PDFlib GmbH

pCOS

Version 4.0

PDF Information Retrieval Tool

PDFlib GmbH
Franziska-Bilek-Weg 9, 80339 München, Germany
www.pdflib.com
phone +49 • 89 • 452 33 84-0
fax +49 • 89 • 452 33 84-99

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

Licensing contact: sales@pdflib.com
Support for commercial PDFlib licensees: support@pdflib.com (please include your license number)

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

Contents

0 First Steps with pCOS 5

0.1 Installing the Software 5

0.2 Applying the pCOS License Key 6

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 Option Handling 21

2.3 Input Options 22

2.4 Options for Retrieving PDF Elements 23

2.5 Advanced Retrieval Options 25

2.6 Output Options 27

2.7 Unicode Output and Binary Data 29

3 pCOS Library Language Bindings 31

3.1 Exception Handling 31

3.2 C Binding 32

3.3 C++ Binding 34

3.4 COM Binding 36

3.5 Java Binding 37

3.6 .NET Binding 39

3.7 Perl Binding 40

3.8 PHP Binding 41

3.9 Python Binding 43

4 pCOS Library API Reference 45

4.1 Option Lists 45

4.2 General Functions 46

4.3 Document Functions 47

4.4 Exception Handling 49

4.5 Logging *51*
4.6 Option Handling *53*
4.7 pCOS Query Functions *55*
4.8 Unicode Conversion Function *58*
4.9 PDFlib Virtual Filesystem (PVF) *60*

Index *63*

A pCOS Library Quick Reference *65*

B Revision History *66*

0 First Steps with pCOS

0.1 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 3, »pCOS Library Language Bindings«, page 31, corresponding to their environment of choice, and review the installed examples. On Windows, the pCOS programming examples are accessible via the Start menu (for COM and .NET) or in the installation directory (for other language bindings).

If you obtained a commercial 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.

0.2 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.*

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

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

```
# Licensing information for PDFlib GmbH products
PDFlib license file 1.0
pCOS 4.0 ...your license key...
```

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

- A file called *licensekeys.txt* is searched in all default locations (see »Default file search paths«, page 7).
- You can specify the *licensefile* option with the *set_option( )* API function:

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

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

  ```
  pcos --pcosopt "licensefile=/path/to/your/licensekeys.txt" ...
  ```

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

  ```
  pcos --pcosopt "licensefile={/path/to/your/license file.txt}" ...
  ```
- You can set an environment (shell) variable which points to a license file. On Windows use the system control panel and choose *System, Advanced, Environment Variables*; on Unix apply a command similar to the following:

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

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

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

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

```
HKLM\SOFTWARE\PDFlib\PCOS4\license
HKLM\SOFTWARE\PDFlib\PCOS4\4.0\license
```

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

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

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

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

```
<rootpath>/PDFlib/pCOS/4.0/resource/cmap
<rootpath>/PDFlib/pCOS/4.0/resource/codelist
<rootpath>/PDFlib/pCOS/4.0/resource/glyphlst
<rootpath>/PDFlib/pCOS/4.0/resource/fonts
<rootpath>/PDFlib/pCOS/4.0/resource/icc
<rootpath>/PDFlib/pCOS/4.0
<rootpath>/PDFlib/pCOS
<rootpath>/PDFlib
```

On Unix, Linux, and OS X *<roothpath>* will first be replaced with */usr/local* and then with the HOME directory.

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

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

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

Setting 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...
```

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

- In COM/VBScript:

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

- In C:

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

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

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

- In Perl, Python and PHP:

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

- In RPG:

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

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.

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

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

1 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.

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 are 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 emits information about the PDF/UA status of the document:

```
pcos --pcospath pdfua 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 the pCOS Path Reference, 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 and image contents from PDF pages. Text and images 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 are 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 2.4, »Options for Retrieving PDF Elements«, page 23, 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 is 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 is 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 (0-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 is 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 colorspaces 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 are searched in all directories specified as *searchpath*.
- Short forms are available for some options, and can be mixed with long options.
- Long options can be abbreviated provided the abbreviation is unique (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 are 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. Wildcards are evaluated in the current directory, not any searchpath directory.

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

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

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

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

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

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. Refer to the pCOS Path Reference for details on PDF security and pCOS modes.

2.2 Option Handling

Table 2.2 lists options related to general option handling.

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

option	***parameters***	***function***
--		*End the list of options; this is useful in case file names start with a »-« character.*
@filename[1]		*Specify a response file with options; for a syntax description see »Response files«, page 19. Response files will only be recognized before the -- option and before the first filename, and can not be used to replace the parameter for another option.*

[1] *This option can be supplied more than once.*

2.3 Input Options

Table 2.2 lists options related to the input or general processing.

