



SetaPDF- MetaManager API

Manual and Reference

Version 1.1.1, 2010-05-31 10:21:14

Setasign - Jan Slabon
Major-Hirst-Straße 11
38442 Wolfsburg
Germany

<http://www.setasign.de>
support@setasign.de

Table of contents

Introduction	3
System Requirements	4
Installation.....	5
Ioncube	6
Zend.....	7
Constants / Configuration	8
Caching.....	12
SetaPDF	15
SetaPDF::isError()	16
SetaPDF_Error	17
SetaPDF_Parser.....	18
SetaPDF_Parser::cacheDir()	19
SetaPDF_Parser::cacheFlags()	20
SetaPDF_Parser::cacheMkdirMode()	21
SetaPDF_Parser::cacheNoOfObjectsPerInstance()	22
SetaPDF_Parser::cacheHashFunction()	23
SetaPDF_MetaManager	24
SetaPDF_MetaManager::factory()	25
SetaPDF_MetaManager::__destruct()	26
SetaPDF_MetaManager::setUseUpdate()	27
SetaPDF_MetaManager::getEncryption().....	28
SetaPDF_MetaManager::getPageCount().....	29
SetaPDF_MetaManager::getPageData().....	30
SetaPDF_MetaManager::setPageData()	32
SetaPDF_MetaManager::getInfoData()	34
SetaPDF_MetaManager::setInfoData()	35
SetaPDF_MetaManager::getDocumentMetadata()	37
SetaPDF_MetaManager::setDocumentMetadata()	38
SetaPDF_MetaManager::getDocumentIds()	39
SetaPDF_MetaManager::output().....	40

SetaPDF-MetaManager API - Introduction

The SetaPDF-MetaManager API allows you to manage information and metadata in PDF documents.

SetaPDF-MetaManager API - System Requirements

All SetaPDF APIs are written in pure PHP and does not need any other libraries installed except a PHP environment of a version later than 4.3 (until 2010) and an installed Zend Optimizer or installed Ioncube loader.

All releases since 2010 require PHP 5.

As shown in the next paragraph, it is also recommended to install MCrypt.

The SetaPDF APIs have their own integrated RC4 function for encrypting and decrypting the contents of a PDF file. For performance reasons, the APIs initially tries to use the MCrypt library, if that is installed. If MCrypt is available, the APIs require the arcfour algorithm.

The use of MCrypt increases the performance by up to 90% if encryption or decryption is needed.

If MCrypt is not available, the APIs automatically fall back to their internal RC4 function.

Depending on the file size of the PDF files to be processed, some adjustment to the php.ini directives `max_execution_time` and `memory_limit` are recommended.

For performance optimization, all SetaPDF APIs provides a caching system that prevents the unnecessary reparsing of PDF files.

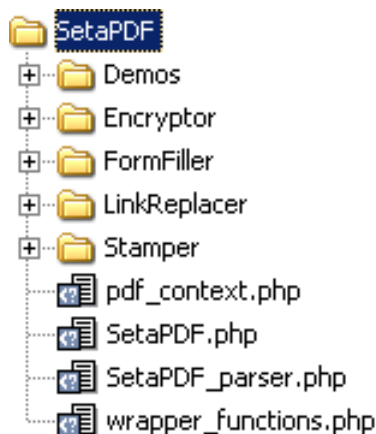
The SetaPDF-MetaManager API did not offers such a caching system.

SetaPDF-MetaManager API - Installation

The SetaPDF API collection includes a directory structure which should be kept, because of the internal usage of paths.

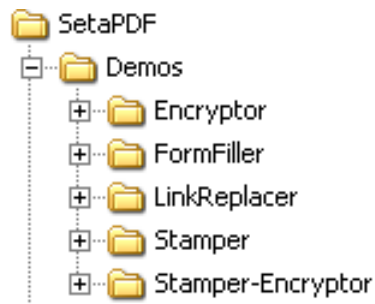
Files and directories

All packages includes a root directory called *SetaPDF*. In this directory you'll find the desired API directories. The directory structure for all current available SetaPDF APIs looks like this:



If you transfer the files via FTP make sure you use binary mode.

