.NET Developer Manual

www.PDFCONTROL.com

07.02.2007

Table of Contents

1 Introduction

The component 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.

Now version 3 of this powerful tool is available. Version 3 is completely backwards compatible so we decided to include this manual also for the last release of version 2 with the version number 2.90 - so you can compare the features of the old and the new version easier. (wPDFControl V3 introduces 2 new properties: CidFont and PDFAMode)

wRTF2PDF Version 3 is based on an entirely new rtf engine. It supports much more formatting features, including sections, text boxes and footnotes. Since RTF2PDF V3.5 you can also use it to create HTML, RTF, MIME or PDF text under ASP or ASP.NET! It is much more than just a PDF conversion engine! Please don't miss our demo! Or check out the product **TextDynamic**, this is the visual edition of this text creation and editing component.

Our component RTF2PDF offers the same possibility as wPDFControl. It does everything what wPDFControl does but, of course, its main feature is the integrated powerful text rendering engine. Since wRTF2PDF V3.5 it is also possible to create text and save in RTF or HTML, even MIME format. This is possible since the component exports all of its internal word processing abilities through a set of interfaces. This interfaces are in fact mostly the same as the interfaces used by our visual word processing control "TextDynamic". Only the interface IWPMemo has been replaced by a different interface (IWPEditor) which contains about 70% of the methods originally found in IWPMemo.

So, if you are using wRTF2PDF V3.5 or later (also called wRTF2PDF "PLUS") please refer to the specialized RTF2PDF manual which explains the API in detail.

wPDFControl can be used as .NET control, as ActiveX and also as raw DLL. For the DLL we have included a C++ include file. This manual describes the use with .NET.

1.1 License

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 or a universal RTF to PDF converter which is used from the command line.

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.

2 wPDFControl .NET

2.1 Quick Start

A) Installation

Please install the .NET wrapper class for wPDFControl into your .NET development system:

MS Visual Studio: Click right on toolbox and select 'Customize'. Use the 'Locate ...' button to

show the file open dialog to add the assembly wPDF.DLL.

Borland C# Builder: Click right on the toolbox and select 'Customize'. Add the path of the assembly wPDF.DLL (3rd page on the dialog).

Please activate the 3 components 'PDFControl, PDFPropDlg, RTF2PDF' which are all in the namespace 'wPDF'.

When you use the included demo projects or create your own project you always need to **add a reference to the wPDF assembly** installed in the wPDFControl directory under **\DLL\.NET\Demo\wPDF.DLL** or **\DLL\.NET\Full\wPDF.DLL**.

Please note that your application will always need
a) wPDF.DLL and
b) either of wPDFControlDemo.DLL or wPDFControl0x.DLL or wRTF2PDF0x.DLL in its directory to run.
Which of this DLLs are required depends on wether you are using the demo, the standard PDFControl or the enhanced RTF2PDF control which includes the PDF functionality with added RTF export functions.

B) Add the components to the form

Select the PDFControl and click on the form.
Select the PDFPropDlg component and click on the form.

Connect both component by selecting the PDFControl instance in the property 'PDFPropDlg.PDFControl'.

C) Add a button to the form to show the important PDF properties

In the OnClick event handler of button1 please use the code:

```
private void button1_Click(object sender, System.EventArgs e)
{
        pdfPropDlg1.Execute(false);
}
```

D) Add code to the form to show a file open dialog and let the user select metafiles and bitmap files which will be exported to PDF.

```
private void button2_Click(object sender, System.EventArgs e)
{
        // We need a filename
        if(pdfControl1.FileName=="") pdfPropDlg1.Execute(false);

        // A dialog to select a file name
        System.Windows.Forms.OpenFileDialog OpenDia =
        new System.Windows.Forms.OpenFileDialog();
        OpenDia.Filter = "Graphic Files|*.WMF;*.EMF;*.BMP;*.JPG;*.JPEG";
        OpenDia.Multiselect = true;
        // If ok then create a PDF file with all selected EMF files
        if(OpenDia.ShowDialog()==System.Windows.Forms.DialogResult.OK)
        {
                pdfControl1.BeginDoc();
                try
                {
                  for(int i=0;i<OpenDia.FileNames.Length;i++)
                  {
                        pdfControl1.DrawImage(OpenDia.FileNames[i],100);
                  }
                }
                finally
                {
```

```
                        pdfControl1.EndDoc();
                }
        }
}
```

E) What you can also do:

