PDFlib GmbH

pCOS Path Reference

PDF Information Retrieval Tool

pCOS Interface Version 11

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
groups.yahoo.com/neo/groups/pdflib/info

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 32000-1. 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.

Mini sample 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 / Acrobat 8 Identical to ISO 32000-1	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
6	PDF 1.7 extension level 3 / Acrobat 9	TET 4.0, TET PDF IFilter 4.0
7	PDF 1.7 extension level 8 / Acrobat X Syntax and encryption method identical to ISO 32000-2, also called PDF 2.0	pCOS 3.0, PLOP 4.1, PDFlib+PDI 8.1, PPS 8.1
8	PDF 1.7 extension level 8 / Acrobat X/XI Syntax and encryption method identical to ISO 32000-2, also called PDF 2.0	TET 4.1+, TET PDF IFilter 4.1+ PDFlib+PDI 9.0/9.1, PPS 9.0/9.1 pCOS 4.0
9	PDF 1.7 extension level 8 / Acrobat X/XI Syntax and encryption method identical to ISO 32000-2, also called PDF 2.0	PLOP 5.0, PLOP DS 5.0
10	PDF 1.7 extension level 8 / Acrobat X/XI/DC	TET 5.0, TET PDF IFilter 5.0
11	PDF 1.7 extension level 8 / Acrobat X/XI/DC including certificate security	PLOP 5.1/5.2, PLOP DS 5.1/5.2 TET 5.1, TET PDF IFilter 5.1

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 program 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, number, 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_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_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 program 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 emits 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 4.10, »Protected PDF Documents and pCOS Mode«, page 39)*
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, pdfua, pdfvt, pdfx	*string*	*PDF/A, PDF/E, PDF/UA, PDF/VT 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 an exception when an attempt is made at retrieving information for which no access is granted (e.g. because the document 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_document(filename, "requiredmode=minimum");
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 suitable password or digital ID 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_document(doc);
```

PDF version. The following code fragment emits 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, PDF/UA, PDF/VT 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/UA status: " + p.pcos_get_string(doc, "pdfua"));
System.out.println("PDF/VT status: " + p.pcos_get_string(doc, "pdfvt"));
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");
```

Transparency. Page transparency may be relevant for printing and other processes. You can identify pages with transparent elements with the *usespagetransparency* key of the *pages* pseudo object:

```
pagenum = 0;        // page 1 (0-based)
if (p.pcos_get_number(doc, "pages[" + pagenum + "]/usespagetransparency"))
{
        ...page contains transparent elements...
}
```

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:

```
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 ICC Profiles

Table 2.5 lists some pCOS paths for objects related to ICC profiles. See Table 4.9 for the full list.

Table 2.5 pCOS paths for properties related to ICC profiles

pCOS path	*type*	*explanation*
length:iccprofiles	*number*	*number of ICC profiles in the document (output intent, color spaces and transparency groups)*
iccprofiles[...]/profilename	*string*	*Internal name of the ICC profile*
iccprofiles[...]/deviceclass	*string*	*Device class of the ICC profile (display device, output device, etc.)*

Listing all ICC profiles. You can report details about all ICC profiles in the document as follows:

```
count = (int) tet.pcos_get_number(doc, "length:iccprofiles");
for (int i = 0; i < count; i++)
{
        System.out.print("profile " + i + ":");

        errmsg = tet.pcos_get_string(doc, "iccprofiles[" + i + "]/errormessage");

        /* Check for ICC profile parsing problems */
        if (!errmsg.equals(""))
        {
            System.out.println(" " + errmsg);
            continue;
        }

        System.out.print(" name='" +
            tet.pcos_get_string(doc, "iccprofiles[" + i + "]/profilename") + "',");

        System.out.print(" cs='" +
            tet.pcos_get_string(doc, "iccprofiles[" + i + "]/profilecs") + "',");

        System.out.print(" class='" +
            tet.pcos_get_string(doc, "iccprofiles[" + i + "]/deviceclass") + "',");

        System.out.print(" version=" +
            tet.pcos_get_string(doc, "iccprofiles[" + i + "]/iccversion"));

