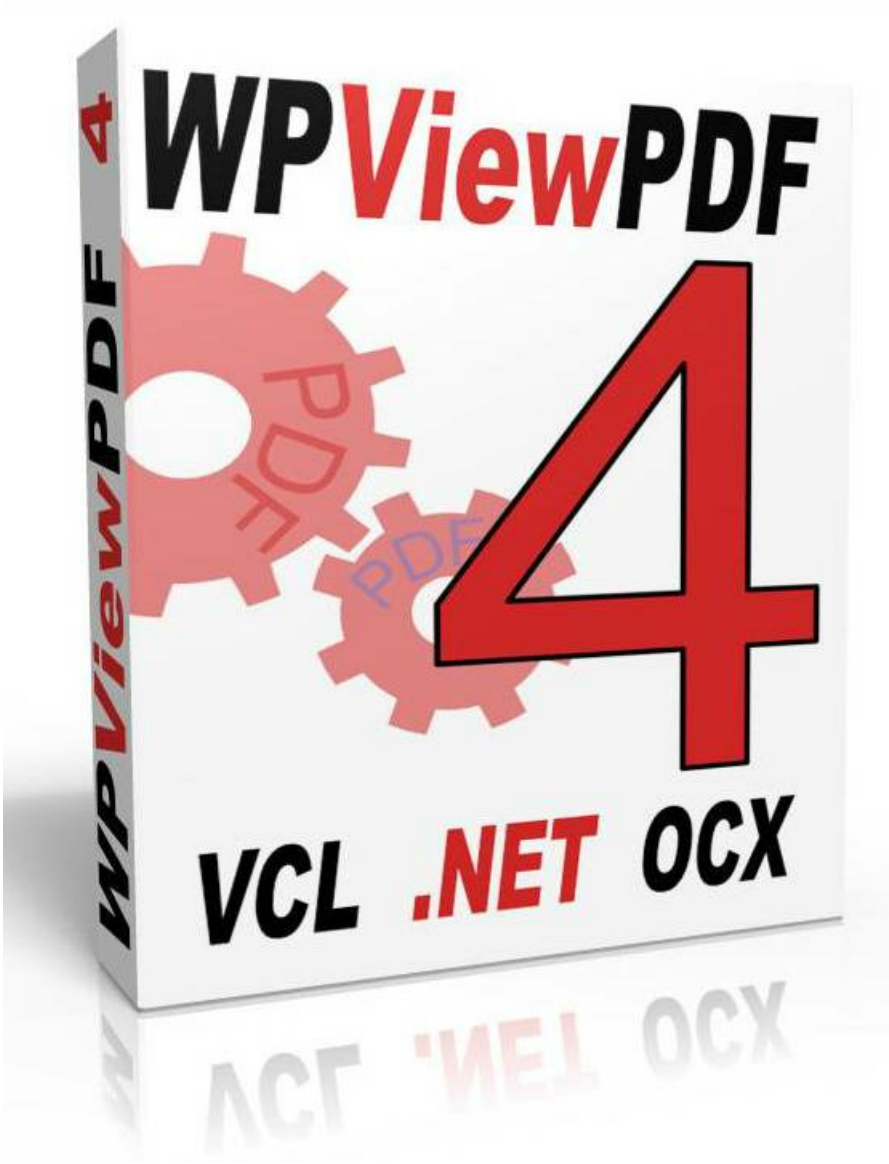



WPViewPDF Version 4

# Contents

# 1 Introduction


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.

WPViewPDF V4 PLUS introduces the ability to create square, highlight and text annotations to PDF files. The annotations can be edited or removed after the PDF file was saved and reloaded. The user can select text and highlight it using different colors. It is also possible to select black as highlight color which makes the text unreadable when printed or exported as image file. (Important: This feature does not delete the text)

You can now also select a PDF file which is then used as watermark for the PDF file. This makes it possible to apply letterheads to PDF files.

The Version 4 is the result of extensive work. Most time was used to implement the support for annotations. Still WPViewPDF 4 is completely compatible to WPViewPDF 3.

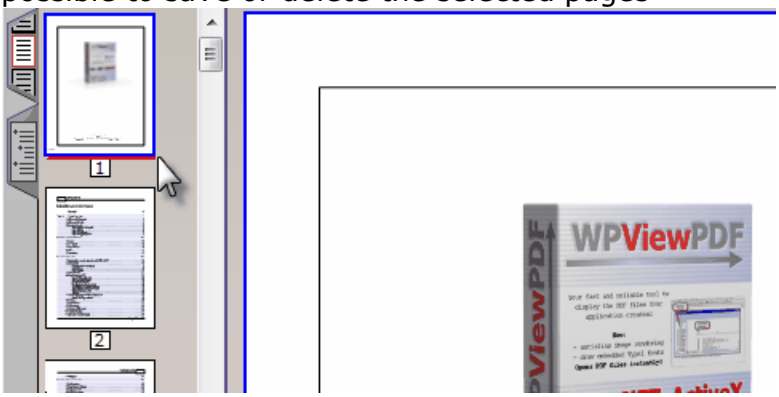
WPViewPDF V4 supports interactive draw objects which remain when loading a different PDF file. This makes it possible to apply the same stamps to different PDF files. WPViewPDF PLUS can also save those draw objects to XML and load them in this format!

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 version 3 and 4 use 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. Using a DLL which can be freely distributed, also JBIG2 support is provided.

WPViewPDF 4 PLUS can in contrast to the standard version save the loaded PDF data as new PDF file.

The user can also select pages in the thumbnail view and reorder the pages. It is possible to save or delete the selected pages




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.

**WPViewPDF is now available as 32 bit and 64 bit edition in the same product setup. (Delphi VCL and .NET).**

## 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.
- Highlight text or black it out before printing.
- Add highlight PDF annotations (PLUS)
- 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), and they can be stored in a database or send via e-mail by capable internet components, such as Synapse !
- WPViewPDF V4 PLUS can add many different kind of annotations, such as frames, highlights, underlines.
- With WPViewPDF V4 PLUS the use can select text and apply a highlight color or make the text background black (which makes the text invisible when printed)
- WPViewPDF V4 PLUS can add other PDF pages as watermarks - you can select a letter head PDF file and apply it to any other PDF files.
- WPViewPDF V4 PLUS can create acroform fields and attach text field widget to it to create an editable PDF form.

## History of WPViewPDF

WPViewPDF V1 was created in 2003, mainly as viewer for PDF files which were created by our own PDF engine. In the meantime WPViewPDF has become one of the most powerful PDF view components on the market. It can be used in different Windows programming IDS, such as Delphi and .NET. Also supported is an OCX Interface to be used legacy projects.

WPViewPDF V4 was released in February 2016 - it introduces a new action based API and most important the possibility to add annotation objects. The "document level draw objects" are a unique feature - they make it easy to apply the same

objects to many PDF files by simply reloading different PDF files.

## 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). You can also use a PDF file as background for a different PDF file for print out (with "PLUS" also save to a new PDF file)

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.

WPViewPDF is extremely useful if you need to store each printed output. You can create a database with all the output which was produced by your application together with the date when a letter was sent. This makes it possible to later check when and what was sent, and also resend an exact copy.

You can also use WPViewPDF to extract attachment data, i.e. ZUGFeRD XML

WPViewPDf *Standard* does not support the creation of fields, checkboxes and popups. The other supported annotation types and draw objects can be created and printed, but it is not possible to save them.

WPViewPDF *Standard* does not support the modification of fields and annotations which is described here.

BTW - WPViewPDF does not use cache files - it does not have to write images of the current page to temporary files to stay responsive.

WPViewPDF works in 32bit and 64bit mode (2 different sets of DLLs are provided)

Powerful interface code for Delphi-VCL (object pascal) and .NET (C#) is included.

For legacy products you can also use WPViewPDF with the included interface OCX.

## 1.2 WPViewPDF PLUS

The PLUS edition works like the standard version but You can also save the loaded PDF data. It can be your PDF work-horse, it merges PDF files, deletes and reorders pages and you can also use it to create and fill out PDF forms.

This makes it possible to

- delete or extract pages
- apply markers (stamps) to certain PDF pages
- move pages
- add highlights
- add PDF watermarks (extracted from other PDF files)
- add annotations
- add fields
- add checkboxes (different styles)
- add combo boxes
- add list boxes
- add popups
- modify fields
- modify annotations
- flatten or remove PDF annotations
- scale the PDF pages when saving to a new PDF file
- retrieve embedded files (attachments)

WPViewPDF PLUS also allows it to load and save draw objects which haven been placed on the "document level" in XML format.
Document level draw objects allow it to apply the same draw objects to all PDF files which are loaded.

With the PLUS 32 bit 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 WPViewPDF MakeImage

This is a special edition of the WPViewPDF DLL which exports the function **pdfMakeImageExt** only.
It does not include the viewer component or other DLL functions.

It was created to add support for PDF viewing with the powerful and highly recommended imaging library 

for Delphi, www.xequte.com.

The *MakeImage* functionality is also included in WPViewPDF Standard and WPViewPDF PLUS.

# 2 Installation

## 2.1 Delphi

**A) To register the component TWPViewPDF:**

Please open the file WPViewPDF_pack.dpk / WPViewPDF_pack_XE.dproj for Delphi XE and later and click on "Install".

You find the "Install" menu entry when you click with the right mouse button on the WPViewPDF..BPL project in the project manager.

**B) You can also create a component package yourself:**

From Delphi Menu select **File/New/Delphi Package**

it will create a new empty BPL project

To this BPL you need to add the pascal unit WPViewPDF_reg.pas

and as required packages "designide" has to be added.

Now you can click on compile and install.

When you put a TWPViewPDF instance on your form, you can make sure it loads the correct DLL by modifying the global string variable
**WPViewPDFDLLNAME**
in either the initialization section or after program start, before the form will be created.

**C) WPViewPDF can also used without installation into the IDE:**

To do so 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, wpShowPag
end;
```

For 64bit you need the DLLs wp_type1ttf64.dll and wPDFView...04**x64.**

The unit **WPViewPDF4** includes the tools to create the GUI in code.

**Important:**
In case you decide to rename the DLL WPViewPDF04 ... do not choose a file name which contains "Demo04".

## 2.2 C++ Builder

### Install with C++Builder

With RAD-Studio it is also possible to create Delphi packages - the compilation will also create the C++ OBJ and HPP files. We included the Delphi packages WPViewPDF_RT and WPViewPDF_DT which can be used with C++Builder as well.

Note: Using #pragma link ... units can be forced to be included in C++Builder project to fix the linker error "... was  not found".

**If you need pure C++Builder packages you can create those easily.**

This packages will be optimal for your system and use the correct paths.
First create the runtime package(s), then the design package.

1) Please select "New" from the file menu and choose "C++Builder Project".

The project must be selected as "**Runtime**" in the project options.

In Project manager click right no "Includes" and add this files to the Runtime (RT) package:
**WPViewPDF3.PAS**
**WPViewPDF4.PAS**
**WPDF_ViewCommands.PAS**

Build the project - the output path must be in the global search path.

(Under Options / Directories make sure the edit "Intermediate Output" is clear, otherwise the OBJ will not be created.)

To create files for 64bit save the package as a copy and make it a 64bit package.

The output path for object and HPP files should be a different subdirectory or ".\$(Platform)\$(Config)"
Alternatively you can simply add a second target platform and switch before compilation.

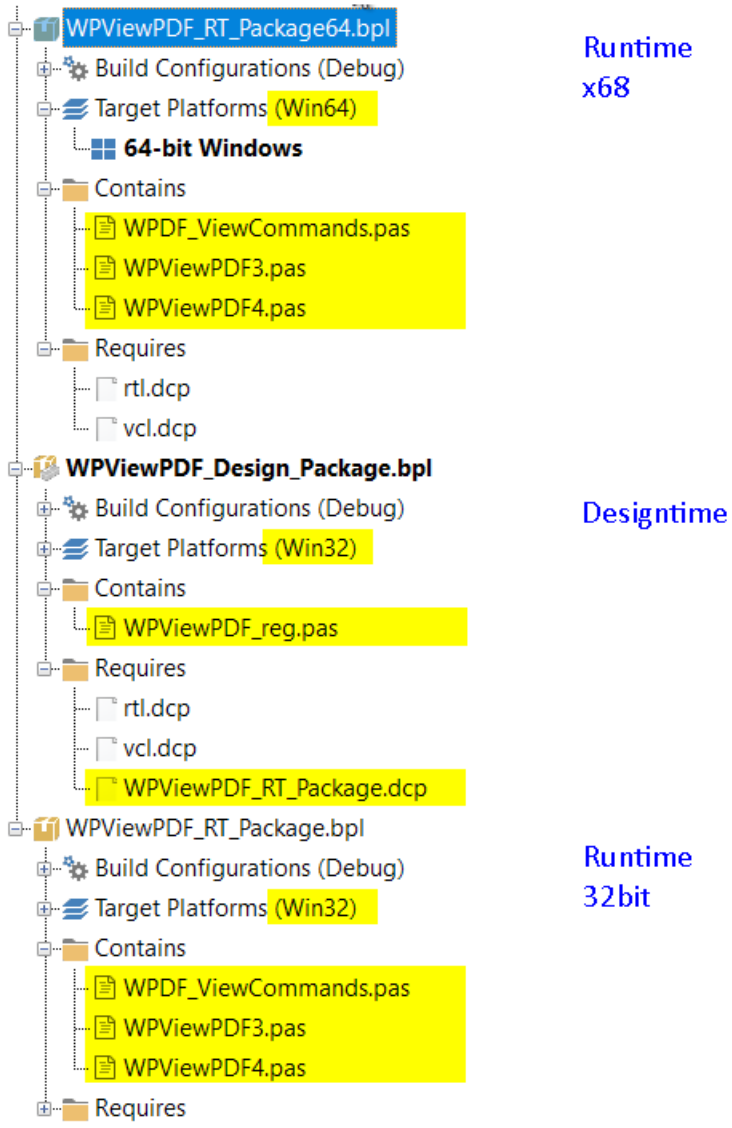
2) Please select "New" from the file menu and choose "C++Builder Project".

Change the package options, under "Description" select "**Designtime only**"

In Project manager click right no "Includes" and add this files to the Designtime (DSG) package:
**WPViewPDF_reg.PAS**

In the package options, under "**Delphi Compiler**/Compiling", "**Other Options**" add
*-LUDesignIDE* if you get the message "file not found DesignIntf.dcu".

You can now install this package.




### Do not forget to call the licensing function:

WPViewPDF1->ViewerStart("", your_lic_name, your_lic_key, your_lic_code);

to activate the control in your application.

### Do not forget to include the WPViewPDF DLLs with the created EXE.

## 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 wPDFView04.dll, wpdecodejp.dll and wp_type1ttf.dll to the executable directory. You can also control which engine DLL is loaded by the wrapper assembly. **Please use WPViewPDF.PDFViewer.SetDLLName to load the engine DLL (and the connected TTF and JBIG2 Dlls) from a different path.**

The 64bit edition requires wPDFView04x64.dll, wpdecodejp64.dll and wp_type1ttf64.dll. If your application was built for "AnyCPU" it will be executed as 64bit or 32bit depending on the host system. Please see our example code for loading the correct DLL in this case.

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.

**The .NET wrapper has been compiled in setting "AnyCPU".**

**It will load the 32bit engine when in 32bit mode, the 64bit engine when**

**in 64 bit mode.**

## 2.4 VB6

We have included a new version of the OCX Interface ***ViewPDF03.ocx*** to work in legacy VB6 applications. (***ViewPDF03.ocx can also be used with WPViewPDF V 4!)***
It makes use of the new methods included in WPViewPDF V3 and V4. 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
```

If you use multiple controls please use ViewerStart with each of the controls. It is necessary that the DLL path is the same in all this function calls.

## 2.5 Distribution

You may distribute the WPViewPDF4 runtime with Your application if all developers who were working (anywhere) on the project have a license for WPViewPDF 4. If your application is modular and only a few persons work on the PDF viewing part, you still need license for all the developers to have the right to include our component with your application.

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

wPDFView04.DLL, alternatively, wPDFViewPlus04.DLL

and wp_type1ttf.dll +wpdecodejp.dll.

**The DLL wp_type1ttf.dll is required**, if it is missing, WPViewPDF 4 runs slower

and not in best quality.

For 64bit you need the DLLs wp_type1ttf64.dll, wpdecodejp64.dll and wPDFView...04**x64.**

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.

Note: WPViewPDF includes JBIG2 decoding implemented in the module **wpdecodejp.dll** and, for 64 bit, **wpdecodejp64.dll**.

## 2.6 Troubleshooting

Note for Delphi Users:

WPViewPDF was designed to keep the loaded PDF file in memory even if the handle (window handle) of the viewing window was destroyed. The data will be released when the component is destroyed. This behaviour makes it possible to implement a docking feature.

To make sure the data is released when the form is closed (but not freed) call the method Clear or disable the compiler symbol **ENABLE_WNDRECREATE** in the file WPViewPDF3.PAS **or add the compiler symbol NOWNDRECREATE to the project conditionals.**

WPViewPDF V4 can be configured to use a different method to calculate the size for fonts which have not been provided with exact heights. (the font.height value is positive)

WPPDF_SetIProp(PDF.dll_pdf_env, **WPPDF_UseWindowsFontMapper**, 1 );

This will emulate the way Windows is looking for a matching font size.

WPPDF_SetIProp(PDF.dll_pdf_env, WPPDF_UseWindowsFontMapper, 0 );

With this setting the PDF engine will shrink the font, starting with the provided height, until the resulting height is equal to the height which was provided in the font structure.

In any case: In your text printing routines avoid positive heights for fonts. Use negative font heights for WYSIWYG text reproduction.

# 3 Create a PDF Editor

In this chapter we describe how to create a PDF view and edit application in Delphi and with C# in VisualStudio.

We also added information about localization and PDF attachment handling.

In both example the menu and the toolbar is created by generic code. The Delphi example makes use of the TAction classes.

## 3.1 Delphi Example

In this chapter we will show you how to quickly create a PDF viewer or PDF editor in Delphi using WPViewPDF V4 and its new "Actions" feature.

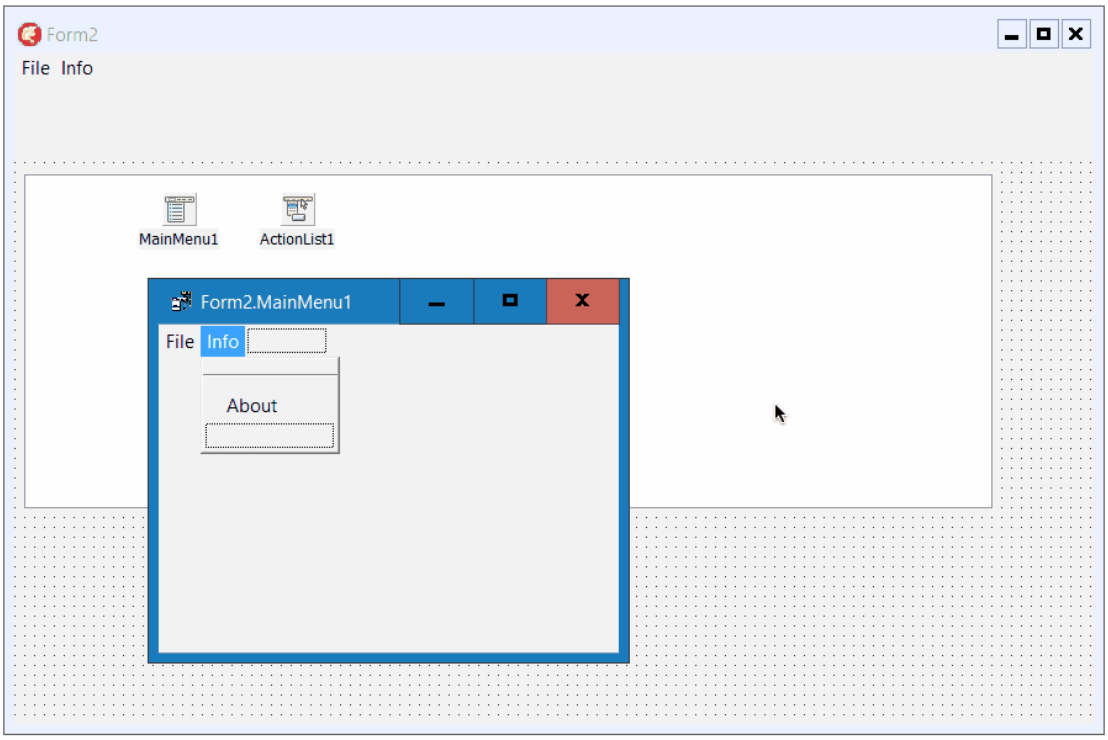
This example is initialized mostly by a scripting. It also uses some generic code which can copy&pasted directly from here. After the scripted initialization you can easily modify all aspects in the designer. You will find this demo in directory "WPViewer4".

First we create a new project. We select the single form template, although we want to implement viewing of multiple PDF files at the same time. Instead we of MDI we will use a page control to switch between PDF files.

*In this example we use the interactive designer to create the form. It is also possible to create it completely in code, and not much more difficult. You will find same code in project "PDFedit".*

### 3.1.1 Add the basic controls

On our main form we first add
**MainMenu**
**ToolBar**
**PageControl**
**ActionList**



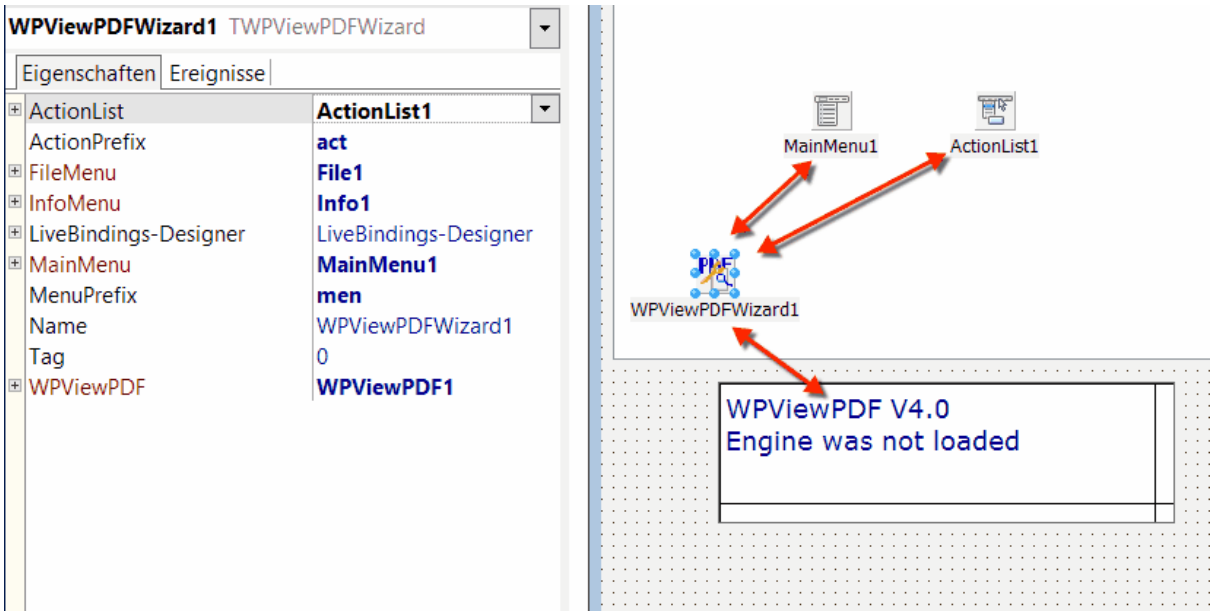

Inside of the MainMenu we create a "File" and and "Info" Menu. We insert some menu items there.

Now we also add a TWPViewPDF component on the main form. Its Visible property should be set to false.

The main TWPViewPDF component is used to initialize the action commands and also prevents the DLL to be unloaded which could badly affect the performance of the program.

### 3.1.2 Initialize the Menu and Actions

Now we add a **TWPViewPDFWizard** and connect it to the action list and the MainMenu.




Please also add event handlers for
WPViewPDFWizard1**ActionClick** and WPViewPDFWizard1**ActionUpdate.**

```
procedure TForm2.WPViewPDFWizard1ActionClick(Sender: TObject);
begin
  //
end;

procedure TForm2.WPViewPDFWizard1ActionUpdate(Sender: TObject);
begin
  //
end;
```

Now please save the form and then double click on the component WPViewPDFWizard1.

This will create menu items and action objects which are automatically connected to the ActionClick and ActionUpdate events.

To create the actions the method `WPPDFViewerInitMainMenu` is executed - it is implemented in the unit WPViewPDF4.pas.

```
procedure WPPDFViewerInitMainMenu( pdf : TWPViewPDF;
     Menu : TMainMenu;
     MenuNamePrefix : String;
     ActionList : TActionList; // may be nil
     ActionNamePrefix : String; // Prefix, i.e. wpv to create names for the
     OnClick : TNotifyEvent;
     OnActionUpdate : TNotifyEvent;
     FileMenu : TMenuItem=nil;
     InfoMenu : TMenuItem=nil;
```

```
    DoCreateAction : TWPDoCreateAction = nil;
    RequiredOptionalActions : String = '' );
```

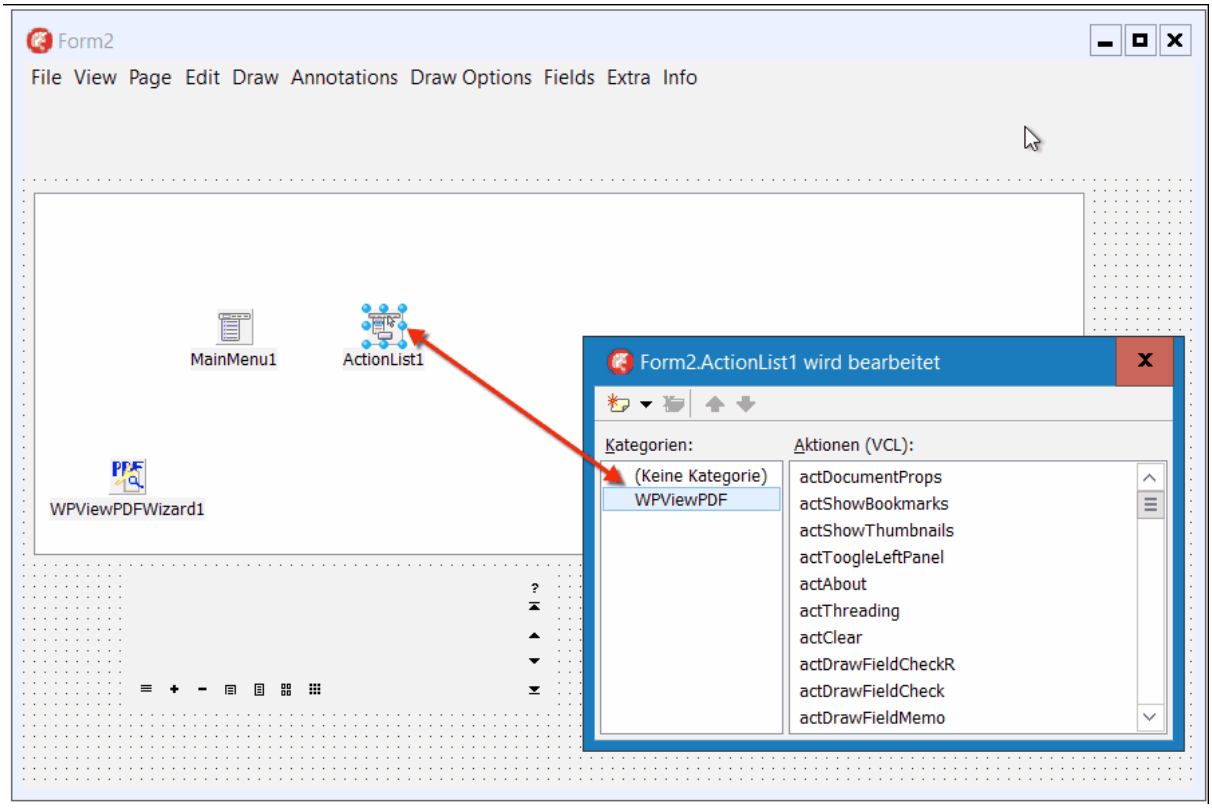
This utility function creates new menu entries. The new menus are inserted before the existing items. The first "File" and the last "Info" menu entry can be specified. They will be extended with new entries.

If you pass an ActionList actions will be created there. Those actions will be automatically be used by the newly created menu items.

Inside WPPDFViewerInitMainMenu we decided to create standard TAction classes and not special classes. Special classes could save other information, such the command name.
This has the advantage to use standard TActions is: you can cop&paste such actions into different projects, also if WPViewPDF was not installed.

When new actions are added to WPViewPDF they will be flagged to be not automatically included in the menu. To include them anyway please add the action name divided by come on parameter RequiredOptionalActions.

**Screenshot of the form after the menu items and actions were created**




The new menu items will be created between the menu items in the file menu and the last menu item in the info menu.

The newly created actions will be attached to the the new menu items.

Now you can delete the TWPViewPDFWizard, it is not required anymore. The action events will be directly assigned and not passed through the wizard.

To compile please also add **WPDF_ViewCommands** to the uses clause.

Hint: It is possible to modify the automatically created menu by some code.

With this code we modify the menu by moving some items which were initially created under "Extra" under a caption which makes the function easier to find.

```
for I := 0 to MainMenu1.Items.Count-1 do
begin
     // We move the "Modify Annotation" item from Special to menu "Annotations"
     if MainMenu1.Items[i].Tag = 6 then
     begin
        AddWeblink1.Parent.Remove(AddWeblink1);
        MainMenu1.Items[i].Add( AddWeblink1 );

        Addlinktopage1.Parent.Remove(Addlinktopage1);
        MainMenu1.Items[i].Add( Addlinktopage1 );

        men := TMenuItem.Create(MainMenu1.Items[i]);
        men.Caption:='-';
        MainMenu1.Items[i].Add( men );

        ModifyAnnotations2.Parent.Remove(ModifyAnnotations2);
        MainMenu1.Items[i].Add( ModifyAnnotations2 );
     end;
     // We move the "Add form field" item from Special to menu "Fields"
     // And also move the add link items
     if MainMenu1.Items[i].Tag = 8 then
     begin
        Addformfield1.Parent.Remove(Addformfield1);
        MainMenu1.Items[i].Add( Addformfield1 );
     end;
end;
```

### 3.1.3 Add code to initialize a PDF viewer

On the form there is a TWPViewPDF component - but it will never be used to load a PDF file.

Instead we create a new viewer automatically when we need to load a new file. This is done in the method **NewPDFDocument** which receives a filename.

It will create a new tab inside the PageControl. The tab will use the class **TPDFTabSheet** which holds additional variables and of course a reference to a

**TWPViewPDF** component.

Create a custom TTabSheet class:

```
type TPDFTabSheet = class(TTabSheet)
  wpviewpdf : TWPViewPDF;
  image     : TImage; // uses Vcl.ExtCtrls, we use that later
end;
```

First we need an handler for the event which is used to update the state of all actions:

```
procedure TForm2.WPViewPDF_DoChangeViewPage(Sender: TObject; const PageNr: Integer);
var i : Integer;
begin
  for I := 0 to ActionList1.ActionCount-1 do
     ActionList1.UpdateAction(ActionList1.Actions[i])
end;
```

This method creates a new tab and the components on it.

```
procedure TForm2.NewPDFDocument( filename : string );
var
    newTab :TPDFTabSheet;
begin
    newTab := TPDFTabSheet.Create(PageControl1);
    newTab.wpviewpdf := TWPViewPDF.Create(newTab);
    try
      newTab.PageControl := PageControl1;

      // And move it to the page control
      newTab.wpviewpdf.Align := alClient;
      newTab.wpviewpdf.Parent := newTab;

      if not newTab.wpviewpdf.LoadFromFile(filename) then
        raise Exception.Create( filename + ' cannot be loaded.');

      // Make sure the annotations work interactively!
      newTab.wpviewpdf.Command(COMPDF_ACRO_MAKEDRAWOBJ,'', 8192 );  // 0=all Annots!

      // Enables saving of annotations which have been added to the page
      newTab.wpviewpdf.Command(COMPDF_Ann_SetAnnotSaveMode, 1);

      // Standard Action Mode 'Click + Pan'
      newTab.wpviewpdf.Command(COMPDF_SetActionMode,'',1);
      // Set WPViewPDF Properties
      newTab.wpviewpdf.ViewOptions := [wpViewThumbnails,wpInteractiveThumbnails];
      newTab.wpviewpdf.ViewControls := [
            wpVertScrollBar,   // Scrollbar vertical
            wpHorzScrollBar,   // Scrollbar horizontal
            wpPropertyPanel,   // '?' Button
            wpNavigationPanel, // Up / down
            wpViewPanel,       // Zooming
```

```
            wpViewLeftPanel    // Thumbnails and Bookmarks -
            // also see COMPDF_ShowNavigation
          ];

        // assign the requires events
        newTab.wpviewpdf.OnChangeViewPage := WPViewPDF_DoChangeViewPage;

        // We need an image for optional background metafiles
        newTab.image := TImage.Create(newTab);
        newTab.image.Visible := false;
        newTab.image.Parent := newTab;

        // Add the tab to the page control
        newTab.Caption := ExtractFileName(filename);
        PageControl1.ActivePage := newTab;
      except
        newTab.PageControl := nil;
        // Loading failed, destroy the control!
        newTab.wpviewpdf.Parent := nil;
        newTab.wpviewpdf.Free;
        newTab.Free;
        raise;
      end;
end;
```

Consequently we add a function "**pdf**" which retrieves the WPViewPDF control which is currently active. If no tab has been added, the result will be nil.

```
function TForm2.pdf : TWPViewPDF;
begin
   Result := nil;
  if PageControl1.ActivePage<>nil then
  begin
     Result := TPDFTabSheet( PageControl1.ActivePage ).wpviewpdf;
  end;
end;
```

### 3.1.4 Create "OnClick"

The action OnClick handler is used to interact with the PDF viewer.

But first add a TOpenDialog and a TSaveDialog to the form - both are used by the code below.

Then fill the event **WPViewPDFWizard1ActionClick** with the following code. It was developed to work as OnClick handler for TAction, TMenuItem and any TControl class, such as TButton. The command ID is always encoded into "Tag".

The procedure fist uses the static TWPViewPDF instance to get more information about the special command. It reads if the command requires a parameter and of which kind the parameter should be. Some parameterkinds require a dialog, such as a file open or file save dialog. You will see in the code below, how this all works. You can copy&paste that code into your own applications, due to its universal nature.

```
procedure TForm2.WPViewPDFWizard1ActionClick(Sender: TObject);
var ac, param, paramkind, res  : Integer;
    actionname, s : string;
begin
  // This is a  universal Action handler ...
  if Sender is TAction then ac := TAction(Sender).Tag
  else if Sender is TMenuItem then ac := TMenuItem(Sender).Tag
  else if Sender is TControl then ac := TControl(Sender).Tag
  else exit;

  param     := WPViewPDF1.Command(COMPDF_ACTION_READ, 'param', ac );
   (* param: bit 2  --> need string *)


  paramkind := WPViewPDF1.Command(COMPDF_ACTION_READ, 'paramkind', ac );

   (*     // paramkind (used for string parameters)
          // 0: Pagenr as Int or string
          // 1: Fontname as string
          // 2: Color as Int or string
          // 3: PDF filename as string OPEN
          // 4: PDF filename as string SAVE
          // 5: text filename as string OPEN
          // 6: text filename as string SAVE
          // 7: image file name as string  OPEN
          // 8: JPEG file name as string   SAVE
          // 9: type @ options_comma_list
          // 10: options_comma_list
          // 11: options_for_DrawObjects
          // 12: Zoom Value as Int
          // 13: JPEG image file name as string to OPEN passed
          //     as "file=...",... + other params
          // 14: some text as string  passed as "contents=...",... + other params
          // 15: some multiline text as string  passed as
          //     "contents=...",... + other params
          // 16: Boolean  on/off 1/0

          // 50: Ask $hint$ yes/no
          // 51: Ask $hint$ yes/no/cancel
    *)

  actionname:= WPViewPDF1.CommandGetStr(COMPDF_ACTION_READ, 'name', ac );

  s := '';
  if paramkind=50 then
  begin
      if MessageDlg(pdf.CommandGetStr(COMPDF_ACTION_READ, 'hint', ac )
          +'?', mtConfirmation, [mbYes,mbNo], 0)=IDNO then  exit;
  end
  else if paramkind=51 then
  begin
       // Ask: YES, NO CANCEL
```

```
    iparam := MessageDlg(pdf.CommandGetStr(COMPDF_ACTION_READ, 'hint', ac )+
       mtConfirmation, [mbYes,mbNo,mbCancel], 0);
    if iparam=IDCANCEL then  exit;
    if iparam = IDYES then
       s := 'true';
end;

// bit 2 is set, we need a string parameter!
if (param and 2) = 2 then
begin
  if paramkind in [3,5,6,13] then
  begin
    if paramkind=3 then
         OpenDialog1.Filter := 'PDF Files (*.PDF)|*.PDF'
    else if paramkind=5 then
         OpenDialog1.Filter := 'Text Files (*.TXT)|*.TXT,*.*'
    else if (paramkind=3) or (paramkind=13) then
         OpenDialog1.Filter := 'Image Files (*.JPG)|*.JPG;*.JPEG';
    if not OpenDialog1.Execute then exit
    else s := OpenDialog1.FileName;

    // This parameter is used for JPEG Draw Objects
    if paramkind=13 then
       s := '"file=' + s + '"'; // + Color params   color= background-color

  end
  else if paramkind in [4,6,8] then
  begin
    if paramkind=4 then
       SaveDialog1.Filter := 'PDF Files (*.PDF)|*.PDF'
    else if paramkind=6 then
       SaveDialog1.Filter := 'Text Files (*.TXT)|*.TXT,*.*'
    else if (paramkind=8) or (paramkind=13) then
       SaveDialog1.Filter := 'Image Files (*.JPG)|*.JPG;*.JPEG';
    if not SaveDialog1.Execute then exit
    else s := SaveDialog1.FileName;
  end
  else if paramkind in [14] then  // A string
  begin
     s := '';
     if InputQuery(pdf.CommandGetStr(COMPDF_ACTION_READ,
            'hint', ac ), '', s) then
        s := '"contents=' + s + '"' // + Color params   color= background-color
     else exit;
  end
  else if paramkind in [15] then  // a multiline string  -> contents
  begin
     s := '';
     if InputQuery('', pdf.CommandGetStr(COMPDF_ACTION_READ,
            'hint', ac ), s) then
          s := '"contents=' + s + '"' // + Color params   color= background-color
     else exit;
  end
```

```
    else if paramkind in [0] then // Just a string
    begin
       s := '';
       if not InputQuery(pdf.CommandGetStr(COMPDF_ACTION_READ,
               'hint', ac ), '', s) then exit;
    end;
  end;

  // We can react on special actions
  if actionname='FileOpen' then
  begin
    NewPDFDocument(s);
    exit;
  end
  else if actionname='FileClose' then
  begin
    if PageControl1.ActivePage<>nil then
        PageControl1.ActivePage.Free;
    exit;
  end;

  if (pdf=nil) and ((WPViewPDF1.Command(
           COMPDF_ACTION_READFLAGS, ac ) and 16)=16) then
      res := WPViewPDF1.CommandStrEx( COMPDF_ACTIONNR, s, ac)
  else
  begin
    if (pdf=nil) then exit;
    res := pdf.CommandStrEx( COMPDF_ACTIONNR, s, ac);
  end;
end;
```

### 3.1.5 Create "OnUpdate"

This event is used to check the state down/disabled of each action.

```
procedure TForm2.WPViewPDFWizard1ActionUpdate(Sender: TObject);
var action : TAction;
    ac : Integer;
    state : Integer;
begin
      if Sender is TAction then
      begin
         action := TAction(Sender);
         ac := action.Tag;
         if ac>1 then
         begin
           // Either the PDF is loaded or we have a global operation
           if PageControl1.ActivePage=nil then
           begin
```

```
                action.Enabled := (WPViewPDF1.Command(
                        COMPDF_ACTION_READFLAGS, ac ) and 16)=16 ;
                action.Checked := false;
             end
             else
             begin
                state := pdf.CommandEx(COMPDF_ACTION_READSTATE, ac);
                // 1=Checked, 2=Disabled
                action.Enabled := (state and 2)=0 ;
                action.Checked := (state and 1)=1 ;
             end;
          end;
       end;
end;
```

### 3.1.6 Add all buttons to the tool bar

The buttons are created on the TToolbar at runtime. A list of action names is used select the actions to be used.
The position of the action in the list is expected to be the image index in the TImageList.

