wPDF V5

Developer Manual

www.wpcubed.com

5/11/2023

Table of Contents

Foreword 0

Part I Introduction 1

Part II Features 2

Part III License Agreement 6

Part IV Whats New 9

Part V Credits / Intellectual Property 18

Part VI Installation 20

1 Create Package for C++Builder ... 21

Part VII WPViewPDF - a PDF view control 22

Part VIII QuickStart 23

1 Examples ... 23

2 C++ Builder Notes ... 25

Part IX Properties 26

1 CidFontMode ... 26

2 PDFAMode ... 26

3 Output Properties (Filename/Stream) ... 27

4 Modify Output ... 28

5 Compression ... 31

6 Encryption ... 32

7 Text Rendering ... 32

8 PDF Options ... 33

9 Mail Merge ... 34

10 DLLName ... 35

11 FontMode ... 36

12 Info ... 36

Part X Methods 36

1 Start/End Output ... 36

BeginDoc ... 37

EndDoc ... 37

StartPage ... 37

EndPage ... 38

StartWatermark ... 38
EndWatermark ... 38
2 Graphic Rendering ... 38
property Canvas ... 39
property CanvasReference ... 40
method DrawBitmap ... 40
method DrawBitmapClone ... 40
Method DrawDIBBitmap ... 40
Method DrawMetafile ... 40
Method DrawMetafileEx ... 41
Method DrawTGraphic ... 41
Method DrawGraphicFile ... 41
Method DrawJPEG ... 41
Method DrawCCITT ... 41
PrintForm ... 42
DrawPNGFile ... 42
DrawPNG ... 42
DrawJPG ... 43
3 Links and Bookmarks ... 43
Method SetBookmark ... 43
Method SetLinkArea ... 44
Method SetOutlineXY ... 45
Method SetOutline ... 46
4 Select Color (CMYK) ... 47
procedure SelectColorMode ... 47
procedure SetColorEx ... 48
5 Fields (Annotations) ... 49
Function DrawAnnotation ... 49
procedure DrawTextField ... 50
procedure DrawCheckbox ... 51
Field Example ... 52
6 Embed Data / Attach File or Stream ... 53
Method EmbedData ... 53
Method AddFileAttachment ... 54
Method AddXMPExtra (ZUGFerD) ... 55
7 GDIComment ... 56
Command IDs to create hyper links ... 57
Create fields ... 57
8 WPDF_ConvertImageFiles ... 59
Part XI Events 60
Part XII Linking with other products 61
1 WPTools ... 62
What is WPTools ... 62
WPTools Version 5, 6, 7, 8 or 9 ... 63
WPTools Version 4 ... 67
Print On Canvas ... 68
Outlines ... 69
Multi Document ... 69
Use PDF Watermarks ... 70

Print on Background of each Page ... 70
Window-less RTF to PDF conversion ... 72
2 ReportBuilder ... 73
How to use the ReportBuilder device? ... 73
How to create a PDF file without display of a file save dialog? ... 74
RichView in RB ... 76
3 FastReport ... 76
4 TRichView / TSRVRichView ... 78
5 RAVE Report ... 79
6 DevExpress ... 82
7 ACE Report ... 83
8 QuickReport ... 83
9 HTMLView ... 84
10 WPForm ... 85
11 RichEdit ... 87

Part XIII FAQ 88
1 Code to draw outlined text ... 90

Part XIV Tips 90

Part XV wPDF SourceCode License 93
1 How to use ... 93
2 Comparision to Standard Version ... 93

Index 0

1 Introduction

Supercharge Your PDF Creation with wPDF - the High-Speed Solution for PDF creation in Delphi and C++Builder applications.

In today's world, documents need to be easily accessible, editable, and shareable, while still maintaining their professional allure. Portable Document Format (PDF) is the go-to file format tailor-made for these exact purposes; it's a staple for businesses, educational institutions, and individuals alike who require an easy and secure way to share and store information. With many software options available to create PDF files, you might find yourself asking what sets wPDF apart from the rest.

Imagine a scenario where you need to generate high-quality PDF files in a fraction of the time compared to your current method while providing seamless flexibility and compatibility with the tools you already use in your application. Whether you're using it directly to convert EMF (enhance metafiles), integrating it with your preferred components, or linking it with the word processing component WPTools, wPDF can be your PDF creation champion.

wPDF is a high-speed PDF creation tool that efficiently produces files compatible with most GDI drawing commands in Windows applications. With its developer-friendly API, wPDF can be used alone, linked with WPTools or integrated with other, 3rd party components.

This versatile PDF maker supports various features like compression, 128-bit encryption, font subsetting and more.

Compatibility with various software components

wPDF, a high-speed PDF maker, boasts impressive compatibility with various software components, making it a versatile choice for developers. This user-friendly tool works seamlessly with popular programs like WPTools, Fastreport, Reportbuilder, QuickReport, and HTMLView, among others.

In addition, its convenient canvas property can be used for direct drawing or in conjunction with your own components to create PDF files instantly. This adaptability makes wPDF the go-to solution for users who need to work with different applications and tools in their projects. You can also mix and match the output from different document creation pipelines by starting the PDF using "BeginDoc" and export from different components until you call "EndDoc".

Compatible canvas property for direct drawing

wPDF provides developers with a compatible canvas property for direct drawing, making it a versatile tool for creating PDF files. This feature allows for effortless integration with various components, resulting in high-quality output. The canvas

property enables developers to create precise and detailed documents with ease, ensuring that the final product meets their expectations.

wPDF utilizes metafiles for its internal PDF engine. Metafiles are used by Windows to record the graphic commands.

Use of DLL for easier engine upgrades

The use of a DLL for the wPDF engine allows for easier upgrades in the future. This is because it enables the incorporation of current compiler technology, even if the main part of the project uses an older version. By taking advantage of this feature, developers can benefit from a more efficient and streamlined process when integrating new components and improvements into their projects. Ultimately, this approach results in a faster and more reliable PDF creation tool. However, if you are a Delphi Developer you can also license the DCU (object files) to link in the PDF engine directly into your applications EXE file.

PDF tagging for separating paragraphs and tables

PDF tagging allows the separating paragraphs and tables within a document. This feature improves acessibility of PDF files and is supported when wPDSF is used in combination with WPToolls, our word processing control.

The creation of PDF files compatible with PDF/A norm is a noteworthy feature of wPDF. This compliance ensures that certain standards are met, providing better accessibility and long-term storage for documents.
In order to achieve PDF/A compatibility, wPDF incorporates elements such as metadata, embedded fonts and PDF tags. These enhancements make it easier to convert PDFs back to text and the use of screen readers.

Embedding external files and streams using API

wPDF is a powerful tool that allows developers to easily embed external files and streams using its API.

2 Features

Used alone, with your own component or linked with WPTools, wPDF instantly creates PDF files at high speed.

wPDF supports:
- WPTools 4 (incl. links and bookmarks, component included) ,
- WPTools 5 (incl. links and bookmarks, component included) , (see demo)
- WPTools 6 (incl. links, fields and bookmarks, component included)
- WPTools 7 (incl. links, fields and bookmarks, component included)

- WPTools 8 (incl. links, fields and bookmarks, component included)
- WPTools 9 (incl. links, fields and bookmarks, component included)
- ReportBuilder, (interface included)
- ACE Reporter, (interface included)
- RichEdit, (example included)
- RichView
- RAVE Report, (interface included)
- HTMLView,
- FAST Report 2, (interface included)
- FAST Report 3, (interface included)
- FAST Report 4, (interface included)
- QuickReport, (interface included)
- Developer Express(tm) EXPRESS PrintingSuite, (example included)
- WPForm, (example included)
- List&Label

and **most other components which can create a metafile or draw to a HDC / Canvas**. You always can combine the output of different components into one PDF file!

wPDF has been become the standard PDF tool for Delphi and C++Builder. This component provides you with a compatible 'canvas' property which you can use for direct drawing. Metafiles (EMF) are created internally and then sent to the PDF engine. This also allows debugging, if a PDF does not look as expected.

wPDF Standard uses the engine encapsulated into a DLL. The DLL is provided in 3 versions, one compiled with Delphi 2006, one compiled with Delphi 10.1 as 32bit and 64bit version.

Using an engine compiled into a DLL makes it easier to upgrade the engine and makes it possible to use current compiler technology even if the main part of the project is compiled using something older.

wPDF PLUS comes with the DCU files for several Delphi Editions. There is also an edition with full source.

wPDF is one of a handful of PDF tools which can create tagged PDF files.
Use wPDF with WPTools 6 or later to create PDF files which include tags to separate paragraphs and tables. That makes it easier to convert a PDF back to text or to use a screen reader.

Create PDF files compatible to PDF/A norm

This norm does not only limitate the use of some special PDF features (encryption is not allowed) but also has requires that certain demands are fulfilled:

a) Metadata must be available for document - wPDF will create an XMP meta data blob to the PDF file
b) Fonts must be embedded - wPDF can embed fonts, also subsets
c) PDF tags - requires proper input, it is best if you use WPTools - the description of a PDF page will be then tagged so that the reader is able to detect which elements are part of the layout and which are part of the text. This feature was difficult to implement since the tagging has to work across different pages, i.e. a paragraph is continued on the next page.
d) creation of a page index as tree instead of a simple list to avoid to have too many entries in an array.

Work with CID fonts

This feature makes it possible to use unicode text (russian, greek etc) on a page without having to select the code page. So this feature is often described as "unicode support". Technically it means that characters are embedded using character ids and the PDF contains information to tell the reader which ID translates to which character. However wPDF5 will use the standard unicode values as character IDs to make it easier to extract text from a PDF file. Of course it will also create the ToUnicode mapping information which is required for PDF/A compliancy. Please note that CID fonts are not used for asian laguages, those use the predefined standard codepages.

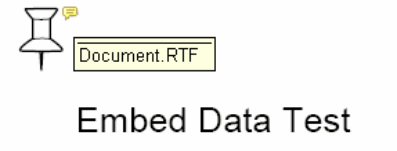
Create smaller PDF files

Using CID fonts will usually make a PDF file smaller, but wPDF 3 will also try to avoid to embed the same picture more than once which can reduce the size of the created PDF file even more.

Embed files

Using the API you can embed files into the PDF file, maybe to save the source document to the PDF file as well.

Example:



wPDF 5 can also embed external files and streams using AddFileAttachment.

Add XMP metadata

This can be done in wPDF 5 using AddXMPExtra.

Support for GDI function TransparentBitBlt

This GDI function let you specify a color which should be transparent.

wPDF 5 also handles the emulated transparency created by GDI+.

Support for standard brush styles

The standard brush styles (hatching) will be translated into a PDF pattern command.

Create edit and checkbox fields

using DrawTextField and DrawCheckbox

Embed bookmarks names (named destinations)

use flag wpPreserveBookmarkNames in the "Modes"

Manual and automatic conversion of text to outlines (glyphs)

RC4 Encryption, 40 and **128 bit!**

Embedding of **true type font subsets**. This means that if you embed a true type fonts and use only a few characters defined in it the new PDF engine is able to remove the description of the characters which are not used. This reduces the size of data which needs to be embedded helping to make the PDF file smaller.

In general wPDF is the must have component if you intend to create PDF files from your Delphi or C++Builder application. (Please use "wPDFControl" for VB, VC or other development systems)

When you use it **with WPTools you have the perfect RTF2PDF converter** - it supports different text styles (bold, italic, underline, strikeout, sub- and superscript), text alignment, tables, header and footer and embedded metafiles or bitmaps. Don't bother with Richedit based solutions - WPTools supports justified text, header and footer, different tabstops, tables, paragraph spacing and hyperlinks.

Do you need to view, print, merge or split
PDF files.
Check out our product WPViewPDF5:

This component is able to view and print PDF files which were created with wPDF or similar PDF engines. It supports encryption, embedded fonts, decryption, embedded JPEG images .... What is the advantage of such a tool?

The Adobe (tm) viewer is very large and most important it is questionable if it is allowed to embed it into applications in future. Since it only loads *files* it is impossible to load information from memory streams. Our solution has this

ability and makes it also easy to load multiple files and print them into one printer job. It is designed as a window class which makes it easy to use it in Delphi, .NET (winforms), VC ... avoiding any OCX hassle.

3 License Agreement

License Agreement, wPDF Version 5
Standard, InternetServer License and 'SourceCode'
Copyright by WPCubed GmbH, all rights reserved

The distribution license requires:

If wPDF is used in a project which is developed by a group of developers, ALL members of this group must have a wPDF development license - so please consider the TEAM or SITE license.

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 Julian Ziersch for volume discounts and site licensing agreements.

The SITE License is valid for any number of developers who work within one company network within one building. Their number should not exceed 20 - otherwise you need the corporate license. We also sell a TEAM license for up to 6 developers.

This documentation and the VCL 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.

wPDF uses the public zlib, aes, jpeg and RC4 routines. LZW is not used, neither are GPL licensed libraries such as "freetype".

wPDF Standard License

This License enables you to use the wPDF technology in all your products and distribute it to your customers without the need to pay any royalties.

Important: You may not distribute any Pascal source or object files or use the technology in a module (VCL, ActiveX, COM ...) which can by 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 the wPDF technology in universal PDF creation tools like virtual printer driver. wPDF 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.

Into the "Producer" entry of the PDF file properties always "wPDF3 by WPCubed GmbH" will be written to the created PDF file. (Note: the registered version does *not* print a watermark or show an info/nag-screen)

wPDF Server License PRO

In Addition to the standard license you may also use the wPDF technology in any programs (services, CGIs, ActiveServers ...) which will work on Internet server to create PDF files to be distributed over the WEB. If you purchase the server license you also get the DCUs (for Delphi 5,7, 2010, XE8, Delphi 10.1 and later, Delphi 11) to compile an application which does not required the PDF-Engine DLL - this avoids threading issues.

wPDF Sourcecode License

In addition to the two licenses above this agreement applies to the "wPDF SourceCode":

The source code cannot be purchased or licensed by companies who produce and/or sell PDF printer driver or PDF or RTF libraries for any programming language! Please contact us before paying for a source code license.

After purchase wpCubed GmbH supplies the source code of wPDF Version 5 PDF Creation. The source makes it possible to compile the PDF-Engine right into an application and debug the PDF conversion. The source

code is provided mainly to give companies the security of having access to the full source to the tools they are using.

The source code includes the algorithms to create PDF files, the same code our product "wPDF Standard" uses except for the DLL interface part. That is not required since the wPDFPrinter will use the PDF Engine directly.
The source code does not include any PDF reading algorithms.

The PDF-Engine source may be only visible to one person within the company who purchased it. In case of the Source "SITE" License it may not leave the main company buildings network. (For example it may not be sent to offshore/freelance programmers)
Distribution of the source in this or any other way invalidates the license immediately. The provided DLL version may be sent to freelance developers to work on the development of a product.

The use of the PDF engine is licensed under the following restrictions:

The part of the PDF Engine which writes the '**Creator**' info may **not** be changed. Created PDF files must contain "wPDF by WPCubed GmbH" as "producer" unless you negotiated a different agreement with WPCubed GmbH.

wPDF or any parts of it (neither DLL nor Sourcecode) may not be part of a printer driver product or any kind of development software, such as DLL, OCX, ActiveX or VCL.

The source code or parts of it many may not be used in any kind of PDF *viewer* application or component.

The Source code or parts may not be used in any stand alone tool which converts RTF or Word tools to PDF. Server applications (built with the Internet Server License) are not affected by this limitation.

The products which use the wPDF Engine *must* give WPCubed GmbH credit in "about box" and/or manual. Other royalties do NOT apply. The distribution within the application EXE is royalty free - if the above limitations are respected.

Contact:

WPCubed GmbH
St. Ingbert Str. 30, 81541 Munich, Germany
Tel.: +49-89-49053501
Fax.: +49-89-49053504
http://www.wptools.de
support@wptools.de

4 Whats New

wPDF Version 5.1.0 (10.5.2023)
- **enhanced font baseline calculation.**
We recommend to activate the switch wpUseNegativeWndExt in the property ModesEx. It makes sure that for upside down coordinate systems the baseline is calculated correctly.
- enhanced text background painting.
- enhaced ZIP support

wPDF Version 5.0.1 (19.10.2022)
* fix: pen styles were not always reset correctly

wPDF Version 5.0 (19.9.2022)

*** custom line styles for GDI drawing code**
*** better text baseline aligning logic**
*** better text rotation logic**
*** better font support**
*** support for color key masked PNG files**

wPDF Version 4.82 (16.11.2021)
* improved stability of font subsetting

wPDF Version 4.78 (8.4.2021)
* improved cmap writing to avoid problem with some PDF reader
* always handle Segoe UI Symbol as symbol font
* improve 64 bit support

wPDF Version 4.77 (21.4.2020)
* improved PDF engine
* embedded files are now saved with optional Size property

wPDF Version 4.76 (27.9.2019)
- improved baseline calculation for fonts on upside down coordinate systems
- auto limit the radius for round rects

wPDF Version 4.75 (18.5.2019)
- fixes problem with CIDSet property for fonts (for PDF/A)
we recommend to use FontMode = wpEmbedCIDFonts if you need PDF/A

wPDF Version 4.71.1 (14.3.2019)
- fixes align problem with text with contains characters which were not part of the current form

wPDF Version 4.71 (16.2.2019)

- fixed problem which caused certain fonts to be exported too large.
- fixed baseline on rotated and centered text

wPDF Version 4.70.0 (2.11.2018)
- revised baseline position when using Type3 fonts
- several optimizations in EMF to PDF conversion

wPDF Version 4.60.0 (8.6.2018)
- revised font height calculation
- improved baseline for certain fonts when converted to Type3
- fix possible stability issue
- the component now sends an error message event if the windows font embedding routine fails. In this case the internal fall-back subsetting is used.

wPDF Version 4.50.0 (7.3.2018)
- improvement of baseline calculation for Type3 fonts
- fix problem that some fonts were not embedded.

wPDF Version 4.38.0 (3.11.2017)
- further improvement of names array
- fix problem with append method despite being depreciated (please use WPViewPDF PLUS to append PDF files)

wPDF Version 4.37.0 (17.9.2017)
- optimize syntax in written names array to solve problem with ZUGFeRD check service
- fix problem with font resources problem within 64 bit programs when subsets were used

wPDF Version 4.36.0 (26.7.2017)
- optimize calculation of baseline for type 3 fonts.

wPDF Version 4.35.2 (18.5.2017)
- fix problem with font embedding introduced in 4.35.1

wPDF Version 4.35.1 (4.5.2017)
+ added support for Delphi 10.2
- fix for stability issue with large unicode fonts in 64bit environment
- improved XMP writing

wPDF Version 4.33 (10.10.2016)
* fixes problem with unicode links
- fixes problem with ArcTo in paths.
* improvment for thai fonts
- fix strikeout problem and Type3 fonts
* use property ModesEx: wpDisableRotationAlignCorrection to fiox problem is rotated textes are moved

wPDF Version 4.15 (16.2.2016)
* updated Arc routine
* outlines now work also when named destinations were used
* improvement to attached files

wPDF Version 4.14 (27.1.2016)
- improvement in AttachFile method to work with unicode file paths
- fix in string compare method used when writing names, such as named destinations.
  (Note - please do not use the sign # in named destinations)

+ new method: WPDF_ConvertImageFiles

wPDF Version 4.12 (2.12.2015)
- text which looked like barcode was not printed correctly
- fix problem with dash sign
- text which looked like barcode was not printed correctly
- fix problem with dash sign
* several fixes in PDF conversion (rotated text)

wPDF Version 4.11 (7.10.2015)
+ support for Delphi 10
* improved combobox fields
- fix for Type3 embedding
- list of embedded files was not always detected correctly by reader
+ added flag wpdfaLevel3B which is essential for the ZUGFeRD support
- the font Segoe UI Symbol was not exported correctly
* improved code to undo a SetWorldTransform in PDF code
- rotated text was not printed at the correct position if it was converted to glyphs

wPDF Version 4.09 (27.5.2015)
- fixes problems with CIDFont export when certain glyphs were not found
- fixes problem with Cambria Math font

wPDF Version 4.08' (9.3.2015)
- fixes a problem with polypolygon api used inside of pathes

wPDF Version 4.08 (28.2.2015)
- calculation of text rectangles have been updated
- wpNoTextScaling was ignored before
- hatches are not drawn with a background (caused black boxes)
- update of annotation used for embedded data

wPDF Version 4.07 (7.11.2014)
* wpAllSymbolFontsAsGlyph mode will now create Type3 fonts for symbols, not individual glyphs.
+Type1 fonts can now be embedded (subset is not possible)
- improved y coordinate calculation in glyph output, i.e. when printing symbols
- improved font ability detection
+ DevModesEx wpPrintTextWordByWord is useful when tabs are printed using the dx parameter of ExtTextOut
(Only required when Type3 and CIDMode is not used)
- fixed broken support for WPTools 4

wPDF Version 4.06' (23.9.2014)
- further improve support for images with transparency masks
- fix problem TextOut problem. Only use background color for the text itself.

wPDF Version 4.06 (19.9.2014)
+ added support for Delphi XE7
- improved Type3 text printing on rotated coordinate systems
- fixed problem with semi transparent images
- with some drawing code a rg command within a path caused a message from the PDF reader
* when creation annotations it is now possible to suppress that they are added to the interactive AcroForm Fields directory by adding 8192 to the AcroMode parameter

wPDF Version 4.05 (17.7.2014)

+ property Type3Fonts is a list of fonts which should always be embedded as Type3
+ Create combobox fields using the API DrawTextField
- fix problem when embedding Verdana Italic

wPDF Version 4.04 (20.6.2014)
+ WPTools 7.14 or later now export PNG with transparency masks
* improvement to Type3 font embedding

wPDF Version 4.03 (5.6.2014)
* improvements to PDF/A creation
* improvement to Type3 font embedding
+ PNG support now handles PNGs which use palette
+ DrawPNG function can now draw into the context of the Canvas (for better Z-Order control)
+ DrawJPG - exprt JPEG data directly
+ DrawAnnotation can create various annotations by directly specifying PDF parameters.
  Also appearance streams can be added and Popup Annotations can be assigned
+ GDI commands to create hyperlinks

wPDF Version 4.02 (11.5.2014)
+ added support for Delphi XE6
- fixed problem with schema extension with non unicode compiler

wPDF Version 4.01 (4.4.2014)
+ add new method: function AddFileAttachment(
        Name, Desc : WideString; Filename : WideString;
        const Typ : WideString = 'text/xml';
        ModDate: TDateTime = 0):Boolean;
+ add new method: AddFileAttachment(
        Name, Desc : WideString; Stream : TStream;
        const Typ : WideString = 'text/xml';
        ModDate: TDateTime = 0):Boolean; overload;
+ add new method: procedure AddXMPExtra(const SchemaPartXML, InfoPartXML : WideString) .
this method can be used to add custom meta data in XMP format.

