings which are supplied for pCOS. The pCOS distribution contains sample code for all supported language bindings.

4.1 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 pCOS 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
- ▶ Trying to open a PDF document with a wrong file name
- ▶ Trying to open a PDF document with damaged file structure.

pCOS 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;

When pCOS 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 pCOS supplies a custom exception handling mechanism which must be used by clients (see Section 4.2, »C Binding«, page 44).

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 *pCOS_delete()*, *pCOS_get_apiname()*, *pCOS_get_errnum()*, and *pCOS_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 pCOS function calls, e.g. *pCOS_open_document()* or *pCOS_open_page()*, can fail without throwing an exception (they will return -1 in case of an error). In this situation the functions *pCOS_get_errnum()*, *pCOS_get_errmsg()*, and *pCOS_get_apiname()* can be called immediately after a failed function call in order to retrieve details about the nature of the problem.

4.2 C Binding

Linking a Program against the pCOS Library. In order to use the library within your C program, you must include the pCOS header file *pcoslib.h* in your source code, and must link your executable against the library:

- ▶ On Unix systems this is achieved by using a command similar to the following (assuming the library *libpcos.a* and the header file *pcoslib.h* are available in the current directory):

```
cc dumper.c -o dumper -I. -L. -lpcos -lm
```

- ▶ On Windows systems pCOS is delivered as a DLL. The DLL should be placed in the Windows system directory. Microsoft Visual C++ project files for the C samples are contained in the pCOS distribution. You must link your executable against the following library:

```
libpcos.lib
```

Exception handling in C. The pCOS 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 *PCOS_TRY()* and *PCOS_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 pCOS client code:

- ▶ *PCOS_TRY()* and *PCOS_CATCH()* must always be paired.
- ▶ *PCOS_new()* will never throw an exception; since a try block can only be started with a valid pCOS object handle, *PCOS_new()* must be called outside of any try block.
- ▶ *PCOS_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 *PCOS_CATCH()*), the *PCOS_EXIT_TRY()* macro must be called before the return statement to inform the exception machinery.
- ▶ As in all pCOS 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 pCOS exceptions in client code (a full sample can be found in the pCOS package):

```
volatile int n_pages, pageno;
...
if ((p = pCOS_new()) == (pCOS *) 0)
```

```

{
    printf("out of memory\n");
    return(2);
}
pCOS_TRY(p)
{
    n_pages = (int) pCOS_get_number(p, doc, "length:pages");
    for (pageno = 1; pageno <= n_pages; ++pageno)
    {
        /* process page */

        if (/* error happened */)
        {
            pCOS_EXIT_TRY(p);
            return -1;
        }
    }
    /* statements that directly or indirectly call API functions */
}
pCOS_CATCH(p)
{
    printf("Error %d in %s() on page %d: %s\n",
        pCOS_get_errnum(p), pCOS_get_apiname(p), pageno, pCOS_get_errmsg(p));
}
pCOS_delete(p);

```

Unicode handling for name strings. The C language does not natively support Unicode. Some string parameters for API functions may be declared as *name strings*. These are handled depending on the *length* parameter and the existence of a BOM at the beginning of the string. In C, if the *length* parameter is different from 0 the string will be interpreted as UTF-16. If the *length* parameter is 0 the string will be interpreted as UTF-8 if it starts with a UTF-8 BOM, or as EBCDIC UTF-8 if it starts with an EBCDIC UTF-8 BOM, or as *host* encoding if no BOM is found (or *ebcdic* on all EBCDIC-based platforms).

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

- ▶ If the option starts with a UTF-8 BOM (`\xEF\xBB\xBF`) it will be interpreted as UTF-8.
- ▶ If the option starts with an EBCDIC UTF-8 BOM (`\x57\x8B\xAB`) it will be interpreted as EBCDIC UTF-8.
- ▶ If no BOM is found, the string will be treated as *winansi* (or *ebcdic* on EBCDIC-based platforms).

Note The `pCOS_utf16_to_utf8()` utility function can be used to create UTF-8 strings from UTF-16 strings, which is useful for creating option lists with Unicode values.

4.3 C++ Binding

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

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

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

4.4 COM Binding

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

- ▶ If you install on an NTFS partition all pCOS users must have read permission to the installation directory, and execute permission to
...\\pCOS 2.0\\COM\\bin\\pCOS_com.dll.
- ▶ The installer must have write permission to the system registry. Administrator or Power Users group privileges will usually be sufficient.

