PDFlib GmbH

pCOS Path Reference

PDF Information Retrieval Tool

pCOS Interface Version 5

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)

Contents

1 Introduction

1.1 What is pCOS?

The pCOS *(PDFlib Comprehensive Object Syntax)* interface provides a simple and elegant facility for retrieving technical information from all sections of a PDF document which do not describe page contents, such as page dimensions, metadata, interactive elements, etc. pCOS users are assumed to have some basic knowledge of internal PDF structures and dictionary keys, but do not have to deal with PDF syntax and parsing details. We strongly recommend that pCOS users obtain a copy of the *PDF Reference*. Since the standardization of PDF 1.7 in 2008 the PDF Reference is available as ISO 32000. This standard document can be purchased from www.iso.org. If you don't want to purchase the official version you can download a free edition which is identical in content:

Document Management – Portable Document Format – Part 1: PDF 1.7, First Edition
Downloadable PDF from *www.adobe.com/devnet/pdf/pdf_reference.html*.

1.2 Roadmap to Documentation and Samples

We provide the material listed below to assist you in using pCOS successfully.

Note On Windows Vista and Windows 7 the mini samples will be installed in the »Program Files« directory by default. Due to the new protection scheme in Windows the output files created by these samples will only be visible under »compatibility files«. Recommended workaround: copy the folder with the samples to a user directory.

Mini samples for all language bindings. The *dumper* mini sample is available in all packages and for all language bindings. It provides minimal sample code for using pCOS. The mini sample is useful for testing your pCOS installation and for getting a quick overview of writing pCOS applications.

pCOS Path Reference. The *pCOS Path Reference* (this manual) contains examples and a concise description of the pCOS path syntax which forms the heart of the pCOS interface. Since the pCOS interface is included in various other PDFlib GmbH products, the pCOS Path Reference can be used with all products that include pCOS.

Corresponding Product Manual. The pCOS interface is available as a stand-alone product as well as an integrated part of other PDFlib GmbH products. Each product comes with one or more additional product-specific manuals which describe the use of the respective programming library (e.g. pCOS or TET) and the corresponding command-line tool if applicable. The product manual covers the various programming languages which are supported by a product, and discusses the API in detail.

pCOS Cookbook. The *pCOS Cookbook* is a collection of code fragments for the pCOS interface. It is available at the following URL:

```
www.pdflib.com/pcos-cookbook/
```

The pCOS Cookbook details the use of pCOS for a variety of applications. It is highly recommended because it serves as a repository of useful pCOS programming idioms.

1.3 Availability of the pCOS Interface

The pCOS interface is available as a separate product called PDFlib pCOS. It is also offered as an integrated feature in several other PDFlib GmbH products. As the interface is extended and support for newer PDF input versions is added, the pCOS interface number is increased. Table 1.1 details the pCOS interface numbers which are implemented in various product versions

Table 1.1 pCOS interface versions implemented in PDFlib GmbH products

pCOS interface	supported PDF input version / corresponding Acrobat version	PDFlib GmbH product name and version
1	PDF 1.6 / Acrobat 7	TET 2.0, 2.1
2	PDF 1.6 / Acrobat 7	pCOS 1.0
3	PDF 1.7[1] / Acrobat 8	PDFlib+PDI 7, PPS 7, TET 2.2, pCOS 2.0, PLOP 3.0, TET 2.3
4	PDF 1.7 extension level 3 / Acrobat 9 excluding AES-256 encryption	PLOP 4.0, TET 3.0, TET PDF IFilter 3.0
5	PDF 1.7 extension level 3 / Acrobat 9	PDFlib+PDI 8, PPS 8

1. Identical to ISO 32000-1

Some aspects of the pCOS interface are available only in the TET product, but not in other PDFlib GmbH products. These features are explicitly marked in this manual.

2 pCOS Examples

This chapter provides examples for pCOS paths which can be used to retrieve the corresponding values from PDF documents. More elaborate examples which require additional programming logic are available in the pCOS Cookbook on the PDFlib Web site.