When using WPTools please make sure to add the conditional **{$DEFINE wPDF5}** to the file WPINC.INC to activate PNG support with transparency.

wPDF Version 4.00 (19.3.2014)
+ add new fontmode: wpEmbedType3 - create type3 fonts which are very compact
+ add new fontmode: wpEmbedCIDFonts - select the CID Font "unicode" mode which was added to wPDF 3
+ add properties: Info.CopyrightNotice, Info.CopyrightURL and Info.CopyrightPublicDomain
+ DrawPNGFile. This methods exports a PNG file. It requires the PDF Engine to be able to decode PNG.
+ DrawPNG: This method exports PNG data. It requires the PDF Engine to be able to decode PNG.
* many optimizations to the EMF conversion engine

wPDF Version 3.82 (2.11.2013)
+ add support for Delphi XE5
* fix problem with internal exception when textheight of 0 was used

wPDF Version 3.81 (5.7.2013)
- fix problem with named destinations which was included in unicode version due to new VCL

- fix problem with external hyperlinks which were sometimes encoded wrongly

wPDF Version 3.80 (2.6.2013)
- adapted for WPTools 7
- fix problem with glyphs created for text which was using € or TM sign.

wPDF Version 3.79 (11.3.2013)
- fixed problem with font subsetting in 64 bit edition
- fixed problem with "newsubsetting"

wPDF Version 3.77 (14.11.2012)
+ added support for Delphi XE3
- certain symbol fonts were embedded with wrong width table. This caused problem with barcode font 2 of 5
- fix problem with font "Times New Roman"

wPDF Version 3.75 (19.12.2011)
- important: fixes problem with image hashing introduced by new MP5 code in V3.70.
* new installed. D2011 is now named XE. Package files are copied to the VCL directory

wPDF Version 3.70 (14.11.2011)
- improved font rendering and text placements

wPDF Version 3.62 (13.6.2011)
- fixed bug in the export of RAW CCITT data
- sometimes debug metafiles were created- This has been fixed.

wPDF Version 3.61.5 (17.2.2011)
* revised code for Delphi 2010 and Delphi 2011 (Check the name of the unicode DLL. Previous detection was not reliable)

wPDF Version 3.61.2.3 (25.9.2010)
* revised font embedding code
* correct saving of unicode info items when using Delphi 2010

wPDF Version 3.61.2.1 (28.8.2010)
+ EmbedData: In case there is no open page R will be interpreted in 72dpi PDF coordinates and the data will be linked to the last page.
+ If You use Delphi 2009, 2010 or Delphi 2011 (XE) you may use the DLL wPDF500W.dll. You can then set the filename as unicode string.
Use property DLLName to select this DLL.

wPDF Version 3.61 (18.8.2010)
- fix problem with embeddata with Delphi 2010 (unicode pointer error)
* SetOutlineXY can now be used after EndPage or after a page was automtically created
- cmap was written incorrectly

wPDF Version 3.60.1 (5.5.2010)
* XMP information is now correctly written as UTF8 encoded data. Unicode texts in info items are now stored.

wPDF Version 3.59 (14.4.2010)
* change in AcroField creation. When used with WPTools 6 not standard PDF fonts but regular windows fonts are used.
* change in the FastReport and ReportBuilder support units.

wPDF Version 3.57 (22.12.2009)
* updated PDF engine
- fix resource leak

wPDF Version 3.56 (24.7.2009)
+ wpPreserveBookmarkNames flag in Modes. If set the bookmarks in the text will not be saved as fixed links, but as named destinations.
  This makes it possible to address them from outside the control.
* fix of possible resource leak
* improved font detection

wPDF Version 3.54.1 (23.6.2009)
* improved Delphi 2009 support

wPDF Version 3.54 (1.6.2009)
* new Mode flag: wpDontSimulateItalicFonts.
  Disable automatic simulation of italic fonts when the font does not support this mode. Instead "Arial" is selected.
* Pen Width will be exported more exact

wPDF Version 3.53 (27.5.2009)
- bugfix in XMP writing code. Depending on locale setting a wrong date format was written
- PDF/A tags were not written

wPDF Version 3.52 (8.5.2009)
- bugfix in D2009 support
- fix problem for drawing data which did not initially use the GDI SelectMapMode but was using a different resolution
* PDFAMode now lets You select "Level B". This switches off the automatic PDF tagging

wPDF Version 3.50 (22.2.2009)
* updated support for Delphi 2009 (requires extensive changes in PDF engine)
* new outline method to render Arabic text correctly. Using the property "Modes" Arabic text can be always exported as outlines
- bugfix in image export - sometimes a bitmap was exported as it it was transparent
- improved PDF/A feature

wPDF Version 3.30 (4.11.2008)
+ support for basic gradient fill
+ automatic BOLD simulation if a font is not available as bold edition
- improved CID font mode for mixed text which uses different charsets
+ Delphi 2009 support - operable, though not 100% tested yet

wPDF Version 3.24 (18.6.2008)
+ automatic simulation of italic in case a font is not available as italic edition. (example: Comic Sans)

wPDF Version 3.23 (20.5.2008)
+ when text is exported as glyphs now center/right alignment is supported
+ fix for bold font simulation

wPDF Version 3.20 (14.3.2008)
+ new SelectColorMode API to generate CMYK pen, font and brush colors
+ new SetColorEx to specify CMYK color values directly
+ new field support: DrawTextField and DrawCheckbox
- improved PDF/A compliancy support (fix for color spaces, cmap and XMP data)

- improved support for the Arc and ArcTo GDI method (there was a problem when the MAP mode was set to HI_ENGLISH)
- MS SansSerif caused problem in "CID" mode. (MS SansSerif will be always exported as "Arial")

wPDF Version 3.14 (16.1.2008)
+ fix for SelectClipPath API
+ added code to support new features in the upcoming WPTools 6

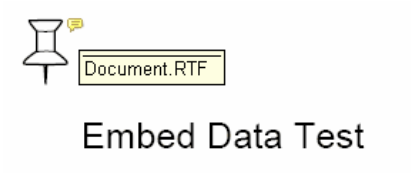
wPDF Version 3.12 (1.12.2007)
+ new DeviceMode: wpSimulateBoldFonts - instead of using a bold font type bolding a font is simulated

wPDF Version 3.12 (19.9.2007)
* writing to a protected file could disturb the DLL load/unload mechanism. Fixed.
* better handling of font embedding of symbol fonts
+ Outlines can now be set as UTF8 - use SetOutlineCharSet(CP_UTF8)

wPDF Version 3.10 (30.4.2007)
+ new font subsetting code (can be deactivated using flag wpUseOldFontSubsetting in Modes)
+ now characters which are not found in the current font or charset are drawn as outline. (can be deactivated using flag wpDisableAutoGlyphs in Modes)
+ optionally the complete text can be exported as outlines (curves) - flag wpTextAsGlyphs in Modes

wPDF Version 3.06 (22.1.2007)
- important fix of a problem with some font files which were not supported by the new CID-Font feature
+ better handling of pen join and end styles
+ new method: EmbedData - with it you can embed binary data into a PDF file. When the user clicks an open dialog will be displayed by Acrobat Reader:

Example for EmbedData API:




```
data := TMemoryStream.Create;
wPDF.Source.SaveToStream(data,'RTF');
wPDF.EmbedData('Document.RTF',
          Rect(100,100,200,200),
          wpemPushPin, data,
          wpemDefault, 'RTF');
data.Free;
```

wPDF Version 3.05 (15.12.2006)
- fix of unexpected plain to bold text style conversion
- better handling of unknown font names
- some general improvements in engine

wPDF Version 3.00 (4.9.2006)
- **new property PDFAMode**
- **new property CIDFont**
- **improved and enhanced PDF engine (see intro)**

wPDF Version 2.75 (18.8.2006)

- Support for Arc() and ArcTo() GDI function
- More stable EMF converter - detects problem in input data before AV happens
- Improvements to DLL and VCL to support multi threaded use
- Carefully checked for resource- and memory leaks
- **new** procedure **PrintForm**(Form: TForm; screenshotmode: Boolean) - exports a screenshot
- Changes to VCL to work with wPDF Version 3 DLL (note: version 3 needs new license info)

wPDF Version 2.12 (18.8.2003)
- compressed BMP are now accepted
- a few fixed to metafile to PDF conversion

wPDF Version 2.10 (12.5.2003)
...makes wPDF the PDF tool which supports the most 3rd party controls!
- wPDF now also works with the Developer **Express(tm) EXPRESS PrintingSuite**
- An all new **ReportBuilder**(tm) device is now included.
- The support for **ACE Reporter**(tm) has been optimized
- wPDF now includes a renderer for **RAVE(tm)** 5 report with introduces an incredible quality when it comes to export of embedded metafiles.
- Code to export from **RichEdit** has been added to this manual.
- Barcodes printed by List&Label (tm) are reproduced correctly
- Mail Merge feature - see 'Properties'
- several other improvements to the PDF Engine to enhance compatibility even more.

wPDF Version 2.05 (30.3.2003)
- better positioning of underlines with rotated text
- support for strike-out fonts
- support for super and subscript when using the export from wptools
- now wptools can be used, too - previously only wptools version 4 worked with wPDF 2
- improved field rendering in wptools
- modification to the wPDF VCL part to optionally allow to use the PDF engine as pascal sourcecode (the optional source-code licenses is now available)

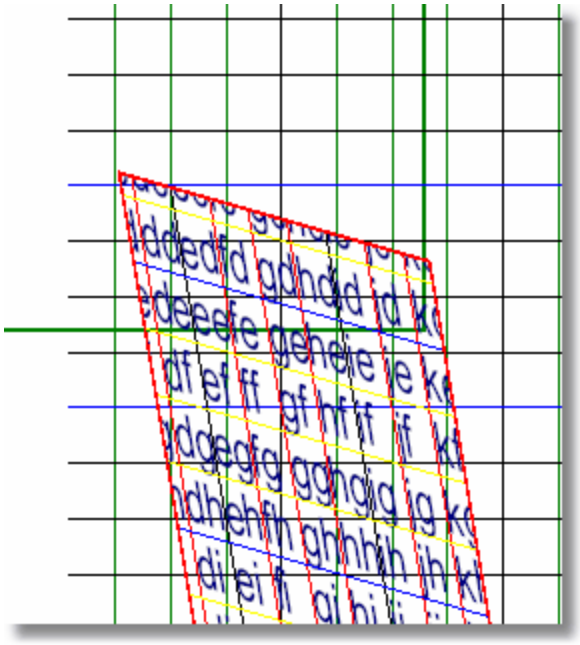
wPDF Version 2.02a (4.3.2003)
- problem with justified text has been solved
- font embedding was deactivated - this is fixed now
- mirrored and rotated coordinate system now works with text, too.
- clipping now also works in an upside down map mode.

wPDF Version 2.02 (28.2.2003)
- several problems with images have been solved (JPEG, 16 color)
- new mode 'wpAllowTransparentBit' to allow transparent painting of monochrome images
- new procedure DrawCCITT to export already compressed CCITT data (for example the output of fax software)
- Better support for underlines and strikeouts, especially when used with rotated text.
- When used with WPTools V4.09e or later metafiles which are embedded in the RTF text will be not any longer be converted to bitmaps prior to the export to PDF. This exports the metafiles without any loss of data or resolution. This option can by activating property 'ConvertMetafileToBitmaps'.
- When used with WPTools 4.09e of later the event OnPrintObject will be triggered for each embedded TWPObject. This makes it possible to draw alternative object data to the PDF "Canvas".
- Support for subscript and superscript when exporting from WPTools.

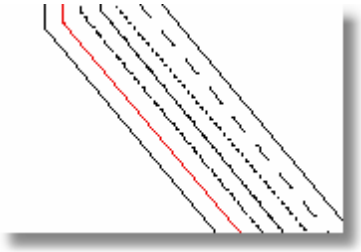
wPDF Version 2.01 (10.2.2003)

- Support for clipping rectangles and regions

This is a screenshot of part of a PDF page which was created using a rotated and scaled coordinate and a clipping rectangle. Please note that clipping has to be activated in the 'Modes' property.



- Better text rendering
- support for different pen styles



- fixed problems of the first release of wPDF V2.0

wPDF Version 2.00 (1.2.2003)

- Completely rewritten metafile to PDF conversion for best quality
- supports rotated coordinate systems
- support for paths (filling, stroking, clipping)
- better bezier curve rendering
- support for different map modes
- very accurate text rendering
- export to PDF is neutral to resolution - you always get the best possible quality.
- Support for font-subset embedding - used chars and charset.
- 128 bit Encryption

5 Credits / Intellectual Property

The architecture of this component is based on the "PDF Reference" document, third edition, published by Adobe. In this reference, page 6, Adobe gives copyright permission under the restriction that files are created which conform the Portable Document Format. In conformance with the reference we include the respective chapter here:

> The general idea of using an interchange format for electronic documents is in the public domain. Anyone is free to devise a set of unique data structures and operators that define an interchange format for electronic documents. However, Adobe Systems Incorporated owns the copyright for the particular data structures and operators and the written specification constituting the interchange format called the Portable Document Format. Thus, these elements of the Portable Document Format may not be copied without Adobe's permission.
> Adobe will enforce its copyright. Adobe's intention is to maintain the integrity of the Portable Document Format standard. This enables the public to distinguish between the Portable Document Format and other interchange formats for electronic documents. However, Adobe desires to promote the use of the Portable Document Format for information interchange among diverse products and applications. Accordingly, Adobe gives anyone copyright permission, subject to the conditions stated below, to:
>
> • Prepare files whose content conforms to the Portable Document Format
> • Write drivers and applications that produce output represented in the Portable Document Format
> • Write software that accepts input in the form of the Portable Document Format and displays, prints, or otherwise interprets the contents
> • Copy Adobe's copyrighted list of data structures and operators, as well as the example code and PostScript language function definitions in the written specification, to the extent necessary to use the Portable Document Format for the purposes above
>
> The conditions of such copyright permission are:
>
> • Software that accepts input in the form of the Portable Document Format must respect the access permissions specified in that document. Accessing the document in ways not permitted by the document's access permissions is a violation of the document author's copyright.
>
> • Anyone who uses the copyrighted list of data structures and operators, as stated above, must include an appropriate copyright notice.
>
> © 1985–2001 Adobe Systems Incorporated. All rights reserved.

The PDF Engine further uses the public zlib, the Independent JPEG Group's JPEG and RC4 routines.

It does not use the LZW algorithm.

RC4

Copyright (c) 2004 Hagen Reddmann

Copyright (c) 2001-2004, Michael Puff ["copyright holder(s)"] All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation

and/or other materials provided with the distribution.
3. The name(s) of the copyright holder(s) may not be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
DISCLAIMED. IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

SHA256:

(C) Copyright 2002-2008 Wolfgang Ehrhardt

This software is provided 'as-is', without any express or implied warranty. In no event will the authors be held liable for any damages arising from the use of this software.

Permission is granted to anyone to use this software for any purpose, including commercial applications, and to alter it and redistribute it freely, subject to the following restrictions:

1. The origin of this software must not be misrepresented; you must not claim that you wrote the original software. If you use this software in a product, an acknowledgment in the product documentation would be appreciated but is not required.

2. Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software.

3. This notice may not be removed or altered from any source distribution.

PNGIMAGE by Gustavo Huffenbacher Daud

The previous versions from this component were unclear about the license to use this component. Here it is:
1. This component should be distributed freely over the internet only when containing the exact same files from the original packaging.
2. Modified files may not be distributed. If you want to contribute with TPNGImage, send the enhancements to the author and if he implements your changes, you will be given the proper credit.
3. The component may be used in commercial projects but may NEVER be sold as source code.
4. Commercial graphics libraries are not allowed to use this component WITHOUT AUTHOR PRIOR AGREEMENT.
5. Credit for the author is required somewhere in the product documentation/or about box/etc.

6. Source code may be changed if it's not redistributed.
If are about to use the component in a major project which is going to be distributed over the internet, I'd love to know, so please send me an email telling me about.

Advanced Encryption Standard (AES),

Delphi implementation - License

The contents of this file are subject to the Mozilla Public License Version 1.1 (the "License"); you may not use this file except in compliance with the License.

You may obtain a copy of the License at http://www.mozilla.org/MPL/. Software distributed under the License is distributed on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the License for the specific language governing rights and limitations under the License.

The Initial Developer of the Original Code is Alexander Ionov.
All Rights Reserved.

Copyright (c) 2001, EldoS, Alexander Ionov

6 Installation

If you install WPTools and wPDF please do not compile the wPDF package.
Simply enabled the $define WPPDFEX in the WPTools file WPINC.INC and compile the WPTools package.

Otherwise **open the wPDF package file (*.dpk) from the folder Dx, add the directory Dx to the library path and compile and install the package file.**

The Sizes of the (uncompressed) DLLs are about 1100 KB for the demo version and about 480 KB for the registered version (32 bit).

To use the registered version you have to tell the engine your license name + code using the procedure WPDF_Start()

V3 and 4 Upgrade Notes:

wPDF Version 3 and 4 add 2 properties to the VCL interface. These properties are enabled by the $define wPDF5 which must be enabled in the file wpdf_inc.inc. Of course the new PDF engine (DLL or DCUs) must be used as well. Since version 3 is activated by just a single define it is possible and the replacement of the engine it is possible to use both, V2 and V3 at the same time. Simply add the compiler symbol wPDF5 to the projects which should use the new engine.
Please note that V2 also uses a new license code. That code contains the code for the previous version as well which internally strips the V3 part. (wPDF V2.75 or later)

Please note that wPDF Standard always requires the PDF-Engine DLL. That can be copied to the system directory - but it is a good idea to always copy it to the application directory. So you won't forget to distribute it with your application.
The distribution is not allowed if the application was build with the demo version of wPDF!

If You use Delphi 2009 or later you need to use the DLL wPDF500W.dll. You can then set the filename as unicode string. wPDF500W64.dll is for 64bit applications.

wPDF PLUS comes with Delphi DCU files to compile the PDF engine right into the application. Please open the file wpdf_inc.inc and disable the $define INTWPSWITCH. Please also add the path to the DCU files which were installed by the setup.

Please also note the extra files which can be added to the package to add additional functionality:

wppdfQR.pas - this is a PDF export filter for QuickReport

WPDF_FRep.pas - the export filter for FastReport

wppdfRBDev.pas - export device for Report Builder (tm)

wppdfRAVE.pas - rendering component for RAVE

6.1 Create Package for C++Builder

You need to select from Menu "File"
New/C++Builder Project/Package

Add the unit "WPPDFREG.pas"

Add the packages under "required":

"C:\Borland\XE2\lib\win32\release\vcl.bpi" "C:\Borland\XE2\lib\win32\release\vclx.bpi"

In case of XE2 also

C:\Borland\XE2\lib\win32\release\vclimg.bpi

Now it should be possible to compile and install the package.

7 WPViewPDF - a PDF view control

1. Fast PDF Viewer Control

WPViewPDF is a highly effective solution that offers a fast PDF Viewer Control for your application. This is an exceptional choice compared to other competing solutions available today. With its speed and performance, users can effortlessly view, print, and manipulate PDF files within your application.

2. Powerful PDF Manipulation

WPViewPDF PLUS offers powerful PDF manipulation capabilities. Users can not only view and print PDF files, they can also alter them in certain ways:

- delete pages
- change page order
- append multiple PDF files to a new one
- use a PDF file to be a watermark for another one
- add PDF annotations:
* highlight annotation
* text background (the used can select text and the annotation will cover the area)
* square annotation
* edit fields (Tx)
* symbols with Popups
* squiggly underline annotation
- move any existing annotation
- delete any existing annotation

3. Memory, File, Stream Data Loading

WPViewPDF allows for seamless handling of memory, file, and stream data loading in applications. In comparison to rival products, WPViewPDF boasts impressive speed while handling memory, files, and streams. Ultimately, this means less waiting time and improved productivity for users, making it a valuable addition to any application.

4. In-Place Field Editor

WPViewPDF offers and "In-Place Field Editor" that makes editing fields in PDF files easy and efficient.

5. Embed data and extract embedded data (i.e ZUGFerd Data)

6. Access PDF form (AcroForm) fields

8 QuickStart

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

BeginDoc - DrawMetafile - EndDoc.

If you need a canvas to draw to you need to open a 'page' first. Then you can do

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

There is only one property in the TWPDFPrinter which is required: FileName - this is the file which will be created.
If you want to create the **PDF in memory** use 'InMemoryMode' set to TRUE and assign the output **memorystream** to the property 'Stream'.

The property StartPage requires the width and the height of the page you need to create. It also requires the resolution which is used to define width and height and which is used for the x and y parameters of other procedures. (for bookmarks, hyperlinks etc). The last parameter is the rotation for the page - usually 0.

To create a 8.5 by 10 inch page use

```
var res := Screen.PixelsPerInch;
StartPage( Round(8.5*res), Round(10*res), res, res, 0);
```

to open a DinA4 page you can do this:

```
var res := Screen.PixelsPerInch;
StartPage( Round(21/2.54*res), Round(29.7/2.54*res), res, res, 0);
```

Don't forget to set your license name and code before using the PDF export. To do this use the procedure WPDF_Start(name,code).
(This does not apply to wPDF demo version or the Source-Code License)

See property DLLName if you want to install the DLL into a different directory than the applications folder or if you want to rename the DLL.

Upgrade note:

wPDF V3 and 4: Please note that for historic reasons the property Info.*Producer* modifies the "Creator" entry in the PDF file. The entry "Producer" will be always set to "wPDF5 by WPCubed GmbH".

8.1 Examples

Please also see chapter "Linking with other products".

A) DrawMetafile

```
WPPDFPrinter1.BeginDoc;
try
```

```
     WPPDFPrinter1.DrawMetafile(
0,0,Image1.Picture.Metafile.Handle);
     WPPDFPrinter1.DrawMetafile(
0,0,Image2.Picture.Metafile.Handle);
     WPPDFPrinter1.DrawMetafile(
0,0,Image3.Picture.Metafile.Handle);
     WPPDFPrinter1.DrawMetafile(
0,0,Image4.Picture.Metafile.Handle);
   finally
     WPPDFPrinter1.EndDoc;
   end;
```

B) Canvas

```
  WPPDFPrinter1.BeginDoc;
  WPPDFPrinter1.StartPage( w,h, res, res, 0);
  SaveDC(WPPDFPrinter1.Canvas.Handle);
  WPPDFPrinter1.Canvas.StretchDraw(
      Rect(0,0,w , h ), Image1.Picture.Graphic );
  RestoreDC(WPPDFPrinter1.Canvas.Handle,-1);
  WPPDFPrinter1.Canvas.Font.Color := clBlue;
  WPPDFPrinter1.Canvas.Font.Name := 'Arial';
  WPPDFPrinter1.Canvas.Font.Height := -13;
  WPPDFPrinter1.Canvas.TextOut(10,h-22,'Example');
  WPPDFPrinter1.EndPage;
  WPPDFPrinter1.EndDoc;
```

C) WPTools

```
   WPPDFExport1.Source := WPRichText1;
   WPPDFExport1.Print;
```

Note: If you need to export multiple RTF texts into the same PDF file use BeginDoc/EndDoc.

Please continue to read here...