Exception Handling. Exception handling for the pCOS COM component is done according to COM conventions: when a pCOS 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 pCOS object is released. The COM exception can be caught and handled in the pCOS client in whichever way the client environment supports for handling COM errors.

Using the pCOS COM Edition with .NET. As an alternative to the pCOS.NET edition (see Section 4.6, »*.NET Binding*«, page 49) the COM edition of pCOS can also be used with .NET. First, you must create a .NET assembly from the pCOS COM edition using the *tlbimp.exe* utility:

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

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

```
using pCOS_com;
...
static pCOS_com.IpCOS p;
...
p = New pCOS();
...
```

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

4.5 Java Binding

Installing the pCOS Java edition. pCOS is organized as a Java package with the name *com.pdflib.pCOS*. 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 *libpcos_java.so* (on Mac OS X: *libpcos_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_pcos.dll* must be placed in the Windows system directory, or a directory which is listed in the PATH environment variable.

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

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

You can check the value of this property as follows:

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

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

```
pCOS p = null;

try {

...pCOS method invocations...

} catch (pCOSException e) {
    System.err.print("pCOS 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 (p != null) {
        p.delete();                /* delete the pCOS object */
    }
}
```

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

4.6 .NET Binding

The Microsoft .NET architecture supports a variety of programming languages based on the Common Language Runtime (CLR) which provides a common environment for executing programs.

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

Note pCOS.NET requires the .NET framework 1.1 or above.

pCOS.NET can be deployed in all environments that support the .NET Framework 1.1 or above. The pCOS distribution package contains code samples for various .NET languages.

The pCOS MSI installer will install the pCOS assembly plus auxiliary data files, documentation and samples on the machine interactively. The installer will also register pCOS so that it can easily be referenced on the .NET tab in the *Add Reference* dialog box of Visual Studio .NET.

Exception handling. pCOS.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.

Note In order to convey exception-related information pCOS defines its own exception class *pCOS_dotnet.pCOSException* with the members *get_errnum*, *get_errmsg*, and *get_apiname*.

4.7 Perl Binding

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

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

```
perl -I/path/to/pcoslib dumper.pl
```

Unix: Perl will search both *pcoslib_pl.so* (the *.so* ending may vary on other platforms) and *pcoslib_pl.pm* in the current directory, or the directory printed by the following Perl command:

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

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

```
/usr/lib/perl5/site_perl/5.8/i686-linux
```

Windows: pCOS supports the ActiveState port of Perl 5 to Windows, also known as ActivePerl. Both *pcoslib_pl.dll* and *pcoslib_pl.pm* will be searched in the current directory, or the directory printed by the following Perl command:

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

Typical output of the above command looks like

```
C:\Program Files\Perl5.8\site\lib
```

Exception handling. When a pCOS exception occurs, a Perl exception is thrown. It can be caught and acted upon using an *eval* sequence.

4.8 PHP Binding

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

Detailed information about the various flavors and options for using pCOS with PHP, including the question of whether or not to use a loadable pCOS module for PHP, can be found in the *PDFlib-in-PHP-HowTo* document which can be found on the PDFlib Web site. Although it is mainly targeted at using PDFlib with PHP the discussion applies equally to using pCOS with PHP.

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

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

```
extension=libpcos_php.so      ; for Unix
extension=libpcos_php.dylib   ; for Mac OS X
extension=libpcos_php.dll     ; for Windows
```

PHP will search the library in the directory specified in the *extension_dir* variable in *php.ini* on Unix, and in the standard system directories on Windows. You can test which version of the PHP pCOS 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 *pCOS*. If this section contains the phrase

```
PDFlib pCOS: PDF Information Retrieval Tool => enabled
```

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

- Load pCOS at runtime with one of the following lines at the start of your script:

```
dl("libpcos_php.so");      # for Unix
dl("libpcos_php.dll");     # for Windows
```

PHP 5 features. pCOS takes advantage of the following features in PHP 5:

- New object model: the pCOS functions are encapsulated within a pCOS object.
- Exceptions: pCOS exceptions will be propagated as PHP 5 exceptions, and can be caught with the usual try/catch technique. New-style exception handling can be used with both the new object-oriented approach and the old API functions.

