

DynaPDF 3.0

for PHP 5.2 or higher

Installation Guide & API Description

DynaPDF API Version 3.0.32

January 28, 2014

Table of Contents

Table of Contents.....	2
Preface	5
Requirements	5
Limited Windows Compatibility	5
Installation.....	6
Apache Module versus FastCGI.....	6
How to compile the PHP extension?	7
Build Process on Windows.....	7
Build Process on Linux / Unix	8
phpfarm	8
Error Handling.....	9
String Formats.....	10
API Differences	11
Safe Pointers	12
PHP Specific Functions.....	13
DisableUTF8Support (PHP specific).....	13
EnableUTF8Support (PHP specific).....	13
GetAPIVersion	13
GetBufSize	13
GetImageBufSize	14
TestCallback (PHP specific)	14
WriteBuffer	15
WriteImageBuffer	15
Differently implemented functions.....	16
AddBookmarkEx2	16
AddDeviceNProcessColorants	16
AddDeviceNSeparations	16
AddMaskImage	16
CalcPagePixelSize (Rendering Engine)	17
CMYK.....	17
CheckConformance	17
CreateDeviceNColorSpace.....	18
CreateExtGState	18
CreateSetOCGStateAction.....	19

EnumHostFonts	19
EnumHostFontsEx.....	19
EnumDocFonts	20
GetAnnotEx.....	20
GetBarcodeDict	21
GetBBox.....	21
GetBookmark	21
GetBuffer.....	21
GetCMap.....	22
GetColorSpaceObj	22
GetColorSpaceObjEx.....	22
GetContent	22
GetDeviceNAttributes	23
GetDocInfo	23
GetDocInfoEx.....	23
GetEmbeddedFile.....	23
GetErrLogMessage.....	24
GetField (obsolete).....	24
GetFieldChoiceValue	25
GetFieldColor.....	25
GetFieldEx	25
GetFieldEx2	26
GetFieldExpValueEx	26
GetFieldMapName.....	27
GetFieldName.....	27
GetFieldToolTip.....	27
GetFont (Font API).....	27
GetFontEx.....	27
GetImageObj	28
GetInBBox.....	28
GetInDocInfo.....	29
GetInDocInfoEx	29
GetInPrintSettings	29
GetJavaScript.....	29
GetJavaScriptAction.....	29
GetJavaScriptAction2.....	29

GetJavaScriptEx	30
GetJavaScriptName	30
GetLogMetafileSize	30
GetLogMetafileSizeEx.....	30
GetMatrix.....	30
GetInMetadata	30
GetMetadata.....	31
GetMissingGlyphs.....	31
GetNameDest.....	31
GetPageAnnotEx	31
GetPageBBox (Rendering Engine)	31
GetPageField (obsolete)	31
GetPageFieldEx.....	32
GetPageLabel	32
GetPageOrientation (Rendering Engine)	32
GetPageText	32
GetPrintSettings.....	33
GetSeparationInfo.....	33
GetSigDict.....	33
GetTextFieldValue.....	34
GetTextRect	34
GetTextWidthF (Font API).....	34
GetViewerPreferences.....	34
InitStack	35
InsertRawImageEx	35
PolygonAnnot.....	35
PolyLineAnnot.....	35
ReplacePageText.....	36
ReplacePageTextEx	36
RGB.....	36
SetColorMask.....	37
SetOnErrorProc.....	37
SetOnPageBreakProc.....	37
SetProgressProc	38
TranslateRawCode (Font API).....	38
TranslateString2 (Font API)	39

Preface

This document describes how DynaPDF for PHP can be installed and compiled, as well as important differences in comparison to the regular DynaPDF versions.

The generic term DynaPDF applies to the following products:

- DynaPDF Starter
- DynaPDF Lite
- DynaPDF Professional
- DynaPDF Enterprise

The DynaPDF PHP Extension can be used with all versions listed above. The available feature set depends on the used license type. Pre-compiled binaries are available for Windows and Linux.

Requirements

In order to load a PHP extension on a web server, the following requirements must meet:

- PHP 5.2 or higher.
- Since PHP 5.3 `dl()` is disabled in most SAPIs, you need access to the `PHP.ini` in order to load the extension.
- If PHP is installed as an Apache module, then you need administrator privileges to enable the extension in the `PHP.ini` and the web server must be restarted afterwards.
- If no pre-compiled extension is available for your PHP version or operating system, then you need also Visual Studio 6 or higher on Windows for PHP 5.2, or Visual Studio 2008 for PHP 5.3 or higher, or a properly installed GCC for other operating systems.

The extension is delivered with source codes so that you can compile it for every PHP version greater or equal 5.2 (see chapter "[How to compile the extension?](#)").

PHP Extensions are not binary compatible to other PHP versions. That means an extension compiled for PHP 5.3 does not work with PHP 5.4 or vice versa.

Limited Windows Compatibility

Extensions for PHP 5.3 or higher must be build with Visual Studio 2008 since the pre-compiled PHP binaries were compiled with this compiler and the compiler version is now part of the version check of a PHP Extension.

However, adding dependencies to a specific compiler version is the worst thing one can do since it results in a huge overhead if the compiler version will be changed some day.

We do not follow this practice and compile the extension currently with Visual Studio 6 to avoid an additional dependency to a Visual Studio Redistributable Package.

Since the extension wouldn't be loadable with the pre-compiled PHP binaries, the define `ZEND_BUILD_SYSTEM` was manually changed in the header file `/Zend/zend_build.h` to `"VC9"`.

Although there are no known issues at this time, there is no guarantee that the extension is fully compatible with pre-compiled PHP binaries which can be downloaded from the PHP website.

We assume that the Windows versions are mainly used for test purposes, so, this should be no problem. However, on a production machine the extension should either be compiled with Visual Studio 2008 or the PHP binaries and the extension should be compiled with the same compiler.

Installation

The PHP Extension is a shared library on Linux or dynamic link library (DLL) on Windows. The name of the library is `php_dynapdf.dll` on Windows and `dynapdf.so` on Linux. You find the libraries in the folder *ext* of the download package.

- Copy the library into the */ext* folder of your PHP installation or into the path that was configured in the `PHP.ini` (`extension_dir`).
- Enable the extension in the `PHP.ini` with:
`extension=php_dynapdf.dll` (Windows)
or
`extension=dynapdf.so` (Linux)

Dynamic loading of PHP extensions is disabled in most SAPIs and no longer recommended.

DynaPDF is implemented as a class module. The name of the class is (lowercase) `dynapdf`. All functions and constants are bound on the class `dynapdf`. No global functions or constants are used which could conflict with other extensions.

Apache Module versus FastCGI

Most pre-defined setups install PHP as Apache module. Although this type of installation is very simple, Apache modules have many drawbacks in comparison to FastCGI. The main disadvantages are: PHP is executed with the Apache user and in the Apache address space.

If the PHP process crashes due to an arbitrary error, then Apache dies too and the website is offline. In addition, files created with PHP scripts (including PDF files), cannot be deleted by a FTP user since the Apache user is the owner of these files.