        System.out.println();
}
```

The output looks similar to the following:

```
profile 0: name='ISO Coated v2 300% (ECI)', cs='CMYK', class='prtr', version=2.0
profile 1: name='eciRGB v2', cs='RGB ', class='mntr', version=2.4
profile 2: name='Adobe RGB (1998)', cs='RGB ', class='mntr', version=2.1
profile 3: name='sRGB IEC61966-2.1', cs='RGB ', class='mntr', version=2.1
profile 4: name='PSO Uncoated ISO12647 (ECI)', cs='CMYK', class='prtr', version=2.4
```

2.7 Interactive Elements

Table 2.6 lists pCOS paths for objects related to interactive elements.

Table 2.6 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*
length:fields	*number*	*number of form fields in the document*

Bookmarks. The following code fragment queries the 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:

```
int linearized_i = p.pcos_get_number(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:

```
string linearized_s = p.pcos_get_string(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 uncom-

pressed. 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 JBIG2 stream compression filter.

If there is an 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 are not 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( )*.

Stream objects are implicitly also dictionary objects, i.e. the entries in the dictionary which is associated with the PDF stream object can be queried by appending the dictionary key (e.g. */Length)* to the pCOS path of the stream object.

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. Since an array may contain nested arrays multi-dimensional data structures can be represented as well.

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 dictionaries in the *pages[ ]* pseudo object. 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:

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

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);
        System.out.print(key + ": ");
```

```
            /* 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. This improves the 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. The entries in 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. pCOS paths start with an entry in the document's Trailer dictionary or some artificial object, called pseudo object, added by pCOS to simplify access to various data structures (e.g. pages). 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 include 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 include an index 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 includes an index 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 ASCII code.
- In Unicode-aware language bindings the path consists of a 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 indexes are only applicable to plain PDF objects and pseudo objects of type *array*, but not any pseudo objects of type dictionary. 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 dictionary (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	*(Number or string) Type of the object as number or string:* ***0, null*** *Null object or object not present (use to check existence of an object). pCOS interface 9: This value is also returned if a PDF syntax error was encountered while trying to access the object specified in the path.* ***1, boolean*** *Boolean object* ***2, number*** *Integer or floating point 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 object types are available for the convenience of C and C++ developers* (`pcos_ot_null` *etc.).*

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	*(Dictionary) Dictionary with keys describing the encryption status of the document:*
	length *(Number) Length of the file encryption key in bits*
	algorithm *(Number)*
	description *(String) pCOS 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)*
	6 *Public key on top of 128-bit AES (Acrobat 7)*
	7 *Adobe Policy Server (Acrobat 7)*[1]
	8 *Adobe Digital Editions (EBX)*[1]
	9 *256-bit AES (Acrobat 9)*
	10 *Public key on top of 256-bit AES (Acrobat 9)*
	11 *256-bit AES (Acrobat X/XI/DC)*
	12 *(pCOS interface 11) Public key on top of 128-bit RC4 (Acrobat 6)*
	master *(Boolean)* `True` *if the document is password-protected and requires a master password to change security settings (permissions, user or master password),* `false` *otherwise*
	user *(Boolean)* `True` *if the document is password-protected and requires a user password for opening,* `false` *otherwise*
	attachment *(Boolean; pCOS interface 8)* `True` *if the document requires a password for extracting attachments (but not for opening),* `false` *otherwise*
	noaccessible, noannots, noassemble, nocopy, noforms, nohiresprint, nomodify, noprint *(Boolean)* `True` *if the respective access protection is set,* `false` *otherwise. In full pCOS mode all values are* `false`**;** *in pCOS minimum mode all values are* `true`*.* *(pCOS interface 11) Certificate security: These values apply to the recipient digital ID which has been supplied for opening the document and all other recipients in the same group.*