Draw using the property 'Canvas' which is compatible to the Graphics class:

```
if(pdfControl1.BeginDoc(filename))
{
        Font font = new Font("Arial", 10);
        SolidBrush brush = new SolidBrush(Color.Black);
        Pen pen = new Pen(Color.Red, 1);
        pdfControl1.StartPage();
        pdfControl1.Canvas.DrawString("Hello World",
                        font, brush,40, 50);
        pdfControl1.Canvas.DrawRectangle(pen, 20,20,300,300);
        pdfControl1.EndPage();
}
```

We have added a "font manager" to make it easier to use fonts. You don't have to create a font object, only to change the property pdfControl1.Font like this:

```
        pdfControl2.Font.Size = 11;
        pdfControl2.Font.Color = Color.Red;
        pdfControl2.TextOut("Hello ",40, 50);
        pdfControl2.Font.Style = FontStyle.Bold;
        pdfControl2.TextOut("PDFCONTROL ");

When you pass "\n" a new line will be created by advancing using
the last font height and resetting x to the position defined the last time with
TextOut(string,x,y). The intern position pointer will be advanced by using the return
value of the MeasureString function.
```

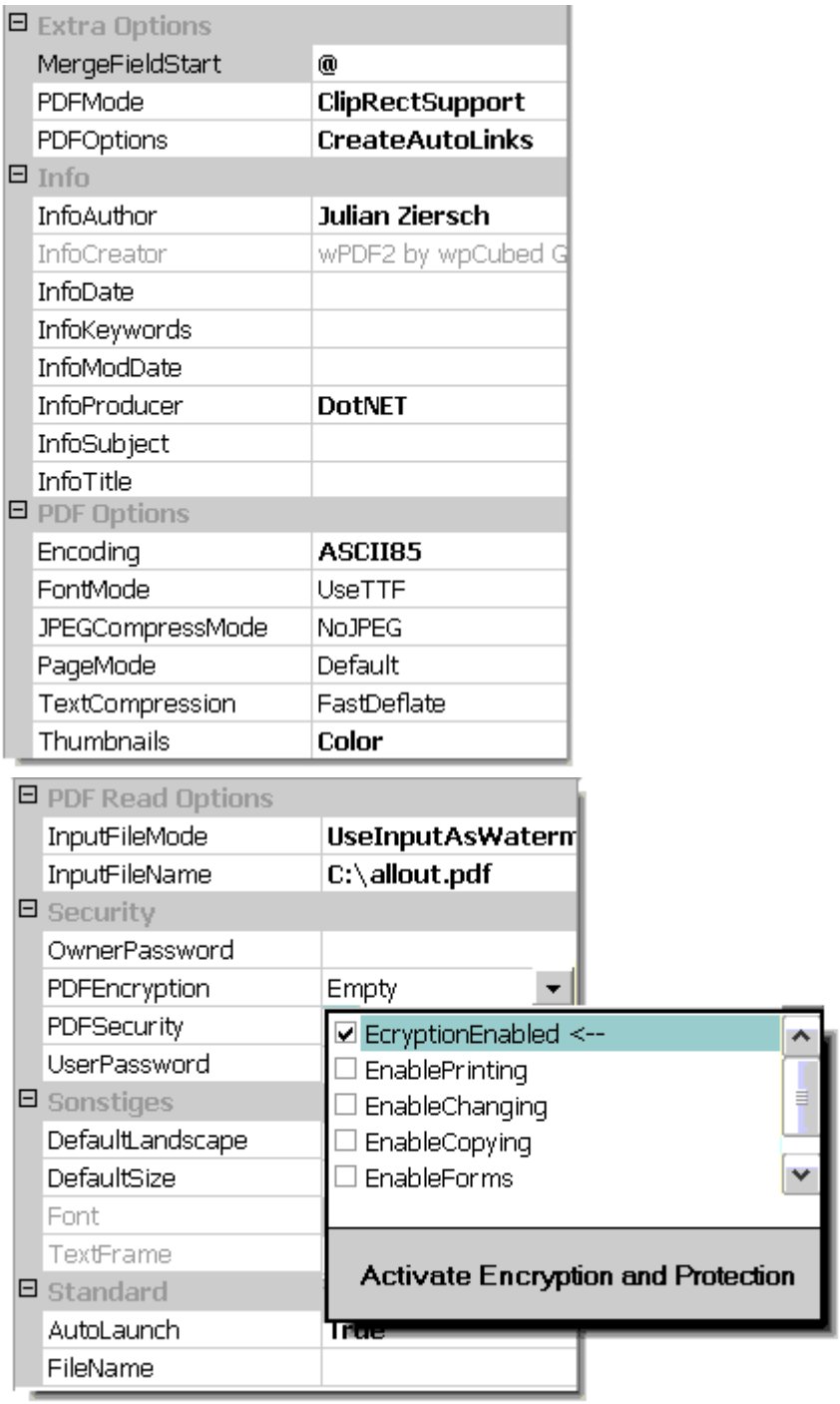
2.2 PDFControl

PDFControl ist the main component of PDFControl.NET. It contains a multiude of properties to change the way PDF is created and makes events available to be used in your C# or VB.NET application.

