Version 2
Developer Manual

www.PDFCONTROL.com

19.05.2003

Table of Contents

1 Overview

1.1 PDFControl

The DLL wPDFControl was created to make it easy for programmers to add PDF support to their application. It is based on the popular PDF engine 'wPDF' which has been successful on the Delphi and C++Builder market since the year 2000.

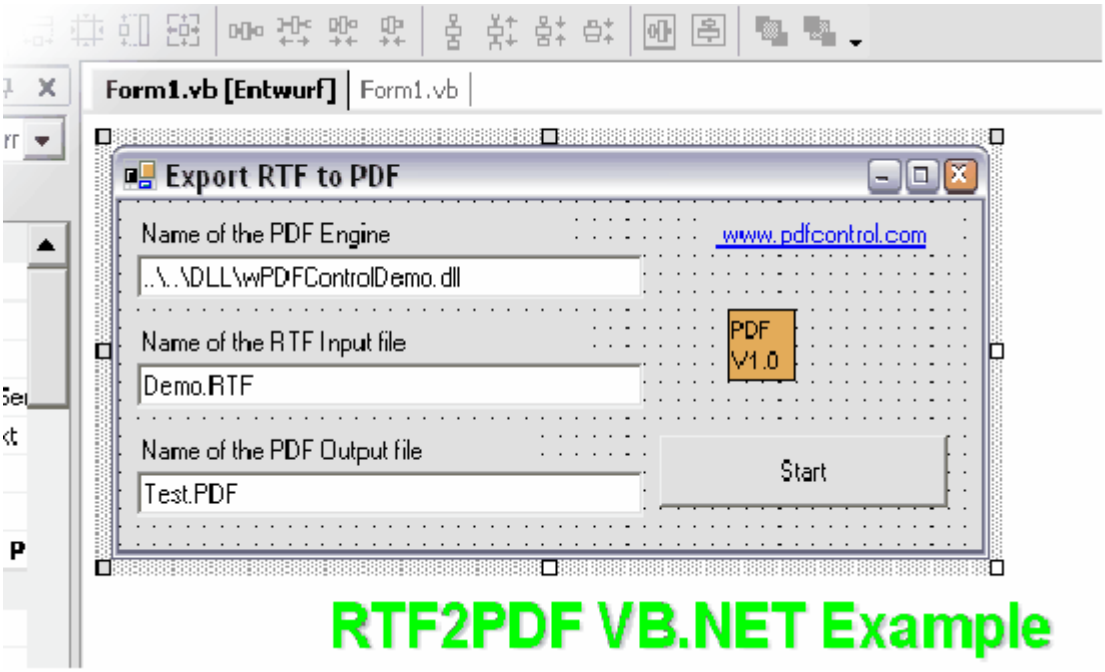
Until now the power of wPDF was not available for other programming languages than Delphi and C++Builder, this has changed since now the wPDFControl DLL can be used in many programming languages which support dynamic link libraries (DLL).

(Last update of this manual: 2.5.2003)

1.2 RTF2PDF

The DLL wRTF2PDF is the big sister of wPDFControl.

It does anything what wPDFControl does but also includes a powerful RTF engine to convert RTF documents to PDF files. Since the wRTF2PDF DLL is quite a bit larger than the PDFControl DLL, the customers of RTF2PDF will receive both DLLs and a key which works for both.



The RTF Engine in RTF2PDF is entirely controlled with the method "ExecCommand."

2 Installation

The setup procedure will create a directory with several sub directories.

Directory DLL:
Here the PDF engine DLL(s) will be copied. The name of the full versions of the PDF-engine is "wPDFControl01.DLL" for wPDFControl and "wRTF2PDF01.DLL" for the PDF-Control with integrated RTF-engine.
"wPDFControlDemo" is the DLL name of for the demo version which also includes the RTF-engine. (It creates a link on each page and displays a nag-screen from time to time.)

The full version of the PDF-engine will create a link on each page unless the license keys have been used properly.

Directory ActiveX:
Here the file wPDF_X01.ocx can be found. It is the interface ActiveX which makes it easy to use PDFControl in Visual Basic and similar development systems which have difficulties to either load a DLL or use callbacks*). The interface ActiveX has no own intelligence and simply uses the PDF-Engine DLL. It implements properties, methods and events which make it possible to use all the features of wPDFControl and wRTF2PDF in interpreter languages.
The setup also copies the ActiveX to the system directory and registeres it using "RegSvr32".

> *During the development we decided to not use an approach based on the COM or ActiveX technology: Although ActiveX controls are easy to use in systems such as VB they, on the other hand, cause a big overhead for the server and the container application. A DLL is easy to use, to load and to unload and the developer has not to worry about installation or version conflict - as long as the DLL is stored in the applications directory and given a unique name. Other advantages of a DLL are the fast initialization, the fast access of the contained procedures through function pointers, the smaller size and the possibility to create custom versions of the same engine.*

In VisualBasic you probably need to **add the ActiveX to the 'toolbox'.** When the ActiveX was properly registered (the setup does it) it show up in the list of the installed ActiveX controls using the name "PDFControl Element'.

Directory Demos:
Here you will find a few demo sources, mainly for C and C++ but also for VB and VBS.
In directory Demos\C\Include you will find a header file wpdf.h which includes the code to load and use the DLL. It implements prototypes for the exported functions. The file wpdf_class implements a C++Class which make the usage of the PDF library even easier.

Deploy Your Application:
If you are using the demo version you may not deploy your application. When you use the registered version you need to deploy the PDF-Engine DLL you used (you can rename it and copy it to the application directory). If you use the ActiveX you also need to deploy it. It needs to be copied to the system directory and to be registered.

*) a *callback* is a function which is called in a special way. Normally the address of a function is passed to a different module (DLL) and this module then calls the function using this address. They can be used to notify the application of certain states or request data. The RTF-engine in wRTF2PDF can use callbacks to do mail merge. Callbacks work fine in AnsiC, Delphi or C++ but don't work in interpreter languages such as Visual Basic or VB-Script.

3 License

License Agreement for the wPDFControl.DLL and wRTF2PDF.DLL
Copyright (C) 2002 by wpcubed GmbH Germany
WEB : http://www.pdfcontrol.com
mail to: support@pdfcontrol.com

*** General

The software supplied may be used by one person on as many computer systems as that person uses. Group programming projects making use of this software must purchase a copy of the software for each member of the group. Contact wpcubed GmbH for volume discounts and site licensing agreements. This documentation and the library are provided "as is" without warranty of any kind, either expressed or implied, including but not limited to the implied warranties of merchantability and/or suitability for a particular purpose. The user assumes the entire risk of any damage caused by this software.

In no event shall Julian Ziersch or wpcubed GmbH be liable for damage of any kind, loss of data, loss of profits, interruption of business or other pecuniary losses arising directly or indirectly from the use of the program.

Any liability of the seller will be exclusively limited to replacement of the product or refund of purchase price.

*** Demo Version

The demo version of the library may only be used to evaluate this product. The projects which were created using the demo version may not be distributed. The demo version will show a nag screen - this makes also the DLL much larger. The demo also includes the RTF engine RTF2PDF - which is not included in the standard wPDFControl.DLL!

The demo displays a nag screen and prints a watermark.

*** wPDFControl Standard License

This License enables you to use our PDF engine technology in all your products and distribute it to your customers without the need to pay any royalties. Important: You may not distribute any of the included source files or object files or use the technology in a module (ActiveX, COM, VCL ...) which can be used by other developers in any kind of programming language or developing environment or which can be embedded into other programs. This also prohibits the use our technology in universal PDF creation tools like a virtual printer driver. The DLL may only be used to create PDF files from the data processed by the same application or application environment. Unless you have purchased the server license you may not use the wPDF technology in any programs (services, CGIs, ActiveServers ...) which will work on Internet servers to create PDF files to be distributed over the WEB. As "Creator" of the PDF file always "wPDF by wpcubed GmbH" will be written to the created PDF file.

Please note that you have to send you license key to the DLL to activate it. Please use wpdfSetLicenseKey()

*** wPDFControl Server License

In Addition to the standard license you may also use the PDF creation technology in any programs (services, CGIs, ActiveServers ...) which will work on one Internet server to create PDF files to be distributed over the WEB.

Plaese note that the DLL can currently not work multithreaded.
Please create CGI EXE (not DLL) applications.

4 Principle of PDF Creation

To use PDFControl in your application you first have to load the DLL an initialize it.

We have create a header file which makes it easy to do this in C and also provide a C++ class which does most of the tasks automatically. If you use the interface ActiveX you need to add one instance of it to your form and use the provided functions. Don't worry about header files.

First you need to initialize the DLL using **Initialize** or InitializeEx.

After you have initialized the DLL you can execute the function **BeginDoc** to start the output. When you are done you execute **EndDoc** to finalize and close the PDF file.

While the PDF file is open you can create a new page with **StartPage**. The page has to be closed with **EndPage**. While a page is active you can use the HDC (Device handle) of the PDF engine to create the output.

You can retrieve this handle with the function **DC** and use it with regular GDI drawing routines.

There are also some functions which make it easier to change the font, and the background and pen colors. If you use these functions the DC will be changes as if you would use the windows API - but they are much easier to use and you don't have to worry about HFONT, HBRUSH and HPEN GDI resources.

The PDF Engine includes a method called **ExecCommand**. This method is used by the RTF2PDF library as an interface to the integrated RTF-engine. Different command codes are used to select the intern functions.

Example C code: (uses wpdf.h. Use the macro 'LOAD_WPDF_ENGINE(dll_handle,dll_name)' to load the DLL and initialize the function pointers)

```
int dll;
if(LOAD_WPDF_ENGINE(dll, "wpdfcontrol01.dll" )=0)
{
  pdf = wpdfInitialize();
  if( wpdfBeginDoc(pdf,"test.pdf",0))
  {
       wpdfStartPageEx(pdf,512,812,0);
       TextOut (wpdfDC(pdf), 72, 72, "Hello World",  11);
       wpdfEndPage(pdf);
       wpdfEndDoc(pdf);
  }
  FreeLibrary(dll);
}
```

Example C++Code: (uses wPDF_Class.cpp)

```
CPDFExport cexp = CPDFExport(PDFENGINE,PDF_LIC_CODE,PDF_LIC_NAME,PDF_LIC_KEY);
if(cexp->EngineWasLoaded())
 {
  cexp->Initialize();
  if(cexp->BeginDoc(PdfFileName,0))
   {
     cexp->StartPageEx(PageH,PageW,0);
     Ellipse(cexp->DC(), 10,100,300,300);
     cexp->EndPage();
   } // if(cexp.BeginDoc(PdfFileName,0))
  cexp->EndDoc();
  cexp->Finalize();
 }
```

Example VB Code: (uses an insance of the ActiveX with the name 'pdf')

```
    If PDF.StartEngine(DLLNAME, LIC_NAME, LIC_KEY, LIC_CODE) <> 0 Then
        PDF.BeginDoc "c:\atest3.pdf", 0
        PDF.StartPage
        PDF.Rectangle 10, 10, 210, 100
        PDF.EndPage
        PDF.EndDoc
        PDF.StopEngine
    End If
```

Example VB.NET Code to convert a RTF file to PDF:

```
       If PDF.StartEngine(DLLNAME.Text, "", "", 0) Then ' you your license info here !
            If PDF.BeginDoc(PDFNAME.Text, 0) > 0 Then ' Start a PDF document
                PDF.ExecIntCommand(1000, 0) ' Initialize the RTF engine
                PDF.ExecStrCommand(1002, RTFNAME.Text) ' Load a RTF file
                PDF.ExecIntCommand(1100, 0) ' Export this RTF file to PDF
                PDF.EndDoc() ' Close the PDF document
            End If
        End If
```

Example VBS Code to convert RTF to PDF: uses the ActiveX)

```
    Set PDF =  CreateObject("wPDF_X01.PDFControl")
    PDF.INFO_Date = Now
    PDF.INFO_Author = "Julian Ziersch"
    PDF.INFO_Subject = "Test file"
    ' ... set other properties

    PDF.StartEngine "c:\wPDFControl\DLL\wRTF2PDF01.dll", "LIC_NAME", "LIC_KEY" ,0 ' License
    Info!
    PDF.BeginDoc "Test.PDF", 0
    PDF.ExecIntCommand 1000, 0
    PDF.ExecStrCommand 1002, "Demo.RTF"
    PDF.ExecIntCommand 1100, 0
    PDF.EndDoc
    PDF.StopEngine
    Set PDF = Nothing
```