If PHP is installed as FastCGI module, then PHP is executed in a separate process that doesn't affect the Apache process when it crashes. The PHP process is simply restarted on the next request and the web site stays online.

The PHP process can also be executed with a FTP user account. This solves the owner problem of files which were created with PHP and every user can use its own `PHP.ini`. Since the Apache user is no longer used, it is also possible to apply independent user rights.

Such an installation is very safe, independent of the programming skill of the web developer. Another advantage is that the extension can be updated without restarting the web server.

That is the reason why most web hosting providers install PHP as FastCGI module. It is some more work, but the benefits outweigh.

How to compile the PHP extension?

The build process for DynaPDF Professional and DynaPDF Enterprise is identically, but both packages contain different Visual Studio workspaces.

Build Process on Windows

Download the normal DynaPDF for Windows package or DynaPDF Enterprise and extract or install it on your system, depending on the version you have downloaded (zip archive or msi installer).

Download the sources of the wished PHP version from <http://php.net/downloads.php> and extract it into the DynaPDF directory.

Download also the corresponding pre-compiled multi-threaded package of PHP for Windows (32 bit x86). We need the **php5ts.lib** from this package. Extract the library into the PHP source directory. The library must be stored in the subfolder `/dev`.

The directory structure should now look as follows:

```
dynapdf
/bin
/borland_lib
/include
/mvs_lib
/php_module // Source codes of the PHP Extension
/php-5.x.x  // PHP sources (must be renamed to php-src)
/dev       // This directory must contain the php5ts.lib
/..        // More directories follow
```

- Rename the PHP source directory to `php-src`.
- Go into the directory `php_module`.
- Open the Visual Studio 6 or VS 2005 workspace depending on which version matches best your Visual Studio version.

For PHP 5.3 or higher Visual Studio 2008 is required if the extension should be usable with a pre-compiled PHP version for Windows. Although it is not officially supported, another compiler can still be used. Open the file `/php-src/Zend/zend_build.h` and set the `ZEND_BUILD_SYSTEM` to `"VC9"`. This change makes it possible to load the extension but there is no guarantee that it is fully compatible with the pre-compiled PHP binaries. For best compatibility, PHP should be compiled with the same compiler too.

- Select the configuration "Release MDLL" in Visual Studio and compile the library. You find the `php_dynapdf.dll` in the directory *Release* after successful compilation. The workspace creates a 32 bit library.

Additional note for DynaPDF Professional:

The `php_dynapdf.dll` depends on the `dynapdfm.dll` which must be copied into `Windows/System32` (32 bit Windows) or `Windows/SysWOW64` on a 64 bit machine. If you use VS 2005 or higher for compilation, then the resulting library depends also on the Visual Studio Redistributable Package of the used VS version.

The `php_dynapdf.dll` must be copied into the `/ext` folder of your PHP installation or into the directory that was configured in the `PHP.ini` (`extension_dir`).

The dependency to the `dynapdfm.dll` does not exist if the library was compiled with DynaPDF Enterprise since these workspaces link the DynaPDF library statically. The pre-compiled library does **not** depend on the `dynapdfm.dll`.

Build Process on Linux / Unix

Install PHP and the development tools on your machine. On CentOS 6.3, for example, this can be done with the following commands:

```
sudo yum install "Development Tools" // Install GCC and the build tools
sudo yum install php 5.4             // Install PHP 5.4 for example
sudo yum install php-devel 5.4       // Install the header files and build tools
```

If the above commands fail then you must maybe update the package manager. Depending on the used Linux distribution the commands are slightly different.

In most cases another machine is used to compile a PHP extension. Important is that the major and minor version of the PHP versions on the web server and on the development machine are identically, only the build number can be different.

phpfarm

If you need to work with different PHP versions then we strongly recommend `phpfarm`. `phpfarm` is a set of command line tools which enable the usage of different PHP versions on the same system. Almost every PHP version can directly be downloaded and compiled with just one command, e.g. `compile.sh 5.4.17`. To activate a specific version call `switch-phpfarm` followed by the version number that should be activated, e.g. `switch-phpfarm 5.4.17`.

One thing that must be considered is that the `--enable-debug` flag is hardcoded in the `compile.sh`. This flag should be deleted and added to the `options.sh` or `custom-options.sh` if needed.

`phpfarm` is available at GitHub (<https://github.com/cweiske/phpfarm>).

Build process for DynaPDF Professional

- Download the sources of the PHP extension and a DynaPDF version that is compatible with your operating system, e.g. DynaPDF for Linux x86. Decompress both packages in the same directory.
- Open the directory `dynapdf/dynapdf` and delete the `libdynapdf.so` or rename the library. The shared library takes precedence in the configure script. If the shared library is present then this one will be linked. Static linking is preferred to avoid unnecessary dependencies.
- Open the directory `php_module` now in a terminal window.
- Type `./confrel` and press enter. If the file cannot be executed then change the access permissions with `chmod` to `777`.
- The library will now be compiled. The finish library can be found the directory `dynapdf/php_module/modules`.

- Copy the library on your web server, finished.
- With `./clean` you can clean up the build directory. Note that this command deletes also the modules directory!

If you get linker errors with the static library of DynaPDF then use the shared library instead. The PHP extension depends then of course on the `libdynapdf.so`. The library must be copied into a `LD_LIBRARY` path in this case so that it can be loaded on the web server.

Note that it is no bug if the static library cannot be linked. Static linking works only if the same GCC version is installed on your machine as the one that we used to compile the library. This is due to dependencies to certain system libraries like the `libc` which must be available in the same version. With which version DynaPDF was compiled can be seen in the download area.

Build process for DynaPDF Enterprise

- Extract the DynaPDF Enterprise package and open the directory `dynapdf_ent/php_module` in a terminal window.
- Type `./confrel` and press enter. If the file cannot be executed then change the access permissions with `chmod` to `777`.
- The above command compiles DynaPDF and the PHP Extension. The finish library can be found the directory `dynapdf_ent/php_module/modules`.
- Copy the library on your web server, finished.
- With `./clean` you can clean up the build directory. Note that this command deletes also the modules directory!

Error Handling

The PHP extension passes error messages and warnings to the PHP engine by default. Depending on the settings in the `PHP.ini`, the messages are either output directly on the website or in the PHP error log. The extension does not use PHP exceptions.

The error handling can be modified if necessary:

- Syntax errors, like invalid or missing parameters, are always handled by the PHP engine since such errors are handled before any DynaPDF function will be executed.
- All (DynaPDF) error messages can be redirected to the error log (see `SetErrorMode()` for further information). In this case, error messages can be accessed with `GetErrLogMessageCount()` / `GetErrLogMessage()`.
- You can set an error callback function with `SetOnErrorProc()`. DynaPDF error messages are passed to the callback function in this case.

Note that error messages cannot be output on the website if the PDF file should be sent to the client since the error messages would be output before the PDF file. If the http headers are already sent then the error messages are stored in the file that the user tries to download! To avoid such issues, the option `display_errors` must be set to `Off` in the `PHP.ini` or manually disabled in a download script.