Except where noted otherwise all programming examples are presented in the Java language. However, with the obvious changes (mostly of syntactic nature) the examples can be used with all programming languages supported by pCOS.

The examples shown in this chapter are not comprehensive. Many more pCOS applications are possible by using other PDF objects.

2.1 pCOS Functions

Basic pCOS function calls. The following functions are the workhorses for querying PDF documents with pCOS:

- *pcos_get_number( )* retrieves objects of type number or boolean;
- *pcos_get_string( )* retrieves objects of type name, string, or boolean;
- *pcos_get_stream( )* retrieves objects of type stream, fstream, or string.

These functions can be used to retrieve information from a PDF document using the pCOS path syntax. The basic structure of a pCOS application looks as follows:

```
/* Open the PDF document */
int doc = p.open_pdi_document(filename, "");
if (doc == -1)
        throw new Exception("Error: " + p.get_errmsg());

/* Retrieve the value of a pCOS pseudo object */
System.out.println(" PDF version: " + p.pcos_get_string(doc, "pdfversionstring"));

p.close_pdi_document(doc);
```

The parameters for the pCOS functions are the same in all products. They are documented in the respective product reference manuals; a quick overview of pCOS function prototypes is available in Appendix A, »pCOS Function Reference«.

Adding programming logic. Many pCOS objects consist of arrays of some length. The length can be retrieved with the *length:* prefix. The array can then be indexed with integer values in the range 0 up to *length-1*. The following code queries the number of fonts in a document and prints the type and name of each font:

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

for (i = 0; i < count; i++) {
    String fonts;

    System.out.print(p.pcos_get_string(doc, "fonts[" + i + "]/type") + " font ");
    System.out.println(p.pcos_get_string(doc, fonts[" + i + "]/name));
}
```

Formatting placeholders in C. The C language binding offers a convenience feature to facilitate the use of parameters within a pCOS path. Analogous to the formatting parameters of the *printf( )* family of functions you can use *%s* and *%d* placeholders for string and integer parameters, respectively. The values of these parameters must be added as additional function parameters after the pCOS path. pCOS will replace the placeholders with the actual values. This feature is particularly useful for paths containing array indices.
For example, the Java idiom above for listing all fonts can be written in C as follows:

```
count = (int) PDF_pcos_get_number(p, doc, "length:fonts");

for (i = 0; i < count; i++)
{
    printf("%s font ", PDF_pcos_get_string(p, doc, "fonts[%d]/type", i));
    printf("%s\n", PDF_pcos_get_string(p, doc, "fonts[%d]/name", i));
}
```

Since modern programming languages offer more sophisticated string handling functions this feature is only available in the C language binding, but not any other language binding.

2.2 Document

Table 2.1 lists pCOS paths for general and document-related objects.

Table 2.1 pCOS paths for document-related items

pCOS path	*type*	*explanation*
pcosmode	*number*	*pCOS mode of the document, i.e. its encryption status (see Section 4.8, »Protected PDF Documents and pCOS Mode«, page 31)*
pdfversionstring	*string*	*string representing the PDF version number of the document*
/Info/Title	*string*	*Document info field* `Title`; *The following field names are predefined in PDF and can be used in a similar manner:* `Title, Author, Subject, Keywords, Creator, Producer, CreationDate, ModDate, Trapped`
/Info/ArticleNumber	*string*	*custom document info field* `ArticleNumber` *(document info entries can use arbitrary names)*
/Root/Metadata	*stream*	*XMP stream with the document's metadata*
pdfa, pdfe, pdfx	*string*	*PDF/A, PDF/E, or PDF/X standard conformance status*

Encryption status and pCOS mode. You can query the *pcosmode* pseudo object to determine the pCOS mode for the document. This is important to avoid exception when later an attempt is made at retrieving information for which no access is granted (e.g. because the password is encrypted and no suitable password has been supplied). The following general structure based on values of *pcosmode* is recommended for all pCOS applications:

```
/* Open the PDF document */
int doc = p.open_pdi_document(filename, "");
if (doc == -1)
        throw new Exception("Error: " + p.get_errmsg());

int pcosmode = (int) p.pcos_get_number(doc, "pcosmode");
boolean plainmetadata = p.pcos_get_number(doc, "encrypt/plainmetadata") != 0;

// Retrieve universal pseudo objects which are always available
System.out.println(" PDF version: " + p.pcos_get_string(doc, "pdfversionstring"));
System.out.println("  Encryption: " + p.pcos_get_string(doc, "encrypt/description"));

// encrypted document, but password was not supplied
if (pcosmode == 0) {
    System.out.println("Minimum mode: no more information available\n");
    p.delete();
    return;
}

// otherwise query more information
System.out.println("PDF/A status: " + p.pcos_get_string(doc, "pdfa"));

// no master password supplied; we cannot retrieve metadata
if (pcosmode == 1 && !plainmetadata && p.pcos_get_number(doc, "encrypt/nocopy") != 0) {
    System.out.print("Restricted mode: no more information available");
    p.delete();
    return;
```

```
}

// otherwise we can query document information fields and XMP metadata
...

p.close_pdi_document(doc);
```

PDF version. The following code fragment prints the PDF version number of a document:

```
System.out.println(" PDF version: " + p.pcos_get_string(doc, "pdfversionstring"));
```

Document info fields. Document information fields can be retrieved with the following code sequence. In order to make sure that an object actually exists in the PDF document and has the expected type we first check its type. If the object is present and has type *string* we can retrieve it:

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

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

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

PDF standards. The PDF/A, PDF/E, or PDF/X standard conformance status can be queried with simple pCOS pseudo objects as follows:

```
System.out.println("PDF/A status: " + p.pcos_get_string(doc, "pdfa"));
System.out.println("PDF/E status: " + p.pcos_get_string(doc, "pdfe"));
System.out.println("PDF/X status: " + p.pcos_get_string(doc, "pdfx"));
```

2.3 Pages

Table 2.2 lists pCOS paths for page-related objects.

Table 2.2 pCOS paths for page-related items

pCOS path	*type*	*explanation*
length:pages	*number*	*number of pages in the document*
pages[...]/width ***pages[...]/height***	*number*	*width and height of the page indexed in the array (keep in mind that array index are 0-based)*

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

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

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

```
pagenum = 2         // page 3 (0-based)
width  = p.pcos_get_number(doc, "pages[" + pagenum + "]/width");
height = p.pcos_get_number(doc, "pages[" + pagenum + "]/height");
```

2.4 Fonts

Table 2.3 lists pCOS paths for objects related to fonts.

Table 2.3 pCOS paths for font-related properties

pCOS path	*type*	*explanation*
length:fonts	*number*	*number of fonts in the document*
fonts[...]/name	*string*	*name of a font*
fonts[...]/vertical	*boolean*	*check a font for vertical writing mode*
fonts[...]/embedded	*boolean*	*embedding status of a font*
fonts[...]/ascender ***fonts[...]/descender***	*number*	*ascender/descender value of a font (not always available, see code sample below)*

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

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

Writing mode. The following code fragment checks whether a font uses vertical writing mode. The font is identified via its id, i.e. the index in the *fonts* array. This *id* can be obtained by enumerating all possible index values:

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

TET The TET product also provides font IDs with the get_char_info( ) function.

Font metrics. Fonts in PDF may contain a font descriptor dictionary with metrics values and other information about the font. Since not all fonts contain a font descriptor you must first query its existence:

```
count = p.pcos_get_number(doc, "length:fonts");
for (i=0; i < count; i++)
{
        ascender  = p.pcos_get_number(doc, "fonts[" + i + "]/ascender");
        descender = p.pcos_get_number(doc, "fonts[" + i + "]/descender");
        /* ... */
}
```

2.5 Raster Images

Table 2.4 lists pCOS paths for objects related to raster images.

Table 2.4 pCOS paths for image-related properties

pCOS path	*type*	*explanation*
length:images	*number*	*number of raster images in the document*
images[...]/Width	*number*	*image width in pixels*
images[...]/Height	*number*	*image height in pixels*

Listing all images. Similar to the font list you can create a list of all images in the document:

```
count = p.pcos_get_number(doc, "length:images");
for (i=0; i < count; i++)
{
        width  = p.pcos_get_string(doc, "images[" + i + "]/Width");
        height = p.pcos_get_number(doc, "images[" + i + "]/Height");
        bpc    = p.pcos_get_number(doc, "images[" + i + "]/bpc");
}
```

2.6 Interactive Elements

Table 2.5 lists pCOS paths for objects related to interactive elements.

Table 2.5 pCOS paths for various PDF objects

pCOS path	*type*	*explanation*
length:bookmarks	*number*	*number of bookmarks in the document*
bookmarks[...]/Title	*string*	*bookmark text*
bookmarks[...]/destpage	*number*	*number of the target page when the bookmark is activated, or -1 if the bookmark does not jump to any page in the document*
pages[...]/annots[...]/A/URI	*string*	*target URL of the Web links on all pages*

Bookmarks. The following code fragment queries the number of bookmarks in the document. For each bookmark its nesting level, destination (target) page and Title are shown:

```
int count = (int) p.get_number(doc, "length:bookmarks");

for (int i = 0; i < count; ++i) {
    int level    = (int) p.get_number(doc, "bookmarks[" + i + "]/level");
    int destpage = (int) p.get_number(doc, "bookmarks[" + i + "]/destpage");

    for (int j = 0; j < level * 4; j += 1) {
        System.out.print(" ");
    }

    System.out.print(p.get_string(doc, "bookmarks[" + i + "]/Title"));

    if (destpage != -1) {
        System.out.print(": page " + destpage);
    }
}
```

3 pCOS Data Types

3.1 Basic PDF Data Types

pCOS offers the three functions *pcos_get_number( ), pcos_get_string( ), and pcos_get_stream( )*. These can be used to retrieve all basic data types which may appear in PDF documents. Refer to the PDF Reference to find out the data type of a particular object in PDF.

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

```
/* get number of pages in the document */
int n_pages = (int) p.pcos_get_number(doc, "length:pages");
```

Names and strings. Objects of type *name* and *string* can be queried with *pcos_get_string( )*. Example:

```
string title = p.pcos_get_string(doc, "/Info/Title");
```

Name objects in PDF may contain non-ASCII characters and the *#xx* syntax (hexadecimal value with prefix) to include certain special characters. pCOS deals with PDF names as follows:

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

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

```
byte[] signature = p.pcos_get_stream(doc, "", "fields[0]/V/Contents");
```

Booleans. Objects of type *boolean* can be queried with *pcos_get_number( )* and will be returned as 1 (true) or 0 (false). Example:

```
string linearized_s = p.pcos_get_string(doc, "linearized");
```

pcos_get_string( ) can also be used to query Boolean objects; in this case they will be returned as one of the strings *true* and *false*. Example:

```
int linearized_i = p.pcos_get_number(doc, "linearized");
```

Streams. Objects of type *stream* can be queried with *pcos_get_stream( )*. Example:

```
byte[] contents = p.pcos_get_stream(doc, "", "/Root/Metadata");
```

Stream data in PDF may be preprocessed with one or more compression filters. Depending on the pCOS data type *(stream* or *fstream)* the contents will be compressed or un-

compressed. Using the *keepfilter* option of *pcos_get_stream( )* the client can retrieve compressed data even for type *stream*.

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

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

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

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

3.2 Composite Data Structures

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

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

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

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

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

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

Reading dictionary entries. The following example enumerates the key/value pairs in the document info dictionary:

```
for (i = 0; i < count; i++) {
        String info;
        String key;

        info = "type:/Info[" + i + "]";
        objtype = p.pcos_get_string(doc, info);

        info = "/Info[" + i + "].key";
        key = p.pcos_get_string(doc, info);