For each API or API combination you'll find demo files in the /Demo directory in nearly the same structure:



To use one of the SetaPDF APIs in your applications you have to add the SetaPDF-directory to your `include_path`:

```
set_include_path(get_include_path() . PATH_SEPARATOR . 'PathTo/SetaPDF/');
```

Now you can simply include the SetaPDF-MetaManager API with the following line:

```
require_once( "MetaManager/SetaPDF_MetaManager.php" );
```

SetaPDF-MetaManager API - Ioncube encoded package

If you own a package of the API, which is encoded with Ioncube you need a loader installed on your server. There are 2 ways to get ioncube encoded files to run:

1. Install the loader in your php.ini
2. Load the loader at runtime

For details how to install ioncube or simply to check if it is installed, just download the loaders from <http://www.ioncube.com/loaders.php>, extract its content to /SetaPDF/ioncube and open the file ioncube-loader-helper.php in the directory /SetaPDF/ioncube in your webbrowser and follow the instructions. For further instructions go to <http://www.ioncube.com/>

Licensing with Ioncube

Each ioncubed package needs a valid license to run. The provided licensefiles for the SetaPDF API are named: **.htSetaPDF-<API-NAME>.icl**

You don't have to rename that file, because the package search for exactly that named file in one of its upper directories. All APIs first searches for this file in the their initial directory. F.g. The SetaPDF-Encryptor API searches first in /SetaPDF/Encryptor/. If the licensefile is not found it goes one directory upwards: /SetaPDF/ and so on...

Please notice that the filename is prefixed with .ht. Some systems hide such prefixed files automatically.

SetaPDF-MetaManager API - Zend encoded package

If you own a package, which is encoded with the Zend Safeguard Suite you have to install the Zend Optimizer. For more information please go to http://www.zend.com/products/zend_optimizer.

Licensing with Zend

Also the zend encoded packages need valid licenses to run. The provided licensefiles for the SetaPDF API are named:

.htSetaPDF-<API-NAME>.z1

Please notice that the filename is also prefixed with .ht. Some systems hide such prefixed files automatically.

For zend encoded packages the name of the license file can be changed and has no real meaning. You can load the licensefile dynamically on runtime in your php script ([zend_loader_install_license\(\)](#)) or by default in your [php.ini](#).

Zended packages are only available for development- and serverlicenses.

SetaPDF-MetaManager API - Constants / Configuration

The API needs some constants which are hard coded into the API.

Also you can define global variables which affects the behaviour of specific tasks.

Global Configuration Variables

```
$GLOBALS[ 'SETAPDF_PARSE_INVALID_FILES' ] (boolean)
```

(DEPRECATED) If this global variable is set the pdf parser tries to read/repair invalid PDF documents. This setting could affect the processtime on huge files very much.

This variable isn't used by the parser as of version 1.3 (all current version of the SetaPDF APIs)

```
$GLOBALS[ 'SETAPDF_SEARCH_FOR_XREF_OFFSET' ] (integer)
```

With this global variable you can adjust the offset position from which the pdf parser should search for the pointer to the xref table. If not defined the default value of 1500 is used.

The pdf specification says it has to be in the last 1024 bytes of a file. But sometimes there are errorious document in the wild that have some garbage at the end so we need the possibility to do a kind of finetuning for them.

Predefined Version Constants

The following constants defines the versions of specific files of the SetaPDF core:

```
SETAPDF_CORE_VERSION (string)1.3
```

Version of the abstract SetaPDF class. Defined in /SetaPDF/SetaPDF.php

```
SETAPDF_PARSER_VERSION (string)1.3
```

Version of the SetaPDF_Parser class. Defined in /SetaPDF/SetaPDF_parser.php

```
SETAPDF_PDF_CONTEXT_VERSION (string)1.3
```

Version of the pdf_context class. Defined in /SetaPDF/pdf_context.php

```
SETAPDF_WRAPPER_FUNCTIONS_VERSION (string)1.2.1
```

Version of the wrapper functions file. Defined in /SetaPDF/wrapper_functions.php

Predefined Constants for Errorhandling

Possible errorcodes for the SetaPDF main class starts at 1 and ends at 99.

```
E_SETAPDF_CANNOT_OPEN_FILE (integer)1
```

cannot open XXXX !