See below for more details on these PHP 5 features.

File name handling in PHP. Unqualified file names (without any path component) and relative file names for PDF, image, font and other disk files 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.

Error handling in PHP 4. When a pCOS exception occurs, a PHP exception is thrown. Since PHP 4 does not support structured exception handling there is no way to catch exceptions and act appropriately. Do not disable PHP warnings when using pCOS, or you will run into serious trouble.

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

```
try {  
  
    ...some pCOS instructions...  
  
} catch (pCOSException $e) {  
    print "pCOS exception occurred:\n";  
    print "[" . $e->get_errnum() . " ] " . $e->get_apiname() . ": "  
        $e->get_errmsg() . "\n";  
}  
catch (Exception $e) {  
    print $e;  
}
```

Note that you can use PHP 5-style exception handling regardless of whether you work with the old function-based pCOS interface or the new object-oriented one.

5 pCOS Library API Reference

5.1 Option Lists

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

```
name value(s)
```

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

```
name=value
```

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

- ▶ Boolean: *true* or *false*; if the value of a boolean option is omitted, the value *true* is assumed. As a shorthand notation *noname* can be used instead of *name false*.
- ▶ String: strings containing whitespace or '=' characters must be bracketed with { and }. An empty string can be constructed with {}. The characters {, }, and \ must be preceded by an additional \ character if they are supposed to be part of the string.
- ▶ Keyword: one of a predefined list of fixed keywords
- ▶ Float and integer: decimal floating point or integer numbers; point and comma can be used as decimal separators for floating point values. Integer values can start with x, X, ox, or oX to specify hexadecimal values. Some options (this is stated in the respective function description) support percentages by adding a % character directly after the value.
- ▶ Handle: several internal object handles, e.g., document or page handles. Technically these are integer values.

Depending on the type and interpretation of an option additional restrictions may apply. For example, integer or float options may be restricted to a certain range of values; handles must be valid for the corresponding type of object, etc. Conditions for options are documented in their respective function descriptions. Some examples for simple values (the first line shows a password string containing a blank character):

```
password={secret string}  
repair=auto
```

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

```
searchpath={/usr/lib/pcos d:\\pcos}
```

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

5.2 General Functions

<i>Perl PHP</i>	<i>resource pCOS_new()</i>
<i>C</i>	<i>pCOS *pCOS_new(void)</i>
	Create a new pCOS context.
<i>Returns</i>	A handle to the new context, or NULL if not enough memory is available. The context must be supplied to all other API functions.
<i>Bindings</i>	Not available in object-oriented language bindings where it will be called automatically when a new pCOS object is created.
<i>Java</i>	<i>void delete()</i>
<i>C#</i>	<i>void Dispose()</i>
<i>Perl PHP</i>	<i>pCOS_delete(resource p)</i>
<i>C</i>	<i>void pCOS_delete(pCOS *p)</i>
	Delete a pCOS context and release all related internal resources.
<i>Details</i>	All open documents in the context are closed automatically. It is good programming practice, however, to close documents explicitly with <i>pCOS_close_document()</i> when they are no longer needed. The pCOS object must no longer be used after this function has been called.
<i>Bindings</i>	<p>In C this function must not be called within a <i>pCOS_TRY()/pCOS_CATCH()</i> clause.</p> <p>In Java this method will be called by the finalizer method of pCOS. However, it is strongly recommended to explicitly call <i>delete()</i> for proper cleanup. The same holds true when an exception occurred.</p> <p>In PHP and COM this function will be called automatically when the pCOS object is destroyed.</p> <p>In .NET <i>Dispose()</i> should be called at the end of processing to clean up unmanaged resources.</p>

5.3 Document Functions

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

Open a PDF document.

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

In 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 (C language binding only) 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 5.1.

Returns -1 (in PHP: 0) on error, or a document handle otherwise. After an error it is recommended to call *pCOS_get_errmsg()* to find out more details about the error.

Details If the document is encrypted its user or master password must be supplied in the *password* option unless the *requiredmode* option has been specified.