Table 2.2 pCOS command-line options related to input or general processing

option	*parameters*	*function*
--docopt	*<option list>*	*Additional option list for pCOS_open_document( ) (see Table 4.1, page 48)*
--firstpage	*1, 2, ..., last*	*The number of the page where page-related processing will start. The keyword* `last` *can be used to specify the last page. Default: 1*
--lastpage	*1, 2, ..., last*	*The number of the page where page-related processing will finish. The keyword* `last` *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 pCOS_set_option( ) (see Table 4.4, page 53). This can be used to pass the* `license` *or* `licensefile` *options.*

2.4 Options for Retrieving PDF Elements

Table 2.3 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 2.3 and Table 2.4 in the order in which they have been specified on the command line. If no retrieval option has been provided, the default --*general* --*docinfo* is used.

All options in Table 2.3 except --*general* require full pCOS mode, i.e. the master password must be provided for encrypted files.

Table 2.3 pCOS command-line options for simple output retrieval

option	***function***
--annotation[1]	*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`.
--attachment[1]	*Description and file name of file attachments on the pages (see also* `--embeddedfile`*). 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* `--extract attachment`.
--block[1]	*Name and subtype of PDFlib Blocks for use with the PDFlib Personalization Server (PPS). This option queries the keys* `Name` *and* `Subtype` *in* `pages[...]/PieceInfo/PDFlib/Private/Blocks` *for all pages, using the format* `KEY: VAL\n`.
--bookmark	*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`.
--destination	*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`.
--docinfo	*Key and value of document info entries. This option queries all keys in* `/Info`, *using the format* `KEY: VAL\n`.
--embeddedfile	*File name and description of named embedded files. This option queries document-level file attachments, while* `--attachment` *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* `--extract embeddedfile`.
--field	*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.*
--font	*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`.
--general	*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), document info fields, presence of XMP metadata, PDF package status and presence of encrypted attachments. This option queries various real and pseudo objects.*

Table 2.3 pCOS command-line options for simple output retrieval

option	*function*
--javascript	*JavaScript at various locations in the document. For each script its length (in Unicode characters) is printed, as well as the total number of scripts found. Depending on the location of the JavaScript in the document, additional information is printed:* *Document open actions: JavaScript which will activated when the document is opened.* *Bookmarks: JavaScript for bookmark activation.* *Document-level JavaScript: additional information for the trigger event* (`didprint`, `didsave`, `willclose`, `willprint`, `willsave`) *Page-level JavaScript: additional information for the trigger event* (`open`, `close`) *JavaScript for annotation activation. Additional information: page number, annotation type* *Field-level JavaScript. Additional information: form field name, trigger* (`activate`, `keystroke`, `format`, `validate`, `calculate`, `enter`, `exit`, `down`, `up`, `focus`, `blur`)
--layer	*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`.
--layer-default	*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 is 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`.
--outputintent	*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`.
--pagesize[1]	*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`.
--pdfa	*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.*
--pdfx	*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.*
--signature	*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`).
--weblink[1]	*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`.
--xfa	*Checks whether the documents contains any XFA information (eXtensible Forms Architecture). This option queries the key* `/Root/AcroForm/XFA`.

[1] *This option is subject to the* `--firstpage` *and* `--lastpage` *options.*

2.5 Advanced Retrieval Options

Table 2.4 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 the pCOS Path Reference for details). If the path designates a simple object, its value is printed, dictionary objects are enumerated recursively up to the level specified with *--depth*, and array objects are completely enumerated recursively.

Table 2.4 pCOS command-line options for advanced output retrieval

long option	*parameters*	*function*
--binary		*Retrieved string objects are 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 are always treated in binary mode.*
--extended[1]	*<type> <keys>*	*Extended object retrieval for one of the following types:* `annotation`, `attachment`, `block`, `bookmark`, `destination`, `docinfo`, `font`, `layer`, `pagesize`, `signature`, `weblink` `<keys>` *contains a list of keys to be retrieved from the respective object(s). Use* `xx` *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).*
--extract[1]	*<type>*	*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* `--firstpage` *and* `--lastpage` *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 is written to a separate disk file. Starting at the directory specified with the* `--targetdir` *option, a directory is created using the name of the input PDF (without any* `.pdf` *or* `.PDF` *suffix, and with critical characters replaced with "_"). Within this directory various subdirectories for the data items are created. The* `--outfile` *option is ignored.* *In addition to the generated data files a description of all extracted data items is created on standard output.*

Table 2.4 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 0. Expressions within (...) will iterate over all existing keys. Format examples can be found in Table 2.3. 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` (or `VAL` for binary data)
--pcospath[1]	<path>...	pCOS path of an object that will be queried. Examples for object paths can be found in Table 2.3, and a full description in the pCOS Path Reference.