        /* Info entries can be stored as string or name objects */
        if (objtype.equals("name") || objtype.equals("string"))
        {
            info = "/Info[" + i + "]";
```

```
            System.out.println("'" + p.pcos_get_string(doc, info) + "'");
        }
    }
```

3.3 Object Identifiers (IDs)

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

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

4 pCOS Path Reference

4.1 pCOS Path Syntax

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

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

The meaning of the various path components is as follows:

- The optional *prefix* can attain the values listed in Table 4.1.
- The optional *pseudo object name* may contain the name of a pseudo object. Pseudo objects are not present in PDF, but are supported in pCOS to provide convenient shortcuts to information which cannot easily be accessed by reading a single value in the PDF document. Pseudo objects of type *dict* can not be enumerated.
- The *name* components are dictionary keys found in the document. Multiple names are separated with a / character. An empty path, i.e. a single / denotes the document's Trailer dictionary. Each name must be a dictionary key present in the preceding dictionary. Full paths describe the chain of dictionary keys from the initial dictionary (which may be the Trailer or a pseudo object) to the target object.
- Paths or path components specifying an array or dictionary can have a numerical index which must be specified in decimal format between brackets. Nested arrays or dictionaries can be addressed with multiple index entries. The first entry in an array or dictionary has index 0.
- Paths or path components specifying a dictionary can have an index qualifier plus one of the suffixes *.key* or *.val*. This can be used to retrieve a particular dictionary key or the corresponding value of the indexed dictionary entry, respectively. If a path for a dictionary has an index qualifier it must be followed by one of these suffixes.

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

- When a path component contains any of the characters */*, *[*, *]*, or *#*, these must be expressed by a number sign # followed by a two-digit hexadecimal number.
- In Unicode-aware language bindings the path consists of a regular Unicode string which may contain ASCII and non-ASCII characters.
- In non-Unicode-aware language bindings the path must be supplied in UTF-8. The string may or may not contain a BOM, but this doesn't make any difference. A BOM may be placed at the start of the path, or at the start of individual path components (i.e. after a slash character).
  On EBCDIC systems the path must generally be supplied in *ebcdic* encoding. Characters outside the ASCII character set must be supplied as EBCDIC-UTF-8 (with or without BOM).

4.2 Path Prefixes

Prefixes can be used to query various attributes of an object (as opposed to its actual value). Table 4.1 lists all supported prefixes.

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

Table 4.1 pCOS path prefixes

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

4.3 Universal Pseudo Objects

Universal pseudo objects are available for all *pcosmode* levels, i.e. regardless of encryption and password availability. Table 4.2, Table 4.3, and Table 4.4 together list all universal pseudo objects.

4.3.1 General Document Information

Table 4.2 Universal pseudo objects for general document information

object name	explanation
encrypt	*(Dict) Dictionary with keys describing the encryption status of the document:* **length** *(Number) Length of the encryption key in bits* **algorithm** *(Number)* **description** *(String) Encryption algorithm number or description:* **-1** *Unknown encryption* **0** *No encryption* **1** *40-bit RC4 (Acrobat 2-4)* **2** *128-bit RC4 (Acrobat 5)* **3** *128-bit RC4 (Acrobat 6)* **4** *128-bit AES (Acrobat 7)* **5** *Public key on top of 128-bit RC4 (Acrobat 5)*[1] **6** *Public key on top of 128-bit AES (Acrobat 7)*[1] **7** *Adobe Policy Server (Acrobat 7)*[1] **8** *Adobe Digital Editions (EBX)*[1] **9** *(pCOS interface 5) 256-bit AES (Acrobat 9)* **10** *(pCOS interface 5) Public key on top of 256-bit AES (Acrobat 9)*[1] **master** *(Boolean)* `True` *if the PDF requires a master password to change security settings (permissions, user or master password),* `false` *otherwise* **user** *(Boolean)* `True` *if the PDF requires a user password for opening,* `false` *otherwise* **noaccessible, noannots, noassemble, nocopy, noforms, nohiresprint, nomodify, noprint** *(Boolean)* `True` *if the respective access protection is set,* `false` *otherwise* **plainmetadata** *(Boolean)* `True` *if the PDF contains unencrypted meta data,* `false` *otherwise*