E_SETAPDF_UNABLE_TO_POINT_TO_XREF_TABLE (integer)2

Unable to find pointer to xref table

E_SETAPDF_UNABLE_TO_FIND_XREF (integer)3

Unable to find xref table - Maybe a Problem with 'auto_detect_line_endings'

E_SETAPDF_UNEXPECTED_HEADER_IN_XREF_TABLE (integer)4

Unexpected header in xref table

E_SETAPDF_UNEXPECTED_DATA_IN_XREF_TABLE (integer)5

Unexpected data in xref table

E_SETAPDF_FILE_IS_ENCRYPTED (integer)6

File is encrypted!

E_SETAPDF_WRONG_TYPE (integer)7

Wrong Type of Element

E_SETAPDF_UNABLE_TO_FIND_OBJECT (integer)8

Unable to find object at expected location

E_SETAPDF_ENC_UNSUPPORTED_FILTER (integer)9

E_SETAPDF_ENC_UNSUPPORTED_ALGO (integer)10

E_SETAPDF_ENC_UNSUPPORTED_REVISION (integer)11

E_SETAPDF_ENC_NO_RIGHTS_FOR_SPECIFIC_ACTION (integer)12

E_SETAPDF_ENC_WRONG_OWNER_PW (integer)13

E_SETAPDF_CANNOT_COPY_FILE (integer)14

Cannot copy file XXXX to YYYY

E_SETAPDF_HEADER_ALREADY_SEND (integer)15

Some data has already been output to browser, can't send PDF file

E_SETAPDF_UNABLE_TO_FIND_TRAILER (integer)16

Trailer keyword not found after xref table

E_SETAPDF_UNSUPPORTED_FILTER (integer)17

An unsupported compression filter is required.

E_SETAPDF_ZLIB_REQUIRED (integer)18

To handle /FlateDecode filter, php with zlib support is needed.

E_SETAPDF_DECOMPRESSION_ERROR (integer)19

Error while decompressing stream.

E_SETAPDF_UNABLE_TO_CREATE_CACHE_DIR (integer)20

Unable to create directories in cache directory.

API Related Predefined Constants for Errorhandling

Possible errorcodes for the SetaPDF-MetaManager API starts at 700 and ends at 799.

E_SETAPDF_MM_NOTHING_TODO (integer)700

You've not changed any metadata, so the API has nothing to do.

E_SETAPDF_MM_WRONG_VALUE (integer)701

The given value is not allowed for the given key.

E_SETAPDF_MM_KEY_NOT_ALLOWED (integer)702

Changes to the given key are not allowed.

E_SETAPDF_MM_DOC_ENC_METHOD_DISABLED (integer)703

Method disabled because the document is encrypted.

E_SETAPDF_MM_PARSER_NO_KIDS (integer)704

Cannot find /Kids in current /Page-Dictionary

E_SETAPDF_MM_WRONG_PAGE_NO (integer)705

Wrong page number.

Constans for Cache Mechanism

The following constants are used to control the behaviour of the caching mechanism of the pdf parser.

SETAPDF_P_CACHE_NO (integer)0x00

Don't read and write cache.

SETAPDF_P_CACHE_READ_XREF (integer)0x01

Try to read the cached xref table.

SETAPDF_P_CACHE_WRITE_XREF (integer)0x02

Write the xref table to cache.

```
SETAPDF_P_CACHE_XREF (integer)0x01 | 0x02
```

Try to read and write the xref table.

```
SETAPDF_P_CACHE_READ_OBJECTS (integer)0x04
```

Try to read cached objects.

```
SETAPDF_P_CACHE_WRITE_OBJECTS (integer)0x08
```

Write read objects to cache.

```
SETAPDF_P_CACHE_OBJECTS (integer)0x04 | 0x08
```

Try to read and write objects to cache.

```
SETAPDF_P_CACHE_ALL (integer)0x01 | 0x02 | 0x04 | 0x08
```

Read and write objects and xref-tables.

SetaPDF-MetaManager API - Caching

PDF parsing and handling can be an expensive task in view of needed cpu-power.

To avoid doing default tasks for a single document a few times the parser class offers a caching mechanism to reduce the overhead and avoid reparsing of PDF documents a few times.

The parser simply saves serialized data in the filesystem and load them back if needed. This data can be used with ANY SetaPDF API. So if for example the [SetaPDF-Merger API](#) creates the cache data, the [SetaPDF-Stamp API](#) can benefit from them.