5 VB Examples

Here we collected some example code for Visual Basic (6)

5.1 EMF to PDF Conversion

A) This code uses a wPDF OCX to convert a EMF file directly to a PDF file:

LoadEMFFile is a common file open dialog.
DLLName is a edit control which holds the path to the PDF engine DLL.

```
Private Sub ExportPDF_Click()
LoadEMFFile.Action = 1
If LoadEMFFile.FileName <> "" Then
  If PDFControl1.StartEngine(DLLName.Text, "LicenseName", "LicenseCode", 0) = True Then
   PDFControl1.BeginDoc FileName.Text, 0
   PDFControl1.DrawBitmapFile 0, 0, 0, 0, LoadEMFFile.FileName
   PDFControl1.EndDoc
   Else
  MsgBox "We were not able to load the PDF Engine DLL", 0, "Error!"
End If
End If
End Sub
```

B) If the image is loaded in a image box you can use this code:

```
If PDFControl1.StartEngine(DLLName.Text, "LicenseName", "LicenseCode", 0) = True Then
   PDFControl1.BeginDoc FileName.Text, 0
   PDFControl1.DrawMetafile Image1.Picture.Handle, 0, 0, 0, 0
   PDFControl1.EndDoc
Else
  MsgBox "We were not able to load the PDF Engine DLL", 0, "Error!"
End If
```

6 Methods

6.1 Overview

The PDF engine includes several methods and uses different constants. Although the functionality is always the same the functions can be called in a different way. Which way you will choose depends on the development system you are using. If you use AnsiC you will probably use the function pointers and prototypes which are initialized in the include file Demos\C\Include\wPDF.H. If you are using C++ you can either use the function pointers or the class implemented in file Demos\C\Include\wPDF_class.cpp. If you are using VB, or VBS, or a different interpreter language you have to use the ActiveX 'PDFControl' which implements an interface to the PDF-Engine DLL. If you are using Delphi we suggest to check out our component set wPDF which is specialized for Delphi and C++Builder.

If you use the function pointers (C) all methods start with "wpdf_". Otherwise the methods are inside of the class or ActiveX without the need to use a prefix.
C and C++ developers can use the WPDFInfoRecord to set the properties of the PDF-Engine. Developers who use the ActiveX can use the properties of this control.

The WPDFInfoRecord includes this elements:

```
typedef struct
  {
     int SizeOfStruct;
     // Encoding
     int Encoding; // 0 = none, 1=ASCII85, 2=Hex
     // Compression 0=none, deflate, runlength
```

```
int Compression;
     // BitmapCompression
     int BitCompression; // 0=auto, 1=deflat, 2=jpeg
     // Thumbnailes
     int ThumbNails; // 0=none, 1=color, 2=monchrome
     // UseFonts
     int FontMode;
     // PageMode
     int PageMode;
     // InputFileMode
     int InputFileMode;
     // JPEGCompress 0=off, - = value, 1..5 for 10, 25, 50, 75, 100
     int JPEGCompress;
     // Enhanced optiuons:
     // Bit 0: PDF text out ignores the width of the characters calculated by windows
     // Bit 1: PDF engine supports clipping (only rectangles)
     // Bit 3: switches OFF Auto Hyperlinks
     // Bit 4: PDF does  NOT use use the a printer HDC as reference
     int EnhancedOptions;
     // PDF Resolution
     int PDFXRes, PDFYRes; // 0->72
     // Default PDF Size (for StartPageA)
     int PDFWidth, PDFHeight; // Default DINa4
     // Read PDF Filename
     char InputPDFFile[128];
     // Several Callback functions
     // For Stream Output. We can use this callbacks to write to STDIO.
     // this is useful for CGI applications which rely to send to the PDF output
     // to stdout
     TWPDF_Callback* OnStreamOpen;
     TWPDF_Callback* OnStreamWrite;
     TWPDF_Callback* OnStreamClose;
     // Message callbacks
     TWPDF_Callback* OnError;
     TWPDF_Callback* OnMessage;
     // The mailmerge string is used to detect mailmerge fields
     // it is possible to the text such as @@NAME be using the callback function OnGetText
     char MailMergeStart[8];
     TWPDF_Callback* OnGetText;
     // Encryption of the PDF file
     int Permission;
     char UserPassword[20], OwnerPassword[20]; // actually only 5 chars long (40 bits)
     // Info record of the PDF file
     char Date[256],
          ModDate[256],
          Author[256],
          Producer[256],
          Title[256],
          Subject[256],
          Keywords[256];
     // Reserved, must be 0
     int Reserved;
} WPDFInfoRecord;
```

6.2 PDFControl

Here we list the methods available by the standard PDF engine.

6.2.1 Initialize

6.2.1.1 Initialize

The function Initialize() creates a new *PDF environment* and returns its identifier. This number can then be used with BeginDoc to create a PDF file. If possible we recommend to use InitializeEx because this function allows you to pass an initialization structure to set up all the PDF properties and callback functions.

Parameters
none

Examples

Ansi-C
```
pdf = wpdfInitialize();
```

C++
```
cexp->Initialize();
```

Visual Basic
The ActiveX uses the procedure StartEngine() instead of "Initialize".

TYPELIB:
```
StartEngine(
        [In] BSTR DLLName,
        [In] BSTR LicenseName,
        [In] BSTR LicenseCode,
        [In] Long LicenseNumber,
        [out, retval] Variant_BOOL * Value );
```

6.2.1.2 InitializeEx

The function InitializeEx creates a new PDF environment and returns its identifier. This number can then be used with WPDF_BeginDoc to create a PDF file. WPDF_InitializeEx expects a reference to an initialization structure which is used to set up all the PDF properties and also the callback functions. If you are using C you can easily use a local initialized variable for the 'info' structure.

In C or C++ you can use this function to initialize the info record. If you are using the ActiveX simply use its properties.

Please see the chapter "Properties" for an overview.

```
static WPDFInfoRecord Info;

void InitInfo()
{
   // used to check if this record is ok
   Info.SizeOfStruct = sizeof(WPDFInfoRecord);
   // Encoding
   Info.Encoding = 0; // 0 = none, 1=ASCII85, 2=Hex
   // Compression 0=none, deflate, runlength
   Info.Compression = 1;
   // BitmapCompression
   Info.BitCompression = 0; // 0=auto, 1=deflate, 2=jpeg
   // Thumbnailes
   Info.ThumbNails = 1; // 0=none, 1=color, 2=monchrome
   // UseFonts
   Info.FontMode = 0;
   // PageMode
   Info.PageMode = 1;
   // InputFileMode
   Info.InputFileMode = 3;
   // JPEGCompress 0=off, - = value, 1..5 for 10, 25, 50, 75, 100
   Info.JPEGCompress  = 0;
   // Enhanced optiuons:
   // Bit 0: PDF text out ignores the width of the characters calculated by windows
   // Bit 1: PDF engine supports clipping (only rectangles)
   // Bit 3: switches OFF Auto Hyperlinks
   // Bit 4: PDF does  NOT use use the a printer HDC as reference
   Info.EnhancedOptions = 0;

   // PDF Resolution
   Info.PDFXRes = 72;
   Info.PDFYRes = 72;

   // Default PDF Size (for StartPageA)
   Info.PDFWidth = 0;
   Info.PDFHeight = 0;

   // Read PDF Filename
   strcpy(Info.InputPDFFile,"");

   // Several Callback functions
   // For Stream Output
   // Info.OnStreamOpen  = wpdfcall_StreamOpen;
   // Info.OnStreamWrite = wpdfcall_StreamWrite;
   // Info.OnStreamClose = wpdfcall_StreamClose;

   // Messages
   Info.OnError   = wpdfcall_ErrMessage;
   Info.OnMessage = wpdfcall_Message;

   // Mailmerge, etc
   strcpy(Info.MailMergeStart,"@@");
   Info.OnGetText = wpdfcall_GetText;
   //wpdfcall_GetText, // OnGetText callback
   Info.Permission = 0;
   strcpy(Info.UserPassword,"");
   strcpy(Info.OwnerPassword,"");
```

```
// INFO Strings
   strcpy(Info.Date,"");
   strcpy(Info.ModDate,"");
   strcpy(Info.Author,"");
   strcpy(Info.Date,"Julian Ziersch");

   strcpy(Info.Producer,"PDFControl Test application");
   strcpy(Info.Title,"Demo");
   strcpy(Info.Subject,"Demo");
   strcpy(Info.Keywords,"");
   // Reserved, must be 0
   Info.Reserved = 0; // int Reserved;
}
```

Example:

```
     InitInfo();
     if(wpdfSetLicenseKey!=NULL)
        // Please Use the correct values here. Otherwise the created PDF files will
        // only contain 3 pages and show a watermark.
        // the registered, licensed DLL does not show a watermark!
        wpdfSetLicenseKey(PDF_LIC_CODE,PDF_LIC_NAME,PDF_LIC_KEY);
        // <---- Change line above ! ---->
     pdf=wpdfInitializeEx(&Info);
```

6.2.2 Finalize

6.2.2.1 Finalize

This function closes a given PDF environment.

The PDF-Environment is identified by the integer returned by function Initialize().

Visual Basic
The ActiveX uses the procedure StopEngine() instead of "Finalize".

TYPELIB:
StopEngine(void);

6.2.2.2 FinalizeAll

The procedure FinalizeAll() closes all open PDF environments.

6.2.3 BeginDoc

This method opens a new PDF file.

After the file was opened you can create pages and watermarks in this PDF file.

When you are done don't forget to execute EndDoc to close the PDF file.

BeginDoc expects 3 parameters:
env - this is the environment identifier received from Initialize.
FileName - this is the filename for the PDF file.
UseStream - Normally you will use the value 0 to create a PDF file. If you prefer to get the PDF data in the OnStreamWrite callback pass 1. The name passed a FileName will then be provided to the callback OnStreamOpen.

If you are using the ActiveX control you don't need to specify the PDF-Enviroment. You can execute PDF.BeginDoc pdfname.pas,0 right away.

TYPELIB:
```
BeginDoc(
        [in] BSTR FileName,
        [in] long UseStream,
        [out, retval] long * Value );
```

6.2.4 EndDoc

This function closes the PDF file which was opened with BeginDoc or BeginDocEx. If you are using streams the OnStreamClose callback will be executed, if you are using files the PDF file will be closed.

TYPELIB:
```
EndDoc( void );
```

6.2.5 StartPage

6.2.5.1 StartPage

The function StartPage initializes the output to one PDF page.

The page-size will be set to the value preset in the info structure used with **InitializeEx**. The default resolution is 72. If you are using the ActiveX you can change the default page size by changing its properties.

If you want use a different page size or want to print in landscape mode use StartPageEx instead of StartPage.

Note: The output using the graphic operations or the device handle (DC) is only possible if either a PDF page was created with StartPage or a watermark was created with StartWatermark.

TYPELIB:
```
StartPage( void );
```

6.2.5.2 StartPageEx

The function StartPageEx initializes the output to one PDF page.

The page -width and -height will be set to the provided values (measured in the units preset in the info structure - items PDFXRES and PDFYRES) used with WPDF_InitializeEx.

You can also set the rotation value. Currently you can use 0 for standard output and 90 for landscape output.
Please remember that a watermark will be also drawn rotated on a rotated page.

TYPELIB:
```
StartPageEx(
        [in] long Width,
        [in] long Height,
        [in] long Rotation );
```

6.2.6 EndPage

This function closes a PDF page which was opened with StartPage or StartPageEx.

TYPELIB:
EndPage(void);

6.2.7 StartWatermark

The function StartWatermark initializes the output to one PDF watermark.
The page -width and -height will be set to the provided values (measured in the units preset in the info structure - items PDFXRES and PDFYRES) used with InitializeEx or in the ActiveX properties.

The watermark has to be closed with EndWatermark. It can later be used for a certain page by executing the procedure **DrawWatermark** which expects the name of the watermark which was provided here, during the creation of the watermark.

StartWatermarkEx works like StartWatermark but lets the developer specify the page size.