This is the list of actions:

```
const  _ActionButtons ='FileOpen,FileAppend,FileSaveAsPDF,SelectStd,SelectObjects,Zoo
 'SelectFillForm,DrawFieldEdit,DrawFieldCheck,DrawAnnotFrame,DrawAnnotHighlight,DrawA
 'DrawAnnotSymbol,DrawAnnotSquiggly,DrawAnnotHighlightText,DrawAnnotBlackText,' +
 'DrawTextline,DrawRect,DrawImage,DrawHighlight,DrawCircle,About';
```

We also a TImageList with the buttons.

We use this glyphs:

Note: WPViewPDf Standard does not support the creation of fields, checkboxes and popups. The other annotations and draw objects can be created and printed, but it is not possible to save them.

The procedure InitToolbar creates all buttons. It expects the actions to use the prefix ***act*** in their names.

```
procedure TForm2.InitToolbar;
 var NotFound : String;
   function Find(aName : String ) :  TAction;
   var i : Integer;
   begin
        for i := 0 to ActionList1.ActionCount-1 do
        if SameStr( ActionList1.Actions[i].Name, 'act' + aName) then
        begin
           Result := ActionList1.Actions[i] as TAction;
```

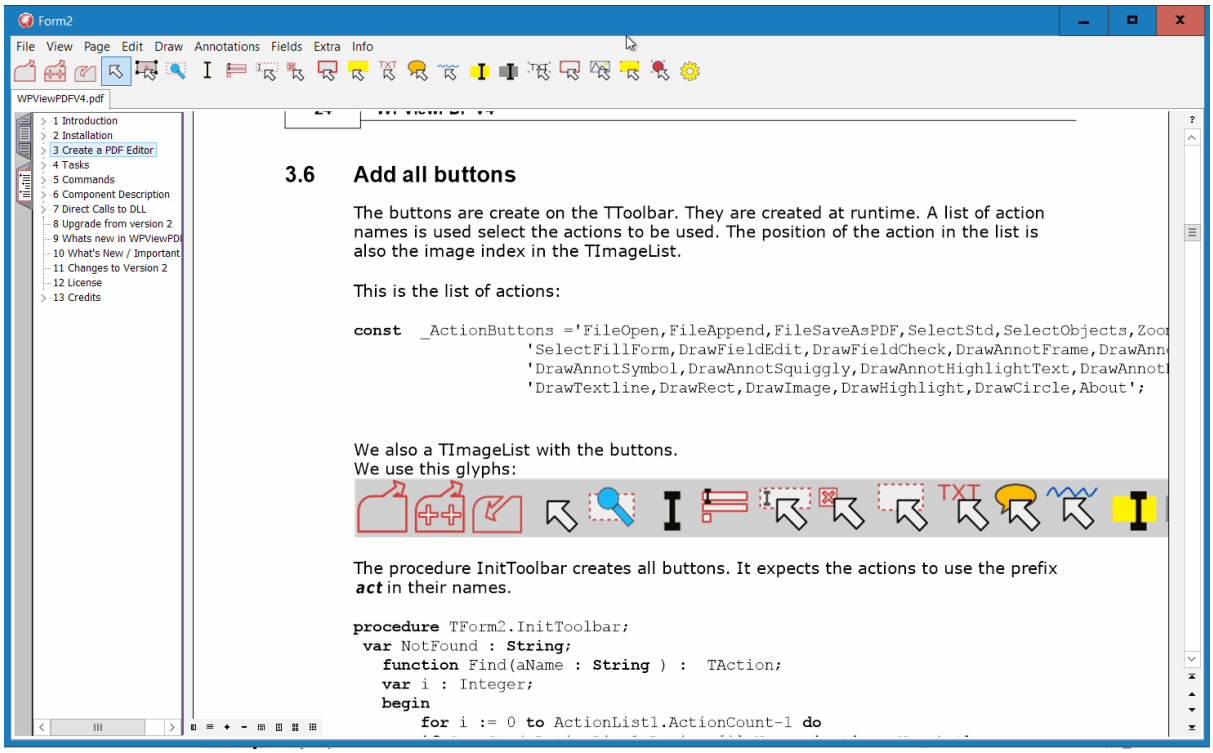
```
          exit;
        end;
        Result := nil;
        if NotFound<>'' then NotFound := NotFound + ', ';
        NotFound := NotFound  + aName;
   end;
   var str : TStringList; i : Integer; btn :TToolButton;
        action : TAction;
begin
 str := TStringList.Create;
 NotFound := '';
 try
   str.Sorted := false;
   str.CommaText := _ActionButtons;

   ToolBar1.Images := ImageList1;
   ToolBar1.ButtonHeight := ToolBar1.Images.Width;
   ToolBar1.ButtonWidth := ToolBar1.Images.Height;
   ToolBar1.Height := ToolBar1.ButtonHeight + 4;

   // create all buttons from our action list
   for I := str.Count-1 downto 0 do
   begin
       action := Find(str[i]);
       if action<>nil then
       begin
           btn := TToolButton.Create(ToolBar1);
           btn.ShowHint := true;
           btn.Width := ToolBar1.Images.Width;
           btn.Height := ToolBar1.Images.Height;
           btn.Parent := ToolBar1;

           action.ImageIndex := i;
           btn.ImageIndex := i;
           btn.Action := action;
       end;
   end;

   // For debug reasons
   if NotFound<>'' then
      ShowMessage('This WPViewPDF actions were not found' + #13 + NotFound);
 finally
  str.Free;
 end;
end;
```

### 3.1.7 Add "Form.Create"

The event **FormCreate** is used to initialize the application and load the PDF files passed as command line.
Here we also call the function which initializes the toolbar.

```
procedure TForm2.FormCreate(Sender: TObject);
var i  : Integer;
    s : String;
begin
  InitToolbar;

  // Do some initialisation
  PageControl1.Align := alClient;

  WPViewPDF1.Visible := false;

  // Open all documents from command line
  for i := 1 to ParamCount do
  begin
    s := ParamStr(i);
    if (CompareText(ExtractFileExt(s), '.pdf') = 0) and (FileExists(s)) then
    begin
       NewPDFDocument(s);
    end;
  end;
end;
```

**When we start the application it should look like this:**




### 3.1.8 Localization

Due to the scripting the localization can be done easily.

Simply load a translated action XML file before the menus and actions are created.

To retrieve the original XML file use this code:

xmlstring := WPViewPDF.CommandGetStr(**COMPDF_ACTION_READ**,'xml',0);

to assign new XML data use this:

WPViewPDF.CommandStr(**COMPDF_ACTION_WRITE**,xmlstring);

It is also possible to update the caption of existing menus and actions. This code will update the main menu and the action list:

```
 // The code only processes Actions and Menus with a certain prefix in the
name
 procedure TForm1.DoUpdateLanguage(Sender : TObject);
 var i : Integer;
     ac : TAction;
     men : TMenuItem;
 begin
   for I := 0 to ActionList1.ActionCount-1 do
   begin
      ac := TAction( ActionList1.Actions[i] );
```

```
      if (Copy(ac.Name, 1, Length(_ActionNamePrefix))=_ActionNamePrefix)
            and (ac.Tag<>0) then
      begin
          ac.Caption := WPViewPDF1.CommandGetStr(COMPDF_ACTION_READ,
               'caption', ac.Tag);
          ac.Hint := WPViewPDF1.CommandGetStr(COMPDF_ACTION_READ,
               'hint', ac.Tag);
          ac.Update;
      end;
   end;
   // Read kcaption - thats the caption of a action group.
   // Only menus which a name which starts with _MenuNamePrefix are updated!
   for I := 0 to MainMenu1.Items.Count-1 do
   begin
      men := TMenuItem( MainMenu1.Items[i] );
      if (Copy(men.Name, 1, Length(_MenuNamePrefix))=_MenuNamePrefix)
               and (men.Tag>0) then
      begin
          men.Caption := WPViewPDF1.CommandGetStr(COMPDF_ACTION_READ,
               'kcaption', men.Tag-1);
      end;
   end;
end;
```

### 3.1.9 Modify Annotation properties

With WPViewPDF 4 PLUS it is also possible to modify a selection of properties of the currently selected annotations.

There is not a GUI for this, but you can implement one which works with XML data.

xmlstring := WPViewPDF.CommandGetStr(**COMPDF_Ann_XMLGetFromAnnots**, '###', 0);

will retrieve a string with the properties of all selected annotations. The properties which have different values in the selection will be set to '###'.

All property values can be modified in the XML code, also the '###' placeholders and then applied to the selection:

WPViewPDF.CommandStrEx(**COMPDF_Ann_XMLSetToAnnots**, modified_xmlstring, 0);

Example of the XML code of a highlight annotation:

```
<?xml version="1.0" encoding="utf-8"?>
<neutral value="###"></neutral>
<annot X="58.80" Y="120.27" W="492.43" H="48.11" GrOptions="64"
Background-Color="blue" prp.n.Subtype="Highlight"
Color="Yellow" Alpha="50"></annot>
```

You can modify X, Y, W and H - all are measured in 72dpi and also Color and Alpha.

Background color is: Background-Color="color_as_string"
Line color is: Color="color_as_string"
Font name is: Font="fontname"
Font Size color is: Font-Size="floatvalue"
Font color is: Font-Color="color_as_string"

COMPDF_Ann_XMLSetToAnnots will modify all named properties, except for those with the value '###'. Note, that you can use a different token instead of '###'.

It is also possible to change the acroform field properties. The acroform fields are connected to widget annotations only. Of widget annotations are selected **COMPDF_Ann_XMLGetFromAnnots** can be used to retrieve the XML data and **COMPDF_Ann_XMLSetToAnnots** can be used to apply it.

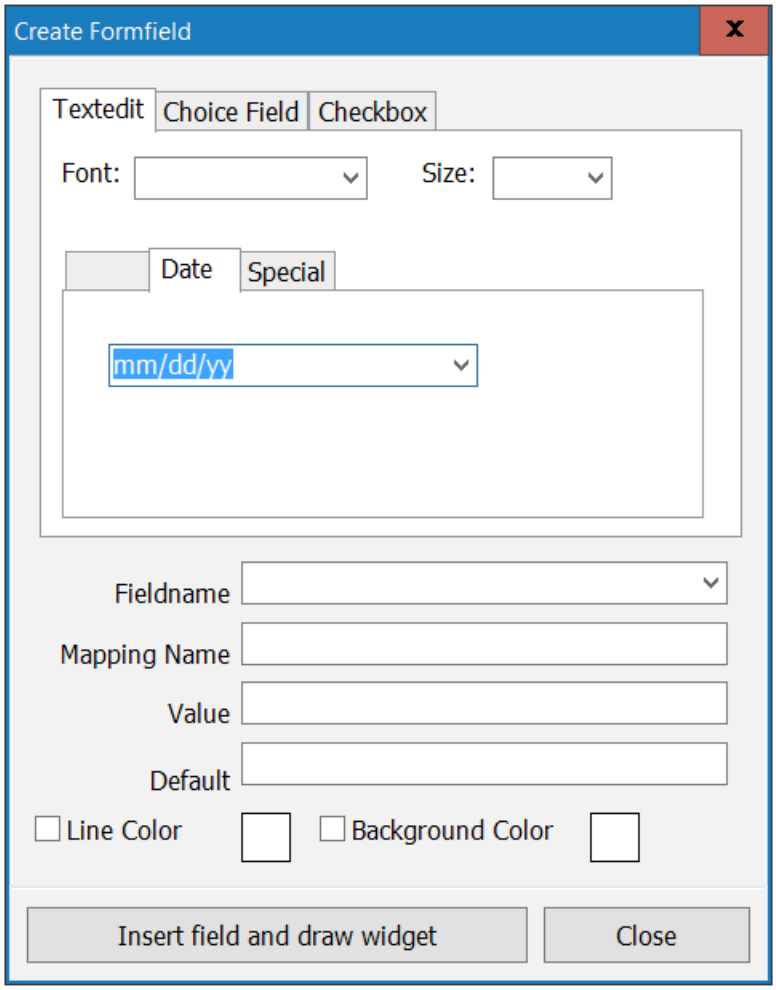
Example of the Acroform Field XML code:

```
<?xml version="1.0" encoding="utf-8"?>
<neutral value="###"></neutral>
<field T="undefined" TU="undefined" F="4" V="" FT="Tx"></field>
```

T is the fieldname, TU the alternative field name to be displayed in gui.
F are the field flags (see PDF specs), V the current field value and FT the field type. The type can be Tx or Btn for a checkbox.

### 3.1.10 Add dialog to create form fields

Our PDF edit demo program features a dialog to create a form.
It can be used to customize most properties of acroform widgets.




You find this dialog in directory PDFEdit\WPViewFieldForm.pas

To use this dialog add the from to your project so it is automatically created call it like this:

```
begin
  // assign the current Viewer - we actually assign the function so this stays
current!
  WPDFFields.WPViewPDF1 := pdf;
  WPDFFields.Show; // Not modal!
end;
```

Please note that "WPViewPDF1" has been defined a function:
```
   WPViewPDF1 : function : TWPViewPDF of Object;
```

So, although it looks like an object reference, it actually is a function which always provides the current Editor.

This makes it possible to work with different instances of WPViewPDF, as our

implementation examples does.

The dialog is opened in a non-modal way, it remains open.

**To a field just to click on "Insert field ad draw widget" and then draw a frame in the current viewer.**

## This code is used to create the field:

```
procedure TWPDFFields.btnInsertClick(Sender: TObject);
var FieldTag : Integer;
    fieldtype, FieldValues : String;
    FieldOptions : TStringList;
    FieldActions : TStringList;
begin
    FieldOptions := TStringList.Create;
    FieldActions := TStringList.Create;
    try
        if TypeSel.ActivePageIndex=0 then  // TextEdit
        begin
          fieldtype := 'Edit';
          FieldValues := '';

          if cbFontName.Text<>'' then
             FieldOptions.Add('font=' + cbFontName.Text);
          if cbFontSize.Text<>'' then
             FieldOptions.Add('font-size=' + cbFontSize.Text);


          case selFormat.ActivePageIndex of
             // Standard TextEdit or Memo
             0 : if chMultiline.Checked then fieldtype := 'Memo';
             // Edit with Date Format
             1 : if cbDateMask.Text<>'' then
             begin
               FieldActions.Add('Action-F=AFDate_FormatEx("' + cbDateMask.Text + '")'
               FieldActions.Add('Action-K=AFDate_KeystrokeEx("' + cbDateMask.Text + '
             end;
             // Edit with Special Format
             2 : if cbMask.Text<>'' then
             begin
               // AFSpecial_FormatEx requires string parameter
               FieldActions.Add('Action-F=AFSpecial_FormatEx("' + cbMask.Text + '")' )
               FieldActions.Add('Action-K=AFSpecial_KeystrokeEx("' + cbMask.Text + '"
             end;
          end;

          if chMultiline.Checked then
                fieldtype := 'Memo'
          else  fieldtype := 'Edit';

        end
```

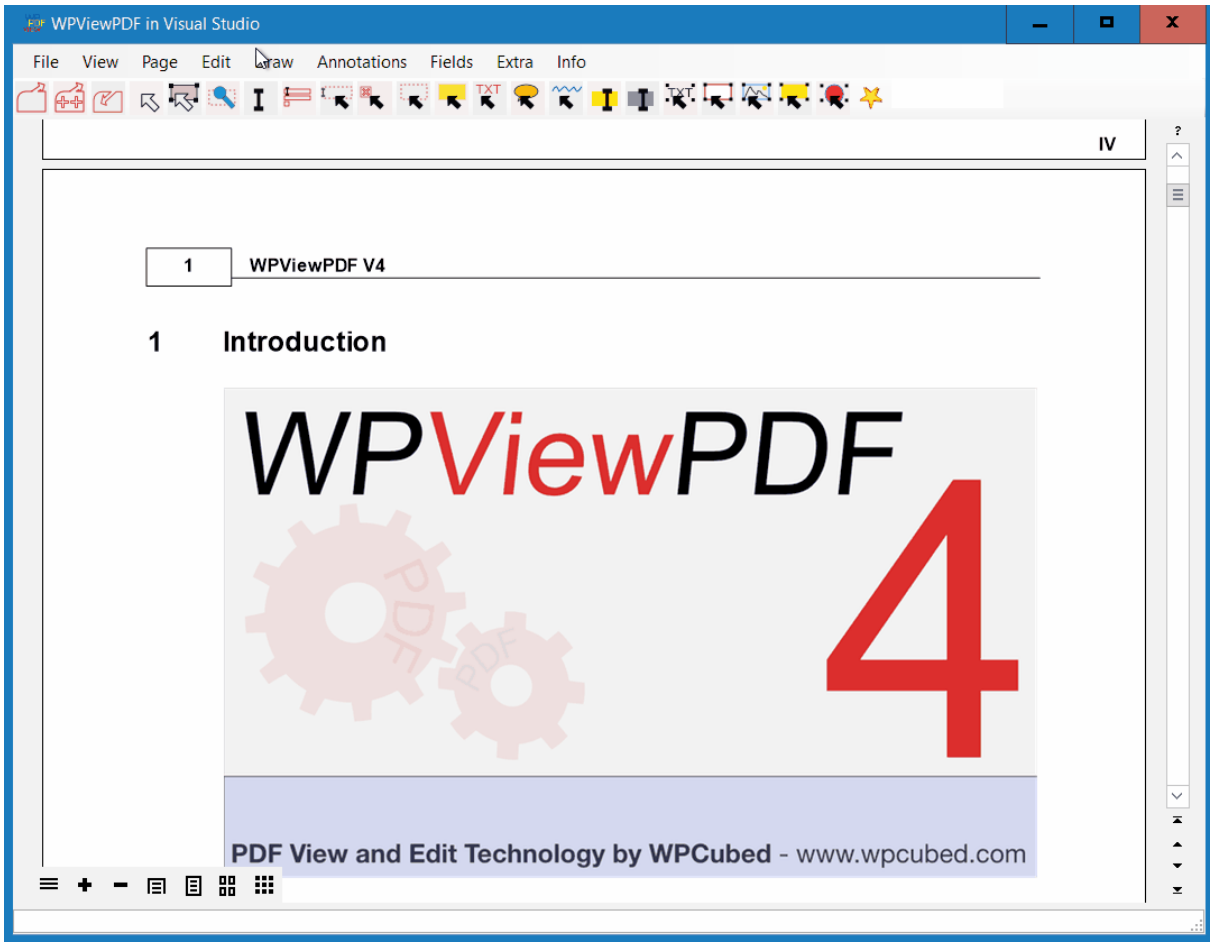
```
else
if TypeSel.ActivePageIndex=1 then  // Choice
begin
  if rbComboBox.Checked then
        fieldtype := 'Combobox'
  else  fieldtype := 'Listbox';

  if chMultiselect.Checked then fieldtype := fieldtype + ',multi';
  if chEditable.Checked then fieldtype := fieldtype + ',edit';

  FieldValues := edChoiceItems.Lines.CommaText;
end
else
if TypeSel.ActivePageIndex=2 then  // Checkbox
begin
  fieldtype := 'checkbox';
  if rgBtnGlyph.ItemIndex>0 then
       fieldtype := fieldtype + ',' + IntToStr(rgBtnGlyph.ItemIndex);
  if edBtnValueTrue.Text='' then
      FieldValues := 'Yes,Off'
  else FieldValues :=  edBtnValueTrue.Text +',Off';
end
else exit;

// Set the annotation widget color
// Here we pass a color string to be set in the Annotation in the
// MK dictionary as BC element.
if chLineColor.Checked then
begin
   (*
   FieldOptions.Add('prp.d.MK.c.BC=' + ColorToString(BCColor.Brush.Color) );
   FieldOptions.Add('prp.c.B=' + ColorToString(BCColor.Brush.Color) );
   *)
   FieldOptions.Add('Color=' + ColorToString(BCColor.Brush.Color) );
end;
if chBackgroundColor.Checked then
begin
   // FieldOptions.Add('prp.d.MK.c.BG=' + ColorToString(BGColor.Brush.Color)
   FieldOptions.Add('Background-Color=' + ColorToString(BGColor.Brush.Color)
end;

// We add or update the field
FieldTag :=
     WPViewPDF1.AddAcrofield(
         chFieldName.Text, // the fieldname including the path
         '', // Path is empty
         edMappingName.Text,
         edFieldValue.Text,
         fieldtype,
         4, // F=4 means visible and pritable
         FieldValues,
         FieldActions.CommaText,  // Actions - only used with wpAddFieldAndDra
         FieldOptions.CommaText,   // Options  - only used with wpAddFieldAndD
         wpAddFieldAndDrawWidget
```

```
            );

        finally
        FieldOptions.Free;
        FieldActions.Free;
        end;

end;
```

## 3.2 .NET (C#) Example

In this chapter we will show you how to quickly create a PDF viewer or PDF editor in VisualStudio using WPViewPDF V4 and its new "Actions" feature.

The PDF viewer uses a pretty large menu and a toolbar. Both elements are initialized by scripted code which can be easily used in your programs, too.




### 3.2.1 Initialize Program

Make sure windows does not display the form blurred on high dpi - we set the DPI-Aware flag in **Program.cs.**

We also specify the engine DLL which should be used by the application using WPViewPDF.PDFViewer.**SetDLLName**.

To check wether the program runs with 64bit ot 32 bit we check the size of an IntPtr.

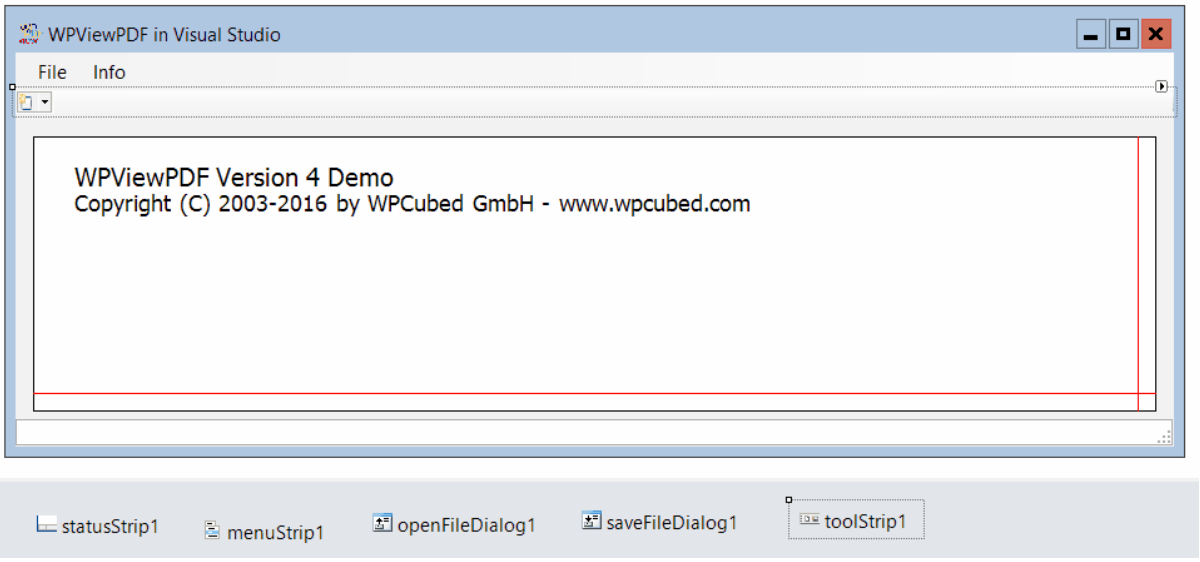
```
static class Program
{
    [System.Runtime.InteropServices.DllImport("user32.dll")]
    private static extern bool SetProcessDPIAware();

    [STAThread]
    static void Main()
    {
        // This code makes sure the form is not blurred on high DPI
        if (Environment.OSVersion.Version.Major >= 6) SetProcessDPIAware();

        // Set the name of the WPViewPDF Engine DLL which should be loaded
        WPViewPDF.PDFViewer.SetDLLName(
              // This is the basis directoy of the application
              AppDomain.CurrentDomain.BaseDirectory +
              // This is the engine DLL - please modify!
              // wPDFViewDemo04, wPDFView04 or wPDFViewPlus04
              "wPDFViewDemo04" +
              // If we run under 64bit system add "x64"
              ((System.Runtime.InteropServices.Marshal.SizeOf(
                          new IntPtr()) == 8)? "x64.dll" : ".dll")
           );

        Application.EnableVisualStyles();
        Application.SetCompatibleTextRenderingDefault(false);
        Application.Run(new Form1());
    }
}
```

Please note: Once a viewer was created on a form, use **ViewerStart** to set the licensing information.

```
pdfViewer1.ViewerStart("xxx", "yyy", 0);
```

### 3.2.2 Add the basic controls

The PDF viewer and editor needs at least a tooltrip, menustrip and of course the pdfviewer:




### 3.2.3 Initialize the viewer

This code is used to initialize the PDF viewer and the main form.

This is what is done here:

1) The usual "InitializeComponent()"
2) Call ViewerStart to set the WPViewPDF license key (please modify unless you use the demo)
3) Set the ViewOptions and ViewControls
4) Call command "commands.COMPDF_ACRO_MAKEDRAWOBJ" to make sure loaded PDF files are displayed with editable fields
5) Call command "commands.COMPDF_Ann_SetAnnotSaveMode" to make sure annotations added to a PDF are saved
6) Call the utility function InitMenu with our menu strip and the first and last predefined menu item.
"doExecuteWPViewAction" will be implemented in next chapter.

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

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

    pdfViewer1.ViewControls =
   eViewControls.wpHorzScrollBar |
   eViewControls.wpNavigationPanel |
   eViewControls.wpPropertyPanel |
```

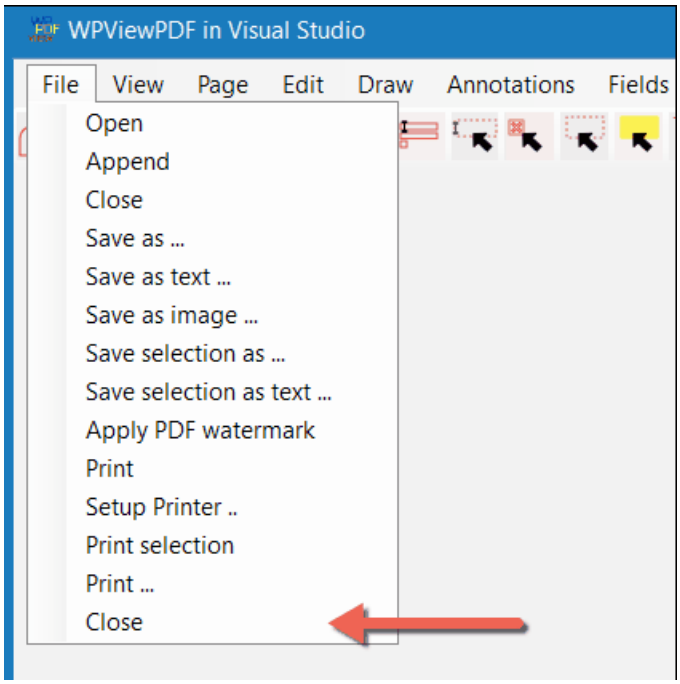
```
        eViewControls.wpVertScrollBar |
        eViewControls.wpViewPanel;

         pdfViewer1.AllowMovePages = true;


          // Make sure the annotations work interactively!
         pdfViewer1.Command(commands.COMPDF_ACRO_MAKEDRAWOBJ, "", 8192);

          // enlarge the zoom buttons
          pdfViewer1.Command(commands.COMPDF_SETBUTTONHEIGHT, 32 );

          // Standard Action Mode 'Click + Pan'
          pdfViewer1.Command(commands.COMPDF_SetActionMode,"",1);

          // ENABLE saving of annotations
          pdfViewer1.Command(commands.COMPDF_Ann_SetAnnotSaveMode, 1);

          // Load the menu from the embedded actions
          pdfViewer1.InitMainMenu(menuStrip1,
               doExecuteWPViewAction,
               fileToolStripMenuItem,
               infoToolStripMenuItem);

          // Initialize the toolbar
          InitToolbar(toolStrip1, _ActionButtons );
    }
```

### 3.2.4 Initialize the menu

The menu is initialized by just this line of code:

```
 pdfViewer1.InitMainMenu(menuStrip1, doExecuteWPViewAction,
fileToolStripMenuItem, infoToolStripMenuItem);
```

**menuStrip1**: The is is our main menu
**doExecuteWPViewAction**: This is the function which handles the click events
**fileToolStripMenuItem**: This is the caption item of the first menu strip. WPViewPDF will insert new file menu items at the start
**infoToolStripMenuItem**: This is the caption item of the last menu strip. WPViewPDF will append new info menu items at the end




*Almost the complete menu was auto generated, only the item "Close" existed before.*

**The function InitMainMenu has been implemented this inside the pdfviewer class**. It first request the count of action "kinds". Each action kind should create one menu caption with a drop down menu. Inside the dropdown menu there are the action "operations". Certain flags are used to hide a submenu or to make a submenu caption item for a deeper drop down menu.

```
public void InitMainMenu(System.Windows.Forms.MenuStrip menuStrip,
   System.EventHandler OnClick ,
   System.Windows.Forms.ToolStripMenuItem FileMenu ,
   System.Windows.Forms.ToolStripMenuItem InfoMenu
   )
        {
             int kmax = Command( commands.COMPDF_ACTION_READ, "kinds", 0);
             bool isPlus = (Command(commands.COMPDF_GetWPViewPDFPLUSFlag) != 0);


             for(int k = 0; k<kmax; k++)
             {
              System.Windows.Forms.ToolStripMenuItem men;
              System.Windows.Forms.ToolStripMenuItem asubmenu = null;
              int mencount = 0;
              bool reverse = true;
              //
             ------------------------------------------------------------------------------
              if ((k == 0) && (FileMenu != null))
                  men = FileMenu;
              else if ((k == kmax - 1) && (InfoMenu != null))
              {
```

```
                men = InfoMenu;
                reverse = false;
            }
            else
            {
                men = new System.Windows.Forms.ToolStripMenuItem();
                men.Text = CommandGetStr(commands.COMPDF_ACTION_READ, "kcaption", k);
            }
            //
        ------------------------------------------------------------------------------
            int omax = Command(commands.COMPDF_ACTION_READ, "operation", k);


            for(int o = omax-1; o>=0; o--)
            {
               int ac = (k << 16) + o;
               string s = CommandGetStr(commands.COMPDF_ACTION_READ, "caption", ac);
               int flags =  Command(commands.COMPDF_ACTION_READFLAGS, ac );
                 /*  flags:
              wpNeedSeperator,  // 1: add an seperator after the menu
              wpNoMenuItem,     // 2: Dont create menu items
              wpIsPlusAction,   // 4: Requires WPViewPDF PLUS
              wpRequireWriting,  // 8: Requires the PDF to be not protected
              wpGlobalOperation, // 16: This action doe not require PDF to be loaded
              wpMakeSubmenu      // 32: The following items until  wpNeedSeperator should
        be sub menus of this
                */
               if ((s!="") && ((flags & 32)!=0))
               {
                  asubmenu = new System.Windows.Forms.ToolStripMenuItem();
                  asubmenu.Text = s;
                  men.DropDownItems.Add(asubmenu);
               }
               else
               if ((s!="") && (isPlus || ((flags & 4)==0))  && ((flags & 2)==0))
               {
                     System.Windows.Forms.ToolStripMenuItem submen;
                     submen = new System.Windows.Forms.ToolStripMenuItem();
                     submen.Text = s;
                     submen.ToolTipText =  CommandGetStr(commands.COMPDF_ACTION_READ,
                    "hint", ac) ;
                     submen.Tag = ac;
                     submen.Click += new System.EventHandler(OnClick);

                     if (asubmenu != null)
                         asubmenu.DropDownItems.Add(submen);
                     else
                     {
                         if(reverse)men.DropDownItems.Insert(0, submen);
                         else men.DropDownItems.Add(submen);

                     }

            if ((flags & 1)!=0)
              asubmenu = null;
            mencount++;
               }
```

```
        }
        if (mencount>0)
        {
            menuStrip.Items.Add(men);
        }
        else men = null;
    }
}
```

InitMainMenu will create new menu items with the propery "Tag" set to an integer value. The integer can be used to execute the action in the viewer. (It consists of the action kind in the high word and the action "Woperation" in the low word). The value of 0 is not used, 1 refers to "File open", 2 will be used for "File Append" - any other value should not be expected to be fixed.

### 3.2.5 OnClick event handler

Since just one integer is enough to identify an action, the event handler for the click event can be implemented very versatile. It is even possible to get information which parameters are required for an action from WPViewPDF - so you can copy&paste the following code to your projects and use it there with a minimum of changes.