To use it you should specify a file name it its property 'FileName'. This is the file which will be created when you execute BeginDoc().

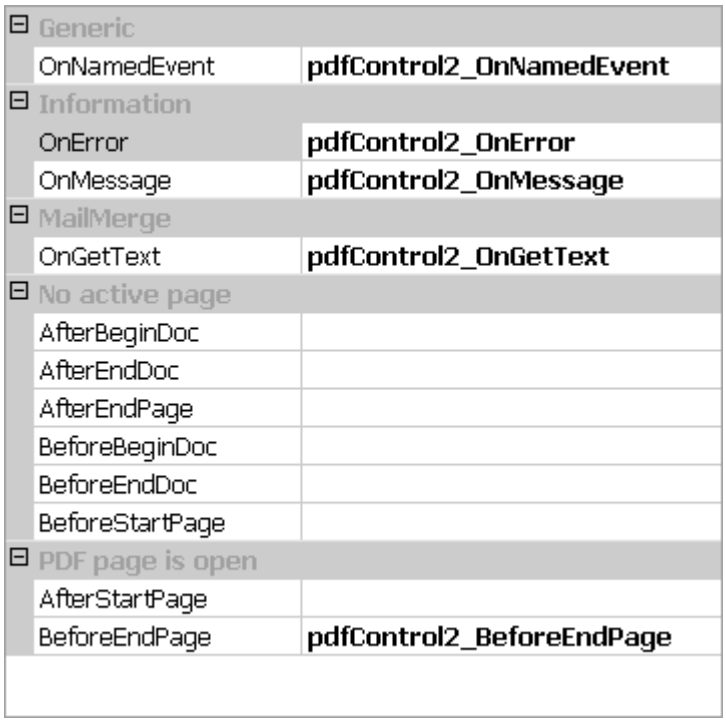
All properties have descriptive comments. The "flags" properties PDFMode, PDFCoptions and PDFEncryption will show a dropdown editor to make it easy to select the items you need.

PROPERTIES



In addition there is the property **'Canvas'** which is a true 'Graphics' object to be used with your .NET drawing code.

EVENTS



Generic	
OnNamedEvent	pdfControl2_OnNamedEvent
Information	
OnError	pdfControl2_OnError
OnMessage	pdfControl2_OnMessage
MailMerge	
OnGetText	pdfControl2_OnGetText
No active page	
AfterBeginDoc	
AfterEndDoc	
AfterEndPage	
BeforeBeginDoc	
BeforeEndDoc	
BeforeStartPage	
PDF page is open	
AfterStartPage	
BeforeEndPage	pdfControl2_BeforeEndPage

The event 'OnNamedEvent' will be triggered for all events under 'No active page' and 'PDF page is open'.

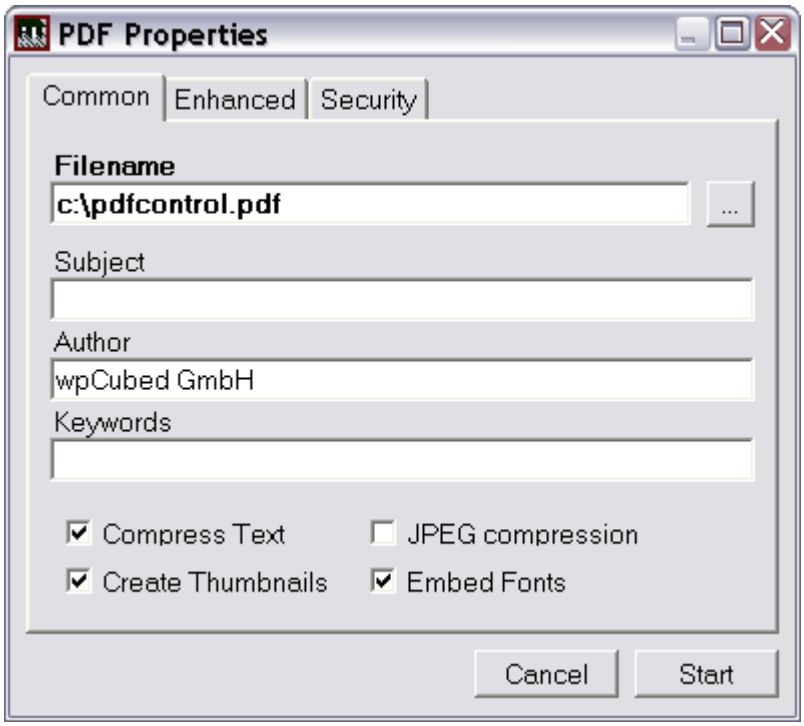
Esspecially useful is 'AfterStartPage' to draw a watermark on each page. This can also be used when you export and RTF file using RTF2PDF:

