SetaPDF-Encryptor API

Manual and Reference

Version 1.6.10, 2010-03-25 16:55:19

Setasign - Jan Slabon
Max-Planck-Weg 7
38350 Helmstedt
Germany

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

Table of contents

SetaPDF-Encryptor API - Introduction

The SetaPDF-Encryptor API is a collection of PHP classes that allows PHP developers to encrypt PDF documents and to grant user rights to them.

SetaPDF-Encryptor 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.

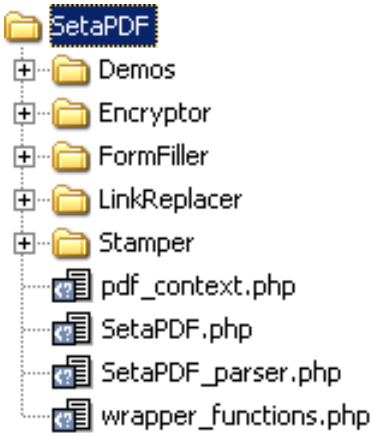
To use the AES algorithm for encryption MCrypt is required!
Furthermore for AES encryption with a 256-bit key the API needs the hash()-function which is available since PHP 5.1.2 or through PECL.

SetaPDF-Encryptor API - Installation

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

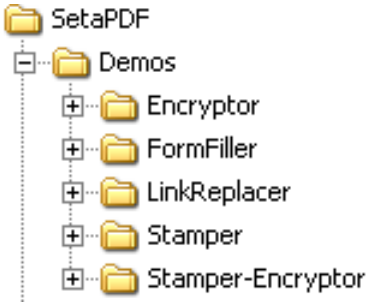
Files and directories

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-Encryptor API with the following line:

```
require_once("Encryptor/SetaPDF_Encryptor.php");
```

SetaPDF-Encryptor 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-Encryptor API - Zend encoded package

If you own a package, which is encoded with the Zend Safguard Suite you have to install the Zend Optimizer or Zend Guard Loader (as of PHP 5.3) - both are FREE of charge! For more information please go to http://www.zend.com/en/products/guard/runtime-decoders.

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>.zl

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 at runtime in your php script before you use the API:

```
$licensePath = realpath('../path/to/.htSetaPDF-
.zl');
zend_loader_install_license($licensePath);
```

...or change or add the license path to the following directive in your php.ini:

Zended packages are only available for development- and serverlicenses.

Standard and Certificate-based encryption

Until version 1.6 of the SetaPDF-Encryptor API it was only possible to use the standard encryption with an owner- and userpassword.

As of the version 1.6 the API also supports certificate-based encryption. Certificate-based encryption lets you encrypt a document for specific recipients by means of public key technology. Various recipients can be given different permissions for the document.

SetaPDF-Encryptor API - Examples of Use

40-bit Encryption

This example encrypts a document with 40-bit, allowing the user to print the document and display it as an inline document. The user needs a user password to open the PDF file.

```
/**
 * set the includepath for SetaPDF APIs
 * You have to point to the root directory "SetaPDF"
 */
set_include_path(get_include_path() . PATH_SEPARATOR .
realpath(dirname(__FILE__).'/../../'));
// require the SetaPDF_Encryptor-class
require_once("Encryptor/SetaPDF_Encryptor.php");
// Create a new instance of SetaPDF_Encryptor and define the original document
$encryptor =& SetaPDF_Encryptor::factory("docs/License_Agreement_SetaPDF.pdf");
// grant the right "print"
$permissions = array(
 "print"
);
// Let's encrypt with 40bit and send it to the client!
$err = $encryptor->encrypt("newpdf.pdf","myownerpw", "myuserpw", $permissions,
SETAPDF_ENC_RC4_40, "I");
// Errorhandling
if (SetaPDF::isError($err)) {
    echo "We've got an error: ";
    print_r($err);
    die();
}
```

128bit Encryption

This example encrypts a document with 128bit, allowing the user to print the document in high quality and to extract/copy text and images. The user does not require a password to open the document. The encrypted file will be offered to the client for downloading. In addition, the document will also be streamed.

```
/**
 * set the includepath for SetaPDF APIs
 * You have to point to the root directory "SetaPDF"
 */
set_include_path(get_include_path() . PATH_SEPARATOR .
realpath(dirname(__FILE__).'/../../'));
// require the SetaPDF_Encryptor-class
```

```
require_once("Encryptor/SetaPDF_Encryptor.php");
// Create a new instance of SetaPDF_Encryptor and define the original document
$encryptor =& SetaPDF_Encryptor::factory("docs/License_Agreement_SetaPDF.pdf");
/**
* grant rights for printing the document
* in high quality and allow the user to copy/extract
* text and graphics from the PDF.
*/
$permissions = array(
    "print",
    "copy",
    "degraded-print"
);
// Let's encrypt with 128bit without an userpassword and send it
// with an download-dialog and in streaming-mode to the client!
$encryptor->encrypt("newpdf.pdf","myownerpw", "", $permissions,
SETAPDF_ENC_RC4_128, "D", true);
```