String Formats

The default string format in the PHP extension is UTF-8 for all strings. However, it is also possible to work with code pages, although this is mostly not required. To disable the UTF-8 processing call:

```
$pdf->DisableUTF8Support( );
```

Functions like WriteText(), WriteFText() or AddContinueText() treat the string parameter now as 8 bit string defined in the code page of the used font. The code page was set in the SetFont() function call.

All other functions, which do not use a font, treat strings now as Ansi strings defined in the Windows code page 1252.

To enable UTF-8 processing again, call EnableUTF8Support(). You can enable or disable UTF-8 support arbitrary often. These functions are very fast and cause no overhead.

Note that you can still work with code pages when UTF-8 processing is enabled. The overhead is very small. It is mostly not required to disable the UTF-8 string format. This should only be if a source string is defined in a specific code page and to avoid an unnecessary conversion to UTF-8.

Note that UTF-8 support affects only string parameters which are passed to DynaPDF. Strings which are returned by DynaPDF are always returned in UTF-8 Unicode format!

API Differences

Most functions in PHP are identically defined in comparison to the C or C++ interface but a few functions are differently implemented. Differently implemented are all functions which use parameters by reference to return certain values, including functions which return structures.

- The PHP extension does not use or access parameters by reference. Multiple values are returned as array instead.
- Enums are defined as class constants in PHP. The constants can be accessed with the class name, e.g. `dynapdf::ConstName`.
- ALL strings (except binary strings) are returned in UTF-8 Unicode format. Structures which contain members for the string length or string format in C/C++ are omitted in PHP since the length of a PHP string is always defined and the returned string format is always UTF-8.
- Functions which have an additional parameter for a string length, like `WriteTextEx()`, are either not implemented, if another version of the same function exists with no length parameter, like `WriteText()` for example, or they have no length parameter in PHP. PHP strings have already an implicit length. There is no need to explicitly define a length as in other programming languages like C or C++.
- Structures are returned as associative PHP arrays. All functions which return structures return NULL on failure or if no value is defined. The keys are set in the same order as defined in the C structure. Keys with no value are omitted.

Example:

```
...
$bbox = $pdf->GetBBox(dynapdf::pbMediaBox);
if ($bbox)
{
    $left    = $bbox['Left'];
    $right   = $bbox['Right'];
    $bottom  = $bbox['Bottom'];
    $top     = $bbox['Top'];
}
...
```

- The PHP extension contains no Ansi or Wide string versions of functions which accept string parameters, since all string parameters are either defined as UTF-8 or Ansi, depending on the used string format (see section [String Formats](#)).
- If a structure contains an Ansi and Wide string key for the same value, e.g. `ValueA` and `ValueW`, then the value is set to the natural key name without the ending 'A' or 'W' ('Value' in this example).

Safe Pointers

Several DynaPDF functions return normally a pointer but pointers cannot be used in PHP since pointers are unsafe.

Functions which return a pointer in C/C++ return a safe pointer in PHP. A safe pointer is an integer hash that encodes the original pointer and the pointer type in a 32 bit integer value. The hashes and the original pointers are stored in a list.

Whenever a safe pointer is passed to a DynaPDF function, and if it is not 0, then a search routine tries to find the pointer in the list, and if it can be found, then it checks whether the requested pointer type matches the pointer type that was stored in the list. If anything is ok then the DynaPDF function will be executed.

This makes it impossible to modify a safe pointer in way that a bad pointer access could occur. It is also impossible to pass a wrong pointer type to a function since the type is encoded in the hash.

A safe pointer uses the entire integer range. That means it can be positive or negative. A NULL pointer represents the integer value 0. If a pointer should be set to NULL then pass the integer value 0 or NULL to the function.

The internal pointer list is released whenever the corresponding PDF objects will be released. This is the case when the PDF file is closed with `CloseFile()` for example, or if you call `FreePDF()`, or if the PDF instance will be released.

After such a function was called, all previously returned safe pointers become invalid. Any try to use such a pointer results in an error message but nothing critical happens.

PHP Specific Functions

DisableUTF8Support (PHP specific)

Syntax:

```
void DisableUTF8Support ( )
```

The function can be used to disable UTF-8 support for string parameters.

Functions like WriteText(), WriteFText() or AddContinueText() treat the string parameter now as 8 bit string defined in the code page of the used font. The code page was set in the SetFont() function call.

All other functions, which do not use a font, treat strings now as Ansi strings defined in the Windows code page 1252.

To enable UTF-8 processing again, call EnableUTF8Support(). You can enable or disable UTF-8 support arbitrary often. These functions are very fast and cause no overhead.

EnableUTF8Support (PHP specific)

Syntax:

```
void DisableUTF8Support ( )
```

The function enables UTF-8 support for string parameters if it was previously disabled.

GetAPIVersion

Syntax:

```
int GetAPIVersion ( )
```

The function returns the API version of the PHP Extension as an integer value, e.g. 1003 for the API version 1.0.0.3. The API version is always incremented if something was changed or if new features were added.

The API version of the PHP Extension and the API version of the DynaPDF library are independent from each other. The API version does not necessarily change when the extension is updated to the actual DynaPDF version. If no new feature was added then the API version remains as is.

GetBufSize

Syntax:

```
int GetBufSize ( )
```

The function returns the size of the PDF file if it was created in memory. The function can be called before the buffer is send to the client with [WriteBuffer\(\)](#) so that the http header Content-Length can be filled in.

Return values:

If the function succeeds the return value is the length of the PDF file in bytes. If the function fails the return value is zero.

GetImageBufSize

Syntax:

```
int GetImageBufSize()
```

The function returns the size of the image file if it was created in memory. The function can be called before the buffer is send to the client with [WriteImageBuffer\(\)](#) so that the http header Content-Length can be filled in.

Return values:

If the function succeeds the return value is the length of the image file in bytes. If the function fails the return value is zero.

TestCallback (PHP specific)

Syntax:

```
void TestCallback(TCallbackFuncs Param)

const cbfAll                = 0;  // Fire all
const cbfOnErrorProc        = 1;  // SetOnErrorProc()
const cbfOnFontNotFoundProc  = 2;  // CheckConformance()
const cbfOnInitProgressProc = 4;  // SetProgressProc()
const cbfOnPageBreakProc    = 8;  // SetOnPageBreakProc()
const cbfOnProgressProc     = 16; // SetProgressProc()
const cbfonReplaceICCPProfileProc = 32; // CheckConformance()
```

The function can be used to check whether callback functions work as expected. The integer constants can be combined with a binary or operator. If all currently defined callback functions should be fired set the parameter to cbfAll or 0.

It is no error in PHP if a callback function contains more or less parameters as specified.

If a function must return something while it does not, or if it returns a wrong data type then TestCallback() throws an error. If no error messages are output then anything is ok.

Example:

```
function InitProgressProc($ProgType, $MaxCount)
{
    echo '<p>ProgType: ' . $ProgType . '</p>';
    echo '<p>MaxCount: ' . $MaxCount . '</p>';
}

// This callback function must return an integer value.
// TestCallback() throws as error for this function.
function ProgressProc($ActivePage)
{
    echo '<p>ActivePage: ' . $ActivePage . '</p>';
}

$pdf->SetProgressProc('InitProgressProc', 'ProgressProc');

// Check whether the callback functions work as expected
$pdf->TestCallback($pdf::cbfAll);
```

WriteBuffer

Syntax:

```
void WriteBuffer( )
```

The function outputs the buffer of a PDF file that was created in memory. WriteBuffer() is more efficient in comparison to [GetBuffer\(\)](#) since it outputs the buffer directly. GetBuffer() must create a copy of the already existing buffer. This is inefficient and wastes memory.

