WPViewPDF

Copyright (C) 2004-2012 WPCubed GmbH

Inhaltsverzeichnis

1 Introduction

WPViewPDF Version 3

PDF view, -print and manipulation technology by WPCubed GmbH

WPViewPDF is a component to load one or many PDF files to display or print as one. It is possible to export pages a bitmaps or as text. It is possible to add drawings which will be displayed and printed on top of the original data. It is possible to change field data, for example to fill out forms.

With WPViewPDF PLUS you can also add graphical objects and images to the PDF data (stamp PDF). It is possible to combine several PDF files into one new (merge PDF). It is also possible save selected pages (extract pages) or delete certain pages.

Welcome to the exciting new Version 3

The Version 3 is the result of extensive work. We completely re-thought the logic which is require to load, render and manipulate PDF data to create this new version. It makes use of clever and effective caching for quick response times. It also makes use of multithreading for better user interaction.

We revised the PrintHDC method - printing to any windows device should be now much easier to do than before and produce higher quality.

The multithreaded scrolling viewer can change quickly change between zoom states and various layout modes, including multi column display and side by side page layout. It can also display a separate thumbnail view to the PDF.

Unlike version 1 and 2 the new version 3 uses floating point numbers for graphic output which offers better print results for many PDF files. Despite the higher text rendering quality, printing will be faster since less data has to be transferred to the printer.

Version 3 PLUS introduce a new stamping method which also it to place objects or highlighting rectangles on the page. This objects can be moved and sized by the user. But we also implemented the scripted stamping because it makes it so easy to add titles or page numbers to a range of pages. It is also possible to move pages once they were selected.

Text extraction now also creates text in rich text format (RTF) - here the logic tries to make use of PDF tags to keep text together which belongs together.

The field support has been enhanced for better compatibility with existing PDF

files. We work to add the ability to *create* new fields to the "PLUS" Edition.

Why do I need a PDF viewer component?

- If you need to embed a PDF viewer into your application, then you need WPViewPDF since this will, most likely, no longer be allowed with the Acrobat (tm) Viewer Version 6 or later.)
- If you need to load PDF files from memory, then you need WPViewPDF which will allow you to load PDF files from any stream. The stream interface makes it possible for you to use your own encryption/decryption scheme for the loading process.
- If you need to print the PDF files created by your own application, then you need WPViewPDF which makes it possible to print several PDF files using just one printer job without starting any external application
- If you need to use information from PDF files as background images in your application, then you need WPViewPDF since it has the ability to extract PDF pages as metafiles or print to a windows device (HDC).
- You can offer the user the ability to add custom texts and highlighting areas to a PDF file.
- You can extract text from PDF under program control
- Versatile printing, with auto scaling and multi column/row printing.
- Create a transparent highlight rectangle on a page and move it under program control (or let the user drag and move it)
- Read and write (PLUS Edition) to fields on PDF frorms. This makes it possible to fill out such forms under program control.
- Last but not least: Imagine a powerful and versatile print and preview which is based completely on PDF files. The PDF files can be viewed, printed (with WPViewPDF or Acrobat(tm) Reader), stored or send via e-mail!

History of WPViewPDF

WPViewPDF V1 was created in 2003, mainly as viewer for PDF files which were created by our own PDF engine. While version 2 already produced much better display than V1 it still suffered from the limitation to internally use a graphics library which was based on integer coordinates.
So we decided to rewrite most of the code for version 3 - it is based on a new extensible architecture which makes it much easier to extend the system.

1.1 WPViewPDF Standard

WPViewPDF is meant to be a viewer for PDF text created by your application.

It can convert PDF pages to bitmaps (JPEG, PNG and BMP), metafiles and print to a windows device (HDC).

It can load several PDF files and display and print it as if it was just one.

Of course printing of the PDF is possible.

1.2 WPViewPDF PLUS

The PLUS edition works like the standard version but You can also save the loaded PDF data.

This makes it possible to
- delete or extract pages
- apply markers (stamps) to certain PDF pages
- move pages

With the PLUS edition You can also save the PDF file as a monochrome or color multipage TIFF file.

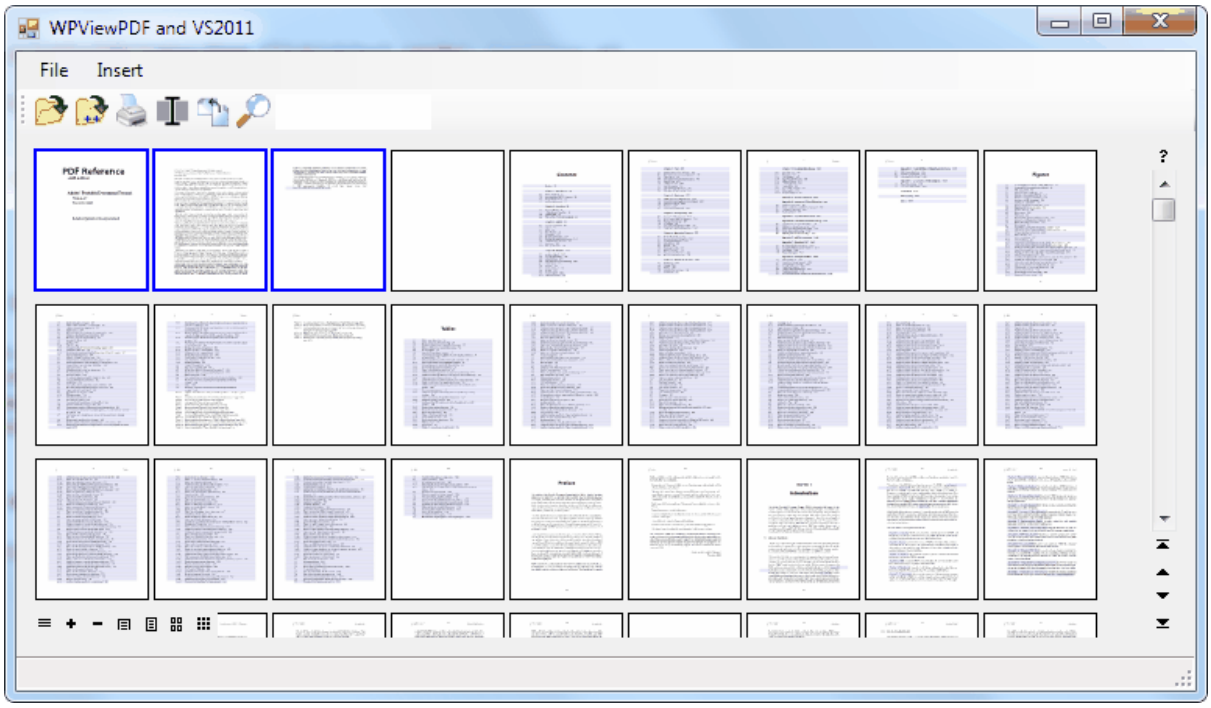
Further more, WPViewPDF has several possibilities to add images, text and vector graphics to PDF files.

When a new PDF file is written from the data loaded into WPViewPDF, Version 3 now tries to only integrate the font and image resources, which are actually used by the text. This can reduce the required size a lot.

1.3 Example Projects

1.3.1 .NET C# Example: PDFViewNET

A simple PDF viewer with image export



The component has been dropped on the from. It is initialized like this:

```
        public Form1()
        {
            InitializeComponent();
            pdfViewer1.ViewerStart("xxx",  "yyy", 0);

            pdfViewer1.ViewOptions = eViewOptions.wpExpandAllBookmarks |
                   eViewOptions.wpExpandAllBookmarks |
                   eViewOptions.wpSelectPage |
                   eViewOptions.wpShowPageSelection;

            pdfViewer1.ViewControls =
                  eViewControls.wpHorzScrollBar |
                  eViewControls.wpNavigationPanel |
                  eViewControls.wpPropertyPanel |
                  eViewControls.wpVertScrollBar |
                  eViewControls.wpViewPanel;

            pdfViewer1.Command(commands.COMPDF_SetDocumentProperties, "Eigenschaften"
);

        }
```

Load and append PDF files:

```
        private void loadToolStripMenuItem1_Click(object sender, EventArgs e)
        {
            if (openFileDialog1.ShowDialog() == DialogResult.OK)
            {
```

```
            pdfViewer1.LoadFromFile(openFileDialog1.FileName);
        }
    }

    private void appendToolStripMenuItem_Click(object sender, EventArgs e)
    {
        if (openFileDialog1.ShowDialog() == DialogResult.OK)
        {
            pdfViewer1.AppendFromFile(openFileDialog1.FileName);
        }
    }
```

Show the print dialog:

```
    private void Print_Click(object sender, EventArgs e)
    {
        pdfViewer1.Command(commands.COMPDF_PrintDialog);
    }
```

Implement the find method. It will locate next location unless the string was changed:

```
static string LastFind;

    private void FindBtn_Click(object sender, EventArgs e)
    {
         int p = pdfViewer1.FindText(FindTextEdit.Text, true, LastFind ==
FindTextEdit.Text, true, true);
         if (p < 0) MessageBox.Show("Text not found.");
         LastFind = FindTextEdit.Text;
    }
```

Implement saving to a new PDF file (requires Demo or PLUS license)

```
    private void pDFToolStripMenuItem_Click(object sender, EventArgs e)
    {
        saveFileDialog1.Filter = "PDF Files|*.PDF";
        if (saveFileDialog1.ShowDialog() == DialogResult.OK)
        {
            if (!pdfViewer1.Plus.SaveToFile(saveFileDialog1.FileName))
                MessageBox.Show("Saving the file was not successful!");
        }
    }
```

Create a bitmap from the current page. Possible formats are BMP, PNG and JPEG. It simply uses the caption of the sender menu item.

```
    private void jPEGToolStripMenuItem_Click(object sender, EventArgs e)
    {
        saveFileDialog1.Filter = ((ToolStripMenuItem)sender).Text + " Files|*." +
((ToolStripMenuItem)sender).Text;
```

```
            if (saveFileDialog1.ShowDialog ()== DialogResult.OK)
            {
                if (!pdfViewer1.WriteBitmap(pdfViewer1.Page-1, BitmapFormat.Automatic,
saveFileDialog1.FileName))
                    MessageBox.Show("Saving the file was not successful!");
            }
        }
```

Rotate the selected pages

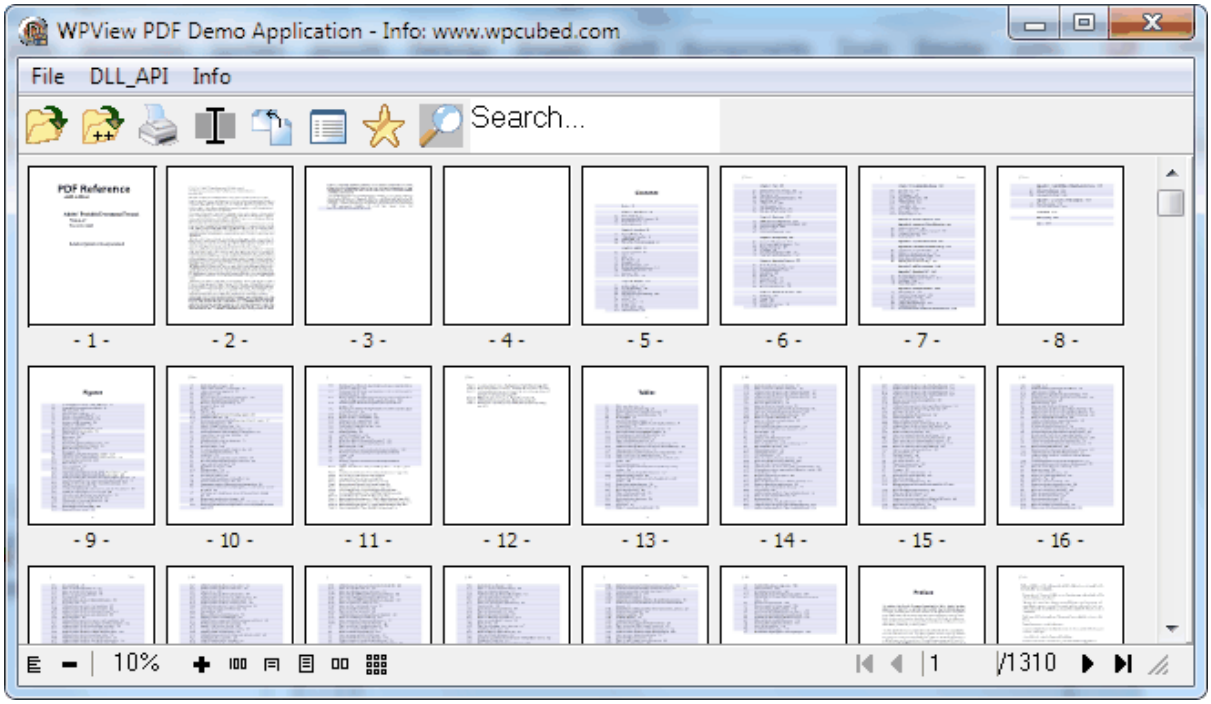
```
        private void RotateBtn_Click(object sender, EventArgs e)
        {
            pdfViewer1.Command(commands.COMPDF_RotatePage, "selected", -90);
        }
```

Switch between select and pan mouse mode

```
        private void SelectBtn_Click(object sender, EventArgs e)
        {
            SelectBtn.Checked = !SelectBtn.Checked;
            if (SelectBtn.Checked)
                pdfViewer1.Command(commands.COMPDF_SelectMode, 1);
            else pdfViewer1.Command(commands.COMPDF_SelectMode, 0);
        }
```

1.3.2 Delphi: PDFView

The simple pdf viewer test application - "PDFView"



This demo is as closely as possible based on the code of the demo developed for version2. The buttons are not part of WPViewPDF but part of this little test application. You can start the demo with a certain PDFView DLL as command line parameter. This makes it possible to test different DLLs.

 You can search for the given text in the PDF. Unless the text was modified, following clicks will search on subsequent pages.

 Activates the selection mode. You can select text on one page and press CTRL+C to copy it to the clipboard.

 This button rotates the selected page or pages. Click right on a page to select it. It will be displayed with a blue frame. Pages can also be selected with Shift+Cursor Left/Right.

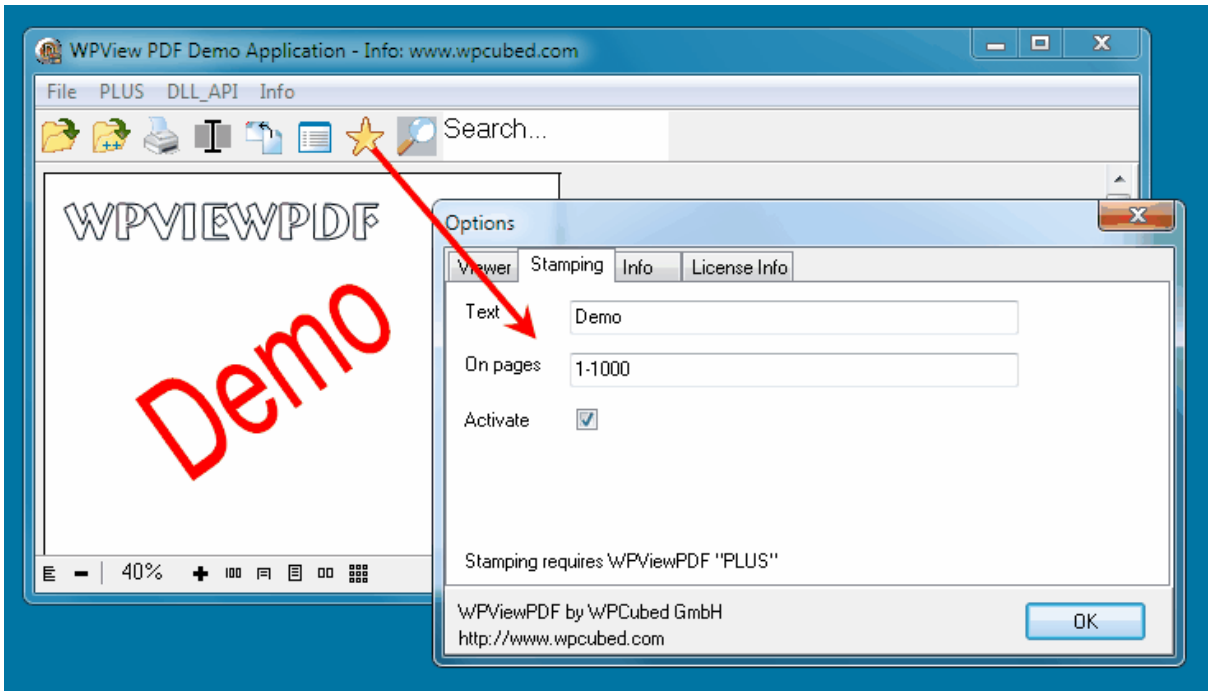
 Using the star icon the property dialog can be shown.

 This buttons open the field property dialog. The fields which are contained in the document will be listed. With WPViewPDF "PLUS" it is also possible to modify the texts!

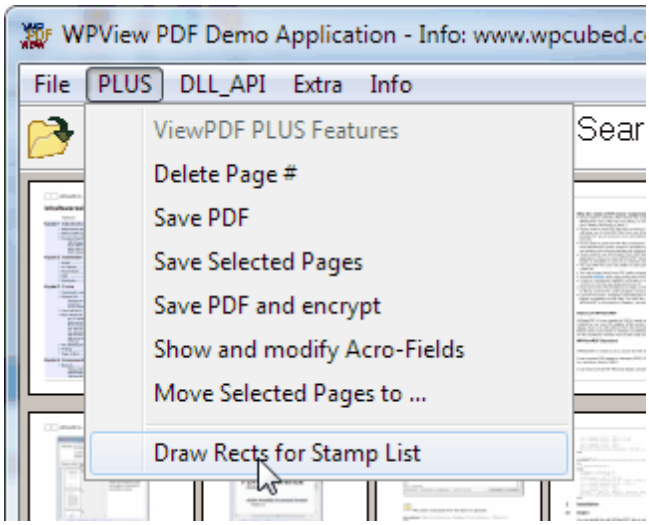
You can enter your license data in this dialog and also change the renderer for the PDF pages.

It is also possible to view the PDF document information.

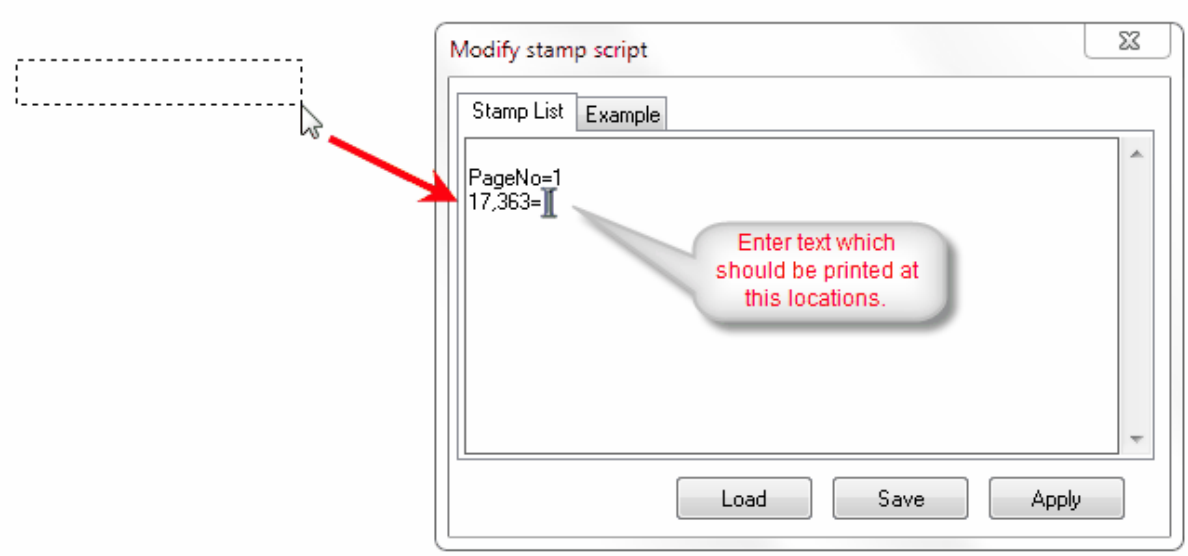
In the property dialog, in case WPViewPDF "Demo" or "PLUS" was used, the graphical stamping can be utilized. In this simple example just a rotated text is drawn on a metafile canvas. (Please note that currently only simply text and vector drawing is allowed using metafile stamping. Images cannot be used. All text will be converted to vectors)



You can try out the second stamping method available in WPViewPDF PLUS using this menu:



After a click on this menu you can draw a rectangle on the page.
The script dialog will be displayed to edit the stamping script. After a rectangle has been drawn, a new position will be added to the end to let You enter some text for this position. You can also select the "Example" tab, to try that out using the "Apply" button.