	plainmetadata *(Boolean)* `True` *if the document is encrypted with password or certificate security, but contains unencrypted metadata,* `false` *otherwise*
	recipients *(Array of strings; pCOS interface 11) Each string contains a CMS object with encrypted keys for a group of one or more recipients with identical permissions. If* `length:encrypt/recipients` *is different from 0 the document is encrypted with certificate security (algorithm 5, 6, 10, or 12) and requires a suitable digital ID for opening in restricted or full pCOS mode. Since each CMS object may contain one or more recipients the array length does not necessarily indicate the total number of recipients.*
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*

Table 4.2 Universal pseudo objects for general document information

object name	explanation
pcosmode **pcosmode-name**	*(Number or string) pCOS mode as number or string:* ***0, minimum*** ***1, restricted*** ***2, full***
revisions	*(Number; pCOS interface 9) Number of document revisions included in the PDF, where each revision is described by an incremental PDF update section. If a document contains multiple signatures each signature is applied in a separate update section, but an update section may also contain other changes, e.g. annotations added or deleted, form fields filled in, etc. The first signature does not necessarily add an incremental PDF update.*
shrug	*(Boolean; only in the products TET, PDFlib+PDI, PPS, PLOP, PLOP DS)* `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. The value is* `true` *if the following conditions are true:* ► *Shrug mode has been enabled with the* `shrug` *option.* ► *Password security: the document has a master password but this has not been supplied, and the user password (if required for the document) has been supplied.* ► *(pCOS interface 11) Certificate security: a suitable recipient digital ID has been supplied for opening the document, but the document does not set master permission for this ID.* ► *TET product only: 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
extension-level	*(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/XI/DC create extension level 8.*
fullpdf-version	*(Number) Numerical value for the PDF version number as* `100 * BaseVersion + ExtensionLevel`*, 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*** *PDF 1.7 Adobe Extension Level 8 (Acrobat X/XI/DC)* ***200*** *PDF 2.0 = ISO 32000-2*
pdfversion	*(Number) PDF version number multiplied by 10, e.g. 17 for PDF 1.7*
pdfversion-string	*(String) Full PDF version string in the form expected by various API functions for setting the PDF output compatibility, e.g.* `1.5`, `1.6`, `1.7`, `1.7ext3`, `1.7ext8`, `2.0`

In pCOS minimum mode (i.e. a required user password is not available for an encrypted file) the *ExtensionLevel* may be missing from the version information (e.g. 1.7 is reported instead of 1.7ext3) because information about the Extension Level cannot be decrypted.

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 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:* none PDF/A-1a:2005, PDF/A-1b:2005 PDF/A-2a, PDF/A-2b, PDF/A-2u PDF/A-3a, PDF/A-3b, PDF/A-3u *(pCOS interface 8)*
pdfe	*(String) PDF/E (ISO 24517-1 and 24517-2) conformance level of the document. Possible values:* none PDF/E-1 PDF/E-2
pdfua	*(String) PDF/UA (ISO 14289) conformance level of the document. Possible values:* none PDF/UA-1
pdfvt	*(String) PDF/VT (ISO 16612-2) conformance level of the document. Possible values:* none PDF/VT-1 PDF/VT-2
pdfx	*(String) PDF/X (ISO 15930-1 etc.) conformance level of the document. Possible values:* 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 dictionaries) 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 dictionaries) Page resources according to Table 4.10.* **annots** *(Array of dictionaries) 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 or a* `GoTo` *action is present) Number of the target page (first page is 1)* **blocks** *(Dictionary of dictionaries, or array of dictionaries) Shorthand for* `pages[]/PieceInfo/PDFlib/Private/Blocks`*, i.e. the page's list of Blocks for use with PDFlib Personalization Server (PPS). 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.* *An individual Block in the* `blocks` *pseudo object can be addressed numerically or via its name. For example, assuming Block 5 (remember that Block indices are 0-based) has the name* `zipcode` *it can be addressed as* `pages[...]/blocks[5]` *or* `pages[...]/blocks/zipcode`**.** **height** *(Number) Height of the page in points. The MediaBox or the CropBox (if present) will be used to determine the height. Rotate entries will also be applied.* **fields** *(Array of dictionaries; pCOS interface 9) Array with dictionaries for the form fields on the page. The same pseudo dictionary keys as for the global* `fields[]` *array are supported (see Table 4.7, page 31). For signature fields the same pseudo dictionary keys as for the* `signaturefields[]` *array are also supported (Table 4.8, page 33).