filename	*(String) Name of the PDF file.*
filesize	*(Number) Size of the PDF file in bytes*
linearized	*(Boolean)* `True` *if the PDF document is linearized,* `false` *otherwise*
pcosmode **pcosmode_name**	*(Number/string) pCOS mode as number or string:* **0** *minimum* **1** *restricted* **2** *full*
shrug	*(Boolean; only in the TET product)* `True` *if and only if security settings were ignored when opening the PDF document; the client must take care of honoring the document author's intentions. For TET the value will be* `true`*, and content extraction will be allowed, if all of the following conditions are true:* ► *Shrug mode has been enabled with the* `shrug` *option.* ► *The document has a master password but this has not been supplied.* ► *The user password (if required for the document) has been supplied.* ► *Content extraction is not allowed in the document's permission settings.*

1. Documents encrypted with this algorithm can be identified, but actual decryption is not supported.

4.3.2 PDF Version Information

Table 4.3 Universal pseudo objects for PDF version information

object name	explanation
extensionlevel	*(Number) Adobe Extension Level based on ISO 32000, or* 0 *(zero) if no extension level is present. Acrobat 9 creates documents with extension level 3; Acrobat X creates extension level 8.*
fullpdfversion	*(Number) Numerical value for the PDF version number. The numbers increase with each PDF/Acrobat version. The value* `100 * BaseVersion + ExtensionLevel` *will be returned, e.g.* **150** *PDF 1.5 (Acrobat 6)* **160** *PDF 1.6 (Acrobat 7)* **170** *PDF 1.7 (Acrobat 8) = ISO 32000-1* **173** *PDF 1.7 Adobe Extension Level 3 (Acrobat 9)* **178** *(pCOS interface 5) PDF 1.7 Adobe Extension Level 8 (Acrobat X)* **200** *(pCOS interface 5) PDF 2.0 = ISO 32000-2*
pdfversion	*(Number) PDF version number multiplied by 10, e.g. 16 for PDF 1.6*
pdfversionstring	*(String) Full PDF version string in the form expected by various API functions for setting the PDF output compatibility, e.g. 1.5, 1.6, 1.7, 1.7ext3*

4.3.3 Library Identification

Table 4.4 Universal pseudo objects for library identification

object name	explanation
major **minor** **revision**	*(Number) Major, minor, or revision number of the library, respectively.*
pcosinterface	*(Number) Interface version number of the underlying pCOS implementation. See Section 1.3, »Availability of the pCOS Interface«, page 6, to learn which version of the pCOS interface is implemented in a particular PDFlib GmbH product version.*
version	*(String) Full library version string in the format <major>.<minor>.<revision>, possibly suffixed with additional qualifiers such as beta, rc, etc.*

4.4 Pseudo Objects for PDF Standard Identification

Table 4.5 lists pseudo objects for PDF standard identification. The values of these pseudo objects are created based on the respective standard identification entries in the document. They do not apply any validation against the standard.

Table 4.5 Pseudo objects for PDF standard identification

object name	*explanation*
pdfa	*(String) PDF/A (ISO 19005-1 and 19005-2) conformance level of the document. Possible values are the following:* none PDF/A-1a:2005, PDF/A-1b:2005 PDF/A-2a, PDF/A-2b, PDF/A-2u
pdfe	*(String; pCOS interface 5) PDF/E (ISO 24517-1 and 24517-2) conformance level of the document. Possible values are the following:* none PDF/E-1
pdfx	*(String) PDF/X (ISO 15930-1 etc.) conformance level of the document. Possible values are the following:* none PDF/X-1:2001, PDF/X-1a:2001, PDF/X-1a:2003 PDF/X-2:2003 PDF/X-3:2002, PDF/X-3:2003 PDF/X-4, PDF/X-4p PDF/X-5g, PDF/X-5n, PDF/X-5p

4.5 Pseudo Objects for Pages