128-bit AES Encryption + Caching function

This example uses the caching function of the API.

```
/**
 * set the includepath for SetaPDF APIs
 * You have to point to the root directory "SetaPDF"
 */
set_include_path(get_include_path() . PATH_SEPARATOR .
realpath(dirname(__FILE__).'/../../'));
// require the SetaPDF_Encryptor-class
require_once("Encryptor/SetaPDF_Encryptor.php");
// define the cache-path
define('SetaPDF_ENCRYPTOR_CACHEPATH', dirname(__FILE__).'/cache/');
// Create a new instance of SetaPDF_Encryptor and define the original document
$encryptor =& SetaPDF_Encryptor::factory("docs/License_Agreement_SetaPDF.pdf");
// enable Caching-System
$encryptor->setUseCache(true);
/**
* grant rights for printing the
* document in low quality
*/
$permissions = array(
    "print"
);
// Let's encrypt with AES 128-bit without an userpassword
```

```
// and send it to the client!
$encryptor->encrypt("newpdf.pdf","myownerpw", "", $permissions,
SETAPDF_ENC_AES_128, "I");
```

SetaPDF-Encryptor API - Constants / Configuration

The API needs some constants which are hard coded into the API or have to be defined by you. There is also a constant that requires definition prior to first use.

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

Constants to define

`SetaPDF_ENCRYPTOR_CACHEPATH` `(string)`

The constant SetaPDF_ENCRYPTOR_CACHEPATH has to contain a path to which the PHP process can

write and read. This constant **must** be set as soon as the caching system is activated. Otherwise the caching system will not be used.

As of version 1.5 a new global caching function exists and the API own functionality will be removed in coming version.

Predefined Constants

SETAPDF_ENC_RC4_40 (integer)0

Standard encryption with 40-bits.

SETAPDF_ENC_RC4_128 (integer)1

Standard encryption with 128-bits.

SETAPDF_ENC_AES_128 (integer)2

AES (Advanced Encryption Standard) encryption with 128-bits.

SETAPDF_ENC_AES_256 (integer)3

AES (Advanced Encryption Standard) encryption with 256-bits.

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-Encryptor API starts at 300 and ends at 399.

E_SETAPDF_ENC_ERR_PERM (integer)300

Error description: Incorrect permission

E_SETAPDF_ENC_ERR_PERM_40BIT (integer)301

Error description: Incorrect permission for 40bit

E_SETAPDF_ENC_ERR_CACHEUPDATE_NOT_POSSIBLE (integer)302

Error description: A Cache Update is only possible when *$this->usecache* is true.

E_SETAPDF_ENC_ERR_MCRYPT_NOT_AVAILABLE (integer)303

AES encryption requires mcrypt to be installed.

E_SETAPDF_ENC_ERR_MCRYPT_RIJNDAEL_128_NOT_AVAILABLE (integer)304

Needed cipher (RIJNDAEL_128) not available.

E_SETAPDF_ENC_TMPDIR_DOES_NOT_EXISTS (integer)306

Fallback temporary directory _tmp/ does not exists.

E_SETAPDF_ENC_DIR_DOES_NOT_EXISTS (integer)307

Directory "%s" does not exists.

E_SETAPDF_ENC_OPENSSL_ERROR (integer)308

An OpenSSL error occurs + all results from openssl_error_string()

E_SETAPDF_ENC_OPENSSL_BOUNDARY_ID_NOT_FOUND (integer)309

Cannot find boundary id in smime message.

E_SETAPDF_ENC_OPENSSL_ENC_EXTRACTION (integer)310

Error while extracting the encrypted content of the smime message.

E_SETAPDF_ENC_PKSEC_NOT_POSSIBLE (integer)311

Public-Key Security is not supported for RC4 40bit encryption.

E_SETAPDF_ENC_OPENSSL_NOT_AVAILABLE (integer)312

This module requires php compiled with openssl.

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-Encryptor 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-Stamper 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 relys 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 comparsions 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 functionallity 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 fintuning:

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 manual is out-of-date and covers version 1.x.
For version 2.x follow this link.

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

Class Overview

SetaPDF

Child Classes

- SetaPDF_Encryptor

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 manual is out-of-date and covers version 1.x.

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

This manual is out-of-date and covers version 1.x.