* **isempty** *(Boolean)* `True` *if the page is 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.* **usespagetransparency**[1] *(Boolean; pCOS interface 8)* `True` *if the page contents include any transparent elements,* `false` *otherwise.* **usesanytransparency**[1] *(Boolean; pCOS interface 8)* `True` *if the page contents or any annotation on the page includes any transparent elements,* `false` *otherwise.* **width** *(Number) Width of the page in points (same rules as for* `height`*)* *The following entries will be inherited:* `CropBox`*,* `MediaBox`*,* `Resources`*,* `Rotate`*.*

1. These checks report transparency found in the resources of a page (e.g. Form XObjects, images), regardless of whether or not these resources are actually used for creating visible page content. Transparency is defined as in the PDF/VT standard.

4.6 Pseudo Objects for PDF Objects and interactive Elements

Table 4.7 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 dictionaries) 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 dictionaries) Bead directory with the standard PDF keys, plus the following:* **destpage** *(Number) Number of the target page (first page is 1)*
bookmarks	*(Array of dictionaries) 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:* **destpage** *(Number; only if a* `Dest` *entry or a* `GoTo` *action is present) Number of the target page (first page is 1) if the bookmark contains a destination or* `GoTo` *action which points to a page in the same document, -1 otherwise.* **level** *(Number) Indentation level in the bookmark hierarchy*
destpage	*(Number; pCOS interface 9) Number of the target page (first page is 1) which is displayed when the document is opened. The value is taken from the document's open action or destination if present, otherwise 1*
fields	*(Array of dictionaries) Array containing the form field dictionaries for the document. In addition to the standard PDF keys in the field dictionary and the entries in the associated Widget annotation dictionary pCOS supports the following pseudo keys for dictionaries in the* `fields` *array:* **exportvalue**[1] *(String; pCOS interface 9) Export value of the field* **fullname** *(String) Complete name of the form field. For unnamed fields the name of the parent field is used. If there are multiple unnamed siblings on the same level, the name is suffixed with* `#N`*, where* `N` *is a consecutive integer starting at 0.* **level** *(Number) Level in the field hierarchy (determined by ».« as separator)* **parent** *(Number; pCOS interface 9) Index of the field's parent node in the* `fields[]` *array; if the field doesn't have a parent field, this value is -1* **type** *(String; pCOS interface 9) Field type:* `barcode`, `container` *(node in the form tree which does not represent a field on its own, but only serves as a container for other fields which are not radio buttons),* `checkbox`, `combobox`, `listbox`, `pushbutton`, `radiobutton`, `radiogroup` *(container of radio buttons),* `signature`, `textfield` **value**[1] *(Various types; pCOS interface 9) Field value (obtained from the* `V` *key or from the* `Opt` *array for radio buttons and checkboxes)* **visible** *(Boolean; pCOS interface 9)* `True` *if the field is visible.*

Table 4.7 Pseudo objects for PDF objects and interactive elements

object name	*explanation*
names	*(Dictionary) 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 key for dictionaries in the* `Dests` *names tree is 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*

1. This pseudo object is not always available. Its existence must be checked with the `type` *prefix.*

4.7 Pseudo Objects for Signatures

Table 4.8 lists pseudo objects which can be used for retrieving signature-related information.

Table 4.8 Pseudo objects for signatures

object name	*explanation*
signaturefields	*(Array of dictionaries; pCOS interface 9) Array containing all signed and unsigned signature field dictionaries in the document. The array contains all signed fields in the order of signing, and then all unsigned fields. In addition to the entries in the signature field's corresponding entry in the* `fields[ ]` *pseudo object pCOS supports the following pseudo keys for dictionaries in this array:* ***cades*** *(Boolean)* `True` *if the field is signed and contains a CAdES signature, otherwise* `false` ***field*** *(Integer) Index of the corresponding form field in the* `fields[]` *array* ***fillablefields*** *(Boolean; only for* `sigtype=certification`*)* `True` *if the certification signature protects form fields, otherwise* `false` ***permissions*** *(String; only for* `sigtype=certification`*) Document changes which are allowed without invalidating the certification signature:* `nochanges`, `formfilling`, `formsandannotations` ***preventchanges*** *(Boolean; only for* `sigtype=certification`*)* `True` *if the certification signature instructs Acrobat to hide user interface elements which would invalidate the signature if used* ***sigtype*** *(String) Type of signature:* `none` *(for unsigned fields),* `approval`, `certification`, *or* `doctimestamp`