2.3 PDFPropDlg

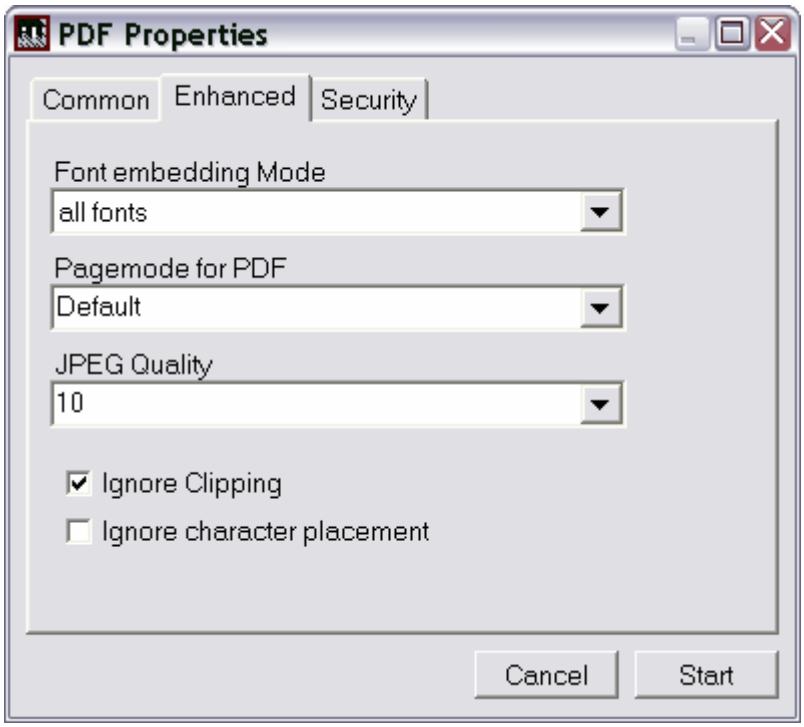
This component displays this dialog when the procedure Execute() is called:

To use it you need to assign its property 'PDFControl' to either an instance of PDFControl or RTF2PDF.

the first page with some standard properties



the second page with advanced settings



the third page can be used to change the security settings

Before the dialog is displayed the event **BeforeShowDialog** is triggered. You can use it to change the captions and or visibility status of certain elements:

```
private void pdfPropDlg1_BeforeShowDialog(object Sender, wPDF.ConfigPDF Dialog)
{
        Dialog.OKBtn.Text = "Start"; // Change the button text
        Dialog.linklabel.Visible = false; // Hide the URL label
}
```

This dialog does not publish all properties - only those which are likely of interest for the user of your application.

2.4 Properties

2.4.1 PDF Options

Property PageMode
By modifying this property you can select how the PDF file has to be displayed when the reader opens it.

Property Thumbnails
If this property is set to 'Monochrome' or 'Color' the PDF export component will create thumbnails in the PDF file.
Please note the the Acrobat(R) Reader Version 5 will create thumbnails automatically if none are found in a PDF document. This thumbnails have a high resolution since they are created using the complete information of a page.

Property TextCompression
By modifying this property you can let the PDF engine compress text. By using compression the file will be reasonable smaller. On the other had compression will create binary data rather than ASCII data. Please note that bitmaps and embedded fonts will be always compressed, either using Deflate or JPEG compression.

Property JPEGCompressMode
wPDF can compress bitmaps using JPEG. This will work only for true color bitmaps (24 bits/pixel) and if you have set the desired quality in this property.

Property Encoding
If data in the PDF file is binary it can be encoded to be ASCII again. Binary data can be either compressed text or graphics. You can select HEX encoding or ASCII95 which is more effective then HEX.

Property FontMode
Using this property you can decide weather TrueType fonts should be embedded in the PDF file or not. If fonts were not embedded in the PDF file text can be displayed wrongly if the used fonts are not installed on the PC of the reader of the PDF file. On the other hand embedded fonts causes the PDF files to be much larger. The size of the usual font file is 150KB! The embedding also slows the creation process down.