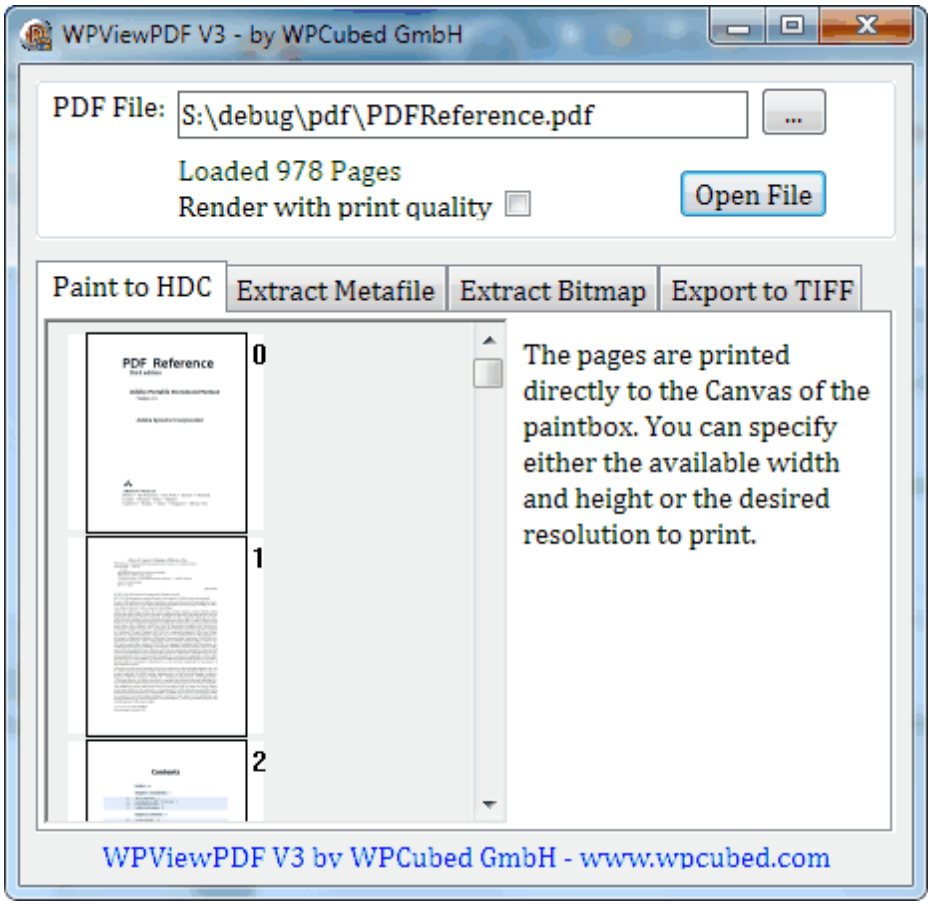


After Apply the pages will be updated at once. When the document is saved, the stamped text will be saved with it.

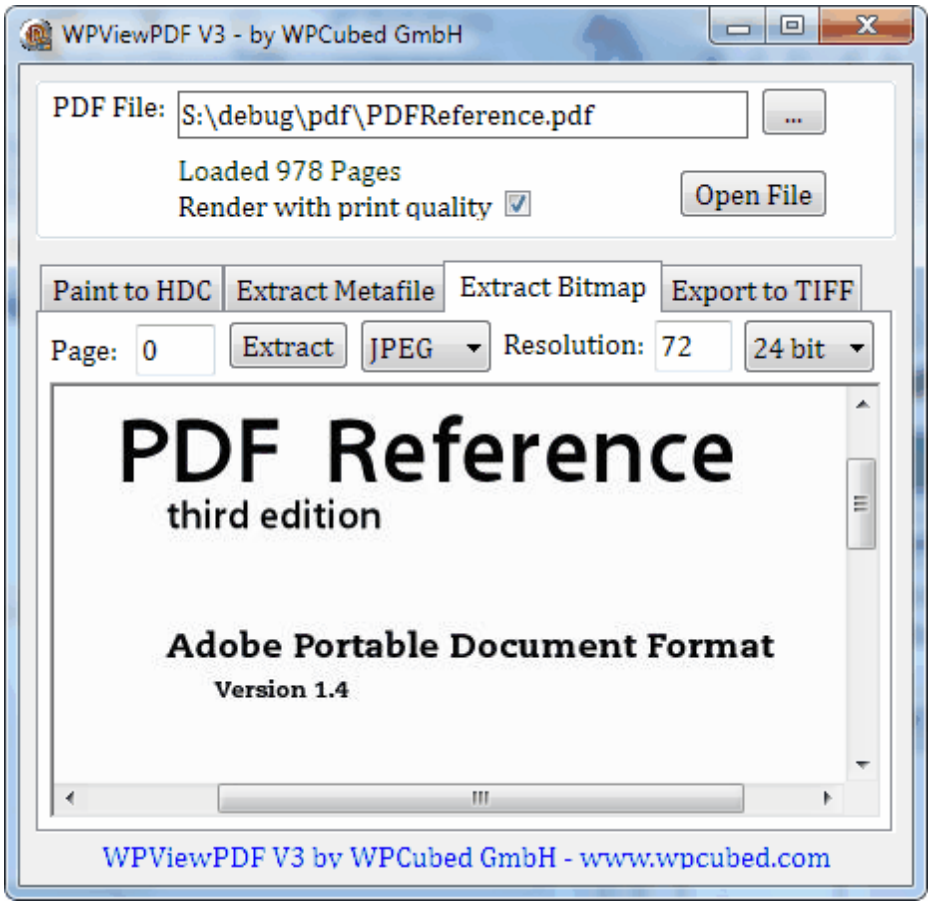
With WPViewPDF PLUS You can also move pages after a certain page ("0" would be the start). To do so select one or more pages (usually with the right mouse button) and click on "Move Selected Pages ..." to enter the number.

1.3.3 Delphi: PDF to Bitmap

The demo PDFImgExtract shows how to use PrintHDC and extract bitmap methods



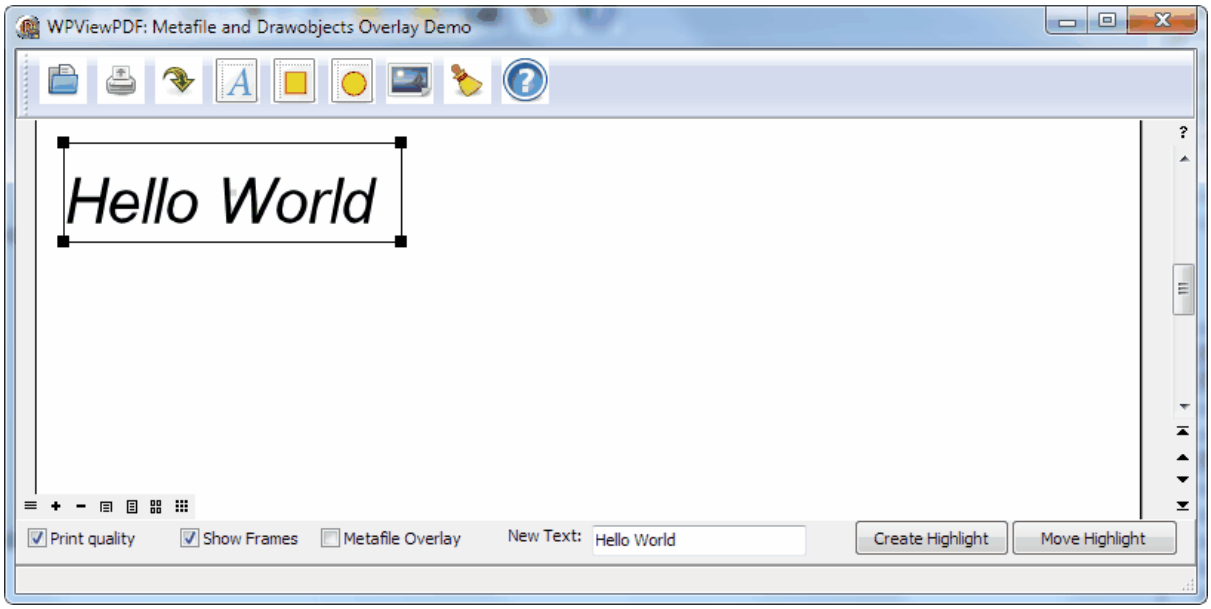
Please select a PDF file first and then click on "Open File" to actually load it. You can test the "print renderer" (default) and the bitmap renderer.



1.3.4 Delphi: Add graphics to PDF

The demo MetaOverlay let You try out the graphic objects. You can add text, rectangle and circle objects.

With WPViewPDF PLUS you can save the data and the objects will be permanently added to the PDF.



 This code is executed when the button is pressed:

```
procedure TMetafileOverlay.DrawRectClick(Sender: TObject);
var
  t: TPDFDrawObjectRec;
begin
  FillChar(t, SizeOf(t), 0);
  t.ColorBrush := clRed;
  t.Alpha := 100; // transparent
  t.grtyp := 1; // Rectangle
  ShowMyHint;
  WPViewPDF1.CommandStrEx(COMPDF_MouseAddOneDrawObject,
    'REDRECT', Cardinal(@t));
end;
```

It is also possible to create an object a specific position and to modify its properties after the object was created. The buttons "Create Highlight" and "Move Highlight" showcase this possibility:

```
// Create an object
procedure TMetafileOverlay.CreateHighlightClick(Sender: TObject);
var
  t: TPDFDrawObjectRec;
begin
  FillChar(t, SizeOf(t), 0);
  t.PageNo := 0; // Page 1
  t.ColorBrush := clYellow;
  t.Alpha := 100; // transparent
  t.grtyp := 1; // Rectangle
  // Position, 720 dpi
```

```
  t.units_xywh := 10; // 720 dpi
  t.x := Round( 2/2.54 * 720); // 2 cm
  t.y := Round( 3/2.54 * 720); // 3 cm
  t.w := Round( 5/2.54 * 720);
  t.h := Round( 1/2.54 * 720);
  WPViewPDF1.AddDrawObject(wpAddNow, 'YELLOW', t, nil, '');
end;

// and move it
procedure TMetafileOverlay.MoveHightlightClick(Sender: TObject);
var
  t: TPDFDrawObjectRec;
begin
  FillChar(t, SizeOf(t), 0);
  t.PageNo := 0; // Page 1
  t.units_xywh := 10; // 720 dpi
  t.x := Round( Random(10)/2.54 * 720); // move somwhere
  t.y := Round( Random(10)/2.54 * 720); //
  t.w := Round( 5/2.54 * 720);
  t.h := Round( 1/2.54 * 720);
  t.Fields :=  OBJFL_X + OBJFL_Y + OBJFL_W + OBJFL_H;
  WPViewPDF1.AddDrawObject(wpModifyExistingObj, 'YELLOW', t, nil, '');  //not: wpMoveExistingObj
end;
```

2 Installation

2.1 Delphi

You can install the unit WPViewPDF_reg.pas into a new package to register the new component TWPViewPDF or open the DPK file and click on "Install".

Alternatively add the units WPViewPDF3 and WPDF_ViewCommands to the project and create the component in code:

```
procedure TWPViewPDFDemo.FormCreate(Sender: TObject);
begin
  WPViewPDF1 := TWPViewPDF.Create(Self);
  WPViewPDF1.DLLName := dllname;
  WPViewPDF1.ViewerStart('', your_lic_name, your_lic_key, your_lic_code);
  WPViewPDF1.Parent := Self;
  WPViewPDF1.Align := alClient;
  WPViewPDF1.ViewControls := [wpHorzScrollBar, wpVertScrollBar];
  WPViewPDF1.ViewOptions := WPViewPDF1.ViewOptions +
    [wpExpandAllBookmarks, wpDontUseHyperlinks, wpSelectClickedPage, wpShowP
end;
```

2.2 C++ Builder

You can install the unit WPViewPDF_reg.pas into a new package to register the new component TWPViewPDF.

In the C++Builder Menu you can select "New ... Package". Into the newly created project add the unit WPViewPDF_reg.PAS. Also add the package vcl.bpi

and vclx.bpi which can be found in the CBuilder\Bin directory.

In the project options make this a designtime only package. Now you can save under a new name and compile and install.

We included a package project created with RAD Studio XE under the name WPViewPDFLIB.

2.3 Visual Studio

WPViewPDF also comes with a component to be used in .NET Forms application. The name of the assembly is PDFViewerLib.

There are different versions for the Demo, the regular and the PLUS edition. Please see directory "DotNET".

In the full version the source which was written in C# is also included. You can use this source to compile the assembly if you need it for a different framework version.

To use WPViewPDF drag the assembly to the toolbox. You can then drop one instance to the form.

Please copy the DLLS wPDFView03.dll and wp_type1ttf.dll to the executable directory.

Unless You use the demo version You need to set the license keys from the delivery (e-mail) using **ViewerStart**()

```
public Form1()
{
    InitializeComponent();
    // Set some properties
    pdfViewer1.ViewerStart("name", "xxx", 0);
}
```

To avoid redundancy this manual shows how to use the VCL in objectpascal and/or the dot-net assembly in C#.

Note:
In a **standard C or C++** VisualStudio program please call command
COMPDF_CPP_PROGRAM, 1
which translates to
SendMessage(WM_PDF_COMMAND, 1289, (LPARAM)1);
right after start.

2.4 VB6

We have included a new version of the OCX Interface ***ViewPDF03.ocx*** to work in legacy VB6 applications.
It makes use of the new methods included in WPViewPDF V3. Please do NOT use the OCX in .NET Applications.

If you do not need a PDF viewer but only the merge or print functionality it is better to access the DLL directly. You can import the required functions and access them without having to deal with the OCX interface.

To install it in VB6 please drag the OCX from the explorer to the tool palette.

Please make sure the engine DLL has been copied to your application directory.

The WPViewPDF setup also creates a registry entry with the installation directory path. This makes sure the ViewPDF engine can be loaded when the IDE is open. The OCX does not work if it cannot load the PDF engine.

You can use this code in Form_Load() to load the DLL and set the license information

```
DLLNAME = "{hkcu}Software\WPCubed\WPViewPDF\Path"
LICNAME = "" 'license info
LICKEY = "" 'license info
LICCODE = 0 'license info
WPViewPDFX1.ViewerStart DLLNAME, LICNAME, LICKEY, LICCODE
```

2.5 Distribution

You may distribute WPViewPDF 3 with Your application if all developers who were working (anywhere) on the project have a license for WPViewPDF 3.
(Or the company has a TEAM or SITE Licenses, depending on the count of developers)

To distribute You need to copy this 2 DLLs to the directory of Your application EXE:

wPDFView03.DLL, alternatively, wPDFViewPlus03.DLL
wp_type1ttf.dll

If wp_type1ttf.dll is required, if it was missing, WPViewPDF 3 will show a message on startup and will not render in best quality.

You need to provide your licenses codes to the component using ViewerStart, or, if You use Delphi, use a proper PDFLicenses.INC file. (Setup creates one for You)

You must not provide anybody with your licenses code or distribute any other

included files.

3 Tasks

3.1 Command() - execute procedures of WPViewPDF

WPViewPDF exposes all its methods through a set of methods which all mainly execute a command inside the library.

The command at least needs an ID as parameter, and, depending on the feature other parameters as integer, cardinal, character pointer or record pointer.

List of the commands

3.2 Change GUI

WPViewPDF 3 was created to be very easy to use. So it is possible to plug it into an application, run a few commands and are set for PDF view and print.

The control incorporates very small navigation and zoom controls. They are small but sufficient to select the desired operation.

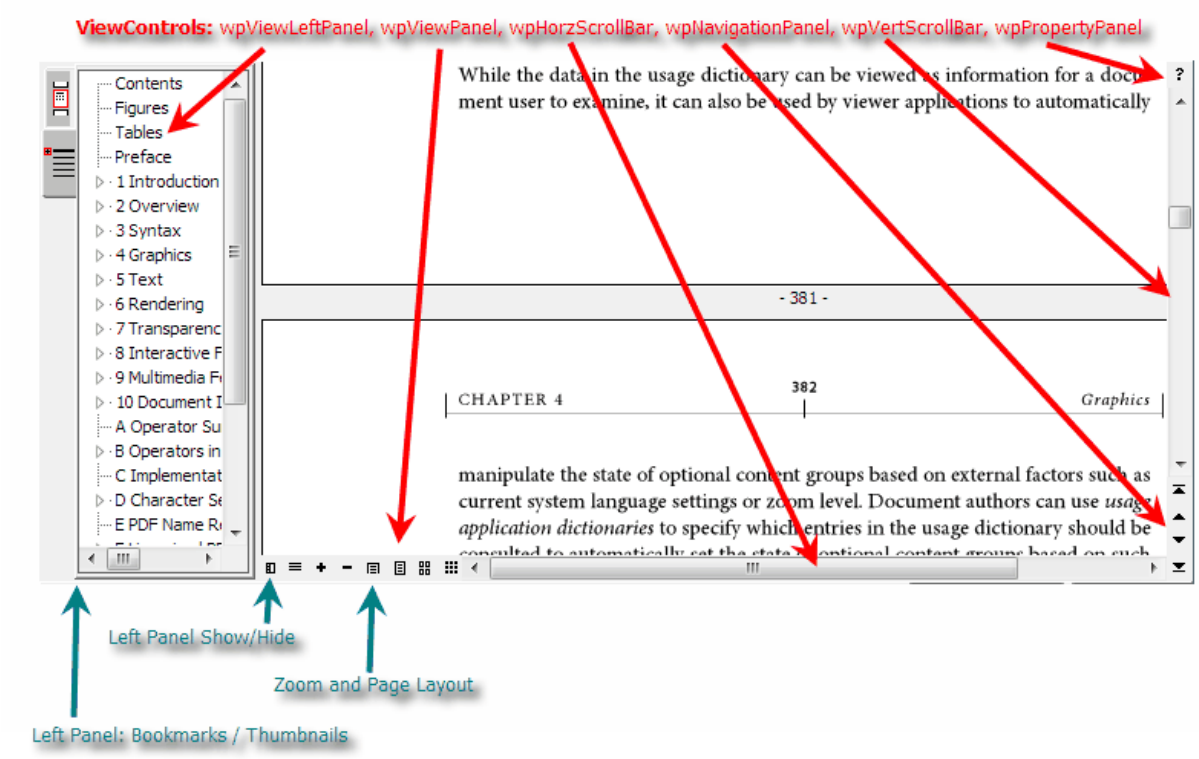
Of course it is possible to use own controls, not inside the viewer but outside in statusbar or toolbar.

You can switch the rendering engine as well.
By default it is using the "gdi renderer" which provides the best compatibility to many PDF files.
You can switch it off using command(COMPDF_UseGDIPainter, 0) and on using command(COMPDF_UseGDIPainter, 1).

If you switch the GDI renderer off, the redraw is faster, text is usually better aliased but complex clipping is not supported.
Printing will always use the GDI renderer.

3.2.1 ViewControls and ViewOptions

Using the property ViewControls (.NET enum `eViewControls`) You can select optional GUI elements.



VCL

```
WPViewPDF1.ViewControls := [wpViewLeftPanel, wpHorzScrollBar,
wpVertScrollBar,wpNavigationPanel, wpPropertyPanel, wpViewPanel];
```

.NET

```
pdfViewer1.ViewControls =
      eViewControls.wpHorzScrollBar |
      eViewControls.wpNavigationPanel |
      eViewControls.wpPropertyPanel |
      eViewControls.wpVertScrollBar |
      eViewControls.wpViewPanel;
```

The property ViewOptions (.NET type: eViewOptions) controls how the page is rendered and how the GUI elements work:

wpDontUseHyperlinks : Hyperlinks are ignored
wpDontHighlightLinks: Hyperlinks will not painted with a blue background
wpDontAskForPassword: When a PDF requires a password the control will not ask for one.
wpSelectPage: The user can select pages by pressing Ctrl+Cursor left/right
wpPageMultiSelection: like wpSelectPage

wpShowPageSelection: Display selected pages with blue frame
wpDisablePagenrHint: Don't display a page number during scrolling
wpDisableZoomHint: Don't display a zoome value during zooming
wpDisableBookmarkView: Do not load bookmarks
wpInactivateHyperlinks: Display hyperlinks but do not use the internal jump on clicks
wpExpandAllBookmarks: Expand all bookmarks
wpShowDeletionCross: Show pages which are marked for deletion with a cross
wpPaintCursor: (not used by WPViewPDF Standard and PLUS) Paint a cursor in PDF text paths
wpPaintPathRects: Show rectangle around text paths
wpPaintObjectsRects: Show frames for all draw objects
wpPaintObjectsSizers: Show sizer rectangles when a draw object is selected
wpHighlightFields: Show colored backgrounds for fields (widget annotations)
wpViewThumbnails: Display thumbnails in left panel


```
WPViewPDF1.ViewOptions :=
   [wpExpandAllBookmarks,
   wpSelectPage,
   wpShowPageSelection,
   wpPageMultiSelection];
```


```
pdfViewer1.ViewOptions =
   eViewOptions.wpExpandAllBookmarks |
   eViewOptions.wpExpandAllBookmarks |
   eViewOptions.wpSelectPage |
   eViewOptions.wpShowPageSelection;
```

You can also select the background color for the viewer.

Use this commands:

COMPDF_SETDESKCOLOR (=53): select the color for the background
COMPDF_SETDESKCOLORTO (=59): select the bottom color for the background. If it was specified, the background will use a marquee effect.

COMPDF_SETPAPERCOLOR (=54): Select the paper color. The standard is clWhite.

You can also hide the page frame (thin black line round paper) or show the page numbers.