As of this, the handling of the cache mechanism is done through static methods of the [SetaPDF_Parser class](#). Calls to this methods will change [static variables](#) in their method contexts, so that changes doesn't depend on the object instance but applies to all instances of a parser object. (We used static variable because of compatibility to PHP4)

There are 2 parts that the parser can cache:

1. The Xref Table

This is a kind of table of contents of a PDF document. It includes information about all objects in a document and their byte-offset positions in the document. Often documents include several hundreds or thousands of entries in that table. Further more a PDF document can include more than one xref table, which relays on several updates of a document (incremental updates). But at least all tables have to be processed to get the final state of the document... By caching that data, the parser don't have to reparse the xref table out of the document.

2. Objects

Each entry in the above described xref table points to an object representing specific data, like Images, Fonts, Pages,... If the parser should read such an object it have to go to the desired byte-offset position in the document, known from the xref-table, and have to parse the object token-wise. This process needs several string comparisons and also runs recursive until the object is totally read.

The parser can cache the read objects and use the cached versions at the next situation when it is needed. No byte-position change or parsing of any string is done but simply unserializing the data from the cached data.

Usage

As already written the handling of the cache functionality is done by static methods of the [SetaPDF_Parser class](#).

You can use the static method right after including a desired API like the [SetaPDF-Merger API](#):

```
require_once( 'Merger/SetaPDF_Merger.php' );  
// at this point you can access the SetaPDF_Parser class
```

First of all you have to tell the API where you would like to save the cached data. You have to use the [SetaPDF_Parser::cacheDir\(\)](#)-method for this:

```
SetaPDF_Parser::cacheDir(realpath('../..'/path/for/cached/data/'));
```

Now you were able to activate the caching by calling the [SetaPDF_Parser::cacheFlags\(\)](#)-method with special flags. The flags are predefined in [Constants](#):

```
// Will read and write all data (xref table and objects) from/to cache.
SetaPDF_Parser::cacheFlags(SETAPDF_P_CACHE_ALL);
// Will just read and write the xref table from/to cache.
SetaPDF_Parser::cacheFlags(SETAPDF_P_CACHE_XREF);
// Will just read and write objects from/to cache.
SetaPDF_Parser::cacheFlags(SETAPDF_P_CACHE_OBJECTS);
```

After this the cache is active for all instances of any SetaPDF API.

Furthermore you can do some finetuning:

Build the cache slowly

If you want the cache to be build piecemeal you can use the [SetaPDF_Parser::cacheNoOfObjectsPerInstance\(\)](#)-method to define a maximum of objects to cache in a single script instance. With this method you can avoid performance peaks because the cache writing process, for sure, also needs cpu time.

```
// cache a maximum of 100 objects per script instance
SetaPDF_Parser::cacheNoOfObjectsPerInstance(100);
```

How is a file identified and how you can control it

By default the cache mechanism uses the [md5_file\(\)](#)-function to get an unique file identifier of the document. This file identifier is used as the directoryname in the [cache output directory](#). To give you the possibility to use another method for the fileidentification you can define your own function/method, which will be called when a fileidentifier is needed, with the [SetaPDF_Parser::cacheHashFunction\(\)](#)-method.

An Example: You already have your documents arranged in a database. This data have already unique ids related to the documents local path in your filesystem. As the ids are already known and are unique you should use the ids as a fileidentifier to avoid creating a hash with [md5_file\(\)](#).

Furthermore it is easier for you to manage the cache data, as you can for example delete the cache data if the data in the database table were deleted or changed.

The passed argument is of the pseudo-type [callback](#) and will be used with [call_user_func\(\)](#)-function.

```
function mapFilenameToId($filename) {
    // just pseudo code
    $db = YourDbClass::getInstance();
```

```
$id = $db->getOne("SELECT id FROM documents WHERE filename =  
".$db->quote($filename));  
return $id;  
}  
SetaPDF_Parser::cacheHashFunction('mapFilenameToId');
```

SetaPDF - Class

This class is the base class for nearly all SetaPDF APIs. It offers some public static helper methods.

Class Overview

SetaPDF

Child Classes

▶ [SetaPDF_MetaManager](#)

Methods

▶ [SetaPDF::isError\(\)](#)

SetaPDF::isError()

Description

```
SetaPDF {  
    boolean isError ( mixed $obj )  
}
```

Determines if a variable is a SetaPDF_Error object.