D) Export from RichView

```
  WPPDFPrinter1.BeginDoc;
  try
    for I := PageF to PageT do
    begin
      WPPDFPrinter1.StartPage(
         Round(RichView1.PageProperty.PageWidth),
         Round(RichView1.PageProperty.PageHeight),
         Res, Res, 0);

      RichView1.DrawPage(I,
         Round(RichView1.PageProperty.PageWidth),
         Round(RichView1.PageProperty.PageHeight),
        0, 0, WPPDFPrinter1.Canvas, False, False, False);
      pdf.EndPage;
    end;
```

```
  finally
    WPPDFPrinter1.EndDoc;
  end;
```

E) WPForm

```
var meta : TMetafile;
    i : Integer;
begin
   WPPDFPrinter1.BeginDoc;
   meta := nil;
   for i:=1 to WPFormEditor1.PageCount do
   try
     meta :=  WPFormEditor1.GetMetafile(i);
     WPPDFPrinter1.DrawMetafileEx(0,0,0,0,meta.handle,
            Screen.PixelsPerInch,Screen.PixelsPerInch);
   finally
     meta.Free;
   end;
   WPPDFPrinter1.EndDoc;
end;
```

8.2 C++ Builder Notes

You can use wPDF within C++Builder very similar to the usage in Delphi. The only difference is the way procedures and properties are addressed. In C++Builder you use '->' instead of '.'. Methods which do not require any parameters require at least a empty pair of parentheses ().

Example:

Create a PDF file from a TImage with a loaded metafile:

```
WPPDFPrinter1->AutoLaunch = true;
WPPDFPrinter1->Filename = Edit1->Text;
WPPDFPrinter1->BeginDoc();
WPPDFPrinter1->DrawTGraphic(0,0,0,0,Image1->Picture->Graphic);
WPPDFPrinter1->EndDoc();
```

Execute the property dialog:

```
WPPDFProperties1->Execute();
```

Troubleshooting

You can create a new wPDF package and add the files

```
WPPDFR1.pas
WPPDFR2.pas
WPPDFREG.pas
```

This creates a new package which works optimal on your system with your paths

In case units are not found please set this unit aliases in the project option (under Delphi)
```
Winapi;System.Win;Data.Win;Datasnap.Win;Web.Win;Soap.Win;Xml.Win;Bde;Sys
tem;Xml;Data;Datasnap;Web;Soap;Vcl;Vcl.Imaging;Vcl.Touch;Vcl.Samples;Vcl
.Shell
```

If your C++Builder project compiles but does not start
```
- deactivate "dynamic RTL"
- deactivate runtime packages
```

9 Properties

9.1 CidFontMode

This property enables the support for text written used by **C**haracter **Id**entifiers. 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 charset to be known to be properly exported.

CidFontMode Values:

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

The Cid-Font support requires Windows NT, 2000, XP, Vista or later

wpCIDUnicode can also be selected by setting the FontMode to: **wpEmbedCIDFonts**

9.2 PDFAMode

This property enables PDF/A support.

Values: wpdfaOff, wpdfaLevelA or wpdfaLevelB

PDF/A is a new norm which based on PDF 1.4 - it was created to provide a guideline for the creation of document files which stay readable for the time to come. So the major demand for PDF/A compliant files is that the used font files are embedded. Security measures are forbidden in PDF/A compliant 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 WPTools 5 or WPTools 6 with wPDF to create the PDF files 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.

To activate the PDF layout tags when exporting from WPTools increment the variable Memo._WPDFParRun:

```
inc(WPRichText1.Memo._WPDFParRun);
WPPDFExport1.PDFAMode := wpdfaLevelA;
WPPDFExport1.Print;
```

wPDF5 will also add the document information as XMP data to the PDF file.

wpdfaLevelB differs only little from wpdfaLevelA:
- not's add ToUnicode CMAP information
- don't force PDF tagging

Please note:
According to ISO 19005 - Chapter 6.3.8 Fonts require character mapping information. Such a table is always added, for CidFonts and also if the encoding of a font is "Difference". So the requirement 6.3.8 is met, although some PDFA validators seem to ignore the existence of the date referenced by the *ToUnicode* tag. For PDF/A we recommend the CIDFont font embedding.

When PDF/A has been selected a sRGB color profile will be added to the document.

The use of form fields (edits, check box fields) in a PDF file which should be PDF/A conform is not possible with wPDF.

9.3 Output Properties (Filename/Stream)

propery FileName

This is the filename of the PDF file which should be created. To be able to create PDF either filename or stream must be defined.

property AutoLaunch

Set this property to true if you want to execute the PDF file once it was created.
Please read the Acrobat(tm) "readme" text if Acrobat Reader 5 does not automatically start.
Normally you have to disable the plugins to allow the execution while a debugger (such as the Delphi or C++Builder IDE!) is active.

property InMemoryMode

If this property is true the PDF file will be written when it was fully created. This is used with the property Stream which must be assigned the latest when 'EndDoc' is executed. The stream will then receive the data which was just created.

property Stream
You set a stream as destination for the created PDF file. **This makes it possible to create PDF files in memory without having to use a file**. Either filename of stream must be defined.

The following properties should not be used anymore.
Please use the product WPViewPDF PLUS if you need to merge PDF.

property InputFile

property InputfileMode

9.4 Modify Output

Property CanvasReference

wPDF provides a Canvas property for easy creation of PDF files. The canvas property is internally managed as a TMetafileCanvas which is created for a metafile which is sent to the PDF engine when EndPage is executed.

Using the property CanvasReference you select if the canvas should use the properties of the screen (wprefScreen) device or the default printer (wprefPrinter). Using the printer requires of course that a printer is installed, but has the advantage that the resolution is independent from the size of the monitor.

In wPDF V2/V3 the resolution used in 'StartPage' will be automatically applied to the Canvas using the SetViewPortAPI.

If the screen device is used as reference Windows 1.) uses the dimension of the screen as maximum clipping rectangle (WinNT and Win2K) and 2.) changes the resolution to the pyhsical size of the monitor! Because of that we recommend to use the printer as reference.

property Modes

This property 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.

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

wpWhiteBrushIsTransparent: 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.

wpExactTextPositioning: 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)

wpNoTextRectClipping: 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.

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

wpDontStackWorldModifications: 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.

wpNoTextRectClipping: 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.

wpDontAdjustTextSpacing: 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.

wpDontCropBitmaps: 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.

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

wpAllowTransparentBit: 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.

wpAlwaysHighResPDF, wpNeverHighResPDF:

PDF does only allow coordinates up to 32000. Our PDF engine tries to preserve the original windows coordinates in the PDF file. Unfortunately those coordinates can become larger than 32000. This is especially problematic for Acrobat 5 and before.

Usually the PDFEngine will try to avoid such large coordinates and then activate internal division by 10.

Using this flags this mode can be forced or disabled.

wpUseFontWidthArgument: Ignore Font widths defined by the CreateFont API call.

wpNoTextScaling: Never enlarge fonts to match width reported by windows.

wpMetaIsDOTNETEMF: internally used only

wpInputIsWMFData: internally used only

wpHatchBrushIsSolid: switch off the new hatching

wpTestIfWeHaveCID: used with the new CidFontMode. When characters are missing from selected font print ?

wpTextAsGlyphs: Export the entire text using curves

wpDisableAutoGlyphs: Disable the new code which draws characters which were not exported otherwise (missing from current code page, missing from selected font) as curves

wpUseOldFontSubsetting: Use the old font subsetting code. This old code produces slightly smaller embedded TTF data but also stripped out too much information from the TTF program.

wpSimulateBoldFonts: Instead of selecting a bold font the bold mode is simulated

wpConvertColorsToCMYK: Convert all font, brush and pen colors to CMYK

wpWriteToUnicodeCMAP: Some PDF/A validators require cmaps with fonts files. Unfortunately the tested build of Acrobat produced an error with this extra info.

wpDetailedGradients: if set, gradients will be reproduced more detailed, but PDF files will be bigger

wpOutlineNumberDetection: keep word distances when the text looks like numbered or bulleted text

wpNoAutoTagsForPDFA: do not add a tag at the beginning of each page when PDF/A is selected

wpArabicAlwaysAsOutline: export arabic as outline to make sure the glyph combination is prefect

wpHebrewAlwaysAsOutline: export hebrew as outline

wpNoOutlinesForASIANInPDFA: Don't select outlines for asian text automatically when PDF/A is activated

wpDontSimulateItalicFonts: Disable automatic simulation of italic fonts when the font does not support this mode

wpPreserveBookmarkNames: Create named destinations instead of fixed hyperlink jumps to certain bookmarks.

property ModeEx (trouble shooting options)

wpDontRecreateClipLevel don't recreate clipping after automatic qQ

wpWriteNeedAppearances write the "NeedAppearances" flag to the PDF file

wpProhibitStretchDIBitsWithMask Do not draw StretchDIBits with Alpha transparent

wpDontSearchForImageDuplicates Disable automatic image reuse. Usually wPDF tries to minimize the file size by reusing images.

wpDontDetectHorizontalDistortion If true distorted StretchDraw is not detected. This can be useful with WMFs

wpPrintTextWordByWord Useful setting if the GDI textout use character distances for tabs. (Usually tabs are separated in individual text outs)

wpThaiAlwaysAsOutline Thai language will always use Type3 Fonts

wpMoveTopAlignedTextToBaseline Optionally calculate the baseline alignemnt (off)

wpDisableRotationAlignCorrection Disable the text alignment correction for rotated worlds

wpRenderTextInPathAsGlyph If text is exported inside BeginPath/EndPath it is rendered as outlines

wpDontWriteOutlineCountProperty Do not write the Count property in PDF outlines

wpConvertColorsToGray Convert all colors to gray

wpUseNegativeWndExt Detect negative scaling and modify baseline calculation algorithm (this flag should be on)

wpSavePenstyleBeforeCliprect Fixes problems with certain EMF which do not use SaveDC/RestoreDC around clip rectangles

9.5 Compression

Property CompressStreamMethod

By modifying this property you can let the PDF engine compress (deflate) text. By using compression the file will be reasonable smaller. On the other had compression will create binary data rather than ASCII data. While "deflate" produces the smallest files, "run-length" compression is compatible even to very old PDF reader programs.

Property JPEGQuality

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 EncodeStreamMethod

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 ConvertJPEGData

Note: Only applies to TWPDFExport.
If this property is true JPEG data found in the TWPRichText editor will not be embedded as JPEG data. Instead the bitmap will be compressed using deflate or run length compression. It is necessary to set this property to TRUE if the PDF files must be compatible to older PDF reader programs which are incapable to read JPEG data.

9.6 Encryption

Property Encryption

wpEncryptFile - to switch on encryption
wpEnablePrinting - allow the user to print the PDF file
wpEnableChanging - allow to change the PDF file
wpEnableCopying - allow to copy text from the PDF file
wpEnableForms - enable interactive elements (do not yet apply)

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 wPDF supports the 40-bit and 128 bit PDF encryption. The later also causes the PDF to use PDF version 1.4.

property Security (wpp40bit, wpp128bit)
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.

9.7 Text Rendering

property Modes

If flag wpExactTextPositioning is used the characters will be placed at the same positions as they were in the input metafile. Usually this is not desirable since the PDF reader is able to render in a higher quality if it may use its intern logic to calculate the character spacing.

If flag wpNoTextRectClipping is not used text pained with TextRect will be not clipped to the bounding box.

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.

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

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

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

wpUseBase14Type1Fonts : Do not use true type. When this mode is selected you can only use Arial, Courier New and Times New Roman fonts.

Subset Embedding: to *reduce the size of the embedded data* by removing the description for unused characters.

There are 3 options which do this for you:

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

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

wpEmbedCIDFonts - activate the CID Font mode for unicode text support and PDF/A

Type 3 fonts are made new for each used character (only) from the glyph of the character provided by Windows

wpEmbedType3 - Build Type3 Subset Fonts. You can also select Type3 fonts for certain fonts using the string list Type3Fonts

Property ExcludedFonts
You may specify the font's in this stringlist which you expect to be installed on system the PDF will be viewed. Even if you use FontMode=EmbedTrueType fonts which are in this list will not be embedded. Please note that symbol fonts should be embedded since they can't be substituted.

Property ExtraMessages
Switches on/off additional messages:
wpOnEmbedFonts : Create a message when font-data is embedded.
Messages are provided in the OnError event.

Property HeaderFooterColor
Note: Only applies to TWPDFExport for WPTools export
If this property is not clNone the headers and footers will be drawn in this color. You can select clGray to draw headers and footers in the same way as they are displayed in the layout view of WPTools.

Property: PreselectedCJK
This is the charset for text written using CYK unicodes. It can be Japanese, Chinese or Korean. This property is only used for text which was not written using a specific charset.

9.8 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 Info

This property contains several sub properties, 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.

wPDF V3: Please note that for historic reasons the property Info.*Producer* modifies the "Creator" entry in the PDF file. The entry "Producer" will be always set to "wPDF5 by WPCubed GmbH".

Property CreateThumbnails
If this property is TRUE 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 Options

wpCreateAutoLinks - 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(10,20,'https://www.wpcubed.com');
```

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 SetLinkArea procedure.

These flags set the "ViewerPreference" for the PDF file

wpHideToolbar,
wpHideMenuBar,
wpHideWindowUI
wpScalingNone
wpFitWindow
wpCenterWindow
wpDisplayDocTitle,
wpPickTrayByPDFSize

These flags change the way text is exported

wpDisableAutoBoldSimulation
wpAllSymbolFontsAsGlyph

wpDecorateAllFontnames Append "-Bold" etc to all font names. Otherwise this is only appended to "common" names.

9.9 Mail Merge

Since wPDF V2.10 the PDF Engine is able to do "mail-merge", this means it can replace fields found in the input data with text provided by your application.

Such fields are defined by a certain start text, for example '@@' followed by any number of characters which are the field name.

The start text is defined with the property **'MergeStart'** - if it is empty this feature is disabled.

Before the engine renders a text which starts with the defined start text it executes the event **OnMergeText**. In this event you can modify the provided string which is passed as 'var' parameter:

This feature can be very useful to enter data into the PDF file which is different on each run of the export for example the current date or time. It is especially useful if you are combing the output of different report engines (RAVE, ReportBuilder) to the same PDF file. In this case each report engine would use its own page counting. If you, instead of using the standard page number fields use a mail-merge field you can enter the page number calculated by the PDF engine:

```
procedure TForm1.WPPDFPrinter1MergeText(Sender: TObject; var Text: String);
begin
   Text := 'Page ' + IntToStr( WPPDFPrinter1.PageNumber );
end;
```

Please note that you can probably use the text alignment flags for the text object to align the text properly.

9.10 DLLName

This property specifies the path to the PDF engine. wPDF Version 2 used the file wPDF200A.dll, wPDF Version 3 uses the file wPDF300A.dll and wPDF300**W**.dll. and wPDF Version 4 uses wPDF500A.dll and wPDF500**W**.dll.

64bit Applications use wPDF500W64.dll.

The path may be an absolute path or you may reference a registry entry using the tokens {hkcu} or {hklm}.

In the demo applications we use the code

pdf.DLLName := '{hkcu}Software\WPCubed\wPDF\4.**0**\path';

this lets the VCL load the location from the registry entry which was created by our wPDF setup script.

If you are using inno setup you can use code similar to this line to set the location.

```
Root: HKCU; Subkey: Software\MyCompany\wPDF; ValueType: string; ValueName: path; ValueData:
{app}\wPDF500a.dll; Flags: uninsdeletekey
```

and for the DLLName property use `'{hkcu}Software\MyCompany\wPDF\path';`

Alternative you can specify the path relatively to your application using ExtractFilePath(Application.EXEName);

Important:
If the DLL name ends with a "W", it is expected to be the unicode version of the PDF engine. The 64bit version is always expected to be unicode enabled.

9.11 FontMode

Selects the font embedding modes:

wpUseTrueTypeFonts - use TTF but do not embed
wpEmbedTrueTypeFonts - use TTF and also embed
wpEmbedSymbolTrueTypeFonts - use TTF and also embed symbol fonts
wpUseBase14Type1Fonts - map TTF fonts to predefined fonts Helvetica, Times and Courier
wpEmbedSubsetTrueType_Charsets - embed only subsets of TTF fonts
wpEmbedSubsetTrueType_UsedChar - embed only subsets of TTF fonts
wpEmbedType3 - create new fonts from the outlines of the drawn characters - best for asian fonts
wpEmbedCIDFonts - embed CID subsets - this does the same as CidFontMode = wpCIDUnicode.

It is also possible to only embed certain fonts as Type3 fonts:

The font names must be listed in the stringlist Type3Fonts. All other fonts will be handled according to property FontMode.

9.12 Info

This properties contains several sub fields with information to be added to the information record inside the PDF.

Please make sure to populate this fields before BeginDoc is executed.

property Author: string
property Producer: string - this modifies the "Creator" in the PDF
property Title: string
property Subject: string
property Keywords: string
property Strings: TStrings

property CopyrightNotice : String - select the PDF to be copyrighted
property CopyrightURL : String - select the PDF to be copyrighted
property CopyrightPublicDomain : Boolean - select the PDF to be copyrighted, but under public domain

10 Methods

10.1 Start/End Output

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

BeginDoc - DrawMetafile - EndDoc.

If you need a **canvas** to draw to, you need to open a 'page' first. To open a PDF page use StartPage.

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

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

If you have assigned 'Canvas' to a local variable you need to assign it again after 'CloseCanvas'.

To create a PDF stream set the property Stream to the instance of a Stream object.

10.1.1 BeginDoc

This procedure initializes the PDF export.

If you are using the WPPDFExport component it is not necessary to use this procedure unless you need to execute 'Print' several times to export into one PDF file.

```
WPPDFExport1.PageCount := WPPDFExport1.Source.CountPages*10;
WPPDFExport1.BeginDoc;
for i:=1 to 10 do
   WPPDFExport1.Print;
WPPDFExport1.EndDoc;
WPPDFExport1.PageCount := 0;
```

You can check if a PDFFrinter has already started a file by checking the property 'Printing'. In this case you should not call BeginDoc and EndDoc if you want to create a combined PDF file.

To implement a "Start/Stop" button you can use this code:

```
procedure TForm1.StartStopPDFClick(Sender: TObject);
begin
  WPPDFPrinter1.Printing := not WPPDFPrinter1.Printing;
end;
```

Please note that if you are rendering different reports into the same PDF file the filename used by the different report engines (ReportBuilder, RAVE ...) is ignored.

10.1.2 EndDoc

This procedure finalizes the PDF export whic was started with BeginDoc.
It closes the PDF file or -stream.

If you set the property MemoryMode = TRUE then you can now extract the PDF stream by using the property Stream. Otherwise the PDF file saved under the name 'filename' can be used or it will be automatically displayed if the property AutoLaunch was set to TRUE.

10.1.3 StartPage

Creates an new page in the PDF file. The Canvas property is not valid unless you execute StartPage or StartWatermark.

Please note that there is no need to execute StartPage if you only export an enhanced metafile (**DrawMetafile**). In this case the engine will open a page and close it after the data has been exported.

The parameters of StartPage are the size of the page (width and height) and the resolution the values are measured in. If you use twips values set the resolution to 1440, if you are using the Width/Height of the current form use Screen.PixelsPerInch. The Resolution the PDF engine will use will be changed accordingly!

The parameter options can be used to rotate the page. Possible values are 0, 90, 180 and 270.

NOTE: The resolution parameter does per default not change the resolution of the PDFPrinter.Canvas. But it changes the resolution of the PDF file. The default resolution of PDF files is 72 which is a good value for text. wPDF V2 also changes the resolution of the 'Canvas' using the SetViePortAPI.

You can use this formula to set the width and height accordingly to the values you are using in screen coordinates:

```
StartPage(
        MulDiv(Width,72,Screen.PixelsPerInch),
        MulDiv(Height,72,Screen.PixelsPerInch),72,72,0);
```

To set a certain resolution (res) in the canvas property use this code:

```
DC := WPDFPrinter1.Canvas.Handle;
curr_resX := GetDeviceCaps(DC,LOGPIXELSX);
curr_resY := GetDeviceCaps(DC,LOGPIXELSY);
SetMapMode(DC, MM_ANISOTROPIC);
SetWindowExtEx(DC, res, res, nil);
SetViewPortExtEx(DC, curr_resX, curr_resY, nil);
SetViewPortOrgEx(DC, 0, 0, nil);
```

Note: If you only want to export a metafile you don't need to execute StartPage. The procedure DrawMetafile will automatically create a PDF with the correct dimensions.

10.1.4 EndPage

This procedure closes a PDF page. It must be executed if a page was opened with StartPage.

10.1.5 StartWatermark

This procedure initializes the engine to create a new water mark - it works similar to StartPage but you have to provide a name.

The name can be used in **UseWatermark**.

10.1.6 EndWatermark

This procedure closes a watermark which was started with StartWatermark.

10.2 Graphic Rendering

Using wPDF you can create PDF files by exporting bitmaps, metafiles or using the property 'Canvas' which provides you with a compatible TCanvas object. This canvas object can be used similar to the Canvas provided by a TPaintBox, a TPrinter or similar. It's 'handle' can be used with many Windows GDI commands, such as IntersectClipRect, SetWorldTransform or SetViewPort.

Usually you have to use StartPage to make the export possible. Otherwise the Canvas is not valid and DrawBitmap is not possible. An important exception of this rule is DrawMetafile which can automatically create and close a page which exactly matches the dimensions of the exported metafile.

Notes:

1) All coordinate values used by the following procedures use the x,y resolution which was selected in StartPage()!

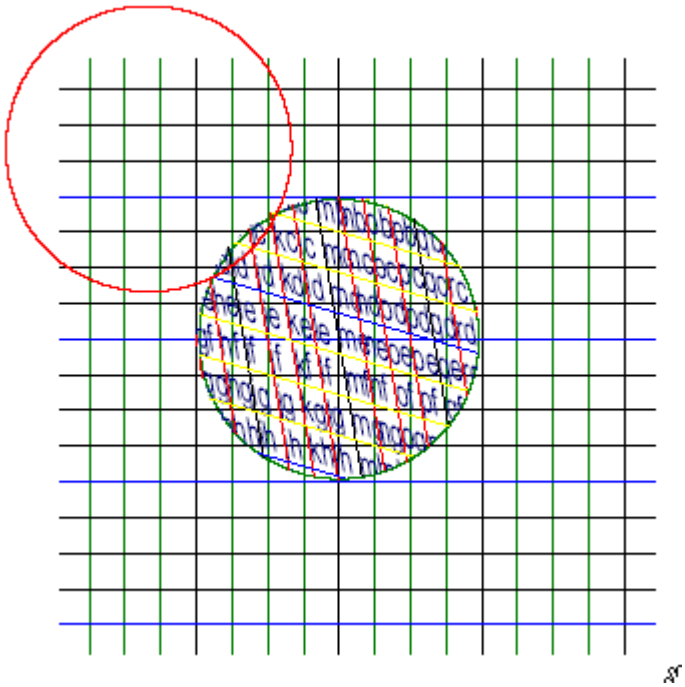
2) DrawBitmap and other "Draw" functions export the data right away to PDF - unlike the API functions used with the 'Canvas' object, i.w. Canvas.StretchDraw. Here all collected graphic operations are exported when (a) the page is closed or (b) the API **CloseCanvas** is executed. So you need to use the method CloseCanvas if you mix Draw... and Canvas functions.

10.2.1 property Canvas

This property is valid if you used StartPage to open a new PDF page. If you want to close the 'Canvas' to for example draw a bitmap using DrawBitmap execute the procedure *CloseCanvas*. It will flush the stored graphics commands to the PDF Engine.

The property Canvas provides access to a TCanvas object you may use with almost any drawing code. You can use the Canvas procedures and windows API commands such as SetWorldTransform(Canvas.Handle,..).

Example for a graphic which was drawn to the Canvas and exported to PDF. It creates a elliptical clipping region, then rotates the pages using SetWorldTransform and prints a grid with letters.



Currently you install a clipping region when the canvas has been rotated. Please create the clipping region before calling SetWorldTransform or use paths for clipping.
To enable support for clipping please activate wpClipRectSupport in the 'Modes'.

10.2.2 property CanvasReference

wPDF provides a Canvas property for easy creation of PDF files. The canvas property is internally managed as a TMetafileCanvas which is created for a metafile which is sent to the PDF engine when EndPage or CloseCanvas is executed.

Using the property CanvasReference you select if the canvas should use the properties of the screen (wprefScreen) device or the default printer (wprefPrinter). Using the printer requires of course that a printer is installed, but has the advantage that the resolution is independent from the size of the monitor.

Note: If the screen device is used as reference Windows 1.) uses the dimension of the screen as maximum clipping rectangle (WindNT and Win2K)

10.2.3 method DrawBitmap

This function exports a bitmap to the PDF file. You can specify the desired size of the bitmap (measured in PDF resolution, see StartPage). Please provide a valid HBITMAP handle to this function, for example TBitmap.Handle.

The function returns the id of the bitmap in the PDF file. If you want to export this bitmap again you can use the function DrawBitmapCloneI to do so. Important: Please note that the bitmap has to be in either in RGB (24 bit) or in monochrome format!

10.2.4 method DrawBitmapClone

This function exports a bitmap to the PDF file. You can specify the desired size of the bitmap (measured in PDF resolution, see StartPage). Please provide a valid bitmap ID to this function. Such an ID is provide by the functions DrawBitmap, DrawJPEG, DrawTGraphic and DrawDIBBitmap. The return value of this functions is the ID which was passed to it.

10.2.5 Method DrawDIBBitmap

This function exports a bitmap to the PDF file. You can specify the desired size of the bitmap (measured in PDF resolution, see StartPage). The function returns the id of the bitmap in the PDF file. If you want to export this bitmap again you can use the function DrawBitmapClone to do so.

This function renders expects the parameters which describe a DIB bitmap, consisting of the INFO and the BITS parameter.

10.2.6 Method DrawMetafile

You can use this procedure to draw a metafile which was created by a component such as WPForm or QuickReport. The x and y parameters are the position on the PDF page. 'Meta' is the handle of the metafile, provided by TMetafile.Handle.
If you have created the metafile using the screen as reference please provide the x and y resolution in the last 2 parameters of procedure DrawMetafileEx. (all parameters can be set to 0 except for 'handle')

Now you can also use DrawMetafileEx if you want to also provide a width and height parameter.