You can set wpOnEmbedFonts in property ExtraMessages to get a message (event OnError) once font data is embedded.

UseTrueTypeFonts : use true type fonts but does not embed the font data.

EmbedTrueTypeFonts : embed data of all used fonts. (You can still use ExcludedFonts to specify certain standard fonts)

EmbedSymbolTrueTypeFonts : embed only symbol true type fonts. (such as WingDings, etc.)

UseBase14Type1Fonts : Do not use true type. When this mode is selected you can only use Arial, Courier New and Times New Roman fonts.

Subset Embedding: Since version 2 it is now possible to ***reduce the size of the embedded data*** by removing the description for unused characters.

There are 2 options which do this for you:

EmbedSubsetCharsets will embed all the characters which are used in the codepage you selected.

EmbedSubsetUsedChar includes only those characters which have been used and so produces the smallest files.

2.4.2 CidFonts

This property enables the support for text written used by Character Identifiers. Here in the PDF text numbers are written which are mapped to certain glyphs in an embedded font. A special additional mapping table makes sure that the text can be extracted as unicode text. When the CID feature is used this means the fonts are embedded as subsets in a highly efficient way. PDF files become smaller this way. Since it works with character ids and not with charsets the export for say, Russian text, also works without having specified the charset explicitly. wPDF uses as character IDs the UNICODE values of each character - it can have used any number but we wanted to preserve the most information of the source text in the PDF as possible. Please note that the support for asian languages does NOT use the CID feature. Asian languages use special, perdefined fonts and mapping tables and require the charyset to be known to be propperly exported.

CidFontMode Values:

0 - CID feature is not used. The property FontMode rules font embedding
1 - all fonts are embedded. Unicode values will be used as character ids
2 - only symbol fonts will be embedded

Cid Font support is a new feature in wPDFControl V3.

2.4.3 PDFAMode

This property enables PDF/A support:

Values:
0=off
1=enabled

PDF/A is a new norm which based on PDF 1.4 - it was created to provide a guildline for the creation of document files which stay readable for the time to come. So the major demand for PDF/A complient files is that the used font files are embedded. Security measures are forbidden in PDF/A complient files as are links to external files. But there is more to PDF/A. We have checked the component carefully against the final documentation of PDF/A.

When you use RTF2PDF V3 additional (invisible) tags will be added to the PDF data. This tags make it possible to identify layout elements (such as header or footer texts) on a page. They can be also used by a PDF reader to convert the PDF data into text paragraphs, something which is otherwise at least difficult and impossible if a paragraph spans a page. The wPDF engine will also create tags to mark table cells. wPDFControl3 will also add the document information as XMP data to the PDF file.

PDF/A support is a new feature in wPDFControl V3.

2.4.4 Extra Options

PDFOptions

CreateAutoLinks - if active the PDF engine will create web links over text which starts with http://. This makes it easy to create a link to a web page.

```
Canvas.TextOut("http://www.wptools.de",10,20);
```

Please note that the autolink feature requires that the complete link is printed at once. Links which are inside longer text strings are not recognized.
Note: You can always create links using the Hyperlink procedure.

PDFMode

makes it possible to change the way the drawing commands are interpreted. For standard output this property should remain unchanged but if you create the output in your application its use might be required.

NoBITBLTFillRect: BitBlt is not used to draw a filled rectangle.

WhiteBrushIsTransparent: Do not fill objects which use a white background. Please note that this has no effect on bitmaps, only on rectangles or text which is printed opaque.

ExactTextPositioning: In general the PDF engine draws text to not only match the input but also look good. If your application requires that each character has to be at the same position as in the input you can use this flag to get a 1:1 output. (Implementation note: In the PDF file an efficient multi string output command is used, not several single text drawing commands)

NoTextRectClipping: Text can be clipped to its bounding box. To switch this off you can use this flag. It is a good idea to use this flag if your application uses a lot of TextRect() without expecting clipping.

ClipRectSupport: IntersectClipRect() and SelectClipRgn() are supported.

DontStackWorldModifications: If your application uses the windows function SetWindowOrgEx() a lot please set this flag. Normally please do not use this flag. It is only required if you see a small, 1 pixel shift in vertical or horizontal lines. If you have access to the printing code you can add SaveDC/RestoreDC around blocks which work with a different origin or resolution to avoid this small visual problem in the created PDF file. Normally you should not use this option!

NoTextRectClipping: Use the windows text output API you can provide a rectangle which is used to clip you are just printing. This is useful if you are printing table cells and don't want to overprint the contents of the neighbor cell. Usually this clipping rectangle is not active and wPDF will not add unnecessary clipping - but if your application uses this clipping without a real need for it, you can improve the quality of the PDF file by switching this flag on. This will skip the creation of clipping rectangles for text output commands.