COMPDF_SetExViewOptions (=81) requires a bitfield::

1: Show Page Numbers in main viewer (default: no page numbers)
2: Hide Page Frames in main viewer (default: frames)
4: FastZoom Mode in main viewer (default: off)
16: Hide Page Numbers in thumbnail viewer (default: display page numbers)
32: Hide Page Frames in thumbnail viewer (default: frames)
64: FastZoom Mode in thumbnail viewer (default: off)

COMPDF_SetPageNumberString

This command can be used to set a format string for the page number display. Default is " %d "

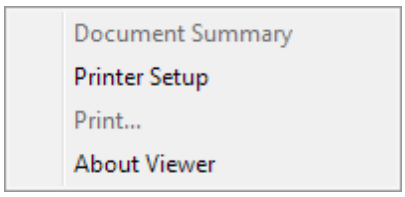
An alternative would be "Page %d of %d" to display "Page 1 of 100" under pages.

COMPDF_ShowNavigation = 134

This command can be used to force the display of the navigation panel (Bookmarks and Thumbnails).
Use IntPar=0 to hide it, 1 to show it and 2 to toggle its visibility.

3.2.2 Localization

Localize menu texts:



the displayed strings can be controlled with this commands:

COMPDF_SetDocumentProperties
COMPDF_SetPrintSetup
COMPDF_SetPrint

COMPDF_SetShowAbout

Example:


```
WPViewPDF1.CommandStr(COMPDF_SetDocumentProperties, 'Eigenschaften')
```


```
pdfViewer1.Command(commands.COMPDF_SetDocumentProperties, "Eigenschaften");
```

```
Activate the hints:

  WPViewPDF1.CommandStrEx(COMPDF_SetShowHint,'1', pdf_hint_ONOFF);

The hints for the zoom panel can be localized with this code:

  WPViewPDF1.CommandStrEx(COMPDF_SetShowHint,'bookmarks',
pdf_hint_LeftPanel);
  WPViewPDF1.CommandStrEx(COMPDF_SetShowHint,'bookmarks',
pdf_hint_LeftPanel);
  WPViewPDF1.CommandStrEx(COMPDF_SetShowHint,'100%', pdf_hint_Zoom100);
  WPViewPDF1.CommandStrEx(COMPDF_SetShowHint,'zoom in', pdf_hint_ZoomIn);
  WPViewPDF1.CommandStrEx(COMPDF_SetShowHint,'zoom out', pdf_hint_ZoomOut);
  WPViewPDF1.CommandStrEx(COMPDF_SetShowHint,'page width',
pdf_hint_ZoomWidth);
  WPViewPDF1.CommandStrEx(COMPDF_SetShowHint,'full page',
pdf_hint_ZoomPage);
  WPViewPDF1.CommandStrEx(COMPDF_SetShowHint,'two pages',
pdf_hint_ZoomTwoPages);
  WPViewPDF1.CommandStrEx(COMPDF_SetShowHint,'thumbnails',
pdf_hint_ZoomThumbnails);
```

3.2.3 Create a toolbar

You can start all functions using the command method.

The following IDs can be used

A) Show dialogs:

COMPDF_DocumentProperties (=1) - display the window with PDF property
COMPDF_ShowAbout (=6) - display the WPViewPDF Info window
COMPDF_PrinterSetup (=30) - display the printer setup.
COMPDF_PrintDialog (=32) - display the print dialog. The user may change the printer.

B) Goto certain positions in the PDF

```
COMPDF_GotoFirst        = 20; // Goto first page
COMPDF_GotoPrev         = 21; // Goto Previous page
COMPDF_GotoPage         = 22; // Goto Page Nr in parameter
COMPDF_GotoNext         = 23; // Goto next page
COMPDF_GotoLast         = 24; // Goto last page. Pass 1 as parameter to go to end of page.
COMPDF_ShowGotoPage     = 25; // Show page nr editfield (RESERVED)
COMPDF_ShowGotoBookmark = 26; // Show bookmark edit (RESERVED)
COMPDF_GotoYPos         = 27; // Goto 'B' as y in 72 dpi (also see GetYpos!)
COMPDF_GotoXPos         = 28; // Goto 'B' as x in 72 dpi
COMPDF_ScrollXY         = 29; // Bit 1: Horz, Bit 2: Large, Bit 3=Next
```

C) Control Zooming

```
COMPDF_Zoom100          = 41; //! 100 % Zoom
COMPDF_ZoomIn           = 42; //!  + 10%
COMPDF_Zoom             = 43; //! Zoom to StrPar/IntPar - if IntPar=0 retrieve zoom! If StrPar<>'' the set to zooming mode
COMPDF_ZoomOut          = 44; //!  - 10%
COMPDF_ZoomFullWidth    = 45; //! Page Width
COMPDF_ZoomFullPage     = 46; //! Page Width
COMPDF_ZoomTwoPages     = 47; //! Toggle 2 Pages Display
COMPDF_ZoomThumbs       = 48; //! Thumbnail Preview
```

To red the current zooming use

```
COMPDF_ZoomGetCurrent   (= 49); //! read current zoom
```

3.3 Load and Save

WPViewPDF can load the PDF from file and from stream.

Load a file. If an error happens return false, otherwise true.

```
function LoadFromFile(const filename: string): Boolean;
```

Load file completely - close the file stream afterwards.

function LoadFromFileAsCopy(const filename: string): Boolean;

Append a file to the currently loaded.

function AppendFromFile(const filename: string): Boolean;

Load PDF from a stream. Optionally clear the already loaded data.

function LoadFromStream(Stream: TStream ; WithClear : Boolean = false): Boolean;

Attach a stream - the stream will be used while the PDF is accessed.

function AttachStream(Stream: TStream): Boolean;

WPViewPDF PLUS can also save the PDF data

It is also possible to save in RTF, TXT and HTML format!

This methods are located in the sub interface "Plus"

function SaveToFile(const filename: WideString): WordBool;

function SaveSelectionToStream(Stream: TStream; FileExt : AnsiString = ''): WordBool;

Hint: To save only certain pages as PDF without having to use a selection use arange form-to. The numbers are 1 based to make it easier provide a user interface.

```
SaveSelectionToStream(stream, 'from-to;PDF')
```

function SaveToStream(Stream: TStream; FileExt : AnsiString = ''): WordBool;

Also see Load PDF commands.

function **CheckOwnerPassword** can be used to pass an owner password to lift **save restrictions**. TRUE is returned if the password was accepted.

function MaySave can be used to check if the PDF file may be saved.

Please note: If security settings of a PDF file forbid saving, the component will not save. You as developer can override this at Your own risk. Use command

(COMPDF_DisableSecurityOverride,1) to disable this check.

When extracting text from a PDF file WPViewPDF will first sort the text element using their horizontal coordinate. This can be switched off using COMPDF_TextExtractDontSort.

It is possible to disable saving using command(COMPDF_DisableSave). It is not possible to enable it again.

If your application allows fast scrolling between files we I suggest to use PostMessage to uncouple the update of the viewer from the scrolling. So the users can scroll fast but the viewer does not have to load a new file each time.

3.4 Draw Shapes on PDF

WPViewPDF Version 3 introduces "Draw Objects".
This objects can rectangle and circle shapes with different color and transparency. Also possible are single text lines and images.
It is possible to move the objects under program control or the user can move and resize the object.

With the PLUS edition the shapes can be rendered into the PDF and saved as a new PDF file.

You can use this feature to highlight certain areas on the PDF file, for display or print.

To create a shape this commands can be used:
COMPDF_AddDrawObject = 518
COMPDF_MouseAddDrawObject = 520
COMPDF_MouseAddOneDrawObject=521

Hint: Using COMPDF_DrawObjectLocateAtXY it is possible to get the name of the object under the mouse pointer. This makes it possible to create sensible areas on a PDF page, i.e. buttons.

"AddDrawObject" will immediately add a shape to a certain page.
"MouseAddDrawObject" will switch the cursor in a special mode which lets the user draw a rectangle. After the rectangle has been drawn, the shape will be created. The user can then draw another object, unless "Mouse*AddOne* DrawObject" was used, then the mouse switches into selection mode.

This method can be used to create objects. They wrap the call of the command.

```
procedure AddDrawObject(
        Mode : TWPAddDrawObjectMode;
        Name : WideString;
```

```
var Param : TPDFDrawObjectRec;
data : TMemoryStream = nil;
StrParam : WideString = '');
```

Mode can have this values:
wpAddNow - Add a new object at once
wpDrawAndAdd - select the object draw mode. The user can draw a rect and a new object will be created
wpDrawAndAddOne - like wpDrawAndAdd but only one object will be created. The viewer goes then in select mode
wpMoveExistingObj- Don't add an object. Adds to the X,Y W and H properties.
wpModifyExistingObj- Don't add an object. Modifies the named object according to the bitfield "fields"

Name is optional. It is the name of the shape which makes it possible to access it later.

Param is a record of type TPDFDrawObjectRec. It should hold the required values for color and type, but no attached data.

data is reserved.

StrParam is used for text.

The overloaded method allows the data top be passed as pointer:

procedure AddDrawObject(Mode : TWPAddDrawObjectMode; Name : WideString; var Param : TPDFDrawObjectRec; StrParam : WideString; data : PAnsiChar=nil; datalen : Integer = 0); overload;

Example:
This Delphi dialog will let the user select an image file. Now he or she may draw a rectangle on the page where the image will be displayed:

```
procedure TMetafileOverlay.DrawJPEGClick(Sender: TObject);
var
  t: TPDFDrawObjectRec;
  i: Integer;
begin
  if OpenPictureDialog1.Execute then
  begin
    i := WPViewPDF1.Plus.AddImage(OpenPictureDialog1.FileName);
    if i > 0 then
    begin
      FillChar(t, SizeOf(t), 0);
```

```
      t.grtyp := 20; // Image
      t.typparam := i;  // Image ID
      t.ColorBrush := $B0B0B0; // gray
      t.ObjectOptions := OBJGR_KEEP_ASPECTRATIO+ OBJGR_OPAQUE;
      t.Padding := 100; // Padding in 1/10 Point
      //t.Angle := 30;
      ShowMessage('Please draw rectangle ...');
      WPViewPDF1.CommandStrEx(COMPDF_MouseAddOneDrawObject, '', Cardinal(@t)
    end
    else
      ShowMessage('Cannot load image');
  end;
end;
```

3.4.1 Record TPDFDrawObjectRec

The commands to create a shape require as parameter a pointer to the record TPDFDrawObjectRec.

It has this basis elements:
structsize - should be initialized as structsize = SizeOf(TPDFDrawObjectRec)
PageNo - the page number the shape should be created on (0 = first)
x,y,w,h - for COMPDF_AddDrawObject - the position from the upper left corner in points (1/72 inch). Note: A different resolution can be selected using the parameter **units_xywh**.

ColorBrush - the RGB color of the background
ColorPen - the RGB color for the outline
ColorText - the RGB color for the text
PenWidth - the width of the outline in pt*100
Alpha - the transparency in the range 0..255. 255 and 0 are solid.
Angle - the angle, used for text only
Padding - padding inside the bounds - used for images.

grtyp - select the shape type.
0=default highlight (alpha=120)
1=rectangle
2=circle
3=ellipse
20= Image. Use typparam as ID of the image. (COMPDF_AddJPEG)
100= Text.

ObjectOptions - this is a bitfield to change attributes of object
1 : Keep aspect ratio when adapting the size of image JPEG to the bounding

box
2 : Stretch text to fill the rectangle
4 : Center text horizontally in the box
8 : Used for text and JPEGs . Draw Background in selected Brush Color and Pen.
16 : Apply ColorBrush after painting the object using color multiplication to create highlight rectangles.
This mode is only effective on screen, when rendering to PDF regular transparency will be used.
The Alpha property should be also used.
32 : Once the object was created it cannot be moved anymore
64 : The size of the object cannot be changed by the user

HRad, VRad - the vertical and horizontal radius to make rectangles round. (always Uses 720 dpi)

Other elements are either reserved or used only for certain objects.

At the end of the structure binary or text data can be stored. The offset to the data and the length has to be provided using this parameters. If you use the API **AddDrawObject()** You do not have to worry about this.

textoff - the offset to the text data - this must be unicode text
textlen - the length of the text data.

NameOff - the offset to the name using wide characters.
NameLen - the length of the name.

DataOff, Datalen - various data. DataTyp tells which:
1 = ANSI Text
2 = Unicode Text

The API AddDrawObject simplifies the use of the record since it copies the extra data. In the VCL it is implemented like this:

```
procedure TWPViewPDF.AddDrawObject(Mode : TWPAddDrawObjectMode; Name : WideString;
                           var Param : TPDFDrawObjectRec; StrParam : WideString;
                           data : PAnsiChar=nil; datalen : Integer = 0);
var t : PPDFDrawObjectRec; tl, i : Integer;
    p : PByte;
begin
   tl := SizeOf(TPDFDrawObjectRec);
   if Name<>'' then  inc(tl, Length(Name)*SizeOf(WideChar));
   if StrParam<>'' then  inc(tl, Length(StrParam)*SizeOf(WideChar));
```

```
    if data<>nil then
    begin
      if datalen=0 then datalen := StrLen(Data);
      inc(tl, datalen);
    end;
    GetMem(t, tl);
    t^ := Param;
    t.structsize := tl;
    try
      p := PByte(t);
      i := SizeOf(TPDFDrawObjectRec);
      inc(p, i);
      if Name<>'' then
      begin
         t.NameOff := i;
         t.NameLen := Length(Name);
         Move(Name[1], p^, t.NameLen*SizeOf(WideChar) );
         inc(i, t.NameLen*SizeOf(WideChar) );
         inc(p, t.NameLen*SizeOf(WideChar) );
      end else t.NameOff := 0;
      if StrParam<>'' then
      begin
         t.textoff := i;
         t.textlen := Length(StrParam);
         Move(StrParam[1], p^, t.textlen*SizeOf(WideChar) );
         inc(i, t.textlen*SizeOf(WideChar) );
         inc(p, t.textlen*SizeOf(WideChar) );
      end else t.textoff := 0;
      if data<>nil then
      begin
         t.DataOff := i;
         t.Datalen := datalen;
         Move(data^, p^, datalen );
      end else t.DataOff := 0;
      case Mode of
         wpAddNow:         CommandStrEx(COMPDF_AddHighlightRect, Name, Cardinal(t));
         wpDrawAndAdd:     CommandStrEx(COMPDF_MouseAddDrawObject, Name, Cardinal(t));
         wpDrawAndAddOne: CommandStrEx(COMPDF_MouseAddOneDrawObject, Name, Cardinal(t));
         wpMoveExistingObj: CommandStrEx(COMPDF_ModifyDrawObjectPos, Name, Cardinal(t));
         wpModifyExistingObj: CommandStrEx(COMPDF_SetDrawObjectProp, Name, Cardinal(t));
      end;
    finally
      FreeMem(t);
    end;
end;
```

The .NET assembly also defines this method. (data is not used yet)

```
public void AddDrawObject(AddDrawObjectMode Mode, string Name, TPDFDrawObjectRec
Param, string Text)
```

3.4.2 Delete and modify shapes

The shapes can be removed with the command COMPDF_ClearDrawObjects = 519.

The API ClearDrawObject can also be used, it wrapps this command:

```
procedure TWPViewPDF.ClearDrawObject(PageNo  : Integer = -1; typselect : Integer = -1);
begin
   CommandStrEx(COMPDF_ClearDrawObjects, IntToStr(typselect), Cardinal(PageNo));
end;
```

The parameters are:
PageNo : the page number, -1 for all
typselect : what should be selected. Any positive number deletes only the objects of a certain grtyp.
-1 delete all,
-2 delete only the selected.

You can also use the overloaded method and pass the name of the shape to be deleted. It will be found on all pages if PageNo is -1.

It is possible to modify a shape using AddDrawObject(**wpModifyExistingObj** , ..)

To use this method set in the TPDFDrawObjectRec record all parameters You need to change. Then add a bit for each element which should be changed to the element Fields.

VCL Example:

```
var
  t: TPDFDrawObjectRec;
begin
  FillChar(t, SizeOf(t), 0);
  t.PageNo := 0; // First Page
  t.units_xywh := 10; // 720 dpi
  t.x := Round( Random(10)/2.54 * 720); // move somewhere
  t.y := Round( Random(10)/2.54 * 720); //
  t.w := Round( 5/2.54 * 720);
  t.h := Round( 1/2.54 * 720);
  t.Fields :=  OBJFL_X + OBJFL_Y + OBJFL_W + OBJFL_H;
  WPViewPDF1.AddDrawObject(wpModifyExistingObj, 'SHAPE_NAME', t, nil, '');
end;
```

It is also possible to move an object to a different page.

If you need to move an object to a position in relation to its current position use

wpMoveExistingObj instead of wpModifyExistingObj.

You can use COMPDF_DrawObjectLocateAtXY to check for an object at a certain mouse X,Y position and COMPDF_DrawObjectReadProp to retrieve its position in points.

```
procedure TMetafileOverlay.FormMouseMove(Sender: TObject; Shift: TShiftState; X,
  Y: Integer);
begin
   StatusBar1.SimpleText := '-' +
      WPViewPDF1.CommandGetStr(COMPDF_DrawObjectLocateAtXY, '', Cardinal(-1)) +
      '@' +
      IntToStr( WPViewPDF1.Command(COMPDF_DrawObjectReadProp, 1) )  + ',' +
      IntToStr( WPViewPDF1.Command(COMPDF_DrawObjectReadProp, 2) );
end;
```

3.4.3 Render Shapes into PDF

If you need to save the objects with the PDF you need to call the command COMPDF_RenderDrawobjects.

The parameter is a bit field. The following bits are used
1: RenderAsAnnotations (reserved)
2: UpdateAnnotations (reserved)
4: RenderAsPaths,
8: DeleteRenderedObjects,
16: DeleteAnnotations - requires WPEditPDF
32: UnderPageLayer,
64: OverPageLayer

The objects are not deleted - use WPViewPDF1.ClearDrawObject(-1, -1); to delete all shapes.

3.4.4 VCL: Example - highlight rectangle

Draw a highlighted rectangle at a certain position:

```
var
  t: TPDFDrawObjectRec;
begin
  FillChar(t, SizeOf(t), 0);
  t.PageNo := 0; // Page 1
  t.ColorBrush := clYellow;
  t.Alpha := 100; // transparent
  t.grtyp := 1; // Rectangle
  t.ObjectOptions := 16; // Use multiply transparency
  // Position, 720 dpi
  t.units_xywh := 10; // 720 dpi
  t.x := Round( 2/2.54 * 720); // 2 cm
  t.y := Round( 3/2.54 * 720); // 3 cm
```

```
  t.w := Round( 5/2.54 * 720);
  t.h := Round( 1/2.54 * 720);
  WPViewPDF1.AddDrawObject(wpAddNow, 'YELLOW_RECT', t, nil, '');
end;
```

Move that rectangle to a different position:

```
var
  t: TPDFDrawObjectRec;   pw : Double;
begin
  FillChar(t, SizeOf(t), 0);
  t.PageNo := 0; // Page 1
  t.units_xywh := 10; // 720 dpi
  t.x := Round( Random(10)/2.54 * 720); // move somewhere
  t.y := Round( Random(10)/2.54 * 720); //
  t.w := Round( 5/2.54 * 720);
  t.h := Round( 1/2.54 * 720);
  t.Fields :=  OBJFL_X + OBJFL_Y + OBJFL_W + OBJFL_H;
  WPViewPDF1.AddDrawObject(wpModifyExistingObj, 'YELLOW_RECT', t, nil, '');
end;
```

Note: If you use wpMoveExistingObj instead of wpModifyExistingObj the values of X,Y,W,H and PageNo are added to the current values of this properties.