[1] This option can be supplied more than once.

2.6 Output Options

Table 2.5 lists options for controlling details of the generated output.

Table 2.5 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 is 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 (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. \%030) is replaced with the formatted value of the character. The replacement is 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 is output in UTF-8 format, and stream contents are output without any modification.

Table 2.5 pCOS command-line options for controlling output details

option	***parameters***	***function***	
--verbose ***-v***	*0, 1, 2, 3*	*Verbosity level (default: 1):*	
		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.

2.7 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 are 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 are 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 is 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 are output in binary format without any modification. This includes XMP metadata streams, but these are usually stored in the PDF as UTF-8 anyway. Be careful with the *--format* and *--replace* options since these may have undesired effects on binary data.

3 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.

3.1 Exception Handling

Errors of a certain kind are called exceptions in many languages for good reasons – they are mere exceptions, and are not expected to occur very often during the lifetime of a program. The general strategy is to use conventional error reporting mechanisms (read: special error return codes) for function calls which may go wrong often times, and use a special exception mechanism for those rare occasions which don't justify cluttering the code with conditionals. This is exactly the path that 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 which is damaged beyond repair.

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 is thrown instead of passing a special error return value to the caller. In languages which support native exceptions throwing the exception is 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 3.2, »C Binding«, page 32).

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.

3.2 C Binding

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

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

Exception handling. 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)
{
        ...open document...
        n_pages = (int) pCOS_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 is interpreted as UTF-16. If the *length* parameter is 0 the string is interpreted as UTF-8 if it starts with a UTF-8 BOM, or as EBCDIC UTF-8 if it starts with an EBCDIC UTF-8 BOM, or as *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 is used to determine the format of the string. More precisely, interpreting a string option works as follows:

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

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

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

- Include *pcoslibdl.h* instead of *pcoslib.h*.
- Use *pCOS_new_dl( )* and *pCOS_delete_dl( )* instead of *pCOS_new( )* and *pCOS_delete( )*.
- Use *pCOS_TRY_DL( )* and *pCOS_CATCH_DL( )* instead of *pCOS_TRY( )* and *pCOS_CATCH( )*.
- Use function pointers for all other pCOS calls.
- Compile the auxiliary module *pcoslibdl.c* and link your application against the resulting object file.

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

3.3 C++ Binding

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

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

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

- Strings of the C++ standard library type *std::wstring* are used as basic string type. They can hold Unicode characters encoded as UTF-16 or UTF-32. This is the default behavior and the recommended approach for new applications unless custom data types (see next item) offer a significant advantage over *wstrings*.
- Custom (user-defined) data types for string handling can be used as long as the custom data type is an instantiation of the *basic_string* class template and can be converted to and from Unicode via user-supplied converter methods.
- Plain C++ strings can be used for compatibility with existing C++ applications which have been developed against pCOS 2.0. This compatibility variant is only meant for existing applications (see below for notes on source code compatibility).

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

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

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

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

Using pCOS as a DLL loaded at runtime. Similar to the C language binding the C++ binding allows you to dynamically attach pCOS to your application at runtime (see »Using pCOS as a DLL loaded at runtime«, page 33). Dynamic loading can be enabled as follows when compiling the application module which includes *pcos.hpp:*

```
#define PCOSCPP_DL    1
```

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

3.4 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 4.0\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 is 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 3.6, ».NET Binding«, page 39) 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 is 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.

3.5 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 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"));
```

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

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

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

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

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

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

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

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

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

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.

3.6 .NET Binding

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

The .NET edition of 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.

Installing the pCOS Edition for .NET. Install pCOS with the supplied Windows MSI Installer. The pCOS.NET 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.

Error 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.

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*.

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

```
using namespace pCOS_dotnet;
```

3.7 Perl Binding

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

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

For the pCOS binding to work, the Perl interpreter must access the pCOS Perl wrapper and the modules *pcoslib_pl.pm* and *PDFlib/pCOS.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 *pcoslib_pl.so* (on OS X: *pcoslib_pl.bundle)*, *pcoslib_pl.pm* and *PDFlib/pCOS.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/5.10/i686-linux
```

Windows: pCOS supports the ActiveState port of Perl 5 to Windows, also known as ActivePerl. The DLL *pcoslib_pl.dll* and the modules *pcoslib_pl.pm* and *PDFlib/pCOS.pm is* searched in the current directory, or the directory printed by the following Perl command:

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

Typical output of the above command looks like

```
C:\Program Files\Perl5.10\site\lib
```

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

```
eval {
        ...some pCOS instructions...
};
die "Exception caught: $@" if $@;
```

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

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

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

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.