The function can be called after CloseFile() or CloseFileEx() since the buffer does not exist before the file was closed.

The function [GetBufSize\(\)](#) returns the size of the PDF buffer.

WriteImageBuffer

Syntax:

```
void WriteImageBuffer( )
```

The function outputs the buffer of an image that was created in memory. WriteImageBuffer() is more efficient in comparison to [GetImageBuffer\(\)](#) since it outputs the buffer directly. GetImageBuffer() must create a copy of the already existing buffer. This is inefficient and wastes memory.

The function [GetImageBufSize\(\)](#) returns the size the image buffer.

Differently implemented functions

This chapter describes most but not all functions, which are differently implemented in comparison to C or C++. Only functions are listed which require some more explanation.

Note that only the differences are described here. The definition of data types or how certain functions can be used is described in the DynaPDF help file.

AddBookmarkEx2

Syntax:

```
int AddBookmarkEx2(
    $Title,          // Title of the bookmark (string)
    $Parent,         // Parent bookmark if any or -1 for a root node (int)
    $NamedDest,      // Name of a named destination (string)
    $Open)           // Open or close the node? (bool)
```

The parameter *Unicode* is omitted since the string format is always UTF-8 Unicode or Ansi.

AddDeviceNProcessColorants

Syntax:

```
bool AddDeviceNProcessColorants(
    $DeviceNCS,      // Handle of a DeviceN or NChannel color space (int)
    $Colorants,      // Array of colorant names (array of strings)
    $ProcessCS,      // Process color space (int)(TExtColorSpace)
    $Handle)         // Color space handle or -1 (int)
```

The array length of the *Colorants* array is already implicitly defined in PHP.

AddDeviceNSeparations

Syntax:

```
LBOOL pdfAddDeviceNSeparations(
    $DeviceNCS,      // Handle of a DeviceN or NChannel color space
    $Colorants,      // Array of colorant names (array of strings)
    $SeparationCS)   // Array of Separation color space handles (int)
```

The *Colorants* and *SeparationCS* arrays must contain the same number of elements. The parameter *NumColorants* is omitted since the array length is implicitly defined in PHP.

AddMaskImage

Syntax:

```
bool AddMaskImage(
    $BaseImage,      // int
    $Buffer,         // Binary string
    $Stride,         // int
    $BitsPerPixel,   // int
    $Width,          // int
    $Height)         // int
```

The C or C++ version contains an additional parameter *BufSize* which is omitted in PHP.

CalcPagePixelSize (Rendering Engine)

Syntax:

```
array CalcPagePixelSize(  
    $PageNum,          // Page number (int)  
    $DefScale,         // Scaling type (int)(TPDFPageScale)  
    $Scale,            // Scaling factor (if DefScale = psFitZoom)(double)  
    $FrameWidth,       // Width of the output rectangle (int)  
    $FrameHeight,     // Height of the output rectangle (int)  
    $Flags)            // Additional flags (int)(TRasterFlags)
```

The C or C++ version of the function contains two additional out parameters *Width* and *Height*. In addition, the first parameter is normally a pointer of the page object.

In PHP, the first parameter is the page number. The first page is denoted by 1.

The function returns an associative array with integer values for the Width and Height or NULL on failure.

Example:

```
$val = $pdf->CalcPagePixelSize(1, dynapdf::psFitBest, 1.0, 800, 600,  
    dynapdf::rfDefault);  
if ($val)  
{  
    $width  = $val['Width'];  
    $height = $val['Height'];  
}
```

CMYK

Syntax:

```
int CMYK(  
    $c, // int (0..255)  
    $m, // int (0..255)  
    $y, // int (0..255)  
    $k) // int (0..255)
```

The function converts CMYK values to one integer value. The function does exactly the same as the PDF_CMYK() macro of the C interface. The resulting value can be passed to functions which accept CMYK colors in integer format. CMYK colors use the entire integer range.

This function can only fail if one or more parameters are missing. If the function fails the return value is NULL and an error message is passed to the PHP engine.

CheckConformance

Syntax:

```
int CheckConformance(  
    $Type,                // int (TConformanceType)  
    $Options,             // int (TCheckOptions)  
    callback $OnFontNotFound, // can be NULL, see below  
    callback $OnReplaceICCPfile) // can be NULL, see below  
  
// The function must return an integer value, typically the return  
// value of ReplaceFont() or ReplaceFontEx() or -1 to break  
// processing.
```

```

function OnFontNotFoundProc(
    $PDFFont,      // Safe pointer
    $FontName,     // string
    $Style,        // int (TFStyle)
    $StdFontIndex, // int
    $IsSymbolFont) // bool

// The function must return an integer value, typically the return
// value of ReplaceICCProfile().
function TOnReplaceICCProfile(
    $Type,          // int (TICCProfileType)
    $ColorSpace)    // int

```

The parameter *UserData* is omitted in PHP.

CreateDeviceNColorSpace

Syntax:

```

int CreateDeviceNColorSpace(
    $Colorants,      // Array of colorant names (required)
    $PostScriptFunc, // Postscript calculator function (string)
    $Alternate,      // Alternate color space (int)(TExtColorSpace)
    $Handle)         // Handle of the alternate color space or -1

```

The parameter *NumColorants* is omitted since the array length is implicitly defined in PHP.

CreateExtGState

Syntax:

```

int CreateExtGState(array $GS)

```

The creation of an extended graphics state is a bit easier in comparison to other programming languages since no structure must be initialized before the function can be called.

Add only key / value pairs to the array which should be modified!

The following keys can be defined:

```

AutoStrokeAdjust // bool or int (0 or 1)
BlendMode        // int (TBlendMode)
FlatnessTol      // double (0.0 .. 1.0)
OverPrintFill    // bool or int (0 or 1)
OverPrintStroke  // bool or int (0 or 1)
OverPrintMode    // bool or int (0 or 1)
RenderingIntent  // int (TRenderingIntent)
SmoothnessTol    // double (0.0 .. 1.0)
FillAlpha        // double (0.0 .. 1.0)
StrokeAlpha      // double (0.0 .. 1.0)
AlphaIsShape     // bool or int (0 or 1)
TextKnockout     // bool or int (0 or 1)
SoftMaskNone     // bool or int (0 or 1)
SoftMask         // Safe pointer returned by CreateSoftMask()

```

Example:

```

...
$gs = $pdf->CreateExtGState(array('FillAlpha' => 0.5));
$pdf->SetExtGState($gs);

```

CreateSetOCGStateAction

Syntax:

```
int CreateSetOCGStateAction(  
    $On,           // Array of OCG handles which should be set to On  
    $Off,          // Array of OCG handles which should be set to Off  
    $Toggle,       // Array of OCG handles which should toggle the state  
    $PreserveRB) // Preserve radio button relationships if any? (bool)
```

The parameters to define the number of array values are omitted since the array length is implicitly defined in PHP.