```
  private void doExecuteWPViewAction(object sender, EventArgs e)
{
    int param, paramkind, res;
    string actionname, actionparam;

    int ac = 0; // This is the action identifyer

    if (sender is System.Windows.Forms.ToolStripMenuItem)
         ac = (int)(sender as System.Windows.Forms.ToolStripMenuItem).Tag;
            else if (sender is System.Windows.Forms.ToolStripButton)
         ac = (int)(sender as System.Windows.Forms.ToolStripButton).Tag;

    param     = pdfViewer1.Command(commands.COMPDF_ACTION_READ, "param", ac );
    paramkind = pdfViewer1.Command(commands.COMPDF_ACTION_READ, "paramkind", ac );
    actionname= pdfViewer1.CommandGetStr(commands.COMPDF_ACTION_READ, "name", ac );
    actionparam="";

    if (paramkind==50)
    {
           string s = pdfViewer1.CommandGetStr(commands.COMPDF_ACTION_READ, "hint", ac )+"?"
       ;
           if(MessageBox.Show(s,"",MessageBoxButtons.OKCancel)==DialogResult.Cancel) return;
    }

    // Bit 2 is set, we need a string parameter!
    if ((param & 2) == 2)
    {
         /*     // 0: Pagenr as Int or string
         // 1: Fontname as string
```

```
  // 2: Color as Int or string
  // 3: PDF filename as string OPEN
  // 4: PDF filename as string SAVE
  // 5: text filename as string OPEN
  // 6: text filename as string SAVE
  // 7: image file name as string  OPEN
  // 8: JPEG file name as string   SAVE
  // 9: type @ options_comma_list
  // 10: options_comma_list
  // 11: options_for_DrawObjects
  // 12: Zoom Value as Int
  // 13: JPEG image file name as string to OPEN passed as "file=...",... + other
params
  // 14: some text as string  passed as "contents=...",... + other params
  // 15: some multiline text as string  passed as "contents=...",... + other params
  // 16: Boolean  on/off 1/0

  // 50: Ask $hint$ yes/now
  // 51: Ask $hint$ yes/no/cancel
   */

   if ( (paramkind == 3) || (paramkind == 5) || (paramkind == 6) || (paramkind ==
 13) )
       {
  if (paramkind==3)   openFileDialog1.Filter = "PDF Files (*.PDF)|*.PDF";
  else if (paramkind==5)  openFileDialog1.Filter = "Text Files (*.TXT)|*.TXT,*.*";
  else if ((paramkind==3) || (paramkind==13))
                                          openFileDialog1.Filter = "Image Files (*.
 JPG)|*.JPG;*.JPEG";

  if (openFileDialog1.ShowDialog()==DialogResult.Cancel) return;
  else actionparam = openFileDialog1.FileName;

  // This parameter is used for JPEG Draw Objects
  if (paramkind==13)
     actionparam = "\"file=" + actionparam + "\""; // + Color params   color=
 background-color

       }
       else if ( (paramkind == 4) || (paramkind == 6) || (paramkind == 8)  )
       {
             if (paramkind == 4) saveFileDialog1.Filter = "PDF Files (*.PDF)|*.PDF";
             else if (paramkind == 6)  saveFileDialog1.Filter = "Text Files (*.TXT)|*.
            TXT,*.*";
             else if ((paramkind == 8) || (paramkind==13)) saveFileDialog1.Filter =
            "Image Files (*.JPG)|*.JPG;*.JPEG";
             if (saveFileDialog1.ShowDialog() == DialogResult.Cancel) return;
             else actionparam = saveFileDialog1.FileName;
       }
       else if ((paramkind == 14)||(paramkind == 0))   // A string
        {
              InputForm dlg = new InputForm();
              dlg.label1.Text = pdfViewer1.CommandGetStr(commands.
           COMPDF_ACTION_READ, "hint", ac );
              if (dlg.ShowDialog(this)==DialogResult.Cancel) return;

              actionparam = (paramkind == 0) ? dlg.textBox1.Text :
```

```
                    "\"contents=" + dlg.textBox1.Text + "\"";
                dlg.Dispose();
            }
            else if (paramkind == 15)   // A multiline string
            {
                InputForm dlg = new InputForm();
                dlg.label1.Text = pdfViewer1.CommandGetStr(commands.COMPDF_ACTION_READ,
               "hint", ac);
                dlg.textBox1.Multiline = true;
                dlg.Height = dlg.Height * 2;
                if (dlg.ShowDialog(this) == DialogResult.Cancel) return;
                actionparam = "\"contents=" +  dlg.textBox1.Text + "\"";
                dlg.Dispose();
             }
    }

    pdfViewer1.CommandStrEx(commands.COMPDF_ACTIONNR, actionparam, ac);
}
```

At the start we check if the event was used by a menu item or toolbar item. Then we get the "ac", the integer value which is identifying an action. We request information about the action from WPViewPDF to check if any parameters are required, and if they are, open dialogs to request the information from the user. We use the open and save dialog here and also a simple InputForm which has been implemented in the .NET example.

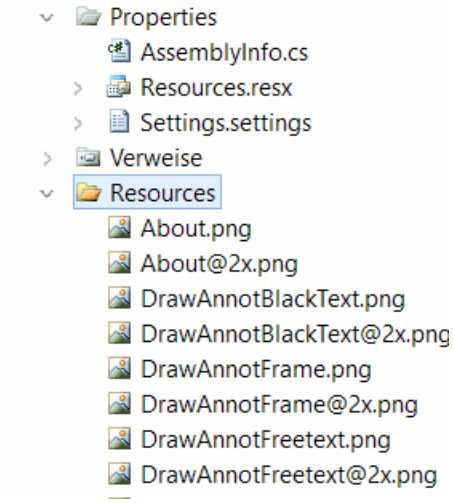
Using "commands.COMPDF_ACTIONNR" the action is processed.

### 3.2.6 Initialize the toolbar

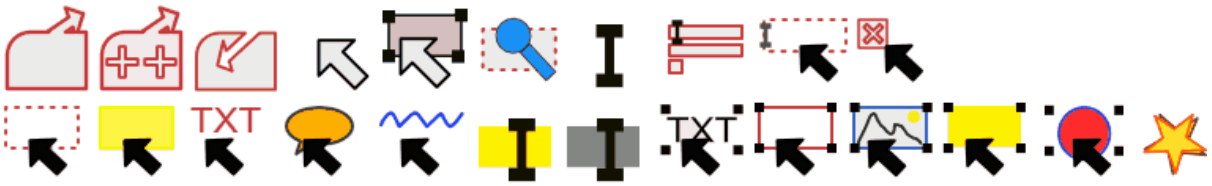
To initialize the toolbar we use an array of strings which contains the name of each action we want to list in the toolbar.

```
private string[] _ActionButtons = new string[23] {
"FileOpen", "FileAppend", "FileSaveAsPDF", "SelectStd", "SelectObjects",
"ZoomToRect", "SelectText", "SelectFillForm", "DrawFieldEdit", "DrawFieldCheck",
"DrawAnnotFrame",  "DrawAnnotHighlight", "DrawAnnotFreetext", "DrawAnnotSymbol",
"DrawAnnotSquiggly",
"DrawAnnotHighlightText", "DrawAnnotBlackText", "DrawTextline", "DrawRect", "DrawImage",
"DrawHighlight", "DrawCircle", "About" };
```

We also add PNG images for the buttons to the resources. All images use the build mode "Embedded Resource":




In this case the name of the image is the same as the action in the array _ActionButtons. The images are stored in a smaller and in a larger "2x" resolution. This are the symbols in the order of the actions in the array:




```
private void InitToolbar( ToolStrip toolStrip, string[] Actions )
{
    toolStrip.Height = 40;
    bool highdpi;

    // Enable the large buttons if >120dpi!
    Graphics g = Graphics.FromHwnd(new IntPtr(0));
    highdpi = (g.DpiX>120);

    // Create the toolbar
    for (int i = 0; i < Actions.Length; i++)
    {
 string pngname =
"PDFViewNET.Resources." + Actions[i] + ((highdpi) ?  "@2x.png" : ".png");

 System.Reflection.Assembly thisExe;

 // use this to check resource names in debugger!
 // string[] db = GetType().Assembly.GetManifestResourceNames();

 thisExe = System.Reflection.Assembly.GetExecutingAssembly();
 System.IO.Stream imagestream =  thisExe.GetManifestResourceStream(pngname);
 // If you get an exception here
```

```
 // a) check name of resource
 // b) check if resource Buildmoude was set to "Embedded"

 Image img = Image.FromStream(imagestream);

 // Create a new button
 ToolStripButton ActionBtn = new ToolStripButton("", img, null, "");
 ActionBtn.ImageScaling = ToolStripItemImageScaling.None;


 // and get the correct id
 ActionBtn.Tag = pdfViewer1.CommandStr(
   commands.COMPDF_ACTION_READ, "?" + Actions[i] );
 ActionBtn.Click += new System.EventHandler(doExecuteWPViewAction);

 toolStrip.Items.Add(ActionBtn);
    }
}
```

This method is called at the end of the initialization

```
public Form1()
{
     ...

     // Load the menu from the embedded actions
     pdfViewer1.InitMainMenu(menuStrip1, ...

     // Initialize the toolbar
     InitToolbar(toolStrip1, _ActionButtons );

}
```

All buttons call the event handler **doExecuteWPViewAction** which is also used by the menu items.

### 3.2.7 Update GUI

This method can be used to update the checked and enabled state of the buttons.

```
private void UpdateGUI()
{
    for (int i = 0; i < toolStrip1.Items.Count; i++)
    if ( toolStrip1.Items[i] is ToolStripButton )
    {
         ToolStripButton btn = toolStrip1.Items[i] as ToolStripButton;
         int ac = (int)btn.Tag;
         if (ac > 0 )
         {
             int state = pdfViewer1.Command(commands.COMPDF_ACTION_READSTATE, ac);
        // 1=Checked, 2=Disabled
        btn.Enabled = (state & 2)==0 ;
        btn.CheckState = ((state & 1) == 1) ? CheckState.Checked : CheckState.Unchecked;
         }
    }
}
```

WPViewPDF triggers an event `OnViewerMessage` which can be used to call the method UpdateGUI.

```
        private void pdfViewer1_OnViewerMessage(object Sender, ref int ID, int param)
{
    switch (ID)
    {
  case commands.MSGPDF_NEEDPASSWORD:
    {
   break;
    }

  case commands.MSGPDF_CHANGESELPAGE: // Moved to different page (=wparam)
    {
   break;
    }

  case commands.MSGPDF_CHANGEVIEWPAGE: // Moved to different page (=wparam)
    // MSGPDF_CHANGEVIEWPAGE is also triggered if the action mode was changed.
                      // This makes MSGPDF_CHANGEVIEWPAGE
    // to update GUI elements, such a buttons
    {
   UpdateGUI();
   break;
    }

  case commands.MSGPDF_CHANGEANNOT: // WPViewPDF 4 only: The annot have been moved, created or
deleted.
    {
   break;
    }

  case commands.MSGPDF_CHANGESELOBJECT: // A Draw object has been selected or deselected
    {
   break;
    }
   }
}
```

This is the method which updates the state of the buttons

### 3.2.8 Extract Attachments

To extract attachments of a PDF we have added a menu item to the "Info" menu. This menu item will get a drop down with all attachment names in the current document.

This event is triggered when the "Info" menu drops down:

```
private void infoToolStripMenuItem_DropDownOpening(object sender, EventArgs e)
{
    menFileattachment.DropDownItems.Clear();
    int j = pdfViewer1.Command(commands.COMPDF_Attachment_List);
    for(int i = 0; i<j;i++)
    {
        System.Windows.Forms.ToolStripMenuItem men = new System.Windows.Forms.
      ToolStripMenuItem();
        men.Text = pdfViewer1.CommandGetStr( commands.COMPDF_Attachment_GetProp, "", i );
        men.Tag = i;
        men.Click += new System.EventHandler(OnClickAttachment);
        menFileattachment.DropDownItems.Add(men);
    }
    if(j<=0) menFileattachment.DropDownItems.Add("<empty>").Enabled = false;

}
```

This event handles the click on any of the attachment menu items to save the attachment to a file.

```
private void OnClickAttachment(object sender, EventArgs e)
{
    System.Windows.Forms.ToolStripMenuItem men = sender as System.Windows.Forms.
ToolStripMenuItem;
    int l = pdfViewer1.Command(commands.COMPDF_Attachment_GetData, (int)men.Tag);
    if (l > 0)
    {
          byte[] databytes = pdfViewer1.GetMemory();
          saveFileDialog1.FileName = men.Text;
          if ( saveFileDialog1.ShowDialog()==System.Windows.Forms.DialogResult.OK )
          {
              System.IO.FileStream file = new System.IO.FileStream(saveFileDialog1.FileName,
        System.IO.FileMode.Create );
              file.Write(databytes,0,databytes.Length);
              file.Close();
          }
    }
}
```

# 4 Tasks

## 4.1 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

## 4.2 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.

**New**: You can also select a single page view mode and also move the thumbnail window to a different parent window.

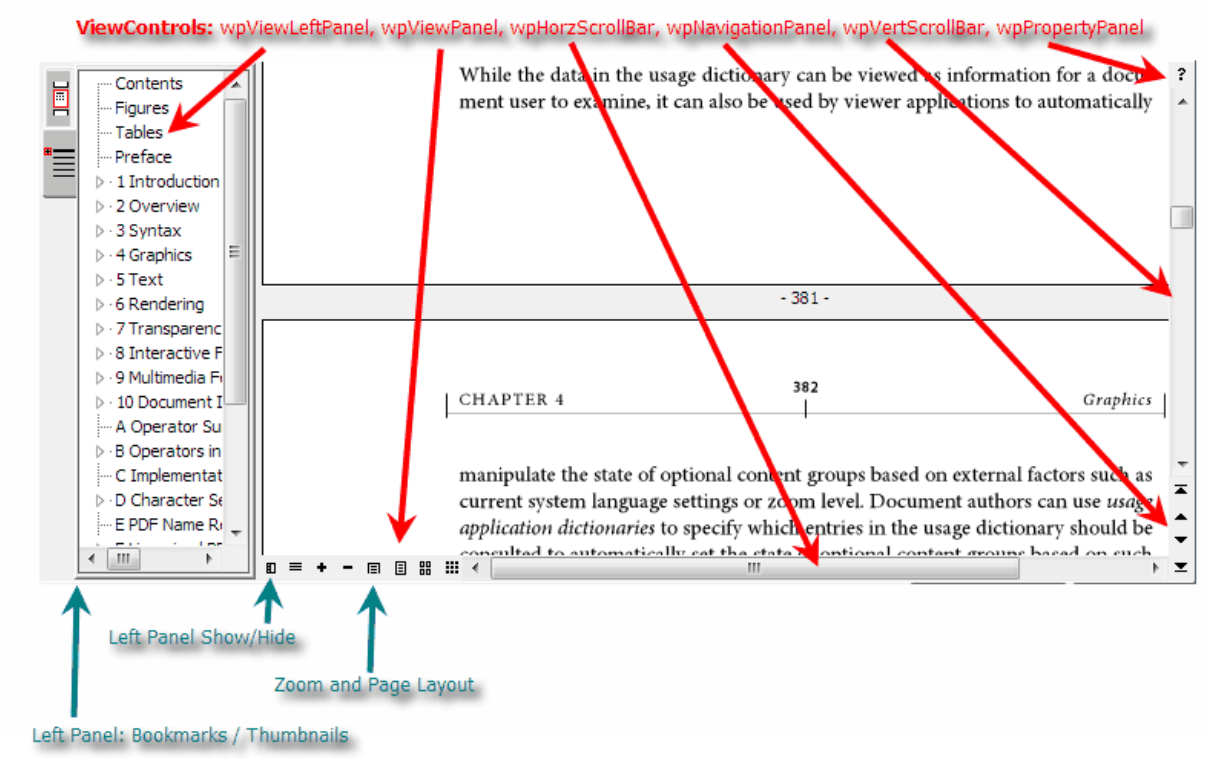
### 4.2.1 ViewControls and ViewOptions

**If you prefer a single page view, please use**

**Command( COMPDF_SinglepageMode, 1 )**

(the main windows shows only one page and does not scroll)

**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 - however the hyperlink event will still be triggered.
wpDontHighlightLinks: Hyperlinks will not painted with a blue background
wpNoHyperlinkCursor: Do not switch to hand point cursor on links.

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: (wrong name) Allow page selection with PageUp + Down + Shift / Ctrl
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: Enable display thumbnails in left panel
- use CommandEx(COMPDF_SetPageModeDefault, val) to actually display them
wpAllowPageDragging : Allows move of selected pages
wpHidePageSelection : Disable display of selection in main viewer
wpHidePageSelectionThumbnails: disable display of selection in thumbnail viewer
wpInteractiveThumbnails: Allows page moving in thumbnail viewer
wpThumbnailAtozoomToSquareWH : reserve the maximum square rectangle for thumbnails. This avoids scaling when pages are rotated.
wpHideFocusRectThumbnails: Hide the red line which highlights the current page


```
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.

**COMPDF_SetPageModeDefault = 615:**
0=Auto, 1=None, 2=Outlines, 3=Thumbnails
(Note: The VCL has the property PageModeDefault)

This command disables, that the user can switch off the navigation n(left) panel and
it stays switched off after loading a new file.
It further can override the PageMode defined in the PDF:

**COMPDF_EnableNavigationAfterLoad = 616:**
0: AsBeforeLoad - persistent, default
1: AsDefinedInDefaultPageMode - always use COMPDF_SetPageModeDefault
2: DefinedInPDFOrDefault - reset navigation after loading a new file

Delphi Example:
Display the thumbnails from the beginning (after loading a PDF file)

```
FViewer := TWPViewPDF.Create(Parent); // Parent can be a TPanel for example
FViewer.Parent := Parent;
FViewer.ViewControls := [ wpViewLeftPanel, wpViewPanel,  wpVertScrollBar, w
FViewer.ViewOptions  := [ wpViewThumbnails ];
FViewer.PageModeDefault := wpPageModeThumbnails;
FViewer.SetBounds(1,1,Parent.Width-2,Parent.Height-2);
```

### 4.2.2 Localization

**1) Localize the actions, hints and captions**

Use the command
str := WPViewPDF1.CommandGetStr(**COMPDF_ACTION_READ**, 'xml', 0 );

to read the action XML script. (You can use nthe PDFEdit.EXE, Menu "Info" to do this)

This will create XML code similar to

```
<?xml version="1.0" encoding="windows-1252"?>
<infomenu>
 <menu1 text="Document Summary"></menu1>
 <menu2 text="Security Information"></menu2>
 <menu3 text="Modification Rights"></menu3>
 <menu4 text="Font Information"></menu4>
 <menu5 text="Viewer Options"></menu5>
 <menu6 text="About Viewer"></menu6>
 <menu7 text="Edit Options"></menu7>
 <menu8 text="Printer Setup"></menu8>
 <menu9 text="Print..."></menu9>
</infomenu>
<kinds><kind0 text ....
```

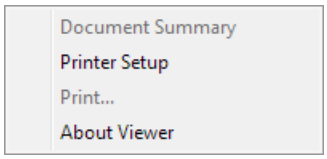
Save to a file and change the language in that XML data.

When your program starts load the changed data back **before** the GUI is created.

To load it back use the command

**COMPDF_ACTION_WRITE**

### 2) Localize the "?" 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);
```

### 4.2.3 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 int parameter.
the optional string parameter can be "y" or "x,y" or "x,y%z" to specify the zoom value z
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

### 4.2.4 Zooming

Control Zooming:

```
  COMPDF_Zoom100          = 41.............! 100 % Zoom
  COMPDF_ZoomIn           = 42.............!  + 10%
  COMPDF_Zoom             = 43.............! Zoom to StrPar/IntPar - if IntPar=0
retrieve zoom!
```

if StrParam = "MP" zooming will center to mouse position
if StrParam = "RECT" the viewer will zoom to the frame rectangle which was drawn last. This allows the implementation
of a zoom tool.

```
  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
```

Note: Maximum zooming value is 500

To read the current zooming use

```
  COMPDF_ZoomGetCurrent   (= 49).............! read current zoom
```

This command saves and restores the zooming

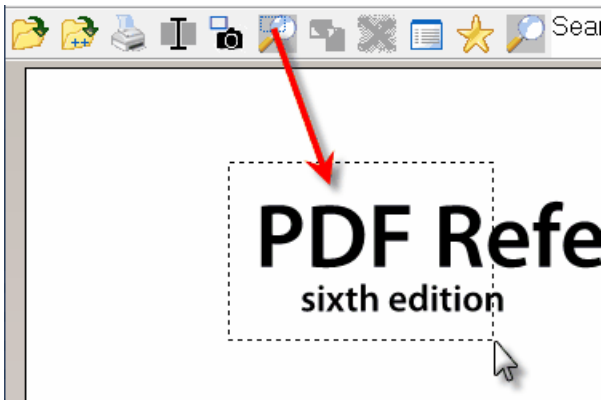
```
  COMPDF_ZoomSaveRestore    = 76;
```

IntPar=1 Saves, IntPar=0 Restores

This command controlls zooming in thumbnail window (left panel)

```
 COMPDF_ZoomThumbnails     = 77;
```

IntPar>=10 sets the thumbnail zoomsize (default 12), -9..9 increases or decreases the zoom value




**Example:**

Implementation of Zoom Tool:

```
var FSelectZoomRect, FSaveToClip : Boolean;

procedure TWPViewPDFDemo.ZoomToolClick(Sender: TObject);
begin
   FSelectZoomRect := true;
   WPViewPDF1.CommandEx(COMPDF_SelectMode, 2);
end;
```

Use the OnSelRectEvent event:

```
procedure TWPViewPDFDemo.DoSelRectEvent(Sender: TObject; const PageNr: Integer
  R: TRect);
begin
  if FSelectZoomRect then
  begin
     Screen.Cursor := crDefault;
     WPViewPDF1.CommandStr(COMPDF_ZOOM, 'RECT');
  end else
   // This code is used to capture a bitmap
  if FSaveToClip then
  begin
     if WPViewPDF1.CommandEx(COMPDF_SaveBMPToClipboard, 200)>0 then  // Save i
        ShowMessage('An image @200 dpi was copied to clipboard.');
  end else
   // This code is used to capture as text
  if FCopyTextRect then
  begin
     if WPViewPDF1.CommandEx(COMPDF_CopyToClibrd,8)>0 then
         ShowMessage('Text copied to clipboard.');
  end;
end;
```

## 4.3 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**

Hint: If the SaveToFile or CopyToClipboard function does not work for you, please check the setting of property SecurityOptions!

The flag wpDisableSave must not be set - if it is set once, saving cannot be enabled again!

Using the compiler switch **IGNORE_SECOPT_IN_DFM** it is possible with Delphi to disable that the property SecurityOptions is loaded from the DFM data. This makes it possible to set the property later in code.

It is also possible to save in RTF, TXT, XML and HTML format! (TWPViewPDF. GetPageText Method)

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 arrange 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" and "Save 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.

(Note: PDF which use 256 bits AES encryption may never be saved unless the owner password was specified)

If your Delphi APP cannot save a PDF file, please check if you use the flag **wpDisableSave** in the property **SecurityOptions**. Please note, that once wpDisableSave was used, it cannot be enabled again to prevent hacking the application. The Security options can also be set with commands.

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.

When you save to a PDF file you can use **COMPDF_SetSaveMode = 613** to remove certain PDF document elements the next time PDF is saved.

Parameter of COMPDF_SetSaveMode can be this bit values:
1: Remove the Annots except for Hyperlinks. Ommits "AcroForm"
2: Remove the Hyperlinks
4: Remove the Bookmarks
8: Remove the StructElements
16: Remove Transition Effects
32: remove Page AA Actions
64: remove PDFA flag
128: Delete Extra XML Data
256: Delete extra commands (such as images and DrawObjects)
512: Delete named destinations
1024: DO not create PDF A Marker
2048: DO always create PDF A Marker
4096: Do not save modified page sizes

8192: Never write Cropbox parameter

8192*2: Delete extra commands (such as images and DrawObjects)
8192*4: Delete named destinations
8192*8: DO not create PDF A Marker
8192*16: DO always create PDF A Marker
8192*32: Don't modify page Size
**8192*64**: **Flatten the PDF on save time.** This requires that proper Appearance streams are present for the Fields.
8192*128: Delete Metadata
8192*256: write "NeedAppearances=true"
8192*512: write "NeedAppearances=false"
8192*1024: Save all fonts found on a page, not just the ones which are used.

**When saving a PDF with WPViewPDF PLUS it is possible to reduce or enlarge the page size.**

This feature is controlled by 3 command ids:

**COMPDF_SaveScaledPDFMode = 617** - Activate the scaling mode while saving
0: wpNoScaling = OFF
1: wpScaleToWH = Use desired Width/H as exact value
2: wpShrinkToWH = Use desired Width/H as MAXIMUM value
3: wpScaleToWHIgnoreAspcect = Use desired Width/H as exact value and override aspect ratio
4: wpScalePerThousand = Use scaling factor DesiredWidth/1000

COMPDF_SaveScaledPDFSetX = 618 - This is either the desired page width in pt (=1/72 inch), or the horizontal page scaling in 1/1000
COMPDF_SaveScaledPDFSetY = 619 - This is either the desired page height in pt, or the horizontal page scaling in 1/1000

Please note that only the page content and the basis rectangle of annotations will be scaled.

**Example**:

```
pdf.command(COMPDF_SaveScaledPDFMode, 4); // wpScalePerThousand
try
   // Skaliere von DinA4 auf DinA5
   pdf.command(COMPDF_SaveScaledPDFSetX, round(1/sqr(2)*1000));
   pdf.command(COMPDF_SaveScaledPDFSetY, round(1/sqr(2)*1000));
   pdf.Plus.SaveToFile(SaveDialog1.FileName);
finally
   pdf.command(COMPDF_SaveScaledPDFMode, 0);
end;
```

When using Delphi you can use the method ActivateScaledPDFWriting to enable the scaling.

## 4.4 Draw Shapes / Text objects on PDF

WPViewPDF offers the unique feature of "Draw Objects".

This objects can be text 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.

New: With WPViewPDF V4 the objects can also be created to belong to the "document" instead of a certain PDF file. (Currently images cannot be document dependent). With WPViewPDF Plus is is possible to save the objects to XML and load in this format. (command COMPDF_DrawObjects_XML)

This makes it possible to load a different PDF file with the objects stay at the same place on a certain page number!

The draw objects are always rendered independently of the PDF page contents - this avoids flickering and slow downs while the user move the objects around, even on PDF pages which are rendered slowly.

New: It is also possible to trigger an mail-merge event for text objects - so they can be updated on each time they are painted!

WPViewPDF PLUS: Draw objects can be "rendered to the page". This applies postscript code which becomes part of the page contents. The draw objects are not automatically deleted and can be moved around and rendered a a different place -or- if they belong to the "document", be applied to a different PDF file!

The modified PDF file can also be 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;
```

This code creates a text field

```
procedure TCertificatePrint.InsertFieldClick(Sender: TObject);
var Param: TPDFDrawObjectRec;
begin
  FillChar(Param, SizeOf(Param), 0);
  Param.PageNo := WPViewPDF1.Page-1;
  Param.grtyp := 100;

  Param.w := 100;
  Param.h := 30;
  Param.ObjectOptions := OBJGR_KEEP_ASPECTRATIO + OBJGR_STRETCH+ OBJGR_CENTER + OBJGR_

  Param.CreateOptions := 8192 * 8;

  WPViewPDF1.AddDrawObject(wpAddNow, cbField.Text, Param, '***' );
end;
```

### 4.4.1 Record TPDFDrawObjectRec

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

It structure 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.
**FontSize** - the font size *100. Set to 0 when you need stretched text

**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. (add JPEG image with 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. New: This now also works for text objects. |
| 2 | : Stretch text to fill the rectangle. The FontSize should be 0 in this case. |
| 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.<br>This mode is only effective on screen, when rendering to PDF regular transparency will be used.<br>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 |
| 2048 | : Right align text horizontally in the box |
| 16384 | : New: Trigger merge event MSGPDF_DRAWOBJECT_GETTEXT for text objects. |

**CreateOptions** - how should the object be created. The following bits can be set

| | |
|---|---|
| 1 | : Place the object UNDER the Page |
| 2 | : Place at the Right Border of the page (ignore X) |
| 4 | : Place at the Bottom Border of the page (ignore Y) |
| 8 | : Scale the object to the page horizontally (uses X as right and left margin) |
| 16 | : Scale the object to the page vertically (uses Y as right and left margin) |
| 32 | : Create the object and select it (clear selection) |
| 64 | : Create the object and add it to the selction (do not clear selection) |

Offset Modes

| | |
|---|---|
| 128 | : Page_Center_Y = add y to page height / 2, subtract height / 2 |
| 256 | : Page_Bottom_Y = add y to page height, subtract height |
| 512 | : Page_Center_X = add x to page width /2, subtract width / 2 |
| 1024 | : Page_Right_X = add x to page width, subtract width |

Change meaning of W and H

| | |
|---|---|
| 2048 | : Width and Height are measured in % of Page Width and Height |

Various:
8192*2 : Do NOT refresh the screen
8192*4 : Rotate the object boundaries backwards according to Page rotation
This can be combined with Angle := PageRotation[PageNumber-1] to Make sure a drawn object appears inside the drawn frame.
8192*8 : New: Added to WPViewPDF Version 4: **Place the object in the global document layer**

**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 AddDrawObject. It takes a structure of type TPDFDrawObjectRec as parameter.

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

See examples here....

### 4.4.2 Delete and modify shapes

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

The API ClearDrawObject can also be used, it wraps 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;
```

### 4.4.3 Modify attributes of draw objects

It is also possible to change the attributes of the currently selected annotation and/ or draw objects using the command **COMPDF_Ann_ModifyAddProps**. It will expect a comma separated list with name-property pair. The names should be the same as the names used with the internal actions.

```
procedure TForm1.btnSelectColorClick(Sender: TObject);
begin
  if (pdf<>nil) and ColorDialog1.Execute(Handle) then
  begin
    pdf.command(COMPDF_Ann_ModifyAddProps,
      'Brush-Color=' + ColorToString(ColorDialog1.Color) , 4 + 8);
  end;
end;
```

For draw objects this names can be used

Font
Font-Size
Alpha
Line-Width
Line-Color
Brush-Color
Text

With "'HighlightType=..." it is also possible to set the annotation which will be created for selected text. The default is "Highlight", a possible alternative would be "Square".

pdf.CommandStrEx( **COMPDF_Ann_ModifyAddProps**,
**'HighlightType=Square'** , 1+8 );

Note: The name "Color" and "Background-Color" is used for annotations, not draw objects.

Please make sure the " signs are paired.

**The integer parameter is a bit field (if 0 nothing will be changed!)**

1 : modify the "current" attributes. This attributes are used by the currently active action or "Draw mode".
2 : modify the attributes of the currently selected annotations - mode 2.

4 : Auto Mode: If annotations are selected, they will be modified.
If nothing is selected, the "current" attributes are modified.
8 : If the current attributes are changed,
also change the defaults for the highlight, frame and freetext actions
16: Do not update the screen
32: Limit use of undo buffer (only stores first element)

You can also use command(COMPDF_Ann_Undo, 4) to disable the undo buffer. Dont' forget to call command(COMPDF_Ann_Undo, 5) to enable it again.

### 4.4.4 Render objects and annotations into the 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 : RenderAnnotations - Render the annotations which are not widgets and not Popups
2 : RenderWidgets - Render widgets annotations.
4 : RenderPopups - also render popup annotations
8 : DeleteRenderedObjects - deletes draw objects and annotations after rendering
16 : UseOriginalDataForRendering - use the original V and AS values when windgets are rendered
32 : UnderPageLayer - Render the draw objects under the page
64 : OverPageLayer - Render the draw objects over the page (or document draw objects)
128: Do not render or delete. Just count how many objects would be affected

Note: Links, file attachments, movie and sound annotations are never rendered!

The return value is the count of objects which were converted.

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

**The command can be used to "flatten" the annotations.**

To do so, it is required that the annotations have been converted into draw objects (which makes them editable) using COMPDF_ACRO_MAKEDRAWOBJ.

This commands will flatten all annotations and remove them:

```
Command(COMPDF_ACRO_MAKEDRAWOBJ,'',0);
Command( COMPDF_RenderDrawobjects, 1+2+8);
```

to render the objects. You can change the save mode to remove the annotations when saving the file
see **COMPDF_Ann_SetAnnotSaveMode**.

### 4.4.5 XML Support

To write the document level draw objects with WPViewPDF PLUS to XML use
COMPDF_DrawObjects_XML, 1

To create document level draw objects from XML with WPViewPDF PLUS use
COMPDF_DrawObjects_XML, xmlstring, 2

Example:

```
procedure TCertificatePrint.LoadFromXMLClick(Sender: TObject);
begin
   WPViewPDF1.CommandStrEx(COMPDF_DrawObjects_XML, Memo1.Text, 2);
end;

procedure TCertificatePrint.SaveToXMLClick(Sender: TObject);
begin
  Memo1.Text := WPViewPDF1.CommandGetStr(COMPDF_DrawObjects_XML, '', 1);
end;
```

### 4.4.6 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.

### 4.4.7 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.

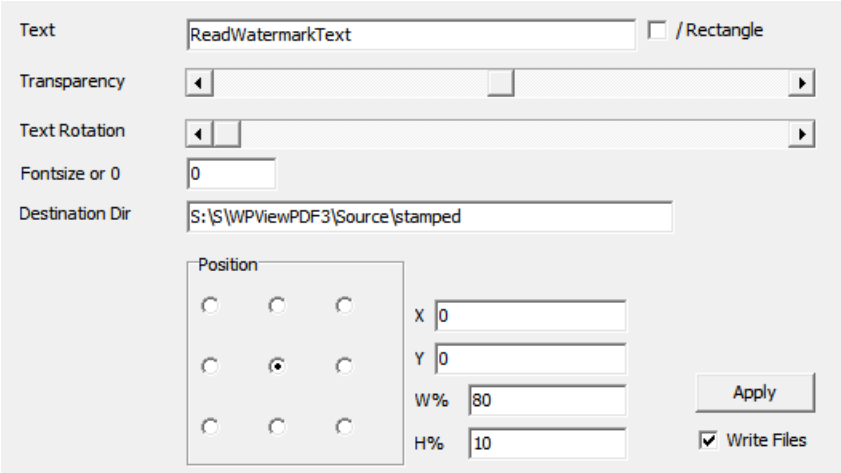
```
var
  t: TPDFDrawObjectRec;
  s : AnsiString;
begin
  FillChar(t, SizeOf(t), 0);
  t.grtyp := 100;
  t.typparam := 2000; // Textfield, Height = 20
  t.ColorText := ColorToRGB( clBlue ); // Text Color
  t.ColorPen := ColorToRGB( clYellow ); // Background Color
  t.ObjectOptions := 4+8; // Center Text + Opaque
  // use 2048 instead of 4 for right aligned text

  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', PAnsiChar(s));
  end;
end;
```

If you want the user to "draw" the object with the mouse use

AddDrawObject(**wpDrawAndAdd** ...

### 4.4.8 VCL: Add text draw object to all pages

**The code below is the central part of this Delphi Demo:**




```
var
   WPPDF: TWPViewPDF;
   T: TPDFDrawObjectRec;
   cnt, FTransparencyPercent, FRotationAngle : Integer;

....

// loop through all pages of the PDF
   for cnt := 0 to WPPDF.PageCount-1 do
   begin
       FillChar(T, SizeOf(T), 0);
       T.structsize := SizeOf(T); //!!
       T.PageNo := cnt;
       if DrawRect.Checked then
       begin
         T.grtyp := 0;
         T.ColorBrush := clGreen;
       end else
       begin
         T.grtyp := 100;
         T.ColorText := clRed;
       end;
       T.Alpha := Round(FTransparencyPercent / 100 * 255);
       T.Angle := FRotationAngle;
       T.ObjectOptions := 64;
       T.FontSize := StrToIntDef(FontSize.Text,0)*100;

       // The offset mode is under development:
       T.CreateOptions :=
          PDFDrawObjectRecPositionArray[PositionMode.ItemIndex]
          + 2048;  // W and H = %
```

```
        T.units_xywh := 10; // 720 dpi
        T.x :=  StrToIntDef( XOFF.Text, 0);
        T.y :=  StrToIntDef( YOFF.Text, 0);

        if T.FontSize=0 then
        begin
          T.w := StrToIntDef( WPZ.Text, 0); // % due to flag 2048 in CreateOpti
          T.h := StrToIntDef( HPZ.Text, 0);
        end;

        OptionStr := 'FONT=TimesNewRoman'; // alternative: CourierNew';

        WPPDF.AddDrawObject(wpAddNow, WideString('TEXTOBJECT'), T, WideString(P
          , PAnsiChar(OptionStr), Length(OptionStr)
          );
      end;
```

### 4.4.9 VCL: AddHighlightAnnotationForText

Using this function it is possible to not only find text but also to apply highlight draw objects to the found text.

```
procedure TWPViewPDF.AddHighlightAnnotationForText(
s : string;
Color : TColor;
Alpha : Integer = 255
);

var b : Boolean;
    page, x,y,w,h : Integer;
begin
    b := false;
    try
       command(COMPDF_BEGINUPDATE);
       while FindText(s, false, b, true)>=0 do
       begin
          b := true;
          page := command(COMPDF_FindGetXYWH, 10);
          if page>=0 then
          begin
              x := command(COMPDF_FindGetXYWH, 11);
              y := command(COMPDF_FindGetXYWH, 12);
              w := command(COMPDF_FindGetXYWH, 13);
              h := command(COMPDF_FindGetXYWH, 14);
              AddHighlightRect(page, x,y,w,h, Color, [wpAsAnnot,wpAnnotAtFound
          end
          else break;
       end;
    finally
       command(COMPDF_ENDUPDATE, 2);
    end;
end;
```

### 4.4.10 .NET C# Example: Add text, image or rectangle

Add 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, ""
);
        }
```

Add a text object:

```
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");
        }
```

This code adds a JPEG image:

```
private void imageToolStripMenuItem_Click(object sender, EventArgs e)
        {
            int gaphicid = pdfViewer1.CommandStr(commands.COMPDF_AddJPEG,
"C:\\debug\\test.jpg");

            TPDFDrawObjectRec r = new TPDFDrawObjectRec();

            r.typparam = gaphicid;
            r.grtyp = 20;
            r.PageNo = 0;
```

```
            r.x = 100;
            r.y = 100;
            r.w = 400;
            r.h = 400;

            pdfViewer1.AddDrawObject(AddDrawObjectMode.AddNow, "IMG1", r,
"");
        }
```

### 4.4.11 VB6 add rectangle 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, ""
```

### 4.4.12 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, zero based! (0..) |
| 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 of 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 command 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)

Pascal Example:

```
WPViewPDF1.Plus.UseImage(ImageID, 0, 0, 0, 0, 0, 0,
  [wpAtPageHorzCenter, wpAtPageVertCenter,
     wpPageWidthPC, wpPageHeightPC, wpUnderPage]);
```

Exampe:

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

It is possible to use this commands to later hide and display certain images using this commands:

COMPDF_ImageSetHidden = 323 : param = ID, hide image with ID param (all inserted positions).
this command may be called with a string parameter "on", "off" and "toggle"

COMPDF_ImageSetDisplayed = 324: param = ID, show image with ID param (all inserted positions)

### 4.4.13 AppendPage and add Shape

The command COMPDF_AppendPage can be used to append a page.

**Example:**

```
var
  Param: Cardinal;
  StrParam: String;
begin
  Param := (612 shl 16)+792; // size of page expressed as - hi(8.5*72) + 11*72
  StrParam := '1 0 0 rg 0 G 0 0 612 792 re f'; // RED PAGE
  WPViewPDF1.CommandStrEx(COMPDF_AppendPage, StrParam, Param);
  StrParam := '0 0 1 rg 0 G 0 0 612 792 re f'; // BLUE PAGE
  WPViewPDF1.CommandStrEx(COMPDF_AppendPage, StrParam, Param);
end;
```

It is also possible to append a page and draw a shape. Make sure to use some PS code to draw to the page at the start.