```
procedure TForm1.WPPDFExport1BeforePrintPage(Sender: TObject; Number, FromPos, Length:
Integer);
var
  TheMetafile: HENHMETAFILE;
begin
  TheMetafile := GetEnhMetaFile('c:\test.emf');
  WPPDFExport1.DrawMetafile(0, 0, TheMetafile);
  DeleteEnhMetaFile(TheMetafile);
end;
```

10.2.7 Method DrawMetafileEx

You can use this procedure to draw a metafile which was created by a component such as WPForm or QuickReport. 'Meta' is the handle of the metafile, provided by TMetafile.Handle, x,y,w,h is the position and size measured in PDF pixel (see StartPage). The last two parameters are the resolution of the reference canvas used to create this metafile. Please provide it if you know it, otherwise set the parameter to 0.

If you have to draw multiple metafiles better use Canvas.StretchDraw( ) to draw the metafiles.

Note: All functions which draw **metafiles** do not need an open PDF page. If no PDF page is open there will be one created and automatically closed.

10.2.8 Method DrawTGraphic

You can render Metafile or Bitmap TGraphic objects using this function. X,y,w,h is the position and size measured in PDF pixel (see StartPage).

10.2.9 Method DrawGraphicFile

This draws a graphic files, it can be a metafile , bitmap or jpeg file.

10.2.10 Method DrawJPEG

function DrawJPEG(
x, y, w, h,
SourceW, SourceH: Integer; Buffer: PChar; BufLen: Integer): Integer;

Using this method you can export compressed JPEG data. Please provide the correct values for the parameters SourceW and SourceH. This is the native width and height of the JPEG image.

Also see DrawJPG and DrawPNG.

10.2.11 Method DrawCCITT

function DrawCCITT(x, y, w, h, SourceColumns, SourceRows: Integer; mode : TWPCCITTMode; Buffer: PChar; BufLen: Integer): Integer;

Using this method you can export compressed CCITT data.

Please provide the correct values for the parameters SourceColumns and SourceRows.

This is the native width and height of the CCITT image.

Using mode you can select how the data is compressed:
wpCCITT_31D, wpCCITT_32D, wpCCITT_42D

Please note that this command expects already compressed data without any header information. So it is **not** possible to simply pass a TIF file.

10.2.12 PrintForm

This method can be used to export a certain form as screenshot in a PDF file.

Parameters:
Form: TForm - this is the form to be exported

screenshotmode: Boolean - if this parameter is false only the client area of the form will be exported and text remains selectable, if it is true, also the title bar of the form will be captured but the exported information will be only a bitmap.

The form must be completely visible.

10.2.13 DrawPNGFile

```
function DrawPNGFile(x, y, w, h : Integer;
  PNGImageFile : String;
  OnCanvasMode : Boolean = false): Integer;
```

This methods exports a PNG file. It requires the PDF Engine to be able to decode PNG. Optionally also a JPEG file can be exported since it is detected by the PDF engine.

If OnCanvasMode=true the image will be exported in the current working state of the canvas. In that case the result value is 0, it is not possible to print duplicated images.

```
Options is currently not used.
```

Example: draw PNG file and also a copy

```
i := pdf.DrawPNGFile( 100, 100, 400, 400,
     'transparent.png',
     0, true
   );
pdf.DrawBitmapClone(100,100, 300,300, i);
```

10.2.14 DrawPNG

```
function DrawPNG(x, y, w, h : Integer;
  PNGImage : Pointer;
  PNGImageSize : Integer;
  OnCanvasMode : Boolean = false): Integer;
```

This method exports PNG data. It requires the PDF Engine to be able to decode PNG.

If OnCanvasMode=true the image will be exported in the current working state of the canvas. In that case the result value is 0, it is not possible to print duplicated images.

Example: draw PNG data in canvas context

```
mem := TMemoryStream.Create;
mem.LoadFromFile('transparent.png');

pdf.DrawPNG(100, 100, 400, 400, mem.Memory, mem.Size, true);
mem.Free;
```

10.2.15 DrawJPG

```
function DrawJPG(x, y, w, h : Integer;
   JPGImage : Pointer;
   JPGImageSize : Integer;
   OnCanvasMode : Boolean = false): Integer;
```

This method exports JPEG data. It requires the PDF Engine to be able to decode JPEG.

If OnCanvasMode=true the image will be exported in the current working state of the canvas. In that case the result value is 0, it is not possible to print duplicated images.

10.3 Links and Bookmarks

The following methods are used to creates hyperlinks, bookmarks and outlines in the PDF file. All coordinate values use the x,y resolution which was selected in StartPage()!

10.3.1 Method SetBookmark

Creates a bookmark entry in the PDF file.

Parameters:
const BookMarkName : string This is the name of the book mark. The book mark can be used by SetLinkArea or SetOutline.
X,Y : Integer This is the x and y position on the current page which should be located.

Example:

```
WPPDFPrinter1.SetLinkArea(TLabel(c).Hint, c.BoundsRect);
WPPDFPrinter1.Canvas.Font.Style :=
  WPPDFPrinter1.Canvas.Font.Style + [fsUnderline];
WPPDFPrinter1.Canvas.Font.Color := clBlue;
```

10.3.2 Method SetLinkArea

This procedure defines a hotlink in the PDF file. The X and Y values are the expected as twips coordinates relative to the top-left corner of the current page. (You have to add the top margin and left margin)

Parameters:

const BookMarkName : string
This is the name of the book mark which should be located when the user clicks on the hotspot.

If the name starts with **http://** which is a web link, if it starts with **Launch://** it starts a file.

R : TRect
This is the rectangle position on the current page which should be marked as hotlink. Please note that the WPTools export engine TWPDFExport already automatically deals with hyperlinks. (A hyperlink is defined as text with the style afsHyperlink in wptools) followed by some invisible text which is the book mark name.

Example
a) Create a link which opens the file 'c:\test.htm':

```
WPPDFPrinter1.SetLinkArea('Launch://c:\test.htm',
      Rect(10,10,100,100));
```

b) Create a link which opens the file 'c:\test.pdf':

```
WPPDFPrinter1.SetLinkArea('GoToR://c:\test.pdf',
      Rect(10,10,100,100));
```

use this string parameter to open the PDF at page #3:

```
'GoToR://c:\test.pdf#2' // First page = 0
```

and this to open the PDF at named destination "ORANGE" :

```
'GoToR://c:\test.pdf#ORANGE'
```

c) Create link areas for whole lines which have a colored background. (Can be modified to create links in a table of contents)

```
procedure     TForm1.WPPDFExport1AfterPrintPage(Sender: TObject;
Number,
  FromPos, Length: Integer);
var
  r : TRect;
  toppos : Integer;
begin
  WPPDFExport1.Source.CPPosition := FromPos;
  toppos := WPPDFExport1.Source.Memo.active_line^.y_start;
  while Length>0 do
```

```
  begin
     // Only if       at start of a line process the following
code
     if WPPDFExport1.Source.CPColNr =0 then
     begin
        // we check if paragraph is colored. We     could also
        // check the paragraph id or any other property.
        if WPPDFExport1.Source.Memo.active_paragraph^.color<>0
then
        begin
          // Calculate the rectangle for the current line
          WPPDFExport1.Source.GetLinRect(
                      WPPDFExport1.Source.Memo.active_paragraph,
            WPPDFExport1.Source.Memo.active_line, r);

                // Move the rectange according to the defined
margins
          dec(r.Top, toppos-WPPDFExport1.Source.Header.TopMargin);
          dec(r.Bottom,
                     toppos-WPPDFExport1.Source.Header.TopMargin);
          inc(r.Left, WPPDFExport1.Source.Header.LeftMargin);
          inc(r.Right, WPPDFExport1.Source.Header.LeftMargin);
          WPPDFExport1.SetLinkArea('LINK',r);
        end;
     end;
     Dec(Length);
     if not WPPDFExport1.Source.CPMoveNext then break;
  end;
end;
```

10.3.3 Method SetOutlineXY

This procedure creates an outline entry. It returns the id number of the just created outline entry. This id number can be used to create the next entry as the child of the current.
Please note that you can use the procedure **SetOutlineCharset** to define a certain charset for the outline. You can use the value 0 to select the default charset or one of the following constants: EASTEUROPE_CHARSET, RUSSIAN_CHARSET, GREEK_CHARSET , TURKISH_CHARSET and BALTIC_CHARSET.

Parameters:

const Caption : string
This is the text which should be displayed in the outline tree.

X,Y : Integer
This is the x and y position on the current page which should be located when the user clicks on the outline entry. They are expected as page coordinates relative to the top-left corner of the current page. So it uses the units defined in StartPage().

aParent : Integer
If this variable is greater 0 the outline entry will be created as the child of the entry with the id provided in aParent.

aPrevious : Integer
If this variable is greater 0 the outline entry will be created after the entry with give id.

Note: If aParent and aPrevious are 0 the outline will be created at the top level.

10.3.4 Method SetOutline

Method SetOutline

This procedure creates an outline entry. It returns the id number of the just created outline entry. This id number can be used to create the next entry as the child of the current.

Please note that you can use the procedure **SetOutlineCharset** to define a certain charset for the outline. You can use the value 0 to select the default charset or one of the following constants: EASTEUROPE_CHARSET, RUSSIAN_CHARSET, GREEK_CHARSET , TURKISH_CHARSET and BALTIC_CHARSET.

Parameters:

const Caption : string
This is the text which should be displayed in the outline tree.

const BookMarkName : string
This is the name of the book mark which should be linked to this outline entry. The book mark can be already known or defined later during the export of the text. You can also use the book mark feature of WPTools. (afsBookmark text style).

aPrevious : Integer
If this variable is greater 0 the outline entry will be created after the entry with give id. Note: If aParent and aPrevious are 0 the outline will be created at the top level.

aParent : Integer
If this variable is greater 0 the outline entry will be created as the child of the entry with the id provided here.

```
Example to create dummy entries:
procedure TForm1.WPPDFPrinter1BeforeEndDoc(Sender: TObject);
var last,inner : Integer;
    i,j : Char;
begin
  last := 0;
  for i:='A' to 'E' do
  begin
      Last := WPPDFPrinter1.SetOutline(i, 'DUMMY',last, 0);
      inner := 0;
      for j := 'A' to 'Z' do
         inner := WPPDFPrinter1.SetOutline(i+'.'+j,
                        'DUMMY', inner, Last);
  end;
end;
```

Example: Create test outline.

This code for the event OnAfterPrintPage will create some dummy information to test the outline feature.

Please see SetOutlineXY for example code.

This example code can be used to export from a TTreeView component to the PDF outline. It does not add the bookmark references, you have to add the logic for that.

```
procedure TForm2.WPPDFExportAfterPrintPage(Sender: TObject; Number,  FromPos, Length:
Integer);
  procedure CreateOutline(Node: TTreeNode; ParentItem: Integer);
  var ThisItem, i: Integer;
  begin
      // Create an entry for the current Node
      ThisItem := WPPDFExport.SetOutline(Node.Text,
        '', // MISSING: bookmark for this item
        0, ParentItem);
      // If the Node has children process them
      if Node.hasChildren then
         CreateOutline(Node.getFirstChild, ThisItem);
      // Only if the is the first node on this level process all the others
      // on the same level
      if Node.getPrevSibling=nil then
      while Node<>nil do
      begin
        Node := Node.GetNextSibling;
        if Node<>nil then
           CreateOutline(Node, ParentItem);
      end;
  end;
begin
  CreateOutline(TreeView1.TopItem, 0);
end;
```

10.4 Select Color (CMYK)

10.4.1 procedure SelectColorMode

Using the method SelectColorMode you can force the engine to convert the RGB colors used by the drawing code to either CMYK or grayscale.

Parameters:

Select : TSetColorExSelect

Possible values are:

wpFontColor - select the font color (Canvas.Font.Color)
wpPenColor - select the line color (Canvas.Pen.Color)
wpFillColor - select the fill color (Canvas.Brush.Color)

Mode : TSetColorExMode

Possible values are:

wpStandardColor, wpRGBMode - select the standard RGB mode
wpCMYKMode - the engine converts the RGB colors to CMYK

wpGrayMode - the engine converts to grayscale

10.4.2 procedure SetColorEx

Using the method SetColorEx you can force the engine to use a certain color for either the lines, the filling or the fonts. The color is valid until you use SetColorEx(xxx, wpStandardColor) with xxx beeing a value out of wpFontColor, wpPenColor, wpFillColor.

Parameters:

Select : TSetColorExSelect;

wpFontColor - select the font color (Canvas.Font.Color)
wpPenColor - select the line color (Canvas.Pen.Color)
wpFillColor - select the fill color (Canvas.Brush.Color)

Mode : TSetColorExMode;

wpStandardColor - disables the special color
wpRGBMode - select a RGB color
wpCMYKMode - select CMYK
wpGrayMode - select grayscale

RC: Single = 0

Select either Red in RGB mode or Cyan in CMYK. In Grayscale it should be 0.

GM: Single = 0;

Select either Green in RGB mode or Magenta in CMYK. In Grayscale it should be 0.

BY: Single = 0

Select either Blue in RGB mode or Yellow in CMYK. In Grayscale it should be 0.

K: Single = 0

Select either Black CMYK or the grayscale value. In RGB mode it should be 0.

Example:

```
WPPDFPrinter1.SetColorEx(wpPenColor, wpCMYKMode,
  0, 0.60, 1, 0.55  );

WPPDFPrinter1.SetColorEx(wpFillColor, wpCMYKMode,
  0, 0.60, 1, 0.55  );

WPPDFPrinter1.Canvas.Brush.Style := bsClear;

WPPDFPrinter1.Canvas.Rectangle( 100, 200, 300, 400);

WPPDFPrinter1.Canvas.Brush.Style := bsSolid;

WPPDFPrinter1.Canvas.Rectangle( 500, 500, 700, 700);

WPPDFPrinter1.SetColorEx(wpPenColor, wpStandardColor );
```

10.5 Fields (Annotations)

10.5.1 Function DrawAnnotation

The methods DrawAnnotation are very variable to create PDF annotations. However insights of the PDF syntax are required. Both methods return 0 or the number of the newly created annotation object.

```
function TWPCustomPDFExport.DrawAnnotation(
  pt : array of TPoint;
  AnnotType : AnsiString;
  AnnotPopupID : Integer;
  AnnotParams : array of String;
  AnnotAppearances : array of String;
  AnnotMode : Integer;
  AnnotData : Pbyte=nil; AnnotDataLen : Integer=0 ) : Integer;overload;

function DrawAnnotation( pt : array of TPoint;
        AnnotType : AnsiString;
        AnnotPopupID : Integer;
        AnnotParams : TStrings;
        AnnotMode : Integer;
        AnnotData : Pbyte=nil; AnnotDataLen : Integer=0 ) : Integer; overlo
```

Parameters:

pt This is an array of 4 or 8 points. In case of 4 points the PDF parameter /Rect will be created, otherwise also /QuadPoints will be created.

AnnotType This is the most important parameter. It can contain any PDF parameters as clear text, i.e. "/Subtype/FreeText/F 4" or "/Subtype/Text/Name/Comment/F 28/C [1 0 1]".

AnnotPopupID Unless it is 0, this is expected to be the number of a different annotation which was created before. This annotation will be then linked to the new annotation through the PDF parameter /Popup *ref*.

AnnotParams This array or stringlist contains a string parameters for the annotation in the form "name=value". Since string parameter have to be specially encoded into PDF it is not possible to pass them through the parameter AnnotType.

AnnotAppearances This are the names of the XForms (which can be created with StartWatermark) which should be used as appearance stream. In case AnnotParams is of type TStrings there is no AnnotAppearances parameter. In this case add the appearance names to the AnnotParams with a leading "#", i.e. "#N=FREETEXT1"

Example:

```
pdf.StartWatermark('FREETEXT1', 100,20,0,0);
pdf.Canvas.TextOut(0,0,'Watermarktext');
pdf.EndWatermark;

pdf.DrawAnnotation([Point(100,100), Point(200,120)],
      '/Subtype/FreeText/F 4',
```

```
i,
 ['Subj=Watermarktext',
  'Contents=Watermarktext',
  'DA=1 0 0 rg /Helv 12 Tf'
  ],
  ['N=FREETEXT1'],
  0, nil, 0);
```

AnnotMode Bitfield.
8192 - the annotation will not be listed in the Fields[] array of the AcroForm object

AnnotData and AnnotDataLen is currently not being used. It can be useful for multimedia.

10.5.2 procedure DrawTextField

Using the method DrawTextField you can create a editable field on the PDF page.

Parameters:

Text : String; - the default text

R : TRect - the rectangle in canvas coordinates

FieldName : String = '' - the name of this field

Hint : string = '' - a hint

FontSize : Integer = 0 - the font size for the editable text, 0 for default

Options : TWPEditControl - Possible Values are

wpecAutosizeFont - adjust the text height to avoid the text to scroll (default value)
wpecAlignRight - align text to right (not supported in a combo box)
wpecAlignCenter - align to center (not supported in a combo box)
wpecDrawBorder - draw a border around the field
wpecMultiLine - allow multiline input
wpecIsCombobox - create a combo box

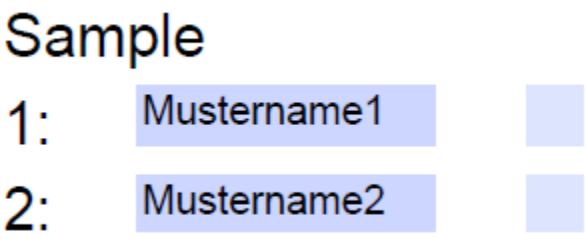
Font : TWPEditControlFont

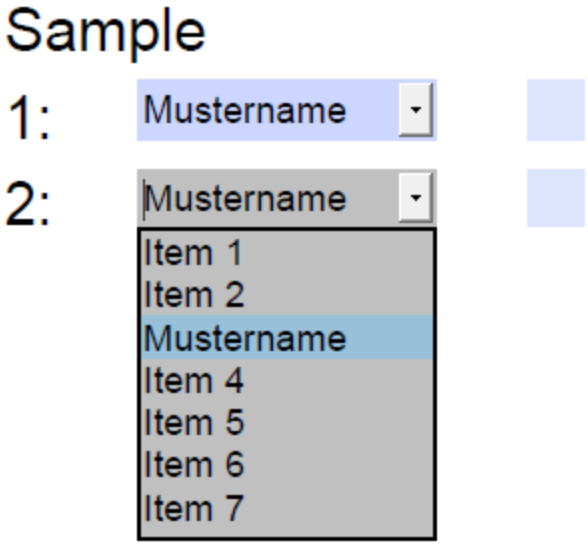
BGColor : TColor - this, unless clNone is the background color for the field

Items : String - this is a comma separated list with items for the combobox. The value is ignored if Options is not [wpecIsCombobox]

Select a font:

Values: wpecHelvetica (default) , wpecTimes, wpecCourier and wpecZapfDingbats





10.5.3 procedure DrawCheckbox

To draw a field please use the method

DrawCheckbox

It accepts the following parameters

R : TRect - the bounding box in logical coordinates
Value : Boolean - the initial value
FieldName : String - the name of the field (if empty it will be auto created)
Hint : string - a hint message, can be empty

10.5.4 Field Example

Example:

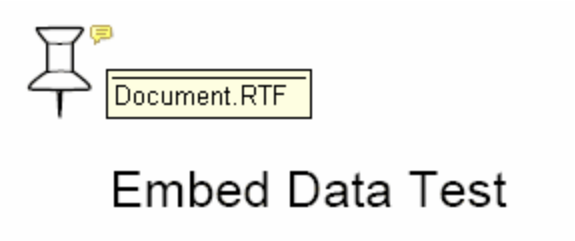
```
procedure TForm1.Button1Click(Sender: TObject);
var pdf : TWPPDFPrinter;  res : Integer;
    i, x, y, h, hh, w : Integer;
    can : TCanvas;
begin
    pdf := TWPPDFPrinter.Create(nil);
    try
      pdf.Filename := PDFName.Text;
      pdf.AutoLaunch := true;
      pdf.BeginDoc;

         res := Screen.PixelsPerInch;
         pdf.StartPage(
           Round( 21 * res / 2.54 ),
           Round( 29.7 * res / 2.54 ),
           res, res, 0
           );
         x := Round( 3 * res / 2.54 );
         y := Round( 3 * res / 2.54 );
         h := Round( 1 * res / 2.54 );
         can := pdf.Canvas;
         hh := Round( 0.7 * res / 2.54 );
         can.Font.Height :=  -hh;
         can.Font.Name := 'Arial';
         // ---
         can.TextOut(x,y, SomeText.Text);
         inc(y,h);
         for i:=1 to StrToIntDef(FIELDCOUNT.Text, 1) do
         begin
           can.TextOut(x,y, Format('%d:', [i]));
           w := Round( 1.5 * res / 2.54 );

           pdf.DrawTextField(
                  Edit1Text.Text + IntToStr(i),
                  Rect( x+w, y, x + Round( 5 * res / 2.54 ), y + hh),
                  Edit1Name.Text  + IntToStr(i)
              );

           pdf.DrawCheckbox(  Rect( x+Round( 6 * res / 2.54 ), y, x + Round( 6 * res / 2.54 )+hh, y + h
             Check1.Checked,
             Check1Name.Text + IntToStr(i)
           );


           inc(y, h);
         end;
          // ---
         pdf.EndPage;
         pdf.EndDoc;

    finally
      pdf.Free;
    end;
end;
```

10.6 Embed Data / Attach File or Stream

10.6.1 Method EmbedData

The method: EmbedData can be used to embed binary data into a PDF file.

When the user clicks an open dialog will be displayed by Acrobat Reader.



Parameters:

const aName : string;
This is the name which will be displayed as hint for this object. In the example above "Document.RTF". The name r

r: TRect;
This is the rectangle where to place the icon for the data. It is measured in canvas coordinates, this means you ca

In case there is no open page R will be interpreted in 72dpi PDF coordinates and the data will be linked to the last

Icon : TWPEmElementIcon;
The Icon to be displayed by AcrobatReader.

Possible values are:
- wpemNone,
- wpemPushPin,
- wpemGraph,
- wpemPaperClip,
- wpemTag

If you use wpemNone you can draw any image and place the hot spot at the same location.

data: TStream;

The data to be embedded.

compressmode: TWPEmCompressMode;

The compression mode. If you embed already compressed data you should use wpemDontCompress, otherwise u

const FileExtension: string

The file extension, for example RTF, ZIP or PNG.

10.6.2 Method AddFileAttachment

It is possible to attach a file or a stream to a PDF file. The data will be displayed by acrobat reader in the list of embedded files and can be saved from there or be opened. This feature makes it possible to save the original document with the PDF print out in RTF format or the invoice data with the invoice print in the rather new ZUGFeRD XML structure.