TYPELIB:
```
StartWatermark(
        [in] BSTR Name );

StartWatermarkEx(
        [in] BSTR Name,
        [in] long Width,
        [in] long Height );
```

6.2.8 EndWatermark

This function closes a PDF page which was opened with StartWatermark or StartWatermarkEx.

TYPELIB:
EndWatermark(void);

6.2.9 DrawWatermark

The PDF engine supports watermarks. The watermarks can be created once and used on as many pages as you like.

The DLL can even import PDF data and convert the imported pages into watermarks which are names "inpageN" (with N = 1..count of pages). With the function DrawWatermark you can tell the PDF engine to use one of the stored watermarks. You simply have to pass the name of it. If you want to rotate the watermarks to any degree you can do so. But please note that a PDF file with a rotated watermark will not print on all printers. Especially postscript printers seem to not like this feature.

Important is also that on a landscape page the watermark will be also landscape. This means that the watermark has to be portrait before you use it. It will automatically be rotated by using it on a landscape page.

To create a watermark use the function StartWatermark.

TYPELIB:

```
DrawWatermark(
        [in] BSTR Name,
        [in] long Rotation );
```

6.2.10 ExecCommand

This function is the interface to custom enhancements of the DPF control. The type of command is identified by the number passed as 'id'. The RTF engine included in the RTF2PDF DLL is solely controlled by this function.

TYPELIB:
ExecCommand(
[in] long id,
[in] long param2,
[in] long buf,
[in] long buflen,
[out, retval] long * Value);

ExecStrCommand(
[in] long id,
[in] BSTR Name,
[out, retval] long * Result);

ExecIntCommand(
[in] long id,
[in] long Value,
[out, retval] long * Result);

List of command ids which are supported by PDFControl:

WPCOM_Version=1
Retrieve the version of the PDF Engine. The Result value is the version * 100

WPCOM_AUTOLINK=2
You can use this command to print a link on each page (like the PDFControl demo does).

The link is provided as struct referenced by the parameter "buf".
"buflen" must be initialized with SizeOf(TPDFAuto_Link).

```
typedef struct
{
     int BitmapCloneNr; // has priority  (or 0)
     int BitmapHandle;  // HBITMAP handle (or 0)
     char *VarBuf;      // extra pointer parameter
     int VarBufLen;     // Len of buffer
     // Options:
     //    1: Under Text (but over watermark!)
     //    2: Interpret VarP as pointer to a filename
     int Options;
     // Destination Rectangle (in PDF resolution, not "HDC" Resolution!)
     int DrawX;
     int DrawY;
     int DrawW;
     int DrawH;
     // Destination URL and relative link rectangle
     char LinkURL[100]; // Bookmark or "http://" URL
     int  LinkX; // added to DrawX
     int  LinkY; // added to DrawY
     int  LinkW; // if 0 then = DrawW
     int  LinkH; // if 0 then = DrawH
} WPDFAutoLink;
```

6.2.11 DrawMetafile

Declaration
Procedure WPDF_DrawMetafile(env: pdfEnviroment; meta : Cardinal; x,y,w,h : Integer);

This function exports the GDI commands stored in a metafile to the current PDF page. Please note that you can also use the PlayMetafile API to render a metafile to the WPDF_DC device. The latter sometimes works better if you need to stretch the graphics information.

New in version 2: You can export a metafile without a call to StartPage. In this case a page in the correct dimension will be automatically created.

Parameters:
env - the environment descriptor received from the function WPDF_Initialize. (not in ActiveX)
meta - the HMETA handle used by windows to identify a metafile.
x,y,w,h - the destination coordinates, the width and the height of the graphic.

TYPELIB:
```
DrawMetafile(
        [in] long meta,
        [in] long x,
        [in] long y,
        [in] long w,
        [in] long h );
```

6.2.12 DrawDIB

This function draws a bitmap at a fixed position to the current PDF page. The bitmap is defined by the BitmapInfo and BitmapBits structures which can be retrieved from the windows GID by the Windows GDI function GetDIB.

Note: This function is a good solution if you have already a true color bitmap in memory - for example an image provided by a scanner. If you have just a windows bitmap use DrawBMP and provide the handle (HBITMAP) of this bitmap.

```
Function DrawDIB(
        env: pdfEnviroment;
        x, y, w, h: Integer;
        BitmapInfo: PBitmapInfo;
        BitmapBits: Pointer): Integer;
```

TYPELIB:
```
DrawDIB(
        [in] long x,
        [in] long y,
        [in] long w,
        [in] long h,
        [in] long BitmapInfo,
        [in] long BitmapBits,
        [out, retval] long * Value );
```

6.2.13 DrawBMP

Declaration
Function DrawBMP(env: pdfEnviroment; x, y, w, h: Integer; **Bitmap** : **HBITMAP**): Integer;

Description
This function draws a bitmap at a fixed position to the current PDF page. The bitmap is defined by the bitmap handle, the HBITMAP identifier which is used by the windows API to locate and manage bitmaps. If you don't use this bitmap handles you can use the procedure WPDF_DrawDib to export bitmap data directly.

Parameters:
env - the environment descriptor received from the function Initialize.
x,y,w,h - the destination coordinates, the width and the height of the graphic.
Bitmap - the HBITMAP handle of the bitmap.

Return Value
If the function succeeds it returns an identification number which can be used by the procedure DrawBitmapClone to draw the same bitmap again. In this case the data is not added to the PDF file again which helps to reduce the file size of the PDF file. If an error hapens the retun value is <=0;

TYPELIB:
```
DrawBMP(
        [in] long x,
        [in] long y,
        [in] long w,
        [in] long h,
        [in] long Bitmap,
        [out, retval] long * Value );
```

6.2.14 DrawJPEG

This function can be used to draw a JPEG file which has been loaded already.
Only a pointer to the JPEG data and the width and height of the JPEG bitmap has to be provided to the PDF engine.

Please note the the jpeg_w and jpeg_h are not the width and height of the visible bitmap, they are the amount for scan-lines and scan-columns of the bitmap and have to be exact!

Parameters:
env - the environment descriptor received from the function Initialize.
x,y,w,h - the destination coordinates, the width and the height of the graphic.
jpeg_w, jpeg_h - the number of scan-columns and scan-lines of the JPEG data in memory.
buffer, buflen - A pointer to the buffer and its size.

TYPELIB:
```
DrawJPEG(
        [in] long x,
        [in] long y,
        [in] long w,
        [in] long h,
        [in] long jpeg_w,
        [in] long jpeg_h,
        [in] long buf,
        [in] long buflen,
```

```
        [out, retval] long * Value );
```

6.2.15 DrawBitmapFile

This function draws a bitmap at a fixed position to the current PDF page. The bitmap will be loaded from the specified file. Currently *.BMP, *.JPG and *.JPEG files can be loaded. The function will return a bitmap identifier which can be used to use the bitmap again in the function WPDF_DrawBitmapClone.

TIP: The function can also export metafiles (*.WMF and *.EMF) but in this case no bitmap identifier will be returned.

Parameters:
env - the environment descriptor received from the function WPDF_Initialize.
x,y,w,h - the destination coordinates, the width and the height of the graphic.
filename - a character pointer to the filename (*.BMP, *.WMF, *.WMF, *.JPG, *.JPEG).

TYPELIB:

```
DrawBitmapFile(
        [in] long x,
        [in] long y,
        [in] long w,
        [in] long h,
        [in] BSTR FileName,
        [out, retval] long * Value );
```

6.2.16 DrawBitmapClone

The functions DrawBMP, DrawDib, DrawBitmapFile and DrawJPEG return the identifier of the exported bitmap.

The function DrawBitmapClone can be used to draw a bitmap again.

Parameters:
env - the environment descriptor received from the function Initialize.
x,y,w,h - the destination coordinates, the width and the height of the graphic.
BitmapID - the identifier of the bitmap (as returned by DrawBMP, DrawJPEG etc).

TYPELIB:

```
DrawBitmapClone(
        [in] long x,
        [in] long y,
        [in] long w,
        [in] long h,
        [in] long BitmapID );
```

6.2.17 DC

This is probably the most important function in this DLL. It retrieves a device handle which can be used with Windows graphics API functions.

All the graphic operations which are executed with this 'DC' are sent to the current PDF page. (The page must have been opened with the procedure StartPage)
When the page is closed with procedure EndPage all the graphic output will be converted to PDF data.

Note:
The DLL defines a set of functions which should make it easier to use the graphics API. Esspecially defining a font or a brush is not easy with the GDI CreateFont and CreateBrush API. If you want to avoid this overhead you can also (optionally) use the methods SetTextAttr, SetTextAttrEx, SetBrushAttr, SetPenAttr, SetDefaultAttr.
The following procedures can be used for the output: MoveTo(), LineTo(), Rectangle(), TextOut() and TextRect(). TextRect is a very powerful function to draw text which can be left, center or right aligned or justified.

Since the DLL has to translate the GDI API to PDF commands not all GDI commands are understood. Esspecially 'regions' are not supported. Regions are usually used to draw filled text or circles. Those cannot be translated correctly. In the case you have such complicated output it is better to draw the graphics on a bitmap and then draw this bitmap on the DC.

TYPELIB:
GetDC([out, retval] unsigned long * Value);

6.2.18 TextOut

This function draws text to the PDF device using the current font attributes. It is comparable to the standard Windows TextOut API. If you need aligned or wrapped text output you can use the TextRect.

Note:You can of course always use the standard windows API with the DC provided by DC().

TYPELIB:
```
TextOut(
        [in] long x,
        [in] long y,
        [in] BSTR Text );
```

6.2.19 TextRect

This function draws text to the PDF device using the current font attributes. In contrasts to the simple TextOut() this function is able to handle aligned or wrapped text output: It automatically wraps text to fit into a rectangle and returns a pointer to the first character which did not fit into this rectangle.
You can execute the function again with this returned pointer to print the rest of the text, probably on the text PDF page.

Parameters:
int x,y,w,h: This values define the destination rectangle to print in. If a word does not fit into one line it will be wrapped to the next line.
char *Text: This is the pointer to the 0 terminated char buffer which holds the text.
int Alignment: The alignment: 0=left, 1=center, 2=right, 3=justified text.

Return value:
The function returns a pointer to the first character which was not printed. If the text was printed completely the return value is NULL. You can use the TextRect.

Note:You can of course always use the standard windows API with the DC provided by DC().

TYPELIB:

```
TextRect(
        [in] long x,
        [in] long y,
        [in] long w,
        [in] long h,
        [in] BSTR Text,
        [in] long StartPos,
        [in] long Alignment,
        [out, retval] long * Value );
```

6.2.20 MoveTo

The function **LineTo()** draws a line from the current point to the give point. The current point can be moved with **MoveTo()**. You can of course also use the general Windows API!

TYPELIB:

```
MoveTo(
        [in] long x,
        [in] long y );
```

6.2.21 LineTo

The function **LineTo()** draws a line from the current point to the give point. The current point can be moved with **MoveTo()**. You can of course also use the general Windows API!

TYPELIB:

```
LineTo(
        [in] long x,
        [in] long y );
```

6.2.22 Rectangle

The function **Rectangle** draws a rectangle on the PDF page. The line style can be changed with *SetPenAttr*, the filling color can be changed with WPDF_*SetBrushAttr*.

TYPELIB:

```
Rectangle(
        [in] long x,
        [in] long y,
        [in] long w,
        [in] long h );
```

6.2.23 Hyperlink

The function **Hyperlink()** creates a new hyperlink in the PDF file.

The parameters are the position on the PDF page and the name of the destination bookmark.

If the bookmark name starts with "href://" an internet link will be created.
Bookmarks can be added to the PDF file with **Bookmark()**.

Hyperlink() uses the current device resolution of the PDF DC. This resolution can be changed with the SetViewport windows API.
The PDF engine will make sure your existing drawing code is converted correctly.
Due to this conversion of the PDF engine you have not to change your painting code which you are

used to use for printed output.

TYPELIB:
Hyperlink(
[in] long x,
[in] long y,
[in] long w,
[in] long h,
[in] BSTR Name);

6.2.24 Bookmark

With the procedure WPDF_Bookmark it is possible to create a bookmark in the PDF file. The bookmark can be used by the procedures **Outline()** and **Hyperlink()**.