```
var
  Param: Cardinal;
  t: TPDFDrawObjectRec;
begin
  Param := (612 shl 16)+792;
  WPViewPDF1.CommandStrEx(COMPDF_AppendPage,
        '1 0 0 1 0 0 cm 1 1 1 rg 0 G 0 0 0 0 re f', Param);

  FillChar(t, SizeOf(t), 0);
  t.ColorBrush := clRed;
  t.Alpha := 0; // transparent
  t.grtyp := 1; // Rectangle
  t.PageNo := 1;
  t.x := 20; t.y := 20; t.w := 200; t.h := 50;
  t.structsize := SizeOf(t);

  WPViewPDF1.CommandStrEx(COMPDF_AddHighlightRect, 'REDRECT', Cardinal(@t));
End;
```

### 4.4.14 Render metafiles to pages

The commands
COMPDF_StampMetafile and COMPDF_StampMetafileUnder can be used to apply metafiles to pages.

The expect a metafile handle which will be rendered under or over the page contents. This requires the PLUS edition of WPViewPDF.

The string parameter is a page list "1..x" which contains the pages the metafile should be applied to.

In case you want to stamp a bitmap please us AddImage and UseImage.

Please call command COMPDF_StampMetafile_Scaling with parameter 72 before you use the stamping.

COMPDF_StampMetafile = 495:
StrParam = page list, for example 1-2,
IntParam = Metafile Handle - over page

COMPDF_StampMetafileUnder = 496;

COMPDF_StampMetafile_Scaling = 614 : Set the scaling resolution for the commands COMPDF_StampMetafile and COMPDF_StampMetafileUnder. Default is the screen resolution. We recommend to set it to 0 to select the resolution automatically.

Example:

```
if WPViewPDF1.LoadFromFile(background.pdf') then
begin
            Metafile1 := WPViewPDF1.GetMetafile(0);
end
else Metafile1 := nil;


if WPViewPDF1.LoadFromFile(any.pdf') then
begin
       WPViewPDF1.Command( COMPDF_StampMetafile_Scaling, 0 );
       if Metafile1<>nil then
       begin
           WPViewPDF1.CommandStrEx (COMPDF_StampMetafileUnder, '1', MetaFile1.Handl
           WPViewPDF1.Plus.SaveToFile('test_modified.pdf');
     end;
end;
MetafileS1.Free;
```

## 4.5 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. It is also possible to draw rectangles.

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

**New:** The function pdfMerge can load a stamp script from a file using the option STAMPFILE=sometextfile.txt.

*The script uses a very easy syntax which makes it possible to use precreated*

*macros and just change data parts, for example by using the %s and %d format string used by Format() or sprint().*
*The X and Y offset values (X,Y,XOFF and YOFF command) can be used to move a precreated label to a different place on the page.*

The data is added incrementally, this means normally each subsequent output is added to the existing. To avoid this, the command @cleartext has to be used to clear the previous script on the modified page.
Note: If @cleartext is used, it must be used after the command "PageNo=...".

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. It must be used after "PageNo=..." otherwise you can see overprinting of text.
@ClearText

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

LineColor=0 0 0 will set the line color for a rectangle.

**Select the font**
Font=Arial
Font=Courier New

**Select the Size**
Size=10

**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

**Output Text:**
Texts are printed like this:
X,Y=some text

To continue the text after the last character use "**?**"
?more text

Use "Lineheight", "LH" or "tl" to specify the line height
LH=20

To insert a new line use the command "**CR**" or "T*". TL must be used before CR!
CR

Example:

```
Color=0 0 0
tl=20
67,120=some text
CR
?more 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!

**Specify Offsets for next text and draw commands**
Reset the offset:
**X**=xoffset
**Y**=yoffset
Modify the offset (add to offset):
**XOFF**=offset of xoffset
**YOFF**=offset of yoffset

(Please note that YOffset will be subtracted from the PDF coordinates since PDF coordinate system is bottom-up)

**Switch off macros**
MACROS=off

**The following macros are understood to print page numbers** 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)

When lines have to be drawn this commands can be used:

| | |
|---|---|
| **Save** | saves the current graphic state, mainly the color for line and background |
| **Restore** | restores the saved graphic state |
| **M**=*x y* | move to a certain position. (x and y are delimited by a space!) |
| **L**=*x y* | draw a line from last position to the new position |
| **RE**=*x y w h* | draw a rectangle |
| **S** | draw lines |
| **F** | fill rectangles |

**B** draw lines and fill.
**linewidth** sets the line width in pt
Unless S, F or B is used, no graphics will be visible!

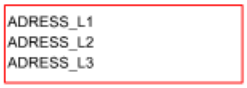
Example 1 - draw underlined text

```
PageNo=1
@cleartext
253,260=AAAAAAAAAAAAAAA
M=250 260
L=380 260
S
```

Example 2:

```
PageNo=1
@cleartext
Save
LineColor=1 0 0
LineWidth=3
re=94 152 198 67
S
Restore
TL=14
100,160=Line 1
CR
?Line 2
CR
?Line 3
```

Example 3 - draw a multiline stamp inside of a frame.




```
PageNo=1
@cleartext
Save
LineColor=1 0 0
X=100
Y=150
re=4 4 175 57
S
Restore
LH=15
6,20=ADRESS_L1
CR
?ADRESS_L2
CR
```

?ADRESS_L3

Example: draw a cross on all pages

```
PageNo=all
@cleartext
LW=5
LineColor=1 0 0
X=100
Y=100
M=0 0
L=200 50
M=200 0
L=0 50
S
```

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;
```

### 4.5.1 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
@cleartext
ORIGIN=2
-40,-25=[#]
@Page nubering part 2 - arabic
NUMFORMAT=1
NUMOFFSET=-3
PageNo=4-9999
@cleartext
ORIGIN=2
```

-40,-25=[#]

## 4.6 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()**

## 4.7 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;
```

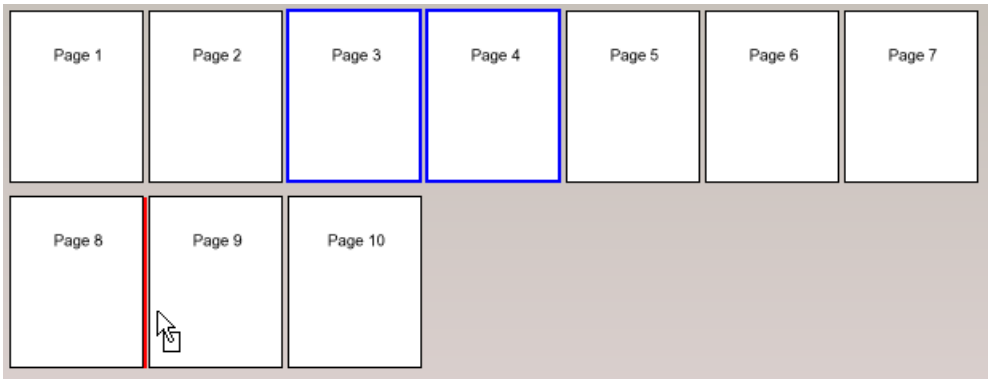
## 4.8 Page moving

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;
```

## 4.9 Initialize JBIG2 plugin

**JBIG2 support is not 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, without showing a window.

WPViewPDF now includes a JBIG2 decode implemented in the module **wpdecodejp.dll** and, for 64 bit, **wpdecodejp64.dll**.
It is **not** required to call the command COMPDF_SetJBIG2Tool when the converter DLLs have been copied to the EXE directory. It is also not required to install the plugin exe.
Command COMPDF_SetJBIG2Tool with an integer parameter 1 and the path as string parameter can be used to manually load the decoding DLL. It will return 1 if the DLL was loaded, 0 if not.

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}").
The token {in} will be replaced by the engine with a temporary file name of the input data,
{out} will be replaced by the engine with the name of the temporary output file. (Both files will be deleted when finished.)
{dll} will be replaced by the engine 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.*

The tool jbig2dec is licensed under GNU license -
You can download a C++Builder project including source and windows binary here (last update 30.4.2014).

WPViewPDF does not integrate the tool, however it is possible to call it as external plugin.

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

**License text file which comes with jbig2dec (it is included in the project ZIP file)**

The files in this directory (folder) and any subdirectories (sub-folders) thereof are part of jbig2dec, with the exception of certain source files included to support portability which are marked otherwise in their copyright headers.

jbig2dec is free software; you can redistribute it and/or modify it under the terms the GNU General Public License as published by the Free Software Foundation, either version 2 of the License, or (at your option) any later version.

This software is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
General Public License for more details.

You should have received a copy of the GNU General Public License along with this program in the file named COPYING. If not, write to the Free Software Foundation, Inc., 59 Temple Place Suite 330, Boston, MA 02111-1307, USA.

In addition, specific permission is given to link jbig2dec to or compile jbig2dec into AFPL Ghostscript and to distribute same under the Aladdin Free Public License (AFPL) version 9.

## 4.10 Trouble Shooting

Important:
In case you decide to rename the DLL WPViewPDF04 ... do not choose a file name which contains "Demo04".

If the SaveToFile or CopyToClipboard function does not work for you, please check the setting of property **SecurityOptions**!

The flag wpDisableSave must not be set - if it is set once, saving cannot be enabled again!

Using the compiler switch **IGNORE_SECOPT_IN_DFM** it is possible with Delphi to disable that the property SecurityOptions is loaded from the DFM data. This makes it possible to set the property later in code.

a) Focus (**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);

b) Unload DLL

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.

c) Access Violation?

In case You use MadExcept please make sure to use the latest version of MadExcept. Otherwise it is possible that you see an Access Violation at address 0x000014 after the control was created.

d) File stays open after Form.Close

WPViewPDF was designed to keep the loaded PDF file in memory even if the window of the viewer was destroyed. The data will be released when the component is destroyed. This behaviour makes it possible to implement a docking feature.

e) To make sure the data is release when the form is closed (but not freed) call the

method Clear or disable the compiler symbol **ENABLE_WNDRECREATE** in the file WPViewPDF3.PAS.

f) If pages in some PDF files appear to be blank, the JBIG2 decoding DLL was probably not loaded. The file wpdecodejp.dll must be copied for 32 bit, the file wpdecodejp64.dll for 64 bit projects.

g) With **SetGlobalParameter("DisableThreading=1")** multi threading can be disabled.
This should be done before the viewer window was created.
If highest possible stability is required, we recommend this setting.

h) In case the project remains in the process list after closing it:

The function wpdfSetGlobalParameter("StopIGDIPlus", 0) can be called before the DLL is unloaded to avoid trouble with GDI+ which under certain circumstances cannot be shutdown in the finalization of a DLL.
(The VCL however will automatically make this call before the FreeLibrary in "StopEngine".)

i) In case the memory consumption of the DLL is too high, SetGlobalParameter can be called with the parameter "MinimizeMemoryUsage=1". This will disable caching of the PDF page paths. Text selection is impossible in this case.

j) It is usually **not** required to call the command COMPDF_SetJBIG2Tool when the converter DLLs have been copied to the EXE directory.
The Command COMPDF_SetJBIG2Tool with an integer parameter 1 and the path as string parameter however can be used to manually load the decoding DLL.
It will return 1 if the DLL was loaded, 0 if not. With an integere parameter=2 it will unload the DLL. If the DLL was already loaded, it will not be loaded again. Please note, that in a 64bit process you need to load the 64bit DLL - the 32 bit DLL will not work.

k) **Security forbids saving** - if you get this message when you try to save a PDF file, the PDF or the component has the save function disabled. Please note, that once the property SecurityOptions was used to disable saving, it cannot be activated again. Please also check out the security commands and the Load&Save topic in the manual. Note: If a PDF was encrypted using AES256 it is not possible to lift the security.

## 4.11 Fields/Widgets and PDF form fill

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

Also see the next chapter "PDF Forms" for information about the interactive form filling.

### 1) Get data from form, set data in form using code

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.

You can use the number -1 to initialize the internal AcroField table. It is always initialized after the viewer was cleared.

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.

It is possible to read a special ID for a field using
  id := WPViewPDF1.CommandEx(COMPDF_ACRO_GET, Cardinal(-3));

The ID can be later used to select a certain field in case you use the interactive form filling

Use the number -4 to extract all field names as XML script, use -5 to get a comma separated list of all fields.

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).

Values for combobox fields (Fieldtype="Ch") can also be written. Right now it is not verified, that the new value is part of the "Opt" array - however the index is set accordingly, if the value is part of that list.

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.

If Acro_Set was used, since version 3.11.6 WPViewPDF will write the parameter "NeedAppearances true" into the acroform object to make sure, Acrobat Reader displays the current values.

Example: This code fills a scrollbox with labels, edits and checkboxes.

```
procedure TWPViewPDFDemo.LoadAcroFieldsExClick(Sender: TObject);
var i, l, y, x, id : Integer;
    s, ts : WideString;
    sa : AnsiString;
    ctrl : TControl;
    lab   : TLabel;
    ed    : TEdit;
    chk  : TCheckBox;
begin
  FFields.Clear;
  FieldScroll.Visible := false;
  for I := FieldScroll.ControlCount - 1 downto 0 do
  begin
      ctrl := FieldScroll.Controls[i];
      ctrl.Parent := nil;
      ctrl.Free;
  end;

  i := 0;
  y := 0;
  x := 96;
  SetLength(FieldToIndex, 100);
  repeat
    ts := WPViewPDF1.CommandGetStr(COMPDF_ACRO_GET,'FT', i);
    // Get AcroID
    id := WPViewPDF1.CommandEx(COMPDF_ACRO_GET, Cardinal(-3));

    l := WPViewPDF1.CommandEx(COMPDF_ACRO_GET, i);
```

```
    if (l>0) and ((ts='Ch') or (ts='Tx') or (ts='Btn')) then
    begin
      SetLength(s, l);
      WPViewPDF1.CommandEx(COMPDF_GetTextBufW, Integer(PWideChar(s)));
      l := Pos('=', s);
      if l=0 then l := Length(s)+1;

      lab := TLabelEx.Create(FieldScroll);
      TLabelEx(lab).FID := id;
      lab.Caption := Copy(s,1,l-1);
      lab.Parent := FieldScroll;
      lab.Left := 2;
      lab.Top  := y;
      lab.Height := 18;
      lab.Width  := x - 2;
      lab.Color := clHighlight;

      lab.OnClick := FieldLabelClick; // procedure FieldLabelClick see below

      FFields.Add(lab);

      if ts='Btn' then
      begin
         chk  := TCheckBoxEx.Create(FieldScroll);
         TCheckBoxEx(chk).FID := id;
         chk.Tag := i;
         chk.Parent := FieldScroll;
         chk.Left := x;
         chk.Top  := y;
         chk.Height := 18;
         chk.Width  := 120;
         chk.OnClick := FieldScrolUpdate;
         chk.Checked := Copy(s,l+1,Length(s))='1';
         FFields.Add(chk);
      end else
      begin
          ed := TeditEx.Create(FieldScroll);
          TeditEx(ed).FID := id;
          ed.Text := Copy(s,l+1,Length(s));
          ed.Tag := i;
          ed.Parent := FieldScroll;
          ed.Left := x;
          ed.Top  := y;
          ed.Height := 18;
          ed.Width  := 120;
          ed.OnChange := FieldScrolUpdate;
          FFields.Add(ed);
      end;

      inc(y, 20);
    end;
    inc(i);
  until l<0;
  FieldScroll.Visible := true;

  WPViewPDF1SelectDrawObject(nil,0);
end;

procedure TWPViewPDFDemo.FieldScrolUpdate(Sender: TObject);
begin
  if Sender is TCheckBox then
  begin
      if TCheckBox(Sender).Checked then
           WPViewPDF1.CommandStrEx(COMPDF_ACRO_SET,
                '1', TCheckBox(Sender).Tag)
      else WPViewPDF1.CommandStrEx(COMPDF_ACRO_SET,
```

```
                'Off', TCheckBox(Sender).Tag);
  end else if Sender is TEdit then
  begin
    WPViewPDF1.CommandStrEx(COMPDF_ACRO_SET,
                TEdit(Sender).Text, TCheckBox(Sender).Tag)
  end;
end;

procedure TWPViewPDFDemo.CurrObjChange(Sender: TObject);
begin
  WPViewPDF1.CommandStrEx(COMPDF_ACRO_SET,
                TEdit(Sender).Text, Cardinal(-2));
end;
```

### 2) Activate interactive form filling:

Activate the flag **wpAllowFormEdit** in the ViewOptions!

For interactive form filling the text and checkbox form fields have to be converted to draw objects first.

This is done by command **COMPDF_ACRO_MAKEDRAWOBJ (=117)**:

WPViewPDF1.CommandStrEx(COMPDF_ACRO_MAKEDRAWOBJ,'',2+8+16);

This commmand converts Acroform fields and annotations into draw objects. This makes the fields EDITABLE!

After that command it is possible to move them however it is not recommended to use the command COMPDF_RenderDrawobjects

The following flags are possible in IntPar
1: The created objects cannot be selected
2: The created objects cannot be moved (create locked objects)

8: Create objects only for selected annotations:
16: only Widgets (editfields, checkboxes ...)
32: only Highlights
64: only Links
128: only FreeText

256: only squares
512: only popups

1024: Read the Annotation F property to select locked and redonly state
2048: Prohibit sizing of the objects (moving is not disabled)
4096: Prohibit edit mode for the widgets (=readonly)

**8192: Automatic Mode** - whenever a new PDF is loaded, COMPDF_ACRO_MAKEDRAWOBJ is executed with the given parameters! We recommend to use this for a PDF viewer which should be

used to edit PDF forms.

The string parameter can further list the annots which are converted or which are NOT converted, i.e. +all, -all, +popup, -popup

You want to convert the fields into draw objects also without using wpAllowFormEdit. In this case the fields are still selectable.

You can use the ID read with CommandEx(COMPDF_ACRO_GET, Cardinal(-3)) to get and change the current field selection, for example to select the current field which is displayed outside of the PDF in Your application.

This code select a certain field in the PDF form after a click on a label:

```
procedure TWPViewPDFDemo.FieldLabelClick(Sender: TObject);
begin
  WPViewPDF1.command(COMPDF_DrawObjectDeSelectAll);
  WPViewPDF1.command(COMPDF_DrawObjectSelect, TLabelEx(Sender).FID );
end;
```

You can use the list SelectedDrawObjects inside the event OnSelectedDrawObjects to check which objects are currently selected:

Example - we use a TLabelEx and TEditEx control which has an additional FID element to store the ID. We do not use the Tag, since we use that already for the index of the field.

```
procedure TWPViewPDFDemo.WPViewPDF1SelectDrawObject
  (Sender: TObject; const ObjID: Integer);
var
  I, J, ID : Integer;
  ctrl : TControl;
begin
  for J := 0 to FFields.Count - 1 do
  begin
    ctrl := TControl(FFields[J]);
    if ctrl is TLabelEx then
    begin
       TLabelEx(ctrl).Transparent := true;
    end;
  end;

  for I := 0 to WPViewPDF1.SelectedDrawObjects.Count - 1 do
  begin
      WPViewPDF1.command(COMPDF_DrawObjectGetSelected, I+1 );
      ID := WPViewPDF1.command( COMPDF_DrawObjectReadProp , OBJPRP_ACROID);
      for J := 0 to FFields.Count - 1 do
      begin
         ctrl := TControl(FFields[J]);
```

```
          if ctrl is TEditEx then
          begin
            if TEditEx(ctrl).FID=ID then
            begin
               TEditEx(ctrl).SetFocus;
            end;
          end
          else if ctrl is TCheckBoxEx then
          begin
            if TCheckBoxEx(ctrl).FID=ID then
            begin
               TCheckBoxEx(ctrl).SetFocus;
            end;
          end
          else if ctrl is TLabelEx then
          begin
            if TLabelEx(ctrl).FID=ID then
            begin
               TLabelEx(ctrl).Transparent := false; // Highlights the field
            end;
          end;
       end;
  end;
end;
```

The list SelectedDrawObjects is updated by this code:

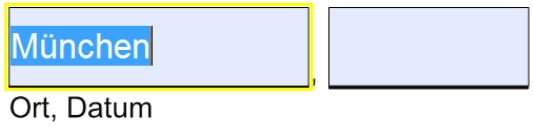
```
var i, j : Integer;
    n : String;
begin
  FSelectedDrawObjects.Clear;
  j := 1;
  if CommandEx( COMPDF_DrawObjectGetSelected, 0 )>0 then
  repeat
     i := CommandEx( COMPDF_DrawObjectGetSelected, j);
     if i<>-1 then
     begin
          n := CommandGetStr( COMPDF_DrawObjectReadProp, '', OBJPRP_NAME );
          FSelectedDrawObjects.AddObject(n, TObject(i))
     end;
     inc(j);
  until i=-1;
end;
```

OnSelectedDrawObjects is triggered by the windows message WM_PDF_DELAYED_UPDATE=$0400 + 88 with wparam=1

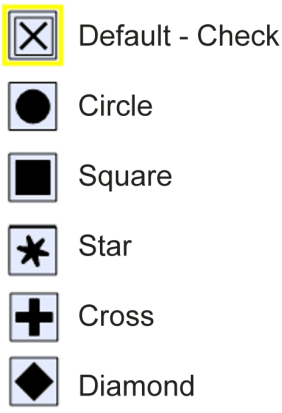
## 4.12 PDF-Forms (AcroForms)

WPViewPDF can be used to fill PDF forms. It supports a subset of the available checkbox appearances and text field types. (Active scripting, lists, combos and radio buttons are not yet supported.)

When in form filling mode the user can use TAB to move to the next field and Ctrl+Tab to move to the previous. Space can be used to toggle a checkbox. The active widget is highlighted:



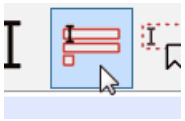

This checkbox types are supported:




To activate form filling mode the editor must use this code in its setup:

```
// Make sure the annotations work interactively!
wpviewpdf.Command(COMPDF_ACRO_MAKEDRAWOBJ,'', 8192 );  // 0=all Annots!

// Enables saving of annotations which have been added to the page (WPViewPDF P
wpviewpdf.Command(COMPDF_Ann_SetAnnotSaveMode, 1);
```

The form filling mode is activated by the action:

 Execute the action "**SelectStdFillForm**"

The command COMPDF_FORMFILLOPTIONS (556) can be used to customize the form filling mode. It also use to activate the new functionality to use the JavaScript actions stored with an annotation to detect date and number fields.

Please use **COMPDF_FORMFILLOPTIONS** with a bitfield:

1: activate the annotation actions
2: disable the use of TAB to move to next annotation
4: do not highlight the focussed annotation
8: use date masks for date fields (uses bit 1 as well). Such fields use AFDate_KeystrokeEx in their action.
16: use number masks for number fields (uses bit 1 as well). Such fields use AFNumber_Keystroke.
32: use special masks for fields which use the AF (uses bit 1 as well). Activated by AFSpecial_Keystroke.

the command **COMPDF_SetAnnotEditModes** (555) can be used to modify the TAB key works when in formular mode:
1: in formular mode skip readonly edit fields
2: keep creation order when tabbing through objects

Please note:
Currently form editing is not supported on forms which are rotated or of fields which are rotated. (MK /F property)

## 4.13 Annotation support

This feature was added to WPViewPDF V4 PLUS.

The command **COMPDF_ACRO_MAKEDRAWOBJ** with a string and an integer parameter can be used to convert all or certain annotations in **existing PDF files** into objects which can be selected, moved and deleted.

The string parameter selects the type of annotations which are converted. The subtyle name can be specified with "+" to be included or "-" to be excluded. Also supported is "+all" and "-all".

Fields use the subtype "widget". Here it is possible to further differentiate using the fieldtype: "ftTx", "ftCh" and "ftBtn" are possible.

Example:
"+all,-popup" will convert all annotations except for popup
"-all,+ftTx" will only convert text fields.
"-all,+ftTx,+square,+highlight" will only convert the types which can be currently created by WPViewPDF.

The integer parameter of command COMPDF_ACRO_MAKEDRAWOBJ can use this bits:
1: The created objects cannot be selected
2: The created objects cannot be moved or sized (create locked objects)
4: The created objects cannot be deleted (Deletion is supported by WPViewPDF V4 only)
1024: Read the fieldflag to select locked and readonly state
2048: The object may be moved, but not resized
4096: The widgets (edit fields) do not display an inplace editor on click.

**8192: Auto mode - execute the command whenever a file was loaded. (recommended)**

**COMPDF_Ann_SetAnnotSaveMode** - Enables saving of annotations which have been added to the page

Command(COMPDF_Ann_SetAnnotSaveMode, 1);

This property bits are supported:

1 : Add the draw annotations which were created on the page
2 : Recreate all annotations, also those which were not modified
4 : reserved
8 : selectively remove all widgets while saving. Must be combined with 1
16: selectively remove all non widgets while saving. Must be combined with 1

You can "flatten" the PDF annotations. Since the annotations have been rendered into the PDF pages you probably do not want to save them. Use **COMPDF_Ann_SetAnnotSaveMode, (1+8+16)** to save the PDF without.

To flatten the annotations use command

command( COMPDF_RenderDrawobjects, 1+32 )

also see command **COMPDF_Ann_ModifyAddProps.**

**COMPDF_Ann_AddAnnotation** is used to either add an annotation at once, or let the user draw a frame to where the annotation is created. It is possible to specify a AcroField ID for a new widget annotation.

COMPDF_Ann_AddAnnotation requires the address of a parameter structure:

```
TWPAddAnnotationParam = record
```

```
    Mode    : Integer;
    pageno : Integer;
    x,y,w,h : Single;
    typ : PWideChar;
    Props: PWideChar; // CommaList TStrings;
    PopupID : PWideChar;
    AddAnnotMode : Integer; // Bitfield: "TWPAddAnnotMode"
    // 1   wpAddWidget,   // Name is FT, not subtype
    // 2   wpAddPopup, // Add Popup - Append Reference to PopupList
    // 4   wpAddAlsoAcroField, // Also adds a field in the acrofield tree at the g
    //     does not work if AcroXID<>0
    // 8   wpAddThenSelectObject, // After object creation the user may select and
    // 16  wpAddAtMouseRect, // The user may draw one rect and an annot is created
    // 32  wpAddAtMouseRectContinue // The user may create another after the first
    // 64  wpSelectTextToQuadPoints // The user may select text and the highlight
    // This also activates the text selection mode!

    FieldPathName : PWideChar;
    FieldValue    : PWideChar;
    AcroXID : Integer;
    Reserved  : Integer; // Must be 0
end;
```

The element Props controls the properties and the type of the new annotation. It must be provided as a comma separated list. The properties which should be written to PDF can be encoded here. The type of the PDF property is determined by the prefix s, a and n:

s. creates a string, i.e. "s.Contents=This is the contents of a field"
a. creates an array, i.e. "a.B=0 0 0" selects the border color
n. writes any number, i.e. "n.F=123"

The values *color* and *alpha* are understood without a prefix, since they are not written to PDF.

The element ***typ*** may be the name of the annotation. Please use "highlight" or "square". To create a widget use "edit" or "memo" which are interpreted internally.

Please note: "F=4" must be defined, otherwise the annotation is visible on screen but will not be printed by Acrobat Reader.

With WPViewPDF 4 PLUS it is also possible to modify a selection of properties of the currently selected annotations.

This commands are used to implement a property inspector for fields and annotations

**COMPDF_Ann_XMLGetFromAcrofield** = 572; // Read data from selected fields. Use StrParam for params which are not shared
**COMPDF_Ann_XMLGetFromAnnots** = 573; // read data from select annots.
**COMPDF_Ann_XMLSetToAcrofield** = 574; // Read data from selected fields.

Dont modify params which use StrParam as param
**COMPDF_Ann_XMLSetToAnnots** = 575; // read data from select annots.
Return count of modified objects.

Also see example.

If you need to convert the annotations into regular PDF drawing code ("flatten" a PDF file) use command
**COMPDF_RenderDrawobjects**

## 4.14 Messages

The following message IDs are sent to the parent window as window message WM_PDF_EVENT (= $0400 + 78)

With the VCL component it is possible to use the event OnViewerMessage to trap the events.

```
MSGPDF_NEEDPASSWORD     = 100.............. Set a new password!

MSGPDF_PROBLEMONLOAD    = 101.............. We have a problem while
loading the file

MSGPDF_PROBLEMONDISPLAY = 102.............. We have a problem while
displaying the file

MSGPDF_INITCOMMANDS     = 103.............. Set the command offsets
(lparam)

MSGPDF_CHANGEVIEWPAGE   = 104.............. Moved to different page
(=wparam)

MSGPDF_SETVERSION       = 105.............. lparam = version * 1000


MSGPDF_INTERNEXCEPTION  = 107.............. Send exception string

MSGPDF_CHANGESELPAGE    = 108.............. Moved to different page
(=wparam)

MSGPDF_MAPFONT          = 109.............. Use COMPDF_MAPFONT to
change the font
```

```
MSGPDF_PRINTSTART       = 110..... Param = page count in the
printing cue (not the total page count)

MSGPDF_PRINTPAGE        = 111..... for each page, LParam =
page number in document (see MSGPDF_PRINTPAGEPROGRESS)

MSGPDF_PRINTEND         = 112..... done, LParam=0

MSGPDF_PRINTPAGEPROGRESS= 116..... Message to set
prograsspar. param=position, max=param of MSGPDF_PRINTSTART


MSGPDF_FIND_START       = 113..... Find process started

MSGPDF_FIND_PAGE        = 114..... Find process running. The
parameter is the current page number

MSGPDF_FIND_END         = 115..... Find process started. The
parameter is the found page number


  MSGPDF_MOVEPages        = 140..... The user dragged
selected pages - they should now be moved. Use command
COMPDF_MovePages

MSGPDF_BEFORE_MOVEPages = 141..... To enable moving the
program must set ResultA to 1

//GUI Events
  MSGPDF_KEYDOWN          = 201..... l Param = Key,
SetResult with
  MSGPDF_KEYPRESS         = 202..... l Param = Key
  MSGPDF_KEYUP            = 203..... l Param = Key

  MSGPDF_DblClick         = 204..... Doubleclick

  MSGPDF_SetFocus         = 205..... Triggered when
internally SetFocus is executed
```

MSGPDF_DrawBackground = 206; // Draw page background - for example to implement skinning

MSGPDF_DRAWOBJECT_GETTEXT= 257; // Get the text for a text draw object. Requires WPViewPDF V4, lparam=current page number

## 4.15 Convert PDF into watermark

This feature was added to WPViewPDF V4. If you have licensed the PLUS edition it is also possible to save to a new PDF file with the new watermarks.

The command **COMPDF_LoadFileAsWatermark** is used to load a PDF file and convert a certain page into a watermark. The ID of the new watermark is retuned. If the PDF file was already loaded, it will be reused. So it is possible to subsequently use different pages from the same PDF file.

The command **COMPDF_ApplyWatermark** is used to apply the watermark with the given id to certain pdf pages. The string parameter of this command is used to select the destination pages. It is possible to select "all", "odd" or "even" pages. Alternatively a range can be specified, such as "1-3,5,7,9-1000".

To add a positioning mode add it after ";". Positioning modes are 0=over page, 256=under page (default)

Instead of page list also the text *odd* and *even* is allowed. Returns the number of modified pages

**Example:**

```
i := WPViewPDF1.CommandStrEx( COMPDF_LoadFileAsWatermark, OpenDialog1.FileName
if i<=0 then
   ShowMessage('The PDF file cannot be loaded as watermark - Code=' + IntToSt
else
begin
  if rbOddPages.Checked then
     i := WPViewPDF1.CommandStrEx( COMPDF_ApplyWatermark, 'odd', i)
  else if rbEvenPages.Checked then
     i := WPViewPDF1.CommandStrEx( COMPDF_ApplyWatermark, 'even', i)
  else   i := WPViewPDF1.CommandStrEx( COMPDF_ApplyWatermark, edPageRange.Text

  if i<=0 then
   ShowMessage('The PDF page cannot be applied as watermark');
end;
```

## 4.16 Use WPViewPDF with ImageEn

**ImageEn** is an extensive component suite for image editing, display and analysis written in pure VCL code for Delphi and C++ Builder, and is also available for .NET. Thousands of software developers use ImageEn to add powerful multimedia functionality to their applications.
Please visit www.imageen.com for more information.

The WPViewPDF DLL include a function called **pdfMakeImageExt** which can be easily used with ImageEn to load bitmap representations into this powerful imaging library.

The unit wpcubed_pdf_plugin has been provided to make this as easy as possible. Usually all you have to do, is to include this unit to the uses clause in your application. Then your application will use the WPViewPDF DLL to show PDF data inside the ImageEn viewer.

It is possible to deactivate the auto-registration in wpcubed_pdf_plugin.pas using a

compiler symbol. In that case please add this code to your application:

```
if TIEWPCubedPDF.Initialize then
    TIEWPCubedPDF.RegisterPlugin
 else ShowMessage('PDF decoder DLL could not be found');
```

The name of the WPViewPDF DLL has to be specified in the file PDFLicense.INC. There also the license codes for the registered version of WPViewPDF must be included. Optionally it is possible to specify a fully qualified DLL name in TIEWPCubedPDF.Initialize.

Please note that this DLLs are required for PDF rendering - they must be included in the applications binary directory.
wPDFView04.dll
wpdecodejp.dll
wp_type1ttf.dll

## 4.17 Internal Actions

WPViewPDF V4 includes powerful feature: internal "Actions".

The internal actions are internal classes wich control the operation of the WPViewPDF viewer.

Each of the classes is of a certain "kind", the operation group. i.e. "File" is the "kind" of the open action.

Then it has a certain operation number, i.e. 1="Open" and 2="Append" within the group "File".

For GUI setup each action has a caption and a hint string property and of course it has a name. Using the name it is possible to execute an action, but with GUI usually its ***number*** is used. The number can be stored as the kind number in the high word and the operation number in the low word. Please do not relay on this number to not change - to identify an action persistently better use its name.

The number for a certain named action can be retrieved with COMPDF_ACTION_READ and "**?**"+actionname:

```
int acn = pdf.CommandStrEx( COMPDF_ACTION_READ, "?" + action_name );
```

COMPDF_ACTION_READ and COMPDF_ACTION_WRITE can also be used to localize the captions.

### 4.17.1 List of Actions

This is a list of the actions, name=caption;hint.

The list was created using the command
WPViewPDF1.CommandGetStr(COMPDF_ACTION_READ, 'actionnames', 3 );

```
***File
FileOpen=Open;Open
FileAppend=Append;Append
FileClose=Close;Close
FileSaveAsPDF=Save as ...;Save as ...
FileSaveAsText=Save as text ...;Save as text ...
FileSaveAsImage=Save as image ...;Save as image ...
FileSaveSelectionAsPDF=Save selection as ...;Save selection as ...
FileSaveSelectionAsText=Save selection as text ...;Save selection as
text ...
PDFWatermark=Apply PDF watermark;Apply PDF watermark
Print=Print;Print
PrinterSetup=Setup Printer ..;Setup Printer ..
PrintSelection=Print selection;Print selection
PrintDialog=Print ...;Print ...

***View
Zoom100=Zoom 100%;Zoom 100%
ZoomIn=Zoom in;Zoom in
ZoomOut=Zoom out;Zoom out
ZoomFullWidth=Zoom to page width;Zoom to page width
ZoomFullPage=Zoom to full page;Zoom to full page
ZoomTwoPages=Doublepage view;Display two pages side by side
ZoomGetCurrent=Read zoom value;Read zoom value
ZoomSave=Save Zoom;Save Zoom
ZoomRestore=Restore Zoom;Restore Zoom
ZoomToRect=Zoom to frame;Zoom to frame
ZoomThumbs=Display thumbnails;Display thumbnails
Zoom=Zoom;Set zoom value directly
ZoomThumbnailsIn=Enlarge thumbnails;Enlarge thumbnails
ZoomThumbnailsOut=Shrink thumbnails;Shrink thumbnails
GotoFirst=First Page;First Page
GotoPrev=Previous Page;Previous Page
GotoPrevPos=Go Back;Go Back
GotoPage=Goto Page;Goto Page
GotoNext=Next Page;Next Page
GotoLast=Last Page;Last Page
***Page

PageSelectToggle=De-/Select Page;De-/Select Page
PageSelectByParam=Select Pages ...;Select Pages ...
PageSelectClear=Clear page selection;Clear page selection
PageSelectInvert=Invert page selection;Invert page selection
AppendPage=Append page;Append page
PageDelete=Delete page;Delete page
```

```
PageUndelete=Undelete page;Undelete page
PageRotateLeft=Rotate page left;Rotate page left
PageRotateRight=Rotate page right;Rotate page right
PageMove=Move selected pages after current;Move selected pages after
current
```

*****Edit**

```
Delete=Delete;Delete selected objects
CopyToClipboard=;
SelectStd=Click and Pan;Click and Pan
SelectFillForm=Fill form;Fill form
SelectText=Select text;Select text
SelectObjects=Select objects;Select objects
SelectObjectsLocked=Select objects, protected mode;Select objects,
protected mode
```

*****Draw**

```
DrawHighlight=Draw highlight;Draw highlight
DrawRect=Draw rectangle;Draw rectangle
DrawCircle=Draw circle;Draw circle
DrawImage=Input image;Input image
DrawTextline=Draw textline;Draw textline
DrawTextBox=Draw textbox;Draw textbox
ApplyDrawObjects=Apply graphic objects;Render the draw objects on
the pages
ClearDrawObjects=Clear all;Clear all draw objects on all pages
ClearSelectedDrawObjects=Clear selected;Clear selected draw objects
added in 4.1.3 (menu is hidden by default)
```

FlattenAnnotations=flattens the annotations. StrParam can be "true" to also delete the annotations.