```
function AddFileAttachment(
    Name, Desc : WideString; Stream : TStream;
    const Typ : WideString = 'text/xml';
    ModDate: TDateTime = 0):Boolean; overload;


function AddFileAttachment(
    Name, Desc : WideString; Filename : WideString;
    const Typ : WideString = 'text/xml';
    ModDate: TDateTime = 0):Boolean;  overload;
```

You can provide a date "ModDate" or leave that field to be 0. In this case the current date will be used for streams, the file date for Files.

This example creates PDF from WPTools and also embeds the original document:

```
var pdf : TWPPDFExport;
    mem : TMemoryStream;
begin
  pdf := TWPPDFExport.Create(nil);
  // This code is only used by our demos.
  // Please remove this line in your code or
  // set a different path (see property DLLName in PDF manual)
  pdf.DLLName   := '{hkcu}Software\WPCubed\wPDF\4.0\path';
  pdf.FontMode  := wpEmbedType3;
  pdf.Source    := WPRichText1;
  pdf.AutoLaunch:= TRUE;
  pdf.FileName:= 'wp7out.pdf';

  pdf.BeginDoc;

  mem := TMemoryStream.Create;
  WPRichText1.SaveToStream(mem, 'RTF');
  pdf.AddFileAttachment( 'Document.RTF',
      'Embedded Document',
       mem,
       'application/rtf', Now );
  mem.Free;
  try
    pdf.Print;
  finally
    pdf.EndDoc;
    pdf.Free;
  end;
end;
```

10.6.3 Method AddXMPExtra (ZUGFerD)

The procedure AddXMPExtra(const **SchemaPartXML**, **InfoPartXML** : WideString) makes it possible to add custom XML (or XMP) data to the PDF metadata.

Please note - it is necessary to call AddXMPExtra before BeginDoc!

The string provided in **SchemaPartXML** will be embedded into the XMP PDFA schema inside a <rdf:Bag>.
So it is required to put it into `<rdf:li...> ... </rdf:li>`

```
<pdfaExtension:schemas>
            <rdf:Bag>
              .....added SchemaPartXML....
              .....other PDF/A related information ....
           </rdf:Bag>
 </pdfaExtension:schemas>
```

The extension of the PDF/A schema is required to be able to see the meta data in AcrobatReader.

The string **InfoPartXML** will be saved inside the XMP data as a separate branch.

Example to create XMP as found in standard ZUGFeRD invoice:

```
schema :=  '<rdf:li rdf:parseType="Resource">'+#13+#10+
             '<pdfaSchema:schema>ZUGFeRD PDFA Extension Schema</pdfaSchema:schema>'+#13+#10+
             '<pdfaSchema:namespaceURI>urn:ferd:pdfa:CrossIndustryDocument:invoice:1p0#' +
                '</pdfaSchema:namespaceURI>'+#13+#10+
             '<pdfaSchema:prefix>zf</pdfaSchema:prefix>'+#13+#10+
             '<pdfaSchema:property>'+#13+#10+
               '<rdf:Seq>'+#13+#10+
                 '<rdf:li rdf:parseType="Resource">'+#13+#10+
                   '<pdfaProperty:name>DocumentFileName</pdfaProperty:name>'+#13+#10+
                   '<pdfaProperty:valueType>Text</pdfaProperty:valueType>'+#13+#10+
                   '<pdfaProperty:category>external</pdfaProperty:category>'+#13+#10+
                   '<pdfaProperty:description>name of the embedded XML invoice file' +
                     '</pdfaProperty:description>'+#13+#10+
                 '</rdf:li>'+#13+#10+
                 '<rdf:li rdf:parseType="Resource">'+#13+#10+
                   '<pdfaProperty:name>DocumentType</pdfaProperty:name>'+#13+#10+
                   '<pdfaProperty:valueType>Text</pdfaProperty:valueType>'+#13+#10+
                   '<pdfaProperty:category>external</pdfaProperty:category>'+#13+#10+
                   '<pdfaProperty:description>INVOICE</pdfaProperty:description>'+#13+#10+
                 '</rdf:li>'+#13+#10+
                 '<rdf:li rdf:parseType="Resource">'+#13+#10+
                   '<pdfaProperty:name>Version</pdfaProperty:name>'+#13+#10+
                   '<pdfaProperty:valueType>Text</pdfaProperty:valueType>'+#13+#10+
                   '<pdfaProperty:category>external</pdfaProperty:category>'+#13+#10+
                   '<pdfaProperty:description>The actual version of the ZUGFeRD data' +
                     '</pdfaProperty:description>'+#13+#10+
                 '</rdf:li>'+#13+#10+
                 '<rdf:li rdf:parseType="Resource">'+#13+#10+
                   '<pdfaProperty:name>ConformanceLevel</pdfaProperty:name>'+#13+#10+
                   '<pdfaProperty:valueType>Text</pdfaProperty:valueType>'+#13+#10+
                   '<pdfaProperty:category>external</pdfaProperty:category>'+#13+#10+
```

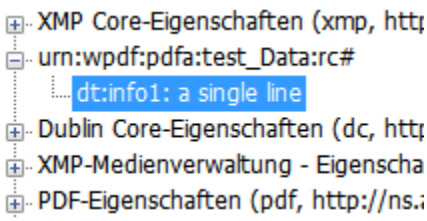
```
                    '<pdfaProperty:description>The conformance level of the ZUGFeRD data' +
                      '</pdfaProperty:description>'+#13+#10+
                  '</rdf:li>'+#13+#10+
                '</rdf:Seq>'+#13+#10+
              '</pdfaSchema:property>'+#13+#10+
            '</rdf:li>';


info :=   '<rdf:Description xmlns:zf="urn:ferd:pdfa:' +
          'CrossIndustryDocument:invoice:1p0#"' +
          ' rdf:about="" zf:ConformanceLevel="BASIC" ' +
          'zf:DocumentFileName="ZUGFeRD-invoice.xml"' +
          ' zf:DocumentType="INVOICE" zf:Version="1.0"/>';
```

```
//note: ZUGFeRD-invoice.xml is a fixed name, it must not be changed.

pdf.AddXMPExtra(schema, info);

pdf.BeginDoc;
```

To add the invoice data (*filename*) after BeginDoc call:

```
pdf.AddFileAttachment( 'ZUGFeRD-invoice.xml', 'Invoice data',
   filename, 'text/xml', now)
```

Display in Acrobat Reader under "Metadata":



10.7 GDIComment

GDIComments can be used to control the PDF engine. The comment data is inserted into the stream of graphical commands and ca so be executed in the context of different modifications to the graphic system, such as scaling or clipping.

This methods can export bitmap data using GDIComments when the optional parameter `OnCanvasMode` ist true.DrawPNGFile, DrawPNG and DrawJPG

To add GDI comments this methods can be used:
procedure WriteGDIComment(Comm: Integer; const r: TRect; data: PAnsiChar; datalen: Integer);

procedure WriteGDICommentStr(Comm: Integer; const r: TRect; text : AnsiString);

procedure WriteGDICommentStr(Comm: Integer; const r: TRect; utext: UnicodeString); overload;

In all cases a command ID "Comm" and a rectangle is exported. Optionally extra data can be added.

You can also use this code to create a GDIComment in your own code which can be compiled independently of wPDF.

```
procedure WPWriteGDIComment(
       Handle : HDC;
       Comm: Integer;
       const r: TRect;
       data: PAnsiChar;
       datalen: Integer);
type PWPComRec = ^TWPComRec;
TWPComRec = packed record a: Integer; b: Integer; c: TRect; d: Integer; end;
var    p: PWPComRec;
       pp: PAnsiChar;
begin
  GetMem(p, SizeOf(TWPComRec) + datalen);
  try
    FillChar(p^, SizeOf(TWPComRec) + datalen, 0);
    p^.b := 120269; p^.a := comm; p^.c := r;
    if datalen > 0 then
    begin
      pp := PAnsiChar(p); inc(pp, SizeOf(TWPComRec));
      Move(data^, pp^, datalen); p^.d := datalen;
    end;
    GdiComment(Handle, SizeOf(TWPComRec) + datalen, PAnsiChar(p));
  finally
    FreeMem(p);
  end;
end;
```

10.7.1 Command IDs to create hyper links

The command ID WPDFCOM_ADD_BOOKMARK = 552 can be used to create a bookmark. The parameter is the the name of the bookmark, the rectangle its destination. The bookmark name can be used for links and outlines. It is not necessary to create the bookmark before it is beeing zused, it can also be created later.

```
pdf.Canvas.TextOut(100,200, 'Goal!');
pdf.WriteGDICommentStr(WPDFCOM_ADD_BOOKMARK, Rect(100,200, 100,200), 'D
```

The command ID WPDFCOM_ADD_HYPERLINK = 551 can be used to create a hyperlink. The parameter is the destination bookmark.

```
pdf.Canvas.Rectangle(Rect(500,500, 600,600));
pdf.Canvas.TextOut(500,500, 'Click!');
pdf.WriteGDICommentStr(WPDFCOM_ADD_HYPERLINK, Rect(500,500, 600,600), '
```

10.7.2 Create fields

wPDF uses GDI comments to store the commands to create fields. This makes it possible to create fields in source code which does not use a reference to the wPDF VCL.

You only would need to copy this code and adapt the class name "`TDrawClass`":

```
type
TWPEditControlPDFOne =
  (wpecAutosizeFont,   wpecAlignRight,   wpecAlignCenter,   wpecDrawBorder, wpecMu
```

```
  TWPEditControl = set of TWPEditControlPDFOne;
  TWPEditControlFont = ( wpecHelvetica,  wpecTimes, wpecCourier, wpecZapfDingbats )


procedure TDrawClass.WriteGDICommentStr(Comm: Integer; const r:
TRect; text: AnsiString);
type PWPComRec = ^TWPComRec; TWPComRec = packed record a: Integer; b:
Integer; c: TRect; d: Integer; end;
var p: PWPComRec; pp: PAnsiChar;
begin
  GetMem(p, SizeOf(TWPComRec) + Length(text));
  try
    FillChar(p^, SizeOf(TWPComRec) + Length(text), 0);
    p^.b := 120269; p^.a := comm; p^.c := r;
    if text <> '' then
    begin
      pp := PAnsiChar(p); inc(pp, SizeOf(TWPComRec));
      Move(text[1], pp^, Length(text)); p^.d := Length(text);
    end;
    GdiComment(Canvas.Handle, SizeOf(TWPComRec) + Length(text),
PAnsiChar(p));
  finally
    FreeMem(p);
  end;
end;


procedure TDrawClass.DrawTextField(
    Text : String; R : TRect; FieldName : String {$IFNDEF D3}= ''{$ENDIF}; Hint : s
    FontSize : Integer {$IFNDEF D3}= 0{$ENDIF}; Options : TWPEditControl
    {$IFNDEF D3}= [wpecAutosizeFont]{$ENDIF}
    ; Font : TWPEditControlFont {$IFNDEF D3}= wpecHelvetica{$ENDIF} );
    var h : string;
const PDFFonts : array[TWPEditControlFont] of String = ('Helv', 'Times', 'Cour',
begin
      FieldName := Trim(FieldName);
      if FieldName='' then
      begin
         inc(FFieldNr);
         FieldName := 'FIELD' + IntToStr(FFieldNr);
      end;
      // Reset Parameters
      WriteGDICommentStr(  518, Rect(0,0,0,0), 'RESET=1');    //WPDFCOMM_TEXTFIELD_
      // Write Parameters
      if Font>wpecHelvetica then
      WriteGDICommentStr(  516, Rect(0,0,0,0), PDFFonts[Font]);   //WPDFCOMM_TEXTFI
      if wpecAlignRight in Options then
         WriteGDICommentStr(  517, Rect(0,0,0,0), '2') //WPDFCOMM_TEXTFIELD_ALIGN
      else if wpecAlignCenter in Options then
         WriteGDICommentStr(  517, Rect(0,0,0,0), '1');

      if wpecDrawBorder in Options then
           WriteGDICommentStr(  518, Rect(0,0,0,0), 'BORDER=1');   //WPDFCOMM_TEXTF
```

```
      if wpecMultiLine in Options then
           WriteGDICommentStr(  518, Rect(0,0,0,0), 'MULTI=1');   //WPDFCOMM_TEXTFI

      if not (wpecAutosizeFont in Options) then
      begin
           if FontSize>0 then
               WriteGDICommentStr(  518, Rect(0,0,0,0), 'SIZE=' + IntToStr(FontSize
           else WriteGDICommentStr(  518, Rect(0,0,0,0), 'SIZE=' + IntToStr(MulDiv(
      end;

      // Create Field
      if Hint = '' then h := '' else h := '@@HINT@@' + Hint;
      WriteGDICommentStr(  519, R, '@' + FieldName + '=' + Text+ h);

end;

procedure TDrawClass.DrawCheckbox( R : TRect; Value : Boolean; FieldName : String {
begin
  WriteGDICommentStr( 515, R, FieldName + '=' + IntToStr(Ord(Value)) + '@@HINT@@' +
end;
```

10.8 WPDF_ConvertImageFiles

This function converts a set of image file (typically *.PNG) to a new PDF file. The input files are specified in a list of strings, i.e. OpenDialog.Files.

```
function WPDF_ConvertImageFiles(
    PDFFile : String;
    InputFiles : TStrings;
    FitInW, FitInH : Integer;
    OptionSource : TWPCustomPDFExport=nil ) : Integer;
```

The encoding and font options are loaded from OptionSource. Internally a new instance of TWPCustomPDFExport is used.
You can specify the maximum width and height in 72dpi values.

This method requires a wPDFV4 engine later than 15.1.2016!

The code for this function is actually pretty simple and a good example how wPDF can be used:

```
function WPDF_ConvertImageFiles( PDFFile : String;
      InputFiles : TStrings;
      FitInW, FitInH : Integer;
      OptionSource : TWPCustomPDFExport=nil ) : Integer;
var i : Integer;
    wpdf : TWPCustomPDFExport;
begin
  Result := 0;
  wpdf := TWPCustomPDFExport.Create(nil);
  try
    if OptionSource<>nil then wpdf.Assign(OptionSource);
```

```
    wpdf.Filename := PDFFile;
    wpdf.BeginDoc;
    try
      for I := 0 to InputFiles.Count-1 do
      begin
         wpdf.DrawGraphicFile(0,0,FitInW,FitInH,InputFiles[i]);
         inc(Result);
      end;
    finally
      wpdf.EndDoc;
    end;
  finally
     wpdf.Free;
  end;
end;
```

11 Events

OnError(Sender: TObject; num: Integer; text: string)

This event is triggered if an error occurs. This error can be an exception the export DLL or a warning that a used book mark was not defined by the time the PDF export was finalized. In property "ExtraMessages" you can select wpOnEmbedFonts to get an event each time a font is embedded.

The following error numbers are defined:

ERR_BOOK = 1
A bookmark cannot be found.

ERR_BITM = 2
A bitmap cannot be embedded properly. It might be corrupt or compressed.

ERR_FILE = 3
The file is write protected.

ERR_META = 4
The metafile is not compatible.

ERR_FONT = 5
We tried to embed a font and failed.

ERR_IMPORT = 6
It was not possible to import the input PDF file. (It cannot be encrypted or LZW compressed)

ERR_MsgEmbedFont = 10
A font was embedded. (Not an error)

BeforeBeginDoc(Sender: TObject)

This event is created before the PDF file has been created. It is the last chance to modify certain parameters such as the "info" properties.

AfterBeginDoc(Sender: TObject)

This event is triggered right after the PDF file has been opened. It is the first chance to create a page, export metafiles or create an outline.

BeforeEndDoc(Sender: TObject)

This event is triggered before the PDF file has been closed. It is the last chance to create a page, export metafiles or create an outline.

AfterEndDoc(Sender: TObject)

This event is triggered when the PDF file or stream has been completed.

The TWPPDFExport class (for export from WPTools) also offers:

OnBeforePrintPage(Sender: TObject; Number, FromPos, Length: Integer)

This event makes it possible to create a watermark for each or some selected exported pages. Examples.

OnAfterPrintPage(Sender: TObject; Number, FromPos, Length: Integer)
This event makes it possible to draw over each or some of the exported pages.

OnPrintObject(Sender: TWPCustomPDFExport;
Memo : TWPRTFTextPaint;
PObj : PTTextObj;
Obj : TWPObject;
R : TRect;
var Abort : Boolean)

This event will only work when you are using **WPTools 4.09e** or later. It is triggered for each exported embedded TWPObject object. You get a pointer to the reference, the object and the rectangle where it should be drawn to. If you set "Abort" to true the default code is not executed. This is useful if you want to export the object (or any replacement data) yourself.
To do so you can use

```
Sender.Canvas.StretchDraw(R, Your_Graphic);
Abort := TRUE;
```

12 Linking with other products

Add wPDF to your toolbox and you are adding a single component that gives you a multitude of possibilities. This multi-talent has everything you need to create PDF files from ReportBuilder™, RAVE™, WPTools, THTMLView, ACE-Reporter, WPForm, QuickReport, FastReport and

DeveloperExpress™ Printing System. Ready-to-use sample code is provided here and filter interfaces are included

wPDF makes it possible for you to offer not only the same PDF export quality from different parts in your application but also combine the output into one PDF file. (see BeginDoc)

12.1 WPTools

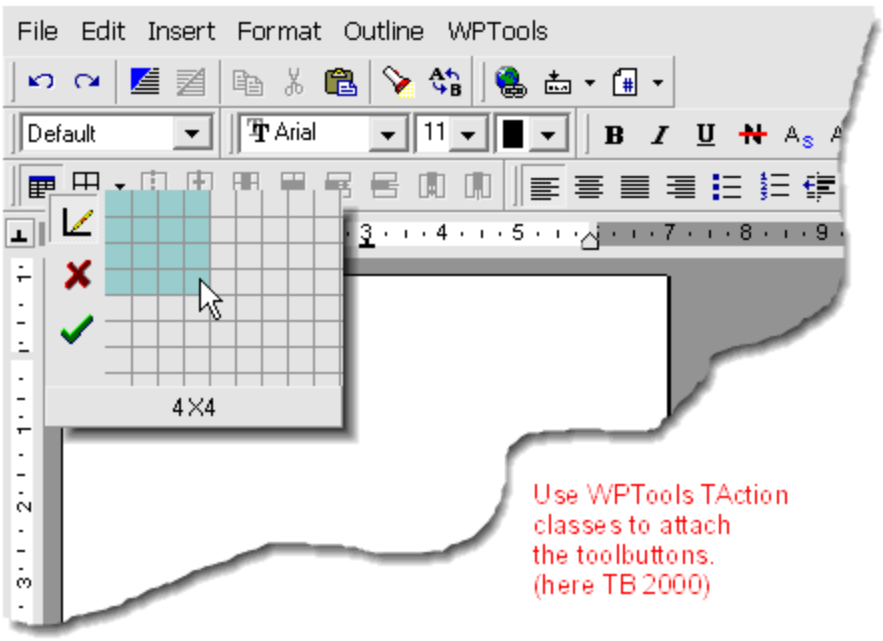
12.1.1 What is WPTools

A Component Suite for Word-Processing
Once you add this powerful VCL library (no OCX, DLL) to your project the users will soon have forgotten about their big word processor. WPTools supports page layout view, WYSIWYG, tables, headers and footers, many of paragraph and character attributes plus named paragraph styles.

A Component Suite for Automatic Text Creation
Use the versatile programming interface or the mail merge facility. The latter allows it also to build database forms since it can also read(!) out text. This "EditField" feature protects the complete text except for areas between certain tags which makes WPTools a perfect tool to fill&print contracts!
With the optional WPReporter (WPTools Bundle) you can also use bands and groups in your mail merge template and also use formulas in tables.

A Component Suite to free your Creativity!
The optional Export to PDF, mouse over effects and events for hyperlinks and fields, the possibility to display HTML tags without their parameters, the ability to work with custom made objects and reader/writer classes will make projects possible which were out of reach before. WPTools is completely customizable, you can change the dialogs (or don't use them at all) and attach GUI elements of your choice using the provided action classes.

Screenshot of a WPTools application - looks like "Word"?

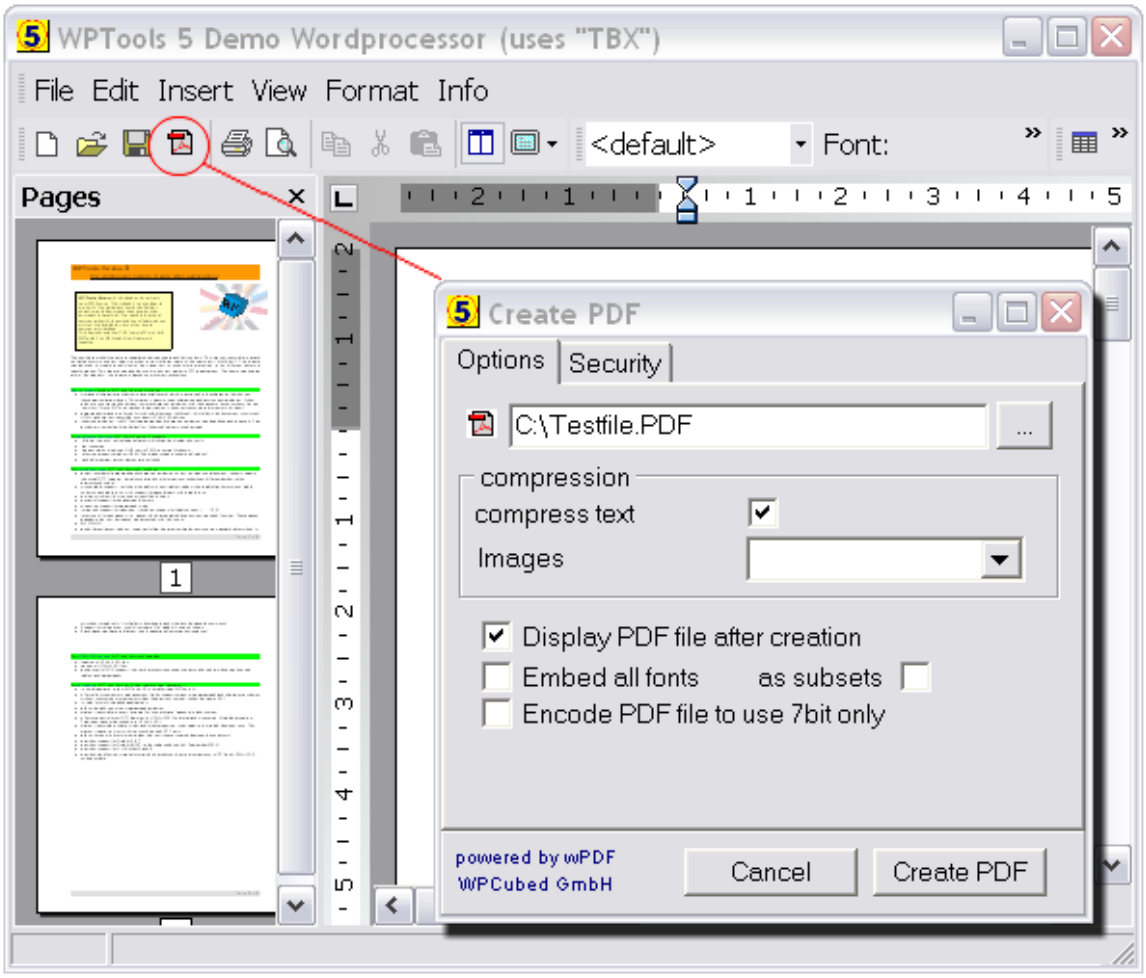


12.1.2 WPTools Version 5, 6, 7, 8 or 9

You can use wPDF also with the new WPTools Version 8 (and also V5, V6 and V7) which introduces a complete rewritten RTF engine with superior RTF and HTML capabilities.

Please make sure to add the conditional **{$DEFINE wPDF5}** to the file WPINC.INC to activate PNG support with transparency.

Outlines will created from paragraphs which use the WPAT_WPAT_ParIsOutline property. The value (>0) will be used as outline level.



To add PDF export fast use the form **TWPCreatePDF** which has been implemented in unit WPToPDFDlg. This will create the form above.