Exception handling. Since PHP 5 supports structured exception handling, pCOS exceptions are 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;
}
```

3.9 Python Binding

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

- Unix and OS X: *pcoslib_py.so*
- Windows: *pcoslib_py.pyd*

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

```
try:
        ...some pCOS instructions...
except pCOSException:
        print("pCOS exception occurred:\n[%d] %s: %s" %
            ((p.get_errnum()), p.get_apiname(), p.get_errmsg()))
```

4 pCOS Library API Reference

4.1 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, 0x,* or *0X* 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

4.2 General Functions

C ***pCOS *pCOS_new(void)***

Create a new pCOS object.

Returns A handle to a pCOS object to be used in subsequent calls. If this function doesn't succeed due to unavailable memory it will return NULL.

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

Java ***void delete( )***

C# ***void Dispose( )***

C ***void pCOS_delete(pCOS *p)***

Delete a pCOS object and release all related internal resources.

Details All open documents in the context are closed automatically. It is good programming practice, however, to close documents explicitly with *pCOS_close_document( )* when they are no longer needed.The pCOS object must no longer be used after this function has been called.

Bindings In object-oriented language bindings this function is generally not required since it is hidden in the pCOS destructor. However, in Java it is available nevertheless to allow explicit cleanup in addition to automatic garbage collection. In .NET *Dispose( )* should be called at the end of processing to clean up unmanaged resources.

4.3 Document Functions

C++ Java C# ***int open_document(String filename, String optlist)***

Perl PHP ***int open_document(string filename, string optlist)***

C ***int pCOS_open_document(pCOS *p, const char *filename, int len, const char *optlist)***

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 is 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 is* interpreted in the current system codepage unless they are preceded by a UTF-8 BOM, in which case they is 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 4.1.

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++ ***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 is 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 4.1.

Returns See *pCOS_open_document( )*.

Details See *pCOS_open_document( )*.

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

Table 4.1 Document options for pCOS_open_document( ) and pCOS_open_document_callback( )

option	***description***
inmemory	*(Boolean; only for pCOS_open_document( )) If* `true`*, 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* `false`*, individual parts of the document are read from disk as needed. Default:* `false`
password	*(String up to 32 characters; required for encrypted documents except with* `requiredmode`*) The user or master password for encrypted documents. See the pCOS Path Reference to find out how to query a document's encryption status, and pCOS operations which can be applied even without knowing the user or master password. On EBCDIC platforms the password is expected in ebcdic encoding.*
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 is made at repairing the document. If there are problems in the PDF the function call will fail.*
requiredmode	*(Keyword) The minimum pcosmode* (`minimum/restricted/full`) *which is acceptable when opening the document. The call will fail (return -1) if the resulting pcosmode (see the pCOS Path Reference) would be lower than the required mode. If the call succeeds it is guaranteed that the resulting pcosmode is at least the one specified in this option. However, it may be higher; e.g.* `requiredmode=minimum` *for an unencrypted document will result in full mode. Default:* `full`
shrug	*(Boolean) Access restrictions are ignored (i.e. PDF processing is allowed) in the following situation: the document is encrypted with a master password, but only the user password (if any) has been supplied. When permissions are ignored, the pCOS pseudo object* `shrug` *is set to* `true`*. Default:* `false`

C++ Java C# void close_document(int doc)
Perl PHP close_document(int doc)
C void pCOS_close_document(pCOS *p, int doc)

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.

4.4 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++ Java C# ***int get_errnum( )***
Perl PHP ***int get_errnum( )***
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++ Java C# ***String get_errmsg( )***
Perl PHP ***string get_errmsg( )***
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++ Java C# ***String get_apiname( )***
Perl PHP ***string get_apiname( )***
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«, page 32.

C *pCOS_CATCH(pCOS *p)*

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

Details See »Exception handling«, page 32.

C *pCOS_EXIT_TRY(pCOS *p)*

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

Details See »Exception handling«, page 32.

C *pCOS_RETHROW(pCOS *p)*

Re-throw an exception to another handler.

Details See »Exception handling«, page 32.

4.5 Logging

The logging feature can be used to trace API calls. The contents of the log file may be useful for debugging purposes, or may be requested by PDFlib GmbH support. Table 4.3 lists the options for activating the logging feature with *pCOS_set_option( )* (see Section 4.6, »Option Handling«, page 53).

Table 4.2 Logging-related keys for pCOS_set_option( )

key	***explanation***
logging	*Option list with logging options according to Table 4.3*
userlog	*String which is copied to the log file*

The logging options can be supplied in the following ways:

- As an option list for the *logging* option of *pCOS_set_option( )*, e.g.:

```
p.set_option("logging", "filename=debug.log remove")
```

- In an environment variable called *PCOSLOGGING*. Doing so will activate the logging output starting with the very first call to one of the API functions.

Table 4.3 Suboptions for the logging option of pCOS_set_option( ) (unsupported)

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

Table 4.3 Suboptions for the logging option of pCOS_set_option( ) (unsupported)

key	explanation
classes	*(Option list) Option list where each option describes a logging class, and the corresponding value describes the granularity level. Level 0 disables a logging class, positive numbers enable a class. Increasing levels provide more detailed output. If no level is mentioned for a class the value 1 must be used (initial value:* `api=1`*).* ***api*** *Log all API calls with their function parameters and results. If* `api=2` *a timestamp is created in front of all API trace lines, and deprecated functions and options are marked. If* `api=3` *try/catch calls are logged (useful for debugging problems with nested exception handling).* ***filesearch*** *Log all attempts related to locating files via* `SearchPath` *or PVF.* ***user*** *User-specified logging output supplied with the* `userlog` *option.* ***warning*** *Log all warnings, i.e. error conditions which can be ignored or fixed internally.* `If warning=2` *messages from functions which do not throw any exception, but hook up the message text for retrieval via pCOS_get_errmsg( ), and the reason for all failed attempts at opening a file (searching for a file in searchpath) will also be logged.*

4.6 Option Handling

C++ Java C# ***void set_option(String optlist)***
Perl PHP ***set_option(string optlist)***
C ***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 4.4. 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 4.4. For unmarked options the new value will override the old one.

Table 4.4 Global options for pCOS_set_option( )

option	description
filename-handling	*(Keyword; not required on Windows) Target encoding for input file names (default:* `unicode` *on OS X, otherwise* `honorlang`*):* ***ascii*** *7-bit ASCII* ***basicebcdic*** *Basic EBCDIC according to code page 1047, but only Unicode values <= U+007E* ***basicebcdic_37*** *Basic EBCDIC according to code page 0037, but only Unicode values <= U+007E* ***honorlang*** *The environment variables LC_ALL, LC_CTYPE and LANG are interpreted and applied to file names if it specifies* `utf8`*,* `UTF-8`*,* `cpXXXX`*,* `CPXXXX`*,* `iso8859-x`*, or* `ISO-8859-x`*.* ***legacy*** *Use* `auto` *encoding (i.e. the current system encoding) to interpret the file name and interpret the LANG variable if the* `honorlang` *parameter is set.* ***unicode*** *Unicode encoding in (EBCDIC-) UTF-8 format* ***all valid encoding names*** *Any (internal or user-defined) encoding recognized by pCOS* *File names supplied in non-Unicode aware language bindings without a UTF-8 BOM and with length=0 are interpreted according to the* `filenamehandling` *option.*
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.*
logging[1]	*(Option list; unsupported) An option list specifying logging output according to Table 4.3. Alternatively, logging options can be supplied in an environment variable called* `PCOSLOGGING` *or on Windows via the registry. An empty option list will enable logging with the options set in previous calls. If the environment variable is set logging will start immediately after the first call to pCOS_new( ).*
userlog	*(Name string) Arbitrary string which is written to the log file if logging is enabled.*

Table 4.4 Global options for pCOS_set_option( )

option	description
searchpath[1]	*(List of name strings) Relative or absolute path name(s) of a directory containing files to be read. The search path can be set multiply; the entries are accumulated and used in least-recently-set order. It is recommended to use double braces even for a single entry to avoid problems with directory names containing space characters. An empty string list (i.e. {{}}) deletes all existing search path entries including the default entries. On Windows the search path can also be set via a registry entry. Default: empty*
shutdown-strategy	*(Integer) Strategy for releasing global resources which are allocated once for all pCOS objects. Each global resource is initialized on demand when it is first needed. This option must be set to the same value for all pCOS objects in a process; otherwise the behavior is undefined (default: 0):* **0** *A reference counter keeps track of how many PLOP objects use the resource. When the last pCOS object is deleted and the reference counter drops to zero, the resource is released.* **1** *The resource is kept until the end of the process. This may slightly improve performance, but requires more memory after the last pCOS object is deleted.*

[1] *Option values can be accumulated with multiple calls.*

4.7 pCOS Query Functions

C++ Java C# ***double pcos_get_number(int doc, String path)***