The procedure expects this parameters:
env - the environment descriptor received from the function WPDF_Initialize.
x,y - the destination coordinates on the PDF page.
name - a character pointer which provides the name of the bookmark.

TYPELIB:
Bookmark(
[in] long x,
[in] long y,
[in] BSTR Name);

6.2.25 Outline

This function creates a new outline entry for the current PDF file. You can either define the destination of the outline entry as a bookmark (name) or define it as a point on the current page. The level controls if the outline will be created on the same level (0), a level deeper (+1) or a level higher (-1). If you pass -1 as level and an empty 'caption' the intern outline pointer will be moved one level up.

Example C code:

```
for(i=1;i<20;i++)
 { wpdfOutline(pdf,0,10,i*10,NULL,"Entry");
        // One level down
    wpdfOutline(pdf,1,10,i*10,NULL,"Entry 1");
        // One level down
    wpdfOutline(pdf,1,10,i*10,NULL,"Entry 2");
        // Next entry shoud be 2 levels up!
    wpdfOutline(pdf,-1,10,i*10,NULL,NULL);
    wpdfOutline(pdf,-1,10,i*10,NULL,NULL);
}
```

TYPELIB:
Outline(
[in] long level,
[in] long x,
[in] long y,
[in] BSTR Name,
[in] BSTR Caption);

6.2.26 SetAttrib...

6.2.26.1 SetTextDefaultAttr

This function resets the settings for the line (widht=1, black), font (Arial standard, size=11, black) and filling (no filling) attributes.

TYPELIB:
SetTextDefaultAttr(
[in] BSTR Font,
[in] long Size);

6.2.26.2 SetTextAttr

This function makes it easy to change the current font style and size. Function **SetTextAttrEx()** lets you also change the font color.

TYPELIB:
SetTextAttr(
[in] BSTR Font,
[in] long Size,
[in] long Bold,
[in] long Italic,
[in] long Underline);

6.2.26.3 SetTextAttrEx

This function makes it easy to change the current font style and size and color.

TYPELIB:
SetTextAttrEx(
[in] BSTR Font,
[in] long Size,
[in] long Charset,
[in] long Bold,
[in] long Italic,
[in] long Underline,
[in] long Color);

6.2.26.4 SetPenAttr

This function makes it easy to change the current line color and line width.

Parameters
Style - 0 no line, 1 solid line
Width - the width of the line
Color - The line color as RGB value

TYPELIB:
SetPenAttr(
[in] long Style,
[in] long Width,
[in] long Color);

6.2.26.5 SetBrushAttr

The function SetBrushAttr makes it easy to change the fill color for the current PDF device.

Parameters
Style - 0 no fill, 1 solid fill
Color - The fill color as RGB value

TYPELIB:
```
SetBrushAttr(
        [in] long Style,
        [in] long Color );
```

6.2.27 SetProperty

Normally all properties are set using the properties of the ActiveX or using the info record and Initialize().

But the following two methods can update certain properties when a PDF file is already open.

6.2.27.1 SetSProp

This method changes a string property. Using an **ID** the property is selected.

TYPELIB:
```
SetSProp(
        [in] long id,
        [in] BSTR Value,
        [out, retval] VARIANT_BOOL * Result );
```

This ids can be used:
```
#define WPPDF_Author 1
#define WPPDF_Date     2
#define WPPDF_ModDate  3
#define WPPDF_Producer 4
#define WPPDF_Title    5
#define WPPDF_Subject  6
#define WPPDF_Keywords 7
#define WPPDF_Creator  8
#define WPPDF_IncludedFonts  9
#define WPPDF_ExcludedFonts  10
#define WPPDF_OwnerPassword  11
#define WPPDF_UserPassword   12
#define WPPDF_InputFile      13
```

6.2.27.2 SetIProp

This method changes an integer property. Using an **ID** the property is selected.

TYPELIB:
SetIProp(
[in] long id,
[in] long Value,
[out, retval] VARIANT_BOOL * Result);

This ids can be used:
#define WPPDF_ENCODE 1
#define WPPDF_COMPRESSION 2
#define WPPDF_PAGEMODE 3
#define WPPDF_USEFONTMODE 4
#define WPPDF_Encryption 5
#define WPPDF_InputFileMode 6
#define WPPDF_MonochromeThumbnail 7
#define WPPDF_JPEGCompress 8
#define WPPDF_EnhancedOptions 9

6.3 RTF2PDF

The dynamic link library PDFControl is a powerful tool to create PDF files by using standard windows API commands sent to a special device handle. It further more the ability to create thumbnails, create bookmarks and outlines and to export graphics.

The DLL RTF2PDF includes all the abilities of PDFControl but also includes a powerful RTF Engine. This makes it possible to load a RTF file, do mail merge and to export this RTF file to a PDF file.

The internal RTF engine is controlled by the function **ExecComand()** with the command numbers listed here.

The ActiveX wraps this function as methods as ExecCommand, ExecIntCommand and ExecStrCommand.
Normally you will use ExecStrCommand() since this method lets you specify a string as a parameter.

C example code to create a PDF engine, load a file and print it to PDF.

```
pdf=wpdfInitializeEx(&Info);                        // Initialize the PDF Engine
if(wpdfBeginDoc(pdf,"C:\\aTestPDF.pdf",0))            // Create PDF File
{
      wpdfExecCommand(pdf,1000,0,"",0);               // Initialize the RTF engine
      wpdfExecCommand(pdf,1002,0,"C:\\test.RTF",0);   // Load a RTF file
      wpdfExecCommand(pdf,1100,0,"",0);               // Export this RTF file to PDF
      wpdfEndDoc(pdf);
}
wpdfFinalize(pdf);
```

Example VB.NET Code to convert a RTF file to PDF:

```
       If PDF.StartEngine(DLLNAME.Text, "", "", 0) Then ' you your license info here !
            If PDF.BeginDoc(PDFNAME.Text, 0) > 0 Then ' Start a PDF document
                PDF.ExecIntCommand(1000, 0) ' Initialize the RTF engine
                 PDF.ExecIntCommand(1024, 1) ' use the printer as reference (suggested)
                PDF.ExecStrCommand(1002, RTFNAME.Text) ' Load a RTF file
                PDF.ExecIntCommand(1100, 0) ' Export this RTF file to PDF
                PDF.EndDoc() ' Close the PDF document
            End If
```

```
End If
```

Example VBS Code to convert RTF to PDF: uses the ActiveX)

```
Set PDF =  CreateObject("wPDF_X01.PDFControl")
PDF.INFO_Date = Now
PDF.INFO_Author = "Julian Ziersch"
PDF.INFO_Subject = "Test file"
' ... set other properties

PDF.StartEngine "c:\wPDFControl\DLL\wRTF2PDF01.dll", "LIC_NAME", "LIC_KEY" ,0 ' License
Info!
PDF.BeginDoc "Test.PDF", 0
PDF.ExecIntCommand 1000, 0
PDF.ExecIntCommand 1024, 1
PDF.ExecStrCommand 1002, "Demo.RTF"
PDF.ExecIntCommand 1100, 0
PDF.EndDoc
PDF.StopEngine
Set PDF = Nothing
```

6.3.1 WPCOM_RTFINIT = 1000

Initializes the RTF engine. Must be used right after **Initialize**().
Only if it was used the other commands may be used!

6.3.2 WPCOM_RTFVersion = 1001

The return value of the routine WPDF_ExecCommand is the version number of the RTF engine.

Tip: You can use WPCOM_RTFVersion to check if the engine understands RTF commands. I if it does not the return code is 0.

6.3.3 WPCOM_RTFLOAD = 1002

Loads the RTF file specified by the provided name.
All other parameters are ignored.

6.3.4 WPCOM_RTFAPPEND = 1003

Appends the RTF file specified by the provided name.
All other parameters are ignored.

6.3.5 WPCOM_RTFMAKEFIELDS= 1004

Converts <fieldnames> in the loaded RTF data into the special fields (insertpoints) which can be used for mailmerge. .

6.3.6 WPCOM_RTFMERGE = 1005

Starts the mailmerge process. Please note that you can execute this, write a PDF file and execute it again without having to reload the RTF data! The data is retrieved using the OnGetText callback/event procedure.

6.3.7 WPCOM_RTFSAVE = 1006

Saves the RTF file to the filename specified as parameter.

6.3.8 WPCOM_RTFBACKUP = 1007

WPCOM_RTFBACKUP writes the RTF data to an intern data storage.
To restore it use the command WPCOM_RTFRESTORE.

6.3.9 WPCOM_RTFRESTORE = 1008

WPCOM_RTFBACKUP writes the RTF data to an intern data storage.
To restore it use the command WPCOM_RTFRESTORE.

6.3.10 WPCOM_RTF_PAGEWIDTH = 1010

Set the page-width of the current RTF file to the width specified in the parameter value as twips (1/1440 inch).

6.3.11 WPCOM_RTF_PAGEHEIGHT = 1011

Set the page-height of the current RTF file to the height specified in the parameter value as twips (1/1440 inch).

6.3.12 WPCOM_RTF_MARGINLEFT = 1012

Set the left margin of the current RTF file to 'value'.

6.3.13 WPCOM_RTF_MARGINRIGHT = 1013

Set the right margin of the current RTF file to 'value'.

6.3.14 WPCOM_RTF_MARGINTOP = 1014

Set the top margin of the current RTF file to 'value'.

6.3.15 WPCOM_RTF_MARGINBOTTOM = 1015

Set the bottom margin of the current RTF file to 'value'.

6.3.16 WPCOM_RTF_PAGECOLUMNS = 1020

Define how many pages should be printed on one PDF page side by side. The standard is 1 to print one RTF page on one PDF page. 2 would print 2 RTF pages on one PDF pages and also double the width of the PDF page.

6.3.17 WPCOM_RTF_PAGEROTATION = 1021

Changes the rotation of the PDF page.
To print landscape use the value 90.

6.3.18 WPCOM_RTF_PAGEZOOM = 1022

Sets the zoom value for the RTF to PDF conversion. It can be used to shrink or enlarge the text.

To print 2 A4 pages on one A3 page use columns=2, Rotation=90, zoom=100. To print 2 A4 pages on one A4 page use zoom=50!

6.3.19 WPCOM_RTF_READEROPTIONS = 1023

This command changes a few options for the RTF reader:

Bit 0: all borders which use the width 0 should be invisible
Bit 1: the file is loaded in landscape mode
Bit 2: the page width and page height is swapped at load time

6.3.20 WPCOM_RTF_USE_PRINTER = 1024

Here you can switch on and off the use of the current printer driver for measuring the text.
Since the printer has a higher resolution this causes a better output (less rounding errors)
but it will require that a printer is installed.

Value=1 switches the mode on
Value=1 switches the mode off (default)

(This command has been added to V1.08)

6.3.21 WPCOM_RTFPRINT = 1100

Start the RTF to PDF conversion.
During the conversion process the callbacks are executed to let you fill in graphics if you need to.
This makes it possible to create a watermark on each page.

7 Properties

This is a list of the properties used by the PDF Engine.
If you don't use the ActiveX you can change this properties inside of the info structure used by the method InitializeEx(). The ActiveX publishes the properties.

PDFXRes : Integer;
Horizontal resolution used by the PDF engine. The default value is 72. Please note that the PDF resolution has no influence on the printed resolution of the bitmaps or the text. The quality will be not affected.

PDFYRes : Integer;
Vertical resolution used by the PDF engine, default = 72.

PDFWidth, PDFHeight: Integer;
This is the default page width measured using the above resolution properties. The default page

width will be used by the functions StartPage. If you use the function StartePageEx you can specify the size for each page.

Encoding : Integer;
Controls the encoding of the PDF file. By default the PDF uses binary encoding, if you need ASCII encoding you can either convert to ASCII85 using the value 1 or HEX using the value 2.

Compression : Integer;
Controls the compression of the text inside of the PDF file:
0=none,
1=deflate,
2=run length,
3=fast deflate.
We recommend to use the value 3. Please note that the library does not use LZW compression.

BitCompression : Integer;
This property controls how bitmaps are compressed. The default is to compress all RGB bitmaps using jpeg (provided JPEGQuality<>0) and all other bitmaps using deflate compression. Possible values are:
0=auto,
1=deflate,
2=jpeg.