```
procedure TWPTBXForm.CreatePDFClick(Sender: TObject);
var pdfcreate: TWPCreatePDF;
begin
  pdfcreate := TWPCreatePDF.Create(Self);
  pdfcreate.EditBox := WPRichText1;
  try
    pdfcreate.ShowModal;
  finally
    pdfcreate.Free;
  end;
end;
```

Alternatives:

a) use TWPPDFExport in code.

```
uses ..., WPPDFWP, WPRTEDefs, WPCTRMemo, WPCTRRich;
```

```
procedure TForm1.ExportFromWPTools(Sender: TObject);
var pdf : TWPPDFExport;
begin
  pdf := TWPPDFExport.Create(nil);
  pdf.Source := WPRichText1;
  try
    pdf.FileName := 'c:\wp5out.pdf';
    pdf.Print;
  finally
    pdf.Free;
  end;
end;
```

b) Basically the code in WPPDFWP.PAS code is based on the following example which uses the TWPPDFPrinter directly.

```
// uses WPRTEPaint, WPPDFR1, WPPDFR2;

procedure TForm1.ExportToPDF(Sender: TObject);
var WPPDFPrinter1: TWPPDFPrinter;
    i,w,h : Integer;
begin
   WPPDFPrinter1 := TWPPDFPrinter.Create(nil);
   WPPDFPrinter1.FileName :='c:\wptools5demo.pdf';
   WPPDFPrinter1.CompressStreamMethod := wpCompressFastFlate;
   WPPDFPrinter1.AutoLaunch := TRUE;
   WPPDFPrinter1.BeginDoc;
   try
    i := 0;
    while i<WPRichText1.CountPages do
    begin
       w := MulDiv(WPRichText1.Memo._PaintPages[i].WidthTw,WPScreenPixelsPe
       h := MulDiv(WPRichText1.Memo._PaintPages[i].HeightTw,WPScreenPixelsP
       if (w=0) or (h=0) then
       begin
           w := Round( WPRichText1.Memo.PaintPageWidth[i] / WPRichText1.Mem
           h := Round( WPRichText1.Memo.PaintPageHeight[i] / WPRichText1.Me
       end;
       WPPDFPrinter1.StartPage( w, h, Screen.PixelsPerInch, Screen.PixelsPe
       try
         // Use 0 as w and h to let the function calculate the width and he
         WPRichText1.Memo.PaintRTFPage(i,0,0,0,0,WPPDFPrinter1.Canvas, [wpp
       finally
         WPPDFPrinter1.EndPage;
       end;
       inc(i);
    end;
   finally
    WPPDFPrinter1.EndDoc;
    WPPDFPrinter1.Free;
   end;
```

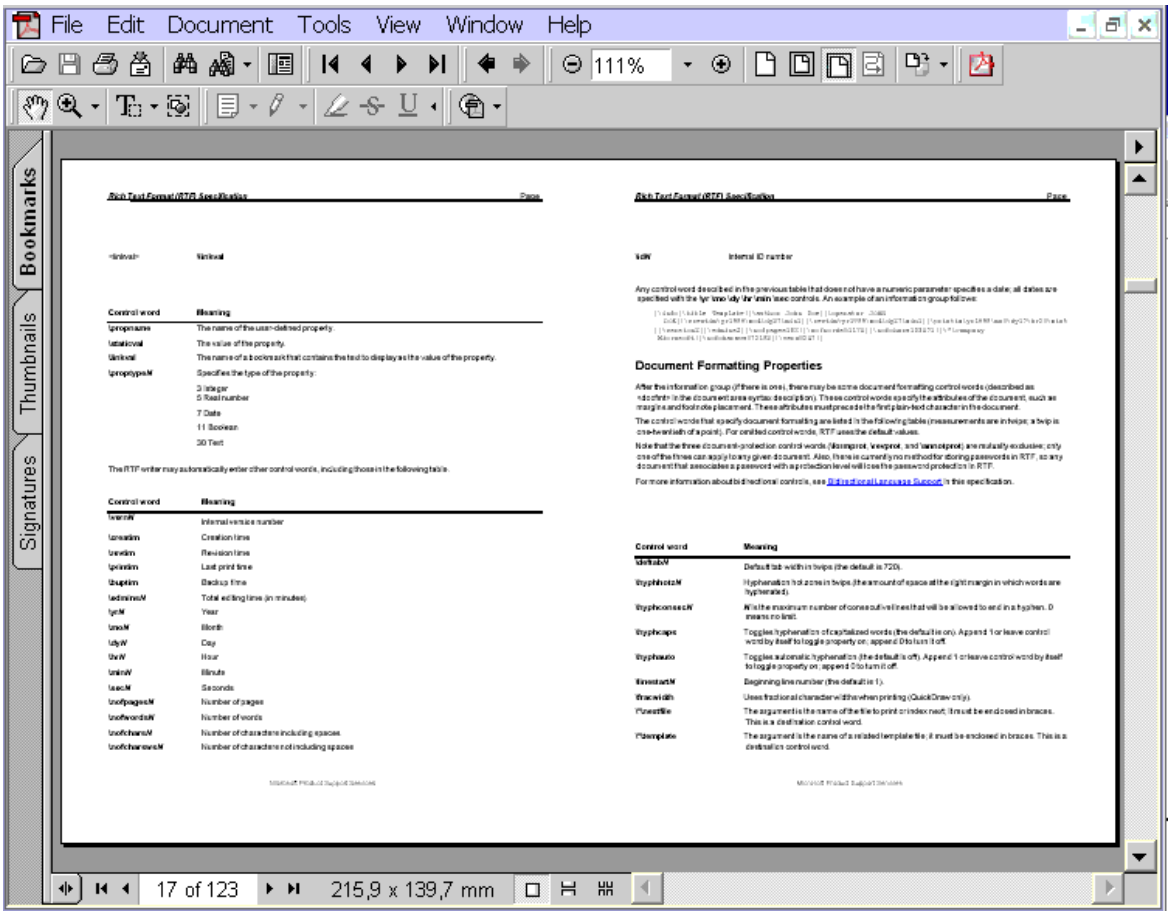
```
end;
```

To create watermarks simply add additional code which prints on the WPPDFPrinter1.Canvas.

Or you can easily **print 2 pages on the same PDF page**, just make changes in 4 lines:

```
var WPPDFPrinter1: TWPPDFPrinter;
    i, w, h : Integer;
begin
   WPPDFPrinter1 := TWPPDFPrinter.Create(nil);
   WPPDFPrinter1.FileName :='c:\wptools5demo.pdf';
   WPPDFPrinter1.CompressStreamMethod := wpCompressFastFlate;
   WPPDFPrinter1.AutoLaunch := TRUE;
   WPPDFPrinter1.BeginDoc;
   try
    i := 0;
    while i<WPRichText1.CountPages do
    begin
         w := MulDiv(WPRichText1.Memo._PaintPages[i].WidthTw,WPScreenPixels
         h := MulDiv(WPRichText1.Memo._PaintPages[i].HeightTw,WPScreenPixel
         if (w=0) or (h=0) then
         begin
           w := Round( WPRichText1.Memo.PaintPageWidth[i] / WPRichText1.Mem
           h := Round( WPRichText1.Memo.PaintPageHeight[i] / WPRichText1.Me
         end;
         WPPDFPrinter1.StartPage(w, h div 2,
         Screen.PixelsPerInch, Screen.PixelsPerInch, 0);
       try
         // Use 0 as w and h to let the function calculate the
width and height
         WPRichText1.Memo.PaintRTFPage(i,0,0,w div 2,h div
2,WPPDFPrinter1.Canvas, [] );
         WPRichText1.Memo.PaintRTFPage(i+1,w div 2,0,w div 2,h div
2,
                      WPPDFPrinter1.Canvas, [wppInPaintForwPDF] );
       finally
         WPPDFPrinter1.EndPage;
       end;
       inc(i,2);
    end;
   finally
    WPPDFPrinter1.EndDoc;
    WPPDFPrinter1.Free;
   end;
end;
```

This is a screenshot of a PDF file created with the above code:



12.1.3 WPTools Version 4

To export simply use the TWPDExport component.

Assign the property Source and execute 'Print'.

For the export from WPTools 4 the TWPRichText must use the property ScreenResMode = **rmNormal** and Zoom must be set to 100%.
Please use the ANSI DLL: wPDF500A.dll or wPDFDemo400A.dll.

This example exports into a memory stream and shows this stream in a WPViewPDF component.

```
var pdf : TWPPDFExport;
    mem : TMemoryStream;
    schema, desc : String;
    z : Integer;
begin
  pdf := TWPPDFExport.Create(nil);
  pdf.DLLName := ExtractFilePath(Application.ExeName) + 'wPDF500DEMOA.dll';
  mem := TMemoryStream.Create;
```

```
  try

    WPDF_Start('','');

    pdf.FontMode  := wpEmbedTrueTypeFonts;
    pdf.Source    := WPRichText1;
    pdf.Stream := mem;
    pdf.InMemoryMode := true;
    pdf.AutoLaunch:= FALSE;

    z := WPRichText1.Zooming;
    WPRichText1.LockScreen;
    try
      WPRichText1.Zooming := 100;
      WPRichText1.ReformatAll;
      pdf.Print;

    finally
      WPRichText1.Zooming := z;
      WPRichText1.UnLockScreen(true);
    end;

  finally
    pdf.Free;
    WPViewPDF1.LoadFromStream(mem, true);
    mem.Free;
  end;
end;
```

If you need to print additional data on each page, such as watermarks, use the OnBeginPrintPage event and draw to the wPDF "Canvas". Of course you can use BeginDoc to export several documents to the same PDf file and combine the exported data with data exported from reporting engine.

When using WPTools please make sure to add the conditional **{$DEFINE wPDF5}** to the file WPINC.INC to activate PNG support with transparency.

Note: Please read "Print On Canvas" for special tasks.

12.1.4 Print On Canvas

If you want to combine the output of WPTools output and the output from another source on the same PDF page it is also possible to print on the PDF canvas from within WPTools.
You don't need the WPPDFExport component here, just a WPPDFPrinter.

Example Code:

```
procedure TForm1.ExportToPDF(Sender: TObject);
var i: Integer;
begin
  WPPDFPrinter1.AutoLaunch := TRUE;
  WPPDFPrinter1.FileName   := 'c:\pdf_from_canvas.pdf';
  WPPDFPrinter1.BeginDoc;
```

```
  try
    for i := 0 to rtfDoc.CountPages - 1 do
    begin
      WPPDFPrinter1.StartPage(
        WPRichText1.Header.LayoutPIX.paperw,
        WPRichText1.Header.LayoutPIX.paperh,
        WPRichText1.Header.FFontXPixelsPerInch,
        WPRichText1.Header.FFontYPixelsPerInch,
        0);
      WPRichText1.PrintPageOnCanvas(
        WPPDFPrinter1.Canvas,
        Rect(0, 0,
             WPRichText1.Header.LayoutPIX.paperw,
             WPRichText1.Header.LayoutPIX.paperh),
        i,
        [ppmUseBorders],
        100);
      WPPDFPrinter1.EndPage;
    end;
  finally
    WPPDFPrinter1.EndDoc;
  end;
end;
```

12.1.5 Outlines

Outline items created in WPTools can be exported to PDF.

As Outline items all paragraphs are recognized which use the flag 'outlinemode'. This can be set in the WPRichText.CurrAttr.OutlineMode property. If you use the WPTools gutter a little dot will be shown for such paragraphs.

As standard behavior the outline items (adobe calls them bookmarks) are created one underneath each other. You can change this order be setting a level in the OnPrintOutline event:

```
procedure TForm1.OnPrintOutline(
      Sender: TWPCustomPDFExport;
      Memo : TWPRTFTextPaint;
      par  : PTParagraph;
      lin  : PTLine;
      var Caption : String;
      var level : Integer;
      var Abort : Boolean);
begin
  level := par^.numlevel-1;
  if level<0 then level := 0;
end;
```

This code will use the level of the outline in the text (the numbering level!) as depth in the PDF outline. You can of course also check for certain text sizes to change the depth.

12.1.6 Multi Document

When you execute

WPPDFExport.Print;

the PDF export component automatically starts a document and exports all pages.

If you create a document explicitly you can export multiple documents or selection of pages:

```
WPPDFExport.BeginDoc;
    for i:=1 to 1000 do
       WPPDFExport.Print;
WPPDFExport.EndDoc;
```

This example simply exports the document 1000 times. In our test it created 1000 pages in 9 seconds.

12.1.7 Use PDF Watermarks

If the watermark is rather big you can use the PDF watermark feature.
You need to create the watermark first to be able to use it.
You can do so in OnBeginDoc event to create a set of watermarks first.

```
procedure TForm1.DoBeginDoc(Sender: TObject);
var res : Integer;
begin
    res := Screen.PixelsPerInch;
    TWPPDFExport(Sender).StartWatermark('water',
          MulDiv(TWPPDFExport(Sender).Source.Header.PageWidth, res,1440),
          MulDiv(TWPPDFExport(Sender).Source.Header.PageHeight, res,1440),
          res, res );
    try
       TWPPDFExport(Sender).DrawGraphicFile(0,0,0,0, 'c:\test.emf' );
    finally
       TWPPDFExport(Sender).EndWatermark;
    end;
end;
```

In this event we start a watermark page which matches the pages in the RTF editor. You can of course also use the 'Canvas' property to draw the watermark.

This watermark can be easily used in the OnBeforePrintPage event:

```
procedure TForm1.DoBeforePrintPage(Sender: TObject; Number,
  FromPos, Length: Integer);
begin
  if TWPPDFExport(Sender).Tag<>0 then
  begin
     TWPPDFExport(Sender).UseWatermark('water');
  end;
end;
```

12.1.8 Print on Background of each Page

Sometimes you need a watermark, an image or just some text on some of the pages.

Although wPDF supports the PDF watermarks it is often a good idea to paint the information on each page. To do so you can use the OnBeforePrintPage event. This event is triggered before the text of one page is exported. You can either paint to the Canvas or use DrawTGraphic to export a metafile to the PDF file.

We modified the previous example to support watermarks. It is easy to modify the example to draw different graphics or notes like 'Confidential' on each page.

```
uses WPDefs, WPPrint, WpWinCtr, WPRich, WPPDFR1, WPPDFWP

procedure TForm1.RTF2PDF( RTFname, PDFname : string);
```

```
var
    WPRichText: TWPRichText;
    WPPDFExport: TWPPDFExport;
    Watermark : TPicture;
begin
  WPRichText:= TWPRichText.CreateParented(Application.Handle);;
  WPPDFExport:= TWPPDFExport.Create(nil);
  Watermark := TPicture.Create;
  WPPDFExport.Tag := Integer( Watermark );
  WPPDFExport.Source := WPRichText;
  WPPDFExport.OnBeforePrintPage := DoBeforePrintPage;
  try
    WPRichText.LoadFromFile( RTFname );
    Watermark.LoadFromFile( watermark_graphic );
    WPPDFExport.FileName := PDFname;
    WPPDFExport.Print;
  finally
    WPRichText.Free;
    WPPDFExport.Free;
  end;
end;

procedure TForm1.DoBeforePrintPage(Sender: TObject; Number,
  FromPos, Length: Integer);
begin
  if TWPPDFExport(Sender).Tag<>0 then
  begin
     TWPPDFExport(Sender).DrawTGraphic(0,0,
          0,0,(TObject(TWPPDFExport(Sender).Tag)as TPicture).Graphic);
  end;
end;
```

If you want to export the **background color** of the TWPRichText use this code:

```
procedure TForm1.DoBeforePrintPage(Sender: TObject; Number,
  FromPos, Length: Integer);
begin
    WPPDFExport1.Canvas.Brush.Color := WPPDFExport1.Source.Color;
    WPPDFExport1.Canvas.Brush.Style := bsSolid;
    WPPDFExport1.Canvas.Pen.Style := psClear;
    WPPDFExport1.Canvas.Rectangle(0,0,
      MulDiv(WPRichText1.Header.PageWidth,Screen.PixelsPerInch,1440),
      MulDiv(WPRichText1.Header.PageHeight,Screen.PixelsPerInch,1440)
      );
end;
```

If you want to export any **background Image**

You will need a variable

```
 BackImageNr : Integer;
```

which is set to -1 before the PDF is exported. This variable is used to fist initialize the bitmap which is then only used as "clone".

The background image is expected in object 'Image2'.

```
procedure TForm1.DoBeforePrintPage(Sender: TObject; Number,
  FromPos, Length: Integer);
var x,y : Integer;
begin
```

```
  for x:=0 to
     (MulDiv(WPRichText1.Header.PageWidth,Screen.PixelsPerInch,1440)+
        Image2.Picture.Bitmap.Width-1) div Image2.Picture.Bitmap.Width do
  for y:=0 to
     (MulDiv(WPRichText1.Header.PageHeight,Screen.PixelsPerInch,1440)+
        Image2.Picture.Bitmap.Height-1) div Image2.Picture.Bitmap.Height do
  begin

    if BackImageNr<=0 then
     BackImageNr := WPPDFExport1.DrawTGraphic(
       x * Image2.Picture.Bitmap.Width,
       y * Image2.Picture.Bitmap.Height,
       Image2.Picture.Bitmap.Width,
       Image2.Picture.Bitmap.Height,
       Image2.Picture.Bitmap)
    else
     WPPDFExport1.DrawBitmapClone(
       x * Image2.Picture.Bitmap.Width,
       y * Image2.Picture.Bitmap.Height,
       Image2.Picture.Bitmap.Width,
       Image2.Picture.Bitmap.Height,
       BackImageNr);
end;
```

12.1.9 Window-less RTF to PDF conversion

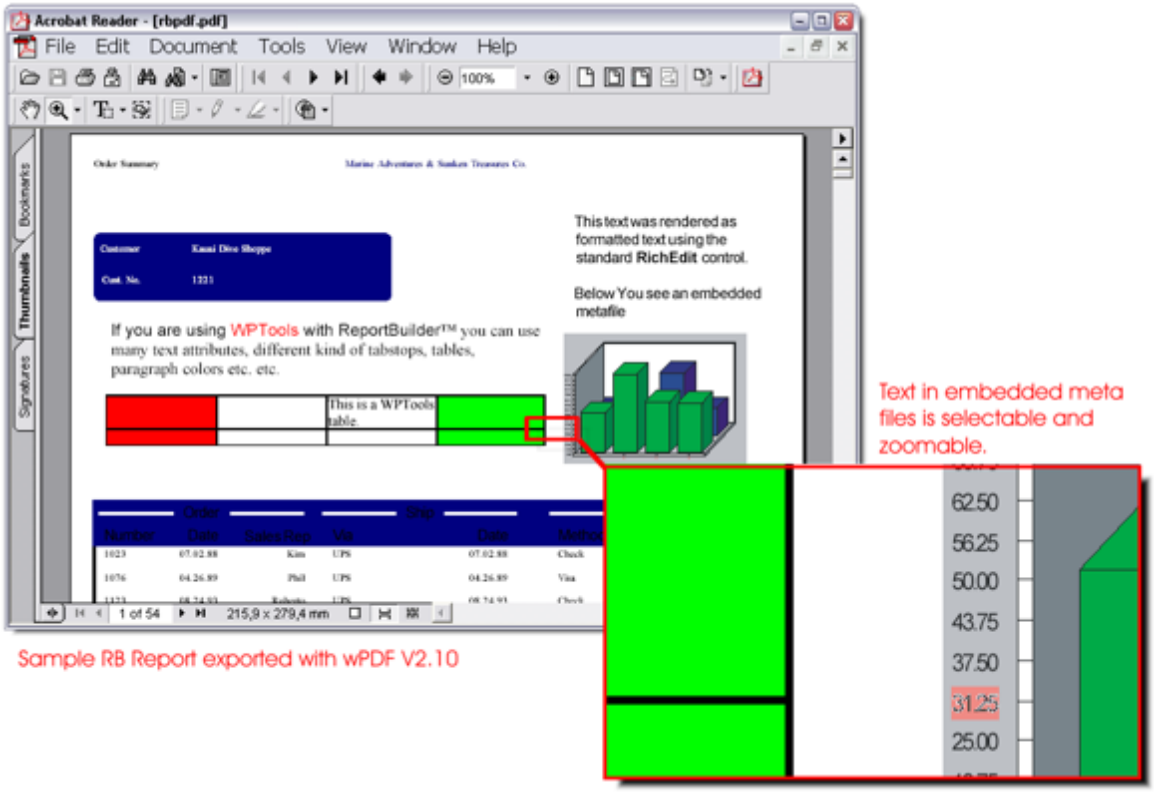
This sample code creates a TWPRichText and a TWPDFExport on the fly and uses both to create a PDF file:

```
uses WPDefs, WPPrint, WpWinCtr, WPRich, WPPDFR1, WPPDFWP

procedure RTF2PDF( RTFname, PDFname : string);
var
    WPRichText: TWPRichText;
    WPPDFExport: TWPPDFExport;
begin
  // WPTools 4
  WPRichText:= TWPRichText.CreateParented(Application.Handle);
  // WPTools 5:
  // WPRichText:= TWPRichText.CreateDynamic;
  WPPDFExport:= TWPPDFExport.Create(nil);
  WPPDFExport.Source := WPRichText;
  try
    WPRichText.LoadFromFile( RTFname );
    WPPDFExport.FileName := PDFname;
    WPPDFExport.Print;
  finally
    WPRichText.Free;
    WPPDFExport.Free;
  end;
end;
```

12.2 ReportBuilder

wPDF includes a device to create PDF files from ReportBuilder(tm) - (www.digital-metaphors.com)

It supports text, shape, barcode, line, bitmap, metafile, richtext (richedit) and also WPTools*) drawing commands. Metafiles are exported as vector graphics so there is no loss of resolution and the created PDF file will be as small as possible.



*) You need the latest version if the WPTools RB support units.

12.2.1 How to use the ReportBuilder device?

Add unit **wppdfRBDev** to your project.

To enable printing to file set the property AllowPrintToFile of the component TppReport to true. Then the print dialog will offer you the option to save to file with a list of possible file formats. Our device will add the item 'PDF Export (wpCubed GmbH)' to that list. (You can change the name in unit wppdfRBDev).

I need to export from *different* sources to ONE PDF file. Is this possible with wPDF?

Yes, and it makes much sense since wPDF is so versatile. In contrast to the usual export devices which are specialized on a certain application wPDF is specialized on a certain format: PDF. We have added a global variable to the unit wppdfRBDev: ppGlobalPDFPrinter.

If this points to an instance of a TWPDFPrinter or TWPDFExport (for WPTools) and that component has started a PDF file already (BeginDoc) the exported pages will go into this file as well. At the end the file will not be closed automatically.

Example:

```
procedure TForm1.StartPDFFile(Sender: TObject);
begin
  WPPDFPrinter1.FileName := 'c:\allout.pdf';
  WPPDFPrinter1.AutoLaunch := TRUE;
  WPPDFPrinter1.BeginDoc;
  ppGlobalPDFPrinter := WPPDFPrinter1;
end;

procedure TForm1.ClosePDFFile(Sender: TObject);
begin
  WPPDFPrinter1.EndDoc;
end;
```

Now export a 2 page WPTools letter and then any other report to the PDF file.