3.4.5 VCL: Example: Text at mouse position

At the end of the example code a font dialog is opened to let the user change a font.

```
var
  t: TPDFDrawObjectRec;
  s : AnsiString;
begin
  FillChar(t, SizeOf(t), 0);
  t.grtyp := 100;
  t.typparam := 2000; // Textfield, Height = 20
  t.ColorText := ColorToRGB( clBlue ); // Text Color
  t.ColorPen := ColorToRGB( clYellow ); // Background Color
  t.ObjectOptions := 4+8; // Center Text + Opaque
  t.ColorBrush := clYellow;
  // Get the page number
  t.PageNo :=  WPViewPDF1.command(COMPDF_GetPageUnderMouse);

  // Position of MOUSE on the page:
  t.x := WPViewPDF1.command(COMPDF_GetPageLogX);
  t.y := WPViewPDF1.command(COMPDF_GetPageLogY);
  t.h := 72;
  t.w := 72*3;

  t.Angle := 45;
```

```
  t.FontSize := 55*100;

  if FontDialog1.Execute then
  begin
     s := '"Font=' + FontDialog1.Font.Name + '"';
     WPViewPDF1.AddDrawObject(wpAddNow, '', t, 'This text in mouse position'
  end;
end;
```

3.4.6 .NET C# Example: Add text or rectangle

```
private void Add_a_rect_MenuItem_Click(object sender, EventArgs e)
        {
            TPDFDrawObjectRec rec = new TPDFDrawObjectRec();
            rec.grtyp = 1;
            rec.x = 100;
            rec.y = 100;
            rec.w = 100;
            rec.h = 100;
            rec.ColorBrush = 0xff0000; // blue
            pdfViewer1.AddDrawObject(AddDrawObjectMode.AddNow, "", rec, ""
);
        }

private void Add_a_text_MenuItem_Click(object sender, EventArgs e)
        {
            TPDFDrawObjectRec rec = new TPDFDrawObjectRec();
            rec.grtyp = 100;
            rec.typparam = 2000;
            rec.x = 100;
            rec.y = 100;
            rec.w = 100;
            rec.h = 100;
            rec.ColorBrush = 0xff0000;
            pdfViewer1.AddDrawObject(AddDrawObjectMode.AddNow, "", rec,
"Some Text");
        }
```

3.4.7 VB6 add rtectangle and text

The ActiveX defines the method AddDrawObject a little different. Here you have to pass the parameters to the function and not in a record:

Add Text:

```
WPViewPDFX1.AddDrawObject DrawAndAddOne, "", 0, 0, 0, 0, 0, 100, 0, 0, 0,
```

255, 3, 0, 0, 0, "HALLO"

Add a rectangle (the user has to draw a rectangle)

```
WPViewPDFX1.AddDrawObject DrawAndAddOne, "", 0, 0, 0, 0, 0, 1, 0, 0, 0,
255, 3, 0, 0, 0, ""
```

3.4.8 AddImage

This method prints (stamps) a JPEG image which was embedded by AddImage / command COMPDF_AddJPEG:

function **Plus.UseImage**(const ImageID, PageNo: Integer; x, y, w, h,
angle: Integer; PosMode: TWPImagePosMode) : Boolean;

The same can be done with command COMPDF_ImagePrint

Parameters:

const ImageID:	the id returned by AddImage (value is > 0!)
PageNo:	the page number (1..)
x, y, w, h:	the position and size in measured 72 dpi or values in %
angle:	an optional angle in degree
PosMode:	the position mode:

This set includes flags which change the way the image is positioned. It is possible to specify the width as % of the page width and also center an image to the page.

1: wpAtPageHorzCenter	Center the Image horizontally
2: wpAtPageVertCenter	Center the Image vertically
4 : wpPageWidthPC	Set width as % value of Page Width
8: wpPageHeightPC	Set height as % value of Page Width
16: wpFillPageAspectRatio	Fill the page with image but keep w/h aspect ratio
32: wpAtPagePageRight	Use x as offset from the right of page
64: wpAtPageBottom	Use y as offset from bottom from page
128: wpTilePage	Tile the image on the page (only use w and h)
256: wpUnderPage	place the image under the page text, default is above text.
512: wpXYIsImageCenter	Use the passed x,y as center of the image
1024: wpRotateToPage	Rotate the image in the same direction as the page

This method can be also called using command COMPDF_ImagePrint = 321. This cammand requires a structur as parameter:

```
TPDFPrintImageRec = struct
{
   int ImageID;
   int PageNo;
   int x,y,w,h;
   int PosMode;
   int angle;
}
```

PosMode is handled as bitfield (wpAtPageHorzCenter=1, wpAtPageVertCenter=2 ... wpUnderPage=256)

Exampe:
Result := View.CommandEx(COMPDF_ImagePrint, (DWORD) @PDFPrintImageRec);

3.5 Use stamping script (COMPDF_StampText)

WPViewPDF **PLUS** has the ability to use a simple script to add text in different colors, font faces and sizes to defined positions on certain PDF pages.

Using the "stamping" feature it is possible to add texts to existing PDF files.

This can be useful to add information while printing PDF files or to add data permanently, i.e. fill out a form or contract.

The data is added incrementally, this means normally each subsequent output is added to the existing.

The following command is used to add the script:

COMPDF_StampText

It just requires a string parameter. The string parameter is expected to be with a string list with strings separated by CR+NL.

If you build such a list in a TStringList object, You can use the "Text" property to read a string which can be used as parameter.

Example:

```
WPViewPDF1.CommandStrEx(
                COMPDF_StampText,
                MyParamStrings.Text,
                0
               )
```

Note: With the .NET assembly write Command(commands.

COMPDF_StampText, ...)

The script can use this commands:

Change page numbering format when using macros.
NUMFORMAT=*x*
Possible values for x are:
1 this creates arabic numbers (default)
i this creates lowercase roman numbers
I create upper case roman numbers
a create lowercase letters, i.e. a b c d
A create uppercase letters, i.e. a b c d

Set a Pagenumber offset (default = 0)
NUMOFFSET=x
The offset added to the page number and the page count.

Important: NUMFORMAT and NUMOFFSET must be used before selection a range of pages using PageNo=.

Selects one ore more pages for the following output.
PAGENO=...

N is a page number between 1 and count of pages. Also possible are ranges and the text "ALL" to change all pages.
PageNo=N
PageNo=A-B
PageNo=N1,N2,N3,A-B
PageNo=ALL

This command removes all output from the currently selected page or pages.
@ClearText

Using the color command it is possible to set the color as RGB (0..1) values, i.e. red:
Color=1 0 0
Color=0 0.1 1

Select the font
Font=Arial
Font=Courier New

Select the coordinate origin - values are 0 - 4. This is useful to add page numbering in a certain distance from the page margin without knowing page size.

Origin=0 -> top left of the page, default
Origin=1 -> top right
Origin=2 -> bottom right
Origin=3 -> bottom left

Switch off macros (see below)
MACROS=off

Texts are printed like this:
X,Y=text

X and Y is the position of the start point in point coordinates (72 dpi) relatively to the Origin (default = top - left)
72,72=Text at one/one inch

Please note: The *rotation* specified for the PDF page is not evaluated!

The following macros are understood unless "MACROS=off" was used:
[#] print the page number
[##] print the page count
[N] print a running number in the current range. (@RESETNR will set this to 1)

Tip: The example program PDFView uses WPViewPDF1.CommandEx (COMPDF_SelectMode, 2); to activate the rectangle drawing mode in the viewer. After the user has drawn a rectangle the event OnSelRectEvent to add a X,Y position parameters to a stringlist. This makes it easy to locate the correct positions if You need to fill out a form.

```
procedure TWPViewPDFDemo.DoSelRectEvent(Sender: TObject; const PageNr : Integer; R : TRect);
begin
  StampText.SetPageNo(PageNr+1); // add PageNo=... if it was not there
  StampText.StampList.Lines.Append( IntToStr(R.Left) + ',' + IntToStr(R.Bottom) + '=' );
  StampText.Show;
end;
```

3.5.1 Example: Add Page numbers

You can use this script to add page numbers. The first 3 pages will use Roman numbers, the subsequent Arabic.

@Page nubering part 1 - roman
NUMFORMAT=I
PageNo=1-3
ORIGIN=2
-40,-25=[#]
@Page nubering part 2 - arabic

NUMFORMAT=1
NUMOFFSET=-3
PageNo=4-9999
ORIGIN=2
-40,-25=[#]

3.6 Printing

WPViewPDF makes it easy for You to print PDF files from your application.

Please note: If security settings of a PDF file forbid printing, the component will not print. You as developer can override this at Your own risk. Use command (COMPDF_DisableSecurityOverride,1) to disable this check.

You can disable printing globally by using command(COMPDF_DisablePrint). It is not possible to enable it again!

This commands control printing: Printing (on paper)

This commands allow printing on HDC: Printing (on device)

Also see pdfPrint()

3.7 Page rotation

It is possible to rotate certain or all pages by increments of 90 degrees or to the angle 0,90,180 and 270.

This can be done with command COMPDF_RotatePage

it expects 2 parameters:
a) a string parameter
- a page number in the range 1...pagecount, i.e. "1"
- a page number list
- "selected" to modify the selected pages
- "all" to modify all pages

b) the rotation angle
either +- 90, +-180 or
1, 2, 3 or 4 * 90

Example:

```
for i:=1 to 10 do
WPViewPDF1.CommandStrEx(COMPDF_RotatePage, IntToStr(i),  90);

does the same as
WPViewPDF1.CommandStrEx(COMPDF_RotatePage, '1-10',  90);
```

Hint:

You can rotate the selected pages using

```
  WPViewPDF1.CommandStrEx(COMPDF_RotatePage, 'selected',  90);
```

To disable/enable this action use the event OnViewerMessage:

```
procedure TWPViewPDFDemo.DoViewerMessage(
  Sender: TObject;
  var ID : Integer;
  Param: Integer);
begin
  case ID of
...
    MSGPDF_CHANGESELPAGE:
    begin
      RotateAction.Enabled := Param>0;
    end;
  end;
end;
```

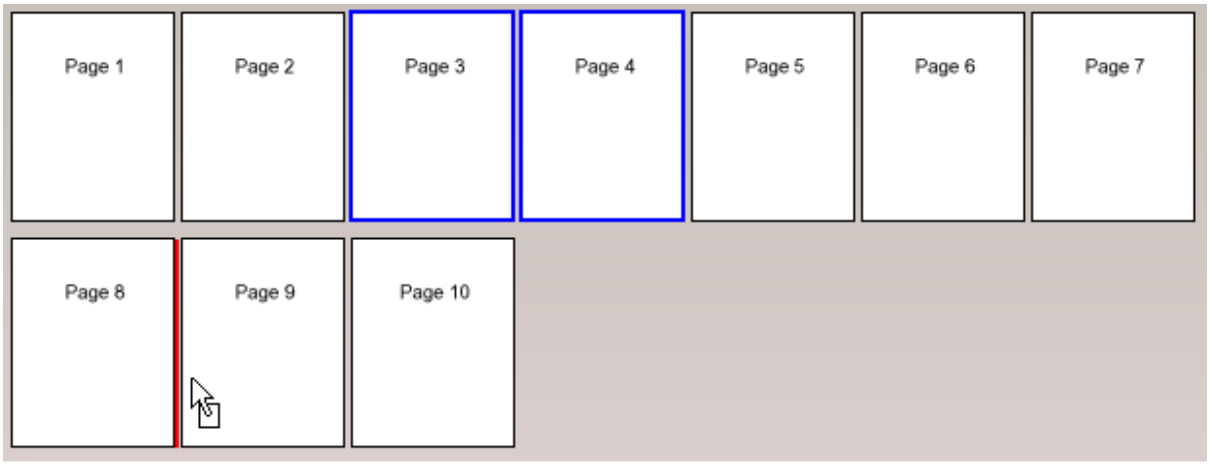
3.8 Page moving

With WPViewPDF PLUS it is able to move selected pages.

The command COMPDF_MOVEPAGES (= 600) can be used to move selected pages.

It is also possible to use interactive page moving.

All You have to do is to set property AllowMovePages to true.



Thumbnail View in WPViewPDF V3 with page selection

The user can click right to select a page and then drag the selection to a different location.

When the mouse button is released an event will be triggered. This makes it possible to intercept the move.

```
procedure TWPViewPDFDemo.DoViewerMessage(Sender: TObject; var ID: Integer;
  Param: Integer);
begin
  case ID of
...
    MSGPDF_MOVEPages:
    begin
       // Check ask user with InputQuery
       WPViewPDF1.Command(COMPDF_MOVEPAGES, Param);
       ID := 0; // Handled here
    end;
  end;
end;
```

3.9 Initialize JBIG2 plugin

JBIG2 support is <u>not</u> linked into the WPViewPDF engine.

However You can use the command
COMPDF_SetJBIG2Tool = 1293
to provide a path and command line parameters to an external tool to convert JBIG2 data to BPM. If the viewer finds the program, it will be used to convert embedded JBIG2 data streams to bitmaps. First the JBIG2 streams are saved into a standard JB2 file and then passed to the conversion tool. The resulting bitmap file (it is expected to be in *PBM "P4"* format) is loaded and displayed. The intermediate files are deleted at once. The conversion program is called invisibly, <u>without</u> showing a window.

For security reasons the conversion is only called to decode image data, not

other stream data, although the PDF specification would allow it.

Example:
WPViewPDF.Command(**1293,** "{dll}convert.exe {in} -o {out}").
{in} will be replaced with a temporary file name of the input data,
{out} will be replaced with the name of the temporary output file.
(Both files will be deleted when finished.)
{dll} will be replaced with the path where the WPViewPDF engine was loaded from.

Please make sure the program name is followed by a space (#32), the tokens {xx} must not contain spaces.
If {in} was not specified, the utility will be called with a temporary file as parameter. The output data will be then expected to have the same name with .pbm as file extension, also if {in} was specified, but {out} was not.

Many similar PDF packages use a project called jbig2dec - *Copyright (C) 2002-2005 Artifex Software, Inc.*
This tool is licensed under GNU license - You can download a VS2010 project including source and binary here.

Notice: We are working on an integrated solution to convert JBIG2 data.
Our implementation is written in native pascal code and does not link other tool sets as obj code.

Hint: If you use pdfPrint you can use the option JBIG2TOOL=...

3.10 Trouble Shooting

Delphi / C++Builder

Some special VCL controls, i.e. the TDrawGrid but not controls like TEdit, will not get the focus back from windows when WPViewPDF got it. So the mouse whell will still scroll the WPViewPDF window after the user click on the grid.

This is easy to fix. Please add a line of code in the Grid.OnClick or Grid.MouseUp event:

Windows.SetFocus(SomeGrid.Handle);

Some developers have reported that their program would not unload when it is closed. This appears to be connected to 3rdparty components. To fix it You can call the global method WPPDFViewerStop in the OnClose of the main form.

3.11 Work with Fields (Widgets)

You need **WPViewPDF PLUS** to work width fields.

Currently supported are text fields and checkboxes.

This code can be used to load all fields into a value list:

```
var i, l : Integer;
    s : AnsiString;
begin
  i := 0;
  SetLength(FieldToIndex, 100);
  repeat
    l := WPViewPDF1.CommandEx(COMPDF_ACRO_GET, i);
    if l>0 then
    begin
      SetLength(s, l);
      WPViewPDF1.CommandEx(COMPDF_GetTextBuf, Integer(PAnsiChar(s)));
      l := Pos('=', s);
      if l=0 then l := Length(s)+1;

      if Length(FieldToIndex)<=FieldValues.RowCount then
             SetLength(FieldToIndex, FieldValues.RowCount+100);

      FieldValues.InsertRow(Copy(s,1,l-1), Copy(s,l+1,Length(s)), true);
      // Save the index of the field.
      FieldToIndex[ FieldValues.RowCount-1 ] := i;
    end;
    inc(i);
  until l<0;
end;
```

Here we use the command **COMPDF_ACRO_GET** - it retrieves the name and value of a field with a certain number in the range [0..N]. The value is separated by '='. If the number is too high, -1 is returned.

The "Value" of a field is usually the text stored in it. In case of checkboxes (Fieldtype = "Btn") the value will be 0 or 1. If the field text is "Off", 0 will be used, if the other name used by the definition of the field, 1 will be used.

This basically means that You can expect the value always to be 1 and 0 for checkboxes, although in PDF the checked state may have different names, usually "Yes" but not always.

To write the field value this command can be used

CommandStrEx(**COMPDF_ACRO_SET**, NewValueString, FieldIndex);

In case of checkboxes "0" and "1" will be translated to "Off" and "Yes" (or the

other name used in appearance stream).

In case of text fields a new appearance stream will be created or an existing will be replaced. This makes sure, the screen is not only updated, but also when the PDF is written, the new value will be displayed by other PDF readers.

4 Commands

WPViewPDF exposes all its methods through a set of methods which all mainly execute a command inside the library.

The command at least needs an ID as parameter, and, depending on the feature other parameters as integer, cardinal, character pointer or record pointer.

When You are using the CXL in Delphi or C++Builder the following methods can be used to execute commands.

In any case a command is send to the viewer window. The different methods are used to add different parameters.

```
function command(command: Integer): Integer; overload;
function command(command, Param: Integer): Integer;  overload;
```

This methods can also be used. They are provided to offer compatibility with older compilers.

```
function CommandEx(command: Integer; Param: Cardinal): Integer;
function CommandStr(command: Integer; str: AnsiString): Integer;
overload;
function CommandStrEx(command: Integer; str: AnsiString; Param:
Cardinal)
  : Integer; overload;
function CommandStr(command: Integer; str: WideString): Integer;
overload;
function CommandStrEx(command: Integer; str: WideString; Param:
Cardinal)
  : Integer; overload;
```

This commands are used when a string result is expected:

```
function CommandGetStr(command: Integer; Str:String; Param: Cardinal):
WideString;
function CommandGetStrA(command: Integer; Str:String; Param: Cardinal):
```

AnsiString;

The commands are defined in the unit WPDF_ViewCommands. They all start with "COMPDF_..."

The .NET assembly implements this variants of the command function:

```
public int Command(int commandnr, string StrParam, uint Param)
public int Command(int commandnr, string StrParam, int Param)
public int Command(int commandnr, string StrParam, byte[] BufferParam)
public int Command(int commandnr, string StrParam, int Param)
```

Also implemented are this two methods to make it easier to convert code provided for the VCL edition to .NET:

```
public int CommandStrEx(int commandnr, string StrParam="", int Param=0)
public int CommandStr(int commandnr, string StrParam = "")
```

If a command shoud return a string or a buffer use this functions:

```
public string CommandGetStr(int CommandID, string StrPar = "", int
IntPar = 0)
public byte[] CommandGetStrA(int CommandID, string StrPar = "", int
IntPar = 0)
```

The commands are defined in the namespace WPViewPDF inside the class "commands". So you need to write Command(commands.COMPDF_.....)

Native C / C++

Here you can use an implementation like this to call the "EX" command which not only passes a string but also an integer parameter.

```
struct TWPComRecStruct
{
        int StrParam;
        int WStrParam;
        int StrLen;
        unsigned int Param;
        int IParam1; // not used
```

```
        int IParam2;
        int IParam3;
        int IParam4;
        int Reserved; // Must be 0
};

int PDFWindow::CommandEx(int cmd, CString StrParam, int Param)
{
        int i = StrParam.GetLength()+1;

        char    *pmb = (char *)malloc( i );
        wchar_t *pwc = (wchar_t *)malloc( sizeof( wchar_t ) *

        strcpy(pmb, StrParam);
        mbstowcs( pwc, pmb, i );

        TWPComRecStruct rec;
        memset(&rec, 0, sizeof(TWPComRecStruct));

        rec.Param = Param;
        rec.WStrParam = (int)pwc;
        rec.StrLen = StrParam.GetLength();
        int iRes = SendMessage(WM_PDF_COMMANDEX, cmd, (LPARAM)
        free(pmb);
        free(pwc);
        return iRes;
}
```

IDs of the following command groups can be used:

4.1 Configuration

COMPDF_SETPAPERCOLOR = 54

Select the paper color. IntParam is a RGB value.

COMPDF_SETDESKCOLOR = 53

Select the background color. IntParam is a RGB value.

COMPDF_SETDESKCOLORTO = 59

WPViewPDF can also paint a vertical marquee effect in the background. To select the second color use this command.

COMPDF_AdvancedFontDrawing= 135

Changes the condition under which fonts are loaded and rendered by the vector engine.

IntParam can have this values:

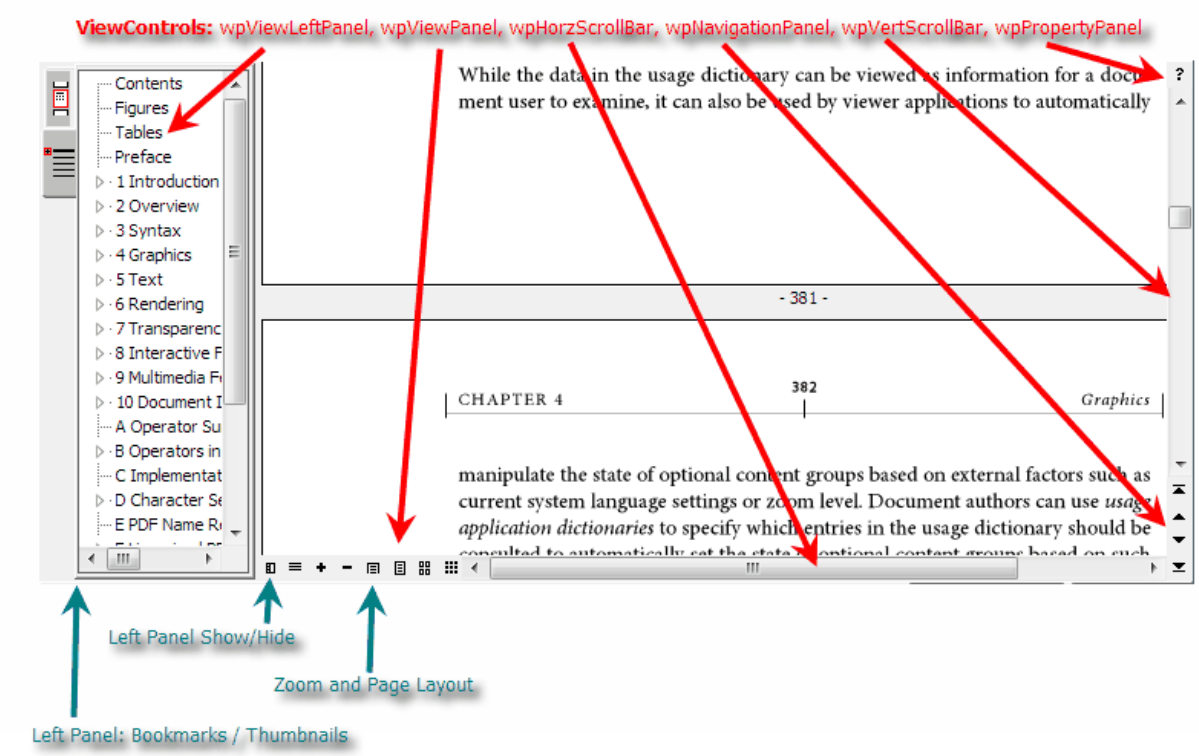
0: (default) Render outlines (only) for embedded subset fonts or fonts which are NOT installed on system
1: renders all fonts as outlines, also installed fonts
2: renders all embedded fonts as outlines

Add 4 to one of the above values and the engine will print unscaled text through regular GDI. This setting must be applied before the PDF is loaded to be effective.