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

C++	<code>int open_document_mem(const char *data, long size, string optlist)</code>
C# Java	<code>int open_document_mem(byte[] data, String optlist)</code>
Perl PHP	<code>long pCOS_open_document_mem(resource p, string data, string optlist)</code>
VB	<code>Function open_document_mem(data As Variant, optlist As String) As Long</code>
C	<code>int pCOS_open_document_mem(pCOS *p, const void *data, size_t size, const char *optlist)</code>

Open a PDF document directly from memory.

data A reference to the data containing the PDF document. In C and C++ this is a pointer. In Java and C# this is a byte array. In PHP this is a string. In COM this is a variant of type byte.

size (Only for the C and C++ bindings) The length of the data in bytes.

optlist An option list specifying document options according to Table 5.1.

Returns See *pCOS_open_document()*.

Details See *pCOS_open_document()*.

```

C++ int open_document_callback(void *opaque, size_t filesize,
    size_t (*readproc)(void *opaque, void *buffer, size_t size),
    int (*seekproc)(void *opaque, long offset), string optlist)
C   int pCOS_open_document_callback(pCOS *p, 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 via a user-supplied function.

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. pCOS will not use the opaque pointer in any other way.

filesize The size of the complete PDF document in bytes.

readproc A 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 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 5.1.

Returns See `pCOS_open_document()`.

Details See `pCOS_open_document()`.

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

Table 5.1 Document options for `pCOS_open_document()`, `pCOS_open_document_mem()`, and `pCOS_open_document_callback()`

option	description
copy	(Boolean; only for <code>pCOS_open_document_mem()</code> , and only useful for the C and C++ bindings) If <code>true</code> , pCOS will immediately make an internal copy of the supplied PDF data. Otherwise the client is responsible for keeping the data available until the corresponding call to <code>pCOS_close_document()</code> . Default: <code>false</code>
inmemory	(Boolean; only for <code>pCOS_open_document()</code>) If <code>true</code> , pCOS will load the complete file into memory and process it from there. This can result in a tremendous performance gain on some systems (especially MVS) at the expense of memory usage. If <code>false</code> , individual parts of the document will be read from disk as needed. Default: <code>false</code>
password	(String up to 32 characters; required for encrypted documents except with <code>requiredmode</code>) The user or master password for encrypted documents. See Section 3.6, »Encrypted PDF Documents«, page 41, 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. On EBCDIC platforms the password is expected in ebcdic encoding.

Table 5.1 Document options for `pCOS_open_document()`, `pCOS_open_document_mem()`, and `pCOS_open_document_callback()`

option	description
repair	(Keyword) Specifies how to treat damaged PDF input documents. Repairing a document takes more time than normal parsing, but may allow processing of certain damaged PDFs. Note that some documents may be damaged beyond repair (default: auto): force Unconditionally try to repair the document, regardless of whether or not it has problems. auto Repair the document only if problems are detected while opening the PDF. none No attempt will be made at repairing the document. If there are problems in the PDF the function call will fail.
requiredmode	(Keyword) The minimum <code>pcosmode</code> (minimum/restricted/full) which is acceptable when opening the document. The call will fail (return -1) if the resulting <code>pcosmode</code> (see Section 3.6, »Encrypted PDF Documents«, page 41) would be lower than the required mode. If the call succeeds it is guaranteed that the resulting <code>pcosmode</code> is at least the one specified in this option. However, it may be higher; e.g. <code>requiredmode=minimum</code> for an unencrypted document will result in full mode. Default: full

C++	<code>void close_document(int doc)</code>
C# Java	<code>void close_document(int doc)</code>
Perl PHP	<code>pCOS_close_document(resource p, int doc)</code>
VB	<code>Sub close_document(doc As Long)</code>
C	<code>void pCOS_close_document(pCOS *p, int doc)</code>

Release a document handle and all internal resources related to that document.

doc A valid document handle obtained with `pCOS_open_document*()`.

Details This function must be called for cleanup when processing is done, and before `pCOS_delete()` is called.

5.4 Exception Handling

pCOS supplies auxiliary methods for handling library exceptions in the C language. Other pCOS language bindings use the native exception handling system of the respective language, such as *try/catch* clauses. The language wrappers will pack information about exception number, description, and API function name into the generated exception object. In the Java language binding these items can be retrieved selectively.