```
procedure TForm1.StartPDFFile(Sender: TObject);
begin
  WPPDFExport1.FileName := 'c:\allout.pdf';
  WPPDFExport1.AutoLaunch := TRUE;
  WPPDFExport1.BeginDoc;
  // Now export our WPTools Text
  WPPDFExport1.Source := WPRichText1;
  WPPDFExport1.Print;

  ppGlobalPDFPrinter := WPPDFExport1;
end;

procedure TForm1.ClosePDFFile(Sender: TObject);
begin
  WPPDFExport1.EndDoc;
end;
```

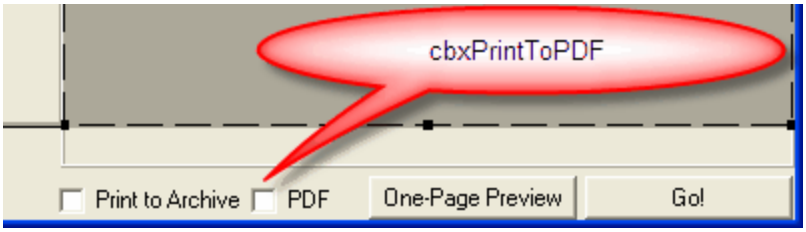
Note: Since the PDF file will not be valid without a WPPDFExport1.EndDoc; please don't forget to execute this function, the latest in Form.OnClose.

You can check if a PDFFrinter has already started a file by checking the property 'Printing'. In this case you should not call BeginDoc and EndDoc if you want to create a combined PDF file.

12.2.2 How to create a PDF file without display of a file save dialog?

It is easy to add PDF export to the ReportBuilder demo application

a) we need a check box:



b) modify the uses clause

```
uses ....  wppdfRBDev, wppdfr1;
```

c) modify the procedure PrintReport()

```
var
...
  pdf : TppwPDFDevice;


...

      if cbxPrintToPDF.checked then
      begin
        with TOpenDialog.Create(nil) do
        try
          Filter := 'PDF Files (*.PDF)|*.PDF';
          FileName := 'c:\rb.pdf';
          if Execute then
          begin
            pdf := TppwPDFDevice.Create(Self);
            pdf.Filename := FileName;
            pdf.PDFPrinter.FontMode := wpUseTrueTypeFonts;
            pdf.PDFPrinter.AutoLaunch := true;
            pdf.Publisher := lForm.Report.Publisher;
            lForm.Report.PrintToDevices;
            pdf.Free;
          end;
        finally
          Free;
        end;
      end else
        if cbxPrintToArchive.Checked then
  ...
```

This is the basic code needed for the task:

```
uses wppdfRBDev, WPPDFR1, WPPDFR2;

  var pdf : TppwPDFDevice;
  begin
    pdf := TppwPDFDevice.Create(Self);
    pdf.Filename := 'c:\rb.pdf';
    pdf.PDFPrinter.FontMode := wpUseTrueTypeFonts;
    pdf.Publisher := some_Report.Publisher;
    some_Report.PrintToDevices;
    pdf.Free;
   end
```

Note: *some_Report* is an instance of a RB report object.

Please note that through *pdf.PDFPrinter* You have access to all properties of the TWPCustomPDFExport object.

It is also possible to create a report as a stream, here "OutputStream":

```
var pdf : TppwPDFDevice;
begin
     WPPDFPrinter1.AutoLaunch:=False;
     WPPDFPrinter1.InMemoryMode:=True;
     WPPDFPrinter1.Stream:= OutputStream;
     pdf := TppwPDFDevice.Create(Self);
     pdf.PDFPrinter.FontMode := wpUseTrueTypeFonts;
     pdf.Publisher := ppReport1.Publisher; //some_Report.Publisher;
     WPPDFPrinter1.BeginDoc;
     ppGlobalPDFPrinter:=WPPDFPrinter1;
     ppReport1.PrintToDevices;
     WPPDFPrinter1.EndDoc;
     pdf.Free;
 end
```

12.2.3 RichView in RB

In case you use a RichView object you need to make this modifications in ppRichView.pas to make the PDF export work

1. Add wppdfRBDev to the uses clause

2. In tppDrawRichView.DoDraw ad a new condition:

```
...
  else if (aDevice is tppwpdfdevice) then begin
    lCanvas := tppwpdfdevice(aDevice).Canvas;
    ADrawRect := DrawRect;
  end
  ...
```

3. Ad a new condition in line 1200:

```
  ...
  Or (aDevice is tppwpdfdevice) ...
```

12.3 FastReport

For FastReport support please use the unit **WPDF_FRep**.

The uses clause should list:
WPDF_FRep, WPPDFR1, WPPDFR2;

Now you can create the component **TWPDF_FrPDFExport** in code.

By creating the component in code, you do not need to compile the wPDF Package with Fastreport. This avoids dependencies between packages.

As soon as the component was created, the preview will show the PDF export option:

You need to attached a TWPDFPrinter component which will be used to produce the PDF output. In this object You can modify the PDF properties as needed. Here we placed the TWPDFPrinter on the form. The FrPDFExport component ist only created at the first call.

Important:
If you need the component create the report prior to the export please create the compiler symbol **PREPARE** in the project options.

To export to a stream assign the stream directly to the TWPDF_FrPDFExport instance, property stream, not to WPPDFPrinter1.stream.

Example 1: Preview Report and let user print or save to PDF

```
var FrPDFExport : TWPDF_FrPDFExport;

procedure TForm1.Button1Click(Sender: TObject);
begin
  if FrPDFExport = nil then
    FrPDFExport := TWPDF_FrPDFExport.Create(nil);
  FrPDFExport.PDFPrinter := WPPDFPrinter1;
  FrPDFExport.Filename := 'c:\test.pdf';
  FrPDFExport.DefaultExt := 'pdf';

  frxReport1.ShowReport;
end;
```

If you want to modify the properties before the PDF file is created you can use the OnShowDialog event.

Example 2: This code can be used to export directly without showing a dialog.

It creates the components dynamically so it is not required to install the TWPDF_FrPDFExport or the component TWPPDFPrinter.

```
uses ... WPDF_FRep, WPPDFR1, WPPDFR2;

var aPDFExport : TWPDF_FrPDFExport;
begin
  aPDFExport := TWPDF_FrPDFExport.Create(nil);
  try
  // Use a instance of TWPPDFPrinter1 which was dropped on the form
  aPDFExport.PDFPrinter := WPPDFPrinter1;
  // Set a filename
  WPPDFPrinter1.Filename := 'c:\test_fr.pdf';
  // Just for debugging
```

```
  WPPDFPrinter1.AutoLaunch := true;

  // Prepare the report
  frxReport1.PrepareReport(true);
  // and export it using our filter
  frxReport1.Export( aPDFExport );
  finally
    aPDFExport.Free;
  end;
end;
```

Example 3: Create the PDF export and the filter in code (requires no packages):

```
// uses: WPPDFR1, WPPDFR2, WPDF_FRep;

procedure TForm1.Button2Click(Sender: TObject);
var FrPDFExport1 : TWPDF_FrPDFExport;
    WPPDFPrinter1 : TWPPDFPrinter;
begin
  FrPDFExport1 := TWPDF_FrPDFExport.Create(nil);
  WPPDFPrinter1 := TWPPDFPrinter.Create(nil);
  FrPDFExport1.PDFPrinter :=  WPPDFPrinter1;

  WPPDFPrinter1.FileName := 'c:\test.pdf';
  WPPDFPrinter1.AutoLaunch := true;
  try
  frxReport1.PrepareReport(true);
  frxReport1.Export(FrPDFExport1);
  finally
    FrPDFExport1.Free;
    WPPDFPrinter1.Free;
  end;
end;
```

Note: wPDF supports FastReport Version 2 and higher. (If you still use V2 please disable the {$DEFINE FASTR

12.4 TRichView / TSRVRichView

Code like this can be used to export the pages from the RichView VCL.

```
  WPPDFPrinter1.BeginDoc;
  try
    for I := PageF to PageT do
    begin
      WPPDFPrinter1.StartPage(
         Round(RichView1.PageProperty.PageWidth),
         Round(RichView1.PageProperty.PageHeight),
         Res, Res, 0);
```

```
      RichView1.DrawPage(I,
         Round(RichView1.PageProperty.PageWidth),
         Round(RichView1.PageProperty.PageHeight),
        0, 0, WPPDFPrinter1.Canvas, False, False, False);
      pdf.EndPage;
    end;
  finally
    WPPDFPrinter1.EndDoc;
  end;
```

Note: If you use the last version of ScaleRichView, set the last parameter to the call of DrawPage to True (= ForMetafile).

```
  WPPDFPrinter1.BeginDoc;
  try
    for I := 0 to ScaleRichView.PageCount -1 do
    begin
      w := Trunc(8.5 * pdfDPI);
      h := Trunc(11 * pdfDPI);

      WPPDFPrinter1.StartPage(w, h, pdfDPI, pdfDPI, 0);

      ScaleRichView.DrawPage(I,
                         w, h,
                         0, 0,
                         WPPDFPrinter1.Canvas,
                         False,
                         False,
                         False,
                         true );

       WPPDFPrinter1.EndPage;
    end;
  finally
    WPPDFPrinter1.EndDoc;
  end;
```

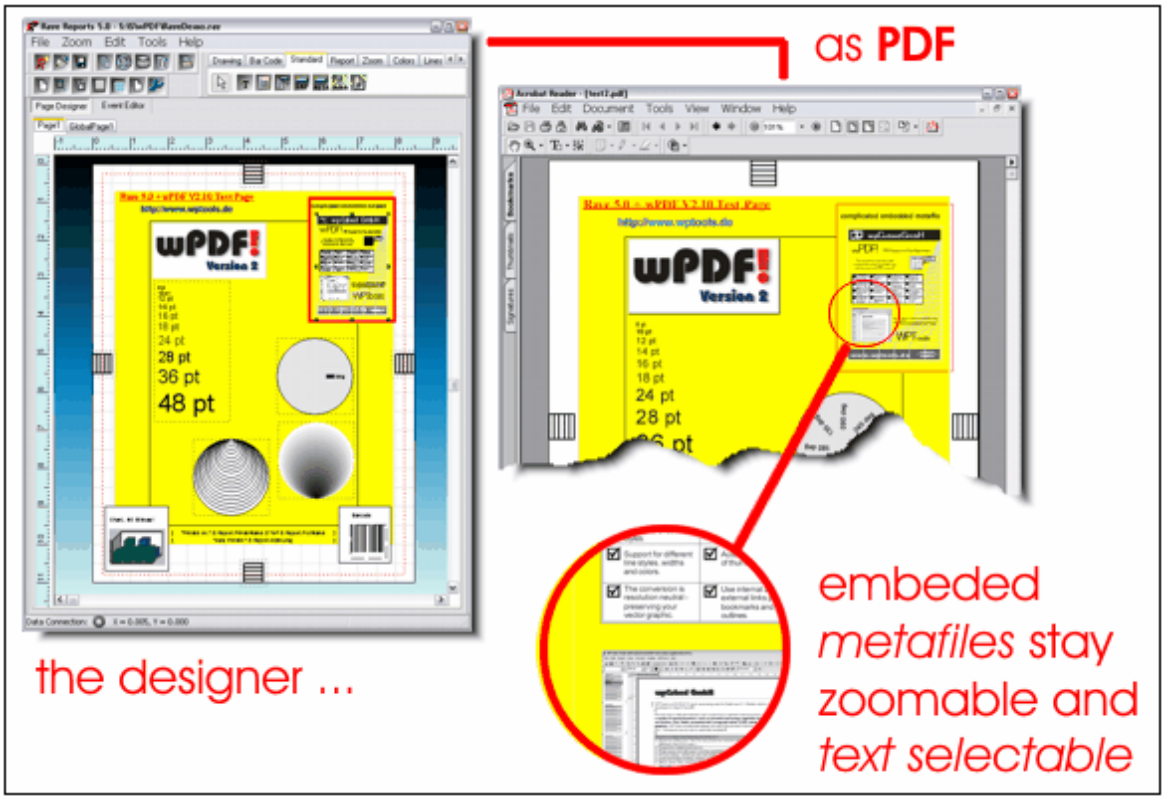
12.5 RAVE Report

Tested with RAVE 5 which is included in Delphi7 Professional.

Although RAVE comes with a PDF export filter you might prefer wPDF because

- it supports security (40, 128 bit)
- embedded metafiles are exported as vector drawing
- you can add custom drawing in the various events of wPDF

Used with a complex test page the export quality is pretty impressive, but please see yourself:



The code for the rave renderer is in unit wppdfRAVE. If you add this to the (wPDF) package the component TRvRenderWPDF will be installed. But you can of course also create it in code.

To use this filter place it on your form (or create it using code) and assign the property PDFPrinter to an instance of a TWPDFPrinter component. Once you set the 'Active' property to true you can select the file format created by this renderer when you save a report from preview

In contrast to the default PDF renderer included in PDF this enhanced product will export metafiles as vectors, not as bitmaps. This will create smaller files if the metafiles are big and result in better preview and print quality. You can also use the enhanced security options of wPDF

Example: Create renderer in code:

```
uses wppdfRAVE;

procedure TForm1.FormCreate(Sender: TObject);
begin
  RvRenderWPDF1 := TRvRenderWPDF.Create(Self);
  RvRenderWPDF1.PDFPrinter := WPPDFPrinter1;
  RvRenderWPDF1.Active := TRUE;
end;
```

Tip: If you want to combine the output of different reports in one PDF file you can execute WPPDFPrinter1.BeginDoc at the start of the loop and WPPDFPrinter1.EndDoc when everything has been exported. (But please don't forget to execute EndDoc!)

Example 1:
Render a NDR file directly to PDF, This example uses an open dialog to let you choose the file:

uses RpDefine, RpRender, RpRenderCanvas, RpRenderPreview, wppdfRave, WPPDFR1, WPPDFR2;

```
procedure TForm1.RenderNDRClick(Sender: TObject);
var
  OpenDialog: TOpenDialog;
  RvRenderWPDF: TRvRenderWPDF;
  WPPDFPrinter: TWPPDFPrinter;
  FileStream: TFileStream;
  output: string;
begin
  OpenDialog := TOpenDialog.Create(Self);
  RvRenderWPDF := TRvRenderWPDF.Create(Self);
  WPPDFPrinter := TWPPDFPrinter.Create(Self);
  try
    OpenDialog.Filter := 'NDF Files|*.NDR';
    RvRenderWPDF.PDFPrinter := WPPDFPrinter;
    RvRenderWPDF.Active := TRUE;
    WPPDFPrinter.AutoLaunch := TRUE;
    WPPDFPrinter.CompressStreamMethod := wpCompressFastFlate;
    if OpenDialog.Execute then
    begin
      output := ChangeFileExt(OpenDialog.FileName, '.PDF');
      FileStream := TFileStream.Create(OpenDialog.FileName, fmOpenRead);
      try
        RvRenderWPDF.PrintRender(FileStream, output);
      finally
        FileStream.Free;
      end;
    end;
  finally
    OpenDialog.Free;
    RvRenderWPDF.Free;
    WPPDFPrinter.Free;
  end;
end;
```

Example 2:
Render multiple NDR files directly to the PDF file 'c:\rave.pdf'. It uses an open dialog to let you choose the files.

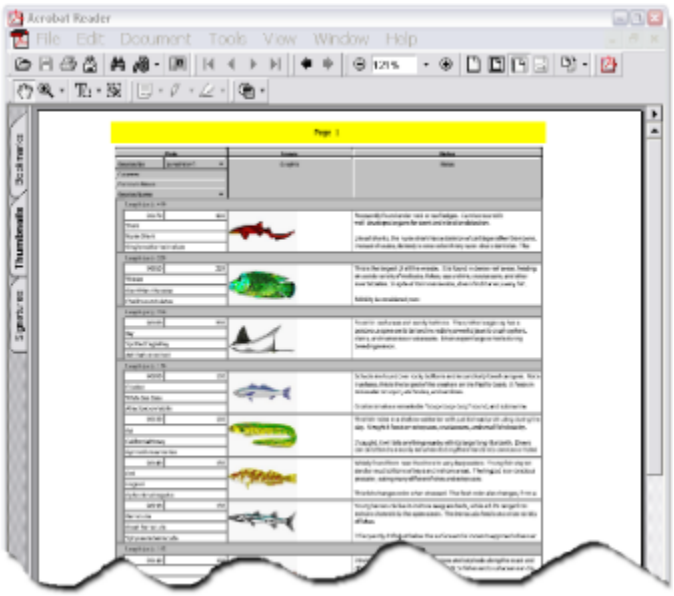
```
procedure TForm1.Button1Click(Sender: TObject);
var
  OpenDialog: TOpenDialog;
  RvRenderWPDF: TRvRenderWPDF;
  WPPDFPrinter: TWPPDFPrinter;
  FileStream: TFileStream;
  output: string;
  i: Integer;
begin
  OpenDialog := TOpenDialog.Create(Self);
  OpenDialog.Options := [ofAllowMultiSelect];
  RvRenderWPDF := TRvRenderWPDF.Create(Self);
  WPPDFPrinter := TWPPDFPrinter.Create(Self);
  try
    OpenDialog.Filter := 'NDF Files|*.NDR';
    RvRenderWPDF.PDFPrinter := WPPDFPrinter;
    RvRenderWPDF.Active := TRUE;
    WPPDFPrinter.AutoLaunch := TRUE;
    WPPDFPrinter.CompressStreamMethod := wpCompressFastFlate;
    if OpenDialog.Execute then
    begin
      output := 'dummy';
```

```
      WPPDFPrinter.Filename := 'c:\rave.pdf';
      WPPDFPrinter.BeginDoc;
      try
        for i := 0 to OpenDialog.Files.Count - 1 do
        begin
          FileStream := TFileStream.Create(OpenDialog.Files[i], fmOpenRead);
          try
            RvRenderWPDF.PrintRender(FileStream, output);
          finally
            FileStream.Free;
          end;
        end;
      finally
        WPPDFPrinter.EndDoc;
      end;
    end;
  finally
    OpenDialog.Free;
    RvRenderWPDF.Free;
    WPPDFPrinter.Free;
  end;
end;
```

TIP: Use Mail Merge to create the page numbers.

12.6 DevExpress

Since version V2.10 wPDF can export the output of the Developer Express(tm) EXPRESS PrintingSuite (www.devexpress.com) in a high quality.



This is the code to do it:

```
procedure TEQGridRLMainForm.PDFExportClick(Sender: TObject);
var i,w,h,res : Integer;
begin
  dxComponentPrinter1.RebuildReport(nil);
  WPPDFPrinter1.Modes := [wpClipRectSupport,wpAlwaysHighResPDF];
  WPPDFPrinter1.BeginDoc;
  try
```

```
    for I := 0 To dxComponentPrinter1.GetPageCount - 1 Do
     begin
     with dxComponentPrinter1.CurrentLink.RealPrinterPage Do
       begin
         res := Screen.PixelsPerInch;
         w := MulDiv(PageSizePixels.X,res,100);
         h := MulDiv(PageSizePixels.Y,res,100);

         WPPDFPrinter1.StartPage(w,h,res,res,0);
         dxComponentPrinter1.PaintPage(WPPDFPrinter1.Canvas,I,
                         Rect(0,0,w,h),
                         PaintRectPixels);
       end;
       WPPDFPrinter1.EndPage;
     end;
  finally
    WPPDFPrinter1.EndDoc;
  end;
end;
```

12.7 ACE Report

wPDF also works with ACE Reporter(tm) by SCT Associates, Inc., www.sct-associates.com

You need to add the unit AceWPDF to your project.

In your application simply execute **PDFPrinter.Attach** to attach a 'Create PDF' button to the global preview dialog .

Use PDFPrinter.Detach to remove it.

You can also create a PDF file from a TACEFile object using