JPEGCompress : Integer;
This property controls the JPEG quality:
0=off,
1=10%,
2=25%,
3=50%,
4=75%,
5=100%.

ThumbNails : Integer;
This property controls the creation of thumbnails. The following values are possible:
0=none,
1=color,
2=monochrome.

FontMode : Integer;
PDFControl supports font embedding. You can use this property to control this feature.
0=Use TrueType Fonts - they will not be embedded
1=Embed all TrueType Fonts
2=Embed symbol TrueType Fonts (suggested setting)
3=Use Base14 Type1 Fonts (TrueType fonts are not used)

PageMode : Integer;
This property controls how the PDF will be displayed when started:
0=Standard,
1=Show Outlines,
2=Show Thumbnails,
3=Show full screen.

Permission : Integer;
This property can be used to switch off copying or printing for the created PDF file. If you don't specify an owner password it will be auto-created once you set this property<>0.
Bit 0: enabled,
bit 1: Enable Printing,

bit 2: Enable Changing,
bit 3: Enable Copying,
bit 4: Enable Forms.

UserPassword, OwnerPassword : array[0..19] of Char / BSTR
This are the passwords for the created PDF file.

EnhancedOptions : Integer;
This property can be used to control certain modes of the PDF engine:
Bit 0: PDF text out ignores the width of the characters calculated by windows
Bit 1: PDF engine supports clipping (only rectangles)
Bit 3: switches OFF Auto Hyperlinks
Bit 4: <depreciated> (PDF does NOT use the a printer HDC as reference=default)
Bit 8: PDF uses the a printer HDC as reference.

InputFileMode : Integer;
PDFControl has an interesting feature to convert an existing PDF file into a watermark or append new pages to an existing file. Please note that it cannot really concatenate PDF files. The input PDF file has to be specified in property "InputPDFFile". The property InputFileMode controls how the file will be used.
0=Ignore "InputPDFFile"
1=Append to input PDF file
2=Convert input to watermarks (names "inpage1" ... "inpageN")
3=Use input as watermark

InputPDFFile: array[0..127] of Char / BSTR
This is the filename of the PDF file. See property InputFileMode.

MailMergeStart: array[0..7] of Char / BSTR
PDFControl can execute the mail-merge event for certain strings which are bout to be printed. This text are handled as mail merge fields if they start with the characters specified here. We suggest to leave this string empty or use for example "@@".

7.1 PDF Info

PDF files can contain certain information strings. This strings can be changed using the "INFO_..." properties of the ActiveX or the following character arrays in the info record:

Date,
ModDate,
Author,
Producer,
Title,
Subject,
Keywords : array[0..255] of Char;

As 'Creator' of the file always "wPDF by wpcubed GmbH" will be used. Please use the item Producer to give credit to your application.

8 Events/Callbacks

The PDF engine can execute a few callback procedures to inform your application about certain events during the PDF creation. It does not only create events to notice about problems but also to do mail merge. RTF2PDF will also create an event for each created PDF to let you paint to this page before the RTF text is printed.
The stream callbacks make it possible to create files using your code and do not create a PDF file as standard. This makes it possible to print to the <stdio>, for example if you want to use the PDF functionality in a CGI.

C/C++:
If you use C or C++ you can specify the callback procedures using the function pointers stored in the info record:

```
TWPDF_Callback* OnStreamOpen;
TWPDF_Callback* OnStreamWrite;
TWPDF_Callback* OnStreamClose;
TWPDF_Callback* OnError;
TWPDF_Callback* OnMessage;
TWPDF_Callback* OnGetText;
```

The prototype for a callback is always the same:

```
typedef void (__stdcall TWPDF_Callback)
             (WPDFEnviroment PdfEnv,
              int number, char *buffer, int bufsize);
```

If and how the parameters are used varies for the different events.

ActiveX:
The interface ActiveX publishes the callbacks as events. This makes the usage easy.

```
void OnError([In] Long Number, [In] BSTR Text );
void OnMessage([In] Long Number, [In] BSTR Text );
void OnStreamOpen([In] Long Number, [In] BSTR Text );
void OnStreamWrite( void );
void OnStreamClose( void );
void OnGetText([In] Long Number, [In] BSTR Text );
```

Parameters:
Number : This is the id of the message or event.
Buffer: This is a character pointer to the parameter of the event. Usually a message as a \0 terminated string but possibly also other, binary data.
BufSize: This is the length of the memory block specified by 'Buffer'. It is important if in the context of the callback binary data is expected.

The ActiveX provides a BSTR for the cases a string is expected. It converts "Buffer" for you.

8.1 Numbers for OnError

This numbers can be sent to the OnError callback / event:

```
#define WPERR_Bookmark  1 // Bookmark not found
#define WPERR_Bitmap    2 // Bitmap Errpr
#define WPERR_FileOpen  3 // Fileopen errr
#define WPERR_Meta      4 // Metafile error
#define WPERR_Font      5 // Font embedding error
#define WPERR_InputPDF  6 // Not able to load PDF file (InputFile)
       // (don't find it, wrong format)
#define WPERR_BeginDocRequired 10 // Not inside of BeginDoc/EndDoc
#define WPERR_StartPageRequired11 // Not inside of StartPage/EndPage or
       // StartWatermark/EndWatermark
#define WPERR_StreamMissing    12 // On OnStreamWrite method is missing
//  #define WPERR_ErrLoadLinkBitmap        13 // Used by 'AutoLinks':
       // the file was not found (param=name)
#define WPERR_ErrCannotDrawLinkBitmap  14 // cannot draw bitmap
       // (each page!). Buf = &WPDFAutoLink
```

8.2 Numbers for OnMessage

This numbers can be sent to the OnMessage callback / event:

```
#define WPMSG_BeforeBeginDoc  1
#define WPMSG_AfterEndDoc     2
#define WPMSG_EmbedFont       3
#define WPMSG_InputPDFFile    4
#define WPMSG_AfterBeginDoc   24
#define WPMSG_BeforeEndDoc    25
#define WPMSG_AfterStartPage  26
#define WPMSG_BeforeEndPage   27
```

The RTF-Engine of RTF2PDF also uses

```
#define WPMSG_RTFLOAD           1001
#define WPMSG_RTFPRINTPAGECOUNT 1002
#define WPMSG_RTFPRINTPAGE      1003
```

8.3 How to use streams in ANSI-C

If you use ANSI-C and started the PDF file with BeginDoc with the last parameter set to 1 the PDF output can be sent to the standard output (stdout):

```
// Open an output stream. Not required if stdout is used
void _stdcall wpdfcall_StreamOpen(WPDFEnviroment PdfEnv, int number, char *buffer, int
bufsize)
{
}

// Create the PDF file as stdout
void _stdcall wpdfcall_StreamWrite(WPDFEnviroment PdfEnv, int number, char *buffer, int
bufsize)
{
  fwrite(buffer, bufsize, 1, stdout);
}

// Close the output stream. It is not required for stdout
void _stdcall wpdfcall_StreamClose(WPDFEnviroment PdfEnv, int number, char *buffer, int
bufsize)
{
}
```

To use the 3 callbacks set the pointer in the info record:

```
Info.OnStreamOpen  = wpdfcall_StreamOpen;
Info.OnStreamWrite = wpdfcall_StreamWrite;
Info.OnStreamClose = wpdfcall_StreamClose;
```

You need to start the PDF document using

```
wpdfBeginDoc(pdf, "", 1 )
```

to use the stream callbacks

8.4 How to use mail-merge

The OnGetText event is created for the mail merge fields. (See property MailMergeStart).

The method SetResult is used to set the field text inside of the PDF engine.

SetResult expects this parameters:

```
int env;
int buffertype;
char *buffer;
int bufsize
```

If buffertype=1 then buffer is expected to point to a string.
If bufsize<0 then the string is expected to be \0 terminated, otherwise bufsize is expected to be the length of the string.
Other buftypes are currently not supported.

```
void _stdcall wpdfcall_GetText(WPDFEnviroment PdfEnv, int number, char *buffer, int
bufsize)
{
  //Debug: show message:
  // fprintf(stderr,"Field: %s\n",buffer);
  // set the field text:
  wpdfSetResult(PdfEnv,1,"*** Test ***",-1);
}
```

9 Declarations

In this chapter we list the declaration used by wPDFControl and RTF2PDF methods.