DontAdjustTextSpacing: wPDF will normally use the character and word spacing to render the text to match the width requirement set by windows. This is necessary because the fonts in PDF have a slightly different width than calculated by windows. The reason are rounding errors in windows (or visual optimation) which works with integer positions while PDF uses floating points.
If you use this flag wPDF will not try to enlarge or shrink the text. Normally you won't see a problem, except that you are printing text lines which consist of different parts (font styles fro example).
This flag has no effect if wpExactTextPoitioning is active.

DontCropBitmaps: Using the command StretchDIB you can specify a rectangle of the source bitmap which will be printed in the destination rectangle. Unless this flag is active the PDF engine will internally remove the unprinted data.

DontCropBitmaps: The PDF engine will crop images if they are only partly painted. You can use this flag to switch this mode off.

AllowTransparentBit: Monochrome bitmaps can be optionally painted transparently if they are painted using an OR "ROP" mode or if the flag wpWhiteBrushIsTransparent. This will set the PDF format version to V1.3.

2.4.5 Security

Property PDFEncryption

EncryptFile - to switch on encryption
EnablePrinting - allow the user to print the PDF file
EnableChanging - allow to change the PDF file
EnableCopying - allow to copy text from the PDF file
EnableForms - enable interactive elements (do not yet apply)
LowQualityPrintOnly - if you use high compression this flag lets you switch off high resolution printing.

The property changes the rights of the user and if the file is encrypted..
You can protected the file from viewing if you also set the property UserPassword or you can simply prohibit copying. Currently wPDFControl supports the 40-bit and 128 bit PDF encryption. The later also causes the PDF to use PDF version 1.4.

property Security (Std40bit, High128bit)
You can change between

Property OwnerPassword
The password is required to edit an encrypted PDF file. If you don't set a password here a

password will be randomly created when you enable Encryption.

Property UserPassword
This is the password which will be used to encrypt the file. If you don't provide a password the rights for the user can be still restricted using 'Encryption'. But the user will then not be prompted for a password.

2.4.6 PDF Info

This properties change the fields which are used for the document info of the created PDF file. This information can be viewed in the PDF reader under 'Document Properties'. It is not printed.

2.5 RTF2PDF

2.5.1 Intro

This is a universal tool to convert RTF files into PDF. It supports embedded images, tables header and footer. Since it uses an internal RTF engine it is very fast but it does not support all features Word(tm) supports. However since Version 3 RTF2PDF uses a highly advanved word processing engine which supports much more RTF features than version 2. (The engine was completely rewritten after 2004) Despite its name RTF2PDF V3 also loads HTML code!

From Version wRTF2PDF V3.5 the engine exports all of its internal word processing abilities through a set of interfaces which are also used by our visual word processing control "TextDynamic". This make it much easier to

- load text (RTF, ANSI, HTML, UNICODE and MIME)
- process text, create tables, insert text and do mail merge
- generate text (RTF, ANSI, HTML, UNICODE and MIME)

Starting point for this new API is the interface **IWPEditor** which can be accessed through property "Memo".

If you have licensed RTF2PDF V3 please stop reading here and see the dedicated RTF2PDF reference manual. If you also use our product TextDynamic you can reuse code which was developed to work with this word processing component.

2.5.2 RTF2PDF Version 2 API

The RTF2PDF component inherits all properties and methods of PDFControl.

It adds methods to load, manage and export RTF files to PDF.

To convert a RTF file you first load it and then export it.

```
System.Windows.Forms.OpenFileDialog OpenDia =
new System.Windows.Forms.OpenFileDialog();
OpenDia.Filter = "RTF Files|*.RTF";
if(OpenDia.ShowDialog()==
     System.Windows.Forms.DialogResult.OK)
{
   rtF2PDF1.LoadRTF(OpenDia.FileName);
```

```
        rtF2PDF1.Export("C:\\Test.PDF");
    }
```

```
//Loads a RTF file
public void LoadRTF(String FileName)

//Loads RTF from a string
public void LoadRTFString(String RTFFormat)

//Appends RTF from a string
public void AppendRTFString(String RTFFormat)

//Saves the RTF file
public void SaveRTF(String FileName)

//Appends RTF file to the data stored in the RTF Engine
public void AppendRTF(String FileName)

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

//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 event procedure.
public void MergeExec()

//writes the RTF data to an intern data storage. To restore it use the procedure
MergeRestore.
public void MergeBackup()

//Restores text saved with MergeBackup()
public void MergeRestore()

//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.
public void SetColumns(int Value)

// 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.
// It uses the 'Filename' property as output name
public void Export()

// 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.
public void Export(String OutputFileName)

//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!
public void SetZoom(int Value)

//Changes the rotation of the PDF page.  To print landscape use the value 90.
public void SetRotation(int Value)
```