When a pCOS exception occurred, no other pCOS function except *pCOS_delete()* may be called with the corresponding pCOS object.

The pCOS language bindings for Java and .NET define a separate *pCOSException* object which offers several members to access detailed error information.

C++ *int get_errnum()*

C# Java *int get_errnum()*

Perl PHP *int pCOS_get_errnum(resource p)*

VB *Function get_errnum() As Long*

C *int pCOS_get_errnum(pCOS *p)*

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

Returns The exception's error number.

Bindings In .NET this method is also available as *Errnum* in the *pCOSException* object.
In Java this method is also available as *get_errnum()* in the *pCOSException* object.

C++ *string get_errmsg()*

C# Java *String get_errmsg()*

Perl PHP *string pCOS_get_errmsg(resource p)*

VB *Function get_errmsg() As String*

C *const char *pCOS_get_errmsg(pCOS *p)*

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

Returns A string describing the error, or an empty string if the last API call didn't cause any error.

Bindings In .NET this method is also available as *Errmsg* in the *pCOSException* object.
In Java this method is also available as *getMessage()* in the *pCOSException* object.

C++ *string get_apiname()*

C# Java *String get_apiname()*

Perl PHP *string pCOS_get_apiname(resource p)*

VB *Function get_apiname() As String*

C *const char *pCOS_get_apiname(pCOS *p)*

Get the name of the API function which threw the most recent exception or failed.

Returns The name of a pCOS API function.

Bindings In .NET this method is also available as *Apiname* in the *pCOSException* object.
In Java this method is also available as *get_apiname()* in the *pCOSException* object.

C *pCOS_TRY(pCOS *p)*

Set up an exception handling frame; must always be paired with *pCOS_CATCH()*.

Details See »Exception handling in C«, page 44.

C *pCOS_CATCH(pCOS *p)*

Catch an exception; must always be paired with *pCOS_TRY()*.

Details See »Exception Handling in C«, page 26.

C *pCOS_EXIT_TRY(pCOS *p)*

Inform the exception machinery that a *pCOS_TRY()* will be left without entering the corresponding *pCOS_CATCH()* clause.

Details See »Exception Handling in C«, page 26.

C *pCOS_RETHROW(pCOS *p)*

Re-throw an exception to another handler.

Details See »Exception Handling in C«, page 26.

5.5 Option Handling

C++
C# Java
Perl PHP
VB
C

void set_option(string optlist)
void set_option(String optlist)
pCOS_set_option(resource p, String optlist)
Sub set_option(optlist As String)
void pCOS_set_option(pCOS *p, const char *optlist)

Set one or more global options.

optlist An option list specifying global options according to Table 5.2. 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.

Details Multiple calls to this function can be used to accumulate values for those options marked in Table 5.2. For unmarked options the new value will override the old one.

Table 5.2 Global options for pCOS_set_option()

option	description
license	(String) Set the license key. It must be set before the first call to pCOS_open_document().
licensefile	(String) Set the name of a file containing the license key(s). The license file can be set only once before the first call to pCOS_open_document(). Alternatively, the name of the license file can be supplied in an environment variable called PDFLIBLICENSEFILE or (on Windows) via the registry.
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. An empty string deletes all existing search path entries. On Windows the search path can also be set via a registry entry. Default: empty

1. Option values can be accumulated with multiple calls.

5.6 pCOS Query Functions

C++
C# Java
Perl PHP
VB
C

double get_number(int doc, string path)
double get_number(int doc, String path)
float pCOS_get_number(resource p, int doc, String path)
Function get_number(doc as Long, path As String) As Double
double pCOS_get_number(pCOS *p, int doc, const char *path, ...)

Get the value of a pCOS path with type *number* or *boolean*.

doc A valid document handle obtained with *pCOS_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
C

string get_string(int doc, string path)
String get_string(int doc, String path)
String pCOS_get_string(resource p, int doc, String path)
Function get_string(doc as Long, path As String) As String
const char *pCOS_get_string(pCOS *p, int doc, const char *path, ...)

Get the value of a pCOS path with type *name*, *string*, or *boolean*.

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

path A full pCOS path for a string, name, or boolean object.