9.1 PDFControl

9.1.1 Ansi C

```
/* wPDF.H
   This file includes
   -   struct WPDFInfoRecord - the PDF Engine initialisation parameter
   -   typedefs for callbacks and engine functions
   -   macro DEF_WPDF_ENGINE_PTR to define the pointers
   -   macro LOAD_WPDF_ENGINE_PTR( DLLVAR, DLLNAME )
        to initialize the pointers
   -   ERROR, MESSAGE and COMMAND constants
*/

#ifndef WPCONTROLDEF
// Tag to identify the PDF enviroment
typedef int WPDFEnviroment;
typedef char* pchar;

// Function definition for callbacks
typedef void (__stdcall TWPDF_Callback)
                  (WPDFEnviroment PdfEnv,
                   int number, char *buffer, int bufsize);

// Structur to set up the PDF Engine with WPDF_InitializeEx
typedef struct
  {
     int SizeOfStruct;
     // Encoding
     int Encoding; // 0 = none, 1=ASCII85, 2=Hex
     // Compression 0=none, deflate, runlength
     int Compression;
     // BitmapCompression
     int BitCompression; // 0=auto, 1=deflat, 2=jpeg
     // Thumbnailes
     int ThumbNails; // 0=none, 1=color, 2=monchrome
     // UseFonts
     int FontMode;
     // PageMode
     int PageMode;
     // InputFileMode
     int InputFileMode;
     // JPEGCompress 0=off, - = value, 1..5 for 10, 25, 50, 75, 100
     int JPEGCompress;
     int EnhancedOptions;
     // PDF Resolution
     int PDFXRes, PDFYRes; // 0->72
     // Default PDF Size (for StartPageA)
     int PDFWidth, PDFHeight; // Default DINa4
     // Read PDF Filename
     char InputPDFFile[128];
     // Several Callback functions
     // For Stream Output. We can use this callbacks to write to STDIO.
     // this is useful for CGI applications which rely to send to the PDF output
     // to stdout
     TWPDF_Callback* OnStreamOpen;
     TWPDF_Callback* OnStreamWrite;
     TWPDF_Callback* OnStreamClose;
     // Message callbacks
     TWPDF_Callback* OnError;
     TWPDF_Callback* OnMessage;
     // The mailmerge string is used to detect mailmerge fields
     // it is possible to the text such as @@NAME be using the callback function OnGetText
     char MailMergeStart[8];
     TWPDF_Callback* OnGetText;
     // Encryption of the PDF file
     int Permission;
     char UserPassword[20], OwnerPassword[20]; // actually only 5 chars long (40 bits)
     // Info record of the PDF file
```

```
char Date[256],
          ModDate[256],
          Author[256],
          Producer[256],
          Title[256],
          Subject[256],
          Keywords[256];
     // Reserved, must be 0
     int Reserved;
} WPDFInfoRecord;

// New in V1.01:  uses this with
//     WPDF_ExecCommand( pdf, WPCOM_AUTOLINK, 0, &WPDFAutoLink, sizeof(WPDFAutoLink));
// Structur to set up a link bitmap which printed on every page (for Demo versions)
typedef struct
  {
     int BitmapCloneNr; // has priority  (or 0)
     int BitmapHandle;  // HBITMAP handle (or 0)
     char *VarBuf;      // extra pointer parameter
     int VarBufLen;     // Len of buffer
     // Options:
     //    1: Under Text (but over watermark!)
     //    2: Interpret VarP as pointer to a filename
     int Options;
     // Destination Rectangle (in PDF resolution, not "HDC" Resolution!)
     int DrawX;
     int DrawY;
     int DrawW;
     int DrawH;
     // Destination URL and relative link rectangle
     char LinkURL[100]; // Bookmark or "http://" URL
     int  LinkX; // added to DrawX
     int  LinkY; // added to DrawY
     int  LinkW; // if 0 then = DrawW
     int  LinkH; // if 0 then = DrawH
} WPDFAutoLink;

// Methods defined in the DLL
typedef WPDFEnviroment (__stdcall WPDF_InitializeEx)(WPDFInfoRecord *Info); //
WPDFInfoRecord
typedef WPDFEnviroment (__stdcall WPDF_Initialize)(void);
typedef void (__stdcall WPDF_Finalize)(WPDFEnviroment PdfEnv);
typedef void (__stdcall WPDF_FinalizeAll)(void);
typedef void (__stdcall WPDF_SetResult)(WPDFEnviroment PdfEnv, int buffertype, char
*buffer, int bufsize);

typedef int (__stdcall WPDF_BeginDoc)(WPDFEnviroment PdfEnv, char *FileName, int
UseStream);
typedef void (__stdcall WPDF_EndDoc)(WPDFEnviroment PdfEnv);
typedef void (__stdcall WPDF_StartPage)(WPDFEnviroment PdfEnv);
typedef void (__stdcall WPDF_StartPageEx)(WPDFEnviroment PdfEnv, int Width, int Height,
int Rotation);
typedef void (__stdcall WPDF_EndPage)(WPDFEnviroment PdfEnv);
typedef void (__stdcall WPDF_StartWatermark)(WPDFEnviroment PdfEnv, char *Name);
typedef void (__stdcall WPDF_StartWatermarkEx)(WPDFEnviroment PdfEnv, char *Name, int
Width, int Height);
typedef void (__stdcall WPDF_EndWatermark)(WPDFEnviroment PdfEnv);
typedef void (__stdcall WPDF_DrawWatermark)(WPDFEnviroment PdfEnv, char *Name, int
Rotation);

// Property and command
typedef void (__stdcall WPDF_SetSProp)(WPDFEnviroment PdfEnv, int id, char *Value);
typedef void (__stdcall WPDF_SetIProp)(WPDFEnviroment PdfEnv, int id, int Value);
typedef int (__stdcall WPDF_ExecCommand)(WPDFEnviroment PdfEnv, int id, int Value,char
*buffer,int buflen);

// PDF Output Functions
typedef void (__stdcall WPDF_DrawMetafile)(WPDFEnviroment PdfEnv, unsigned int meta, int
x, int y, int w, int h);
typedef int (__stdcall WPDF_DrawDIB)(WPDFEnviroment PdfEnv, int x, int y, int w, int h,
void *BitmapInfo, void *BitmapBits);
```

```
typedef int (__stdcall WPDF_DrawBMP)(WPDFEnviroment PdfEnv, int x, int y, int w, int h,
HBITMAP Bitmap);
typedef int (__stdcall WPDF_DrawJPEG)(WPDFEnviroment PdfEnv, int x, int y, int w, int h,
int jpeg_w, int jpeg_h, void *buffer, int buflen);
typedef int (__stdcall WPDF_DrawBitmapFile)(WPDFEnviroment PdfEnv, int x, int y, int w,
int h, char *FileName);
typedef int (__stdcall WPDF_DrawBitmapClone)(WPDFEnviroment PdfEnv, int x, int y, int w,
int h, int BitmapID);

// HDC Output function
typedef HDC (__stdcall WPDF_DC)(WPDFEnviroment PdfEnv); //** Get the DC of the PDF Canvas!
**//
typedef void (__stdcall WPDF_TextOut)(WPDFEnviroment PdfEnv, int x, int y, char * Text);
typedef char* (__stdcall WPDF_TextRect)(WPDFEnviroment PdfEnv, int x, int y, int w, int h,
char *Text, int Alignment);

typedef void (__stdcall WPDF_MoveTo)(WPDFEnviroment PdfEnv, int x, int y);
typedef void (__stdcall WPDF_LineTo)(WPDFEnviroment PdfEnv, int x, int y);
typedef void (__stdcall WPDF_Rectangle)(WPDFEnviroment PdfEnv, int x, int y,int w,int h);

typedef void (__stdcall WPDF_Hyperlink)(WPDFEnviroment PdfEnv, int x, int y,int w,int
h,char *Name);
typedef void (__stdcall WPDF_Bookmark)(WPDFEnviroment PdfEnv, int x, int y,char *Name);
typedef void (__stdcall WPDF_Outline)(WPDFEnviroment PdfEnv, int level,  int x, int y,char
*Name, char *Caption);

// Attribute funtions
typedef void (__stdcall WPDF_SetTextDefaultAttr)(WPDFEnviroment PdfEnv, char *FontName,
int Size);
typedef void (__stdcall WPDF_SetTextAttr)(WPDFEnviroment PdfEnv, char *FontName, int Size,
int Bold, int Italic,int Underline);
typedef void (__stdcall WPDF_SetTextAttrEx)(WPDFEnviroment PdfEnv, char *FontName, int
Charset,
         int Size, int Bold, int Italic,int Underline, unsigned int Color);
typedef void (__stdcall WPDF_SetPenAttr)(WPDFEnviroment PdfEnv, int Style, int Width, int
Color);
typedef void (__stdcall WPDF_SetBrushAttr)(WPDFEnviroment PdfEnv, int Style, int Color);

// Set License Key. This function only exists in the registered DLL, notz in the demo!
typedef void (__stdcall WPDF_SetLicenseKey)(unsigned long number, char *Name, char *Code);

// ---------------------------------------------------------------------------
// Macro to define the function pointers used for the PDF engine DLL
// Usage:    DEF_WPDF_ENGINE_PTR;
// ---------------------------------------------------------------------------

#define DEF_WPDF_ENGINE_PTR                  \
WPDF_InitializeEx* wpdfInitializeEx;         \
WPDF_Initialize*  wpdfInitialize;            \
WPDF_Finalize*    wpdfFinalize;              \
WPDF_FinalizeAll* wpdfFinalizeAll;           \
WPDF_SetResult*    wpdfSetResult;            \
WPDF_BeginDoc*    wpdfBeginDoc;              \
WPDF_EndDoc*      wpdfEndDoc;                \
WPDF_StartPage*   wpdfStartPage;             \
WPDF_StartPageEx* wpdfStartPageEx;           \
WPDF_EndPage*     wpdfEndPage;               \
WPDF_StartWatermark* wpdfStartWatermark;     \
WPDF_StartWatermarkEx* wpdfStartWatermarkEx; \
WPDF_EndWatermark* wpdfEndWatermark;         \
WPDF_DrawWatermark* wpdfDrawWatermark;       \
WPDF_SetSProp*    wpdfSetSProp;              \
WPDF_SetIProp*    wpdfSetIProp;              \
WPDF_ExecCommand* wpdfExecCommand;           \
WPDF_DrawMetafile* wpdfDrawMetafile;         \
WPDF_DrawDIB* wpdfDrawDIB;                   \
WPDF_DrawBMP* wpdfDrawBMP;                   \
WPDF_DrawJPEG* wpdfDrawJPEG;                 \
WPDF_DrawBitmapFile* wpdfDrawBitmapFile;     \
WPDF_DrawBitmapClone* wpdfDrawBitmapClone;   \
WPDF_DC*           wpdfDC;                   \
WPDF_TextOut*      wpdfTextOut;              \
```

```
WPDF_TextRect*     wpdfTextRect;            \
WPDF_MoveTo*       wpdfMoveTo;              \
WPDF_LineTo*       wpdfLineTo;              \
WPDF_Rectangle*    wpdfRectangle;           \
WPDF_Hyperlink*    wpdfHyperlink;           \
WPDF_Bookmark*     wpdfBookmark;            \
WPDF_Outline*      wpdfOutline;             \
WPDF_SetTextDefaultAttr* wpdfSetTextDefaultAttr; \
WPDF_SetTextAttr*   wpdfSetTextAttr;        \
WPDF_SetTextAttrEx* wpdfSetTextAttrEx;      \
WPDF_SetPenAttr*    wpdfSetPenAttr;         \
WPDF_SetBrushAttr*  wpdfSetBrushAttr;       \
WPDF_SetLicenseKey* wpdfSetLicenseKey       // no ';'

// ---------------------------------------------------------------------------
// Macro to load the DLL and initialize the function pointers defined above
// Usage:    LOAD_WPDF_ENGINE( PDFEngine,       - variable used as DLL handle
//                                dllname          - PDF engine name
//                              ) == 0 if loaded OK
//                                == 1 if not found
//                                == 2 if wrong version (function is missing)
// ---------------------------------------------------------------------------

#define LOAD_WPDF_ENGINE( PDFEngine, dllname ) \
(                                              \
 PdfEngine = LoadLibrary(dllname),             \
 ((PdfEngine==0)?1:                            \
  ( wpdfInitializeEx=(WPDF_InitializeEx*)GetProcAddress(PdfEngine, "WPDF_InitializeEx"),
\
    wpdfInitialize=(WPDF_Initialize*)GetProcAddress(PdfEngine, "WPDF_Initialize"),   \
    wpdfFinalize=(WPDF_Finalize*)GetProcAddress(PdfEngine, "WPDF_Finalize"),   \
    wpdfFinalizeAll=(WPDF_FinalizeAll*)GetProcAddress(PdfEngine, "WPDF_FinalizeAll"),   \
    wpdfSetResult= (WPDF_SetResult*)GetProcAddress(PdfEngine, "WPDF_SetResult"),   \
    wpdfBeginDoc=(WPDF_BeginDoc*)GetProcAddress(PdfEngine, "WPDF_BeginDoc"),   \
    wpdfEndDoc=(WPDF_EndDoc*)GetProcAddress(PdfEngine, "WPDF_EndDoc"),   \
    wpdfStartPage=(WPDF_StartPage*)GetProcAddress(PdfEngine, "WPDF_StartPage"),   \
    wpdfStartPageEx=(WPDF_StartPageEx*)GetProcAddress(PdfEngine, "WPDF_StartPageEx"),   \
    wpdfEndPage=(WPDF_EndPage*)GetProcAddress(PdfEngine, "WPDF_EndPage"),   \
    wpdfStartWatermark=(WPDF_StartWatermark*)GetProcAddress(PdfEngine,
"WPDF_StartWatermark"),   \
    wpdfStartWatermarkEx=(WPDF_StartWatermarkEx*)GetProcAddress(PdfEngine,
"WPDF_StartWatermarkEx"),   \
    wpdfEndWatermark=(WPDF_EndWatermark*)GetProcAddress(PdfEngine, "WPDF_EndWatermark"),
\
    wpdfDrawWatermark=(WPDF_DrawWatermark*)GetProcAddress(PdfEngine,
"WPDF_DrawWatermark"),   \
    wpdfSetSProp=(WPDF_SetSProp*)GetProcAddress(PdfEngine, "WPDF_SetSProp"),   \
    wpdfSetIProp=(WPDF_SetIProp*)GetProcAddress(PdfEngine, "WPDF_SetIProp"),   \
    wpdfExecCommand=(WPDF_ExecCommand*)GetProcAddress(PdfEngine, "WPDF_ExecCommand"),   \
    wpdfDrawMetafile=(WPDF_DrawMetafile*)GetProcAddress(PdfEngine, "WPDF_DrawMetafile"),
\
    wpdfDrawDIB=(WPDF_DrawDIB*)GetProcAddress(PdfEngine, "WPDF_DrawDIB"),   \
    wpdfDrawBMP=(WPDF_DrawBMP*)GetProcAddress(PdfEngine, "WPDF_DrawBMP"),   \
    wpdfDrawJPEG=(WPDF_DrawJPEG*)GetProcAddress(PdfEngine, "WPDF_DrawJPEG"),   \
    wpdfDrawBitmapFile=(WPDF_DrawBitmapFile*)GetProcAddress(PdfEngine,
"WPDF_DrawBitmapFile"),   \
    wpdfDrawBitmapClone=(WPDF_DrawBitmapClone*)GetProcAddress(PdfEngine,
"WPDF_DrawBitmapClone"),   \
    wpdfDC=(WPDF_DC*)GetProcAddress(PdfEngine, "WPDF_DC"),   \
    wpdfTextOut=(WPDF_TextOut*)GetProcAddress(PdfEngine, "WPDF_TextOut"),   \
    wpdfTextRect=(WPDF_TextRect*)GetProcAddress(PdfEngine, "WPDF_TextRect"),   \
    wpdfMoveTo=(WPDF_MoveTo*)GetProcAddress(PdfEngine, "WPDF_MoveTo"),   \
    wpdfLineTo=(WPDF_LineTo*)GetProcAddress(PdfEngine, "WPDF_LineTo"),   \
    wpdfRectangle=(WPDF_Rectangle*)GetProcAddress(PdfEngine, "WPDF_Rectangle"),   \
    wpdfHyperlink=(WPDF_Hyperlink*)GetProcAddress(PdfEngine, "WPDF_Hyperlink"),   \
    wpdfBookmark=(WPDF_Bookmark*)GetProcAddress(PdfEngine, "WPDF_Bookmark"),   \
    wpdfOutline=(WPDF_Outline*)GetProcAddress(PdfEngine, "WPDF_Outline"),   \
    wpdfSetTextDefaultAttr=(WPDF_SetTextDefaultAttr*)GetProcAddress(PdfEngine,
"WPDF_SetTextDefaultAttr"),   \
    wpdfSetTextAttr=(WPDF_SetTextAttr*)GetProcAddress(PdfEngine, "WPDF_SetTextAttr"),   \
    wpdfSetTextAttrEx=(WPDF_SetTextAttrEx*)GetProcAddress(PdfEngine,
"WPDF_SetTextAttrEx"),   \
    wpdfSetPenAttr=(WPDF_SetPenAttr*)GetProcAddress(PdfEngine, "WPDF_SetPenAttr"),   \
    wpdfSetBrushAttr=(WPDF_SetBrushAttr*)GetProcAddress(PdfEngine, "WPDF_SetBrushAttr"),
```

```
\
    wpdfSetLicenseKey=(WPDF_SetLicenseKey*)GetProcAddress(PdfEngine,
"WPDF_SetLicenseKey"),   \
    ( (                                                  \
         (wpdfInitialize==NULL)||(wpdfInitializeEx==NULL)||      \
         (wpdfFinalize==NULL)||(wpdfFinalizeAll==NULL)||         \
         (wpdfSetResult==NULL)||                                 \
         (wpdfBeginDoc==NULL)||(wpdfEndDoc==NULL)||              \
         (wpdfStartPage==NULL)||(wpdfStartPageEx==NULL)|| (wpdfEndPage==NULL)|| \
         (wpdfStartWatermark==NULL)||(wpdfStartWatermarkEx==NULL)||     \
         (wpdfDrawWatermark==NULL)||(wpdfEndWatermark==NULL)||          \
         (wpdfSetSProp==NULL)||(wpdfSetIProp==NULL)||(wpdfExecCommand==NULL)|| \
         (wpdfDrawMetafile==NULL)||                                     \
         (wpdfDrawDIB==NULL)||(wpdfDrawBMP==NULL)||(wpdfDrawJPEG==NULL)||   \
         (wpdfDrawBitmapFile==NULL)||(wpdfDrawBitmapClone==NULL)||          \
         (wpdfDC==NULL)||                                                   \
         (wpdfTextOut==NULL)||(wpdfTextRect==NULL)||(wpdfMoveTo==NULL)||    \
         (wpdfLineTo==NULL)||(wpdfRectangle==NULL)||                        \
         (wpdfRectangle==NULL)||(wpdfMoveTo==NULL)||                        \
         (wpdfHyperlink==NULL)||(wpdfBookmark==NULL)||(wpdfOutline==NULL)|| \
         (wpdfSetTextDefaultAttr==NULL)||(wpdfSetTextAttr==NULL)||          \
         (wpdfSetTextAttrEx==NULL)||(wpdfSetPenAttr==NULL)||                \
         (wpdfSetBrushAttr==NULL)                                           \
      )?                                                                    \
      ( FreeLibrary(PdfEngine),   \
        wpdfInitialize=NULL,      \
        wpdfFinalize=NULL,        \
        wpdfFinalizeAll=NULL,     \
        2 ) : 0                   \
      )                           \
    )                             \
  ) \
) \
  // note: wpdfSetLicenseKey *may* be NULL in Demo version

// ---------------------------------------------------------------------------


// This constants are used to modify certain properties of the PDF engine

  #define WPPDF_Author 1
  #define WPPDF_Date     2
  #define WPPDF_ModDate  3
  #define WPPDF_Producer 4
  #define WPPDF_Title    5
  #define WPPDF_Subject  6
  #define WPPDF_Keywords 7
  #define WPPDF_Creator  8
  #define WPPDF_IncludedFonts  9
  #define WPPDF_ExcludedFonts  10
  #define WPPDF_OwnerPassword  11
  #define WPPDF_UserPassword   12
  #define WPPDF_InputFile      13

  #define WPPDF_ENCODE  1
  #define WPPDF_COMPRESSION  2
  #define WPPDF_PAGEMODE  3
  #define WPPDF_USEFONTMODE  4
  #define WPPDF_Encryption  5
  #define WPPDF_InputFileMode   6
  #define WPPDF_MonochromeThumbnail  7
  #define WPPDF_JPEGCompress  8
  #define WPPDF_EnhancedOptions  9

// Message IDs for OnError
 #define WPERR_Bookmark  1 // Bookmark not found
 #define WPERR_Bitmap    2   // Bitmap Errpr
 #define WPERR_FileOpen  3 // Fileopen errr
 #define WPERR_Meta      4 // Metafile error
 #define WPERR_Font      5 // Font embedding error
 #define WPERR_InputPDF  6 // Not able to load PDF file (don't find it, wrong format)
 #define WPERR_BeginDocRequired 10 // Not inside of BeginDoc/EndDoc
 #define WPERR_StartPageRequired11 // Not inside of StartPage/EndPage or
StartWatermark/EndWatermark
```

```
#define WPERR_StreamMissing    12 // On OnStreamWrite method was specified
 // V1.01: For AutoLinks
 #define WPERR_ErrLoadLinkBitmap        13 // the file was not found (param=name)
 #define WPERR_ErrCannotDrawLinkBitmap  14 // cannot draw bitmap (each page!). Buf =
&WPDFAutoLink

// This codes are used in the Message callback

 #define WPMSG_BeforeBeginDoc  1
 #define WPMSG_AfterEndDoc     2
 #define WPMSG_EmbedFont       3
 #define WPMSG_InputPDFFile    4
 #define WPMSG_AfterBeginDoc   24
 #define WPMSG_BeforeEndDoc    25
 #define WPMSG_AfterStartPage  26
 #define WPMSG_BeforeEndPage   27

// Codes for the wpdfCommand API

 #define WPCOM_xxxx 0
 #define WPCOM_Version  1  // Result = version * 100
 #define WPCOM_AUTOLINK 2  // new in V1.01
```