The SetaPDF_Parser class is the base class for all indvidual 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 Values (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 houndres 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-Encryptor - Main class

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

Class Overview

SetaPDF
└— SetaPDF_Encryptor

Methods

- SetaPDF_Encryptor::factory()
- SetaPDF_Encryptor::setTmpDirectory()
- SetaPDF_Encryptor::createNewTmpFileName()
- SetaPDF_Encryptor::cleanTmpDirectory()
- SetaPDF_Encryptor::setUseCache()
- SetaPDF_Encryptor::encrypt()
- SetaPDF_Encryptor::updateCache()

Inherited Methods

Class: SetaPDF

- SetaPDF::isError()

SetaPDF_Encryptor::factory()

Description

```
SetaPDF_Encryptor extends SetaPDF {
        mixed factory ( string $sourcefile[, string $tmpDirectory=null] )
}
```

This method has to be called static and will return an instance of the SetaPDF_Encryptor 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.

$tmpDirectory

A path for temporary files. If not or *null* is passed the default fallback directory SetaPDF/Encryptor/_tmp/ is used.

This parameter only affects the usage if you're going to use Public Key Security.

Return Values

In case of success you get a new instance of the SetaPDF_Encryptor class.

On failure an SetaPDF_Error object will be returned. It is strongly recommened to check this return value with SetaPDF::isError().

Version

The $tmpDirectory-parameter is available since version 1.6

SetaPDF_Encryptor::setTmpDirectory()

Description

```
SetaPDF_Encryptor extends SetaPDF {
        mixed setTmpDirectory ( [string $tmpDirectory=null] )
}
```

If you want to change the path for temporary files at runtime, you can use this method.

A writeable temporary path is only needed if you use Public Key Security.

Parameters

$tmpDirectory

A path for temporary files. If not or *null* is passed the default fallback directory SetaPDF/Encryptor/_tmp/ is used.

Return Values

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

Version

Available as of version 1.6

SetaPDF_Encryptor::createNewTmpFileName()

Description

```
SetaPDF_Encryptor extends SetaPDF {
        string createNewTmpFileName ( void )
}
```

This method is a kind of helper method which is used to create unique filenames in the temporary directory.

You can use this method to create temporary files which will be included in the cleanTmpDirectory routine of the SetaPDF-Encryptor API.

Return value

An absoulte unique filename/path

Version

Available as of version 1.6

SetaPDF_Encryptor::cleanTmpDirectory()

Description

```
SetaPDF_Encryptor extends SetaPDF {
       boolean cleanTmpDirectory ( void )
}
```

This methods deletes olded files in the temporary directory.

If the API causes an error it could be that temporary files will remain in the given temporary directory.

In the PHP 5 version this method is called automatically by the __destruct()-method.

In PHP 4 you can call it your own or register it as a shutdown function:

register_shutdown_function(array(&$encryptorInstance,'cleanTmpDirectory'));

The method makes use of the glob()-function. If this function is disabled cause of any security reason you can define your own function by setting it in the SetaPDF::$globFunctionName property. An example of a replacement function can be found in the user contributed notes on php.net

By default temporary files which are older than 60 seconds will be deleted. To adjust this value, change the SetaPDF::$tmpFilesLifetime property.

Temporary files are only needed if you use the Public Key Security feature.

Return value

True if files were deleted. *False* if no file was deleted.

Version

Available as of version 1.6

SetaPDF_Encryptor::setUseCache()

As of version 1.5 a new global caching function exists and the API own functionality will be removed in coming version.

Description

```
SetaPDF_Encryptor extends SetaPDF {
        void setUseCache ( [boolean $usecache=false] )
}
```

(**DEPRECATED**) Is used to change the status of the caching function. Please note that the constant SetaPDF_ENCRYPTOR_CACHEPATH already needs to have been defined and contain a valid path to your file system, to which the Apache / PHP process can write and read. For more information on the caching function, please turn to the section on caching.

Parameters

$usecache

True or *false* â€“ turns the API caching function on (true) or off (false).

SetaPDF_Encryptor::encrypt()

Description

```
SetaPDF_Encryptor extends SetaPDF {
       mixed enrcypt ( string $targetfile[, string $owner_pass|$pks=null[,
       string $user_pass=''[,  $permissions=array()[, integer
       $encryption_strength=SETAPDF_ENC_RC4_40[,  $dest='F'[,
       $stream=false]]]]]] )
}
```

This method is the main component of the API. It is used to encrypt the documents.

Parameters

$targetfile

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.

$owner_pass|$pks

This parameter can be used in 2 ways:

1. Standard Security

The owner password. If you want to use standard security.

If it is set to *NULL,* the API will, for security reasons, generate a random password.