Table 4.6 lists the pseudo objects for page information.

Table 4.6 Pseudo object for pages

object name	explanation
pages	*(Array of dicts) Each array element addresses a page of the document. Indexing it with the decimal representation of the page number minus one addresses that page (the first page has index 0). Using the length prefix the number of pages in the document can be determined. A page object addressed this way will incorporate all attributes which are inherited via the* `/Pages` *tree. The* `/MediaBox` *and* `/Rotate` *entries are guaranteed to be present. In addition to standard PDF dictionary entries the following pseudo entries are available for each page:* **colorspaces, extgstates, fonts, images, patterns, properties, shadings, templates** *(Arrays of dicts) Page resources according to Table 4.8.* **annots** *(Array of dicts) In addition to the standard PDF keys in the* `Annots` *array pCOS supports the following pseudo key for dictionaries in the* `annots` *array:* **destpage** *(Number; only for* `Subtype=Link` *and if a* `Dest` *entry is present) Number of the target page (first page is 1)* **blocks** *(Array of dicts) Shorthand for* `pages[ ]/PieceInfo/PDFlib/Private/Blocks[ ]`*, i.e. the page's block dictionary. In addition to the existing PDF keys pCOS supports the following pseudo key for dictionaries in the* `blocks` *array:* **rect** *(Rectangle) Similar to* `Rect`*, except that it takes into account any relevant* `CropBox`/`MediaBox` *and* `Rotate` *entries and normalizes coordinate ordering.* **height** *(Number) Height of the page. The MediaBox or the CropBox (if present) will be used to determine the height. Rotate entries will also be applied.* **isempty** *(Boolean)* `True` *if the page is empty, and* `false` *if the page is not empty* **label** *(String) The page label of the page (including any prefix which may be present). Labels will be displayed as in Acrobat. If no label is present (or the PageLabel dictionary is malformed), the string will contain the decimal page number. Roman numbers will be created in Acrobat's style (e.g. VL), not in classical style which is different (e.g. XLV). If* `/Root/PageLabels` *doesn't exist, the document doesn't contain any page labels.* **width** *(Number) Width of the page (same rules as for* `height`*)* *The following entries will be inherited:* `CropBox`, `MediaBox`, `Resources`, `Rotate`.

4.6 Pseudo Objects for interactive Elements

Table 4.6 lists pseudo objects which can be used for retrieving PDF objects or serve as shortcuts to various interactive elements.

Table 4.7 Pseudo objects for PDF objects and interactive elements

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

4.7 Pseudo Objects for Resources

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

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

pCOS supports two groups of pseudo objects for resource retrieval. Global resource arrays contain all resources of a given type in a PDF document, while page-based resources contain only the resources used by a particular page. The corresponding pseudo arrays are available for all resource types listed in Table 4.8:

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

Table 4.8 Pseudo objects for resources; each resource category P creates two resource arrays `P[ ]` *and* `pages[ ]/P[ ]`.

object name	*explanation*
colorspaces	*(Array of dicts) Array containing dictionaries for all color spaces on the page or in the document. In addition to the standard PDF keys in color space and ICC profile stream dictionaries the following pseudo keys are supported:* ***alternateid*** *(Integer; only for* `name=Separation` *and* `DeviceN`*) Index of the underlying alternate color space in the* `colorspaces[]` *pseudo object.* ***baseid*** *(Integer; only for* `name=Indexed`*) Index of the underlying base color space in the* `colorspaces[]` *pseudo object.* ***colorantname*** *(Name; only for* `name=Separation`*) Name of the colorant. Non-ASCII CJK color names will be converted to Unicode.* ***colorantnames*** *(Array of names; only for* `name=DeviceN`*) Names of the colorants* ***components*** *(Integer) Number of components of the color space* ***name*** *(String) Name of the color space:* `CalGray`, `CalRGB`, `DeviceCMYK`, `DeviceGray`, `DeviceN`, `DeviceRGB`, `ICCBased`, `Indexed`, `Lab`, `Separation` ***csarray*** *(Array; not for* `name=DeviceGray/RGB/CMYK`*) Array describing the underlying native color space, i.e. the original color space object in the PDF.* *Color space resources will include all color spaces which are referenced from any type of object, including the color spaces which do not require native PDF resources (i.e.* `DeviceGray`, `DeviceRGB`, *and* `DeviceCMYK`*).*