EnumHostFonts

Syntax:

```
array EnumHostFonts()
```

The function returns an array of arrays and it has no parameters as in other programming languages. If no host fonts are available, the return value is NULL.

The sub arrays contain the parameters which are normally passed to a callback function. The arrays contain the following key / value pairs:

Example:

```
for ($i = 0; $i < count($retval); $i++)  
{  
    $val = &$retval[$i];  
    echo $val['FamilyName']."\n";      // string  
    echo $val['PostScriptName']."\n";  // string  
    echo $val['Style']."\n\n";         // int (TFStyle)  
}
```

EnumHostFontsEx

Syntax:

```
array EnumHostFontsEx()
```

The function returns an array of arrays and it has no parameters as in other programming languages. If no host fonts are available, the return value is NULL.

The sub arrays contain the parameters which are normally passed to a callback function. The arrays contain the following key / value pairs:

Example:

```
for ($i = 0; $i < count($retval); $i++)  
{  
    $val = &$retval[$i];  
    echo $val['FamilyName']."\n";      // string  
    echo $val['PostScriptName']."\n";  // string  
    echo $val['Style']."\n";           // int (TFStyle)  
    echo $val['BaseType']."\n";        // int (TFontBaseType)  
    echo $val['Flags']."\n";           // int (TEnumFontProcFlags)  
    echo $val['FilePath']."\n\n";      // string  
}
```

EnumDocFonts

Syntax:

```
array EnumDocFonts( )
```

The function returns an array of arrays and it has no parameters as in other programming languages. If no fonts are actually used, the return value is NULL.

The sub arrays contain the parameters which are normally passed to a callback function. The arrays contain the following key / value pairs:

```
$retval['PDFFont']      // int (safe pointer)
$retval['Type']         // int (TFontType)
$retval['BaseFont']     // string
$retval['FontName']     // string
$retval['Embedded']     // bool
$retval['IsFormFont']   // bool
$retval['Flags']        // int
```

GetAnnotEx

Syntax:

```
array GetAnnotEx(
    $Handle) // Annotation handle (int)
```

The function returns the structure TPDFAnnotationEx as an associative array on success or NULL on failure.

The following keys are defined:

```
Type           // int (TAnnotType)
Deleted         // bool
BBox            // array (Left, Bottom, Right, Top)
BorderWidth     // double
BorderColor     // int
BorderStyle     // int (TBorderStyle)
BackColor       // int
Handle          // int
Author          // string
Content         // string
Name            // string
Subject         // string
PageNum         // int
HighlightMode   // int (THighlightMode)
DestPage        // int
DestPos         // array (Left, Bottom, Right, Top)
DestType        // int (TDestType)
DestFile        // string
Icon            // int
StampName       // string
AnnotFlags      // int (TAnnotFlags)
CreateDate      // string
ModDate         // string
Grouped         // bool
Open            // bool
Parent          // int
PopUp           // int
State           // string
```

```

StateModel    // string
EmbeddedFile  // int
Subtype       // string
MarkupAnnot   // bool
Opacity       // double

```

Only keys with a value will be returned.

GetBarcodeDict

Syntax:

```

array GetBarcodeDict(
    $IBarcode) // Safe pointer

```

The function returns the key / value pairs of the structure TPDFBarcode as an associative array on success or NULL on failure. Note that only one key is returned for the members CaptionA and CaptionW. The key is 'Caption' and the value is returned in UTF-8 Unicode format if set.

GetBBox

Syntax:

```

array GetBBox(
    $Boundary) // (int)(TPageBoundary)

```

The function returns the structure TPDFRect as an associative array on success or NULL on failure or if the bounding box is not set.

GetBookmark

Syntax:

```

array GetBookmark(
    $Handle // Bookmark handle (int)

```

The function returns the structure TBookmark as an associative array on success or NULL on failure. A bookmark handle is an array index in the range 0 through GetBookmarkCount() -1.

The following keys are defined

```

Color      // int
DestPage   // int
DestPos    // array (Left, Bottom, Right, Top)
DestType   // int (TDestType)
Open       // bool
Parent     // int
Style      // int (TBmkStyle)
Title      // string

```

GetBuffer

Syntax:

```

binary string GetBuffer( )

```

The function has no parameters. Use the PHP function strlen() to get the length of the buffer. It is more efficient to call [WriteBuffer\(\)](#) in combination with [GetBufSize\(\)](#) instead since GetBuffer() must create a copy of the PDF buffer before it can be passed to the PHP engine.

GetCMap

Syntax:

```
array GetCMap(  
    $Index) // CMap index (int)
```

The function returns the members of the structure TPDFCMap as an associative array.

The following keys are defined:

```
BaseCMap          // string  
CIDCount          // int  
CMapName          // string  
CMapType          // int  
CMapVersion       // double  
DSCBaseCMap       // string  
DSCCMapVersion    // double  
DSCResName        // string  
DSCTitle          // string  
FileName          // string  
FilePath          // string  
Ordering          // string  
Registry          // string  
Supplement        // int  
WritingMode       // int
```

Only keys with a value will be returned.

GetColorSpaceObj

Syntax:

```
array GetColorSpaceObj(  
    $Handle) // Color space handle (int)
```

The function returns the members of the structure TPDFColorSpaceObj as an associative array. The members *BufSize* and *ColorantsCount* will never be returned since the buffer size and colorant count are always available for PHP arrays and strings. Only keys with a value will be returned.

GetColorSpaceObjEx

Syntax:

```
array GetColorSpaceObjEx(  
    $IColorSpace) // Safe pointer
```

The function works like [GetColorSpaceObj\(\)](#) but accepts a safe pointer instead of a color space handle.

GetContent

Syntax:

```
binary string GetContent( )
```

The function has no parameters. Use the PHP function `strlen()` to get the length of the buffer.

GetDeviceNAttributes

Syntax:

```
array GetDeviceNAttributes(  
    $IAttributes // Safe pointer
```

The function returns the TDeviceNAttributes structure as an associative array on success or NULL on failure.

The following keys are defined:

```
IProcessColorSpace // Safe pointer -> GetColorSpaceEx()  
ProcessColorants   // array of strings  
Separations        // array of safe pointers -> GetColorSpaceEx()  
IMixingHints        // Safe pointer
```

GetDocInfo

Syntax:

```
string GetDocInfo(  
    $DInfo) // int (TDocumentInfo)
```

The function returns the document info entry on success or NULL on failure or if no value is defined for the key.

GetDocInfoEx

Syntax:

```
array GetDocInfoEx(  
    $Index) // Entry index(int)
```

The function returns the parameters and value of the document info entry as an associative array on success or NULL on failure.

The following keys are defined:

```
Key    // string  
DInfo  // int (TDocumentInfo) -> This key is always set  
Value  // string
```

GetEmbeddedFile

Syntax:

```
array GetEmbeddedFile(  
    $Handle,        // Handle of an embedded file (int)  
    $Decompress)    // If true, the file is decompressed (bool)
```

The function returns the structure TPDFFileSpec as an associative array on success or NULL on failure.