*****Annotations**

```
DrawAnnotAny=Annotation;Annotation
DrawAnnotHighlightText=Highlight text;Highlight text
DrawAnnotBlackText=Black text;Black text (1)
DrawAnnotHighlight=Highlight box;Highlight box
DrawAnnotFrame=Frame;Frame
DrawAnnotSymbol=Symbol with Popup;Symbol with Popup
DrawAnnotFreetext=Freetext;Freetext
DrawAnnotSquiggly=Squiggly Underline;Squiggly Underline
```

*(1) It is possible to select black as highlight color which makes the text unreadable when printed or exported as image file. (Important: This feature does not delete the text)*

*****Draw Options**

```
DrawChangeColor=Change Color;Change Color
DrawChangeBGColor=Change Background Color
DrawChangeAlpha=Change Alpha
DrawChangeFont=Change Font
DrawChangeFontSize=Change FontSize
```

```
***Fields
DrawFieldEdit=Create Textfield;Create Textfield
DrawFieldMemo=Create Memo field;Create Memo field
DrawFieldCheck=Create checkbox;Create checkbox
DrawFieldCheckR=Create round checkbox;Create round checkbox

***Extra
Clear=Clear;Clear
Threading=Threading;Threading

***Info
About=About;About
ToogleLeftPanel=Show/Hide left panel;Show/Hide left panel
ShowThumbnails=Show thumbnails;Show thumbnails
ShowBookmarks=Show bookmarks;Show bookmarks
DocumentProps=Document Properties;Document Properties
```

### 4.17.2 Execute an Action

To execute an action you can use either the command

**COMPDF_ACTION (580)**
or
**COMPDF_ACTIONNR (581)**

COMPDF_ACTION expects the action name + "=" + the parameter as comma separated list in the string parameter. If the action was not found, the result is -2. Otherwise the usual result is returned, -1 means "default".

```
procedure TForm1.Executeanaction1Click(Sender: TObject);
var vals : array[0..1] of string;
begin
   vals[0] := 'DrawAnnotHighlight';
   vals[1] := '"Color=Red","Alpha=50"';
   if InputQuery('Execute Action',['Name','Parameter'],vals) then
   begin
       if pdf.CommandStr(COMPDF_ACTION, vals[0] + '=' + vals[1]   )=-2 then
          ShowMessage('The action ' + vals[0] + ' was not found');
   end;
end;
```

COMPDF_ACTIONNR expects the action number (high word=group, low word = operation) in the integer parameter and the optional parameters in the string parameter.  If the action was not found, the result is -2.

This code works as the example above - it first retrieves the number of the named action. Such a number is also used for COMPDF_ACTION_READ which is discussed below.

```
procedure TForm1.Executeanaction1Click(Sender: TObject);
```

```
var vals : array[0..1] of string;
    acn : Integer;
begin
   vals[0] := 'DrawAnnotHighlight';
   vals[1] := '"Color=Red","Alpha=50"';
   if InputQuery('Execute Action',['Name','Parameter'],vals) then
   begin
      acn := pdf.CommandStr( COMPDF_ACTION_READ, '?' + vals[0] );
      if acn<=0 then
           ShowMessage('The action ' + vals[0] + ' was not found')
      else pdf.CommandStrEx(COMPDF_ACTIONNR, vals[1], acn );
   end;
end;
```

Optionally parameters can be passed to the execution methods. Which parameters are required can be automatically retrieved using **COMPDF_ACTION_READ** with the action number and the string parameter set to "param" or "paramkind":

```
 param     := WPViewPDF1.Command(COMPDF_ACTION_READ, 'param', acn );
 paramkind := WPViewPDF1.Command(COMPDF_ACTION_READ, 'paramkind', acn
);
```

Please see our Delphi and Visual Studio example code.

param is a bit field. Bit 2 is set, if a string parameter is expected by the action.

1=require Intpar,
2=require string par,
4=read intpar,
8=read string par,
16 Boolean,
32 OPTIONAL String - usually properties
64 - this is a GUI Boolean, Show true/false action

paramkind can have the following values (some are reserved)

0: Pagenr as Int or string
1: Fontname as string
2: Color as Int or string
3: PDF filename as string OPEN
4: PDF filename as string SAVE
5: text filename as string OPEN
6: text filename as string SAVE
7: image file name as string OPEN
8: JPEG file name as string SAVE
9: type @ options_comma_list
10: options_comma_list
11: options_for_DrawObjects

12: Zoom Value as Int
13: JPEG image file name as string to OPEN passed as "file=...",... + other params
14: some text as string passed as "text=...",... + other params
15: Transparency in range 0..255
16: Boolean "true"/"false" "1"/"0"
17: Fontsize as number in string
50: Ask $hint$ yes/now

Options for the draw objects and annotations are usually passed as comma separated list, i.e *"Color=Red","Alpha=50".*

Drawobjects and annotations support this property names

Color - this is the color as HTML color
Alpha - this is the transparency, 1=transparent, 255=solid

Font - this is the font for a freetext annotation
Font-Size - the size for the text
Font-Color - the color for the text

The DrawAnyAnnot action also accepts the parameter
type=pdf annotation type, i.e. "type=Link"

Example:

```
pdf.CommandStrEx( COMPDF_ACTION,
    'DrawAnnotFreetext="font-color=red","font-size=18","font=courier new"', 0);
```

Internally the options are stored as XML.

Annotations are saved as PDF objects and can have additional parameters which can also be set using the parameter list.

The name of each of the additional PDF properties must start with "prp.". (lowercase!)

After that a single letter differentiate between the possible parameter types - all are case sensitive!
s - this is a PDF string type
n - this is a PDF name type
v - this is any value type
i - this is an integer
a - this is an array type, it will be written between [ and ].
d - this is a dictionary - it will be written between << and >>.
r - this make it possible to use page *references*.
Internally the function will replace all #nnn or #{xxx} values by the correct page references or /null if not found.

This feature can be used to write a /Dest page reference to be used by a link ( see example).

It is also possible to create parameters for sub dictionaries by simply specifying the name of the dictionary, a "." and then the  property as described above.

### 4.17.3 Add link annotations

Link annotations can be created using the DrawAnnotAny action by specifying the PDF properties which should be created in the PDF file.

Example - add a link to a webpage

```
procedure TForm1.AddWeblink1Click(Sender: TObject);
var s : string;
begin
  s := 'http://www.wpcubed.com';
  if (pdf<>nil) and InputQuery('Add weblink', 'URL', s) then
  begin
    pdf.CommandStrEx( COMPDF_ACTION,
        'DrawAnnotAny="type=Link","prp.i.F=4","prp.a.Border=0 0 0",' +
        '"prp.A.n.Type=Action","prp.A.n.S=URI","prp.A.s.URI=' + s + '"', 0);
  end;
end;
```

Example - add a link to page

In this example the page is provided as number, rage 1..pagecount

```
procedure TForm1.Addlinktopage1Click(Sender: TObject);
var s : string;
    i : Integer;
begin
  s := IntToStr(pdf.Page);
  if (pdf<>nil) and InputQuery('Add link to page', 'Nr', s) then
  begin
    i := StrToInt(s);
    if (i<1) or (i>pdf.PageCount) then
         ShowMessage('Pagenumber not valid')
    else
    begin
     // this code uses the page number directly.
       pdf.CommandStrEx( COMPDF_ACTION,
        'DrawAnnotAny="type=Link","prp.i.F=4","prp.a.Border=0 0 0",' +
        '"prp.r.Dest=[#' + IntToStr(i) + ' /XYZ 0 500]"', 0);
    end;
```

```
  end;
end;
```

In this example the page number is stored as ID. The command COMPDF_GetGetPageObjectID (226) is used to retrieve the page id.
Note: The ID is enclosed in { }.

```
procedure TForm1.Addlinktopage1Click(Sender: TObject);
var s : string;
    i : Integer;
begin
  s := IntToStr(pdf.Page);
  if (pdf<>nil) and InputQuery('Add link to page', 'Nr', s) then
  begin
    i := StrToInt(s);
    if (i<1) or (i>pdf.PageCount) then
         ShowMessage('Pagenumber not valid')
    else
    begin
     // In the code below we use the page identifier
       pdf.CommandStrEx( COMPDF_ACTION,
        'DrawAnnotAny="type=Link","prp.a.Border=0 0 0","prp.i.F=4","prp.r.Dest=[#' +
            pdf.CommandGetStr(COMPDF_GetGetPageObjectID, '', i-1)
          + ' /XYZ 0 500]"', 0);

    end;
  end;
end;
```

### 4.17.4 Modify color of annotation

Annotations are created by this actions:

```
DrawAnnotHighlightText with the defaults "Color=Yellow","Alpha=50"
DrawAnnotBlackText, "Color=Black","Alpha=255"
DrawAnnotHighlight, "Color=Yellow","Alpha=50"
DrawAnnotFrame, "Color=Red","Alpha=255"
```

Using the string parameter other parameters can be passed if required. The default will only be used, if the string parameter is empty.

To modify the currently selected annotations the command COMPDF_Ann_ModifyAddProps can be used.

It expects a string parameter which holds the parameter **and** an integer parameter.

The integer is a bit field:
1 : modify the "current" attributes. This attributes are used by the currently active action or "Draw mode".

2 : modify the attributes of the currently **selected** annotations - mode 2.
4 : Auto Mode: If annotations are selected, they will be modified. If nothing is selected, the "current" attributes are modified.
8 : If the current attributes are changed, also change the defaults for the highlight, frame and freetext actions

Please note that selecting a different draw mode the parameters will be reset to default.

The string parameter can include any of the parameters which can be used with the DrawAnnot action, i.e. Color and Alpha.

Example:

```
procedure TForm1.SetColor1Click(Sender: TObject);
begin
  if (pdf<>nil) and ColorDialog1.Execute then
    pdf.CommandStrEx(COMPDF_Ann_ModifyAddProps,
       'Color=' + ColorToString(ColorDialog1.Color), 4 );
end;

procedure TForm1.SetAlphaClick(Sender: TObject);
begin
  if (pdf<>nil) then
     pdf.CommandStrEx(COMPDF_Ann_ModifyAddProps,
       'Alpha=' + IntToStr((Sender as TMenuItem).Tag ), 4);
end;
```

In Mode 1 also this parameters can be changed:
Fieldname
Value
FieldType
PopupID
AcroXID

This parameters are used by COMPDF_Ann_AddAnnotation when acro fields are created.

For draw objects and annotations this names can be used

Font
Font-Size
Alpha
Line-Width
Line-Color

This names are only used by draw objects

Brush-Color

Text

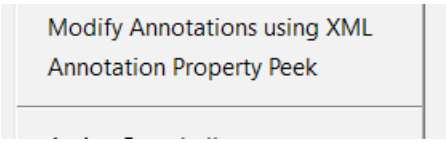
Annotations use

Contents
Values
Color
Background-Color

Highlight annotations use
HighlightType
to select the type. "Highlight" or "Square"

and of course the "prp." property names which can be used to create PDF properties which should be written to the PDF file.

The included "PDFEdit.EXE" has a menu item under "Info":




When "Property Peek" has been activated the XML properties of the object under the mouse cursor will be displayed. This makes it easy to check what parameter names can be used or changed for an object.

## 4.18 ActionModes

The "ActionMode" controls how the user can interact with the editor.

The ActionMode can be controlled by the actions as described with the sample application or it can be changed with this command: **COMPDF_SetActionMode = 557**
This replaces the old command "SelectMode"

**COMPDF_SetActionMode** expects and optional string parameter and an integer parameter which selects the mode:

0 wpacUndefined, // ActionMode was not used yet
1 wpacClickAndPan, // Standard Mode
2 wpacSelectPage, // Select page by click
3 wpacSelectObjects, // Select objects
4 wpacSelectObjectsLocked, // Select objects, does not move them
5 wpacFillForms, // Like wpacClickAndPan and allow widgets to be changed
6 wpacDrawFrameAndZoom, // Zoom to rectangle, Copy rectangle, reset to wpacClickAndPan

7 wpacDrawFrameAndCopyBitmap, // Copy rectangle, reset to wpacClickAndPan
8 wpacSelectText, // Select text and invert visual
8 wpacDrawFrameAndCopyText, // Draw a frame and copy the text within
10 wpacSelectTextAndColor, // Select text and highlight.
11 wpacSelectTextAndBlack, // Select text and black it out
12 wpacSelectTextAndDelete, // Select text and delete (reserved)
13 wpacDrawObject, // Draw a frame, create an object. Uses 2nd parameter
14 wpacDrawObjectContinue, // Draw a frame, create an object.
// Uses 2nd string parameter. Continue drawing
15 wpacDrawAnnot, // Draw a frame, create an Annotation. Uses 2nd parameter
16 wpacDrawAnnotContinue, // Draw a frame, create and the next ...
17 wpacDrawField, // Draw a frame, create an Annotation. Uses 2nd parameter
18 wpacDrawFieldContinue, // Draw a field, create, and the next
19 wpacDrawFrameAndEvent, // Draw Frame and trigger the event

The string parameter is used when creating objects, it can contains information such as the color or the text.

The VCL implements 3 methods to work with the ActionMode.

**SetActionMode can be used to change the action mode and also sets the string parameter.**

```
procedure SetActionMode( aActionMode : TWPViewpdfActionMode;  aActionParam : String = '' );
```

**Read the current action mode - this is useful to update the GUI**

```
function GetActionMode :  TWPViewpdfActionMode; overload;
```

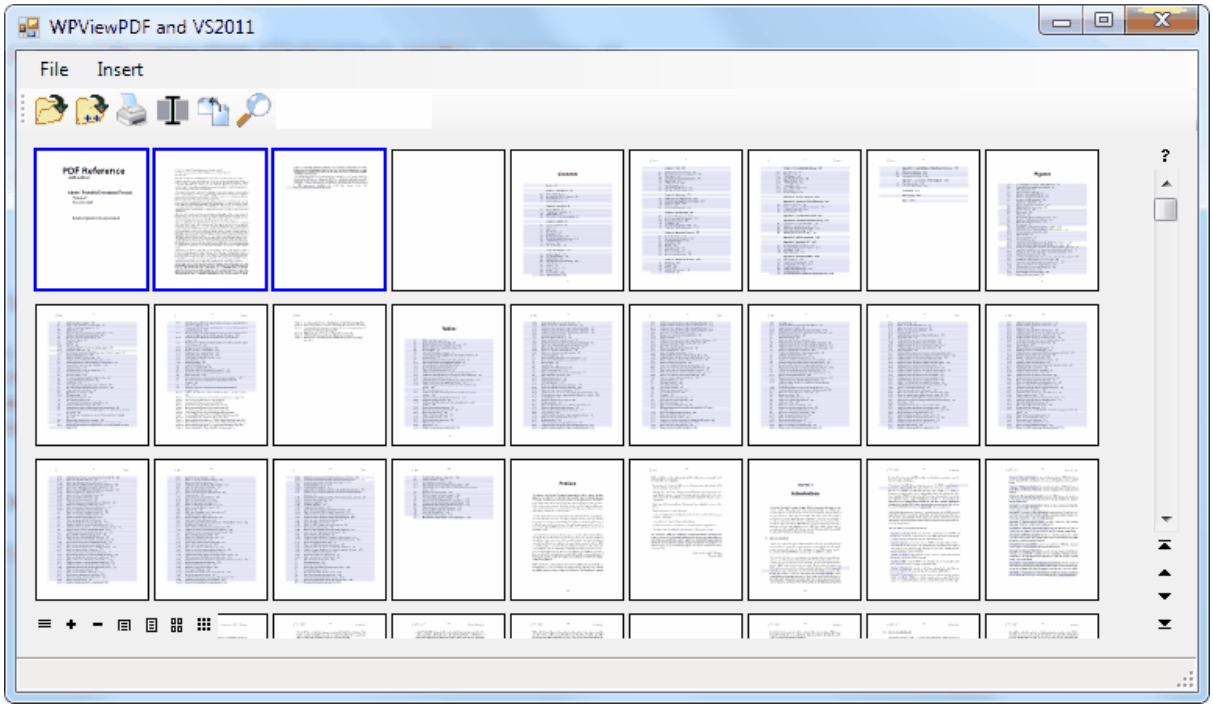
**Read the current ActionMode parameter**

```
function GetActionMode(var aActionParam : String) :  TWPViewpdfActionMode; overload;
```

# 5 Example Projects

## 5.1 .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

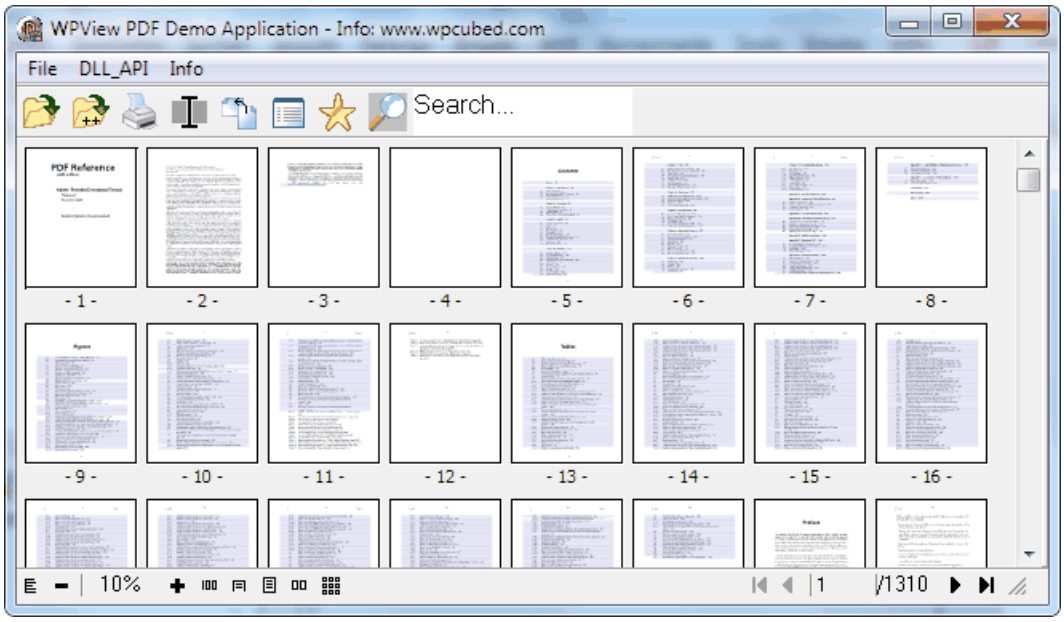
```
        private void RotateBtn_Click(object sender, EventArgs e)
        {
            pdfViewer1.Command(commands.COMPDF_RotatePage, "selected", -90);
        }
```

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);
        }
```

## 5.2 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.

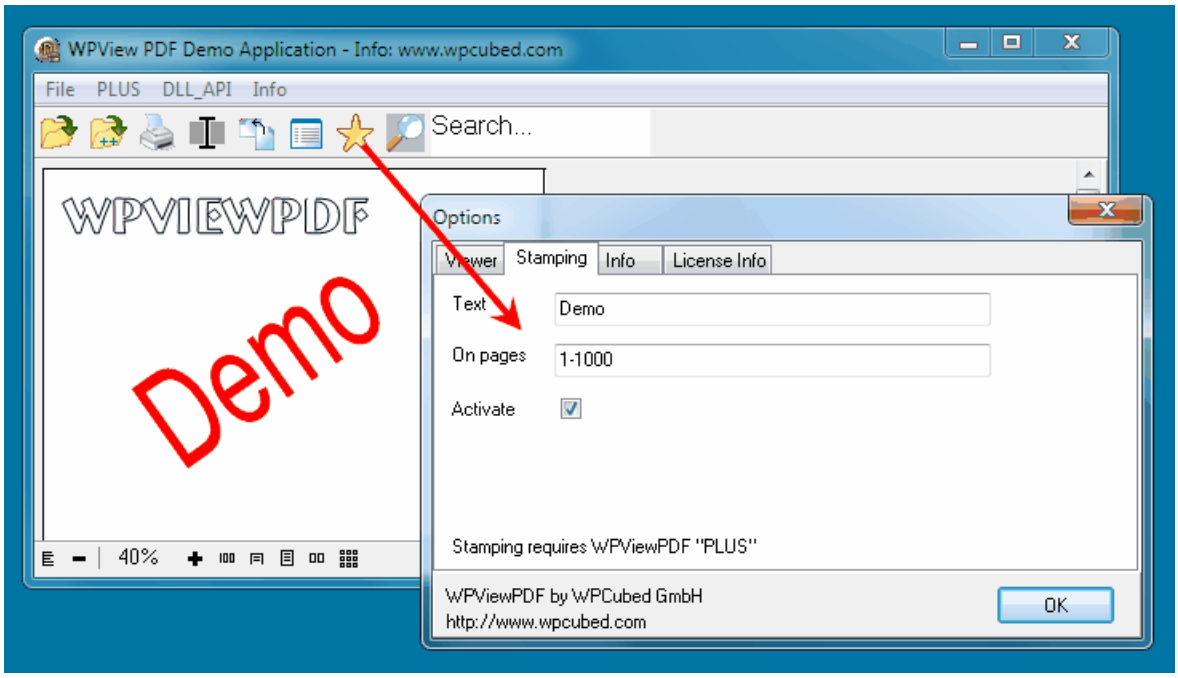
 This buttons open the field property dialog. The fields which are contained in the document will be listed. With WPViewPDF "PLUS" it is also possible to modify the texts!

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

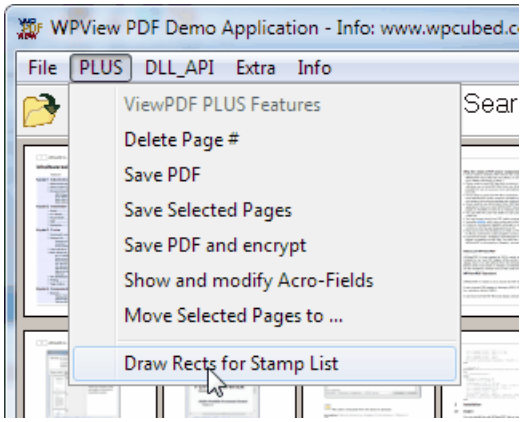
It is also possible to view the PDF document information.

In the property dialog, in case WPViewPDF "Demo" or "PLUS" was used, the

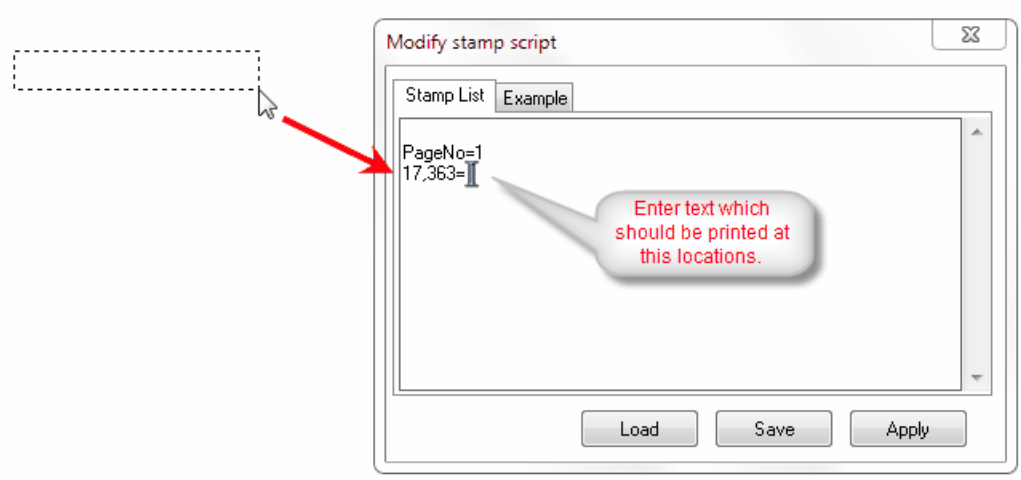
graphical stamping can be utilized. In this simple example just a rotated text is drawn on a metafile canvas. (Please note that currently only simply text and vector drawing is allowed using metafile stamping. Images cannot be used. All text will be converted to vectors)




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.

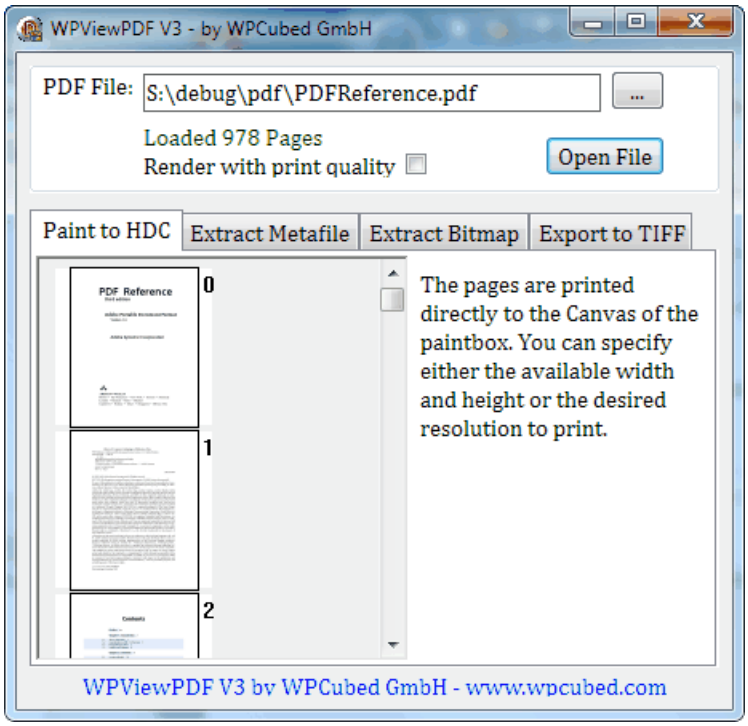



After Apply the pages will be updated at once. When the document is saved, the stamped text will be saved with it.

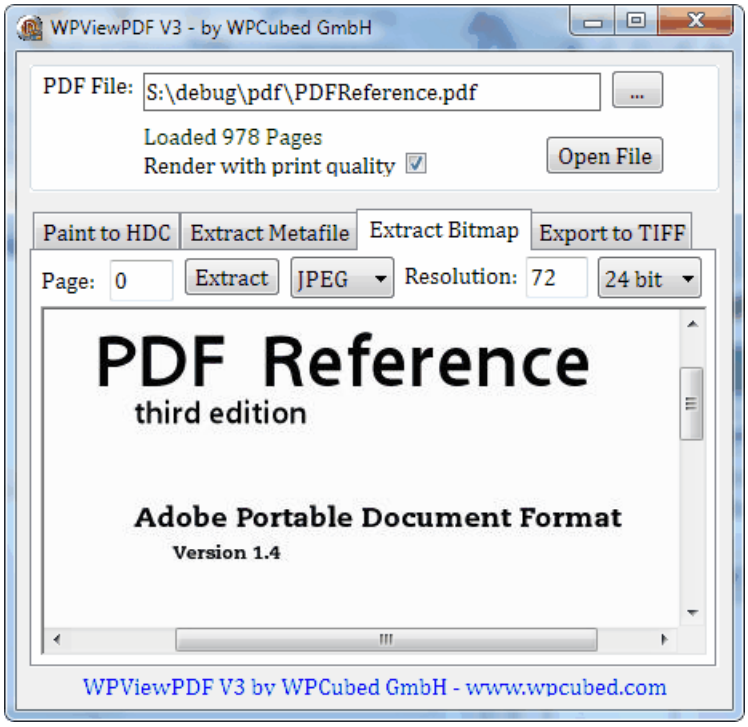
With WPViewPDF PLUS You can also move pages after a certain page ("0" would be the start). To do so select one or more pages (usually with the right mouse button) and click on "Move Selected Pages ..." to enter the number.

## 5.3 Delphi: PDF to Bitmap

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



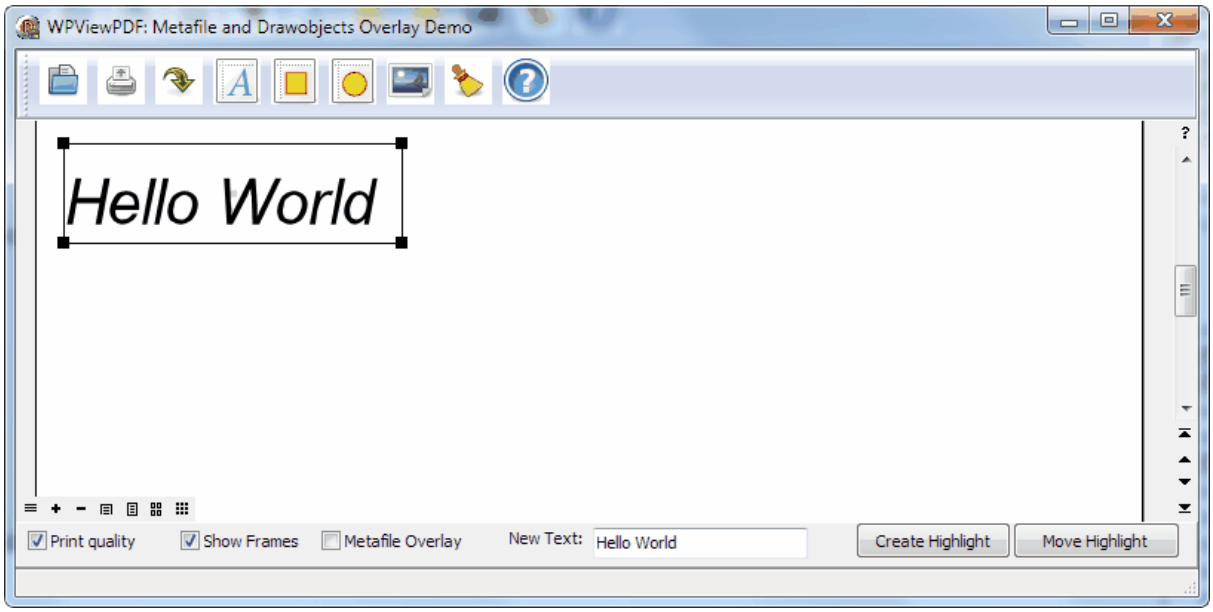
Please select a PDF file first and then click on "Open File" to actually load it. You can test the "print renderer" (default) and the bitmap renderer.




## 5.4 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;
```

# 6 Commands

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

The list of all commands is installed by the setup in pascal language in file WPDF_ViewCommands.PAS.

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

# VB6

When using VisualBasic 6 with the WPViewPDF OCX please use the method **CommandStrEx.** Pass an empty string parameter if no string parameter is expected by a certain command.

# VCL

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_..."

# .NET

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:**

## 6.1 Configuration

**COMPDF_SinglepageMode = 148**

Enables / Disables the single page view for the main viewer. The thumbnail view (if enabled) will always show the whole file. You can use command COMPDF_ZoomThumbnails to change zoom in thumbnail view.

**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.

Possible with WPViewPDF VCL:
Add the conditional **THEMEDWPVIEWPDF** to your project options to use the style service to paint the background of the viewer.
This disables the DeskColor

**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

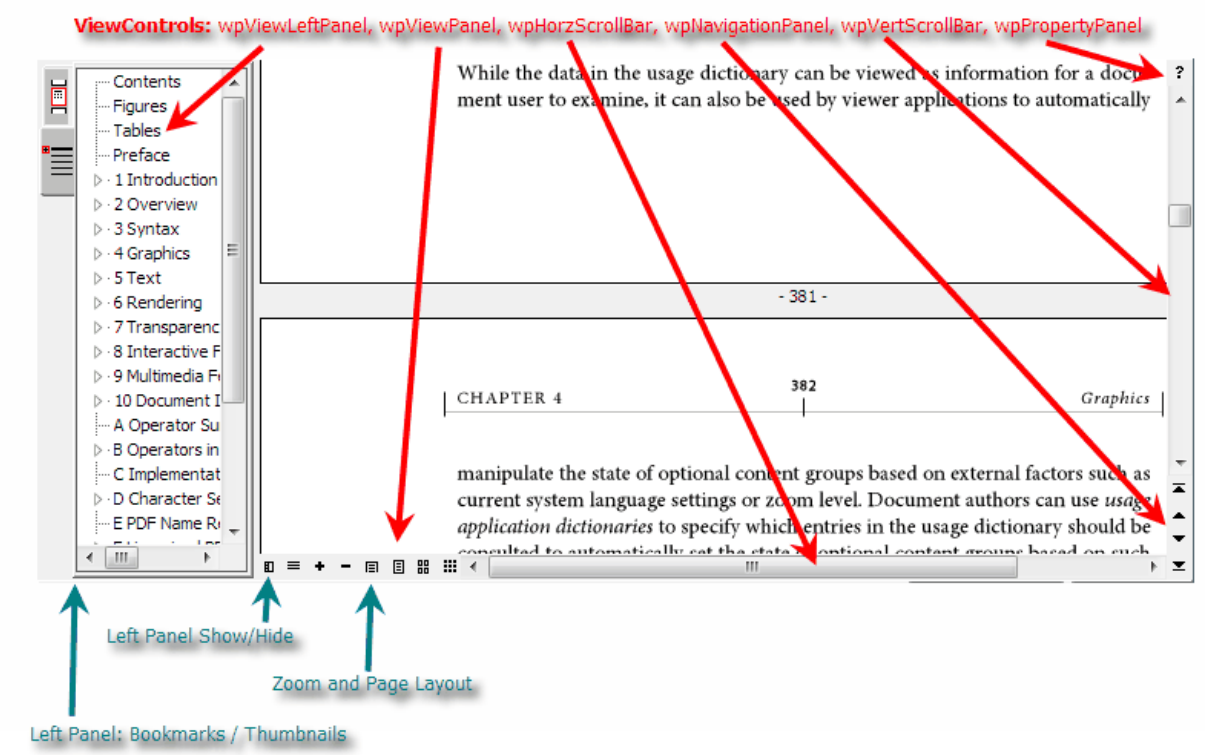
Add 4 to one of the above values and the engine will print unscaled text through regular GDI. This setting must be applied before the PDF is loaded to be effective.

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




wpViewLeftPanel can also be switched on/off with command COMPDF_ShowNavigation.

**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: wpPageSelectionWithKeyboard - activate/deactivate the selection by keyboard

Use Shift + Page up/Down, Home and End keys. Press Ctrl to add to selection. Please also check out the possibilities to change the behaviour of left and mouse button (COMPDF_SelectMode).

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

Mark deleted pages with a cross
8192: wpShowDeletionCross

Don't show a blue rectangle for selected pages in main viewer:
8192*256: wpHidePageSelection

8192*2048: wpHidePageSelectionThumbnails, disable the blue frame in thumbnails

Make the thumbnail view interactive. The user can select pages and reorder them, if also wpSelectPages was used. Also see (COMPDF_SelectMode).
8192*512 : **wpInteractiveThumbnails**

8192*1024: wpThumbnailAtozoomToSquareWH. If used, the thumbnails will be sized to make them fit into the window wether they are rotated or not. This helps to avoid change of zoom when pages are rotated in the thumbnail window.

**COMPDF_SetPageModeDefault = 615:**

Set the page mode for PDF files which do not define the PageMode property.

0=Auto
1=None
2=Outlines
3=Thumbnails

**COMPDF_EnableNavigationAfterLoad = 616:**

0: If the user disabled the left pane it will **not** be reactivated after loading a new file - default
1: The navigation panel will be activated and the the outlines / thumbs as defined with COMPDF_SetPageModeDefault will be displayed
2: The navigation panel will be activated and the the outlines / thumbs as defined in the PDF with PageMode or, if not defined, with COMPDF_SetPageModeDefault will be displayed.

**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.

**COMPDF_ZoomThumbnails = 77**

Pass a value>=10 to set the thumbnail viewer zoom (default 10)
Use -9..9 to increase or decrease the zoom value.

**COMPDF_GetHWND = 1001**

Special method to retrieve the HWND handle of these internal windows:

1: thumbnails, 2: bookmarks, 3: viewer, 4: left panel

If you also pass the string parameter "noresize", it will disable the internal resizing and show/hide logic.