9.1.2 C++

This is the intervace class defined in file wpdf_class.cpp

```
class CPDFExport
 {
  private:
  // ...
  public:
    CPDFExport(char *PDFENGINE, unsigned long code, char *Name, char *Key);
    ~CPDFExport();
   // public API functions
   WPDFEnviroment PDFEnv();
   int EngineWasLoaded();
   int  InitializeEx(WPDFInfoRecord Info);
   int  Initialize(void);
   void Finalize();
   void FinalizeAll(void);
   void SetResult(int buffertype, char *buffer, int bufsize);

   int  BeginDoc(char *FileName, int UseStream);
   void EndDoc();
   void StartPage();
   void StartPageEx(int Width, int Height, int Rotation);
   void EndPage();
   void StartWatermark(char *Name);
   void StartWatermarkEx(char *Name, int Width, int Height);
   void EndWatermark();
   void DrawWatermark(char *Name, int Rotation);

   // Property and command
   void SetSProp(int id, char *Value);
   void SetIProp(int id, int Value);
   int  ExecCommand(int id, int Value,char *buffer,int buflen);

   // PDF Output Functions
   void DrawMetafile(unsigned int meta, int x, int y, int w, int h);
   int  DrawDIB(int x, int y, int w, int h,
              void *BitmapInfo, void *BitmapBits);
   int  DrawBMP(int x, int y, int w, int h, HBITMAP Bitmap);
   int  DrawJPEG(int x, int y, int w, int h,
              int jpeg_w, int jpeg_h, void *buffer, int buflen);
   int  DrawBitmapFile(int x, int y, int w, int h, char *FileName);
   int  DrawBitmapClone(int x, int y, int w, int h, int BitmapID);

   // HDC Output function
   HDC DC(); //** Get the DC of the PDF Canvas! **//

   void TextOut(int x, int y, char * Text);
   char* TextRect(int x, int y,
              int w, int h, char *Text, int Alignment);
   void MoveTo(int x, int y);
   void LineTo(int x, int y);
   void Rectangle(int x, int y,int w,int h);

   void Hyperlink(int x, int y,int w,int h,char *Name);
   void Bookmark(int x, int y,char *Name);
   void Outline(int level,  int x, int y,
              char *Name, char *Caption);

   // Attribute funtions
   void SetTextDefaultAttr(char *FontName, int Size);
   void SetTextAttr(char *FontName, int Size, int Bold,
                    int Italic,int Underline);
   void SetTextAttrEx(char *FontName, int Charset, int Size,
                    int Bold, int Italic,int Underline,
                    unsigned int Color);
   void SetPenAttr(int Style, int Width, int Color);
   void SetBrushAttr(int Style, int Color);
 };
```

9.1.3 Pascal

```
function WPDF_InitializeEx(pInfoRec : pWPDF_InfoRecord) : pdfEnviroment; stdcall;
function WPDF_Initialize : pdfEnviroment; stdcall;
procedure WPDF_Finalize(env: pdfEnviroment); stdcall;
procedure WPDF_FinalizeAll; stdcall;
procedure WPDF_SetResult(env: pdfEnviroment; buffertype : Integer; buffer : PChar; bufsize
: Integer ); stdcall;
function  WPDF_BeginDoc(env: pdfEnviroment; FileName : PChar; UseStream :
Integer):Integer; stdcall;
procedure WPDF_EndDoc(env: pdfEnviroment); stdcall;
procedure WPDF_StartPage(env: pdfEnviroment); stdcall;
procedure WPDF_StartPageEx(env: pdfEnviroment; Width,Height, Rotation : Integer); stdcall;
procedure WPDF_EndPage(env: pdfEnviroment); stdcall;
procedure WPDF_StartWatermark(env: pdfEnviroment; Name : PChar); stdcall;
procedure WPDF_StartWatermarkEx(env: pdfEnviroment; Name : PChar ; Width,Height :Integer);
stdcall;
procedure WPDF_EndWatermark(env: pdfEnviroment); stdcall;
procedure WPDF_DrawWatermark(env: pdfEnviroment; Name : PChar; Rotation : Integer);
stdcall;
function WPDF_SetSProp(env: pdfEnviroment; id: Integer; value: pchar): Boolean; stdcall;
function WPDF_SetIProp(env: pdfEnviroment; id: Integer; value: Integer): Boolean; stdcall;
function WPDF_ExecCommand(env: pdfEnviroment; id: Integer; value: Integer; buf : PChar;
buflen : Integer): Integer; stdcall;
procedure WPDF_DrawMetafile(env: pdfEnviroment; meta : Cardinal; x,y,w,h : Integer);
stdcall;
function  WPDF_DrawDIB(env: pdfEnviroment; x, y, w, h: Integer; BitmapInfo: PBitmapInfo;
BitmapBits: Pointer):Integer; stdcall;
function  WPDF_DrawBMP(env: pdfEnviroment; x, y, w, h: Integer; Bitmap : HBITMAP):Integer;
stdcall;
function  WPDF_DrawJPEG(env: pdfEnviroment; x, y, w, h, jpeg_w, jpeg_h: Integer; buf :
PChar; buflen : Integer):Integer; stdcall;
function  WPDF_DrawBitmapFile(env: pdfEnviroment; x, y, w, h: Integer; filename :
PChar):Integer; stdcall;
procedure WPDF_DrawBitmapClone(env: pdfEnviroment; x, y, w, h: Integer; BitmapID :
Integer); stdcall;
function  WPDF_DC(env: pdfEnviroment): HDC;stdcall;// Result is a HDC !
function  WPDF_GetPointer(env: pdfEnviroment): Pointer;stdcall;
procedure WPDF_SetPointer(env: pdfEnviroment; PTR : Pointer);stdcall;
procedure WPDF_TextOut(env: pdfEnviroment; x,y : Integer; Text : PChar); stdcall;
function  WPDF_TextRect(env: pdfEnviroment; x,y,w,h : Integer; Text : PChar; Alignment :
Integer):PChar; stdcall;
procedure WPDF_MoveTo(env: pdfEnviroment; x,y : Integer); stdcall;
procedure WPDF_LineTo(env: pdfEnviroment; x,y : Integer); stdcall;
procedure WPDF_Rectangle(env: pdfEnviroment; x,y,w,h : Integer); stdcall;
procedure WPDF_Hyperlink(env: pdfEnviroment; x,y,w,h : Integer; Name : PChar); stdcall;
procedure WPDF_Bookmark(env: pdfEnviroment;  x,y : Integer; Name : PChar); stdcall;
procedure WPDF_Outline(env: pdfEnviroment; level,x,y : Integer; Name, Caption : PChar);
stdcall;
procedure WPDF_SetTextDefaultAttr(env: pdfEnviroment; Font : PChar; Size : Integer);
stdcall;
procedure WPDF_SetTextAttr(env: pdfEnviroment;
    Font : PChar; Size : Integer; Bold : Integer; Italic : Integer; Underline : Integer);
stdcall;
procedure WPDF_SetTextAttrEx(env: pdfEnviroment;
    Font : PChar; Size : Integer; Charset : Integer;  Bold : Integer;
    Italic : Integer; Underline : Integer; Color : Cardinal); stdcall;
procedure WPDF_SetPenAttr(env: pdfEnviroment; Style, Width : Integer; Color : Cardinal);
stdcall;
procedure WPDF_SetBrushAttr(env: pdfEnviroment; Style: Integer; Color : Cardinal);
stdcall;
procedure WPDF_SetLicenseKey(Number : Cardinal; Name : PChar; Code : PChar); stdcall;
```