extgstates	*(Array of dicts) Array containing the dictionaries for all extended graphics states (ExtGStates) on the page or in the document*
fonts	*(Array of dicts) Array containing dictionaries for all fonts on the page or in the document. In addition to the standard PDF keys in font dictionaries, the following pseudo keys are supported:* ***name*** *(String) PDF name of the font without any subset prefix. Non-ASCII CJK font names will be converted to Unicode.* ***embedded*** *(Boolean) Embedding status of the font* ***fullname*** *(String; pCOS interface 5) PDF name of the font including subset prefix if present. Non-ASCII CJK font names will be converted to Unicode.* ***type*** *(String) Font type:* `(unknown)`, `Composite`, `Multiple Master`, `OpenType`, `TrueType`, `TrueType (CID)`, `Type 1`, `Type 1 (CID)`, `Type 1 CFF`, `Type 1 CFF (CID)`, `Type 3` ***vertical*** *(Boolean)* `true` *for fonts with vertical writing mode,* `false` *otherwise*

Table 4.8 Pseudo objects for resources; each resource category P creates two resource arrays `P[ ]` *and* `pages[ ]/P[ ]`.

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

4.8 Protected PDF Documents and pCOS Mode

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

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

Restricted pCOS mode (mode 1). If the document has been opened without the appropriate master password and does not require a user password (or only the user password has been supplied) objects with type *string*, *stream*, or *fstream* can not be retrieved. As an exception, if extraction of page contents is allowed, i.e. if *nocopy=false* the objects listed in Table 4.9 are also accessible.

Table 4.9 Objects which are accessible in restricted pCOS mode if text extraction is allowed, i.e. if `nocopy=false`

object	*pCOS path*
document metadata[1]	`/Root/Metadata` *(XMP Metadata)* `/Info/*` *(document info fields)*
bookmarks	`bookmarks[...]/Title`
annotation contents	`pages[...]/annots[...]/Contents`

1. These objects can also be retrieved if `plainmetadata=true`

Minimum pCOS mode (mode 0). Regardless of the encryption status and the availability of passwords, the universal pCOS pseudo objects listed in Table 4.2, Table 4.3, and Table 4.4 are always available. For example, the *encrypt* pseudo object can be used to query a document's encryption status. Encrypted objects can not be retrieved in minimum pCOS mode.

Summary of password combinations. Table 4.10 lists the resulting pCOS modes for various password combinations. Depending on the document's encryption status and the password supplied when opening the file, PDF object paths may be available in minimum, restricted, or full pCOS mode. Trying to retrieve a pCOS path which is inappropriate for the respective mode will trigger an exception.

Table 4.10 Resulting pCOS modes for various password combinations

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

A pCOS Function Reference

The following table contains an overview of the pCOS API functions. Please refer to the corresponding product manual for more details and information for specific programming languages.

pCOS function prototypes

double pcos_get_number(int doc, String path)

String pcos_get_string(int doc, String path)

final byte[ ] pcos_get_stream(int doc, String optlist, String path)

B Revision History

Revision history of this manual

Date	*Changes*
November 29, 2010	► *Republished edition for pCOS interface 5 for PDFlib 8.0.2*
October 29, 2010	► *Updates for pCOS interface 7 in pCOS 3.0*
July 22, 2010	► *Reorganized the reference for pCOS interface 6 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*

Index

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)*