Parameters

\$obj

Variable to check

Return Values

True if *\$obj* is a SetaPDF_Error object

SetaPDF_Error - Class

This class represents an error object thrown by a SetaPDF API. You can get more information about the error by checking the following properties `$obj->message` and `$obj->code`.

You can add your own error handling by defining your own class named `SetaPDF_Error` before you include any SetaPDF-File. The original class looks like this:

```
class SetaPDF_Error {
    var $message;
    var $code;

    function SetaPDF_Error($message = 'unknown error', $code = null,
                           $mode = null, $options = null, $userinfo = null) {
        $this->message = $message;
        $this->code = $code;
    }
}
```

SetaPDF_Parser - Class

The SetaPDF_Parser class is the base class for all individual SetaPDF parser classes. It is for example responsible for reading the xref table or objects of a document.

The SetaPDF_Parser class is an abstract class and *just* offers some static methods which let you control the cache functionality.

Class Overview

SetaPDF_Parser

Methods

- ◆ [SetaPDF_Parser::cacheDir\(\)](#)
- ◆ [SetaPDF_Parser::cacheFlags\(\)](#)
- ◆ [SetaPDF_Parser::cacheMkdirMode\(\)](#)
- ◆ [SetaPDF_Parser::cacheNoOfObjectsPerInstance\(\)](#)
- ◆ [SetaPDF_Parser::cacheHashFunction\(\)](#)

SetaPDF_Parser::cacheDir()

Description

```
SetaPDF_Parser {  
    mixed cacheDir ( [string $dir=null] )  
}
```

Sets the directory for cache data.

This method should be called static.

Parameters

\$dir

Path to the directory where to write the cache data. If *null* the directory will not be changed.

Return Values

The actual path.

SetaPDF_Parser::cacheFlags()

Description

```
SetaPDF_Parser {  
    mixed cacheFlags ( [string $flags=null] )  
}
```

Sets the flags how the parser should handle read and write processes of objects or xref-tables.

This method should be called static.

You can use this flags to do fine tuning of the caching mechanism. The flags can be combined using a bitwise AND (&) operation.

If any flag is set, except `SETAPDF_P_CACHE_NO`, a valid writeable path should be set with [SetaPDF_Parser::cacheDir\(\)](#).

Parameters

\$flags

The parameter defines the caching behaviour of the API. Available values are:

- ▶ `SETAPDF_P_CACHE_NO` - Don't read and write cache.
- ▶ `SETAPDF_P_CACHE_READ_XREF` - Try to read the cached xref table.
- ▶ `SETAPDF_P_CACHE_WRITE_XREF` - Write the xref table to cache.
- ▶ `SETAPDF_P_CACHE_XREF` - Try to read and write the xref table.
- ▶ `SETAPDF_P_CACHE_READ_OBJECTS` - Try to read cached objects.
- ▶ `SETAPDF_P_CACHE_WRITE_OBJECTS` - Write read objects to cache.
- ▶ `SETAPDF_P_CACHE_OBJECTS` - Try to read and write objects to cache.
- ▶ `SETAPDF_P_CACHE_ALL` - Read and write objects and xref-tables.

Return Value (see also [Constants / Configurations](#))

The actual value.

SetaPDF_Parser::cacheMkdirMode()

Description

```
SetaPDF_Parser {  
    mixed cacheMkdirMode ( [integer $mode=null] )  
}
```

As the caching mechanism creates directories for each pdf document the API internally uses mkdir to create the directory. With this method you can define if and which parameter should be passed as the \$mode parameter of the [mkdir](#)-function.

This method should be called static.

Parameters

\$mode

The file mode.

The parameter consists of three octal number components specifying access restrictions for the owner, the user group in which the owner is in, and to everybody else in this order. More informations about the mode-parameter can be found [here](#).

Return Values

The actual value.

SetaPDF_Parser::cacheNoOfObjectsPerInstance()

Description

```
SetaPDF_Parser {  
    mixed cacheNoOfObjectsPerInstance ( [integer $no=null] )  
}
```

For sure a caching process needs more process power as the cached data have to be written to the file system. Often a PDF document is build with more hundres or thousands of objects which can increase the process time to a bad value.