This is the record which can be used to initialize the engine. The ActiveX publishes most items as properties.

```
TWPDF_InfoRecord = packed record
     SizeOfStruct : Integer; // the size of this structure = SizeOf(TWPDF_InfoRecord)
```

```
Encoding : Integer; // Encoding: 0 = none, 1=ASCII85, 2=Hex
     Compression : Integer;// Compression: 0=none, 1=deflate, 2=runlength, 3=fast deflate
     BitCompression : Integer; // Compress bitmaps: 0=auto, 1=deflate, 2=jpeg
     ThumbNails : Integer; // Thumbnails: 0=none, 1=color, 2=monchrome
     FontMode : Integer; // Fonts mode:  0= Use TrueType Fonts, 1=Embed TrueType Fonts,
     // 2=Embed Symbol TrueType Fonts, 3=Use Base14 Type1 Fonts
     PageMode : Integer; // Start PDF file with: 0=Standard, 1=Outlines, 2=Thumbnails,
3=As full screen
     InputFileMode : Integer; // 0=Ignore "InputPDFFile"
     // 1=Append To Input PDF file
     // 2=Convert Input To Watermarks (names inpage1 --- inpageN)
     // 3=Use Input As Watermark
     JPEGCompress  : Integer; // JPEG quality: 0=off, 1=10%, 2=25%, 3=50%, 4=75%, 5=100%
     EnhancedOptions : Integer;
     // Bit 0: PDF text out ignores the width of the characters calculated by windows
     // Bit 1: PDF engine supports clipping (only rectangles)
     // Bit 3: switches OFF Auto Hyperlinks
     // Bit 4: PDF does  NOT use use the a printer HDC as reference
     PDFXRes, PDFYRes : Integer; // PDF resolution (default 72)
     PDFWidth, PDFHeight: Integer; // PDF default page size measured in PDFXRes
     InputPDFFile: array[0..127] of Char; // Input PDF file
     // Several Callback functions
     OnStreamOpen  : TWPDF_Callback;  // open a stream
     OnStreamWrite : TWPDF_Callback;  // write bufsiz bytes from buffer
     OnStreamClose : TWPDF_Callback;  // close stream
     // Messages
     OnError       : TWPDF_Callback;  // an error happens
     OnMessage     : TWPDF_Callback;  // message from PDF engine
     // Mailmerge
     MailMergeStart: array[0..7] of Char; // start identifier
               // for a mailmerge field, for example "@@"
     OnGetText     : TWPDF_Callback;  // found field "buffer"
               // use WPDF_SetResultValue to set contents
     // Encryption
     Permission : Integer; // Bit 0: enabled, bit 1: Enable Printing,
     // bit 2: Enable Changing, bit 3: Enable Copying, bit 4: Enable Forms
     UserPassword, OwnerPassword : array[0..19] of Char; // passwords
     // Info  strings
     Date,
     ModDate,
     Author,
     Producer,
     Title,
     Subject,
     Keywords : array[0..255] of Char;
     // Reserved, must be 0
     Reserved      : Integer;
  end;
```

9.1.4 ActiveX

```
IPDFControl = interface(IDispatch)
    ['{C7B14EA2-E0EC-11D5-8CAE-00E07D9789EA}']
    function  StartEngine(const DLLName: WideString;
                      const LicenseName: WideString;
                      const LicenseCode: WideString;
                      LicenseNumber: Integer): WordBool;
    procedure StopEngine;
    function  BeginDoc(const FileName: WideString;
                      UseStream: Integer): Integer;
    procedure EndDoc;
    procedure StartPage;
    procedure StartPageEx(Width: Integer; Height: Integer;
                      Rotation: Integer);
    procedure EndPage;
    procedure StartWatermark(const Name: WideString);
    procedure StartWatermarkEx(const Name: WideString;
                      Width: Integer; Height: Integer);
    procedure EndWatermark;
    procedure DrawWatermark(const Name: WideString; Rotation: Integer);
    function  SetSProp(id: Integer; const Value: WideString): WordBool;
    function  SetIProp(id: Integer; Value: Integer): WordBool;
    function  ExecCommand(id: Integer; param2: Integer;
                      buf: Integer; buflen: Integer): Integer;
    procedure DrawMetafile(meta: Integer; x: Integer;
                      y: Integer; w: Integer; h: Integer);
    function  DrawDIB(x: Integer; y: Integer;
                      w: Integer; h: Integer; BitmapInfo: Integer;
                      BitmapBits: Integer): Integer;
    function  DrawBMP(x: Integer; y: Integer; w: Integer;
                      h: Integer; Bitmap: Integer): Integer;
    function  DrawJPEG(x: Integer; y: Integer; w: Integer;
                      h: Integer; jpeg_w: Integer;
                      jpeg_h: Integer; buf: Integer;
                      buflen: Integer): Integer;
    function  DrawBitmapFile(x: Integer; y: Integer;
               w: Integer; h: Integer; const FileName: WideString): Integer;
    procedure DrawBitmapClone(x: Integer; y: Integer;
               w: Integer; h: Integer; BitmapID: Integer);
    function  GetDC: LongWord;
    procedure TextOut(x: Integer; y: Integer; const Text: WideString);
    function  TextRect(x: Integer; y: Integer; w: Integer;
               h: Integer; const Text: WideString;
               StartPos: Integer; Alignment: Integer): Integer;
    procedure MoveTo(x: Integer; y: Integer);
    procedure LineTo(x: Integer; y: Integer);
    procedure Rectangle(x: Integer; y: Integer; w: Integer; h: Integer);
    procedure Hyperlink(x: Integer; y: Integer; w: Integer;
               h: Integer; const Name: WideString);
    procedure Bookmark(x: Integer; y: Integer; const Name: WideString);
    procedure Outline(level: Integer; x: Integer; y: Integer;
               const Name: WideString; const Caption: WideString);
    procedure SetTextDefaultAttr(const Font: WideString; Size: Integer);
    procedure SetTextAttr(const Font: WideString; Size: Integer;
               Bold: Integer; Italic: Integer; Underline: Integer);
    procedure SetTextAttrEx(const Font: WideString; Size: Integer;
               Charset: Integer; Bold: Integer; Italic: Integer;
               Underline: Integer; Color: Integer);
    procedure SetPenAttr(Style: Integer; Width: Integer; Color: Integer);
    procedure SetBrushAttr(Style: Integer; Color: Integer);

    property DC: OLE_HANDLE;
    property PAGE_PDFXRes: Integer;
    property PAGE_PDFYRes: Integer;
    property PAGE_PDFWidth: Integer;
    property PAGE_PDFHeight: Integer ;
    property PDF_Encoding: TxPDFEncoding;
    property PDF_Compression: TxPDFCompression;
    property PDF_BitCompression: TxPDFBitCompression;
    property PDF_ThumbNails: TxPDFThumbNails;
```

```
property PDF_FontMode: TxPDFFontMode;
    property PDF_PageMode: TxPDFPageMode;
    property PDF_JPEGCompress: TxPDFJPEGCompress;
    property PDF_EnhancedOptions: Integer;
    property PDF_MailMergeStart: WideString;
    property SECURITY_Permission: Integer ;
    property SECURITY_UserPassword: WideString;
    property SECURITY_OwnerPassword: WideString;
    property INFO_Date: WideString;
    property INFO_ModDate: WideString ;
    property INFO_Author: WideString;
    property INFO_Producer: WideString;
    property INFO_Title: WideString;
    property INFO_Subject: WideString;
    property INFO_Keywords: WideString;
    property INPUT_PDFFile: WideString;
    property INPUT_Mode: TxPDFInputFileMode;
    property StreamBuffer: Integer;
    property StreamBufferSize: Integer;
  end;
```

9.2 RTF2PDF

The RTF Engine in RTF2PDF is entirely controlled with the method "ExecCommand."

This IDs are currently supported:

9.2.1 Command Codes (C Syntax)

```
 #define WPCOM_RTFINIT       1000
 #define WPCOM_RTFVersion    1001
 #define WPCOM_RTFLOAD       1002
 #define WPCOM_RTFAPPEND     1003
 #define WPCOM_RTFMAKEFIELDS 1004
 #define WPCOM_RTFMERGE      1005
 #define WPCOM_RTFSAVE       1006
 #define WPCOM_RTFBACKUP     1007
 #define WPCOM_RTFRESTORE    1008

 #define WPCOM_RTF_PAGEWIDTH     1010
 #define WPCOM_RTF_PAGEHEIGHT    1011
 #define WPCOM_RTF_MARGINLEFT    1012
 #define WPCOM_RTF_MARGINRIGHT   1013
 #define WPCOM_RTF_MARGINTOP     1014
 #define WPCOM_RTF_MARGINBOTTOM  1015
 #define WPCOM_RTF_PAGECOLUMNS   1020
 #define WPCOM_RTF_PAGEROTATION  1021
 #define WPCOM_RTF_PAGEZOOM      1022
 #define WPCOM_RTF_READEROPTIONS 1023
 #define WPCOM_RTF_USE_PRINTER   1024

 #define WPCOM_RTFPRINT          1100

// Message IDs
 #define WPMSG_RTFLOAD           1001
 #define WPMSG_RTFPRINTPAGECOUNT 1002
 #define WPMSG_RTFPRINTPAGE      1003
```

9.2.2 Command Codes (Pascal Syntax)

```
const
 WPCOM_RTFINIT      = 1000;
 WPCOM_RTFVersion   = 1001;
 WPCOM_RTFLOAD      = 1002;
 WPCOM_RTFAPPEND    = 1003;
 WPCOM_RTFMAKEFIELDS= 1004;
 WPCOM_RTFMERGE     = 1005;
 WPCOM_RTFSAVE      = 1006;
 WPCOM_RTFBACKUP    = 1007;
 WPCOM_RTFRESTORE   = 1008;
 WPCOM_RTF_PAGEWIDTH    = 1010;
 WPCOM_RTF_PAGEHEIGHT   = 1011;
 WPCOM_RTF_MARGINLEFT   = 1012;
 WPCOM_RTF_MARGINRIGHT  = 1013;
 WPCOM_RTF_MARGINTOP    = 1014;
 WPCOM_RTF_MARGINBOTTOM = 1015;
 WPCOM_RTF_PAGECOLUMNS  = 1020;
 WPCOM_RTF_PAGEROTATION = 1021;
 WPCOM_RTF_PAGEZOOM     = 1022;
 WPCOM_RTF_READEROPTIONS= 1023;
 WPCOM_RTF_USE_PRINTER  = 1024;

 WPCOM_RTFPRINT     = 1100;

 // Message IDs:
 WPMSG_RTFLOAD = 1001;
 WPMSG_RTFPRINTPAGECOUNT = 1002;
 WPMSG_RTFPRINTPAGE      = 1003;
```

Index

- R -

- S -

- T -

- U -

- W -