2.5.3 Example

Code to convert a RTF file to PDF

Example VB.NET

```
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 C#

```
System.Windows.Forms.OpenFileDialog OpenDia =
new System.Windows.Forms.OpenFileDialog();
OpenDia.Filter = "RTF Files|*.RTF";
if(OpenDia.ShowDialog()==System.Windows.Forms.DialogResult.OK)
{
   rtF2PDF1.LoadRTF(OpenDia.FileName);
   rtF2PDF1.Export("C:\\Test.PDF");
}
```

2.6 Methods

2.6.1 Initialisation

Creating metafile with wPDFControl is as easy as 1 - 2 - 3 or better as

BeginDoc - DrawImage - EndDoc.

If you need a a "Graphics" object to draw to, you need to open a 'page' first. To open a PDF page use StartPage.

BeginDoc - StartPage - Canvas.DrawString(..) - EndPage - EndDoc.

Note: Using 'CloseCanvas' you can always flush the graphic output stored in the Canvas property to the PDF file.

To use registered version of PDFControl and RTF2PDF you need to execute the function **SetLicense(String Name, String Code, uint Number)** to pass your license name, the code and the number. This activates the PDF engine.

Note:

Version 2 uses a license key similar to

(a) StartEngine("somename", "xxxxyyyy", 123456);

Version 3 can use a key such as

(b) StartEngine("somename", "xxxxyyyy@zzzzz", 123456);

(note the @ sign)

or this

(c) StartEngine("somename", "WWW-XXXX-YYYY-ZZZZ", 123456);

RTF2PDF V3.5 and later will only work with the key using schema (c)!

Note: You can also create a PDF document in a Stream object:
Simply pass the a Stream instance to the function BeginDoc.

Overview:

BeginDoc/EndDoc

```
// Use this function to set your license information
public bool SetLicense(String Name, String Code, uint Number)

// Use this function to start a new PDF file
// using the filename set in property FileName
public bool BeginDoc()

// Use this function to start a new PDF file using give filename
public bool BeginDoc(String FileName)

// Use this function to start a new PDF stream (not a file)
public bool BeginDoc(Stream OutStream)

// EndDoc closes a PDF file which has been opened with BeginDoc
public void EndDoc()
```

StartPage/EndPage

```
// StartPage starts a new page in a PDF file opened with BeginDoc.
// Y and Y are measured pt, this is 1 inch / 72
public bool StartPage(int w, int h, bool landscape)

// Starts a page to make that image exactly fit. You can sepcify a
zooming value
public bool StartPage(Image image, int ZoomValue)

// Starts a page with a certain ePage
// format: Letter, Legal, Executive, DinA3, DinA4, DinA5 or DinA6
public bool StartPage(ePage format, bool landscape)

// Starts a page using the DefaultPageSize
public bool StartPage()

// Closes a PDF page and writes it
public void EndPage()
```

StartWatermark/EndWatermark

```
// Starts a watermark with a certain name
// and  ePage format: Letter, Legal, Executive, DinA3, DinA4, DinA5 or
DinA6
public bool StartWatermark(String Name,ePage format)

// Starts a watermark with the DafaultPageSize and a certain name
public bool StartWatermark(String Name)

/ Closes a PDF watermark and writes it
public void EndWatermark()
```

2.6.2 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.

To draw a watermark use

```
public void DrawWatermark(String Name, int Rotation)
```

Rotation can be 0, 90 or 180.

2.6.3 Image Output

wPDFControl provides the function **DrawImage()** with several alternatives to draw a metafile or a bitmap.

If no page has been opened with StartPage a page will be automatically opened in either the DefaultPageSIze, or, if ZoomValue is not 0 to match the (zoomed) size of the image or metafile.

Possible Parameters:

image :	This is the Image instance with a bitmap or metafile
FileName:	an image file on disc
x,y,w,h :	This is the location the image should be drawn on the PDF page - measure in pt (1/72 inch)
ZoomValue:	This value (if not 0) can be used to resize the image. This is useful if the image should be drawn in correct aspect ratio