Add 8 to switch off any outline drawing. All text will be rendered as text and not as outlines. This only works for fonts which are installed on the system. Please use it with care. The usual "Helvetica" font will be rendered as Arial.

WPViewPDF can display different panels with buttons.
If a certain panel is displayed or not is controlled by **COMPDF_SelectControls = 50**.

IntParam is a bitfield to select which panels to see:
1=Vertical Scroll,
2=Horizontal Scroll,
4=View(Zoom +-..)Panel,
8=Search(Navigation <->)Panel,
16=Option "?" Button



COMPDF_SelectViewOptions = 51

This command is used to enable or disable certain features of the UI. It is also used to disable the usual highlighting of hyperlinks.

IntParam is expected as bitfield with this bit values.

1 : wpDontUseHyperlinks - do not auto jump on click on hyperlinks
2 : wpDontHighlightLinks - do not paint links with blue background
2048: wpInactivateHyperlinks - don't use links at all
4 : wpDontAskForPassword - don't display a password dialog for protected files

This flags allow page selection:

16: wpPageSelection - activate/deactivate the selection by mouse click
32: wpPageMultiSelection - the user can press CTRL to select multiple pages

Instead of hiding pages which are marked for deletion, cross them out:

8192: wpShowDeletionCross (for deleted pages)

Disable the hint displayed when the user scrolls:

128: wpDisablePagenrHint

Disable the zoom hint:

256: wpDisableZoomHint

Modify the bookmark view

1024: wpDisableBookmarkView
4096: wpExpandAllBookmarks

COMPDF_SetExViewOptions = 81

IntParam is expected to be a bit field with this values
1: Show Page Numbers in main viewer. (default: no page numbers)
Use COMPDF_SetPageNumberString to modify the displayed page number text.
2: Hide Page Frames in main viewer (default: frames)
4: FastZoom Mode in main viewer (default: off)
16: Hide Page Numbers in thumbnail viewer (default: display page numbers)
32: Hide Page Frames in thumbnail viewer (default: frames)
64: FastZoom Mode in thumbnail viewer (default: off)

COMPDF_SetPageNumberString = 82

Set the page number format string. First %d=page number, second %d=page count. Default is ' %d ', you can also set '%d/%d'

Configure Popup Menu

The following IDs can be used to set the captions of the popup menu selectable on the [?] button in the upper right corner. You can pass "" to disable the menu entry.

COMPDF_SetDocumentProperties = 61
COMPDF_SetShowAbout = 66;
COMPDF_SetPrintSetup = 68;
COMPDF_SetPrint = 69;

Configure Hints

The following ID can be used to set the hints for certain buttons:
COMPDF_SetShowHint = 71;

The value of IntParam selects the hint, StrParam is the new hint text.
pdf_hint_ONOFF = 0 - Use StrParam="1" to activate, "0" to deactivate
pdf_hint_LeftPanel = 1- Set string: left panel, thumbnails etc

```
pdf_hint_Zoom100 = 10;
pdf_hint_ZoomIn = 11;
pdf_hint_ZoomOut = 12;
pdf_hint_ZoomWidth = 13;
pdf_hint_ZoomPage = 14;
pdf_hint_ZoomTwoPages = 15;
pdf_hint_ZoomThumbnails = 16;
```

COMPDF_UseGDIPainter = 141

Switches the renderer for screen display. The default is IntPar=1 which selects the GDI+ Renderer, IntPar=0 selects the AGG Renderer. The latter produces better looking letters and better scaled images (antialias), but implements only basic clipping. Printing will always use GDI+.

COMPDF_DisableThreading = 146

Pass 1 to disable the multithreaded paint, 0 to enable multithreaded painting. The change will take effect after the next load operation.

COMPDF_ShowNavigation = 134

This command can be used to force the display of the navigation panel (Bookmarks and Thumbnails).
Use IntPar=0 to hide it, 1 to show it and 2 to toggle its visibility.

4.2 Show internal Dialogs

COMPDF_DocumentProperties = 1

Display a dialog which shows a dialog with the information strings inside the current PDF file.

COMPDF_ShowAbout = 6;

Display the about box of the viewer control. It contains the version number and release date of the engine.

COMPDF_PrinterSetup = 30

Show printer selection and setup dialog

COMPDF_PrintDialog = 32

Display the print dialog.

This commands can be used to preset the values for the print dialog:

COMPDF_SelectPrintDiaFromPage = 56 - set the "from page" value
COMPDF_SelectPrintDiaToPage = 57 - set the "to page" value
COMPDF_SelectPrintDiaDontCollate = 58 - unset the "collate" checkbox.

4.3 Navigate in PDF

a) Navigate Page wise

COMPDF_GotoFirst = 20

Goto the first first page in current PDF data.

COMPDF_GotoPrev = 21

Goto previous page. If Param =1 it scrolls one screen height up.

COMPDF_GotoPage = 22

Goto Page Nr in parameter. The first page has number 0.

StrParam can be optionally used. It will be interpreted as
"" = start of page
"y" = certain y coordinate from top of page measured in 72 dpi values
"x,y" = certain X, Y position
"x,y%z" = certain X and Y position and Zoom value

COMPDF_GotoNext = 23

Goto next page. If Param =1 it scrolls one screen height down.

COMPDF_GotoLast = 24

Goto last page. Pass intpar=1 to go to end of last page.

b) Navigate to X, Y position measured in 72 dpi from start of the PDF data

COMPDF_GotoYPos = 27

Move to a certain Y (top offset) position.

COMPDF_GotoXPos = 28

Move to a certain X (left offset) position.

COMPDF_ScrollXY = 29

Scroll horizontally or vertically.

IntParam is a bitfield:
1: scroll horizontally, otherwise vertically
2: scroll by 4/5 of the box size, otherwise 1/5
4: move down, otherwise up.

COMPDF_GotoNamedDest = 270

Goto bookmark.
StrParam is the name of a bookmark.
Result=PageNumber or -1 if not found

c) Zooming

```
 COMPDF_Zoom100          = 41   ---------->  100 % Zoom
 COMPDF_ZoomIn          = 42; ---------->  + 10%
 COMPDF_Zoom            = 43; ----------> Zoom to IntPar - if IntPar=0
retrieve zoom!
    If StrPar='MP' it will center to mouse position
 COMPDF_ZoomOut         = 44; ---------->  - 10%
 COMPDF_ZoomFullWidth    = 45;----------> Page Width
 COMPDF_ZoomFullPage     = 46; ----------> Page Width
 COMPDF_ZoomTwoPages      = 47; ----------> Toggle 2 Pages Display
 COMPDF_ZoomThumbs       = 48; ----------> Thumbnail Preview
 COMPDF_ZoomGetCurrent    = 49; ----------> read current zoom
 COMPDF_ZoomSaveRestore = 76;----------> IntPar=1 Saves, IntPar=0
Restores
```

Example:

This Delphi code will implement temporarily zooming to 200% when clicking on a certain point :

Must be called before for custom mouse handling:

```
WPViewPDF1.CommandEx(COMPDF_SelectMode, 8);
```

We need a variable to store the zoomed mode

```
var FZoomed : Boolean;
```

The MouseDown event

```
procedure TForm1.WPViewMouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
  if not FZoomed then
  begin
     WPViewPDF1.command(COMPDF_ZoomSaveRestore, 1); // Save Position
     WPViewPDF1.command(COMPDF_Zoom, 'MP', 200);
     FZoomed := true;
  end;
end;
```

The MouseUp event

```
procedure TForm1.WPVIewMouseUp(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
  if FZoomed then
  begin
     FZoomed := false;
     WPViewPDF1.command(COMPDF_ZoomSaveRestore, 0); // Goto saved position
  end;
end;
```

Variation if the example:

Temporarily zoom in with middle mouse.

```
// Disable Middle Mouse
WPViewPDF1.Command(COMPDF_RefineMouseMode, '0', 1);

var FZoomed : Boolean;

procedure TForm1.WPViewMouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
  if not FZoomed and (Button=mbMiddle) then
  begin
     WPViewPDF1.command(COMPDF_ZoomSaveRestore, 1); // Save Position
     WPViewPDF1.command(COMPDF_Zoom, 'MP', 200);
     FZoomed := true;
  end;
end;

procedure TForm1.WPVIewMouseUp(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
  if FZoomed then
  begin
     FZoomed := false;
     WPViewPDF1.command(COMPDF_ZoomSaveRestore, 0); // Goto saved position
  end;
end;
```

4.4 Printing (on paper)

Please note: If security settings of a PDF file forbid printing, the component will not print. You as developer can override this at Your own risk. Use command (COMPDF_DisableSecurityOverride,1) to disable this check.

COMPDF_DisablePrint = 123

You can disable printing globally by using this command. It is not possible to enable it again!

COMPDF_DisableHQPrint = 124

Disable high quality print - if print, only low quality!

COMPDF_PrinterSetup = 30

Show printer selection and setup dialog

COMPDF_PrintDialog = 32

Display the print dialog.

This commands can be used to preset the values for the print dialog:
COMPDF_SelectPrintDiaFromPage = 56 - set the "from page" value
COMPDF_SelectPrintDiaToPage = 57 - set the "to page" value
COMPDF_SelectPrintDiaDontCollate = 58 - unset the "collate" checkbox.

COMPDF_Print = 31

Print the loaded PDF data.

IntParam: if not 0 it is interpreted as the page range. The low word is the first page, the high word is the last page to be printed. Pages numbers are in the range 1...page count.

StrParam:

If not empty, this string is interpreted as page range, i.e. "1-3,7,15". It is also possible to specify the number of copies with "*", i.e. "1-3*2" to print two copies of the pages 1 to 3.

COMPDF_PrintUseScaling = 155

This command selects different supported scaling methods. Possible values for IntParam are:

0=off, do not scale at all.
1=shrink only if required,
2=shrink to page area, this is the default setting. (Do not enlarge.)
3=scale to printable area - this can also cause enlargement
4=scale to page area- this can also cause enlargement
5=multi page mode. Print multiple pages on one paper sheet. The default is two pages side by side. The command id *COMPDF_MultiPagePrintColRow* (=139) is used to change the count of rows and columns. The high byte of the IntParam is the count of rows, the low byte the count of columns.

COMPDF_GetPrinter = 33

Read the current printer name as string. It expects IntParam to be either 0 a pointer to a 256 ansi char buffer which will be filled with the name. If IntParam was 0, the name can be loaded by the command COMPDF_GetTextLen and COMPDF_GetTextBuf.

COMPDF_SelectPrinter = 35

Select a certain printer. The routine will first try an exact match, later searches for the matching printer by checking if the printer name contains the name in the string parameter StrParam.

COMPDF_BeginPrint = 36 & COMPDF_EndPrint = 37

Opens and closes a print job. This makes it possible to send several PDF files into one printing cue using COMPDF_Print.

COMPDF_SetPrintTitle = 70

Set the name of the printing cue

COMPDF_SelectDuplexMode = 34

IntParam is used a duplex mode identifier. Possible values are 0=off, 1=horizontal, 2=vertical

COMPDF_SelectCopies = 55

Select the count of copies to be printed.

COMPDF_SelectPrinterBin0 = 38

Select a certain paper bin for all pages. The paper bin id is used in DEVMODE dmDefaultSource element.

COMPDF_SelectPrinterBin1 = 39

Sets the paper bin for the first page.

COMPDF_PrintUseBitmaps = 138

Prints the pages to a bitmap and the prints this bitmap on the printer device.

If the IntParam is in range=1..10 a colored bitmap with Resultion = Printer-Resultion / Value will be created
A larger value will be used directly as printing resolution.
A negative value will enable the use of monochrome image
0 disable the buffered print.

COMPDF_AdvancedFontDrawing= 135

Changes the condition under which fonts are loaded and rendered by the vector engine. See "configuration" command group.
Please note that since printing is done through GDI+ many text elements will be converted to graphic paths, even if our engine is using the installed fonts. After using the value 4 in this command, the engine will print unscaled text through regular GDI to avoid this effect.

COMPDF_DONTSETDEVMODE = 158

If IntParam=1 the printer configuartion will not be updated, if it is 0 it will if necessary.

COMPDF_PrintSetDEVMODE = 156

Pass a unicode DEV mode as IntParam. The current DEVMODE will be overwritten. IntParam must be a pointer to the DevModeW record.

COMPDF_PrintGetDEVMODE = 159

Result is a HGLOBAL of the printer DEVMODE record.

COMPDF_PrinterSetMediatype = 181

Set mediatype for printing.
Internally this calls: WinSpool.DeviceCapabilities(Device, Port, DC_MEDIATYPES, nil, nil);

COMPDF_PrintAbort = 180

Usually this has no effect since the data is sent to the spooler much quicker than the actual printing takes.

COMPDF_SelectPaperWidth = 73

If larger than 0, set the value for the DEVMODE dmPaperWidth member which will be set in the printer structure.

COMPDF_SelectPaperLength = 74

If larger than 0, set the value for the DEVMODE dmPaperLength member which will be set in the printer structure.

COMPDF_SelectPaperSize = 75

If larger than 0, set the value for the DEVMODE dmPaperSize member which will be set in the printer structure.
If -1 is used, the value will be not set and the default paper size defined for the printer will be preserved. (Switch off automatic paper size switching)

4.5 Printing (on device)

WPViewPDF is also able to print certain PDF pages to a windows device handle (HDC). The printing will be internally done by GDI+.

COMPDF_DisablePrintHDC = 126

Disable print to HDC - it is not possible to enable again!

COMPDF_PrintHDC_SelectPage = 160

Select the page to be printed next

COMPDF_PrintHDC_SelectedPage = 161

Print the selected page on the HDC device with the handle passes as IntParam. The result value is -1 on error or the printed with and height as high and low

word.

COMPDF_UseGDIForPrinting = 145

Select the standard GDI renderer instead of GDIPLUS, with parameter=1 or the GDIPLUS render with parameter=0 (default).

Using 1 can result in smaller print files and faster output. For difficult PDF files it can cause a decrease in output quality. When using pdfPrint use option "STDGDI=1".

COMPDF_PrintHDCSetXRes = 152

Set X Resolution for the next COMPDF_PrintHDC. Use negative value to set desired *width* in pixels.

COMPDF_PrintHDCSetYRes = 153

Set Y Resolution for the next COMPDF_PrintHDC. Use negative value to set desired *height* in pixels.

The Delphi VCL implements wrapping methods which can be directly called:

```
function TWPViewPDF.PrintHDC(
  PageNo: Integer;
  DC: HDC;
  ResXOrW, ResYOrH: Integer): Boolean;
var
  IsW, IsH: Integer;
begin
  Result := PrintHDC(PageNo, DC, ResXOrW, ResYOrH, IsW, IsH);
end;

function TWPViewPDF.PrintHDC(
  PageNo: Integer;
   DC: HDC;
  ResXOrW, ResYOrH: Integer;
  var IsW, IsH: Integer)  : Boolean;
var
  wh: Integer;
begin
  CommandEx(COMPDF_PrintHDCSetXRes, ResXOrW);
  CommandEx(COMPDF_PrintHDCSetYRes, ResYOrH);
  // 1. Set Pagenumber
  CommandEx(COMPDF_PrintHDC_SelectPage, PageNo);
  // 2. Print using this page number
  wh := CommandEx(COMPDF_PrintHDC_SelectedPage, DC);
  if wh = -1 then
```

```
    Result := false
  else
  begin
    IsW := (wh shr 16) and $FFFF;
    IsH := wh and $FFFF;
    Result := (IsW > 0) and (IsH > 0);
  end;
end;
```

4.6 Load PDF

When loading an encrypted PDF file (which uses a non-empty password) a message box to enter the password will pop up.

To prevent this You can use

COMPDF_SetLoadPassword = 120

Sets master load password. This can also be done in the NeedPassword event!

COMPDF_AddPassword = 122

Adds a password to the list of passwords to be used.

COMPDF_ClearPasswords = 121

Clear the complete list of passwords.

When using the VCL it is better to use the high level load methods.

COMPDF_FastLoad = 99

Loads but do not display a PDF file. The name is passed as StrParam. The result value is the count of pages.

COMPDF_UpdatePages = 107

Force the update of the bookmark and scroller. This is not required for the load methods below.

COMPDF_Clear = 100

Removes all loaded PDF data and closes any streams opened by the component.

COMPDF_Load = 101

Load a PDF file. If IntPar=1 the file will be copied to memory - this makes it possible to delete, move or overwrite the original file.
The result value is the count of pages.

COMPDF_Append = 102

Appends a PDF file. If IntPar=1 the file will be copied to memory - this makes it possible to delete, move or overwrite the original file.
The result value is the count of pages.

COMPDF_AppendHGlobal = 103
COMPDF_LoadEHGlobal = 95

Load or append the PDF data from a HGLOBAL memory handle.

COMPDF_AppendIStream = 104
COMPDF_LoadIStream = 96

LOad or append PDF from a COM stream. IntParam is an interface pointer.

COMPDF_AppendIStreamKeepOpen = 108

Works like COMPDF_AppendIStream but keep the stream open!

COMPDF_AppendEStream = 105 (create copy)
COMPDF_LoadEStream = 97 (create copy)
COMPDF_AppendEPStream = 106 (keep stream open)
COMPDF_LoadEPStream = 98 (keep stream open)

Load or Append PDF from an event stream. Event streams are implemented as a structure with 3 function pointers.
The first 2 create a copy of the data, the second 2 keep the stream open.

The WPViewPDF VCL uses this streams internally.

Definition:

```
TEventStreamFkt = packed
record
    OnStreamRead: function(data: Pointer; buffer: Pointer; len: Integer)
      : Integer;
    stdcall;
    OnStreamWrite: function(data: Pointer; buffer: Pointer; len: Integer)
      : Integer;
    stdcall;
    OnStreamSeek: function(data: Pointer; Offset: Integer; Origin: Integer)
      : Integer;
```

```
    stdcall;
    Stream: TStream;
end;

function ReadEvent(data: Pointer; buffer: Pointer; len: Integer): Integer;
  stdcall;
begin
  Result := PEventStreamFkt(data).Stream.Read(PAnsiChar(buffer)^, len);
end;

function WriteEvent(data: Pointer; buffer: Pointer; len: Integer): Integer;
  stdcall;
begin
  Result := PEventStreamFkt(data).Stream.Write(PAnsiChar(buffer)^, len);
end;

function SeekEvent(data: Pointer; Offset: Integer; Origin: Integer): Integer;
  stdcall;
begin
  Result := PEventStreamFkt(data).Stream.Seek(Offset, Origin);
end;
```

Usage:

```
function TWPViewPDF.LoadFromStream(Stream: TStream; WithClear: Boolean = false): Boolean;
var
  events: TEventStreamFkt;
begin
  events.Stream := Stream;
  events.OnStreamRead := Addr(ReadEvent);
  events.OnStreamWrite := Addr(WriteEvent);
  events.OnStreamSeek := Addr(SeekEvent);
  try
    if WithClear then
    begin
      if FEngineVersion < 3000 then
      begin
        CommandEx(COMPDF_Clear, 0);
        Result := CommandEx(COMPDF_AppendEStream, Cardinal(@events)) = 0;
      end
      else
        Result := CommandEx(COMPDF_LoadEStream, Cardinal(@events)) = 0
    end
    else
      Result := CommandEx(COMPDF_AppendEStream, Cardinal(@events)) = 0;
  except
    Result := false;
  end;
end;
```

4.7 Save PDF

WPViewPDF PLUS can also save the loaded PDF data to a new PDF file.
This makes sense if you need to merge multiple PDF files into a new file, if you need to delete certain pages, if you changed info strings or added or removed encryption.

Unlike some competing products, WPViewPDF PLUS 3 checks all exported pages for used images and fonts - and only exports the fonts and images which are actually used.

If security settings of a PDF file forbid saving, the component will not save.
You as the developer can override this at Your own risk.
Use command(COMPDF_DisableSecurityOverride,1) to disable this check.

When extracting text from a PDF file WPViewPDF will first sort the text element using their horizontal coordinate. This can be switched off using COMPDF_TextExtractDontSort.

It is possible to disable saving using command(COMPDF_DisableSave). It is not possible to enable it again.

To check wether the PDF file may be saved, use command

COMPDF_MaySavePDF = 500.

If Result > 0 the document may be saved (it is not protected).

COMPDF_CheckOwnerPassword = 291

Checks if the given password (StrParam) matches the owner password.
If yes, the security is cleared and TRUE is returned.

The commands below are used for saving.

When using the VCL it is better to use the high level save methods.

COMPDF_SaveToFile = 501

Save the combined contents to file.