The following keys are defined:

```
Buffer        // binary string  
Compressed     // bool  
ColItem        // Safe pointer  
Name           // string  
FileName       // string  
IsURL          // bool  
UF             // string
```

```

Desc           // string
FileSize       // Decompressed stream size or zero if not known (int)
MimeType       // string
CreateDate     // string
ModDate        // string
Checksum       // string

```

Only keys with a value will be returned.

GetErrLogMessage

Syntax:

```

array GetErrLogMessage(
    $Index) // Message index (int)

```

The function returns the structure TPDFError as an associative array on success or NULL on failure.

The following keys are defined:

```

Message // string
ObjNum   // int
Offset   // int
SrcFile  // string
SrcLine  // int

```

GetField (obsolete)

Syntax:

```

array GetField(
    $Handle) // int

```

The function returns the structure TPDFField as an associative array on success or NULL on failure.

The following keys are defined:

```

FieldType      // int (TFieldType)
Deleted        // bool
BBox           // array (Left, Bottom, Right, Top)
Handle         // int
FieldName      // string
BackCS         // int (TPDFColorSpace)
TextCS         // int (TPDFColorSpace)
BackColor      // int
BorderColor    // int
TextColor      // int
Checked        // bool
Parent         // int
KidCount       // int
Font           // string
FontSize       // double
Value          // string
ToolTip        // string

```

Only keys with a value will be returned.

GetFieldChoiceValue

Syntax:

```
array GetFieldChoiceValue(  
    $AField,    // Field handle (int)  
    $ValIndex) // Value index (int)
```

The function returns the structure TPDFChoiceValue as an associative array on success or NULL on failure.

The following keys are returned if set:

```
ExpValue    // string  
Value       // string  
Selected    // bool (always present)
```

GetFieldColor

Syntax:

```
array GetFieldColor(  
    $AField,    // Field handle (int)  
    ColorType) // Color type (int)(TFieldColor)
```

The function contains normally two additional parameters to return the color space and the corresponding color value. These parameters are returned as an associative array instead.

The returned key / value pairs are:

```
ColorSpace // int (TPDFColorSpace)  
Color      // int
```

GetFieldEx

Syntax:

```
array GetFieldEx(  
    $Handle) // Field handle (int)
```

The function returns the structure TPDFField as an associative array on success or NULL on failure.

The following keys are defined:

```
Deleted        // bool  
BBox           // array (Left, Bottom, Right, Top)  
FieldType      // int (TFieldType)  
GroupType      // int (TFieldType)  
Handle         // int  
BackColor      // int  
BackColorSP    // int (TExtColorSpace)  
BorderColor    // int  
BorderColorSP  // int (TExtColorSpace)  
BorderStyle    // int (TBorderStyle)  
BorderWidth    // double  
CharSpacing    // double  
Checked        // bool  
CheckBoxChar   // int  
DefState       // int (TCheckBoxState)  
DefValue       // string  
IEditFont      // Safe pointer -> GetFont()
```

```

EditFont      // string
ExpValCount   // int
ExpValue      // string
FieldFlags    // int (TFieldFlags)
IFieldFont    // Safe pointer -> GetFont()
FieldFont     // string
FontSize      // double
FieldName     // string
HighlightMode // int (THighlightMode)
IsCalcField   // bool
MapName       // string
MaxLen        // int
Kids          // array of safe pointers -> GetFieldEx2()
Parent        // Safe pointer -> GetFieldEx2()
PageNum       // int
Rotate        // int
TextAlign     // int (TTextAlign)
TextColor     // int
TextColorSP    // int (TExtColorSpace)
TextScaling   // double
ToolTip       // string
UniqueName    // string
Value         // string
WordSpacing   // double
IBarcode      // Safe pointer -> GetBarcodeDict()
ISignature    // Safe pointer -> GetSigDict()

```

Only keys with a value will be returned.

GetFieldEx2

Syntax:

```

array GetFieldEx2(
    $IField) // Safe pointer of a field

```

The function work exactly like [GetFieldEx\(\)](#) but it accepts a safe pointer instead of a field handle.

GetFieldExpValueEx

Syntax:

```

array GetFieldExpValueEx(
    $AField,    // Field handle (int)
    $ValIndex)  // Value index (int)

```

The function contains normally three additional parameters to return the field value. These parameters are returned as an associative array on success or NULL on failure.

The following key / value pairs are returned:

```

Value      // string
ExpValue    // string
Selected   // bool

```

GetFieldMapName

GetFieldName

GetFieldToolTip

Syntax:

```
string pdfGetFieldToolTip(  
    $AField) // Field handle (int)
```

The above functions require the same parameter and return a string value on success or NULL on failure.

GetFont (Font API)

Syntax:

```
array GetFont(  
    $IFont) // Safe pointer
```

The function returns the structure TPDFFont as an associative array on success or NULL on failure.

The following keys are defined:

```
Ascent           // double  
BaseFont         // string  
CapHeight        // double  
Descent          // double  
Encoding         // array of int -> 256 UTF-16 Unicode indexes if set  
FirstChar        // int  
Flags            // int  
FontFamily       // string  
FontName         // string  
FontType         // int (TFontType)  
ItalicAngle      // double  
LastChar         // int  
SpaceWidth       // double  
Widths           // array of double  
XHeight          // double  
DefWidth         // double  
FontFile         // binary string or file path (if Length1 is zero)  
Length1          // int  
Length2          // int  
Length3          // int  
FontFileType     // int (TFontFileSubtype)
```

It is guaranteed that the sum of Length1 + Length2 + Length3 does not exceed the size of the font buffer (FontFile).

GetFontEx

Syntax:

```
array GetFontEx(  
    $Handle) // Font handle (int)
```

The function returns the properties of a font like [GetFont\(\)](#) but accepts a font handle instead.

GetImageObj

Syntax:

```
array GetImageObj(  
    $Handle, // int  
    $Flags) // int -> TParseFlags
```

The function returns the structure TPDFImage as an associative array on success or NULL on failure.

The following keys are defined:

```
Buffer           // binary string  
Filter           // int (TDecodeFilter)  
OrgFilter        // int (TDecodeFilter)  
JBIG2Globals     // binary string  
BitsPerPixel     // int  
ColorSpace       // int (TExtColorSpace)  
NumComponents    // int  
MinIsWhite       // bool  
IColorSpaceObj   // safe pointer  
ColorTable       // binary string  
Width           // int  
Height          // int  
ScanLineLength   // int  
InlineImage      // bool  
Interpolate      // bool  
Transparent      // bool  
ColorMask        // binary string (one byte per mask value)  
IMaskImage       // safe pointer  
ISoftMask        // safe pointer  
Decode           // float array  
Intent           // int (TRenderingIntent)  
SMaskInData      // bool  
OCG              // safe pointer  
Metadata         // binary string  
ResolutionX      // double  
ResolutionY      // double
```

Only keys with a value will be returned.

GetInBBox

Syntax:

```
array GetInBBox(  
    $PageNum, // int  
    $Boundary) // int (TPageBoundary)
```

The function returns the structure TPDFRect as an associative array on success or NULL on failure or if the bounding box is not set.