Perl PHP ***float pcos_get_number(int doc, string path)***
C ***double pCOS_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 is returned if they are *true,* and 0 otherwise.

C++ Java C# ***String pcos_get_string(int doc, String path)***
Perl PHP ***string pcos_get_string(int doc, string path)***
C ***const char *pCOS_pcos_get_string(pCOS *p, int doc, const char *path, ...)***

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

doc A valid document handle obtained with *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* is returned.

Details This function raises an exception if pCOS does not run in full mode and the type of the object is *string*. However, the objects */Info/** (document info keys) can also be retrieved in restricted pCOS mode if *nocopy=false* or *plainmetadata=true*, and *bookmarks[...]/Title* as well as all paths starting with *pages[...]/annots[...]/* can be retrieved in restricted pCOS mode if *nocopy=false*.

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

Bindings C binding: The returned strings are stored in a ring buffer with up to 10 entries. If more than 10 strings are queried, buffers are reused, which means that clients must copy the strings if they want to access more than 10 strings in parallel. For example, up to 10 calls to this 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.

Bindings C language binding: The string is returned in UTF-8 format (on zSeries and i5/iSeries: EBCDIC-UTF-8) without BOM. The returned strings are stored in a ring buffer with up to 10 entries. If more than 10 strings are queried, buffers are reused, which means that clients must copy the strings if they want to access more than 10 strings in parallel. For example, up to 10 calls to this function can be used as parameters for a *printf( )* statement since the return strings are guaranteed to be independent if no more than 10 strings are used at the same time.

C++ language binding: The string is returned as *wstring* in the default *wstring* configuration of the C++ wrapper. In *string* compatibility mode on zSeries and i5/iSeries the result is returned in EBCDIC-UTF-8 without BOM.

C++ Java C# final byte[ ] pcos_get_stream(int doc, String optlist, String path)

Perl PHP string pcos_get_stream(int doc, string optlist, string path)

C const unsigned char *pCOS_pcos_get_stream(pCOS *p, int doc, int *length, const char *optlist, const char *path, ...)

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

doc A valid document handle obtained with *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 4.5.

path A full pCOS path for a stream or string object.

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

Returns The unencrypted data contained in the stream or string. The returned data is empty (in C and C++: NULL) if the stream or string is empty, or if the contents of encrypted attachments in an unencrypted document are queried and the attachment password has not been supplied.

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

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

Details This function will throw an exception if pCOS does not run in full mode (see the pCOS Path Reference). As an exception, the object */Root/Metadata* can also be retrieved in restricted pCOS mode if *nocopy=false* or *plainmetadata=true*. An exception will also be thrown if *path* does not point to an object of type *stream*, *fstream*, or *string*.

Despite its name this function can also be used to retrieve objects of type *string*. Unlike *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 4.5 Options for pCOS_pcos_get_stream( )

option	*description*
convert	*(Keyword; ignored for streams which are compressed with unsupported filters) Controls whether or not the string or stream contents are converted (default:* none*):* ***none*** *Treat the contents as binary data without any conversion.* ***unicode*** *Treat the contents as textual data (i.e. exactly as in pCOS_pcos_get_string( )), and normalize it to Unicode. In non-Unicode-aware language bindings this means the data is converted to UTF-8 format without BOM.* *This option is required for the data type »text stream« in PDF which is rarely used (e.g. it can be used for JavaScript, although the majority of JavaScripts is contained in string objects, not stream objects).*
keepfilter	*(Boolean; Recommended only for image data streams; ignored for streams which are compressed with unsupported filters) If* true*, the stream data is compressed with the filter which is specified in the image's* filterinfo *pseudo object (see the pCOS Path Reference). If* false*, the stream data is uncompressed. Default:* true *for all unsupported filters,* false *otherwise*

4.8 Unicode Conversion Function

C++ Java C# ***String convert_to_unicode(String inputformat, byte[ ] input, String optlist)***

Perl PHP ***string convert_to_unicode(string inputformat, string input, string optlist)***

C ***const char *pCOS_convert_to_unicode(pCOS *p,***