COMPDF_SaveToEStream = 497

Save to event stream. Result = count of saved pages.

COMPDF_SaveSelectionToEStream = 498

Save selected pages to event stream. (Used internally in Delphi VCL - see load commands)

COMPDF_SaveSelectionToFile = 511

Save selected pages to a file.

COMPDF_SaveToIStream = 513

Save to IStream, must be passed as IUnknown reference

COMPDF_SaveSelectionToIStream = 514

Save selected pages to IStream, must be passed as IUnknown reference

This commands are used to set the security features of the saved PDF

COMPDF_SetSecurityMode = 507

Set security when saving 0=off, 1=40 bit, 2=128 bit RC4

COMPDF_SetSecurityFlags= 508

Set the "P flags" bitfield.
To disable a feature the bit must be clear
Bit 3 = printing
Bit 4 = modification
Bit 5 = allow copy and extract
Bit 6 = add annotations

COMPDF_SetSecurityUser = 509

StrParam is the user password.

COMPDF_SetSecurityOwner= 510

StrParam is the owner password.

4.8 Change the way the mouse works

COMPDF_SelectMode = 133

Changes the way the **left mouse** button works. The following modes are possible:

wpmouse_Pan = 0

The user can press and drag the mouse to move the displayed area.

wpmouse_SelectText = 1

The user can click and drag the mouse to select text. The selected text can be extracted with COMPDF_GetTextLen/COMPDF_GetTextBuf.

wpmouse_DrawCustom = 2

The user can click and drag the mouse to draw a frame. When the frame is completed, the message WM_PDF_STAMPTEXT is sent which triggers the event OnSelRectEvent.

wpmouse_DrawObject = 3

The user can draw a rectangle - a new object will be created when finished. See "Draw Shapes". The object must be first initialized with COMPDF_MouseAddDrawObject.

wpmouse_SelectPage = 4

When the user click, the page is selected. Also see ViewOptions.

wpmouse_SelectObject = 5

The user can select and move draw objects.

wpmouse_Point = 6

Do nothing.

wpmouse_SelectPath = 7

Reserved. Cannot be used in Standard and PLUS edition.

To change the way the **right mouse** button works, add 100 to the above values.

4.9 Set and get additional properties

When you need WPViewPDF to reurn a string value, the usual way is to call a "get" command first which returns the required buffer length. Then this commands can be used to fill a prepared buffer with the data:

COMPDF_GetTextBuf = 261
Read ANSI buffer.

COMPDF_GetTextBufW = 263
Read unicode buffer.

The WPViewPDF implements the utility functions CommandGetStr and CommandGetStrA to simplify the task:

Read unicode string:
```
function TWPViewPDF.CommandGetStr(command: Integer; str: String;
  Param: Cardinal): WideString;
var
  i: Integer;
begin
  i := CommandStrEx(command, str, Param);
  if i > 0 then
  begin
    SetLength(Result, i);
    CommandEx(COMPDF_GetTextBufW, Cardinal(@Result[1]));
  end
  else
    Result := '';
end;
```

Read ANSI string:
```
function TWPViewPDF.CommandGetStrA(command: Integer; str: String;
  Param: Cardinal): AnsiString;
var
  i: Integer;
begin
  i := CommandStrEx(command, str, Param);
  if i > 0 then
  begin
    SetLength(Result, i);
    CommandEx(COMPDF_GetTextBuf, Cardinal(@Result[1]));
  end
  else
    Result := '';
end;
```

COMPDF_GetInfoItemsLen = 264

This command is used to read the info items of the PDF as string list. The items will be comma separated.
You first need to call COMPDF_GetInfoItemsLen to get the required buffer length, and then COMPDF_GetTextBuf to read the actual text.

COMPDF_GetInfoItemsLenW = 265

Works like COMPDF_GetInfoItemsLen but creates a unicode string. Use COMPDF_GetTextBufW to read the text.

When you use WPViewPDF PLUS You can also set the info items:

COMPDF_SetString = 502 - Set info entry name=text
COMPDF_SetTitle = 503 - Set title info string
COMPDF_SetSubject = 504 - Set subject info string
COMPDF_SetAuthor = 505 - Set author info string
COMPDF_SetKeywords= 506 - Set keywords info string

5 Component Description

WPViewPDF is a component to view and print PDF files.

If you licensed WPViewPDF **PLUS** you can save a new PDF file from WPViewPDF.
This feature can be used to remove pages from PDF files, to merge several PDF files into a new file and to remove or apply security features or set info record items.

It is also possible to add text, images and vector objects to a PDF file.

5.1 Methods

5.1.1 TWPViewPDF.ViewerStart Method

This method is slightly different in VCL and .NET edition:

A) .NET

This method is used to set the license keys. *(You cannot set the DLL name - it must be encoded in the interface assembly since the [DLLImport] requires a fixed name.)*

B) VCL (for Delphi and C++Builder)

```
procedure ViewerStart(DLLNameAndPath, licensename, licensekey: string;
licensecode: Integer);
```

This method is used to set the license keys. Optionally the DLL name can be defined.

In case the file PDFLicenses.INC contains valid information it is used automatically.

5.1.2 TWPViewPDF.AppendFromFile Method

Declaration
```
function AppendFromFile(const filename: string): Boolean;
```

Description
This command opens a different PDF file at the end of the currently loaded PDF file. Both PDF files can now be scrolled and printed as if they are one file. Please note that the PDF file is opened in shared mode and remains open until the editor is cleared or closed. The information of the PDF file is not loaded when required.

5.1.3 TWPViewPDF.AttachStream Method

Declaration
```
function AttachStream(stream: TStream): Boolean;
```

Description
This command attacheds a PDF stream to the viewr. This means the viewer will display the PDF information. **The provided stream object has to be valid until the editor is closed or cleared!** It will be used for read access whenever the viewer needs new data, for example if it needs to load a page description or image data.

5.1.4 TWPViewPDF.BeginPrint Method

Declaration
```
function BeginPrint(Printername: string): Boolean;
```

Description
This procedure starts a new print job. You can pass the name of the printer which should be used. (for propper names see the list Printer.Printers)

5.1.5 TWPViewPDF.Clear Method

Declaration
```
procedure Clear;
```

Description
This commands frees any data allocated by the viewer and closes open files. If you have attached a stream (AttachStream) you may free that stream now. The command LoadFromFile implies a 'Clear'.

5.1.6 TWPViewPDF.Command Method

Declaration
```
function Command(command: Integer): Integer;
```

Description
This is the general command used to communicate with the PDF engine. It receives an integer as command id.

more ...

5.1.7 TWPViewPDF.DeletePage Method

Declaration
```
function DeletePage(N: Integer): Boolean;
```

Description
This method marks a page to be deleted. The first page has the number 0. Please store the original page count before you use DeletePage since after DeletePage this page will not be counted anymore - although the array DeletePage and UndeletePage works on doe not change. You can also use Command() with id COMPDF_DeletePage = 490. To enable the display of the page again use UnDeletePage (COMPDF_UnDeletePage = 491).

5.1.8 TWPViewPDF.EndPrint Method

Declaration
```
procedure EndPrint;
```

Description
Closes a print job started with BeginPrint.

5.1.9 TWPViewPDF.FindText Method

Declaration
```
function FindText(Text: string; HighLight, FindNext: Boolean;
CaseInsensitive: Boolean = false; DontGoToPage: Boolean = false): Boolean;
```

Description
This function searches text in the loaded PDF file. Please set the parameter HighLight to TRUE to also highlight the found text. Use FindNext=TRUE to continue a search on the following pages. You need to pass the same search string. To switch of the highlighting pass an empty search text. Please note: The search function does not check spaces.

This functions is implemented like this:

```
function TWPViewPDF.FindText(
    Text: string;
    HighLight, FindNext: Boolean;
    CaseInsensitive: Boolean = false ;
    DontGoToPage: Boolean = false ): Integer;
begin
  CommandEx(COMPDF_FindGotoPage, Integer(DontGoToPage));
  CommandEx(COMPDF_FindCaseInsitive, Integer(CaseInsensitive));

  // If we search case insensitive we simply search 2 versions of the same string
```

```
  // This allows the support of charsets
  if CaseInsensitive then
  begin
    CommandStr(COMPDF_FindAltText, AnsiUpperCase(Text));
    Text := AnsiLowerCase(Text);
  end;

  if FindNext then
    Result := CommandStr(COMPDF_FindNext, Text)  // Next
  else
    Result := CommandStr(COMPDF_FindText, Text); // First

  if HighLight then
    CommandStr(COMPDF_HighlightText, Text);
end;
```

5.1.10 TWPViewPDF.GetMetafile Method

Declaration
```
function GetMetafile(PageNO: Integer): TMetafile;
```

Description
This procedure is one of the most valuable in this library: it extracts a PDF page as metafile. You only have to specify the page number as a value between 1 and PageCount.

Note: PageNo is 0 based.

5.1.11 TWPViewPDF.GetMetafilePrn Method

Declaration
```
function GetMetafilePrn(PageNO: Integer): TMetafile;
```

Description
This procedure is one of the most valuable in this library: it extracts a PDF page as metafile. You only have to specify the page number as a value between 1 and PageCount.

GetMetafile**Prn** uses the printer as reference to create the metafile.

Note: PageNo is 0 based.

5.1.12 TWPViewPDF.GetPageText Method

Declaration
```
function GetPageText(PageNo: Integer; format: string = ''): string;
```

Description
This function retrieves the text of a certain text as an ANSI string.

You can specify the format You need:

"ANSI" - Ansitext

"HTML" - HTML with CSS styles

"XYHTML" - HTML with CSS styles - each characters will be placed directly using absolute CSS positions

"RTF" - RTF code

The method is implemented like this:

```
function TWPViewPDF.GetPageText(PageNo: Integer; format: string = ''):AnsiString;
var
  len: Integer;
begin
  len := CommandStrEx(COMPDF_GetTextLen, format, PageNo);
  SetLength(Result, len);
  if len > 0 then
    CommandEx(COMPDF_GetTextBuf, Cardinal(PAnsiChar(Result)));
end;
```

Note: PageNo is 0 based.

5.1.13 TWPViewPDF.GetPageTextW Method

Declaration

```
function GetPageTextW(PageNo: Integer; format: string = ''): WideString;
```

Description

This function retrieves the text of a certain text as an unicode string.

See GetPageText.

This function is implemented like this:

```
function TWPViewPDF.GetPageTextW(PageNo: Integer; format: string = '')
  : WideString;
var
  len: Integer;
begin
  len := CommandStrEx(COMPDF_GetTextLenW, format, PageNo);
  SetLength(Result, len);
  if len > 0 then
    CommandEx(COMPDF_GetTextBufW, Cardinal(PWideChar(Result)));
end;
```

Note: PageNo is 0 based.

5.1.14 TWPViewPDF.LoadFromFile Method

Declaration

```
function LoadFromFile(const filename: string): Boolean;
```

Description
This function opend a PDF file to be displayed in the PDF viewer. The PDF files remains open until the viewer is closed ot cleared since it will load data from the PDF files when required.

5.1.15 TWPViewPDF.LoadFromStream Method

Declaration
```
function LoadFromStream(stream: TStream): Boolean;
```

Description
This function loads PDF information from a stream. The stream will be fully loaded - you may close and free it after using this command. To load from a stream which is not loaded at once please use AttachStream.

5.1.16 TWPViewPDF.PrintHDC Method

Declaration
```
function PrintHDC(PageNO: Integer; DC: HDC; ResX, ResY: Integer): Boolean;
```

Description
This command (which is internally used by GetMetafile prints a PDF page to any HDC handle. You can specify the page number and the resolution which should be used for the draw process.

5.1.17 TWPViewPDF.PrintPages Method

Declaration
```
function PrintPages(StartPage, EndPage: Word): Integer;
```

Description
This procedure Prints the PDF file. You may specify a from-to page range (1..) or (0,0) to print the complete file. Please note that you can open a print job first using BeginPrint/EndPrint to avoid multiple printer jobs if you need to execute PrintPages more than once.

You can use Command(155, 2) "COMPDF_PrintUseScaling" to activate the automatic scaling to the page.

5.1.18 TWPViewPDF.UnDeletePage Method

Declaration
```
function UnDeletePage(N: Integer): Boolean;
```

Description
Reverts the change done by DeletePage.

Note: N is 0 based.

5.1.19 TWPViewPDF.WriteJPEG Method

Declaration

```
function WriteJPEG(const Filename: string; PageNo,
      Resolution, Compression: Integer): Boolean;
```

Description

Converts the page into a JPEG file.

Note: `PageNo` is 1 based.

Also see the DLL method pdfMakeJPEG.

5.1.20 TWPViewPDF.WritePNG Method

Declaration

```
function WritePNG(const Filename: string; PageNo,
  Resolution: Integer; bitmap_format : TWritePNGMode): Boolean;
```

Description
Converts the page into a PNG file.

This bitmap formats are supported: wp256Color, wpGrayscale, wp24FullColor

Note: `PageNo` is 1 based.

Also see the DLL method pdfMakeJPEG.

5.1.21 TWPViewPDF.WriteBitmap

Declaration - VCL

```
function WriteBitmap(PageNo: Integer;
      format: TWPBitmapFormat;
      filename: WideString;
      Memory: TMemoryStream = nil;
      Resolution: Integer = 0;
      Compression: Integer = 0;
      LongSidePx: Integer = 0;
      HeightPx: Integer = 0): Boolean;
```

This is an universal method to convert certain pages in the loaded PDF data to bitmaps.
PageNo is the page number - 0 based.
Format can be: wpJPEG_RGB, wpJPEG_Gray, wpPNG_RGB, wpPNG_256,

wpPNG_GRAY, wpPNG_BW, wpBMP_RGB, wpBMP_256, wpBMP_Gray, wpBMP_BW, wpAutomatic
Memory can be provided to create the bitmap data inside of a memory stream. Resolution can be used to specify the size of the created bitmap. Alternatively the parameters LongSidePx and HeightPx can be used. LongSidePx can be used to speizfy the desired width or, if HeightPX=0, the exact pixel count of the longest side.
Compression is only used for JPEG images to specify the JPEG compression in range 1-100

Declaration - .NET

```
public bool WriteBitmap(int PageNo, BitmapFormat Format, string Filename,
       int Resolution = 0, int Compression = 0,
       int LongSidePx = 0, int HeightPx = 0)
```

Note: "Memory" is currently not supported in .NET edition.

6 Direct Calls to DLL

WPViewPDF implements a set of direct calls to the DLL. This calls can be used to merge, convert and print PDF data without the need to create a TWPViewPDF instance.

6.1 pdfMakeImage - convert selected pages to bitmaps

This function is exported by the engine DLL to make it easy to convert PDF pages into bitmap files.
No object has to be created and no initialization is required.

This function defined as

Pascal:

```
function pdfMakeImage(
  filename: PAnsiChar;
  password: PAnsiChar;
  licname, lickey: PAnsiChar;
  liccode: Cardinal;
  destpath: PAnsiChar;
  frompage, to_page: Integer;
  jpegres: Integer): Integer; stdcall;

  fktpdfMakeImageW = function(filename: PWideChar;
   password: PWideChar;
    licname, lickey: PWideChar;
```

```
  liccode: Cardinal; destpath: PWideChar;
  // Use %d store page number !
  frompage, to_page: Integer; jpegres: Integer): Integer; stdcall;
```

Note: The unit WPViewPDF initializes the pointer wpview_....

C:

```
stdcall int pdfMakeImage(
  char *filename,
  char * password,
  char * licname,
  char * lickey,
  long liccode,
  char * destpath,
  int frompage,
  int to_page,
  int jpegres);
```

VB

```
Declare Function pdfMakeImage Lib "wPDFViewDemo03.dll" _
                Alias "pdfMakeJPEG" _
                (ByVal zfilename As String, _
                 ByVal zpassword As String, _
                 ByVal zlicname As String, _
                 ByVal zlickey As String, _
                 ByVal liccode As Long, _
                 ByVal zdestpath As String, _
                 ByVal frompage As Long, _
                 ByVal To_page As Long, _
                 ByVal jpegres As Long) As Long
```

Parameters:

filename: the full path to the input PDF file. Please pass a UTF8 string.

password: optional user password required to open the PDF file

licname, lickey, liccode: when using registered version use your license data here

destpath: the path to the created image file. The placeholder %d is replaced with the page number. Please pass an UTF8 string.

The variable "destpath" should contain the file extension (JPG, JPEG).

It is also possible to create PNG files. To do so use the extension PNG.

frompage: the first exported page, starting with **1**

to_page: the last exported page

jpegres: - low-word (0000XXXX): the resolution for the JPEG file (default = 72),

hi-word: various options:

The lower nibble of the higher word is used to select the color depth.
It may may have this values:

1 : 1 bit monochrome dithered
2 : 1 bit monochrome not dithered
3 : 8 bit color
4 : 8 bit gray
otherwise: 24 bit color

The high byte of the high word is used to select the JPEG compression level (ignored for PNG)

6.1.1 Similar functions

pdfMakeJPEG uses the same functions as pdfMakeImage and works the same way.
The only difference is that it does not expect UTF8 strings. It was mainly provided for compatibility to WPViewPDF Version 2.

```
function pdfMakeJPEG(filename: PAnsiChar; password: PAnsiChar;
  licname, lickey: PAnsiChar;
  liccode: Cardinal;
  destpath: PAnsiChar; // Use %d store page number !
  frompage, to_page: Integer;
  jpegres: Integer): Integer; stdcall;
```

pdfMakeImageW works like pdfMakeImage but expects unicode strings.

```
function pdfMakeImageW(filename: PWideChar; password: PWideChar;
  licname, lickey: PWideChar;
  liccode: Cardinal;
  destpath: PWideChar; // Use %d store page number !
  frompage, to_page: Integer;
  jpegres: Integer): Integer; stdcall;
```

6.2 pdfConvertToTIFF - convert selected PDF pages to TIFF

This method is only available in WPViewPDF "PLUS"

This function is exported by the engine DLL to make it easy to convert PDF pages into TIFF files. No object has to be created and no initialization is required.

This function defined as

Pascal:

```
fktpdfConvertToTIFF = function(
   filename: PAnsiChar;
    password: PAnsiChar;
    licname, lickey: PAnsiChar; liccode: Cardinal;
    destname: PAnsiChar;
    // filename for created TIFF file
    frompage, to_page: Integer; tiffres: Integer // low word = resolution
    ): Integer; stdcall;
```

alternatively

```
function pdfConvertToTIFFW(
  filename: PWideChar;
  password: PWideChar;
  licname, lickey: PWideChar; liccode: Cardinal; destname: PWideChar;
  // filename for created TIFF file
  frompage, to_page: Integer; tiffres: Integer // low word = resolution
  ): Integer; stdcall;
```

Note: The unit WPViewPDF initializes the function pointers wpview_....

C:

```
stdcall int pdfConvertToTIFF(
  char *filename,
  char * password,
  char * licname,
  char * lickey,
  long liccode,
  char * destpath,
  int frompage,
  int to_page,
  int tiffres);
```

VB

```
Declare Function pdfConvertToTIFF Lib "wPDFViewDemo03.dll" _
                 Alias "pdfMakeJPEG" _
                 (ByVal zfilename As String, _
                  ByVal zpassword As String, _
                  ByVal zlicname As String, _
                  ByVal zlickey As String, _
                  ByVal liccode As Long, _
                  ByVal zdestpath As String, _
                  ByVal frompage As Long, _
                  ByVal To_page As Long, _
                  ByVal tiffres As Long) As Long
```

Return Value:

The count of converted pages.

Parameters:

filename: the full path to the input PDF file - please pass a UTF8 string

password: optional user password required to open the PDF file

licname, lickey, liccode: when using registered version use your license data here

destpath: the name of the created image file. Note, unlike with pdfMakeImage only one file will be created. Please pass a UTF8 string.

frompage: the first exported page, 0 based

to_page: the last exported page. (Example: To export the first page uses 0,0)

tiffres: - low-word (0000XXXX): the resolution for the TIFF file (default = 200),

hi-word: various options:

0 = create a CITTFAX compressed, monochrome TIFF file
1 = create a CITTFAX compressed, monochrome TIFF file without dithering
2 = create a 24 bit LZW compressed TIFF file

Alternative:

```
function pdfConvertToTIFFW(filename: PWideChar; password: PWideChar;
  licname, lickey: PWideChar;
```

```
  liccode: Cardinal;
  destname: PWideChar; // filename for created TIFF file
  frompage, to_page: Integer;
  tiffres: Integer // low word = resolution
  ): Integer; stdcall;
```

works like **pdfConvertToTIFF** but requires unicode instead of UTF8 strings.