usagerights	*(Boolean; pCOS interface 9)* `True` *if the document contains signed usage rights; such documents are also known as Reader-enabled.*

4.8 Pseudo Objects for ICC Profiles

Table 4.9 lists pseudo objects for embedded and referenced ICC profiles.

Table 4.9 Pseudo objects for embedded and referenced ICC profiles

object name	*explanation*
iccprofiles	*(Array of dictionaries; pCOS interface 10) Array containing all embedded and referenced ICC profiles in the document. The profiles are collected from output intents, color spaces and page transparency groups. The index into this array can be obtained with the following pCOS pseudo objects:* `colorspaces[]/iccprofileid` `pages[]/colorspaces[]/iccprofileid` `/Root/OutputIntents[]/DestOutputProfile/iccprofileid` `/Root/OutputIntents[]/DestOutputProfileRef/iccprofileid` *pCOS supports the following keys for dictionaries in this array:* ***checksum***[1] *(String) Hexadecimal representation of the MD5 checksum of the profile according to the ICC 4.2 algorithm. If an external profile is referenced as output intent, the checksum is taken from the PDF entry* `CheckSum` *which is defined as plain MD5 hash, i.e. different from the ICC 4.2 method.* ***deviceclass***[1,2] *(String) Device class of the profile:* `mntr` *(display device),* `prtr` *(output device, such as printer or printing process),* `scnr` *(input device such as scanner or digital camera) or* `spac` *(for converting to a device-independent color space)* ***embedded*** *(Boolean)* `True` *if the profile data is embedded in the PDF,* `false` *for referenced profiles (only for output intent profiles)* ***errormessage*** *(String) Text of the exception which may have occurred while parsing the ICC profile. This string is usually empty, but will be populated if the profile is damaged.* ***fromCIE***[1,2] *(Boolean)* `True` *if the* `BtoA1` *tag is present in the profile. This tag must be present in profiles used for a color space or an output intent.* ***iccversion***[1] *(String) Internal profile version number as string, e.g.* `4.2` ***profilecs***[1] *(String) Internal colorspace of the profile:* `»GRAY«`, `»RGB «`, `»CMYK«`, *or* `»Lab «`. *Note that the trailing space character is not removed. For a referenced PDF/X-5n output intent profile the value* `»xCLR«` *is returned where* `x` *denotes the hexadecimal number of colorants.* ***profilename***[1] *(String) Internal name of the profile* ***toCIE***[1,2] *(Boolean)* `True` *if the* `AtoB1` *tag is present in the profile. This tag must be present in profiles used for a transparency group; it may optionally be present in profiles used for an output intent for print proofing (simulated output device).*

1. Only available if `errormessage` *is empty.*
2. Only available for embedded profiles.

4.9 Pseudo Objects for PDF Resources

PDF 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 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. fonts which are not used on any page).
- In addition to the original PDF dictionary keys pCOS 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 4.10:

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

object name	*explanation*
colorspaces	*(Array of dictionaries; however, for* name=ICCBased *the type is* stream; *in the rare case of device-dependent color spaces as resources the type is* name) *Array containing dictionaries or streams for all color spaces on the page or in the document. Color space resources 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). *In addition to the standard PDF keys in color space dictionaries (if the color space is represented by a dictionary in PDF) 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 *and* Pattern) *Index of the underlying base color space in the* colorspaces[] *pseudo object* ***colorantname*** *(Name; only for* name=Separation) *Name of the spot color. Non-ASCII CJK color names are converted to Unicode.* ***colorantnames*** *(Array of names; only for* name=DeviceN) *Names of the spot colors* ***components*** *(Integer) Number of components of the color space* ***iccprofileid*** *(Integer; pCOS interface 10; only for* name=ICCBased) *Index of the corresponding ICC profile in the* iccprofiles[] *pseudo object* ***name*** *(String) Name of the color space:* CalGray, CalRGB, DeviceCMYK, DeviceGray, DeviceN, DeviceRGB, ICCBased, Indexed, Lab, Pattern, Separation ***csarray*** *(Array; not for* name=DeviceGray/RGB/CMYK) *Array describing the underlying native color space, i.e. the native color space object in the PDF. This pseudo object is not available for JPX-compressed images without any PDF colorspace.*