```
public int DrawImage(String FileName)
public int DrawImage(String FileName, int ZoomValue)
public int DrawImage(String FileName, int x, int y, int w, int h)
public int DrawImage(Image image)
public int DrawImage(Image image, int ZoomValue)
public int DrawImage(Image image, int x, int y, int w, int h)
```

Speciality for metafiles only: - DrawMetafile let you specify the x and y resolution which was used to create the metafile.
This can be useful if the PDFEngine does not pick up the stretching value correctly. (This function implies the ZoomValue set to 100)

```
public void DrawMetafile(Metafile metafile, int xres, int yres)
```

2.6.4 Hyperlinks

Hyperlinks mark a certain position on the PDF page which can be clicked by the user to either open a certain website or jump to a poition in the PDF file which has been or will be marked with a "BookMark".

To create a link you can use

public void Hyperlink(int x, int y, int w, int h, String BookMark)

or

public void Hyperlink(Rectangle Rect, String BookMark)

Please note that both functions are using the current Canvas coordinate trasnformation - this means you can use the same coordinates you use for DrawString(), Line() etc.

To create a weblink the bookmark must start with "http://"
To create a link to a certain file it must be start with "Launch://"

2.6.5 Bookmarks

Bookmarks can be used to mark link destinations in a PDF file.

To create a bookmark you can use

public void Bookmark(int x, int y, String BookMark)

or

public void Bookmark(Rectangle Rect, String BookMark)

Please note that both functions are using the current Canvas coordinate trasnformation - this means you can use the same coordinates you use for DrawString(), Line() etc.

2.6.6 Outlines

Outlines are displayed in a seperate window by Acrobat Destiller and a good way to navigate a document.

To create an entry use

Create an Outline level as jump to a certain bookmark
public void Outline(String Text, String Bookmark)

Create an Outline level as jump to the current page
public void Outline(String Text)

Create an Outline level as jump to a certain location on the page measured in pt
public void Outline(int x, int y, String Text)

Create an Outline level as jump to a certain location on the page measured in pt
public void Outline(Rectangle Rect, String Text)

To build the outline tree use

public void OutlineChild() // one level down

public void OutlineParent() // one level up
public void OutlineParent(int n) // n levels up

Example:

```
pdfControl2.Outline("Level 1");
pdfControl2.OutlineChild(); // Child on Level 2
pdfControl2.Outline("Level 2 a");
pdfControl2.Outline("Level 2 b");
pdfControl2.OutlineChild(); // Child on Level 3
pdfControl2.Outline("Level 3 a");
pdfControl2.OutlineParent(2); // Go 2 Levels up
pdfControl2.Outline("Level 1");
```

2.6.7 using "Graphics Canvas"

PDFControl exportes metafiles to PDF.

The wrapper class PDFControl and RTF2PDF also includes a object 'Canvas' which inherits of the class 'Graphics'. This means you can use it with the graphics operations you would usually use in C# or VB.NET.

```
pdfControl2.StartPage();
pdfControl2.Canvas.DrawString("Hello World",this.Font,new SolidBrush(Color.Black),40, 50);
pdfControl2.Canvas.DrawRectangle(new Pen(Color.Red, 1), 20,20,300,300);
pdfControl2.EndPage();
```

2.6.8 Mailmerge

Mailmerge is using the property MergeFieldStart and the event OnGetText.

If MergeFieldStart is not "" and any text which is sent to the PDF engine starts with it the OnGetText event will be triggered to let You modify the text which is actually displayed in the PDF file.

```
private void pdfControl1_OnGetText(object Sender, string Name, System.Text.StringBuilder
Value)
{
        // Clear the text
        Value.Length = 0;
        // and set a new text
        Value.Append("WPCubed GmbH");
}
```

2.7 Events

PDFControl publishes this events:

Generic	
OnNamedEvent	**pdfControl2_OnNamedEvent**
Information	
OnError	**pdfControl2_OnError**
OnMessage	**pdfControl2_OnMessage**
MailMerge	
OnGetText	**pdfControl2_OnGetText**
No active page	
AfterBeginDoc	
AfterEndDoc	
AfterEndPage	
BeforeBeginDoc	
BeforeEndDoc	
BeforeStartPage	
PDF page is open	
AfterStartPage	
BeforeEndPage	**pdfControl2_BeforeEndPage**

On Error is triggered for example if a bitmap cannot be converted by the PDF Engine.

OnGetText is triggered to let you specify the contents for a certain field (see MailMerge)

The Events under 'No active page' are triggered during the PDF creation. Do not use the 'Canvas' property inside your event handler for this events.

The 2 events under 'PDF page is open' is triggered to let you draw to the current PDF page while it is open.

3 wPDFControl DLL / ActiveX

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.

Please see the dedicated manual file PDFControl.PDF

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, "wpdfcontrol02.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();
 }
```