```
   PDFPrinter.MakePDF(af: TAceFile ; PDFFileName: String);
```

You don't need to care about creation and destrcution of PDFPrinter!

In case you want to create a PDF file which consists of multiple reports attach a TWPPDFPrinter to the property PDFPrinter.PDFPrinter and execute the TWPPDFPrinter.BeginDoc/EndDoc procedures.

12.8 QuickReport

a) using the filer component

For QuickReport support please compile the unit wppdfQR.pas into your wPDF (or WPTools) package. Now you can place the component TQRwPDFFilter on the form.

b) using code

It is also possible use Your own code to create the PDF output:

```
var
  aMeta: TMetaFile;
```

```
  PageNum: Integer;
begin
  WPPDFPrinter1.FileName := 'c:\temp\DEMO.PDF';
  FCount := 0;
  try
    Quickreport1.Visible := FALSE;
    WPPDFPrinter1.CanvasReference := wprefScreen;
    WPPDFPrinter1.BeginDoc;
    Quickreport1.Prepare;
    for PageNum := 1 to Quickreport1.QRPrinter.PageCount do
    begin
      WPPDFPrinter1.StartPage(Quickreport1.QRPrinter.PaperWidthValue,
          Quickreport1.QRPrinter.PaperLengthValue,254,254, 0);
      aMeta := Quickreport1.QRPrinter.GetPage(PageNum);
      try
       WPPDFPrinter1.DrawMetafileEx(0,0,0,0,aMeta.Handle,
         Screen.PixelsPerInch, Screen.PixelsPerInch );
      finally
        aMeta.Free;
        WPPDFPrinter1.EndPage;
      end;
    end;
    WPPDFPrinter1.EndDoc;
    Quickreport1.QRPrinter.Free;
    Quickreport1.QRPrinter := nil;
  finally
    Quickreport1.Visible := TRUE;
  end;
end;
```

12.9 HTMLView

With the latest THTMLView version we recommend to use this code:

```
var MFPrinter : TMetafilePrinter;
    page : Integer;
    w,h,res : Integer;
    WPPDFPrinter1 : TWPCustomPDFExport;
begin
   MFPrinter := TMetafilePrinter.Create(Self);
   WPPDFPrinter1 := TWPCustomPDFExport.Create(Self);
   WPPDFPrinter1.FileName := 'c:\fromhtml.pdf';
   WPPDFPrinter1.AutoLaunch := TRUE;
   try
   Viewer.PrintPreview(MFPrinter);
   WPPDFPrinter1.CanvasReference := wprefPrinter;
   WPPDFPrinter1.BeginDoc;
   res := MFPrinter.PixelsPerInchX;
   for page := 0 to MFPrinter.LastAvailablePage-1 do
   begin
      w := MulDiv(MFPrinter.PaperWidth,res,MFPrinter.PixelsPerInchX);
      h := MulDiv(MFPrinter.PaperHeight,res,MFPrinter.PixelsPerInchy);
      WPPDFPrinter1.StartPage(w,h,res, res, 0 );
      WPPDFPrinter1.Canvas.Draw(0,0,MFPrinter.MetaFiles[Page]);
      WPPDFPrinter1.EndPage;
   end;
   WPPDFPrinter1.EndDoc;
   finally
     MFPrinter.Free;
     WPPDFPrinter1.Free;
   end;
end;
```

The line "WPPDFPrinter1.CanvasReference := wprefPrinter;" is optional.

A customer sent us this tip:

I have fixed the problem with the THTMLViewer component. In case it helps, in the TMetaFilePrinter.pas supplied with the THMTLViewer package, the reference DC used in the call to TMetaFileCanvas.Create was 0, i.e. the screen, but everything else is based off the printer, hence the mismatch. Easily fixed by changing the 0 to PrinterDC:

```
procedure TMetaFilePrinter.NewPage;
...
begin
...
   NewCanvas  := TMetaFileCanvas.Create(MetaFile, PrinterDC);
```

12.10 WPForm

WPForm is a component set to provide a tool to create labels and forms with ease. It has the ability to create lists as well. Its user interface is intuitive and cusomizable - you dont have to provide the enduser a tool which provides too many features to explain - in WPForm you can customize everything!

To create metafiles with WPForm you can use this code:

```
var meta : TMetafile;
    i : Integer;
begin
   Init;
   WPPDFPrinter1.BeginDoc;
   meta := nil;
   for i:=1 to FD.PageCount do
   try
     meta :=  GetMetafile(i);
     WPPDFPrinter1.DrawMetafile(0,0,meta.handle);
   finally
     meta.Free;
   end;
   WPPDFPrinter1.EndDoc;
end;

This is the function which creates the metafiles:

function TForm1.GetMetaFile(PageNo: Integer): TMetaFile;
var
  FCanvas: TMetafileCanvas;
  PageSize: TWPFPageSize;
  Twips: Integer;
  PixelsPerInch: Integer;
  Width, Height: Integer;
  DC: Cardinal;
begin
  PageSize := FD.PageSize(PageNo);
  Result := TMetafile.Create;
  try
    DC := GetDc(FD.Handle);
    try
      Twips := PageSize.PageWidthTW;
      PixelsPerInch := GetDeviceCaps(HDC(DC), LOGPIXELSX);
      Width := Round((Twips / 1440) * PixelsPerInch);
```

```
      Twips := PageSize.PageHeightTW;
      PixelsPerInch := GetDeviceCaps(HDC(DC), LOGPIXELSY);
      Height := Round((Twips / 1440) * PixelsPerInch);
    finally
      ReleaseDC(FD.Handle,DC);
    end;

    Result.SetSize(Width, Height);
    Result.Enhanced := TRUE;

    FCanvas   := TMetafileCanvas.CreateWithComment(Result,
      0,
      'WPForm - www.wptools.com','');
    try
      FD.PrintPageOnCanvas(FCanvas,PageNo,0,0);
    finally
      FCanvas.Free;
    end;
  except
    Result.Free;
    raise;
  end;
end;
```

If you need to redirect the printing of the preview dialog (such as a report or labels) you can set the property TWPPreviewDlg.CustomPrinting to true and use the three printing events to create output:

```
// WPFPreviewDlg1.CustomPrinting muse be TRUE

procedure TForm1.WPFPreviewDlg1CustomPrintEnd(Sender: TWPFReportEngine;
  FormEditor: TWPFormEditor);
begin
  WPPDFPrinter1.EndDoc;
end;

procedure TForm1.WPFPreviewDlg1CustomStartPrint(Sender: TWPFReportEngine;
  FormEditor: TWPFormEditor; var Abort: Boolean);
begin
  WPPDFPrinter1.Filename := 'c:\testwpform.pdf';
  WPPDFPrinter1.AutoLaunch := TRUE;
  WPPDFPrinter1.BeginDoc;
end;

procedure TForm1.WPFPreviewDlg1CustomPrintPage(Sender: TWPFReportEngine;
  FormEditor: TWPFormEditor; PageNo: Integer; var Abort: Boolean);
var res : Integer;
begin
  res := Screen.PixelsPerInch;
  WPPDFPrinter1.StartPage(
    MulDiv(FormEditor.Page.PageWidthTW,res,1440),
    MulDiv(FormEditor.Page.PageHeightTW,res,1440),
    res,res,0);
  try
  FormEditor.PrintPageOnCanvasZoom(WPPDFPrinter1.Canvas, PageNo, 0, 0, 100);
  finally
    WPPDFPrinter1.EndPage;
  end;
end;
```

It is also possible to create the PDF without first showing the the preview.
Please use the above code but instead of showing the report with

```
// Create report/labels and show it
WPFPreviewDlg1.PreviewReport(WPFReportEngine1);
```

Print it at once with

```
// Create report/labels and PRINT
WPFPreviewDlg1.PrintReport(WPFReportEngine1);
```

12.11 RichEdit

Procedure to export the text in any RichEdit object using wPDF to PDF.

It only requires the Handle of the RichEdit control (not the control!) the desired page size + margins in CM or INCH and a valid instance of a TWPDFPrinter class. In this class please set the filename as minimum. (The procedure can be used without linking in the unit "richedit" - it includes all type definitions it needs)

```
Example:
   WPPDFPrinter1.FileName := 'c:\richedit.pdf';
   RichEditPDFPrint(RichEdit1.Handle,
        21, 29.7,       // Page Size in CM
        2,2,2,2,        // Margins in CM
        true,           // Use CM - otherwise INCH are used
        WPPDFPrinter1); // A valid TWPDFPrinter Instace
```

If you need to export different styles of tab-stops, header and footer please check out WPTools to render the RTF text. This also eliminates the dependency on the RichEdit DLL

```
procedure RichEditPDFPrint(RichEditHandle : Cardinal;
  PageWidth, PageHeight, LeftMargin, TopMargin, RightMargin, BottomMargin : Extended;
  UseCM : Boolean; PDFPrinter : TWPCustomPDFExport);
const
  EM_FORMATRANGE = WM_USER + 57;
type
  TRichCharRange = record
    cpMin: Longint;
    cpMax: LongInt;
  end;
  TRichFormatRange = record
    hdc: HDC;
    hdcTarget: HDC;
    rc: TRect;
    rcPage: TRect;
    chrg: TRichCharRange;
  end;
var
  Range: TRichFormatRange;
  LastChar, MaxLen, LogX, LogY, OldMap, PageW, PageH: Integer;
  PageRect: TRect;
  DC : HDC;
begin
  FillChar(Range, SizeOf(TRichFormatRange), 0);
  DC := GetDC(0);
```

```
  LogX := GetDeviceCaps(DC, LOGPIXELSX);
  LogY := GetDeviceCaps(DC, LOGPIXELSY);
  try
    PDFPrinter.BeginDoc;
    // Initialize Page and output parameter
    if UseCM then
    begin
     PageW := Round( PageWidth * LogX / 2.54);
     PageH := Round( PageHeight * LogY / 2.54);
     PageRect.Left := Round( LeftMargin * 1440 / 2.54);
     PageRect.Right := Round( (PageWidth-RightMargin) * 1440 / 2.54);
     PageRect.Top := Round( TopMargin * 1440 / 2.54);
     PageRect.Bottom := Round( (PageHeight-BottomMargin) * 1440 / 2.54);
    end else
    begin
     PageW := Round( PageWidth * LogX);
     PageH := Round( PageHeight * LogY);
     PageRect.Left := Round( LeftMargin * 1440);
     PageRect.Right := Round( (PageWidth-RightMargin) * 1440);
     PageRect.Top := Round( TopMargin * 1440);
     PageRect.Bottom := Round( (PageHeight-BottomMargin) * 1440);
    end;
    // Initilaize Format Parameters
    LastChar := 0;
    MaxLen := SendMessage(RichEditHandle, WM_GETTEXTLENGTH, 0, 0);
    Range.rcPage := PageRect;
    Range.chrg.cpMax := -1;
    Range.hdcTarget := DC;
    Range.hdc := DC;
    SendMessage(RichEditHandle, EM_FORMATRANGE, 0, 0);
    try
      repeat
        PDFPrinter.StartPage(PageW,PageH,LogX, LogY, 0);
        Range.rc := PageRect;
        Range.chrg.cpMin := LastChar;
        Range.hdcTarget := PDFPrinter.Canvas.Handle;
        Range.hdc := PDFPrinter.Canvas.Handle;
        LastChar := SendMessage(RichEditHandle, EM_FORMATRANGE, 1, Longint(@Range));
        PDFPrinter.EndPage;
      until (LastChar >= MaxLen) or (LastChar = -1);
      PDFPrinter.EndDoc;
    finally
      SendMessage(RichEditHandle, EM_FORMATRANGE, 0, 0);  // flush buffer
    end;
  finally
    ReleaseDC(0,DC);
  end;
end;
```

13 FAQ

Does wPDF only work with WPTools ?

No, wPDF can also be used to create PDF files from general drawing code, such as Canvas.TextOut(). It works in two way 1) as export component for WPTools 2) as a universal PDF creation tool which provides a compatible canvas (class TCanvas). We have developed interface classes many important report engines.

Can I use wPDF to read and print PDF files ?
Sorry, this is not possible. What wPDF does is to reuse the data stored in an existing PDF file as watermarks for a new PDF file. While doing this it preserves the original information and does not interpret and translate the PDF file. This functionality is in particular useful when 'printing' to government forms.

When exporting text from my application characters overlap
Please make sure to specify the height for the fonts as negative values. Otherwise Windows works with rounded values and you do not get the same output in PDF as visible on screen.

Is the source code available ?

The pascal source for the VCL class that interacts with the PDF-Engine DLL is provided in wPDF, allowing the library to be compiled using a different Delphi or C++Builder Compiler. However, the PDF-Engine source code is not included and would require additional payment for licensing.

When do I need the 'Internet Server' License?
You need this license type if you want to create an application which runs on an internet server, such as an automatic order processing system.

I have the registered version but I still get the blue text / message box
You probably have forgotten to tell the engine your license information. You need to use the procedure WPDF_START(). The necessary codes have been sent to you. (Does not apply to the 'Source license')

The created PDF files are too big!
1) Please check the Font-Mode property. If you embed true type fonts this adds about 200KB data to the PDF file for each font if you don't use subsets. wPDF V4 can also create Type3 fonts which will be as small as possible.
2) It makes a difference whether you use jpeg or deflate compression for the images. Jpeg is better for photos, deflate is better for charts. Monochrome bitmaps will be always deflate compressed.

Certain text which should be italic is not exported italic
This happens when the used font does not exist as italic version.

Certain text which should be bold is not exported bold
This happens when the used font does not exist as bold version. As work around You can use the SimulateBold Mode.

Some of my graphics do not come out correctly. What can I do?

Although the PDF engine converts the output produces by a huge number of different applications sometimes not everything can be converted.
Clipping pathes can be converted but regions cannot if they are used within a rotated coordinate system (very seldom). We suggest to use paths where you used regions before. Not supported are text paths (for example text filled with a certain pattern), please use curves instead. (please see sample code) Filling supports solid colors or the standard hatch styles. Custom patterns or bitmaps cannot be used to fill areas. You need to create a clipping path and print the graphic multiple times. (wPDF will automatically detect the duplicates)
It is also not possible to export files generated by the spooler, those files are not really EMF files.

13.1 Code to draw outlined text

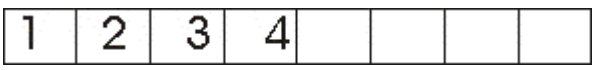
```
var i, l : Integer;
var points: array of TPoint;
var types: array of Byte;


 WPPDFPrinter1.Canvas.Font.Name := 'Arial';
 WPPDFPrinter1.Canvas.Font.Height := 300;

CanvasHandle2 := CreateCompatibleDC(WPPDFPrinter1.Canvas.Handle);
oldfont := SelectObject(CanvasHandle2, WPPDFPrinter1.Canvas.Font.Handle
BeginPath(CanvasHandle2);
TextOut(CanvasHandle2, 100, 100, 'Test', 4);
SelectObject(CanvasHandle2, oldfont);
EndPath(CanvasHandle2);

SetLength(points, 1);
SetLength(types, 1);
i := GetPath(CanvasHandle2, @points[0], @types[0], 0);
SetLength(points, i);
SetLength(types, i);
GetPath(CanvasHandle2, @points[0], @types[0], i);

AbortPath(CanvasHandle2);
DeleteDC(CanvasHandle2);


WPPDFPrinter1.Canvas.Brush.Style := bsClear;
WPPDFPrinter1.Canvas.Pen.Color := clRed;
WPPDFPrinter1.Canvas.Pen.Width := 0;

BeginPath(WPPDFPrinter1.Canvas.Handle);

PolyDraw(WPPDFPrinter1.Canvas.Handle, points[4],types[4], i-4);
EndPath(WPPDFPrinter1.Canvas.Handle);
StrokePath(WPPDFPrinter1.Canvas.Handle);
```

14 Tips

Our PDF Engine takes several efforts to produce the best looking PDF files you can get from your graphic output/files. For example text output is optimized to meet the width calculated by windows but also uses the character positioni9ng calculated by PDF. This results in a much better presentation. Internally rounding errors are avoided to avoid problems with difficult drawing objects, such as bar codes.

Some tips will help you to optimize the graphic output even more:

A) If you need output such as

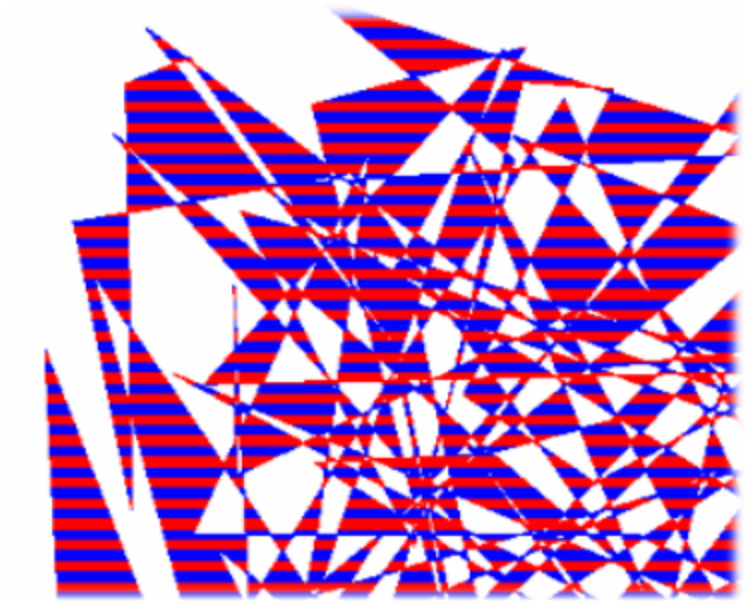


Please position each character using a separate text output command. Do not use TextOutEx command. You could also activate the ExactCharacterPosition mode in the property 'Modes' but that results is a lower quality of the other printed text.

B) When you export JPEG graphics you can do so by using the function DrawJPEG. This will export the JPEG data as it is without any decompression and compression and so is absolutely lossless.

C) Better Text Output - in the 'Modes' you can switch on the flag 'wpDontAdjustTextSpacing'. This will switch off the adaptation of the PDF output text to the length calculated by windows. If you don't use right aligned or justified text and don't print several text blocks in one line the output will look even better then. The reason is that the PDF reader controls the spacing to use the best quality for a certain font.

D) Our PDF engine now supports clipping regions. But you will get a better performance if you avoid regions and create clipping paths instead. Clipping paths are very easy to use. The following example produce a random polygon and then fills it. The output will look like:



```
var
  res,i : Integer;
  pt : array[0..100] of TPoint;
begin
  WPPDFPrinter1.Filename := 'c:\temp\test.pdf';
  res := Screen.PixelsPerInch;
  wpPDFPrinter1.BeginDoc;
  wpPDFPrinter1.StartPage(Round((21.0 / 2.54) * res),
      Round((29.7 / 2.54) * res), res, res, 0);

  // Create the points for a polygon
  for i:=0 to 99 do
  begin
    pt[i].x := res + Random(res*4);
    pt[i].y := res + Random(res*4);
  end;
  pt[100] := pt[0];

  // Paint the path
  SaveDC(wpPDFPrinter1.Canvas.Handle);
```

```
  BeginPath(wpPDFPrinter1.Canvas.Handle);
  wpPDFPrinter1.Canvas.Polygon(pt);
  EndPath(wpPDFPrinter1.Canvas.Handle);

  // and select it as clipping path
  SelectClipPath(wpPDFPrinter1.Canvas.Handle,RGN_AND);

  // Draw the "background"
  for i:=0 to Res do
  begin
     if wpPDFPrinter1.Canvas.Brush.Color = clRed then
           wpPDFPrinter1.Canvas.Brush.Color := clBlue
     else wpPDFPrinter1.Canvas.Brush.Color := clRed;
     wpPDFPrinter1.Canvas.FillRect(Rect(Res,Res+i*4,Res*5,Res+(i+1)*4));
  end;
  RestoreDC(wpPDFPrinter1.Canvas.Handle,-1);

  wpPDFPrinter1.EndPage;
  wpPDFPrinter1.EndDoc;
```

E) In seldom cases it is better to create a bitmap and export it to PDF. You can use this code as a template for a powerful solution which adapts itself to the size of a metafile.

```
procedure wPDFExportAsBitmap( Metafile : TMetafile; PDFPrinter : TWPCustomPDFExport;
StartPage : Boolean );
var
    bit : TBitmap;
    w,h : Integer;
    multa, multb : Extended;
const
    resolution = 96;
begin
  bit := TBitmap.Create;
  try
    w := Round(Metafile.MMWidth/2540*resolution);
    h := Round(Metafile.MMHeight/2540*resolution);
    multa := w/1000;  // maximum Size
    multb := h/1000;
    if multa>multb then multb := multa;

    bit.Width := Round(w/multb);
    bit.Height:= Round(h/multb);
    if StartPage then
       PDFPrinter.StartPage(Round(Metafile.MMWidth/2540*72),
        Round(Metafile.MMWidth/2540*72),72,72,0);
    try
      bit.Canvas.StretchDraw(Rect(0,0,bit.Width,bit.Height),Metafile);
      PDFPrinter.DrawBitmap(0,0,
        Round(Metafile.MMWidth/2540*PDFPrinter.XPixelsPerInch),
        Round(Metafile.MMWidth/2540*PDFPrinter.YPixelsPerInch),
        bit.Handle);
    finally
      if StartPage then  PDFPrinter.EndPage;
    end;
  finally
    bit.Free;
  end;
end;

procedure TForm1.ExportAsBitmapClick(Sender: TObject);
begin
  WPPDFPrinter1.Filename := 'c:\test.pdf';
  wpPDFPrinter1.BeginDoc;
```

```
  wPDFExportAsBitmap( Image1.Picture.Metafile, wpPDFPrinter1, true );
  wpPDFPrinter1.EndDoc;
end;
```

15 wPDF SourceCode License

Most of the wPDF Version 5 SourceCode can be purchased (excluded are only some obsolete PDF reading code and the also obsolete DLL interface)

Please check out the license agreement.

15.1 How to use

It is very easy to use the wPDF Source code. Simply copy the provided files in the same directory the units WPPDFR1.PAS etc reside.

Please open the file wpdf_inc.inc and make sure the define {$DEFINE WPDF_SOURCE} is active.

In your projects please add a

{$ wpdf_inc.inc} right at the beginning.

Now modify the uses clause to **use the unit WPPDFR1_src, WPGenDC and WPPdfDC** if the compiler symbol WPDF_SOURCE is defined:

```
{$I wpdf_inc.inc}

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs
  {$IFDEF WPDF_SOURCE} ,WPPDFR1_src, WPGenDc, WPPdfDC {$ELSE} ,WPPDFR1 {$ENDIF}
 etc etc ...
```

If you are using the function WPDF_START() please also put it into a condition:

```
{$IFNDEF WPDF_SOURCE}
  WPDF_START( LIcenseName, LicenseCode );
{$ENDIF}
```

Now you can compile the project. You can pass the project source to anybody who has not licenses the wPDF source code and it will work with the DLL version the same as does with the source.

NEW: The option types are now defined in the PDF engine source files, not in the unit wppdfr1_src anymore.

15.2 Comparision to Standard Version

The wPDF SourceCode is almost 100% alike the DLL version - there are only a few differences.

- Uses unit **WPPDFR1_src** instead of **WPPDFR1**
- Does not require WPDF_START() function

- Does not use the DLL interface. It uses the PDF-Engine directly
- TWPDFPagesImport component is not supported (in the STD version it is usually disabled since it is now obsolete)