const char *inputformat, const char *input, int inputlen, int *outputlen, const char *optlist)

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

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

- Unicode text formats: *utf8, ebcdicutf8, utf16, utf16le, utf16be, utf32*
- An encoding name
- The keyword *auto* specifies the following behavior: if the input string contains a UTF-8 or UTF-16 BOM it is used to determine the appropriate format, otherwise the current system codepage is assumed.

input String to be converted to Unicode.

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

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

optlist An option list specifying options according to Table 4.6:

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

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

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

Bindings C binding: the returned strings is stored in a ring buffer with up to 10 entries. If more than 10 strings are converted, the buffers is reused, which means that clients must copy the strings if they want to access more than 10 strings in parallel. For example, up to 10 calls to this function can be used as parameters for a *printf( )* statement since the return strings are guaranteed to be independent if no more than 10 strings are used at the same time.

C++ binding: The parameters *inputformat* and *optlist* must be passed as *wstrings* as usual, while *input* and returned data must have type *string*.
Python binding: UTF-8 results is returned as a string, Python 3: non-UTF-8 results is returned as bytes.

Table 4.6 Options for TET_convert_to_unicode( )

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

4.9 PDFlib Virtual Filesystem (PVF)

C++ ***void create_pvf(string filename, const void *data, size_t size, string optlist)***

Java C# ***void create_pvf(String filename, byte[] data, String optlist)***

Perl PHP ***create_pvf(string filename, string data, string optlist)***

C ***void pCOS_create_pvf(pCOS *p,***
const char *filename, int len, const void *data, size_t size, const char *optlist)

Create a named virtual read-only file from data provided in memory.

filename (Name string) The name of the virtual file. This is an arbitrary string which can later be used to refer to the virtual file in other pCOS calls.

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

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

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

optlist An option list according to Table 4.7. The following option can be used: *copy*

Details The virtual file name can be supplied to any API function which uses input files. Some of these functions may set a lock on the virtual file until the data is no longer needed. Virtual files is kept in memory until they are deleted explicitly with *pCOS_delete_pvf( )*, or automatically in *pCOS_delete( )*.

Each pCOS object will maintain its own set of PVF files. Virtual files cannot be shared among different pCOS objects. Multiple threads working with separate pCOS objects do not need to synchronize PVF use. If *filename* refers to an existing virtual file an exception is thrown. This function does not check whether *filename* is already in use for a regular disk file.

Unless the *copy* option has been supplied, the caller must not modify or free (delete) the supplied data before a corresponding successful call to *pCOS_delete_pvf( )*. Not obeying to this rule will most likely result in a crash.

Table 4.7 Options for pCOS_create_pvf( )

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

C++ Java C# ***int delete_pvf(String filename)***

Perl PHP ***int delete_pvf(string filename)***

C ***int pCOS_delete_pvf(pCOS *p, const char *filename, int len)***

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

filename (Name string) The name of the virtual file as supplied to *pCOS_create_pvf( )*.

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

Returns -1 if the corresponding virtual file exists but is locked, and 1 otherwise.

Details If the file isn't locked, pCOS will immediately delete the data structures associated with *filename*. If *filename* does not refer to a valid virtual file this function will silently do nothing. After successfully calling this function *filename* may be reused. All virtual files will automatically be deleted in *pCOS_delete( )*.

The detailed semantics depend on whether or not the *copy* option has been supplied to the corresponding call to *pCOS_create_pvf( ):* If the *copy* option has been supplied, both the administrative data structures for the file and the actual file contents (data) is freed; otherwise, the contents will not be freed, since the client is supposed to do so.

C++ Java C# ***int info_pvf(String filename, String keyword)***

Perl PHP ***int info_pvf(string filename, string keyword)***

C ***int pCOS_info_pvf(pCOS *p, const char *filename, int len, const char *keyword)***

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