With this method you can define how many maximum objects should be cached per script instance. So you can chop the cache creation over several script executions.

This method should be called static.

If you set the \$no-parameter, for example, to 100, the parser will cache 100 objects per script instance maximum, until all objects are cached.

By default the parser will cache ALL objects.

Parameters

\$no

The maximum number of objects to cache per instance.

Return Values

The actual value.

SetaPDF_Parser::cacheHashFunction()

Description

```
SetaPDF_Parser {  
    mixed cacheHashFunction ( [callback $hashFunction=null] )  
}
```

To identify a pdf document the API uses the [md5_file\(\)](#)-function by default.

If you want to create your own identification process or if you already know a hash or unique property of the document you can use this method to define an own function/method which will be called when the parser needs the hash.

This hash/value will be used as the directory name in the cache directory (see [SetaPDF_Parser::cacheDir\(\)](#)).

The given value will be used as the function parameter of a [call_user_func\(\)](#)-call.

This method should be called static.

Parameters

\$hashFunction

The function to be called.
(See also informations about the [callback](#) type.)

Return Values

The actual value.

SetaPDF-MainManager - Main class

This is the main class of the SetaPDF-MetaManager API.

Class Overview



Methods

- ▶ [SetaPDF_MetaManager::factory\(\)](#)
- ▶ [SetaPDF_MetaManager::destruct\(\)](#)
- ▶ [SetaPDF_MetaManager::setUseUpdate\(\)](#)
- ▶ [SetaPDF_MetaManager::getEncryption\(\)](#)
- ▶ [SetaPDF_MetaManager::getPageCount\(\)](#)
- ▶ [SetaPDF_MetaManager::getPageData\(\)](#)
- ▶ [SetaPDF_MetaManager::setPageData\(\)](#)
- ▶ [SetaPDF_MetaManager::getInfoData\(\)](#)
- ▶ [SetaPDF_MetaManager::setInfoData\(\)](#)
- ▶ [SetaPDF_MetaManager::getDocumentMetadata\(\)](#)
- ▶ [SetaPDF_MetaManager::setDocumentMetadata\(\)](#)
- ▶ [SetaPDF_MetaManager::getDocumentIds\(\)](#)
- ▶ [SetaPDF_MetaManager::output\(\)](#)

Inherited Methods

Class: [SetaPDF](#)

- ▶ [SetaPDF::isError\(\)](#)

SetaPDF_MetaManager::factory()

Description

```
SetaPDF_MetaManager extends SetaPDF {  
    mixed factory ( string $sourcefile[, string $encoding='UTF-8'] )  
}
```

This method has to be called static and will return an instance of the [SetaPDF_MetaManager](#) class or an [SetaPDF_Error](#) object.

Parameters

\$sourcefile

A string that defines the path (relative or absolute) to the original document. Only local paths are allowed.

\$encoding

Defines the encoding which is used for in- and out-going strings. Internally the API uses [mb_convert_encoding\(\)](#) to convert from one encoding to another.

If [mb_convert_encoding\(\)](#) is not available an own function named [mb_convert_encoding](#) will be created which tries to use [iconv](#) if available. If you want to write your own wrapper, just define the desired function prior loading any SetaPDF API.

Return Values

In case of success you get a new instance of the [SetaPDF_MetaManager](#) class.

On failure an [SetaPDF_Error](#) object will be returned. It is strongly recommended to check this return value with [SetaPDF::isError\(\)](#).

SetaPDF_MetaManager::__destruct()

Description

```
SetaPDF_MetaManager extends SetaPDF {  
    void __destruct ( void )  
}
```

This method will releases memory. It'll be called automatically if you "destroy" the object in PHP5. PHP4 users have to call this method manually if the object is not needed anymore.

SetaPDF_MetaManager::setUseUpdate()

Description

```
SetaPDF MetaManager extends SetaPDF {  
    void setUseUpdate ( [boolean $useUpdate=true] )  
}
```

Defines if the resulting document should only be updated (very fast) or if the resulting document should be rebuild from scratch.

Parameters

\$useUpdate

True or false

SetaPDF_MetaManager::getEncryption()

Description

```
SetaPDF MetaManager extends SetaPDF {  
    mixed getEncryption ( [boolean $details=false] )  
}
```

With this method you can query if a document is encrypted.