This Example moves the thumbnails to a different window:

```
Panel1.Visible := true;
FThumbHandle := HWND(  WPViewPDF1.command(COMPDF_GetHWND,1) );
if FThumbHandle<>0 then
begin
   SendMessage( WPViewPDF1.CommandStrEx(COMPDF_GetHWND,'noresize',4), WM_SIZE,200,0);
   Windows.SetParent(FThumbHandle, Panel1.Handle );
```

```
    Panel1Resize(nil);
  end;
```

Use this OnResize handler for the parent panel:

```
procedure TForm1.Panel1Resize(Sender: TObject);
begin
  if FThumbHandle<>0 then
  begin
    SendMessage( FThumbHandle, WM_SIZE, 200, (Panel1.Width-2) or ((Panel1.Height-2) shl 16));
    SendMessage( FThumbHandle, WM_MOVE, 200, (1) or ((1) shl 16));
  end;
end;
```

**Control Page and Page content caching:**

COMPDF_SetMaxCachePixels = 1295;

Set the maximum size of cache bitmap pixels for the viewer (default = 30 MegaPixels)

COMPDF_SetMaxCachePixelsThumbs = 1296

Set the maximum size of cache bitmap pixels for the thumbnails (default = 5 MegaPixels)

COMPDF_SetMaxCachePathLockTime = 1297

Set the maximum count of milliseconds to cache page contents. Use 0 to infinitely cache such contents. This may be useful if interaction with the text is required.

**Initialize the JBIG2 decoding**

It is usually **not** required to call the command COMPDF_SetJBIG2Tool when the converter DLLs have been copied to the EXE directory.
The Command COMPDF_SetJBIG2Tool with an integer parameter 1 and the path as string parameter however can be used to manually load the decoding DLL.
It will return 1 if the DLL was loaded, 0 if not.

## 6.2 Select Pages

WPViewPDF allows to to select pages by mouseclick or through code.

The interactive selection is enabled by **COMPDF_SelectMode**.

To select pages by code and to work with selections this commands can be used:

**COMPDF_PageSelectionCount = 246**

Returns the count of selected pages.

**COMPDF_PageSelectionGet = 247**

Checks if a page is selected. Pass zero based page number. Result = 1, if selected.

**COMPDF_PageSelectionClear = 248**

Removes previous selection

**COMPDF_PageSelectionAdd = 249**

Select one additional page. Pass zero based page number. Result = count of selected pages. (Passs -1 to return Count)

**COMPDF_PageSelectionDel = 250**

Removes Selection from Page. Pass zero based page number.
Result = count of selected pages.

**COMPDF_PageSelectionToggle= 258**

Toggles state for a certain page. Pass zero based page number. Result = count of selected pages.

**COMPDF_PageSelectionInvert= 251**

Inverts Selection. Result = count of selected pages.

**COMPDF_PageSelectionSaveRestore = 259**

Saves the selection markers to a buffer and clears the selection.
With Param=1 it restores the buffer.

**COMPDF_SYNC_CURRENT_AS_SELECTED = 1302**

Synchronizes the current and the selected page. The current page should always also be selected and updated while scrolling the document.

**COMPDF_BEGIN_SELECTION = 1300**
**COMPDF_END_SELECTION = 1301**

COMPDF_BEGIN_SELECTION locks the SelectionChange event and the screen

update until COMPDF_END_SELECTION is called.

The event OnViewerMessage (equal to window message WM_PDF_EVENT) is called when the selection is changed. The id of the selection change messe is MSGPDF_CHANGESELPAGE=108

Also note the ViewOption: wpHidePageSelection

## 6.3 Change the way the mouse works

### COMPDF_SelectMode = 133

a) Changes the way the **left mouse** button works in main viewer. 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. This can be also used to zoom to a rectangle (with Command (COMPDF_ZOOM, "RECT") or to capture a frame as bitmap or to copy text (see COMPDF_CopyToClibrd) on clipboard.

**Example:**

```
procedure TWPViewPDFDemo.DoSelRectEvent(Sender: TObject; const PageNr: Int
  R: TRect);
begin
  if FSelectZoomRect then
  begin
     Screen.Cursor := crDefault;
     WPViewPDF1.CommandStr(COMPDF_ZOOM, 'RECT');
  end else
   // This code is used to capture a bitmap
  if FSaveToClip then
  begin
     if WPViewPDF1.CommandEx(COMPDF_SaveBMPToClipboard, 200)>0 then  // Sa
        ShowMessage('An image @200 dpi was copied to clipboard.');
```

```
      end else
       // This code is used to capture as text
      if FCopyTextRect then
      begin
         if WPViewPDF1.CommandEx(COMPDF_CopyToClibrd,8)>0 then
             ShowMessage('Text copied to clipboard.');
      end;
    end;
```

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.

wpmouse_UserDefined = 8, the mouse down event is now triggered

If the value is >90, the current value is returned.

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

c) To change the way the left mouse button works in the **thumbnail view (left panel)**, add 200 to the above values.

d) To change the way the **right mouse** button works in the **thumbnail view**, add 300 to the above values.

To disable the internal use of the middle mouse button (=autoscroll) call this command:

WPViewPDF1.Command(COMPDF_RefineMouseMode, '0', 1);

If used, a click to zoom can be implemented:

```
procedure TWPViewPDFDemo.WPViewMouseDown(Sender: TObject; Button: TMouseButtor
  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;
```

**The VCL defines this function to get and set the mouse mode:**

```
function TWPViewPDF.MouseMode(element : TWPMouseModeElement; value : TWPMouseM
var i : Integer;
begin
  if value=wpmInvalid then
        i := Command(COMPDF_SelectMode,  90+(100*Integer(element)))
  else  i := Command(COMPDF_SelectMode,  Integer(value)+(100*Integer(element))
  if (i>=0) and (i<=Integer(High(TWPMouseMode))) then
      Result := TWPMouseMode(i)
  else Result := wpmInvalid;
end;
```

## 6.4 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.

## 6.5 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 / position

If bit 1 is set in Param =1 it scrolls screen height wise
If bit 2 is set, it will go to previously logged position (same as BACKSPACE key)
If bit 3 is set, the result will be > 0 if there is a logged position

**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 named destination.
Result=PageNumber or -1 if not found

You can specify an integer parameter:
1: The command will return a string list with the names of all named destinations in the PDF
2: The command will jump to the bookmark (akn outline) with the given text.

```
procedure TForm1.Gotoabookmark1Click(Sender: TObject);
var i : Integer;
    s : String;
begin
  s := '';
  if (pdf<>nil) and InputQuery('Jump', 'Bookmark' , s) then
  begin
     i := pdf.command(COMPDF_GotoNamedDest, s, 2); // Try the name of an outline
     if i<0 then
     begin
        i := pdf.command(COMPDF_GotoNamedDest, s, 0); // try a named destination
        if i<0 then ShowMessage(s +#10 + 'was not found and outline ore named destination.');
     end;
  end;
end;
```

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

Controls thumbnail window:
 COMPDF_ZoomThumbnails   = 77; // Value>=10 sets the thumbnail zoomsize (default 12), -9..9 increases or decreases the zoom value

**Example:**
**How to implement a zoom tool (zoom to rectangle) - see here...**

## 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.Command(COMPDF_RefineMouseMode, '0', 1);
```

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 of 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;
```

## 6.6 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.

16= Do not scale the pages, except for pages larger that A3. They will be scaled on DinA3 which will be selected in the printer. This avoids problems with A3 printers which allow larger paper to be selected.

**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.

Hint: You can use the command COMPDF_AppendPage =325 with parameter 0 to add an empty page prior to printout in case you use duplex printing and the page count of a PDF file is uneven.

**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_SelectPrintColorMode = 350**

Select the color mode for printing. 0=default, 1=monochrome, 2=color

**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 configuration will **not** be updated, if it is 0 it will if necessary.
If IntParam=3 none of the printer parameter will be updated, except for the page orientation.

**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)

**Useful to create a page list prior to printing:**

**COMPDF_GetPageNumbersInView = 223**

Gets a string result with all the numbers of the pages which are currently displayed (at least partly).
First Page is "1" so the string can be directly displayed to the user.

## 6.7 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.

*Note: Delphi Developers please use Canvas.Lock / CanvasUnlock when using Canvas.Handle.*

If you need to use the **standard GDI** renderer with PrintHDC please use command
**COMPDF_UseGDIPainter** with parameter **-2**. To activate the default GDI+ renderer use parameter -1.

**COMPDF_UseGDIForPrinting = 145**

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

This only changes the printing, not the display!

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.

### 6.7.1 PrintHDC

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;
```

**Example:**

```
  PaintBox1.Canvas.Lock; // REQUIRED!
  try
      WPViewPDF1.PrintHDC( 0, PaintBox1.Canvas.Handle,
                           - PaintBox1.Width, -PaintBox1.Height );
  finally
     PaintBox1.Canvas.Unlock;
  end;
```

**Also see the PDFWorkBench example.**
**It is very useful to render PDF on any HDC.**

### 6.7.2 PrintHDC on TPrinter

This example prints on TPrinter VCL Object.

```
var dia : TPrintDialog;
    i : Integer;
    b : Boolean;
begin
   dia := TPrintDialog.Create(Self);
   try
     dia.MaxPage := pdf.PageCount;
```

```
      dia.ToPage  := pdf.PageCount;
      dia.FromPage:= 1;
      dia.Options := dia.Options - [poSelection] + [poPageNums];
      if dia.Execute then
      begin
         b := false;
         Printer.BeginDoc;
         try
           for I := dia.FromPage to dia.ToPage do
           begin
               if b then Printer.NewPage;
               Printer.Canvas.Lock;
               try
                 SetViewportOrgEx(
                     Printer.Canvas.Handle,
                      - GetDeviceCaps(Printer.Handle,PHYSICALOFFSETX ),
                      - GetDeviceCaps(Printer.Handle,PHYSICALOFFSETY), nil );
                 pdf.PrintHDC(I, Printer.Canvas.Handle,
                   - GetDeviceCaps( Printer.Canvas.Handle,  PHYSICALWIDTH ),
                   - GetDeviceCaps( Printer.Canvas.Handle,  PHYSICALWIDTH) )
               finally
                 Printer.Canvas.Unlock;
               end;
               b := true;
           end;
         finally
           Printer.EndDoc;
         end;
       end;
    finally
      dia.Free;
    end;
end;
```

## 6.8 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;
```

## 6.9 Save PDF, RTF, TXT, HTML and XML

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.

**A PDF viewer has to respect the PDF security, so does our viewing component.**
**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.**

This pascal code saves all pages in the PDF to individual files:

```
WPViewPDF1.CommandEx(COMPDF_DisableSecurityOverride,1);
for cnt := 0 to WPViewPDF1.PageCount-1 do
begin
  WPViewPDF1.SelectPage(0,0);  //Clear
  WPViewPDF1.SelectPage(1,cnt); // add #cnt
  WPViewPDF1.PLUS.SaveSelectionToFile('s:\page' + IntToStr(cnt) + '.pdf');
end;
```

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

**COMPDF_SetSaveMode = 613**

Set the save mode. It is possible to use this bits:

1: Remove the Annots except for Hyperlinks. Ommits "AcroForm"
2: Remove the Hyperlinks
4: Remove the Bookmarks
8: Remove the StructElements
16: Remove Transition Effects
32: remove Page AA Actions
64: remove PDFA flag
128: Delete Extra XML Data

256: Delete extra commands (such as images and DrawObjects)
512: Delete named destinations
1024: Do not create PDF/A Marker
2048: Do always create PDF/A Marker
4096: Do not save modified page sizes
8192: Never write Cropbox parameter
// To save a scaled PDF use: COMPDF_SaveScaledPDFMode !

8192*2 Delete extra commands (such as images and DrawObjects)
8192*4 Delete named destinations
8192*8 DO not create PDF A Marker
8192*16 DO always create PDF A Marker
8192*32 Don't modify page Size
8192*64 **Flatten the PDF on save time**. This requires that proper Appearance streams are present for the Fields.

**COMPDF_GetModified = 515**

Read state of internal "Modified" flag. With IntPar=1000 the modified flag will be also cleared.
The modified flag is set by the page deletion, rotation and page moving commands and actions.

**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.

**COMPDF_GetTextLen=260, COMPDF_GetTextBuf=261**

Get the page text as character buffer as ANSI, RTF, TXT, HTML and XML.

```
bufsize = Command(COMPDF_GetTextLen, format, PageNo);
CommandEx(COMPDF_GetTextBuf, ansi_buffer);
```

(also see: TWPViewPDF.GetPageText Method )

**It is possible to limit the area where the text is extracted by specifying a rectangle, x,y,x1,y1.**

Please note that the values are measured in 72dpi and are not using any rotation which may be applied to the PDF page.

```
COMPDF_GetTextSetOptions   = 272; // Bitfield:

// bit 2=2: xyhtml writes Y position as baseline position. Default = 2 = on

// bit 3=4: activate/deactivate the  GetTextFilterRect

COMPDF_GetTextFilterRectX = 273;

COMPDF_GetTextFilterRectY = 274;

COMPDF_GetTextFilterRectX1 = 275;

COMPDF_GetTextFilterRectY1 = 276;
```

**COMPDF_CopyToClibrd  = 268**;

Copy selected text to clipboard (RTF, ANSI, UNICODE format). Result = length. (same as Ctrl+C)

if Bit 1 is set in IntParam, the complete text and images are copied
if Bit 2 is set in IntParam, the complete text and images on current page are copied
If bit 3 is set, the complete page will be copied as bitmap (high word = resolution, default = screen dpi)
If bit 4 is set, only the text inside the drawn rectangle will be copied. Use with COMPDF_SelectMode, 2

**COMPDF_MODIFIED = 601**

The command is used to retrieve the "*modified*" bit field.

This values are used:
1 Append PDF,
2 Move Page,

4 Delete Page,
8 Add Object or Annotation
16 Move Object or Annotation
32 Object Properties,
64 Delete Object,
128 Delete Acrofield,
256 Edit Tx Annotation (edit, memo)
512 Edit CH Annotation (checkbox)
1024 Add Attachment,
2048 Add XMP Information

You can pass 1 as integer parameter to set deactivate all bits. If the user is able to edit annotations it is recommended to always save the PDF as a new version since there are other ways to modify annotations not covered by *Modified*.

## 6.10 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

## 6.11 Find X,Y Position

**COMPDF_ScreenToClient expects the X coordinate in hi word, the Y coordinate in low word.**
The result is the page page. The PDF X and Y coordinate can be read with COMPDF_GetScreenToClientX and COMPDF_GetScreenToClientY

The function ScreenToPage can be used to calculate the point on a certain PDF page which corresponds to a certain point on screen.

It is implemented like this:

procedure TWPViewPDF.ScreenToPage( X, Y : Integer; Var pdf_x, pdf_y, pdf_page : Integer );
begin

```
  pdf_page := Command(COMPDF_ScreenToClient, X shl 16 or Y);
  pdf_x := Command(COMPDF_GetScreenToClientX);
  pdf_y := Command(COMPDF_GetScreenToClientY);
end;
```

Command COMPDF_ClientToScreenPage sets the page number (0 based) for the next call to COMPDF_ClientToScreenXY. It returns -1 if the page number is not valid.
COMPDF_ClientToScreenXY expects the PDF page X coordinate (in 72 dpi!) in hi word, the Y coordinate in low word.
The Result is the screen coordinate X in high word, the y coordinate in lo word.

The VCL also implements:

```
function TWPViewPDF.PageToScreen( pdf_x, pdf_y, pdf_page : Integer; var X, Y :
Integer ) : Boolean;
var i : Integer;
begin
  Result := Command(COMPDF_ClientToScreenPage, pdf_page)>=0;
  if Result then
  begin
    i := Command(COMPDF_ClientToScreenXY, pdf_x shl 16 or pdf_y);
    x := (i shr 16) and $FFFF;
    y := i and $FFFF;
  end else
  begin
    x := 0;
    y := 0;
  end;
end;
```

## 6.12 Get/Set Bookmarks

Bookmarks or Outlines are optionally displayed in a treeview on the left panel (see Configuration).

WPViewPDF also allows it to extract the bookmark list as XML. To do so, the command **COMPDF_GetBookmarkXML = 351** can be used. It expects a string and an integer parameter.

COMPDF_GetBookmarkXML will always extract the original bookmark structure of the PDF, not the information which was previously set with **COMPDF_SetBookmarkXML**.

The XML structure is very easy. Each branch is using the tag <outline>. If the branch has children, they are enclosed inside of <outline>...</outline>, if not, the tag is closed "<outline/>".

This parameters are used by <outline>:

Title: The displayed Text
Dest: Optional, a named destination
pid: The internal page ID
pnr: The page number (not used if "Dest" was used)
X: The X coordinate, may be 0
Z: The zoomvalue, may be 0
Y: The Y coordinate in 72 dpi. Usually top is 0

**Example:**

```
<?xml version="1.0" encoding="utf-8"?>
<File id="1" name="WPViewPDFV3.pdf">
 <Outline Title="Introduction" pid="7" pnr="4" X="57" Y="85" Z="0">
  <Outline Title="WPViewPDF Standard" pid="9" pnr="6" X="57" Y="269" Z="0"/>
  <Outline Title="WPViewPDF PLUS" pid="9" pnr="6" X="57" Y="425" Z="0"/>
  <Outline Title="Example Projects" pid="10" pnr="7" X="57" Y="85" Z="0">
   <Outline Title=".NET C# Example: PDFViewNET" pid="10" pnr="7" X="57" Y="111" Z="0"/>
   <Outline Title="Delphi: PDFView" pid="12" pnr="9" X="57" Y="462" Z="0"/>
   <Outline Title="Delphi: PDF to Bitmap" pid="16" pnr="13" X="57" Y="85" Z="0"/>
   <Outline Title="Delphi: Add graphics to PDF" pid="17" pnr="14" X="57" Y="430" Z="0"/>
   </Outline>
 </Outline>
<Outline Title="Installation" pid="19" pnr="16" X="57" Y="341" Z="0">
  <Outline Title="Delphi" pid="19" pnr="16" X="57" Y="376" Z="0"/>
  <Outline Title="C++ Builder" pid="19" pnr="16" X="57" Y="657" Z="0"/> ....
```

The following flags can be used in the integer parameter:
1: do not normalize the Y values to Top->Bottom, in PDF normally Y=0 is bottom of page.
2: do not include page numbers (pnr)
4: do not include page IDs (pid)

The extracted XML can be used to display an independent bookmark viewer or -editor.

It is possible to set predefined bookmarks in WPViewPDF with the command

**COMPDF_SetBookmarkXML = 352**

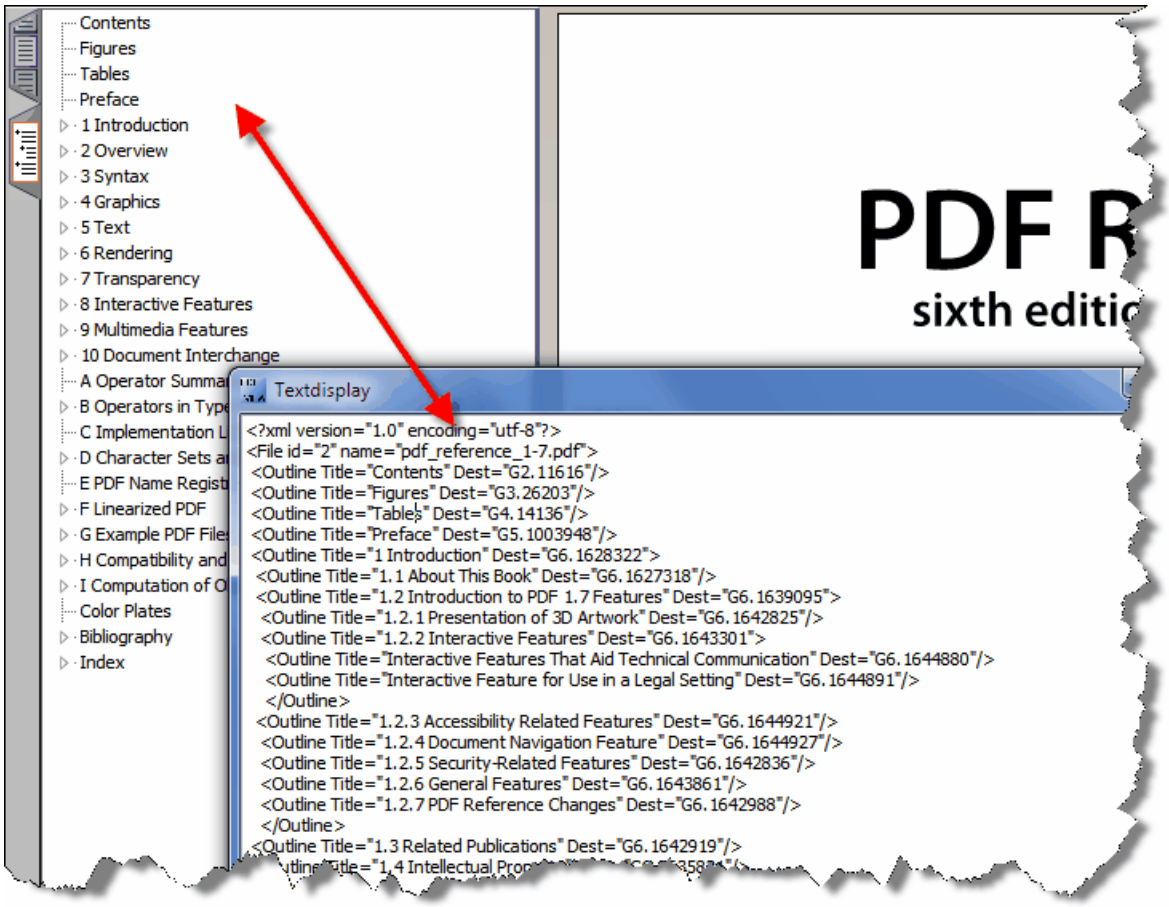
It returns a string.

This flags are understood:
1: Y is measured from bottom to top
2: page numbers (pnr) should be ignored
4: page IDs (pid) should be ignored
512: do not update the internal treeview

When a PDF file is saved, the new structure will be written.
Use COMPDF_SetBookmarkXML with an empty string to clear the tree. This is also required, after a new PDF was loaded, unless the bookmark should persist.

**If you want to implement navigation use command**

**COMPDF_GotoPage = 22** to goto the page Nr in int parameter. The optional string parameter can be "y" or "x,y" or "x,y%z" to specify the zoom value z




## 6.13 Security - Disable Save ...

Please also see Load&Save section in this manual.

The following commands are used to DISABLE functionality.
Once used, the setting cannot be changed anymore!

COMPDF_DisablePrint = 123;
Disable print - it is not possible to enable again!

COMPDF_DisableHQPrint = 124
Disable high quality print - if print, only low quality!

COMPDF_DisableSelectPrinter=125
Disable Select Printer or PrintDialog (print at once!)

COMPDF_DisablePrintHDC = 126
Disable print to HDC - it is not possible to enable again!

COMPDF_DisableSave = 127
Disable saving of the PDF file - it is not possible to enable again!

COMPDF_DisableCopy = 128
Disable copy to clipboard - it is not possible to enable again!

*COMPDF_DisableForms = 129 - reserved in WPViewPDF V3, not yet used.*
*Disable form editing - it is not possible to enable again!*

*COMPDF_DisableEdit = 130 - reserved in WPViewPDF V3, not yet used.*
*Disable editing - it is not possible to enable again!*

COMPDF_DisableSecurityOverride = 131
By standard the viewer respects the P protection flag.
With COMPDF_DisableSecurityOverride you can change this standard behavior at your own responsibility
Bit 1: Ignore the save PDF restriction
Bit 2: Ignore the low quality only printing restriction
Bit 3: Ignore the *no printing at all* restriction

(Note: PDF which use 256 bits AES encryption may never be saved unless the owner password was specified)

Example: The property SecurityOptions is interpreted by this code:

```
FSecurityOptions := x;
if wpDisablePrint in FSecurityOptions then
  command(COMPDF_DisablePrint);
if wpDisableHQPrint in FSecurityOptions then
  command(COMPDF_DisableHQPrint);
if wpDisableSelectPrinter in FSecurityOptions then
  command(COMPDF_DisableSelectPrinter);
if wpDisablePrintHDC in FSecurityOptions then
  command(COMPDF_DisablePrintHDC);
if wpDisableSave in FSecurityOptions then
  command(COMPDF_DisableSave);
if wpDisableCopy in FSecurityOptions then
  command(COMPDF_DisableCopy);
if wpDisableForms in FSecurityOptions then
  command(COMPDF_DisableForms);
if wpDisableEdit in FSecurityOptions then
  command(COMPDF_DisableEdit);

if wpDisablePDFSecurityOverride in FSecurityOptions then i := 1 else i := 0;
if wpDisablePDFSecurityLowQualityPrint in FSecurityOptions then i := i or 2;
if wpDisablePDFSecurityPrint in FSecurityOptions then i := i or 4;
command(COMPDF_DisableSecurityOverride, i);
```

## 6.14 Actions

WPViewPDF V4 implements "Actions".

These are **predefined commands** which combine several other commands. They

can be executed by using an ID but also by specifying the name. The PDF engine can also provide information if a command is active right now, i.e. to highlight a button or if it is disabled.
It is possible to retrieve the name, a caption and also a hint. It is possible to localize the strings using an XML script.

The action IDs consist of the group and the operation id. The group is encoded into the high word of the id, the operation into the low word. The operation 0 in group 0 does nothing, this means id 0 is void.

This command executes an action when you know the action name, such as "FileOpen". A String parameter can be attached to the action name using "=", i.e. "FileOpen=c:\test.pdf"

COMPDF_ACTION = 580;

This command executes an action when you know the action id
  Command(COMPDF_ACTIONNR, 1, "c:\test.pdf") will open a PDF file.

COMPDF_ACTIONNR = 581; // HighWord=Kind, LowWord = Operation, StrParam is passed

This command is not used right now:

COMPDF_ACTIONEX=582; // Extended Action - with param as record (RESERVED)

This command is used to read information about an eaction:

COMPDF_ACTION_READ = 583;

  strparam="xml" - read all strings and commands as XML list!
  strparam="kinds" - read count of kinds
  strparam="kindX" - read count of operations in kind X
  strparam='caption", IntParam = HighWord=Kind, LowWord=Operation. Result can be ''
  strparam='hint", IntParam = HighWord=Kind, LowWord=Operation. Result can be ''
  strparam='command", IntParam = HighWord=Kind, LowWord=Operation. Result can be ''

This command writes the XML definition. Caption and hint can be modified.
If IntParam<>0 it is expected to be the action ID. Then you can use string parameter such as
caption=, hint=, command=.., option=.. to modify a certain action.

 COMPDF_ACTION_WRITE = 584;

This command read action flags. The flags are used to create a menu automatically:
bit 1: need separator after this item
bit 2: Should not display a menu item for this action
bit 3: Requires WPViewPDF PLUS
bit 4: Requires permission to save a PDF file
bit 5: This is a global operation which affects all viewers
bit 6: This action should create a submenu with all following actions until one with bit 1 flag in it.

COMPDF_ACTION_READFLAGS = 585;

Note: The procedure WPPDFViewerInitMainMenu defined in WPViewPDF4.pas uses the flags.

This command read action states: 1=checked, 2=disabled

COMPDF_ACTION_READSTATE = 586

## 6.15 Extract and add Attachments, i.e. ZUGFeRD XML. Read XMP

### Read Metadata

**COMPDF_Get_XMPBuffer = 598**

This command can be used to read the loaded PDF XMP metadata.

The IntParam is used to select the loaded PDF-File to examine. (default=0).

It is also possible to provide the NAME of the PDF-File to check in the StrParam. The Result Value = -2 if IntParam is not valid or the provided PDF filename was not loaded before.

Example:

```
ShowMessage( WPViewPDF1.CommandGetStr(COMPDF_Get_XMPBuffer,'',0)  );
```

### Attachments, i.e. ZUGFeRD XML

With WPViewPDF it is also possible to extract the attachments in a PDF file. This can be useful to extract the invoice data stored using the ZUGFeRD standard.

**COMPDF_Attachment_List = 590**

Get the number of the attachments in the loaded PDF file.
Use the Index in next commands.

**COMPDF_Attachment_GetProp = 591**

Get the name of a certain annotation. StrParam selects which PDF property to read. To read the attachment name pass an empty string parameter.

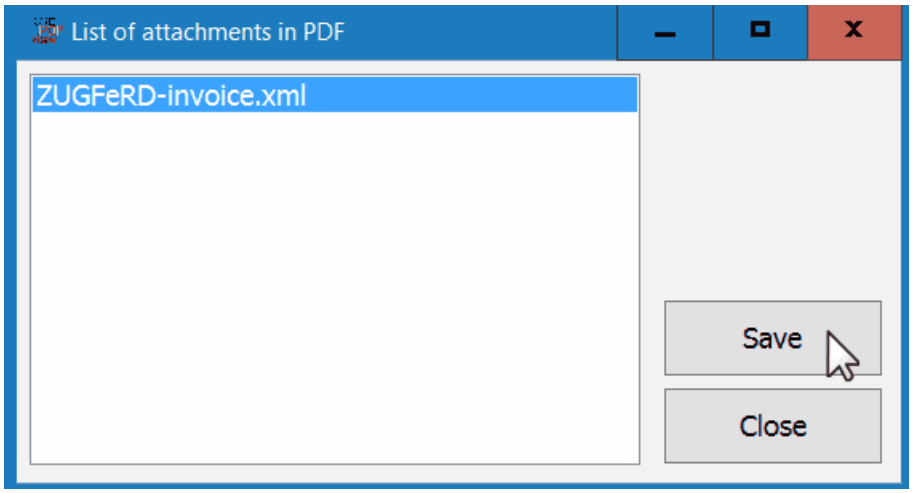
**COMPDF_Attachment_GetData = 592**

Get the attachment data as memory buffer (Result = length).
StrParam can be "F" or "UF"
IntParam = index

Use **COMPDF_MakeGetMEMORY** to read the data.
The IntParam is the index of the attachment in the file.

**Example - Delphi:**

On a form there is a TListbox to show all attachments, a SaveDialog and a button to start the extraction process.




This code updates the listbox with a list of all attachments

```
procedure TWPViewListAttachments.Update;
var i, j : Integer;
begin
   ListBox1.Items.Clear;
   j := WPViewPDF.Command(COMPDF_Attachment_List);
   for i := 0 to j-1 do
     ListBox1.Items.Add( WPViewPDF.CommandGetStr( COMPDF_Attachment_GetProp, '
end;
```

This code extracts the selected data

```
procedure TWPViewListAttachments.btnSaveClick(Sender: TObject);
var mem : TMemoryStream;
    l : Integer;
begin
   if ListBox1.ItemIndex>=0 then
   try
      mem := TMemoryStream.Create;
      l := WPViewPDF.Command( COMPDF_Attachment_GetData, ListBox1.ItemIndex );
      if l>=0 then
      begin
        mem.SetSize(l);
        WPViewPDF.CommandEx( COMPDF_MakeGetMEMORY, {$IFDEF WIN64} IntPtr {$ELS

        SaveDialog1.FileName := WPViewPDF.CommandGetStr(COMPDF_Attachment_GetP

        if SaveDialog1.Execute then
        begin
           mem.SaveToFile(SaveDialog1.FileName);
        end;
      end;
   finally
      mem.Free;
   end;
end;
```

### Example C#

This example uses a sub menu which is filled with all attachments in a PDF file. If the user clicks on one item, the command COMPDF_Attachment_GetData is used to extract the data. The data is retrieved from the viewer control with the function GetMemory which was implmented in the wrapper class.

This method fill the sub menu:

```
        private void infoToolStripMenuItem_DropDownOpening(object sender, EventArgs
e)
        {
            menFileattachment.DropDownItems.Clear();
            int j = pdfViewer1.Command(commands.COMPDF_Attachment_List);
            for(int i = 0; i<j;i++)
            {
                System.Windows.Forms.ToolStripMenuItem men = new System.Windows.
Forms.ToolStripMenuItem();
                men.Text = pdfViewer1.CommandGetStr( commands.
COMPDF_Attachment_GetProp, "", i );
                men.Tag = i;
                men.Click += new System.EventHandler(OnClickAttachment);
                menFileattachment.DropDownItems.Add(men);
            }
            if(j<=0) menFileattachment.DropDownItems.Add("<empty>").Enabled = false
;
```

```
        }
```

This method handles a click:

```
        private void OnClickAttachment(object sender, EventArgs e)
        {
            System.Windows.Forms.ToolStripMenuItem men = sender as System.Windows.
Forms.ToolStripMenuItem;
            int l = pdfViewer1.Command(commands.COMPDF_Attachment_GetData, (int)
men.Tag);
            if (l > 0)
            {
                byte[] databytes = pdfViewer1.GetMemory();
                saveFileDialog2.FileName = men.Text;
                if ( saveFileDialog2.ShowDialog()==System.Windows.Forms.
DialogResult.OK )
                {
                    System.IO.FileStream file = new System.IO.FileStream(
                           saveFileDialog2.FileName,  System.IO.FileMode.Create );
                    file.Write(databytes,0,databytes.Length);
                    file.Close();
                }
            }
        }
```

Info: This is how GetMemory was implemented:

```
        public byte[] GetMemory()
        {
            int l = Command(commands.COMPDF_MakeGetMEMORY, null);
            IntPtr buffer = Marshal.AllocCoTaskMem(l + 16);
            byte[] databytes = new byte[l];
            Command(commands.COMPDF_MakeGetMEMORY, buffer);
            Marshal.Copy(buffer, databytes, 0, l);
            Marshal.FreeCoTaskMem(buffer);
            return databytes;
        }
```

**COMPDF_Attachment_AddAF = 593**

This command adds a structure to hold and embedded file. Result value is the index or -1 to be used in next commands.
The Index is different to the one used with COMPDF_Attachment_List! The added attachment will be listed by COMPDF_Attachment_List.

**COMPDF_Attachment_SetProp = 594**

Set the name of a certain annotation. StrParam selects which property to modify. Currently only "Desc" is supported.

**COMPDF_Attachment_SetData = 595**

Set the attachment data. The attachment is loaded from the file with the name strparam. The time stamp is also used.
As second parameter the index value returned by **COMPDF_Attachment_AddAF** must be used.

**Example:**

```
procedure TForm1.AddAttachmentClick(Sender: TObject);
var id : Integer;
begin
  with TOpenDialog.Create(Self) do
  try
    Filter := '*.*';
    if Execute then
    begin
      // Create a slot
      id := pdf.CommandStr(COMPDF_Attachment_AddAF, ExtractFileName(Fil
      if id<0 then  ShowMessage('The file was not attached') else
      begin
        // Load the embedded Data into the object with the given index
        pdf.CommandStrEx(COMPDF_Attachment_SetData, Filename, id);
        // Set additional property (F already has been set)
        pdf.CommandStrEx(COMPDF_Attachment_SetProp, 'Desc=File was atta
      end;
    end;
  finally
    Free;
  end;
end;
```

# 7 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.

A call to PDFView.Plus.Enable is not required anymore to activate the PLUS license,

it can, however, be used to check wether a PDF file contains security measures which forbid saving.

With **SetGlobalParameter("DisableThreading=1")** multithreading can be disabled.
If highest possible stability is required, we recommend this setting.

**Instead of SetGlobalParameter(stringvalue) it is also possible to call Command(200000, stringvalue)**

## 7.1 Methods

### 7.1.1 TWPViewPDF.AddDrawObject

Add graphical objects. Uses the TPDFDrawObjectRec structure and the additional data to prepare a TPDFDrawObjectRec parameter record which is passed to the engine.