extgstates	*(Array of dictionaries) Array containing the dictionaries for all extended graphics states (ExtGStates) on the page or in the document*

Table 4.10 Pseudo objects for resources; each resource category P creates two resource arrays `P[ ]` *and* `pages[ ]/P[ ]`.

object name	*explanation*
fonts	*(Array of dictionaries) 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:*
	ascender *(Float) Ascender of the font. Depending on the availability the value will be taken from the FontDescriptor dictionary in PDF, or an estimated value. The value is expressed relative to a font scaling factor of 1000, i.e. 1000 units refer to the full fontsize.* *TET product: in addition to dictionary values in PDF, embedded fonts and fonts installed on the Mac or Windows system will be parsed in order to determine font metrics values. Results of font parsing are only available after calling TET_get_char_info( ) with a glyph in this particular font. In other words, using font ids returned by TET_get_char_info( ) is safe, while enumerating all fonts in the* `fonts[]` *array does not necessarily provide metrics values from embedded font data, but the possibly inaccurate values from the PDF font descriptor.*
	capheight *(Float) Capheight of the font; see* `ascender`
	italicangle *(Float) Italic (slant) angle of the font in degrees*
	name *(String) PDF name of the font without any subset prefix. Non-ASCII CJK font names will be converted to Unicode.*
	descender *(Float) Descender of the font; see* `ascender`
	embedded *(Boolean) Embedding status of the font*
	fullname *(String) 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*
	weight *(Float) Font weight in the range 0...900: 0=no information available, 400=normal, 700=bold*
	xheight *(Float) X height of the font; see* `ascender`

Table 4.10 Pseudo objects for resources; each resource category P creates two resource arrays P[ ] *and* pages[ ]/P[ ].

object name	*explanation*
images	*(Array of streams) 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 the PDF key* BitsPerComponent*, but unlike this it is guaranteed to be available (in particular, image masks may not contain this key). For JPX-compressed images the* bpc *value is derived from the compressed data and may therefore report values outside the set of values which are allowed in PDF image dictionaries (i.e. 1/2/4/8/16).* ***colorspaceid*** *(Integer) Index of the image's color space in the* colorspaces[ ] *pseudo object. This can be used to retrieve detailed color space properties. If no PDF colorspace is present for a JPX-compressed image, the internal JPEG 2000 colorspace is reported. The reported colorspace may be inaccurate since ICC profiles are ignored for JPX images. Internal JPEG 2000 colorspaces are mapped to one of Lab/DeviceGray/RGB/CMYK.* *For mask images the id of* DeviceGray *is returned.* ***filterinfo*** *(Dictionary) 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*** *(Dictionary) The* DecodeParms *dictionary if one is present for the filter* ***maskid*** *(Integer; pCOS interface 9; only available in TET) Index of the image's mask in the* images[ ] *pseudo object if the image has a* Mask *or* SMask *entry; otherwise -1. This can be used to identify masked images and to retrieve properties of the mask.* ***mergetype*** *(Integer; only in TET) 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.* ***small*** *(Boolean; pCOS interface 10; only available in TET) Describes whether the image has been identified as small by the small image filter. The decision depends on TET options.* ***stencilmask*** *(Boolean; pCOS interface 9; only available in TET) The image's stencil mask flag. This is the same as the PDF key* ImageMask *if present, but unlike this it is guaranteed to be available.*
patterns	*(Array of dictionaries) Array containing dictionaries for all patterns on the page or in the document*
properties	*(Array of dictionaries) Array containing dictionaries for all properties on the page or in the document*
shadings	*(Array of dictionaries) 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 streams) Array containing dictionaries for all templates (Form XObjects) on the page or in the document*

4.10 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 or digital ID to be supplied when opening the document. Depending on the availability of these credentials encrypted documents can be processed in one of the pCOS modes described below.

Full pCOS mode (mode 2). Unencrypted documents are always opened in full pCOS mode. Documents protected with password security can be processed without any restriction if the master password has been supplied upon opening the file. All objects are returned unencrypted.

If an unencrypted document contains encrypted file attachments, but the attachment password has not been supplied, retrieving the following pCOS paths (i.e. the attachment contents) results in an empty return value (in C and C++: NULL):