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

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

keyword A keyword according to Table 4.8.

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

Table 4.8 Keywords for pCOS_info_pvf( )

option	***description***
filecount	*Total number of files in the PDFlib Virtual File system maintained for the current pCOS object. The* `filename` *parameter is ignored.*
exists	*1 if the file exists in the PDFlib Virtual File system (and has not been deleted), otherwise 0*
size	*(Only for existing virtual files) Size of the specified virtual file in bytes.*
iscopy	*(Only for existing virtual files) 1 if the* `copy` *option was supplied when the specified virtual file was created, otherwise 0*
lockcount	*(Only for existing virtual files) Number of locks for the specified virtual file set internally by pCOS functions. The file can only be deleted if the lock count is 0.*

Index

A

API (Application Programming Interface)
  reference 45

C

C binding 32
C++ and .NET 39
C++ binding 34
CLI 34
COM binding 36
command-line tool: see pCOS command-line tool 19
commercial license 8
cookbook 16
CSV (comma-separated values) format 13

D

dictionary: querying contained keys 14
document and page functions 47

E

evaluation version 5
examples
  extended mode 12
  extracting data 11
  pCOS library calls 16
  raw pCOS paths 15
  simple 9
Excel 13
exception handling 31
  functions 49
  in C 32

I

installing pCOS 5

J

J2EE application servers 37
Java binding 37

L

license key 6
list values in option lists 45

N

.NET binding 39

O

option handling functions 53
option lists 45

P

pCOS
  exception handling functions 49
  option handling functions 53
  query functions 55
  Unicode conversion function 58
pCOS command-line tool 19
  binary data 29
  encrypted PDF 20
  examples with raw pCOS paths 15
  exit codes 20
  extended output mode examples 12
  extracting data from PDF 11
  file names 19
  input options 22
  option handling 21
  output options 27
  retrieval options 23
  simple output mode examples 9
  Unicode output 29
pCOS Cookbook 16
pCOS library API reference 45
pCOS_CATCH() 50
pCOS_close_document( ) 48
pCOS_convert_to_unicode( ) 58
pCOS_create_pvf() 60
pCOS_delete_pvf() 61
pCOS_delete() 46
pCOS_EXIT_TRY() 32, 50
pCOS_get_apiname() 49
pCOS_get_errmsg() 49
pCOS_get_errnum() 49
pCOS_info_pvf() 61
pCOS_new() 46
pCOS_open_document_callback() 47
pCOS_open_document() 47
pCOS_pcos_get_number() 55
pCOS_pcos_get_stream() 56
pCOS_pcos_get_string() 55
pCOS_RETHROW() 50
pCOS_set_option() 53
pCOS_TRY() 49

Perl binding 40
PHP binding 41
Python binding 43

Q

query functions 55

R

response file 19

S

servlets 37
spreadsheets: creating output for 13

U

Unicode conversion function 58

A pCOS Library Quick Reference

The following tables contain an overview of all pCOS API functions. The prefix *(C)* denotes C prototypes of functions which are not available in the Java language binding.

Setup Functions

Function prototype	***page***
*(C) pCOS *pCOS_new(void)*	*46*
void delete( )	*46*

Exception Handling Functions

Function prototype	***page***
String get_apiname( )	*49*
String get_errmsg( )	*49*
int get_errnum( )	*49*

Document Functions

Function prototype	***page***
int open_document(String filename, String optlist)	*47*
void close_document(int doc)	*48*

pCOS Query Functions

Function prototype	***page***
double pcos_get_number(int doc, String path)	*55*
String pcos_get_string(int doc, String path)	*55*
final byte[ ] pcos_get_stream(int doc, String optlist, String path)	*56*

Option Handling

Function prototype	***page***
void set_option(String optlist)	*53*

Unicode Conversion Function

Function prototype	***page***
String convert_to_unicode(String inputformat, byte[ ] input, String optlist)	*58*

PVF Functions

Function prototype	***page***
void create_pvf(String filename, byte[] data, String optlist)	*60*
int delete_pvf(String filename)	*61*
int info_pvf(String filename, String keyword)	*61*

B Revision History

Revision history of this manual

Date	***Changes***
August 02, 2013	► *Updates for pCOS 4.0*
October 29, 2010	► *Updates for pCOS 3.0*
July 22, 2010	► *Moved the pCOS reference for pCOS interface version 6 to a separate manual for use in multiple products*
December 07, 2009	► *Updates for pCOS interface 5 in PDFlib+PDI 8, PPS 8*
February 01, 2009	► *Updates for pCOS interface 4 in PLOP 4.0, TET 3.0, TET PDF IFilter 3.0*
October 19, 2007	► *Updates for pCOS interface 3 in pCOS 2.0*
March 28, 2006	► *Added a description of the Perl language binding*
September 30, 2005	► *Edition for pCOS interface 2 in pCOS 1.0*
June 20, 2005	► *Edition for pCOS interface 1 in TET 2.0*

PDFlib GmbH

PDFlib GmbH
Franziska-Bilek-Weg 9
80339 München, Germany
www.pdflib.com
phone +49 • 89 • 452 33 84-0
fax +49 • 89 • 452 33 84-99

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

Licensing contact
sales@pdflib.com

Support
support@pdflib.com *(please include your license number)*