Parameters

\$details

If set to true the API will return the raw structure of the encryption dictionary. If set to false the return value is a boolean value.

Return Value

A boolean value if \$details is set to false saying if the document is encrypted or not.

An array of the raw encryption dictionary if \$details is set to true.

Version

Available since version 1.1.

SetaPDF_MetaManager::getPageCount()

Description

```
SetaPDF MetaManager extends SetaPDF {  
    integer getPageCount ( void )  
}
```

Returns the page count of the source document.

Return Value

The page count of the source file.

SetaPDF_MetaManager::getPageData()

Description

```
SetaPDF MetaManager extends SetaPDF {  
    mixed getPageData ( integer $pageNo[, string $key=null[, boolean  
        $limit=false]] )  
}
```

Returns information of a specific page.

Parameters

\$pageNo

The page number you want to get the information from.

\$key

The key or an array of keys which values you want to get.

Possible keys:

- ▶ MediaBox
- ▶ CropBox
- ▶ BleedBox
- ▶ TrimBox
- ▶ ArtBox
- ▶ LastModified
- ▶ Rotate

If no key is given all available values will be taken.

\$limit

With this parameter you can define that the API will load/extract only the pages you defined in parameter *\$pageNo*.

This is usable if you just want to get information of only a single page in a script call. If you use this method several times, you should leave that parameter to be false.

Return Value

If *\$key* is an array or null this method will return an array with the desired key and value pairs or the single value if only a single key is given in the *\$key*-parameter.

If no value is available the value will be *null*.

The box type values (MediaBox, CropBox,...) will be in the following structure:

```
array(  
  'llx' => ..., // Lower Left X  
  'lly' => ..., // Lower Left Y  
  'urx' => ..., // Upper Right X  
  'ury' => ..., // Upper Right Y  
  '_w' => ..., // Width  
  '_h' => ... // Height  
);
```

The Rotate value will be an integer (multiple of 90).

The LastModified value will be returned as a datetime string (see PDF Reference 3.8.3).

Version

Available since version 1.1.

SetaPDF_MetaManager::setPageData()

Description

```
SetaPDF MetaManager extends SetaPDF {  
    mixed setPageData ( integer $pageNo, mixed $key[, $value=null] )  
}
```

Sets page related data.

Parameters

\$pageNo

The page number of the page you want to add/change values.

\$key

The key or an array of key and value pairs of the values you want to set.

Possible keys:

- ◆ MediaBox
- ◆ CropBox
- ◆ BleedBox
- ◆ TrimBox
- ◆ ArtBox
- ◆ LastModified
- ◆ Rotate

\$value

The value of the *\$key*.

The box type values (MediaBox, CropBox,...) have to be in the following structure:

```
array(  
    'llx' => ..., // Lower Left X  
    'lly' => ..., // Lower Left Y  
    'urx' => ..., // Upper Right X  
    'ury' => ... // Upper Right Y  
);
```

The Rotate has to be an integer value (0 or a multiple of 90).

The LastModified value has to be a datetime string (see PDF Reference 3.8.3).

Return Value

An array of new page data values on success, a SetaPDF_Error object if an error occurs.

Version

Available since version 1.1.

SetaPDF_MetaManager::getInfoData()

Description

```
SetaPDF_MetaManager extends SetaPDF {  
    integer getInfoData ( [string $key=null] )  
}
```

Returns the data of the document information dictionary (for example Title, Author, Subject, Keywords, Creator, Producer, CreationDate, ModDate, Trapped).

Parameters

\$key

The name of the desired in value.

Return Value

An array with the desired key and value pairs or the value if a key is given in the key-parameter. If no value is available the method will return *null*.

In case of an error the method will return a SetaPDF_Error object.

SetaPDF_MetaManager::setInfoData()

Description

```
SetaPDF MetaManager extends SetaPDF {  
    mixed setInfoData ( mixed $key[, string $value=null] )  
}
```

Sets the data of the document information dictionary (for example Title, Author, Subject, Keywords, Creator, Producer, CreationDate, ModDate, Trapped).

Actually the API doesn't automatically synchronize the values between XMP metadata and metadata in the document information dictionary.

Parameters

\$key

The key for entry in the document information dictionary.