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


procedure AddDrawObject(Mode: TWPAddDrawObjectMode;
     Name: WideString;
     var Param: TPDFDrawObjectRec;
     data: TMemoryStream = nil;
     StrParam: WideString = ''); overload;
```

**Also see: AddHighlightRect which can be used in combination with FindText.**

Parameters:

**var Param: TPDFDrawObjectRec;**

```
type TPDFDrawObjectRec=
  packed record
    PageNo     : Integer;  // Page number to place the object
    x,y,w,h    : Integer;  // 72 dpi, for grtyp>0 ist 720 dpi
    ColorBrush : Cardinal; // Background Color
    grtyp      : Integer;  // Graphic type. (Ignore all other props if 0 = V2 compatibility mode)
    // 0=default highlight (alpha=120)
    // 1=rectangle
    // 2=circle
    // 3=ellipse
    // 20= Image
    // 100= Text
    structsize : Integer;  // Size of this structure.
```

```
  typparam  : Integer;   // Extra Parameters for this type
  // For Images it is the Image ID
  units_xywh: Integer;   // 0 and 1=72 dpi, divisor
  // -------
  Alpha     : Integer;   // 0 or Alpha in range 1..255.
  // Use ColorBrush=clNone (=$1FFFFFFF) to draw transparently
  Angle     : Integer;   // 0..360
  ColorPen  : Cardinal;  // Line Color
  ColorText : Cardinal;  // Textcolor, forground
  FontSize  : Integer;   // For Texts it is the font size in point multiplied by 100
  PenWidth  : Integer;   // Line Width in pt * 1000, 0 does not paint a line
  ObjectOptions : Integer;   // Option Bitfield
  // 1  : Keep AspectRatio
  // 2  : Stretch (used for text)
  // 4  : Center horizontally (text in the box)
  // 8  : Used for Text and JPEG. Draw Background in selected Brush Color and Pen
  // 16 : Apply Brush Color after painting the object
  // 32 : prohibit moving the object
  // 64 : prohibit changing size of object

  Padding : Integer;     // Padding inside - using 720dpi
  HRad, VRad : Integer;  // Horizontal, Vertical Radius - using 720dpi
  PenStyle   : Integer;  // 4 bytes to define a stroking pattern (reserved)
  CreateOptions : Integer; // How should the object be created  - Bitfield
  // 1  : Place the object UNDER the Page
  // 2  : Place at the Right  Border of the page  (ignore X)
  // 4  : Place at the Bottom Border of the page  (ignore Y)
  // 8  : Scale the object to the page horizontally (uses X as right and left margin)
  // 16 : Scale the object to the page vertically (uses Y as right and left margin)
  // 32 : Create the object and select it (clear selection)
  // 64 : Create the object and add it to the selction (do not clear selection)
  // ...
  // 8192*2 : Do NOT refresh the screen
  // ----
  // Offsets to BLOB Data
  Fields      : Cardinal; // Select fields for the "Get" and "Set" commands
  // 1 : PageNo  (move to a different page!)
  // 2 : X
  // 4 : Y
  // 8 : W
  // 16 : H
  // ....   OBJFL_....
  textoff : Integer;     // Offset to the wide char text data (should follow the predefined data)
  textlen : Integer;     // length of Text
  NameOff   : Integer;   // Offset to name (widechar)
  NameLen   : Integer;   // Length of name
  DataOff   : Integer;   // Offset to the object data. (should follow the predefined data)
  Datalen   : Integer;   // Length of the data.
  DataTyp   : Integer;   // Certain flags to tell what to do with the data
  // 1 = ANSI Text
  // 2 = Unicode Text
  // 3 = JPEG Data ... (reserved ..)
  // ... now text and data can follow. Offsets are measured from start of structure.
end;
```

### StrParam: WideString

This parameter can contain various additional properties separated by comma:

"font=x" - select the font x
"size=x" - select the font size x / 100

"text=x" - set the text x
"color=x" - select the color name x
"background=x" - select the background color x
"stretch=x" - if x=1 then stretch the font, otherwise don't stretch

**Examples:**

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.

**Add a text object:**

```
var
   t: TPDFDrawObjectRec;
```

```
begin
      FillChar(t, SizeOf(t), 0);
      t.PageNo := 0; // First Page
      t.units_xywh := 10; // 720 dpi
      //
      t.grtyp := 100; //
      t.x := Round( 3.2 * 720); // 3.2 inches
      t.y := Round( 1.3 * 720); // 1.3 inch down
      t.w := Round( label1.Width/96 * 720);
      t.h := Round( label1.Height/96 * 720);
      WPViewPDF1.AddDrawObject(wpAddNow, '', t,
        'This is a sample text',
        '"Font=Arial","size=1100"');
end
```

### 7.1.2 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.

### 7.1.3 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.

### 7.1.4 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)

### 7.1.5 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'.

### 7.1.6 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 ...

### 7.1.7 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).

### 7.1.8 TWPViewPDF.EndPrint Method

**Declaration**
```
procedure EndPrint;
```

**Description**
Closes a print job started with BeginPrint.

### 7.1.9 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;
```

It is also possible to create an highlight annotation which *covers* the found text. (It is not possible to delete the text)

This requires WPViewPDF 4 PLUS.

```
procedure TForm1.FindtextAndAddHighlightAnnotClick(Sender: TObject);
var s : string;
    b : Boolean;
    page, x,y,w,h : Integer;
begin
   s := '';
  b := false;
  if (pdf<>nil) and InputQuery('Red Text', '', s) then
  begin
    try
       pdf.command(COMPDF_BEGINUPDATE);

       while pdf.FindText(s, false, b, true)>=0 do
       begin
          b := true;
          page := pdf.command(COMPDF_FindGetXYWH, 10);
          if page>=0 then
          begin
              x := pdf.command(COMPDF_FindGetXYWH, 11);
              y := pdf.command(COMPDF_FindGetXYWH, 12);
```

```
                w := pdf.command(COMPDF_FindGetXYWH, 13);
                h := pdf.command(COMPDF_FindGetXYWH, 14);

                pdf.AddHighlightRect(page, x,y,w,h, clRed, [wpAsAnnot,wpAnnotAtP
            end
            else break;
         end;
      finally
         pdf.command(COMPDF_ENDUPDATE, 2);
      end;
   end;
end;
```

### 7.1.10 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 have to specify the page number as a value between 0 and PageCount-1.

We do not recommend this feature to create new PDF files. Instead the original PDF file should be saved.

Note: Use SetGlobalParameter('rotatemeta=1') if you need to extract metafiles with applied page rotation.
By default the rotation is ignored.

Alternatively Command(200000, 'rotatemeta=1') can be called. But please note, this setting is valid for all viewers.

**Note:** PageNo is 0 based.

### 7.1.11 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 have to specify the page number as a value between 0 and PageCount-1.

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

We do not recommend this feature to create new PDF files. Instead the original PDF file should be saved.

Note: Use SetGlobalParameter('rotatemeta=1') if you need to extract metafiles with applied page rotation.
By default the rotation is ignored.

Alternatively Command(200000, 'rotatemeta=1') can be called. But please note, this setting is valid for all viewers.

**Note:** PageNo is 0 based.

### 7.1.12 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.

*PageNo* is 0 based.

You can specify the *format* You need:

**"ANSI" - Ansitext**

**"HTML" - HTML with CSS styles**

**"XYHTM" or "XYHTML" - HTML with CSS styles** - each characters will be placed directly using absolute CSS positions. Pages are separated by <page/> - since this not supported by HTML reader, it is best to export the text page by page - otherwise you will see overprinted text.

Use command COMPDF_GetTextSetOptions (=272) to modify the extracted text:

Set bit 2 if you need y position written as text baseline position (=default) or clear bit 2 to save the top position

**"XML" - simplified XML code**. The text is exported encoded into UTF8 format. <?xml...> tags are suppressed, to make it easier to append the text.

The following tags are used:

<table>, <tr>, <td> are used to separate tables from the text

<page units="pt" n="pagenumber 0..x" w="width" h="height"> </page> encloses one page

<text ff="fontface" fs="fontsize" x="xpos" y="ypos" fc="fontcolor"> </text> encloses text

i="1" will be written for italic text, b="1" will be used for bold text.

The engine will combine consecutive characters into one <text> tag encoded to UTF8.

**"RTF" - RTF code**

The method is implemented like this:

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

**It is possible to limit the area where the text is extracted by specifying a rectangle, x,y,x1,y1.**

Please note that the values are measured in 72dpi and are not using any rotation which may be applied to the PDF page.

pdf.command(COMPDF_GetTextSetOptions, **4**+2); // Activate the filter

pdf.command(COMPDF_GetTextFilterRectX, x ); // Left and Top Values

pdf.command(COMPDF_GetTextFilterRectY, y );

pdf.command(COMPDF_GetTextFilterRectX1, x1 ); // Right and Bottom values

pdf.command(COMPDF_GetTextFilterRecty1, y1 );

Don't forget to call pdf.command(COMPDF_GetTextSetOptions, 2) to deactivate the filter when you are done.

**Example:**

```
// event handler for OnSelRect
procedure TForm1.SelRectEventToDrawrectangleandextracttext(Sender: TObject; co
var WPViewPDF : TWPViewPDF;
    s : string;
begin
    WPViewPDF:= (Sender as TWPViewPDF);
    WPViewPDF.OnSelRectEvent := nil;
    try
        WPViewPDF.command(COMPDF_GetTextSetOptions, 4+2); // Activate the filt
        WPViewPDF.command(COMPDF_GetTextFilterRectX , r.Left );
        WPViewPDF.command(COMPDF_GetTextFilterRectY , r.Top );
        WPViewPDF.command(COMPDF_GetTextFilterRectX1, r.Right );
        WPViewPDF.command(COMPDF_GetTextFilterRecty1, r.Bottom );
```

```
        s := WPViewPDF.GetPageText(PageNr);
        // WPViewPDF.AddHighlightRect(0, r.Left, r.Top, r.Width, r.Height , 25
        ShowMessage( s );   // 1...
    finally
        WPViewPDF.command(COMPDF_GetTextSetOptions, 2); // DE-Activate the fil
    end;
end;
```

// on button click ....

```
procedure TForm1.Drawrectangleandextracttext_Click(Sender: TObject);
begin
  if pdf<>nil then
  begin
    pdf.OnSelRectEvent := SelRectEventToDrawrectangleandextracttext;
    pdf.CommandEx(COMPDF_SelectMode, 2); // let the user draw ...
  end;
end;
```

### 7.1.13 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.

Also note the possibility to specify a rectangle for the extraction

### 7.1.14 TWPViewPDF.LoadFromFile Method

**Declaration**

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

**Description**
This function opened a PDF file to be displayed in the PDF viewer.

The PDF files remains open until the viewer is closed or cleared since it will load data from the PDF files when required.

### 7.1.15 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.

### 7.1.16 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.

### 7.1.17 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.

### 7.1.18 TWPViewPDF.UnDeletePage Method

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

**Description**
Reverts the change done by DeletePage.

**Note: N** is 0 based.

### 7.1.19 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.

### 7.1.20 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

Filename is either the base filename, or "Memory" can be used to create the bitmap data inside of a memory stream. "Clipboard" 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.

### 7.1.21 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.

### 7.1.22 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.

## 7.2 TIEWPCubedPDF

**TIEWPCubedPDF = class**

This class has been provided as interface to the WPViewPDF DLL to load PDF files as if they were image files with *ImagEn*.

**ImagEn** is the powerful imaging library for Delphi - please visit www.xequte.com for additional information.

To use this class you need to included the unit wpcubed_pdf_plugin.pas.

Usually wpcubed_pdf_plugin.pas registeres the PDF support but it is possible to deactivate the auto-registration in wpcubed_pdf_plugin.pas using a compiler symbol. In that case please add this code to your application:

```
if TIEWPCubedPDF.Initialize then
    TIEWPCubedPDF.RegisterPlugin
 else ShowMessage('PDF decoder DLL could not be found');
```

WPViewPDF will create a bitmap from a certain PDF page. The size will be either fit into the size provided using the variables (IOParams.Dict)
"PDF:DesiredHeight" and "PDF:DesiredWdith" (both must be provided!) or using the dpi setting provided by "PDF:Density".

If no variables were used, the setting in fDpiX will be used.

You can download a demo of WPViewPDF at http://www.wpcubed.com.

To add PDF support to ImageEn you only need the edition WPViewPDF *MakeImage* however all other editions also support the required API.

# 8 PDFWorkbench

The PDFWorkBench is an object which can be created using some functions which can be imported from the WPViewPDF DLL.

By a simple call to 'pdfWorkbenchCreate' a new workbech object is created. It will be freed by 'pdfWorkbenchFree'.
Using 'pdfWorkbenchLoad' a PDF file can be loaded or appended.

With 'pdfWorkbenchCommand' most commands IDs used by WPViewPDF can be sent to the work bench object.

This are the definitions of the functions:

```
fktpdfWorkbenchCreate = function(
```

```
      licname, lickey: PWideChar; liccode: Cardinal ) : Pointer; stdcall;

fktpdfWorkbenchFree = function(
      workbench : Pointer ) :  Integer; stdcall;

fktpdfWorkbenchLoad = function(
      workbench : Pointer;
     Filename : PWideChar;
      Append, InMemory : Integer  ) : Integer; stdcall;

fktpdfWorkbenchCommand = function(
      workbench : Pointer;
      commandid : Integer;
      intpar : Integer;
      strpar : PWideChar;
      ptr : Pointer ) : Integer; stdcall;
```

Under Delphi the function pointer are loaded by WPViewPDF3.pas:

```
wpview_pdfWorkbenchCreate := GetProcAddress(WPViewPDFDLLHandle,'pdfWorkbenchCr
wpview_pdfWorkbenchFree := GetProcAddress(WPViewPDFDLLHandle,'pdfWorkbenchFree
wpview_pdfWorkbenchLoad := GetProcAddress(WPViewPDFDLLHandle,'pdfWorkbenchLoad
wpview_pdfWorkbenchCommand := GetProcAddress(WPViewPDFDLLHandle,'pdfWorkbenchC
```

## 8.1 Example: Read page count

In this example the workbech is created "on the fly" to just read out the page count.

```
procedure TForm1.TestnonvisualPDFworkbench1Click(Sender: TObject);
var workbench : Pointer;
    s : String;
    function CommandGetString(
              id : Integer;
              strparam : String = '';
              intparam : Integer = 0 ) : String;
    var
      i: Integer;
    begin
      i := wpview_pdfWorkbenchCommand(workbench,id, intparam, PWideChar(strpar
      if i > 0 then
      begin
        SetLength(Result, i);
        wpview_pdfWorkbenchCommand(workbench, COMPDF_GetTextBufW, 0, nil, (@Re
      end
      else
        Result := '';
    end;
```

```
begin
  if OpenDialog1.Execute then
  begin
   if not assigned(wpview_pdfWorkbenchCreate) then
     raise Exception.Create('wpview_pdfWorkbenchCreate is not available');
   workbench := wpview_pdfWorkbenchCreate(PWideChar(LicName), PWideChar(LicKey
   if not assigned(workbench) then
     raise Exception.Create('wpview_pdfWorkbenchCreate failed');
   try
     s := OpenDialog1.FileName;
     if wpview_pdfWorkbenchLoad( workbench, PWideChar(s), 0 , 0 )<0 then
         raise Exception.Create('wpview_pdfWorkbenchLoad failed');
     ShowMessage(
       Format('This PDF File has %d pages. DLL-Version = %s', [
        wpview_pdfWorkbenchCommand( workbench, COMPDF_GetPageCount, 0, nil, ni
        CommandGetString(COMPDF_GET_DLLVERSION)
         ] )
        );
   finally
     wpview_pdfWorkbenchFree( workbench );
   end;
  end;
end;
```

## 8.2 Create a reusable work-bench in a dialog (TForm)

The form requires a pointer variable "workbench"

uses
  ... WPViewPDF3,  WPDF_ViewCommands;

{$I PDFLicense.INC}

```
var
    workbench : Pointer;
    PageCount : Integer;
```

In OnCreate it is initialized

```
procedure TPDFRenderDemo.FormCreate(Sender: TObject);
begin
  WPViewPDFLoadDLL( 'wPDFViewDemo04' );
  if Assigned(wpview_pdfWorkbenchCreate) then
  begin
    workbench := wpview_pdfWorkbenchCreate(
         WPViewPDF_LicName, WPViewPDF_LicKey, WPViewPDF_LicCode );
  end;
  OpenBtn.Enabled := Assigned(workbench);
end;
```

In OnDestroy the workbench is freed.

```
procedure TForm1.FormDestroy(Sender: TObject);
begin
  if workbench<>nil then
      wpview_pdfWorkbenchFree(workbench);
end;
```

This code is used to load a PDF file

```
procedure TForm1.OpenBtnClick(Sender: TObject);
var s : WideString;
begin
  s := PDFFileName.Text;
  PageCount := wpview_pdfWorkbenchLoad(workbench, PWideChar(s), 0, 0);
  CloseBtn.Enabled := PageCount>0;
  PageNo.Text := '1';
end;
```

This function will show the print dialog

```
procedure TForm1.PrintBtnClick(Sender: TObject);
var bit : TBitmap;
begin
  wpview_pdfWorkbenchCommand( workbench, COMPDF_PrintDialog, 0, '', nil );
end;
```

## 8.3 Render a PDF page to HDC

This code renders a page in the PDF file to a PaintBox Canvas.

```
PageNumber := 0; // The page to be painted

wpview_pdfWorkbenchCommand( workbench, COMPDF_PrintHDCSetXRes, -PaintBox1.Wid
wpview_pdfWorkbenchCommand( workbench, COMPDF_PrintHDCSetYRes, -PaintBox1.Hei

wpview_pdfWorkbenchCommand( workbench, COMPDF_PrintHDC_SelectPage,
            PageNumber, '', nil );

PaintBox1.Canvas.Lock;
try
  wpview_pdfWorkbenchCommand( workbench, COMPDF_PrintHDC_SelectedPage,
        0, '', Pointer( PaintBox1.Canvas.Handle  ));
finally
  PaintBox1.Canvas.Unlock;
end;
```

To move the position you can use SetWindowOrgExt, example:

SetViewportOrgEx(PaintBox1.Canvas.Handle, 100,100, nil );

# 9 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.

Please note: You can also use the "PDFWorkBench" which was added in 4.1.6

## 9.1 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.

**Please note that the page numbers frompage and to_page start with 1.**

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;
```

filename and destpath are expected to be UTF8 encoded!

Better use the "W" function which supports unicode.

```
  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,  // only in C, in C# use int!
  char * destpath,
  int frompage,
  int to_page,
  int jpegres);


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

**C#**

```
        [DllImport(DLLNAME, EntryPoint = "pdfMakeImageW",
                                CharSet = CharSet.Unicode, ExactSpelling = true,
                                CallingConvention = CallingConvention.StdCall)]
        public static extern int MakeImage(
                    string filename,
                    string password,
                    string licname,
                    string lickey,
                    int liccode,  // not "long" !
                    string destpath,
                    int frompage,
                    int to_page,
                    int jpegres);
```

**VB**

```
 Declare Function pdfMakeImage Lib "wPDFViewDemo04.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 unless you use the "W" function.

**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)

Note: **pdfMakeJPEG** is still supported but should not be used anymore.
In contrast to pdfMakeImage it expects an ANSI string and not UTF8.

Previously characters % needed to be escaped with % in the function pdfMakeImage and pdfMakeJPEG.
We changed the code so only %d is reserved as placeholder for the page number.

### 9.1.1 Example .NET - C#

Please note the const **DLLName** - they have to be changed for the demo vs. the PLUS version.
The full version also requires the correct licenses data.

This DLLs are required in the binary directory:
32 bit: *wPDFViewDemo04*.dll, wp_type1ttf.dll, wpdecodejp.dll
64 bit: *wPDFViewDemo04x64*.dll, wp_type1ttf64.dll, wpdecodejp64.dll

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Runtime.InteropServices; // for DllImport

namespace CallWPViewPDF
{

    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            int res;
            if (IntPtr.Size == 8)
                res = wpviewpdf64.MakeImage(textBox1.Text, "",
                     "testlic", "testkey", 12345, // License
                     textBox2.Text + "\\page%d.png", // Dest path
                     0, 10000,
                     200);
            else res = wpviewpdf.MakeImage(textBox1.Text, "password",  // pdf file,
password
                     "testlic", "testkey", 12345, // License
                     textBox2.Text + "\\page%d.png", // Dest path
                     0, 10000,
                     200);
        }
    }
```

```
    public class wpviewpdf : object
    {
        const string DLLNAME_32 = "wPDFViewDemo04.dll";

        /* 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; */

        [DllImport(DLLNAME_32, EntryPoint = "pdfMakeImageW",
                                CharSet = CharSet.Unicode, ExactSpelling = true,
                                CallingConvention = CallingConvention.StdCall)]
        public static extern int MakeImage(
                    string filename,
                    string password,
                    string licname,
                    string lickey,
                    int liccode,  // not "long" !
                    string destpath,
                    int frompage,
                    int to_page,
                    int jpegres);
    }

    public class wpviewpdf64 : object
    {
        const string DLLNAME_64 = "wPDFViewDemo04x64.dll";

        [DllImport(DLLNAME_64, EntryPoint = "pdfMakeImageW",
                                CharSet = CharSet.Unicode, ExactSpelling = true,
                                CallingConvention = CallingConvention.StdCall)]
        public static extern int MakeImage(string filename,
                    string password,
                    string licname,
                    string lickey,
                    int liccode,  // not "long" !
                    string destpath,
                    int frompage,
                    int to_page,
                    int jpegres);
    }
}
```

### 9.1.2 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;
```

### 9.1.3 pdfMakeImageExt

This special version of pdfMakeImage has been created to read PDF from a COM stream and write the created bitmap data to a COM stream.

function pdfMakeImageExt(
 InStream : IUnknown; // Load PDF from here.
 OutStream : IUnknown; // Save the image data here
 filename : PWideChar; // Not used if InStream was specified
 PageCount : PInteger; // Receives count of pages in PDF
 password : PWideChar; // Password for the PDF
 licname, lickey: PWideChar; liccode: Cardinal;
 destpath: PWideChar; // Not used except to specify format
 // if OutStream was specified (Use %d store page number)
 frompage, to_page: Integer; // If OutStream was specified only one page is written
 imageres : Integer; // The resolution, i.e. 200
 // or HighWord=Height, LowWord=Width of desired image size.
 // The bitmap will fit into the specified rectangle
 imagemode : Integer // This is a bit field to specify the color and stream mode.
 ): Integer; stdcall;

or, in C++ syntax:

stdcall int pdfMakeImageExt(
 IUnknown InStream, // Load PDF from here.
 IUnknown OutStream, // Save the image data here
 wchar *filename, // Not used if InStream was specified
 int *PageCount, // Receives count of pages in PDF
 wchar *password, // Password for the PDF
 wchar *licname,
 wchar *lickey,
 int liccode,
 wchar *destpath,; // Not used except to specify format
 // if OutStream was specified (Use %d store page number)
 int frompage,

```
  int to_page: Integer, // If OutStream was specified only one page is written
  int imageres,  // The resolution, i.e. 200
  // or HighWord=Height, LowWord=Width of desired image size.
  // The bitmap will fit into the specified rectangle
  int imagemode // This is a bit field to specify the color and stream mode.
  );
```

All strings must be passed as wide character strings.

Example - C++Builder

```
  res := pdfMakeImageExt(
          InStream,
          OutStream,
          PWideChar(filename),
          @PageCount,
          '',
          WPViewPDF_LicName,
          WPViewPDF_LicKey,
          WPViewPDF_LicCode,
          PWideChar(os),
          IOParams.ImageIndex+1,
          IOParams.ImageIndex+1,
          ImageResolution,
          0
          );
```

**imageres** is the desired resolution for the created image, i.e. 200.

You can also specify the desired width and height of the bitmap inpixels. To do so, pass the width as high-word and the height as low-word of parameter imageres.

**imagemode** is used to specify the color depth.
1 - select BW Images
2 - Grayscale Image - default 24 bit color Image
otherwise a 24 bit color Image will be created

Bit 17-24 is used to specify the JPEG Compression (0-100)

Use the value 8 to create a gray scale image, 0 to create 24bit color.

**InStream** can be nil or a IStream reference. If it is a stream reference the PDF data will be loaded from there.

**OutStream** can be nil or a IStream reference. In the latter case the created bitmap data is written there. Filename must be still provided to make it possible to determine the desired image format. Possible file extensions are .BMP, .JPG, .PNG.

If the parameter **imagemode** is set to 32768 multiple images can be exported to a single stream with one call. The stream will use this header data:
INTEGER = total size of data (=StreamSize)
INTEGER = Count of Images
INTEGER[Count of Images] = Offset of each Image

If the Offset is 0, the image is empty.
The Length of the image data can be calculated as difference of the following, or, for the last image, the stream size

## 9.2 pdfConvertToTIFF - convert selected PDF pages to TIFF

**This method is only available in WPViewPDF "PLUS" in the 32bit edition.**

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;
```

filename and destpath are expected to be UTF8 encoded!

Better use the "W" function which supports unicode.

```
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 "wPDFViewDemo04.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 unless you use the "W" function.

**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); // 'wPDFViewPlus04.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;
```

## 9.3 pdfPrint / pdfPrintW - PRINT PDF function

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

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.

Please note: pdfPrint expects char * parameters which are ANSI characters. If the filename can contain special characters/umlauts it is better to use pdfPrint**W** and pass 2 byte unicodes. In both cases the strings have to be terminated by \0.

**pdfPrintW** has two additional parameters **data** and **datalen**. If not 0, a buffer with PDF data is expected which is loaded or, if a file was also specified, appended prior to printing.

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

**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
REVERSE =1 - print in reversed order

ADDPRINTER = xxx. The mentioned printer (probably a network printer) will be added to the list of printers and also selected unless a different name has been specified with PRINTER=xxx

**Show the print dialog**

DIALOG = 1

**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

**Print from memory**

The filename can be used to transfer PDF data as a memory block.. To do so pass its size as option:
MEMORYSIZE = x

**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

LIMITA3 = 1: force any pages larger than A3 to be scaled down to A3 using default stretch mode.
Stretchmode is automatically set to 0.

**Troubleshooting:**

DONTSETDEVMODE=1 will disable any modifications to the printer setup
DONTSETDEVMODE=2 will only allow the change of the page orientation

you can also activate the creation of a logfile
LOGFILE=c:\temp\pdfview.log
DEBUGMODE=1

**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.

**Switch off anti alias printing for images - that can be important for barcodes:**

DISABLEAA=1
DISABLEANTIALIAS=1

***Initialize JBIG2 EXE plugin***

This option is obsolte - the JBIG2 decoding DLLs are wpdecodejp.dll for 32 bit, the file wpdecodejp64.dll for 64 bit projects.
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; pdfPrintW): 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,
 unsigned long liccode,
 char *options);
```

**MSVC++ 6.0 / MFC Example:**

```
HINSTANCE hiDll = LoadLibrary( "wPDFViewDemo04.dll" );
// int pdfPrint(string filename, string password, string license_name, string license_key, int licens
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 "wPDFView04.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("wPDFViewDemo04.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("wPDFViewDemo04.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: PAnsiChar;
  licname, lickey: PAnsiChar; liccode: Cardinal;
  options: PAnsiChar): Integer; stdcall;
  external 'wPDFViewDemo04.dll' name 'pdfPrint';


function pdfPrintW(filename: PWideChar; password: PWideChar;
  licname, lickey: PWideChar; liccode: Cardinal;
  options: PWideChar;
  data: Pointer; datalen: Integer): Integer; stdcall;
  external 'wPDFViewDemo04.dll' name 'pdfPrintW';
```

Note: The unit WPViewPDF initializes the function pointer wpview_.... which can be used directly:

```
wpview_pdfPrintW: fktpdfPrintW;
wpview_pdfPrint: fktpdfPrint;
```

are initialized by the function **WPViewPDFLoadDLL**(DLLName: string; Quiet : Boolean = FALSE): Boolean; which is called automatically when a viewer will be created but can also be called directly.

Please update the code to use wPDFView*Demo*04.dll or wPDFView04.dll.

## 9.4 pdfMerge / pdfMergeW - 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.

Please note: pdfMerge expects char * parameters which are ANSI characters. If the filename can contain special characters/umlauts it is better to use pdfMerge**W** and pass 2 byte unicodes. In both cases the strings have to be terminated by \0.

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

```
Private Declare Function pdfMerge Lib "wPDFViewPlus04.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
```

Example - merge a.pdf and b.pdf and extract a total of 4 pages into an encrypted file:

```
   Dim i
   i = pdfMerge("""a.pdf"",""b.pdf""", "out.pdf", "", "licname", "lickey", 0,
   """PAGELIST=1-3,5"",""UPASSWORD=123""")

(Note: to escape a " in VB6 type "" )
```

Tip: This works in ASP .NET. If you need to use this method in the "old" ASP You can use VB to create a simple ActiveX class which exports just this method.

```
        Public Function pdfMerge_Access(ByVal strFilenames As String, ByVal st
            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 function pointer wpview_.... which can be used directly:

```
wpview_pdfMerge: fktpdfMerge;
wpview_pdfMergeW: fktpdfMergeW;
```

are initialized by the function **WPViewPDFLoadDLL**(DLLName: string; Quiet : Boolean = FALSE): Boolean; which is called automatically when a viewer will be created but can also be called directly.

## Declaration of the merge function in C#

```
// using System.Runtime.InteropServices;
[DllImport("wPDFViewPlus04.dll", CharSet=CharSet.Ansi)]
   public static extern int pdfMerge(string filenames, string newfile, string
   string license_name, string license_key, int license_code, int license_plus
   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 (creates also debug message) |
| "TIFF2PDF=path" | full path to converter DLL |
| "LOGFILE=path" | logs errors in the specified file. Can be combined with DEBUG=1 |

"PAGELIST=1,2,3,10-15" merges the input files but only saves the pages in the list

"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

"COPY_NON_ENCRYPTED=1" pdfMerge will simply copy the original file if only one source file was specified and that source file was not encrypted.
This method is useful if pdfMerge is only used to decrypt PDF files. Do not use this option if you need to apply security!

"DeletePDFAFlag=1" will remove any PDFA marker

"DeletePDFMarks=1" will remove the StructTree, akn as PDF Tags

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

"DeleteFormFields=1" Removes the widget annotations when saving the file.

"FlattenFormFields=1" Applies the appearance streams of the annotations to the PDF file and remove the annotation.

"SaveMode=xx" Sets the save mode using an integer value as used by COMPDF_SetSaveOptions

"PASSWORDxxx=..." additional up to 999 user password to open the PDF files. (xxx is a numer between 1 and 999)

"STAMPFILE=sometextfile.txt" After loading the input files, a stamp script loaded from a file will be applied.
The script uses the same syntax as the command COMPDF_StampText.

"STAMPTEXT=...." uses the provided text as stamp script. It may not contain any quotes. CRNL must be encoded as \r\n since otherwise the options cannot be loaded correctly.

Example Delphi: options := options + ',"STAMPTEXT=' +
StringReplace(StampScript.Lines.Text, #13+#10, '\r\n', [rfReplaceAll])
+ '"';

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

**Result**

The Result is >0 if the operation was successful.
Result = -3 if one or more files could not be converted. You can use the logging to find the problem, i.e. a file could not be found.

## 9.5 pdfGetInfoW

### a) Option = 0: Read info strings

**int pdfGetInfoW**(wchar * filename, wchar * buffer, int buflen, wchar * password, wchar * licname,
wchar * lickey, dword liccode, int Option)

With Option=0, this method can be used to quickly fill a string list with the info items from a certain PDF file. This makes it possible to read "Author" or "Keywords".

The function will encode CRNL characters into "<br>" unless the Option 1024 was used.

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;
```

Also possible Option=1024:
Read info strings with comma as separator between values

## b) Option = 1..4: Read PDF properties of a single page

Using different values for option this method reads certain integer properties and returns the selected value. buflen is used as page number parameter.

**int pdfGetInfoW**(wchar * filename, **int * unused, int pageno**, wchar * password, wchar * licname,
wchar * lickey, dword liccode, int Option)

Values for Option:

1: Read pagecount
2: Read page[pageno].width
3: Read page[pageno].height
4: Read page[pageno].rotate

Width and height are measured in *pt*, this is 1 inch / 72. Rotate can be 0, 90, 180 and 270 (degree)

## c) Option=5: Read page properties of multiple pages with one call.

The function returns the page count. Since the PDF file is only loaded a single time, this works much more efficiently than multiple calls.

**int pdfGetInfoW**(wchar * filename, **int * values, int maxvaluescount**, wchar * password, wchar * licname, wchar * lickey, dword liccode, int Option)

values is used a integer array.
values[(pageno*3)+0] = width of page # pageno
values[(pageno*3)+1] = height of page # pageno
values[(pageno*3)+2] = rotate of page # pageno

maxvaluescount is the size of the array, it should be larger or equal to pagecount * 3;

maxvaluescount must be large enough to hold all values, otherwise not all page sizes can be returned.
In any case (and if values=nil) the return value will be the count of pages in the document.

## d) Option = 1024

Read info strings with comma as separator between values

## Declaration:

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

fktpdfGetInfoW2 = function(filename: PWideChar; IntRef: PInteger;
  param: Integer; password: PWideChar; licname, lickey: PWideChar;
  liccode: Cardinal; Option: Integer): Integer; stdcall;
```

The Return Value of pdfGetInfoW is <0 if the PDF file could not be loaded.

### Delphi Example:

Create TShapes in a scrollbox for all pages in a document

```
var i, n, y : Integer;
    s : string;
    pag : array of Integer;
    aPage : TShape;
    lic_name, lic_key : WideString;
const MAXPAGES = 3000;
begin
  if not assigned(wpview_pdfGetInfoW2) then
  begin
    ShowMessage('function pdfGetInfoW2 is not available');
    exit;
  end
  else if OpenDialog1.Execute then
  begin
    s := OpenDialog1.FileName;
    ListPagesFilename.ReadOnly := false;
    ListPagesFilename.Text := s;
    ListPagesFilename.ReadOnly := true;

    for i := ListPagesScroll.ControlCount-1 downto 0 do
              ListPagesScroll.Controls[i].Free;
    SetLength(pag, MAXPAGES*3);  // expect MAXPAGES pages ...
    lic_name := WPViewPDF_LicName;
    lic_key := WPViewPDF_LicKey;
    n := wpview_pdfGetInfoW2(PWideChar(s), @pag[0], MAXPAGES*3,
      '', // Password
      PWideChar(lic_name), PWideChar(lic_key), WPViewPDF_LicCode,
      5);
    if n <0 then ShowMessage('Cannot open file!')
    else
    begin
```