Simple conversion demo in Delphi - uses 2 TEdit:

```
uses ....,WPViewPDF3;

{$I PDFLicense.INC}

....


a) in OnCreate load the DLL
procedure TForm2.FormCreate(Sender: TObject);
begin
  WPViewPDFLoadDLL(
     ExtractFilePath(Application.Name) + WPViewPDF_DLLName); // 'wPDFViewPlus03.dll');
end;

b) On Button click convert the file

procedure TForm2.Button1Click(Sender: TObject);
begin
   if not assigned(wpview_pdfConvertToTIFFW) then
       ShowMessage('wpview_pdfConvertToTIFFW not found')
  else ShowMessage(  'Convert Result=' +
    IntToStr(
     wpview_pdfConvertToTIFFW(
      PWideChar(Edit1.Text),
      '', // password
      PWideChar(WPViewPDF_LicName),
      PWideChar(WPViewPDF_LicKey),
      WPViewPDF_LicCode,
      PWideChar(Edit2.Text),0,10,
      300 // 300 dpi, monochrome
      )));
end;
```

6.3 pdfPrint - PRINT PDF function

If you need to print from any application you can use some simple code which imports the pdfPrint function directly. You do not need to create any control or any form for it. Simply import this function from the DLL. This works in VB, in Delphi, in .NET. (Please see the declarations at bottom of this page)

Important: Please make sure that pdfPrint is not called before the previous call to pdfPrint has been completed. For example disable the menu item which was used to start the printing process before the call and enable it again after the method has been returned.

pdfPrint will return the number of pages, the value -1 if an error happened (more information is available using DebugView). The value -2 is returned when the method was called while a previous job within the same thread was not completed. When used from multiple threads internally the calls are automatically serialized using critical sections.

Options:

Several parameters can be passed inside the option string.

The string "options" can contain several parameters. They need to be placed in quotes (") and separated by comma characters, example:

options = "\"FROM=1\",\"TO=2\"" to print pages 1 to 2.

This options are supported:

Standard print options:

PRINTER =xxx - select printer name
COPIES =N - select count of copies
FROM =N - the first page (1..)
TO =N - the last page
COLLATE =1 - enable collate mode

Print as bitmap:

LOWQUALITY*) =1 - buffer all output to monochrome bitmap in screen resolution
USEBITMAP =1 ... 10. A colored bitmap will be sent to the printer. The resolution is the printer resolution defined by the value.
Suggested values is 2. Using this settings embedded fonts can be reproduced more thoroughly.
BUFFERED*) =1 - buffer all output to monochrome bitmap in [BUFFERRES] dpi.
BUFFERRES*) =X - resolution for the buffered printing. Default = print resolution / 2

Select paper tray:

TRAY1 =N - printer tray for first page
TRAY2 =N - printer tray for all pages

Select media type

MEDIATYPE = N - this must be a valid media type identifier

Select duplex mode:

DUPLEX select duplex mode:
0= simplex,
1=horizontal,
2=vertical

Stretch the pages:

STRETCH = N
0 : Print page on paper ignoring the physical margins
1 : Reduce the print size to printable area
2 : Reduce the print size to fit the physical page (default)
3 : Scale the print size to fit printable area
4 : Scale the print size to fit the physical page

NO_OFFSET = 1 - with this setting the engine will not subtract the physical offsets

Print watermark metafiles:

WATERMARK =name of a enhanced meta file to print a watermark on all pages (stretched to page size!)
OVERPAGE =name of a enhanced meta file to print a drawing over all pages (stretched to page size!)

Print header and footer texts or page numbers:

HEADERFONT*) =name, default = Arial
HEADERSIZE*) =size in pt, default = 11
The mode can be used to set the font name for the header text.
FOOTERFONT*) =name
FOOTERSIZE*) =size in pt
Use it to set the font name for the footer text.

HEADERL*) - string to print in header on left side (at the top of printable area)
HEADERC*) - string to print in header centered
HEADERR*) - string to print in header on right side
FOOTERL*) - string to print in footer on left side (at the bottom of printable area)
FOOTERC*) - string to print in footer centered
FOOTERR*) - string to print in footer on right side

In these strings You can use the placeholder [#] to print the current page number and [##] to print the page count.

Select Paper width/Height

PAPERWIDTH = ...

If larger than 0, set the value for the DEVMODE dmPaperWidth member which will be set in the printer structure.

PAPERLENGTH = ...

If larger than 0, set the value for the DEVMODE dmPaperLength member which will be set in the printer structure.

PAPERSIZE = ...

If larger than 0, set the value for the DEVMODE dmPaperSize member which will be set in the printer structure.
If -1 is used, the value will be not set and the default paper size defined for the printer will be preserved. (Switch off automatic paper size switching)

Use Printer ESCAPE codes:

WRITEPRINTER - string of hex encoded characters to be sent to the printer using the Escape() function [1]
WRITEPRINTERBEFORE - string of hex encoded characters [2]
WRITEPRINTERAFTER - string of hex encoded characters [3]
WRITEPRINTERBEFORESTART - string of hex encoded characters [4]

[1] will be sent before each page
[2] will be sent before all pages
[3] will be sent after all pages, before EndDoc()
[4] will be sent before the document is started, before StartDoc()

Switch off any modifications to the DEVMODE structure of the printer:

DONTSETDEVMODE=1

Modify the way fonts are drawn:

OUTLINEFONTS=x
0: Renderer only draws embedded fonts as outlines which are either subsets or not also installed
1: renders all fonts as outlines, also installed fonts
2: renders embedded fonts as outlines

STDGDI=1
Selects the standard GDI renderer instead of GDIPLUS. This can result in smaller print files and faster output. For difficult PDF files it can cause a decrease in output quality.

Initialize JBIG2 plugin

JBIG2TOOL=*{dll}convert.exe {in} -o {out}*

Debug Options:

LISTTRAY =1 - list all paper trays to debug console
LISTPRINTER =1 - list all printer names to debug console

PROGRESSWND*) =handle of a window to receive progress messages. Must be passed as integer number.

DONTWAIT =1 - the function returns quicker

Declaration of the print function in Delphi

```
fktpdfPrint = function(filename: PAnsiChar; password: PAnsiChar;
```

```
    licname, lickey: PAnsiChar; liccode: Cardinal; options: PAnsiChar)
    : Integer; stdcall;

  fktpdfPrintW = function(filename: PWideChar; password: PWideChar;
    licname, lickey: PWideChar; liccode: Cardinal; options: PWideChar;
    data: Pointer; datalen: Integer): Integer; stdcall;
```

Note: The unit WPViewPDF initializes the pointer wpview_....

Declaration of the print function in C

```
stdcall int pdfPrint(
 char *filename, char *password:
 char *licname, char *lickey: PChar,
 unsigned long liccode,
 char *options);
```

MSVC++ 6.0 / MFC Example:

```
HINSTANCE hiDll = LoadLibrary( "wPDFViewDemo03.dll" );
// int pdfPrint(string filename, string password, string license_name, string license_key, i
typedef int( __stdcall * TypePdfPrint) ( char*, char*, char*, char*, unsigned long, char* );
TypePdfPrint pDllPdfPrint = (TypePdfPrint) GetProcAddress( hiDll, "pdfPrint" );
if(pDllPdfPrint)
{
   CString csOptions = "HEADERC=" + csFilePath + ",FOOTERC=" + csFilePath;
   CString csLicPwd  = ""; // empty
   CString csLicName = "..."; // add license data
   CString csLicKey  = "...";
   int     iLicCode  = ...;

   int iR = pDllPdfPrint( csFilePath.GetBuffer(0),
         csLicPwd.GetBuffer(0),
         csLicName.GetBuffer(0),
         csLicKey.GetBuffer(0),
         iLicCode,
         csOptions.GetBuffer(0) );
   if(iR <= 0) AfxMessageBox( "Cannot print the file " + csFilePath );
}
```

Visual Basic 6 Example:

```
Private Declare Function pdfPrint Lib "wPDFView03.dll" ( _
         ByVal strFilenames As String, _
         ByVal strPassword As String, _
         ByVal strLicName As String, _
         ByVal strLicKey As String, _
         ByVal lngLicCode As Long, _
         ByVal strOptions As String _
      ) As Long


Private Sub Command1_Click()
```

```
   If pdfPrint(Text1.Text, "", "LIC_NAME", "LIC_CODE", 0, "") <= 0 Then
      MsgBox ("Cannot print PDF file")
   End If
End Sub
```

.NET C# Example:

```
// .NET C# Code to print directly using the wPDFViewDemo02 Engine DLL
// using System.Runtime.InteropServices;
[DllImport("wPDFViewDemo03.dll", CharSet=CharSet.Ansi)]
 public static extern int pdfPrint(string filename, string password,
 string license_name, string license_key, int license_code,
 string options);
private void Print_Click(object sender, System.EventArgs e)
{
 pdfPrint(FileName.Text,
 "", // Password or ""
 "","",0, // License Information
 ""); // Options
}
```

.NET VB Example:

```
// .NET VB Code to print directly using the wPDFViewDemo02 Engine DLL
// requires System.Runtime.InteropServices;
<DllImport("wPDFViewDemo03.dll", CharSet:=CharSet.Ansi)> _
Public Shared Function pdfPrint(ByVal filename As String, ByVal password As String, _
ByVal license_name As String, ByVal license_key As String, ByVal license_code As Inte
ByVal options As String) As Integer


Private Sub Print_Click(ByVal sender As Object, ByVal e As EventArgs)
    WinForm.pdfPrint(Me.FileName.Text, "", "", "", 0, "")
End Sub
```

Delphi Example

```
function pdfPrint(filename: PChar; password: PChar;
  licname, lickey: PChar; liccode: Cardinal;
  options: PChar): Integer; stdcall;
  external 'wPDFViewDemo03.dll' name 'pdfPrint';
```

Note: The unit WPViewPDF initializes the pointer wpview_....

Please update the code to use wPDFView*Demo*03.dll or wPDFView03.dll.

6.4 pdfMerge - Merge PDF files (PLUS Edition)

If you need to merge different PDF and create one new file you can use the function pdfMerge. It receives the license codes and a list of files (comma delimited) .

The **PLUS** addon comes with an extra DLL "tiff_to_pdf.dll" which helps to also merge black and white TIF files which were produced by a scanner as if they were PDF files!

Please place this DLL in the same directory as the WPViewPDF main DLL.

If you intend to use the pdfMerge function (or the stamping feature) on an internet or intranet server, you need a special WEB-License. Please see order page.

Declaration of the merge function in VB (not .NET)

```
Private Declare Function pdfMerge Lib "wPDFViewPlus03.dll" ( _
            ByVal strFilenames As String, _
            ByVal strNewFile As String, _
            ByVal strPassword As String, _
            ByVal strLicName As String, _
            ByVal strLicKey As String, _
            ByVal lngLicCode As Long, _
            ByVal lngLicPlusCode As Long, _
            ByVal strOptions As String _
     ) As Long
```

Tip: If you need to use this method in ASP (not .NET) you can use VB to create a simple ActiveX class which exports just this method.

```
        Public Function pdfMerge_Access(ByVal strFilenames As String, ByVal
            pdfMerge_Access = pdfMerge(strFilenames, strNewFile, "", LicName
        End Function
```

Declaration of the merge function in C

```
stdcall int pdfMerge(char *filenames, char *newfile, char *password,
  char *licname, char *lickey, uint liccode, uint licpluscode,
  char *options);
```

Declaration of the merge function in Delphi

```
  fktpdfMerge = function(filename: PAnsiChar; newfile: PAnsiChar;
    password: PAnsiChar; licname, lickey: PAnsiChar; liccode: Cardinal;
    licpluscode: Cardinal; options: PAnsiChar): Integer; stdcall;
```

```
fktpdfMergeW = function(filename: PWideChar; newfile: PWideChar;
  password: PWideChar; licname, lickey: PWideChar; liccode: Cardinal;
  options: PWideChar): Integer; stdcall;
```

Note: The unit WPViewPDF initializes the pointer wpview_....

Declaration of the merge function in C#

```
// using System.Runtime.InteropServices;
[DllImport("wPDFViewPlus03.dll", CharSet=CharSet.Ansi)]
   public static extern int pdfMerge(string filenames, string newfile, strin
   string license_name, string license_key, int license_code, int license_pl
   string options);
```

Parameters:

filename: a list of filenames separated using comma, each filename in double quotes: "a.pdf","b.pdf","c.pdf"
newfile: the name of the new PDF file which should be created
password: the user password which should be used to open a PDF file
licname: your license name
lickey: the license key
liccode: the license code
licpluscode: obsolete, not used.

options:
This is a string with options, separated by comma

"DEBUG=1" switches on the debug mode. See debug console for messages
"CHECKEXIST=1" files which do not exist will be ignored
"TIFF2PDF=path" full path to converter DLL
"LOGFILE=path" logs errors in the specified file. Can be combined with DEBUG=1

"UPASSWORD=a" Set the user password for the new file to "a"
"OPASSWORD=b" Set the owner password for the new file to "b"
"SECURITY=x" Set the security PFlags
- Bit 3: Enable Print (default)
- Bit 4: Allow Modification
- Bit 5: Copy
- Bit 6: Add Annotations

"DELETESOURCE=1" After loading the input files, they are all (!) deleted.

In case the user- or ownerpassword is set, the file will be encrypted with 128 bit RC4 security.

Result

The Result is >0 if the operation was successful.

6.5 pdfGetInfoW

This method can be used to quickly fill a string list with the info items from a certain PDF file.

Declaration:

```
fktpdfGetInfoW = function(filename: PWideChar; buffer: PWideChar;
  buflen: Integer; password: PWideChar; licname, lickey: PWideChar;
  liccode: Cardinal; Option: Integer): Integer; stdcall;
```

You need to pass a buffer which is big enough to hold the data. The buffer (unicode char) will be filled with the items of the information record of the PDF file. The function returns the count of bytes which were copied.

```
var s, b : WideString; os: Ansistring; n: Integer;
begin
  os := WorkPath.Text + 'page_x%d.' + FileFormat.Text;
  if not assigned(wpview_pdfGetInfoW) then
  begin
    ShowMessage('function pdfGetInfoW is not available');
    exit;
  end
  else ShowMessage('Check Info Items');
  if OpenDialog1.Execute then
  begin
    s := OpenDialog1.FileName;
    SetLength(b, 10000);
    n := wpview_pdfGetInfoW(PWideChar(s), PWideChar(b), Length(b),
      '', // Password
      PWideChar(WPViewPDF_LicName), PWideChar(WPViewPDF_LicKey), WPViewPDF_LicCode,
      0);
    if n <0 then ShowMessage('Cannot open file!')
    else if n >= 0 then
    begin
       SetLength(b, n);
       ShowMessage(b);
    end;
  end;
```

```
end;
```

7 V3.0 notes

You are welcome to use this new Version 3 for the display of PDF files.

It was created mainly from scratch to include new technology, improve the drawing quality and to make it easier to service.
The new architecture made it possible for us to implement a multi threaded scrolling window.

Please note that the page number parameters are usually based on 0. (0=first page).

The exceptions are the property PageNumber and the PageNo command used by scripted stamping.

Also the command and API "PrintPages" uses 1 based page numbers (0 would mean "neutral").
If a command requires a page range, i.e. "4-7,9,15", this numbers are also 1 based.

Currently this features are not supported by WPViewPDF V3:

Image, fill and line *patterns and shading*, ICC Color Profiles, animated content, JPX and the rarely used JBIG1 images.

JBIG2 images can now be loaded through a plugin interface. We also added support for separation colors and DeviceN colors.

Especially PDF files which were created by "InDesign" are currently beyond the scope of the component. However PDF files created by embedded PDF creators, such as wPDF and wPDFControl (or competing products) and the popular printer drivers should be displayed and printed fine. If there are problems please send us a sample.

8 Whats New

28.9.2012: V3.06.5'
* change in JPEG routine to ignore internal JPEG errors

15.9.2012: V3.06.5
- improvement to color space decoding
- fix problem with named color space usage and stencil images

12.9.2012: V3.06.4
- improvement in handling compressed xref tables
- fix problem in prediction decoding code.

13.8.2012: V3.06.2
+ handle 2 Tr command (bold text)

24.7.2012: V3.06.1
- OnHyperlink message also gets URLs which do not start with "http:" or "file:"
- certain links did not scroll to correct y coordinate

19.7.2012: V3.06.0
+ OnViewerMessage now received the message code MSGPDF_SetFocus=205 when internally the focus is set.
+ handle PDF files with wrong page height definitions.
+ when writing PDF files empty images will be automatically replaced by white 1 pixel images so other PDF reader will not throw an error.

10.7.2012: V3.05.9
- fix problem with setting of info items.

24.6.2012: V3.05.8
- update to CCITT decoding to solve problem with few FAX files which were not rendered correctly.

18.6.2012: V3.05.7
- inline image were sometimes printed pink
- PLUS: improvement to better preserve PDF metafile data
- change in predition decoding
- use COMPDF_AdvancedFontDrawing with parameter 8 to force gdi text output

12.6.2012: V3.05.6
- special printing code to work around a problem when printing narrow bitmaps on certain printers.
- pdfPrint supports NO_OFFSET
- fix problem with text rendering
- changed printing strech mode 1. The bottom and right margins were too large and did not use the full printable area.
- fix problem in rendering with type 3 fonts
- fix problem in rendering with monochrome images when using white background color fill
- fix exception in PS interpreter when "c" was used outside of path

31.5.2012: V3.05.5
- fixes a problem which caused the PDF loading to fail on an application server
- possibility to disable shading with command COMPDF_DISABLE_SHADECOMMAND = 2010 (works globally)
+ command COMPDF_ZoomSaveRestore can be used to save / restore a zoom setting
+ use COMPDF_DrawObjectLocateAtXY to locate a draw object and COMPDF_DrawObjectReadProp to read its position

7.5.2012: V3.05.4
- load nested acro fields
- improvement to indexed color space
- use the commands COMPDF_SelectPaperWidth, - Length and - Size to specify the paper size the printer should use.
- options for pdfPrint to select paper size

18.4.2012: V3.05.2
- improved performance of pdfMerge function
- fixed problem with embedded JPEG images which were made transparent

16.4.2012: V3.05.1
* fixed a possible problem caused during multithreading
* after loading the first pages are painted at once before multi threading starts.
+ multithreading can be disabled with command COMPDF_DisableThreading=146
- fix for monochrome indexed images

30.3.2012: V3.05
- improved handling for fonts which name starts with @
* (WPViewPDF PLUS) **improved handling for fields. Now also text fields can be updated, which did not contain an appearance stream.**

26.3.2012: V3.04'
- DrawObjects with images could not be rendered into the PDF file

23.3.2012: V3.04
- improve handling of PDF files with corrupt font information which does not define font width
- fast subsequently loading of PDF data sometimes crashed the editor - this has been fixed.
- scroll tracking sometimes froze the viewer (.NET only)
- IStreams were not implemented correctly - so the LoadFromStream did not work with .NET before

6.3.2012: V3.03.4'
* improved display of grayscale JPEG images
- fixed small memory leak in function pdfMerge

5.3.2012: V3.03.4
- fix problem with SetFocus
+ it is now possible to select a different renderer for printing.
  use command COMPDF_UseGDIForPrinting (145) with parameter 1
  or, with pdfPrint, the option STDGDI=1
+ implemented the MapFont event to change font names

17.2.2012: V3.03.3'
- fix problem: AttachStream was not working
* SecurityOptions also disabled saving as text. This has been changed. Only saving as PDF is switched off.
- fix problem with encrypted PDF files which were using an empty file ID
+ new chapter in this manual: Commands

9.2.2012: V3.03.3
- printing did always try to change paper size and so scaling did not work as expected.
+ support for axial shading (solo and pattern)
- fixed a potential resource leak when form xobjects were used
+ improved: when saving to text (with GetPageText) You can choose as format "xyhtm". In this case the position will be added to the created and tags. This mode is only suitable when single pages are exported.

3.2.2012: V3.03.2
* improved support for CMYK graphics
* some improvements for color functions
+ function parser for separation color functions now also does if and ifelse statements

27.1.2012: V3.03.1
+ viewer sends message WM_PDF_EVENT with parameter MSGPDF_DblClick on double click
+ OnDblClick event in Delphi component - unlike usual event it also receives the PageNr.
+ fixes problem when merging AES encrypted files.
- fixes problem with saving some PDF files

+ command: **COMPDF_SaveBMPToClipboard** - when called within the DrawRect event the selected piece of the page will be copied as a bitmap
+ command: COMPDF_SaveBMPToFile. Here the selected rectangle will be saved to a BMP file.
- AttachStream was not working.
- fixes problem with scrollbars

6.1.2012: V3.03.0
+ Changed Clipboard Routine now places RTF, UNICODE and ANSI
+ Overlay draw objects are now printed
+ Now also renders fonts which fail to load by GDI+ (i.e. "Vivaldi")

3.1.2012: V3.02.9'
- fixes problem with decryption when FileID contained #0 character
- WPViewPDF DLL now uses english resources for error messages
* Better handling for font encoding. Solves problems with unknown characters in certain PDF files.
- fixes problem with inverse image masking
- fixes problem with width of special characters when fonts were not embedded

24.12.2011: V3.02.8
- fix problem when color is defined in paint path and not before
- fix option "DELETESOURCE" for pdfMerge
- PLUS - solves problem when saving PDF files with links.
* pdfMerge did not delete temporary files when merging PDF files
+ new option "DELETESOURCE" for pdfMerge
* improvement to text extraction to solve some problems when font used Encoding and ToUnicode properties

9.12.2011: V3.02.7'
+ Delphi VCL: Save methods will now raise the exception EPDFSecurityForbidsSaving if saving of document is not allowed. You as developer can override this at Your own risk.