```
pages[...]/annots[...]/FS/EF/F
names/EmbeddedFiles[...]/EF/F
```

(pCOS interface 11) Documents protected with certificate security are opened in full pCOS mode if a suitable digital ID has been supplied and the document sets master permission for this ID.

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* cannot be retrieved. As an exception, if extraction of page contents is allowed, i.e. if *nocopy=false* the objects listed in Table 4.11 are also accessible.

Table 4.11 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)* `/Root/Lang` *(pCOS interface 8)* `/Info/*` *(document info fields)*
bookmarks	`bookmarks[...]/Title`
annotation contents	*all paths starting with* `pages[...]/annots[...]`
document-level file attachments	*all paths starting with* `names/EmbeddedFiles[...]`

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

(pCOS interface 11) Documents protected with certificate security are opened in restricted pCOS mode if a suitable digital ID has been supplied and the document does not set master permission for this ID.

Minimum pCOS mode (mode 0). Regardless of encryption status and 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 doc-

ument's encryption status. Encrypted objects can not be retrieved in minimum pCOS mode.

(pCOS interface 11) Documents protected with certificate security are opened in minimum pCOS mode if no suitable digital ID has been supplied, i.e. no digital ID with a private key corresponding to one of the recipients' public keys.

In pCOS minimum mode the *ExtensionLevel* may be missing from the version information reported by the *extensionlevel, fullpdfversion, pdfversion,* and *pdfversionstring* pseudo objects (e.g. 1.7 is reported instead of 1.7ext3) or the reported version may be too low because information about the Extension Level cannot be decrypted.

Summary of password combinations. Table 4.12 lists the resulting pCOS modes for protected documents and various combinations of available credentials. Depending on the document's encryption status and the credentials 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 triggers an exception.

Table 4.12 Resulting pCOS modes for protected documents and various combinations of available credentials

If you know or have...	***...pCOS will run in...***
password security	
none of the passwords	► *document requires user password: minimum pCOS mode* ► *document does not require user password: restricted pCOS mode* ► *document contains encrypted file attachments: full pCOS mode, but attachments can not be retrieved*
user password	*restricted pCOS mode*
master password, or attachment password for documents with encrypted file attachments	*full pCOS mode*
certificate security	
no suitable digital ID	*minimum pCOS mode*
a suitable digital ID for which the document does not set master permission	*restricted pCOS mode*
a suitable digital ID for which the document sets master permission	*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

API function
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*
May 22, 2017	► *Republished for TET 5.1 and TET PDF IFilter 5.1*
February 15, 2017	► *Republished for PLOP/PLOP DS 5.2*
April 12, 2016	► *Updates for pCOS interface 11 in PLOP/PLOP DS 5.1 (certificate security)*
October 23, 2015	► *Updates for pCOS interface 10 in TET 5.0 and TET PDF IFilter 5.0*
December 04, 2014	► *Updates for pCOS interface 9 in PLOP/PLOP DS 5.0*
August 1, 2013	► *Bundled with pCOS 4.0; no major changes in content*
May 16, 2013	► *Bundled with TET 4.2 and TET PDF IFilter 4.2; no major changes in content*
March 14, 2013	► *Updates for PDFlib 9.0.0*
February 13, 2012	► *Updated to pCOS interface 8*
March 04, 2011	► *Mentioned PLOP 4.1 for pCOS interface 7*
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 groups.yahoo.com/neo/groups/pdflib/info

Licensing contact
sales@pdflib.com

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