Additional parameters (C language binding only) A variable number of additional parameters can be supplied if the *key* parameter contains corresponding placeholders (%s for strings or %d for integers; use %% for a single percent sign). Using these parameters will save you from explicitly formatting complex paths containing variable numerical or string values. The client is responsible for making sure that the number and type of the placeholders matches the supplied additional parameters.

Returns A string with the value of the object identified by the pCOS path. For Boolean values the strings *true* or *false* will be returned.

Details This function will raise an exception if pCOS does not run in full mode and the type of the object is *string* (see Section 3.6, »Encrypted PDF Documents«, page 41). As an exception, the objects */Info/** (document info keys) can also be retrieved in restricted pCOS mode if *nocopy=false* or *plainmetadata=true*, and *bookmarks[...]/Title* and *pages[...]/Annots/Contents* can be retrieved in restricted pCOS mode if *nocopy=false*.

This function assumes that strings retrieved from the PDF document are text strings. String objects which contain binary data should be retrieved with `pCOS_get_stream()` instead which does not modify the data in any way.

Bindings C and C++ language bindings: The string will be returned in UTF-8 format without BOM. C binding: The returned strings will be stored in a ring buffer with up to 10 entries. If more than 10 strings are queried, buffers will be reused, which means that clients must copy the strings if they want to access more than 10 strings in parallel. For example, up to 10 calls to this function can be used as parameters for a `printf()` statement since the return strings are guaranteed to be independent if no more than 10 strings are used at the same time.

C++	<code>const unsigned char *get_stream(int doc, int *length, string optlist, string path)</code>
C# Java	<code>final byte[] get_stream(int doc, String optlist, String path)</code>
Perl PHP	<code>string pCOS_get_stream(resource p, int doc, String optlist, String path)</code>
VB	<code>Function get_stream(doc as Long, optlist As String, path As String)</code>
C	<code>const unsigned char *pCOS_get_stream(pCOS *p, int doc, int *length, const char *optlist, const char *path, ...)</code>

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

doc A valid document handle obtained with `pCOS_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 options according to Table 5.3.

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. The returned data will be empty (in C and C++: NULL) if the stream is empty.

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

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

Details This function will throw an exception if pCOS does not run in full mode (see Section 3.6, »Encrypted PDF Documents«, page 41). 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* or *fstream*.

Despite its name this function can also be used to retrieve objects of type *string*. Unlike `pCOS_pcos_get_string()`, which treats the object as a text string, this function will not

modify the returned data in any way. Binary string data is rarely used in PDF, and cannot be reliably detected automatically. The user is therefore responsible for selecting the appropriate function for retrieving string objects as binary data or text.

Bindings COM: Most client programs will use the Variant type to hold the stream contents. JavaScript with COM does not allow to retrieve the length of the returned variant array (but it does work with other languages and COM).
C and C++ language bindings: The returned data buffer can be used until the next call to this function.

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

Table 5.3 Options for `pCOS_get_stream()`

option	description
convert	(Keyword; will be ignored for streams which are compressed with unsupported filters) Controls whether or not the string or stream contents will be converted (default: none): none Treat the contents as binary data without any conversion. unicode Treat the contents as textual data (i.e. exactly as in <code>pCOS_get_string()</code>), 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 <code>true</code> , the stream data will be compressed with the filter which is specified in the image's <code>filterinfo</code> pseudo object (see Table 3.5, page 39). If <code>false</code> , the stream data will be uncompressed. Default: <code>true</code> for all unsupported filters, <code>false</code> otherwise

5.7 Unicode Conversion Functions

These functions may be useful for Unicode string conversion. They are provided for the benefit of users working with language environments that are not Unicode-aware.

C++ *string utf16_to_utf8(string utf16string)*

Perl PHP *string pCOS_utf16_to_utf8(resource p, string utf16string)*

C *const char *pCOS_utf16_to_utf8(pCOS *p, const char *utf16string, int len, int *size)*

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

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