Use command(COMPDF_DisableSecurityOverride,1) to disable this check.
+ improvement to decryption

7.12.2011: V3.02.7
+ Support for 128 bit AES decryption

2.12.2011: V3.02.6
- rgb was swapped by CMYK conversion
- fixed freezing problem when using in standard C application.
  Please call at first Command(1289,1) // COMPDF_CPP_PROGRAM
- Print function now scales down the page to print on a paper which was smaller. Use COMPDF_PrintUseScaling to specify scaling mode.
- the freetype dll "*wp_type1ttf.dll*" is now loaded explicitly from the same location as the main WPViewPDF engine and only if not found there, from the current system path.
- pdfMakeImage created wrong image format
- decode parameter of CIITT filter was not detected if relative object
- Images are not cached by default anymore. (COMPDF_CacheImages)
- fix problem with Encoding property of some fonts

25.11.2011: V3.02.5'
- PLUS - tiff to PDF was not working
- fixed problem in YCCK jpeg conversion
- fix new problem with grayscale indexed images
- fixed problem with threads not being closed when window was destroyed.

21.11.2011: V3.02.4
* improved clipping support when nested clipping regions were used
+ added support for **separation color** type 2
+ added support for **separation color** type 4
+ added support for DeviceN colors, also in images
+ Use Command WPDF_CacheImages,0 to disable the image caching to save memory
* in case a JBIG2 decoder was not set up the text "X JBIG2]" will be displayed on the pages which are missing the image (only on screen)
- fix problem with command DeletePages. The ranges 1-3 were not working as expected.

17.11.2011: V3.02.3
+ added support for **separation color** type 0 (much improved display of many government forms)
+ **when using StampText you can change the origin of the coordinates, create roman page numbering and use page offset**
* improved Type3 font support (avoids wrong recursion)
- improved postscript path handling
- fix problem with scrolling after search operation
- trigger OnChangePage event when scrolling text
- fix exception after right click in bookmark viewer

13.11.2011: V3.02.2
* The image handling has been changed to prepare and improve performance and support for different color spaces.
+ embedded JBIG2 data (**JBIG2Decode**) can now be decoded by external tool.
  Please use the command COMPDF_SetJBIG2Tool to initialize a plugin.
+ new color space handling

+ support for LAB colors in Images and on pages
+ added 3000 unicode names for conversion
- fix problem with character code #0 used inside text
- fix problem with certain image masks

3.11.2011: V3.02.1
+ use command COMPDF_PrintUseBitmaps to print using a bitmap buffer.

23.10.2011: V3.02.0
+ much improved for Type3 fonts with optimization for bitmap types.
- fixes problem for some PDF files which use encryption
- some fixes in PDF stream loading method
+ added hints to zoom panel (bottom right). Use COMPDF_SetShowHint,1,'1' to activate.

30.9.2011: V3.01.9'
- fix problem with display of some text which were using symbols encoded as unicode
- fix problem when saving files containing special colorspace references
- fix problem: image draw objects where using image ID+1. (COMPDF_MouseAddOneDrawObject)

20.9.2011: V3.01.9
+ CheckOwnerPassword can be used to pass owner password to lift save restrictions. TRUE is returned if the password was accepted.
- fix problem with streams in certain PDF files (problem was introduced in 3.01.7)

16.9.2011: V3.01.8
Replaced wp_type1ttf.dll - it was using MFC DLL.

8.9.2011: V3.01.8
* wp_type1ttf.dll now compiled from new freetype V2.4.6
* increased resolution of font renderer - improves display of bar-fonts
- improved vector rendering
+ when using SaveSelectionToStream it is now possible to specify a range of pages in the FileExt parameter. The syntax is "range;PDF". Rage is 1 based, i.e. 1-1
+ VCL: Added Plus.SavePagesToFile(filename, from, to). from and to is 0 based.
- fix problem with indexed images which used transparancy mask (caused red shading over barcodes)

30.8.2011: V3.01.7'
- improvement to image decoding and rendering
+ COMPDF_PrinterSetMediatype can be used to set the MediyType identifier for the printout. pdfPrint uses option MEDIATYPE=N
- LoadFromFileAsCopy was working like LoadFromFile (and locking the file)
- fix problem with certain encrypted PDFs which use an empty password
* fix problem decoding monochrome images which used an unusual colorspace syntax
- fix to handle the rotated pages some HP scanner write in PDF file
- fix to handle named color spaces
- fix in GDI+ renderer to set font name correctly

4.8.2011: V3.01.6
- fix in CCITT image decoding code
- fix in decoding indexed image code

- move pages method did not move deleted pages correctly

22.7.2011: V3.01.5'
- ICC based images are now decoded using the "Alternate" color space. This fixes the problem with blue becoming orange.

18.7.2011: V3.01.5
+ interactive page moving (PLUS)
* Printing is now selecting also smaller page sizes (important for export to document printer, such as PDF)
* updated VCL, .NET and OCX interface
* print renderer now uses system fonts if fonts were not embedded in PDF file. Use command COMPDF_AdvancedFontDrawing to change this:
    0: Print renderer only draws embedded fonts as outlines which are either subsets or not also installed
    1: renders all fonts as outlines, also installed fonts
    2: renders embedded fonts as outlines
* pdfPrint method now understands option "WRITEPRINTERBEFORESTART=..."

13.7.2011: V3.01.4'
* text extraction further improved. Fix stability problem with certain PDF files.
+ new command for "PLUS" edition: **COMPDF_MOVEPAGES = 600 - moves the selected pages after a certain page**. 0=first page
* several enhancements and optimations
- SetFocus was not working

11.7.2011: V3.01.4
+ PDFView demo now shows RTF extraction (Menu File/Extract page as RTF)
+ optimized text saving
- improved save routine

8.7.2011: V3.01.3
- improved print function. paperbin selection now also works with BeginPrint/EndPrint

6.7.2011: V3.01.2
- fix in bitmap rendering to work around GDI+ problem
* better calculation of current page
* COMPDF_GotoYPos used coordinates of current page. This has been changed. It now uses the absolute coordinates from top of text as it worked in V2
+ COMPDF_GotoPage can now use an optional string parameter which is used as y or x,y coordinate in 72 dpi world, and optionally, after %, the zoom value
- PrintRenderer now handles stencil images correctly. (fix problem with inverted images)
- pdfPrint option to send ESCAPE codes should now work
- fix in save routine - Colorspace property and Annotation were sometimes not saved correctly which caused problems in Acrobat Reader
- when merging PDF files setting the info items now works
- setting the paper bin when printing now works (COMPDF_SelectPrinterBin0)

29.6.2011: V3.01.1
- pdfPrint did not work properly.
- page ranges ("1-3") were not correctly interpreted. They are now always 1 based, as it was in WPViewPDF V2. (see Page rotation)

- MakeBitmap now always rotates according to page setting
- updated PDFView demo (Delphi)
- ViewOption "ShowDeletionCross" now works. If active, deleted pages will not be hidden but crossed out.

28.6.2011: V3.01.0
* MSGPDF_CHANGESELPAGE is now sent when user changes page selection
* zooming now tries to maintain the position in the text - the same line should be displayed in the middle of the window. This also works with
  MouseWheel zooming with ctrl key - here the position at the mouse pointer is locked.

15.6.2011: V3.0.9
+ ActiveX (OCX) for IDEs such as VisualBasic 6 is included now.
- Page up/down navigation has been improved
* page is no better centered in viewer

10.6.2011: V3.0.8
+ new command: COMPDF_GotoNamedDest can be used to jump to a named destination
+ new command: COMPDF_DrawObjectLocateAtXY read the name of a draw object at the mouse position or a given x,y position.
- fixed bug in CCITT decoding method and added possibility to skip incomplete data in G3 decoded images
- fixed problem in outline handling (jumps)
- fixed problem with clicks on scrollbars
* improved saving of PDF, which now better preserves PDFA Information
* the Delphi unit WPViewPDF3 now always included PDFLicense.INC and uses the license keys.
- fix for command COMPDF_GotoPrev - it didn't work on last page

1.6.2011: V3.0.7
+ we now include a .NET wrapper compiled for Framework 3. (full version includes source)
- fixed problem with locating PDF resources
- fixed problem with charsets
* Delphi Demos are now installed in directory Demos.VCL
- fixed problem with command COMPDF_ShowGotoPage
- fixed problem with command COMPDF_GotoYPos

23.5.2011: V3.0.6c
- fixed problem: sometimes an italic font was used instead of the regular.
* implements work around for one mistake found in some XREF tables.
* improves XREF reconstruction
- fix bug: info items retrieved from a PDF file were not provided as unicodes

16.5.2011: V3.0.6
- improvement for small embedded images
- fix for character set decoding problem

- new code to display highlighted text (find method)
+ new possibility to draw highlighting rectangles on page
+ new DLL function to read info items from PDF

13.5.2011: V3.0.5
+ The DLL exports a new function: pdfGetInfoW. It makes it possible to quickly read a PDF file info items.
* modification to scroller control to allocate less memory as buffer
- some improvements to printing code
- fix to handling of images with alpha channel

6.5.2011: V3.0.4
- OnHyperlinkPage and OnHyperlinkWWW is now working
- fix exception when moving shapes and redraw problem
- improved display of PDF watermarks
- change in printing routine to lock screen. This helps to reduce memory consumption since
  caching is deactivated while printing.
- updated to multi-page printing

4.5.2011: V3.0.3
+ support printing of multiple pages on one paper sheet. To activate use COMPDF_PrintUseScaling.
- improved display of images which are build up from very small bitmap elements
- fix redraw problem which caused artefacts after zooming
+ now it is possible to move a shape to a different page. See wpModifyExistingObj.
+ it is possible to delete a named shape
- fixes problem with certain fonts which were not embedded
- fixes run width problem with some Type3 fonts
- fixes character set problem of some fonts
- fixes problem of wrong position of certain annotations (comments added on iPAD)

28.4.2011: V3.0.1
- solves problem with texts which use fonts which are not embedded
- it is now possible to move objects between pages by code

22.4.2011: V3.0.1
- improved display routine to avoid artefacts in the page scroller
- improved find routine works faster and locates the position of the found text
- new COMPDF_SetExViewOptions to control frame lines and page numbers
- new COMPDF_SetPageNumberString to format the page numbers

20.4.2011:initial release V3.0

9 Changes to Version 2

WPViewPDF V3 is based on a new kernel. The DLL interface is very much like the V2 interface but we also added methods which accept widestring parameters.

We tried to mimic WPViewPDF Version 2 as closely as possible but, there are still

changes which were either required to optimize the performance or because options of WPViewPDF 2 became obsolete.

Some features have not been yet implemented into Version 3.

In general please note:

Whenever a page number is used it is based on the range 0 to PageCount-1.

The only exceptions are:
- the property PageNumber - it is based on the range 1...PageCount
- the numbers used by scripted stamping. We wanted to avoid to break old scripts, so the page numbers there are also starting with 1 instead of 0.
- page ranges, for example 1-2,5,7 which can be used for printing, page rotating and page deletion. With page ranges the first page is #1.
- PrintPages uses page numbers from the range 1-PageCount.
- Ranges which are passed as strings "from-to" are always 1 based to make it straight forward to use user input.

The property IsV3 can be used to determine if a V3 DLL was loaded or not.

Changes:

a) changed unit names.

The Delphi interface uses the units WPViewPDF3 and WPDF_ViewCommands instead of WPViewPDF1, and PDFViewCommands.

b) The DLL wp_type1ttf.dll always has to be installed with the application. Othewise the main DLL cannot be loaded.

c) In V3.0 this events do not work yet:
- OnHyperlinkWWW
- OnHyperlinkPage
- OnError
- OnLoadSection
- OnMapFont
- OnMailMergeGetText

d) PrintHDC works differently now. It should work now reliable.

e) GetPageText now expects an optionla format parameter. You can specify ". TXT", ".UNICODE", ".RTF" and ".HTML" text.

f) The new method WriteBitmap can be used to replace WriteJPEG.

g) COMPDF_PrintScannedDocuments is not used anymore and was removed.

h) The method MergeText does not work yet.

i) The flag ViewOptions.wpSelectClickedPage was renamed to ViewOptions. wpSelectPage.

10 License

WPViewPDF - Copyright (C) 2005-2011 by WPCubed GmbH.
St. Ingbert Str. 30,
81541 Munich. Germany.
All rights reserved.
WEB: www.wptools.de, www.PDFControl.com

General

The software supplied may be used by one person on as many computer systems as that person uses.

Single developer licenses are "named" - it is not allowed to pass one single license to a different developer once it was used for developing.

Group programming projects making use of this software must purchase a copy of the software for each member of the group. Contact WPCubed GmbH for volume discounts and site licensing agreements.

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

This documentation and the component 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 unless the damage was caused by gross negligence or wrongful intent of the manufacturer.

WPViewPDF uses the public zlib, jpeg and RC4 routines. It also uses the LZW decompression algorithm. WPViewPDF V3 also uses the FreeType DLL and optionally also the GDIPlus and AGG V2.4.

This License enables you to use the WPViewPDF technology in all your products and distribute it to your customers without paying any royalties under the following restrictions:

- 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. (no modules)
- This also prohibits the use of our technology in universal PDF creation tools such as virtual printer drivers. (no printer drivers) This also prohibits the use as a "special" PDF reader for such a generic PDF creation or PDF conversion tool.
- You may not use WPViewPDF in a tool which is mainly designed to manipulate (such as, but not limited to, "encrypt", "split", "merge", "stamp") PDF files. **(no PDF tools)**
- You may not develop a stand alone tool to print PDF, create bitmap or metafiles or RTF text from PDF files, such as a command line PDF2BMP tool. (no generic graphic extraction tools)

The use in a stand alone PDF viewer application requires this text in the "about" dialog and the manual:

Utilizes PDF Viewing technology by WPCubed GmbH - www.wptools.de

The last paragraph can be removed after paying a fixed price. It still many not be used with a general "pdf-tool".

WPViewPDF PLUS License

With this license you can save the loaded PDF files into a new PDF file. It is possible to change the PDF information, update fields and add images, texts and vector objects. Certain PDF pages can be marked to be excluded prior to save.

If you intend to use this new pdfMerge or the stamping or conversion feature on an internet or intranet server, you need a special **WEB-License**. Please see order page.

11 Credits

11.1 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 also uses the LZW algorithm for decompression.

11.2 LibTIFF Credits

Copyright (c) 1988-1997 Sam Leffler
Copyright (c) 1991-1997 Silicon Graphics, Inc.

Permission to use, copy, modify, distribute, and sell this software and its documentation for any purpose is hereby granted without fee, provided that (i) the above copyright notices and this permission notice appear in all copies of the software and related documentation, and (ii) the names of Sam Leffler and Silicon Graphics may not be used in any advertising or publicity relating to the software without the specific, prior written permission of Sam Leffler and Silicon Graphics.

THE SOFTWARE IS PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EXPRESS, IMPLIED OR OTHERWISE, INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

IN NO EVENT SHALL SAM LEFFLER OR SILICON GRAPHICS BE LIABLE FOR ANY SPECIAL, INCIDENTAL, INDIRECT OR CONSEQUENTIAL DAMAGES OF ANY KIND, OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER OR NOT ADVISED OF THE POSSIBILITY OF DAMAGE, AND ON ANY THEORY OF LIABILITY, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

Additional Credits:

Copyright (c) 2003 Ross Finlayson

Permission to use, copy, modify, distribute, and sell this software and its documentation for any purpose is hereby granted without fee, provided that (i) the above copyright notices and this permission notice appear in all copies of the software and related documentation, and (ii) the name of Ross Finlayson may not be used in any advertising or publicity relating to the software without the specific, prior written permission of Ross Finlayson.

THE SOFTWARE IS PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EXPRESS, IMPLIED OR OTHERWISE, INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

IN NO EVENT SHALL ROSS FINLAYSON BE LIABLE FOR ANY SPECIAL, INCIDENTAL, INDIRECT OR CONSEQUENTIAL DAMAGES OF ANY KIND, OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER OR NOT ADVISED OF THE POSSIBILITY OF DAMAGE, AND ON ANY THEORY OF LIABILITY, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

11.3 FreeType License

Copyright 1996-2002 by David Turner, Robert Wilhelm, and Werner Lemberg

Introduction

The FreeType Project is distributed in several archive packages; some of them may contain, in addition to the FreeType font engine, various tools and contributions which rely on, or relate to, the FreeType Project.

This license applies to all files found in such packages, and which do not fall under their own explicit license. The license affects thus the FreeType font engine, the test programs, documentation and makefiles, at the very least.

This license was inspired by the BSD, Artistic, and IJG (Independent JPEG Group) licenses, which all encourage inclusion and use of free software in commercial and freeware products alike. As a consequence, its main points are that:

o We don't promise that this software works. However, we will be interested in any kind of bug reports. (`as is' distribution)

o You can use this software for whatever you want, in parts or full form, without having to pay us. (`royalty-free' usage)

o You may not pretend that you wrote this software. If you use it, or only parts of it, in a program, you must acknowledge somewhere in your documentation that you have used the FreeType code. (`credits')

We specifically permit and encourage the inclusion of this software, with or without modifications, in commercial products. We disclaim all warranties covering The FreeType Project and assume no liability related to The FreeType Project.

Finally, many people asked us for a preferred form for a credit/disclaimer to use in compliance with this license. We thus encourage you to use the following text:

Portions of this software are copyright © 1996-2002 The FreeType Project (www.freetype.org). All rights reserved.

Legal Terms

0. Definitions

Throughout this license, the terms `package', `FreeType Project', and `FreeType archive' refer to the set of files originally distributed by the authors (David Turner, Robert Wilhelm, and Werner Lemberg) as the `FreeType Project', be they named as alpha, beta or final release.

`You' refers to the licensee, or person using the project, where `using' is a generic term including compiling the project's source code as well as linking it to form a `program' or `executable'. This program is referred to as `a program using the FreeType engine'.

This license applies to all files distributed in the original FreeType Project, including all source code, binaries and documentation, unless otherwise stated in the file in its original, unmodified form as distributed in the original archive. If you are unsure whether or not a particular file is covered by this license, you must contact us to verify this.

The FreeType Project is copyright (C) 1996-2000 by David Turner, Robert Wilhelm, and Werner Lemberg. All rights reserved except as specified below.

1. No Warranty

THE FREETYPE PROJECT IS PROVIDED `AS IS' WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. IN NO EVENT WILL ANY OF THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY DAMAGES CAUSED BY THE USE OR THE INABILITY TO USE, OF THE FREETYPE PROJECT.

2. Redistribution

This license grants a worldwide, royalty-free, perpetual and irrevocable right and license to use, execute, perform, compile, display, copy, create derivative works of, distribute and sublicense the FreeType Project (in both source and object code forms) and derivative works thereof for any purpose; and to authorize others to exercise some or all of the rights granted herein, subject to the following conditions:

o Redistribution of source code must retain this license file (`FTL.TXT') unaltered; any additions, deletions or changes to the original files must be clearly indicated in accompanying documentation. The copyright notices of the unaltered, original files must be preserved in all copies of source files.

o Redistribution in binary form must provide a disclaimer that states that the software is based in part of the work of the FreeType Team, in the distribution documentation. We also encourage you to put an URL to the FreeType web page in your documentation, though this isn't mandatory.

These conditions apply to any software derived from or based on the FreeType Project, not just the unmodified files. If you use our work, you must acknowledge us. However, no fee need be paid to us.

3. Advertising

Neither the FreeType authors and contributors nor you shall use the name of the other for commercial, advertising, or promotional purposes without specific prior written permission.

We suggest, but do not require, that you use one or more of the following phrases to refer to this software in your documentation or advertising materials: `FreeType Project', `FreeType Engine', `FreeType library', or `FreeType Distribution'.

As you have not signed this license, you are not required to accept it. However, as the FreeType Project is copyrighted material, only this license, or another one contracted with the authors, grants you the right to use, distribute, and modify it. Therefore, by using, distributing, or modifying the FreeType Project, you indicate that you understand and accept all the terms of this license.

11.4 IGdiPLUS

Copyright (C) 2008-2010 by Boian Mitov
mitov@mitov.com
www.mitov.com
www.openwire.org

This software is provided 'as-is', without any express or implied warranty. In no event will the author 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.

11.5 AGG

```
//----------------------------------------------------------------------------
// Anti-Grain Geometry - Version 2.4 (Public License)
// Copyright (C) 2002-2005 Maxim Shemanarev (http://www.antigrain.com)
//
// Anti-Grain Geometry - Version 2.4 Release Milano 3 (AggPas 2.4 RM3)
// Pascal Port By: Milan Marusinec alias Milano
//                 milan@marusinec.sk
//                 http://www.aggpas.org
// Copyright (c) 2005-2006
//
// Permission to copy, use, modify, sell and distribute this software
// is granted provided this copyright notice appears in all copies.
// This software is provided "as is" without express or implied
// warranty, and with no claim as to its suitability for any purpose.
//
//----------------------------------------------------------------------------
// Contact: mcseem@antigrain.com
//          mcseemagg@yahoo.com
//          http://www.antigrain.com
//
```

11.6 AES Decryption

```
(************************************************)
(*                                              *)
(*     Advanced Encryption Standard (AES)       *)
(*                                              *)
(*     Copyright (c) 1998-2001                  *)
(*     EldoS, Alexander Ionov                   *)
(*                                              *)
(************************************************)
```

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

There were no changes made to the original ElAES unit dated 27.3.2002

Stichwortverzeichnis

- E -

- J -

- P -