GetInDocInfo

Syntax:

```
string GetInDocInfo(  
    $DInfo) // int (TDocumentInfo)
```

The function returns the document info entry on success or NULL on failure or if no value is defined for the key.

GetInDocInfoEx

Syntax:

```
array GetInDocInfoEx(  
    $Index) // Entry index(int)
```

The function returns the parameters and value of the document info entry as an associative array on success or NULL on failure.

The following keys are defined:

```
Key    // string  
DInfo  // int (TDocumentInfo) -> This key is always set  
Value  // string
```

GetInPrintSettings

Syntax:

```
array GetInPrintSettings( )
```

The function returns the structure TPDFPrintSettings as an associative array on success or NULL on failure. The member *PrintRangesCount* is not returned since the array length is already implicitly defined in PHP.

GetJavaScript

GetJavaScriptAction

Syntax:

```
string GetJavaScriptAction(  
    $Handle) // Handle (int)
```

The above functions have only one parameter in PHP.

GetJavaScriptAction2

Syntax:

```
array GetJavaScriptAction2(  
    $ObjType,    // int (TObjType)  
    $ObjHandle,  // Object handle (int)  
    $ActIndex)   // Action index (int)
```

The two additional out parameters which are present in other programming languages are returned as an associative array.

The following key / value pairs will be returned:

```
result // string  
Event  // int (TObjEvent)
```

GetJavaScriptEx

Syntax:

```
string GetJavaScriptEx(  
    $Name) // Name of the global JavaScript (string)
```

The function has only one parameter in PHP.

GetJavaScriptName

Syntax:

```
string GetJavaScriptName(  
    $Handle) // int
```

The function has only one parameter in PHP.

GetLogMetafileSize

Syntax:

```
array GetLogMetafileSize(  
    $FileName) // string
```

The function has only parameter in PHP and the bounding box is returned as associative array on success or NULL on failure.

GetLogMetafileSizeEx

Syntax:

```
array GetLogMetafileSizeEx(  
    $Buffer) // binary string
```

The function has only parameter in PHP and the bounding box is returned as associative array on success or NULL on failure.

GetMatrix

Syntax:

```
array GetMatrix( )
```

The function has no parameters in PHP and the structure TCTM is returned as associative array on success or NULL on failure.

GetInMetadata

Syntax:

```
binary string GetInMetadata(  
    $PageNum) // int
```

The function returns the metadata stream as binary string on success.

GetMetadata

Syntax:

```
binary string GetMetadata(  
    $ ObjType, // int (TMetadataObj)  
    $ Handle) // int
```

The function returns the metadata stream as binary string on success.

GetMissingGlyphs

```
array GetMissingGlyphs()
```

The function has no parameters in PHP. The return value is an array of integers if one or more glyphs could not be found or NULL if the list is empty.

GetNameDest

Syntax:

```
array pdfGetNamedDest(  
    $Index) // int
```

The function returns the structure TPDFNamedDest as an associative array on success or NULL on failure.

The following key / value pairs will be returned:

```
Name // string  
DestFile // string  
DestPage // int  
DestPos // array (Left, Bottom, Right, Top)  
DestType // int (TDestType)
```

GetPageAnnotEx

Syntax:

```
array GetPageAnnotEx(  
    $Index) // Array index (int)
```

The function returns the structure TPDFAnnotationEx as an associative array on success or NULL on failure like [GetAnnotEx\(\)](#).

GetPageBBox (Rendering Engine)

Syntax:

```
array GetPageBBox(  
    $PageNum, // Page number (int)  
    $Type) // int (TPageBoundary)
```

The first parameter is the page number instead of a pointer as it is the case in C or C++. The structure TFltRect is returned as an associative array on success or NULL on failure.

GetPageField (obsolete)

Syntax:

```
array GetPageField(  
    $Index) // Array index (int)
```

The function returns the structure TPDFField as an associative array on success or NULL on failure like [GetField\(\)](#).

GetPageFieldEx

Syntax:

```
array GetPageFieldEx(  
    $Index) Array index (int)
```

The function returns the structure TPDFFieldEx as an associative array on success or NULL on failure like [GetFieldEx\(\)](#).

GetPageLabel

Syntax:

```
array GetPageLabel (  
    $Index) // Array index (int)
```

The function returns the structure TPDFPageLabel as an associative array on success or NULL on failure.

The following key / value pairs are available:

```
StartRange    // int  
Format        // int (TPDFPageLabelFormat)  
FirstPageNum  // int  
Prefix        // string
```

Only keys with a value will be returned.

GetPageOrientation (Rendering Engine)

Syntax:

```
int GetPageOrientation(  
    $PageNum) // Page number (int)
```

The function requires a pointer of a page object in C or C++. Since the usage of a pointer would cause an unnecessary overhead in PHP, the function uses the page number instead.

GetPageText

Syntax:

```
array GetPageText (  
    $DestSpace) // Optional, int (TPDFColorSpace)
```

The function has one optional parameter in PHP. If the parameter is present, then it specifies the destination color space into which the text color should be converted.

The structure TPDFStack is returned as an associative array or NULL if no more text is available.

[InitStack\(\)](#) must be called as usual before the function can be called the first time.

Execute the function in a while statement as follows:

```
$pdf->InitStack();  
while (($arr = $pdf->GetPageText()) != NULL)  
{
```



```

// The kerning arrays can be accessed like every PHP array:
$kerning = &$arr['Kerning'];
$count   = count($kerning);

for ($i = 0; $i < $count; $i++)
{
    $rec = &$kerning[$i];
    // Do something with the text
    if (strlen($rec['Text']) > 0)
    {
        ...
    }
}
}

```

Take a look into the example `text_extraction` for a complete example.

GetPrintSettings

Syntax:

```
array GetPrintSettings()
```

The function returns the structure `TPDFPrintSettings` as an associative array on success or `NULL` on failure. The member *PrintRangesCount* is not returned since the array length is already implicitly defined in PHP.

GetSeparationInfo

Syntax:

```
array GetSeparationInfo()
```

The function has no parameters in PHP. The two additional out parameters of the C/C++ version are returned as an associative array or `NULL` on failure.

The returned key / value pairs are:

```

Colorant // string
CS       // int (TExtColorSpace)

```

GetSigDict

Syntax:

```
array pdfGetSigDict(
    $ISignature) // Safe pointer
```

The function returns the structure `TPDFSigDict` as an associative array on success or `NULL` on failure.

The following keys are defined:

```

ByteRange    // array of int (start / end range pairs)
Cert         // binary string
Changes      // array of int
ContactInfo  // string
Contents     // binary string
Filter       // string
Location     // string
SignTime     // string

```

```
Name           // string
PropAuthTime   // int
PropAuthType   // string
Reason         // string
Revision       // int
SubFilter      // string
Version        // int
```

Only keys with a value will be returned.

GetTextFieldValue

Syntax:

```
array GetTextFieldValue(
    $AField) // Field handle
```

The field value and default value are returned as an associative array on success or NULL on failure.