The password string has to be in PDFDocEncoding (which is simliar to cp1252) if $encryption_strength is **NOT** set to SETAPDF_ENC_AES_256. If $encryption_strength is set to SETAPDF_ENC_AES_256 the password string has to be in UTF-8.

2. Public Key Security

An array holding recipient certificates and individual permissions.

Each entry defines a set of certificates and individual persmissions:

certs: Either a lone X.509 certificate, or an array of X.509 certificates. (Details)

permissions (optional): an array identical to the $permissions-parameter. If not set the permissions of the $permissions-parameter are set.

owner (optional): A boolean value. When set to true permits the change of encryption and enables all other permissions for the recipients in *certs*.

Public Key Security is only available if the $encryption_strength-parameter is set to

SETAPDF_ENC_RC4_128, SETAPDF_ENC_AES_128 or SETAPDF_ENC_AES_256

$user_pass

The user password. If a user password is set, it will be asked for when opening the processed document in a viewer.

The password string has to be in PDFDocEncoding (which is simliar to cp1252) if $encryption_strength is **NOT** set to SETAPDF_ENC_AES_256. If $encryption_strength is set to SETAPDF_ENC_AES_256 the password string has to be in UTF-8.

Has no meaning when using Public Key Security.

$permissions

This parameter defines the user privileges for a user opening or viewing the document. Each privilege has to be entered in the permissions array. Depending on the encryption strength, the following entries are possible:

Privileges for 40bit encryption (standard):

print

The user is allowed to print the document. If the key length is 128bit or greater the print quality is defined by the "degraded-print" flag.

modify

The user is allowed to modify the contents of the document by operations other than those controlled by "annot-forms", "fill-in" (>=128bit) and "assemble" (>=128bit).

copy

40bit: The user is allowed to copy or otherwise extract text and graphics from the document, including extracting text and graphics (in support of accessibility to disabled users or for other purposes).

128bit: The user is allowed to copy or otherwise extract text and graphics from the document by operations other than those controlled by "screenreaders".

annot-forms

The user is allowed to add or modify text annotations, fill in interactive form fields, and, if "modify" is also set, create or modify interactive form fields (including signature fields).

More privileges for 128/256bit encryption:

fill-in

The user is allowed to fill in existing interactive form fields (including signature fields), even if

"annot-forms" is not set.

screenreaders

The user is allowed to extract text and graphics (in support of accessibility to disabled users or for other purposes).

assemble

The user is allowed to assemble the document (insert, rotate, or delete pages and create bookmarks or thumbnail images), even if "modify" is not set.

degraded-print

The user is allowed to print the document in a form that is equivalent to a true copy of the original PDF. When this is not set (and "print" is set), printing is limited to a lesser quality (Print As Image).

$encryption_strength

Expects an integer or predefined constant.

SETAPDF_ENC_RC4_40 = 0 - The document will be encrypted with the standard algorithm with a 40-bit encryption key.

SETAPDF_ENC_RC4_128 = 1 - The document will be encrypted with the standard algorithm with a 128-bit encryption key.

SETAPDF_ENC_AES_128 = 2 - The document will be encrypted with the AES algorithm with a 128-bit encryption key.

SETAPDF_ENC_AES_256 = 3 - The document will be encrypted with the AES algorithm with a 256-bit encryption key. (Available since PDF 1.7 / Acrobat >=9)

$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.

Return Values

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

Version

Public Key Security is available since version 1.6

SetaPDF_Encryptor::updateCache()

Description

```
SetaPDF_Encryptor extends SetaPDF {
        mixed updateCache ( void )
}
```

(**DEPRECATED**) This method creates a cache file for the PDF file sent to the factory method. It can be used to create the cache file beforehand or to update the cached version manually.

The call of this method presupposes that setUseCache(true) is activated prior to this method call.

Return Values

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

Caching (DEPRECATED)

As of version 1.5 a new global caching function exists and the API own functionality will be removed in coming version.

The SetaPDF-Encryptor API provides a special caching function that saves a serialized version of the parser object in your file system. Depending on the size of the document, this can save considerable processing time.

Documentsize <> Filesize

Unfortunately it is not possible to predefine the cached data of a PDF document in its relation to its document size. The cache function only provides an intermediate storage for information, but not for embedded images or fonts.

Because of this, it is possible that the cache function will not be able to speed the processing of a 3 MB PDF document which includes some very large images. If the 3 MB document contains many pages, but only of text, the caching function will definitely reduce the processing time.

Usage

You can use the caching function simply by defining a writeable path in the constant SetaPDF_ENCRYPTOR_CACHEPATH and activating setUseCache() afterwards. If such as path has not been defined, the caching function is ignored.