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

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

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

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

C++ *string utf8_to_utf16(string utf8string, string ordering)*

Perl PHP *pCOS_utf8_to_utf16(resource p, string utf8string, string ordering)*

C *const char *pCOS_utf8_to_utf16(pCOS *p, const char *utf8string, const char *ordering, int *size)*

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

utf8string String to be converted. It must contain a valid UTF-8 sequence (on EBCDIC platforms it must be encoded in EBCDIC). If a Byte Order Mark (BOM) is present, it will be removed.

ordering Specifies the byte ordering of the result string:

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

size (Only C language binding) Pointer to a memory location where the length of the returned string (in bytes, but excluding the terminating two null bytes) will be stored.

Returns The converted UTF-16 string. In C it will be terminated by two null bytes. The string is valid until the next call to any function, or until an exception is thrown. Clients must copy the string if they need it longer. The memory used for the converted string will be managed by pCOS.

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



A pCOS Library Quick Reference

General Functions

Function prototype	page
<code>pCOS *pCOS_new(void)</code>	54
<code>void delete()</code>	54

Document Functions

Function prototype	page
<code>int open_document(String filename, String optlist)</code>	55
<code>int open_document_mem(byte[] data, String optlist)</code>	55
<code>int open_document_callback(void *opaque, size_t filesize, size_t (*readproc)(void *opaque, void *buffer, size_t size), int (*seekproc)(void *opaque, long offset), string optlist)</code>	56
<code>void close_document(int doc)</code>	57

Exception Handling Functions

Function prototype	page
<code>String get_apiname()</code>	58
<code>String get_errmsg()</code>	58
<code>int get_errnum()</code>	58

Option Handling

Function prototype	page
<code>void set_option(String optlist)</code>	60

pCOS Query Functions

Function prototype	page
<code>double get_number(int doc, String path)</code>	61
<code>String get_string(int doc, String path)</code>	61
<code>final byte[] get_stream(int doc, String optlist, String path)</code>	62

Unicode Conversion Functions

Function prototype	page
<code>const char *pCOS_utf8_to_utf16(pCOS *p, const char *utf8string, const char *ordering, int *size)</code>	64
<code>const char *pCOS_utf16_to_utf8(pCOS *p, const char *utf16string, int len, int *size)</code>	64

B Revision History

Revision history of this manual

<i>Date</i>	<i>Changes</i>
<i>October 19, 2007</i>	► <i>Updates for pCOS 2.0</i>
<i>March 28, 2006</i>	► <i>Added a description of the Perl language binding</i>
<i>September 30, 2005</i>	► <i>Edition for pCOS 1.0.0</i>

Index

A

API (Application Programming Interface)
 reference 53
arrays 32

B

Byte Order Mark (BOM) 64

C

C binding 44
C++ binding 46
COM binding 47
command-line tool
 see *pCOS command-line tool* 19
cookbook 16
CSV (comma-separated values) format 13

D

dictionaries 32
dictionary: querying contained keys 14
document and page functions 55
document info fields 29

E

encrypted PDF documents 41
encryption status 30
evaluation version 5
examples
 document info fields 29
 encryption status 30
 fonts in a document 30
 number of pages 29
 page size 30
 pCOS paths 16, 29
 text extraction status 30
 xmp metadata 30
Excel 13
exception handling 43
 functions 58
 in C 44

F

fonts in a document 30

I

installing pCOS 5

J

Java binding 48

L

license key 6
list values in option lists 53

N

.NET binding 49
number of pages 29

O

option handling functions 60
option lists 53

P

page size 30
pCOS
 data types 31
 encryption 41
 exception handling functions 58
 option handling functions 60
 path syntax 33
 pseudo objects 35
 query functions 61
 Unicode conversion functions 64
pCOS command-line tool 19
 binary data 27
 encrypted PDF 19
 examples with raw pCOS paths 15
 exit codes 19
 extended output mode examples 12
 extracting data from PDF 11
 file names 19
 input options 20
 output options 25
 retrieval options 21
 simple output mode examples 9
 Unicode output 27
pCOS Cookbook 16
pCOS library API reference 53
pCOS_CATCH() 59
pCOS_close_document() 57
pCOS_delete() 54
pCOS_EXIT_TRY() 44, 59
pCOS_get_apiname() 58
pCOS_get_errmsg() 58

pCOS_get_errnum() 58
pCOS_get_number() 61
pCOS_get_stream() 62
pCOS_get_string() 61
pCOS_new() 54
pCOS_open_document() 55
pCOS_open_document_callback() 56
pCOS_open_document_mem() 55
pCOS_RETHROW() 59
pCOS_set_option() 60
pCOS_TRY() 59
pCOS_utf16_to_utf8() 64
pCOS_utf8_to_utf16() 64
PDF Reference Manual 29
Perl binding 50
PHP binding 51

Q

query functions 61

S

spreadsheets: creating output for 13

T

text extraction status 30

U

Unicode conversion functions 64

X

xmp meta data 30