The following key / value pairs are defined:

```
result    // bool (always true if present)
Value     // string
DefValue  // string
```

The key 'result' is returned to avoid a return value of NULL if the field contains no value and no default value.

GetTextRect

Syntax:

```
array GetTextRect( )
```

The function has no parameters in PHP. The out parameters PosX, PosY, Width, and Height, which are present in C/C++, are returned as an associative array on success or NULL on failure.

GetTextWidthF (Font API)

Syntax:

```
double GetTextWidthF(
    $IFont,           // Safe pointer
    $Text,            // binary string
    $CharSpacing,     // double
    $WordSpacing,     // double
    $TextScale)       // double
```

This function refers to `fntGetTextWidth()` of the font API. The function can be used to calculate the width of a binary string returned by `GetPageText()`. The original source strings are stored in the array `RawKern`. Only these strings can be passed to the function! Don't pass strings of the Kerning array to the function since this would cause invalid results. The parameter *Len* is omitted since PHP strings have an implicit length.

GetViewerPreferences

Syntax:

```
array GetViewerPreferences( )
```

The function has no parameters in PHP. The out parameters Preference and AddVal, which are present in C/C++ are returned as an associative array on success or NULL on failure.

InitStack

Syntax:

```
bool InitStack()
```

The function has no parameters in PHP.

InsertRawImageEx

Syntax:

```
int InsertRawImageEx(
    $PosX,          // double
    $PosY,          // double
    $ScaleWidth,    // double
    $ScaleHeight,   // double
    $Image)         // Associative Array (TPDFRawImage)
```

The parameter *\$Image* is a structure in C/C++. The extension initializes the structure TPDFRawImage with zero before setting any value from PHP to it. Values which cannot be zero are required.

The following keys are defined:

```
Buffer           // Binary string
BitsPerComponent // int (required)
NumComponents    // int (required)
CS               // int (TExtColorSpace)
CSHandle         // int
Stride           // int (required)
HasAlpha         // bool
IsBGR           // bool
MinIsWhite       // bool
Width            // int (required)
Height           // int (required)
```

The keys can be defined case-insensitive. Undefined keys will be ignored.

PolygonAnnot

PolyLineAnnot

Syntax:

```
int PolygonAnnot(
    $Vertices, // array of float
    $LineWidth // double
    ...)      // more parameters follow
```

The parameter *Vertices* must be defined as one dimensional array of float. The parameter NumVertices as defined in the C interface is omitted since the array length is always given in PHP.

ReplacePageText

ReplacePageTextEx

Syntax:

```
bool ReplacePageText(  
    $NewText,          // string (required)  
    $DeleteKerningAt) // Optional
```

The functions support an optional parameter *DeleteKerningAt* in PHP. In C/C++ this parameter would be set in the structure *TPDFStack*. Since the structure cannot be accessed directly in PHP, the parameter can be set here.

To delete the original string set the parameter *NewText* to NULL or to an empty string.

RGB

Syntax:

```
int RGB(  
    $r, // int (0..255)  
    $g, // int (0..255)  
    $b) // int (0..255)
```

The function convert RGB values to an integer value. The function does exactly the same as the *PDF_RGB()* macro of the C interface. The resulting value can be passed to functions which accept RGB colors in integer format.

This function can only fail if one or more parameters are missing. If the function fails the return value is NULL and an error message is passed to the PHP engine.

SetLineAnnotParms

Syntax:

```
bool SetLineAnnotParms(  
    $Handle,          // int  
    $FontHandle,      // int  
    $FontSize,        // double  
    $Parms)           // associated array or NULL
```

The parameter *Parms* must be defined as associated array if present. It is not required to set every key, missing keys will be set to zero.

The following keys are defined:

```
Caption           // bool  
CaptionOffsetX    // double  
CaptionOffsetY    // double  
CaptionPos        // int (TLineCaptionPos)  
LeaderLineLen     // double  
LeaderLineExtend  // double  
LeaderLineOffset  // double
```

SetColorMask

Syntax:

```
bool SetColorMask(  
    $ImageHandle, // int  
    $Mask)        // Array of int
```

The C or C++ version contains an additional parameter *Count* which is omitted in PHP. The array length is already implicitly defined in PHP.

SetOnErrorProc

Syntax:

```
bool SetOnErrorProc(callback $ErrProc)  
  
// The function must return an integer value (0 to continue)  
function ErrorProc(  
    $ErrCode,    // Error code (int)  
    $ErrorMessage, // Error message (string)  
    $ErrType)    // Error type (int)  
{}
```

The callback function replaces the internally used callback function that passes error messages to the PHP engine. Note that this callback function handles DynaPDF errors only.

Syntax errors like invalid or missing parameters are still handled by the PHP engine.

Parameters are passed to a callback function by index and not by name. That means the first parameter contains always the error code, the second one the error message and so on, also if the parameter names are different.

It is no error if the callback function contains less parameter as available for the function. If you don't need the error type, for example, then the parameter can be omitted.

SetOnPageBreakProc

Syntax:

```
bool SetOnPageBreakProc(callback $OnBreakProc)  
  
// The function must return an integer value  
function OnBreakProc(  
    $LastPosX, // double  
    $LastPosY, // double  
    $PageBreak) // bool  
{}
```

The function contains only parameter in PHP.

Parameters are passed to a callback function by index and not by name. That means the first parameter contains always the last x-coordinate and the second one the last y-coordinate, independent of the used parameter names.

Usage:

```
$pdf->SetOnPageBreakProc( 'ErrorProc' );
```

SetProgressProc

Syntax:

```
bool SetProgressProc(  
    callback $InitProgress, // Initialization callback function  
    callback $Progress)    // Progress callback function  
  
// This function has no return value  
function InitProgress(  
    $ProgType, // int (see defined constants below)  
    $MaxCount) // int  
{  
  
// The function must return an integer value (0 to continue)  
function Progress(  
    $ActivePage) // Current page number (int)  
{  
  
// ProgType constants  
const ptImportPage = 0 // Start page import  
const ptWritePage  = 1 // Start writing a page
```

To disable the callback functions set both parameters to NULL.

Usage:

```
$pdf->SetProgressProc('InitProgress', 'Progress');
```

TranslateRawCode (Font API)

```
array TranslateRawCode(  
    $IFont,      // Safe pointer  
    $Text,       // binary string  
    $CharSpacing, // double  
    $WordSpacing, // double  
    $TextScale)  // double
```

The C version of this function contains three additional out parameters. These parameters are returned as an associative array.

The following key / value pairs will be returned:

```
result // int -> This is the return value of the DynaPDF function  
Width  // double -> The width of the character  
OutText // string (UTF-8 Unicode)  
Decoded // bool -> If false, then OutText contains no human readable  
          // string
```

The return value of the DynaPDF function is stored in the key 'result'. This value must be used to increment the string position. It is guaranteed that this value is always greater or equal 1 so that no endless loop can occur.

TranslateString2 (Font API)

Syntax:

```
string fntTranslateString2(  
    $IFont, // Safe pointer  
    $Text)  // binary string
```

The function contains only two parameters in PHP. The return value is the converted Unicode string on success or NULL on failure.