The PDF specification predefines following keys:

- ▶ Title - The document's title
- ▶ Author - The name of the person who created the document.
- ▶ Subject - The subject of the document.
- ▶ Keywords - Keywords associated with the document.
- ▶ Creator - The name of the application that created the original document from which the PDF document was converted.
- ▶ Producer - The application that created the PDF document.
- ▶ CreationDate - The date and time the document was created.
- ▶ ModDate - The date and time the document was most recently modified.

For details about the above entries see the PDF Reference 10.2.1.

New keys should be chosen with care so that they make sense to users.

\$value This parameter also takes an array of key and value pairs!

The string value for the desired key.

If you use a date value (CreationDate and/or ModDate) make sure you use the right format (see PDF Reference 3.8.3). The ModDate will be changed by default when the document is saved and you've not defined an own value for it. If you set the ModDate your own, make sure that it's after or the same as set in the XMP metadata.

To delete an entry from the document information dictionary just pass "null" as value.

Return Value

True for success, a SetaPDF_Error object if an error occurs.

SetaPDF_MetaManager::getDocumentMetadata()

Description

```
SetaPDF_MetaManager extends SetaPDF {  
    mixed getDocumentMetadata ( void )  
}
```

Returns the XMP metadata of the document.

Return Value

If document metadata are available the method will return the plain XMP string. If no document metadata available the method will return *false*.

In case of an error the method will return a SetaPDF_Error object.

SetaPDF_MetaManager::setDocumentMetadata()

Description

```
SetaPDF_MetaManager extends SetaPDF {  
    mixed setDocumentMetadata ( mixed $metadata[, boolean $force=false] )  
}
```

Sets the documents XMP metadata.

Actually the API doesn't automatically synchronize the values between XMP metadata and metadata in the document information dictionary.

Parameters

\$metadata

The documents XMP data as a string.

\$force

The API will only rewrite the metadata if they were changed. You can force the metadata to be written with this parameter. It's normally only usable if you want to delete the XMP metadata from the document. Just pass *false* to the *\$metadata* parameter and set this parameter to *true*.

Return Value

The method will return *true* if the data is changed. *False* if the data is not changed.

In case of an error the method will return a SetaPDF_Error object.

SetaPDF_MetaManager::getDocumentIds()

Description

```
SetaPDF_MetaManager extends SetaPDF {  
    mixed getDocumentIds ( [boolean $plain=false] )  
}
```

Gets the documents File Identifiers (see PDF Reference 10.3).

Parameters

\$plain

Defines that the file identifier should be returned as a plain string or hex encoded string (default).

Return Value

If the document includes a file identifier, the method will return an *array* of 2 strings.

The **first** string is the permanent identifier based on the content of the document at the time it was originally created.

The **second** string is a changing identifier based on the document's contents at the time it was last updated.

If both strings are set to the same value, the document was not changed after the initial creation.

If the document doesn't includes file identifiers the method will return *false*.

In case of an error the method will return a SetaPDF_Error object.

SetaPDF_MetaManager::output()

Description

```
SetaPDF MetaManager extends SetaPDF {  
    mixed output ( [string $fileName='doc.pdf'[, string $dest='F'[, boolean  
        $stream=false[, mixed $newDocumentId=null]]]] )  
}
```

Gets the documents File Identifiers (see PDF Reference 10.3).

Parameters

\$fileName

A valid path and filename for the new document if the *dest*-parameter is set to "F". Otherwise the name of the new PDF document.

\$dest

Defines how the encrypted document is handled:

- ▶ "F" saves the file to the file system
- ▶ "D" the file will be send to the client with a download dialogue
- ▶ "I" the file will be displayed in the client's browser window.

\$stream

This parameter is only used if *dest* is set to "D" or "I". If it is set to *true*, the document will be sent immediately as soon as the first content bytes are available. In this case the length-header will not be sent. If this parameter is set to *false*, the whole document is held in memory until it is completely assembled.

The streaming facility is very effective, because the client does not become aware of any script processing time.

\$newDocumentId

You can define your own changing identifier with this parameter. Just pass the plain text value to this parameter and it'll be set as the new changing identifier. If set to *null* the API will create an own new changing identifier for the new revision of the document. (More Informations about creating an file identifier can be found in the PDF Reference 10.3)

Return Value

True - if everything works as expected - an SetaPDF_Error object if an error occurs.