```
        y := 10;
        for i := 0 to n-1 do
        begin
          aPage := TShape.Create(ListPagesScroll);
          aPage.Tag := i;
          aPage.Width := pag[(i*3)+0] div 10;
          aPage.Height := pag[(i*3)+1] div 10;
          aPage.Parent := ListPagesScroll;
          aPage.Left := 10;
          aPage.Top := y;
          inc(y, 5 + aPage.Height);
        end;
        if n>MAXPAGES then ShowMessage('PDF cannot be fully scanned')
    end;
  end;
end;
```

# 10 Whats new in WPViewPDF V4

WPViewPDF V4 was developed to make it possible not only to view PDF files but also to really work with them.
To make it easy to provide the user a powerful GUI a new action system has been integrated. It makes it possible to automatically initialize the menus and toolbars required. Most work has been put into the draw object and the new annotation system. Now it is possible to offer the user the possibility to add annotations to the PDF information. It is also possible to extract attachments from the loaded PDF files.

- All new action system. This allows it to create a GUI efficiently and quick.
- new ActionMode: pan, select objects, draw etc.
- add PDF annotations to a PDF file.
  Supported are at present:
  * highlight annotation
  * text background (the used can select text and the annotation will cover the area)
  * square annotation
  * symbols with Popups
  * squiggly underline annotation
  * links
- PLUS: add field widgets (form fields) to a PDF file
  * text fields (also multi-line)
  * check boxes
  * combo box
  * list box
- move any existing annotation
- delete any existing annotation
- drag&drop support (added January 2019)
- scale the PDF pages when saving to a new PDF file
- PDF form filling
- create draw objects on a "document layer". The document layer survives reloading a PDF file. This makes it possible to apply the same draw objects to various PDF files. With WPViewPDF PLUS it is possible to save the objects to XML and load as XML.
- trigger mail merge events on marked text draw objects.
- flatten PDF forms / annotations

- convert PDF into watermark: The user can select a PDF file and use certain PDF pages as background for the current PDF files. It is possible to reuse the same page on multiple pages.
- Extract attachment data, i.e. ZUGFeRD XML
- Although WPViewPDF does not include java script support, it is possible to use the standard script functions as single lines in the (K)ey and (F)ormat Actions of a Widget.

Please note that saving is disabled for protected PDF files by default.

Using the compiler switch IGNORE_SECOPT_IN_DFM it is possible with Delphi to disable that the property SecurityOptions is loaded from the DFM data. This makes it possible to set the property later in code.

**Please review the list of internal actions for any relevant changes.**

**FireMonkey anyone?**
If you are interested in FireMonkey development,
please join our FireMonkey user group in the support forum at
**www.wpcubed.com/forum/board/**

**WPViewPDF 4.8.2.0 release 29.10.2019**
- font encoding for OSX PDF files.
- fix offset for draw text objects

**WPViewPDF 4.8.1.1 release 9.10.2019**
- fix a problem introduced by version 4.8.1.0
- annotations and draw objects did not paint and export correctly when the page cropbox used a negative x and y parameter
+ COMPDF_ZoomThumbnailsAuto = 78; // Set the auto zoom property of the thumbnail view
// 0 = Off, 1= Width, 2=FullPage, 3=SideBySide, 4=AsManyAsPossibleInRow (default) 4=AsManyAsPossibleInRowMinOne
* PDF fonts which were not using a certain encoding are now using the "Standard" encoding

**WPViewPDF 4.8.0.2 release 8.9.2019**
- fix problem when decoding 2 bit images
+ it is possible to set a filter to limit the area where the text is extracted.
pdf.command(COMPDF_GetTextSetOptions, 4+2); // Activate the filter
pdf.command(COMPDF_GetTextFilterRectX, 150); // ..Y, ..X1, ..Y1

**WPViewPDF 4.8.0.1 release 30.8.2019**
- fix problem with 64 bit use in .NET
* call fall back routine for fonts which are missing cmap table

**WPViewPDF 4.7.3.3 release 21.7.2019**
- improves compatibility with PDF with encode descendent fonts in embedded

objects instead of indirect
- the text export could write #0 to html files. Those will now be skipped.

**WPViewPDF 4.7.3.1 release 7.6.2019**
- checkboxes were not flattened correctly when AS property was not correct.
* command COMPDF_ZoomThumbnails can now be used to set the maximum zoom value for thumbnails. The default is 10%.
(Any increase of the window width will created additional columns if the maximum zoom value has been reached)

**WPViewPDF 4.7.3.0 release 10.4.2019**
+ command COMPDF_LoadActionCursor can be used to change the cursor used by certain
Action modes. Use command( COMPDF_LoadActionCursor, 1, crCross ) to change standard cursor.
+ added command function to VCL interface:
function TWPViewPDF.command(command: Integer;
IntParam: {$IFDEF WIN64} IntPtr {$ELSE} Integer {$ENDIF};
IntParam1: Integer; IntParam2: Integer = 0;
IntParam3: Integer = 0; IntParam4: Integer = 0 ): Integer;
- fix problem with 2 bit indexed color images with
- fix of ToInicode interpretation if no spaces were used as separator

**WPViewPDF 4.7.2.0 release 18.3.2019**
+ the command COMPDF_Zoom can now be used with the string parameters
StrPar='XYWHx,y,w,h' to scroll to the PDF coordinates x,y,w and h (72 dpi on the page)
StrPar='PXYWHp,x,y,w,h' to scroll to the PDF coordinates x,y,w and h on page p

**WPViewPDF 4.7.1.2 release 7.3.2019**
- fixes problem with 64bit DLLs in last build - they were not loading

**WPViewPDF 4.7.1.1 release 2.3.2019**
* PLUS edition: improved form handling

**WPViewPDF 4.7.1.0 release 27.2.2019**
+ added some support for FunctionType 4 separation color
- fixed "stack error" exception

**WPViewPDF 4.7.0.0 release 25.2.2019**
+ new command: COMPDF_SetAnnotEditModes modifies the way the TAB key works in formular mode
* improved TAB handling and scrolling formular mode

**WPViewPDF 4.6.5.1 release 21.1.2019**
+ **new:** Drag&Drop support with Delphi and TWPViewPDF.
To use assign events OnDragOver and OnDragDrop.

- fix stability problem when drag&drop was used with Delphi 10.3 Rio in 64bit application.

**WPViewPDF 4.6.4.6 release 12.1.2019**
- improvements to 64 bit DLL
- fix for bullet sign

**WPViewPDF 4.6.4.5 release 13.12.2018**
* Updated documentation for command COMPDF_Attachment_AddAF.
--> Please pass the id returned by this command as second parameter when you load the data which should be used by a PDF attachment.
- fixed problem when saving PDF file with attachments

**WPViewPDF 4.6.4.4 release 5.12.2018**
- correctly hides text written with Tr3 mode but still allows text selection

**WPViewPDF 4.6.4.3 release 30.11.2018**
- improves compatibility to PDF files which save spaces in front of object numbers
- solves AV problem on problematic CCITT data

**WPViewPDF 4.6.4.1 release 19.10.2018**
- when rendering draw objects with right aligned text into the PDF the text was not positioned correctly
- when rendering draw objects with centered aligned text was not positioned correctly

**WPViewPDF 4.6.4.0 release 16.10.2018**
* PrintHDC, GetMetafile does not cache the page commands anymore which reduces the
memory consumption if a large number of pages are processed.
- fix problem with certain images with indexed color
- fix problem that font size was not read from widget annotations (DA parameter)
- fix line breaks in multi-line edit fields
* .NET interface supports new parameter options in AddDrawObject()

**WPViewPDF 4.6.3.3 release 10.9.2018**
- fix cache problem which caused large memory consumption when extracting text
+ command COMPDF_NoViewer can be used to *temporarily* disable the rendering for fast loading and text extraction
- Annotation Flags "F" were not handled correctly. They are now correctly loaded from prp.v.F
This causes annotations which are supposed to be invisible to be hidden (though they are selectable in certain operation mode)

**WPViewPDF 4.6.3.2 release 22.8.2018**
+ command COMPDF_RenderDrawobjects now returns the number of rendered

objects.
Use Bit 8 in the parameter to only count the objects.
- fixed problems when writing check boxes in PDF forms
* improved text extraction (more exact X coordinate calculation)

**WPViewPDF 4.6.3.1 release 14.8.2018**
- the scaled print mode (COMPDF_PrintUseScaling) also set the paper size which worked against the purpose

**WPViewPDF 4.6.3.0 release 9.8.2018**
- fix cmap reading error when <..> codes were on the same line
- fix problem with acroform objects which used fields as indirect reference
- fix problem with PrintDialog loosing focus on Alt+TAB
- COMPDF_ACTION_WRITE can be used to localize the action captions.
The problem that some sub menus were not been localized has been fixed.
+ understand Encoding: UniGB-UTF16-H

**WPViewPDF 4.6.2.3 release 24.7.2018**
* change default value for COMPDF_GetTextSetOptions = 272 - set bit 2 if you need y position written as text baseline position (=default) or clear bit 2 to save the top position. (Consistency to WPViewPDF V3)

**WPViewPDF 4.6.2.0 release 21.7.2018**
+ new command: COMPDF_GetTextSetOptions (=272) - set bit 2 if you need y position written as text baseline position or clear bit 2 to save the top position

**WPViewPDF 4.6.1.0 release 2.7.2018**
+ The text extraction method has been revised for better detection of lines of text, even if not
printed at the exact same vertical position.

**WPViewPDF 4.6.0.0 release 29.6.2018**
+ command COMPDF_SinglepageMode,2 can now be used to toggle the single page mode.
* improve compatibility with PDF files which use Encoding as indirect reference.

**WPViewPDF 4.5.3.0 release 29.5.2018**
- fixes problem with XForms
- fixes problem with separation color used for stroking
* updated Demo DLL

**WPViewPDF 4.5.2.5 release 27.4.2018**
- fixes problem with annotations (caused by 4.5.2.2 - XForm handling)
- check annotation with CA property = 8 now shows X

**WPViewPDF 4.5.2.3 release 20.4.2018**
+ command DONTSETDEVMODE can now be used to disable the landscape

detection on w>h
- workaround for PDF files which use incorrect ASCII85 encoding

**WPViewPDF 4.5.2.2 release 11.4.2018**
* improve ToUnicode interpreter to not fail on incorrect range
* limit size of intermediate bitmap used for PrintHDC when UseRotation=true
* improve PDF compatibility
- handles inverted CMYK image stream
- use current state as default state for XForm painting.
Solves problem if XForm does not define any colors.

**WPViewPDF 4.5.2.0 release 28.3.2018**
- fix problem with "use rotate" parameter of PrintHDC

**WPViewPDF 4.5.1.0 release 8.3.2018**
- fix internal AV introduced by new "scn" handling
- fix redraw issue after text selection

**WPViewPDF 4.5.0.0 release 7.3.2018**
+ command **COMPDF_Get_XMPBuffer** can be used to read the loaded PDF metadata.
The IntParam is used to select the loaded PDF-File to examine. (default=0).
It is also possible to provide the NAME of the PDF-File to check in the StrParam.
The Result Value = -2 if IntParam is not valid or the provided PDF filename was not loaded before.
- invisible text can now be selected
- Rendering of annotations and draw objects used wrong position when a crop box was used in PDF.
- improvement to comment handling for postscript page description
- improvement of text highlight to annotation conversion. (AddHighlightAnnotationForText)
- MouseUp event was lost when internal code handled mouse event as well.
+ AddHighlightAnnotationForText now accepts Alpha parameter in Range 0..255

**WPViewPDF 4.4.4.0 release 7.2.2018**
* the PDF to text feature now adds spaces if it detects gaps between rendered text elements
this feature can be switched off with Command (COMPDF_DontSynthesizeSpaces,1)
+ COMPDF_Ann_ModifyAddProps can now be used to set the "HighlightType"
* improved text selection and highlighting

**WPViewPDF 4.4.3.3 release 22.1.2018**
* Handle the case when '-' was used in image draw operation and encoded as #2D in dictionary

+ accept PDF 2.x as input format

**WPViewPDF 4.4.3.2 release 15.1.2018**
* do not write MarkInfo if not all PDF files which are merged use this flag
- fix problem in CMYK conversion

**WPViewPDF 4.4.3.0 release 15.12.2017**
* improvement of handling for embedded objects which are using HexEncoding
+use Command(COMPDF_SetSaveMode, 8192 * 1024) to save all fonts, not just the ones which are used.
- fix problem that draw objects do not work correctly with SetGlobalParameter ('ReduceMemoryUsage=1'
+ better alignment of the view panel relative to the thumbnail/bookmark view

**WPViewPDF 4.4.2.0 release 8.11.2017**
+ detect corrupted PDF data with garbage at end of file
* optimized detection of fonts which are using 2-byte CIDs
* improved support for embedded fonts

**WPViewPDF 4.4.1.0 release 3.10.2017**
- write the flag /EncryptMetadata when file encryption was activated
+ use command( COMPDF_SetSecurityMode, 1+4096) to activate encryption also of metadata in 40 bit or
command( COMPDF_SetSecurityMode, 2+4096) in 128 bit security mode.

**WPViewPDF 4.4.0.0 release 22.9.2017**
- fix problem when saving PDF file with named destinations
- fix problem when displaying bookmarks
* improvement when saving PDF which contain unicode strings in name array

**WPViewPDF 4.3.4.0 release 20.9.2017**
+ use command(COMPDF_SetSaveMode, 8192 * 256) to always write NeedAppearances=true
+ use command(COMPDF_SetSaveMode, 8192 * 512) to always write NeedAppearances=false. This fixes the problem that AcrobatReader asks to save changes.
+ experimental: Use SetGlobalParameter('ReduceMemoryUsage=1') to reduce the storage of path objects.
That helps with PDF files which use a lot(!) of text paths on a page but can slow down repaint.

**WPViewPDF 4.3.3.0 release 7.9.2017**
- COMPDF_GetPrinter could not be used with CommandGetStr
- fix problem with desired width/height with GetImageExt

**WPViewPDF 4.3.2.0 release 26.7.2017**

- update to font character selection

**WPViewPDF 4.3.1.0 release 10.7.2017**
+ using the command COMPDF_Ann_Undo = 560 it is now possible to disable and enable the UNDO buffer.
use Command(COMPDF_Ann_Undo, 4) to disable, Command(COMPDF_Ann_Undo, 5) to enable.

**WPViewPDF 4.3.0.0 release 21.6.2017**
+ flatten form (annotations) when saving the file
+ option DeleteFormFields=1 in pdfMerge()
+ option FlattenFormFields=1 in pdfMerge()
+ The save modes can be specified with SaveToFile command
+ The save modes can be specified with pdfMerge as parameter SaveMode=x
- fixed memory issue caused by undo system

**WPViewPDF 4.2.5.0 release 31.5.2017**
+ COMPDF_Ann_ModifyAddProps can now be used without updating the screen at once. Add option 16.
- fix exception error which could happen with some corrupted PDF files.

**WPViewPDF 4.2.4.0 release 11.5.2017**
- fixed hyperlinks to named destinations

**WPViewPDF 4.2.3.0 release 3.5.2017**
- fix problem with wrong character width when font height in PDF was negative.

**WPViewPDF 4.2.2.0 release 20.4.2017**
- fix a stability problem when reading annotations which were using quadpoints
+ detect "caret" annotation
* improve strikeout annotation

**WPViewPDF 4.2.1.0 release 5.4.2017**
- fix problem when ScreenToPage was used directly after a LoadFromFile.

**WPViewPDF 4.2.0.0 release 29.3.2017**
- fix problem when saving attachments from PDF
+ add support for Delphi 10.2 Tokyo

**WPViewPDF 4.1.9.9 release 15.3.2017**
+ new option for pdfMerge: COPY_NON_ENCRYPTED=1 will copy the original file if only one source file was
specified and that source file was not encrypted.
+ new option for pdfMerge: PASSWORDxxx=1 sets a password (xxx=1..999) in a list of passwords used for the loaded PDF files.
+ **Use SetGlobalParameter('DrawImageBands=1')** to make WPViewPDF draw

large monochrome bitmaps using several bands instead of
one big bitmap. This has been introduced to increase compatibility with certain printers. (Experimental - may become default in future)

**WPViewPDF 4.1.9.8 release 9.3.2017**
+ support for text RenderMode 1 (outlined text)
- fix for gray scaled JPEG2000 encoded images
* when writing PDF files Signature annotations are written together with the accompanied field

**WPViewPDF 4.1.9.6 release 2.3.2017**
- fix problem for certain JBIG2 images which showed up after last build.

**WPViewPDF 4.1.9.5 release 24.2.2017**
* **property SecurityOptions defaults now to []. Please check if it is required to disable the save functionality**.
* includes a workaround to avoid printing problems of pixel based barcodes on some printers
* improves text find method

**WPViewPDF 4.1.9.4 release 18.2.2017**
+ the VCL defined a new method AddHighlightAnnotationForText
+ when adding a highlight annotation it is possible to create a non rectangular area to cover the previously found text.
Use "**atfoundtext**=1" in the options. Note: x,y,w,h must also be provided!
- fix a flicker problem when highlighting text

**WPViewPDF 4.1.9.3 release 9.2.2017**
- fix problem with FindText - the scrolled position was slightly off

**WPViewPDF 4.1.9.2 release 3.2.2017**
* the text export now calculates more accurate positions in the xyhtml code
* find text returns better coordinates
- fix: when a PDF file used a cropbox the annotation frame was drawn at a wrong position
- acrofields with auto size text were always written with fontsize 12

**WPViewPDF 4.1.9.1 release 27.1.2017**
* avoid problem with PDF files which use comment in the page description
+ MergeText now uses COMPDF_BeginUpdate/COMPDF_EndUpdate to improve performance

**WPViewPDF 4.1.9.0 release 26.1.2017**
- fix for command COMPDF_ACRO_SET which can be used to set the acrofields used by a PDF form

**WPViewPDF 4.1.8.4 release 4.1.2017**
- fix a problem when saving some rare PDF files
* improve scrolling in thumbnail view

**WPViewPDF 4.1.8.3 release 23.12.2016**
* includes workaround for incorrect PDF files which miss endobj markers.

**WPViewPDF 4.1.8.2 release 21.12.2016**
* bitmaps with of bit height will now be printed without any anti alias. This fixes problems with barcodes which were printed with some shading.
- in scanned documents some characters showed up when the font was not defined.
- display characters corrected when CIDtoGID and ENCODING cmap was used.
* improved interpretation of cmap definitions
* the handliung of ligatures (fi) has been improved

**WPViewPDF 4.1.8.1 release 13.12.2016**
- fixes a problem with some images introduced in last release caused by compiler issue
* fixes DEMO expiration problem

**WPViewPDF 4.1.8.0 release 7.12.2016**
+ Improvments to PDF-Workbench. This object can be created be used without having to use a window handle.
* WPViewPDF also works fine to print the PDF pages on a HDC. No window handle is required - PrintHDC
- fixes a memory leak which sometimes occured when PDFs were using monochrome JPEGs
+ using command COMPDF_UseGDIPainter with parameter = -2 it is now possible to select the standard, non
GDI+ painter to be used by PrintHDC and GetMetafile.
+ added a Delphi demo which is using PDF-Wokbench object.
+ added fall-back for font output in case the font could not be loaded by integrated font engine

**WPViewPDF 4.1.7.4 release 24.11.2016**
- fix an internal exception when creating a field
* don't save AcroForm object if it was detected to have no fields.
+ the parameter "F" of a widget controls if the widget is printed or not.
Use **SetGlobalParameter('AddFieldDefaultF=1')** to make WPViewPDF add the value F=4 to all
fields which do otherwise miss this property.

**WPViewPDF 4.1.7.2 release 22.11.2016**
+ event: OnChangePageSelection
+ Use **SetGlobalParameter('rotatemeta=1')** if you need to extract pages as metafiles with applied page rotation.

Instead of SetGlobalParameter(stringvalue) it is also possible to call Command (200000, stringvalue)

**WPViewPDF 4.1.7.0 release 18.11.2016**
- fix: It was not possible to move a page to last position when a page was deleted
- fix problem with special CID font
+ impove drag&drop in thumbnail view
+ added support for color inline graphics

**WPViewPDF 4.1.6.4 release 18.10.2016**
- fix problem with wide outlines in some texts
+ added support for subforms in AcroForms.
* more strict interpretation for Bit 10 in security P flag (=extract graphics and text)

**WPViewPDF 4.1.6.2 release 6.10.2016**
* ignore path drawing inside text blocks (workaround for some PDF files)
* improved display of FreeText annotations
- VCL: In API FindText a variable was named DontGoToPage although its meaning is GoToPage - it has been renamed to "GoToPage"
- fix for Y position when FindText was used.
- FindNext was broken

**WPViewPDF 4.1.6.0 release 28.9.2016**
- improves saving of highlight annotations
- fixes problem with loaded multi line selection
+ during adding highlights the user can press CTRL+Z to undo the last change

**WPViewPDF 4.1.5.7 release 15.9.2016**
* enhanced font width measurement
* enhanced character location in CID fonts
- fix problem with encrypted PDF files which use a non-standard FileID

**WPViewPDF 4.1.5.5 release 31.8.2016**
- drag over caused AV in Delphi XE up to 10.0
- move pages did not work correctly when pages were deleted before. This problem has been fixed.

**WPViewPDF 4.1.5.4 release 22.8.2016**
* we integrated a fix for a possible problem when printing documents multiple times without using the print dialog.
This problem showed up in some other applications after the windows update released 19.8.2016 and also with the pdfPrint() function.
(To switch off this fix use **SetGlobalParameter**('printerfix=0'), to switch it on again us **SetGlobalParameter**('printerfix=1') )
* do not always draw hairline around filled areas

**WPViewPDF 4.1.5.3 release 2.8.2016**

- fix problem when 64bit DLL was used with .NET 4.5
- fix problem with lost image resource when saving PDF

**WPViewPDF 4.1.5.2 release 28.7.2016**
+ using command COMPDF_MODIFIED it is possible to check if the contents were modified by the user, i.e. if pages were moved.
+ new function pdfMakeImageEx can load the PDF data from a IStream and save one ore multiple images to an IStream.
+ new unit: wpcubed_pdf_plugin has been added to use WPViewPDF with ImageEn , the powerful image component for Delphi.
+ new WPViewPDF Edition "WPViewPDF MakeImage" which supports pdfMakeImageEx only.

**WPViewPDF 4.1.5.1 release 15.7.2016**
- do not show OCR text on certain scanned document
+ support for Win2151Encoding (not standard)

**WPViewPDF 4.1.5 release 30.6.2016**
* improved rendering code for auto-textheight multiline freetext annotations
+ when saving a PDF the Catalog/AF array is also saved.
* the saving of names dictonaries has been changed so save names to merged documents, too.
+ new ViewOption: wpDisableAnnotTextEditInEditMode
+ new commands to add file attachments to a PDF file
COMPDF_Attachment_AddAF, COMPDF_Attachment_SetData, COMPDF_Attachment_SetProp
+ new DLL methods to create a non-visual "PDF workbench". With the created object most
viewer commands can be used to examine and manipulate PDF data. This can be helpful if you need
a powerful PDF tool in C++.
* editor will switch into text edit mode of text annotations on click on selected object, not at once

**WPViewPDF 4.1.4'' release 20.6.2016**
- fix display of some (OCR) fonts which are not embedded but loaded from system

**WPViewPDF 4.1.4' release 14.6.2016**
- fix a checkbox saving glitch

**WPViewPDF 4.1.4 release 8.6.2016**
- URI links were not working - now OnHyperlinkWWW can be used
+ **scale the PDF pages when saving to a new PDF file**
New commands: COMPDF_SaveScaledPDFMode, COMPDF_SaveScaledPDFSetX and COMPDF_SaveScaledPDFSetY
When using Delphi you can use the method ActivateScaledPDFWriting to enable the scaling.

**WPViewPDF 4.1.3' release 17.5.2016**
+ annotation flattening mode UseOriginalDataForRendering

+ allow the sign ~ to be used in PDF names

**WPViewPDF 4.1.3 release 14.5.2016**
+ It is possible to flatten PDF forms - the annotations are rendered into the PDF.
+ **new action "FlattenAnnotations"** in group 4, Draw Operations (by default hidden, use RequiredOptionalActions to show)
+ added annotation save modes to remove annotations and widgets
+ it is now possible to set and modify the draw object options using command COMPDF_Ann_ModifyAddProps
- fix annotation loading problem - transparency was not loaded
* improvement in annotation export code
+ new parameter RequiredOptionalActions in procedure WPPDFViewerInitMainMenu

**WPViewPDF 4.1.2 release 4.5.2016**
**USE WPVIEWPDF PLUS to create and fill PDF forms!**
+ create form fields: edits, memos, checkboxes, combox and listboxes
+ support for masks in edit fields (you can use java script one liner)
+ show special cursor if in draw-object or highlight-text mode
+ tabbing in PDF forms now work over page bounderies
+ new dialog to create form fields has been added to "PDFEdit.EXE"
+ command COMPDF_ACRO_GET can be used to extract a list of acro-fields in the document (use ID=-5)
- fixes to COMPDF_Ann_XMLGetFromAcrofield and COMPDF_Ann_XMLSetFromAcrofield
- fixes to COMPDF_Ann_XMLGetFromAnnots and COMPDF_Ann_XMLSetFromAnnots
- cleaned up some naming inconsistencies used by XML scripts:
  Background color is: Background-Color="color_as_string"
  Line color is: Color="color_as_string"
  Font name is: Font="fontname"
  Font Size color is: Font-Size="floatvalue"
  Font color is: Font-Color="color_as_string"
- fixed background glitch for edit field
- fixed paint glitch in thumbnail view

**WPViewPDF 4.1.1 release 26.4.2016**
+ improved compatibility to PDF files which have been truncated
+ form fill mode now also handles comboboxes
+ it is now possible to display a date dropdown in pdf forms
+ improved display of form annotations
+ The VCL includes the function MakeBitmap to create a TBitmap with the contents of a certain page.
+ support for line caps and line joins

**WPViewPDF 4.1.0 release 18.4.2016**
+ **much enhanced interactive PDF form filling**
  ++ highlight current widget
  ++ mouse hover effect for widgets
  ++ use TAB to move to next widget, also checkboxes
  ++ updated toggle code for checkboxes (use SPACE to toggle)
  ++ six different checkbox appearances are supported now
  ++ BETA: evaludate Javescript actions to detect date and number fields.

This mode is activated by COMPDF_FORMFILLOPTIONS.
+ **PDF repair mode** to help to load PDF files which have been corrupted
+ new methods to get and set the ActionMode
+ it is now possible to use command COMPDF_GotoNamedDest,2 to jump to the position referenced by an outline item
- The event OnChangeViewPage was not called as often as required.
*** improvement to AcroForm handling for fields which use multiple widgets**
*** nicer splitter**
+ COMPDF_FORMFILLOPTIONS control formfill mode
+ also highlight invisible (OCRed) text

### WPViewPDF 4.0.8 release 5.4.2016
**+ it is now possible to update the attributes of annotations**
**+ it is now possible to add link annotations**
- fix problem with URI link annotations - the event did not work
- fix problem with "Threading" Action handling
- add support for colored Type3 text
- fix problem with command COMPDF_ACTION
- fix package to also work before XE2
- fix problem when saving named destinations
* added code to handle PDF files which have been corrupted

See new chapter "Internal Actions"

### WPViewPDF 4.0.7.1 release 15.3.2016
* Type3 fonts with rendering mode 3 will be hidden
* added support for 256 bit AES encrypted files (Revision 5)

### WPViewPDF 4.0.7 release 11.3.2016
+ it is now possible to get the action command id for a certain action name
**+ added example developed in C#**
*** updated PDFViewerLib .NET assembly**
- updated BookmarkXML feature also handles outlines with GoTo actions

### WPViewPDF 4.0.0.1 release 22.2.2016
* transparency support for text (used by OCR software)
- included designtime packages - one built with Delphi 7, the other with XE.
We recommend however to create a new package for your compiler by simply adding unit WPViewPDF_reg.pas

### WPViewPDF 4.0.0 release 19.2.2016
+ added localization API
+ added modification of fields and annotations
+ inplace editing for text draw objects (single line)
- fix some API inconsistencies

### WPViewPDF 4 - Beta 2 - release 9.2.2016
- please see demo **WPViewer4** which shows how to use the actions
- please see demo **PrintCertificates** which shows how to use the new document-level draw objects
- the demo **PDFEdit** replaces the old "PDFView" demo

### WPViewPDF 4 - Beta 1 - release 11.12.2015
Is based on WPViewPDF V3 and should behave the same, unless new functionality is used.

Note for Delphi Users: WPViewPDF was designed to keep the loaded PDF file in memory even if the handle (window handle) of the viewing window was destroyed. The data will be released when the component is destroyed. This behaviour makes it possible to implement a docking feature.
To make sure the data is released when the form is closed (but not freed) call the method Clear or disable the compiler symbol **ENABLE_WNDRECREATE** in the file WPViewPDF3.PAS **or add the compiler symbol NOWNDRECREATE to the project conditionals.**

WPViewPDF includes a JBIG2 decoding implemented in the module **wpdecodejp.dll** and, for 64 bit, **wpdecodejp64.dll**. It is **not** required to call the command COMPDF_SetJBIG2Tool when the converter DLLs have been copied to the EXE directory.

Attention, only OCX: The property SecurityOptions was overwritten while VB was loading a form - this had the effect modifications to PDF were not possible.
To disable the Save function you need to use the security commands.

If the SaveToFile or CopyToClipboard function does not work for you, please check the setting of property SecurityOptions!

NEWS: RTF2PDF/TextDynamic Server V4, based on wPDF V4 with 32 and 64 bit support is available now.

# 11 WPViewPDF V3 History

**2.8.2016: V3.28.3**
- fix problem when 64bit DLL was used with .NET 4.5
- fix in PDF saving code

**17.5.2016: V3.27''**
+ allow the sign ~ to be used in PDF names

**15.3.2016: V3.27'**
* Type3 fonts with rendering mode 3 will be hidden

**14.2.2016: V3.27**
* when moving pages was aborted the current page changed
* after page was moved the current page was not the clicked and moved page
- fix in Image and XObject BBox clipping

**6.2.2016: V3.26.2.1**
+ add new save mode: never write cropbox parameter - use Command ( COMPDF_SetSaveMode, flags+4096 ) to activate
+ add new save mode: do not write modified page size - use Command ( COMPDF_SetSaveMode, flags+8192 ) to activate

**4.2.2016: V3.26.2'**
- Two new jbig2 decoder dlls **wpdecodejp** and **wpdecodejp64** solve a problem which occurred when
FreeLibrary was used multiple times from the same process.

- text objects required the text to be at least 2 characters

**29.1.2016: V3.26.2**
- embedded objects in pdf could be written duplicated when used in actions
- improved painting of draw object frames
* unless the page width or height was modified the original MediaBox / CropBox will be written when a PDF file is saved.

**18.1.2016: V3.26.1**
* COMPDF_GetPrinter can be used now with CommandGetStr

**7.1.2016: V3.26''**
- fix a cliping problem with embedded forms (BBox)

**19.12.2015: V3.26'**
- fix rare problem when saving to new PDF
- fix a memory leak problem which occurred on rare PDF files and improve page caching.

**12.12.2015: V3.26**
* watermarks are now clipped by BBox property
+ support for grayscale indexed color space

25.9.2015**: V3.25.4.9'**
- fix problem with prediction decoder which was not defining BitsPerComponent

18.9.2015**: V3.25.4.9'**
- in rare cases embedded images in CCITT fax compression were not read completely

9.9.2015**: V3.25.4.9**
- fix problem with sometimes visible hairline stripes around rectangles.
- fix problem with screen update after programmatic page selection.
* After moving a page the new current page is the first page which was moved.

30.8.2015**: V3.25.4.8''**
- accept * as character for a PDF name
- experimental: interprets code *EI* in embedded images also if not proceeded by whitespace when **SetGlobalParameter**('LazyDecodeEI=1') was used
- GOTOPREV was not working when the end of the last page was visible.

17.8.2015**: V3.25.4.8'**
- fix for prediction decoding for 1 bit data
- update to 1 bit image decoding to avoid inverse display
- change in standard GDI renderer to fix problem with subset fonts

6.8.2015**: V3.25.4.8**
- Patterns are now not filled anymore. There are PDF files which draw pattern over the page which would otherwise erase the contents
- fix problem with XREF syntax used by few PDF files
- fix problem with PDF files consisting of appended singular PDF files
+ **SetGlobalParameter("LoadAllEncodingNames=1")** activates the use of names to locate glyphs in fonts. This works better in some PDF files but can cause in problems if the names were not correct (which we sometimes saw)

17.7.2015**: V3.25.4.5**

* improvement for inline images -fix for BI marker syntax written by few pdf writers
* workaround for pdf files which misses spaces in their pdf page description

10.6.2015**: V3.25.4.3**
* improvement of auto width handling for fonts which do not define width
- fix problem with some JBIG2 images displayed inverted

26.5.2015**: V3.25.4.2**
+ option DONTSETDEVMODE=1 and DONTSETDEVMODE=2 for pdfPrint

18.5.2015**: V3.25.4.1**
- workaround for fonts which use gXX as character encoding names
- load installed files with Identity Encoding if not embedded

5.5.2015**: V3.25.4**
- fix trailer problem with rare PDF files which were not loaded correctly

8.4.2015**: V3.25.3''**
- in few projects after unloading the engine DLL an exception happened. This problem has been fixed.

3.4.2015**: V3.25.3'**
* Incorrect PDF files which use xref tables with an offset can be loaded
- sometimes the JBIG2 DLL was not found in the path of the main DLL. This has been fixed.
+ the OCX has been updated. It is now possible to call wpdfSetGlobalParameter by using CommandStr(200000, param)
wpdfSetGlobalParameter is used for GDI+ troubleshooting. You can pass "StartIGDIPlus" and "StopIGDIPlus"
- the panel in the top right corner was sometimes hidden.

27.3.2015**: V3.25.3**
- sometimes annotations were not drawn at the correct position. The code responsible for this has been redone.
* improved JIBG2 decoding capability

**18.3.2015: V3.25.2'**
* improved text selection
+ the command COMPDF_GetWordAtMousPos selects the word under the mouse
+ CommandGetStr(COMPDF_GetWordAtMousPos) reads the word under the mouse

**12.3.2015: V3.25.2**
- fix a problem with PDF files which used .notdef in encoding definition
- fix problem introduced with V3.25 - fonts which unusual cmaps were not decoded correctly

**26.2.2015: V3.25.1**
- access char sets in embedded fonts in the order they are embedded.

**15.2.2015: V3.25**
- fixes problem with certain characters from subset fonts

**6.2.2015: V3.24.4**
+ pdfPrint understands option REVERSE=1 to print in reversed order
* updated code for COMPDF_SetJBIG2Tool
* modified save code to avoid problems with single numbers in PDF object

**3.1.2015: V3.24.4**
- includes improvement for fonts which included incomplete charsets
- fixes problems with indirect objects for font width arrays

**19.12.2014: V3.24.3**
+ pdfPrint understands Option "DISABLEAA=1" or "DISABLEANTIALIAS=1" to disable the anti alias for images.
- COMPDF_SetPageModeDefault did not work for empty viewers which were filled with APPEND_PAGE instead of loading a file.

**5.12.2014: V3.24.2**
- fix: UseImage used the Y value incorrectly (this problem was probably only in last release due to a compile misconfiguration)
- added images will now use a GUID as name.
- fix in PDF renderer for rare PDF which used TJ offsets at start of array
- fix problem in rare PDF files where pages directory was stored in a compressed object
- Resize showed left panel which was hidden with COMPDF_SetPageModeDefault
- Toggle left panel command required two clicks

**24.11.2014: V3.24.1**
+ VCL: add the conditional **THEMEDWPVIEWPDF** to your project options to use the styleservice to paint the background of the viewer
- Solves lost focus problem in combination with certain VCL controls, most notably DBGrid
* fixes command COMPDF_SetPageModeDefault

**20.11.2014: V3.24**
* in few PDF files which are using fonts with incorrect cmap data characters were missing
- fixed: DrawObject **images** were drawn rotated by 180 degrees. (The bug was introduced by fixing the rotated text - now text and images are drawn correctly)
+ new PageRotation, PageWidth and PageHeight indexed properties were introduced in the VCL TWPViewPDF
* AddDrawObject(wpModifyExistingObj ..) will change dimensions but keep the center point of the object.

**13.11.2014: V3.23.3**
+COMPDF_SetPageModeDefault ,
+ COMPDF_EnableNavigationAfterLoad - control how outlines and thumbnails are displayed

**11.11.2014: V3.23.2'**
+ add support for further annotation types
+ the PDF property PageMode is now used after initial loading of a PDF file to show or hide the thumbnails or outlines
+ apply transparency state before painting a XForm
* SaveSelectionToStream did not work with ranges i.e. '1-3' as documented.
- Fix problem mit GOTO_PREV command.

**31.10.2014: V3.23.1**
- in some settings a PDF was not displayed directly after loading it. This problem has been solved.
* another change to DLL-OCX interface - previous version could disturb VB6 IDE

**24.10.2014: V3.23**
*** Updated OCX - please see note above! Changed loading code for property SecurityOptions**

**to avoid problem in PLUS edition.**
+ The JBIG DLL can now be loaded by a command COMPDF_SetJBIG2Tool with 1 as Integer parameter and the DLL name as string parameter.

**13.10.2014: V3.22.1**
* add wpDontScrollThumbnailsWithView in property ViewOptions to make the thumbnail view not scrolling with the main view
- fix problem with *Ts* operator (Text rise)
- improvement to text object in pdf rendering (WPViewPDf plus)

**9.10.2014: V3.22**
+ new command: COMPDF_SYNC_CURRENT_AS_SELECTED. With parameter 1 a special mode is activated to automatically always select the current page, i.e. while scrolling
- do not display annotations which set bit 2 in the F property
- fix possible problem when loading the JBIG2 support DLL

**23.9.2014: V3.21'**
- fix to avoid load error on files which load certain PDF attributes
- the save method did not handle not-escaped () in producer names
* save method fixes incorrect info records
- with some PDF files draw objects were drawn shifted outside of their selection rectangles

**10.9.2014: V3.21**
- EndOfLine support for CCITTFAX

**12.8.2014: V3.20.2'**
- in indexed cmyk images the highest index was not interpreted correctly.

**5.8.2014: V3.20.2**
- edit field for text fields was not positioned correctly in some PDF files
- inline monochrome images were not displayed if they used indexing
* added support for a new image type

**28.7.2014: V3.20.1**
- fix save problem which occurred in few PDF files when objects were entirely empty

**20.7.2014: V3.20**
**+ WPViewPDF now comes with JBIG2 decoding DLLs.**
do not call *COMPDF_SetJBIG2Tool* anymore

**16.7.2014: V3.13.1''**
- UseImage did not work with wpPageWidthPC and wpPageHeightPC
- COMPDF_StampMetafile, COMPDF_StampMetafileUnder scale incorrectly
The behavior was not changed for backward compatibility, but we recommend to call command ( COMPDF_StampMetafile_Scaling, 0 ) to fix this problem.
+ decode 2 bit grayscale images
- command COMPDF_MakeGetMEMORY did not return the required size when passing a null pointer but -2
- when rendering left aligned text draw objects they became centered
+text draw objects can now also be right aligned
+SaveSelection did not work correctly with an invisible control

**1.7.2014: V3.13.1**
- fix GDI+ memory leak which occurred on rare image types in PDF

**27.6.2014: V3.13**
* **optimized loading routine will open PDF data a lot faster**
- improved threading routine fixes some stability issues
+ With **SetGlobalParameter("DisableThreading=1")** multithreading can be disabled.
If highest possible stability is required, we recommend this setting.
+ WPViewPDF.SetGlobalParameter now stores the parameters if the DLL was not yet loaded.
The parameters will be sent to the DLL after the DLL was loaded.
+ SetGlobalParameter("MinimizeMemoryUsage=1") will disable caching of the PDF page paths.
Text selection is impossible in this case.
- fix problem with ScreenToPage/PageToScreen on rotated pages
- fix problem when printing was started with a page which uses a different orientation than page 1
+ use COMPDF_SetSaveMode with parameter 1024 to remove PDF/A marker when saving file

**11.6.2014: V3.12.8**
- small change in font handling to render subset fonts which did not define encoding

**2.5.2014: V3.12.7'**
* changes to VCL and engine to allow negative coordinates in commands which expect 2 smallints packed in one integer (i.e. x and y)
- DecodeParams arrays interpretation improved
* support for clip box for annotations

**30.4.2014: V3.12.7**
+ wpdfSetGlobalParameter("StopIGDIPlus", 0) can be called before the DLL is unloaded to avoid trouble with GDI+ which under certain circumstances cannot be shutdown in finalization of a DLL. The VCL will automatically make this call before the FreeLibrary in "StopEngine".
* changed code to handle "EI" which marks the end of an embedded image.
- fix AV which occured when no file was loaded and clicking with right mouse button

**28.4.2014: V3.12.6'**
- when font names in PDF used '-' in their names it was required that they were embedded
- Find method changed zoom to full page - it now will not change zoom anymore
- option "MEMORYSIZE" was broken in function pdfprint
+ pdfPrintW can now alo load data from a memory buffer provided as parameter data + datalen

**20.4.2014: V3.12.6**
- fixed: DrawObject text was drawn rotated by 180 degrees
+ It is now possible to add transparency to draw object texts
+ It is now possible to specify the font of draw objects
- fixed problem with Type1 fonts
+ added possibility to set origin position to for added draw objects (see demo).
- fixed function pdfPrint
- PageToScreen did not correctly revert the coordinates provided by ScreenToPage

**11.4.2014: V3.12.5**
- **characters "%" needed to be escaped as "%%" in the function pdfMakeImage and pdfMakeJPEG**.
We changed the code so only %d is reserved as placeholder for the page number, escaping is not needed anymore.
* updated code for embedded uncompressed Type1 font programs
* accept char(32) before EI in case 2 char(32) follow. (quickpdf fix)

**2.4.2014: V3.12.4**
+ Support for text state "Tz"
* better support for hyperlink with launch action
- fix problem with bitmap masks which are inverted

**22.3.2014: V3.12.3''**
- with R2 PDF security (40bit) the HQ-Print Flag was evaluated, although it is only used in R3 security (128bit)

**13.3.2014: V3.12.3'**
+ added the commands COMPDF_ImageSetHidden and COMPDF_ImageSetDisplayed to change the visibility of images inserted by command COMPDF_ImagePrint.

**12.3.2014: V3.12.3**
* smarter caching which can be also controlled by this new commands:
COMPDF_SetMaxCachePixels
COMPDF_SetMaxCachePixelsThumbs,
COMPDF_SetMaxCachePathLockTime
+ **added additional functionality to command COMPDF_DisableSecurityOverride** to enable print and high quality print.
- improved text stamping, numformat did not always work as expected.
- VCL: Improve popup menu handling

**20.2.2014: V3.12.2'**
+ The 32 bit edition now does JPEG2000 decoding (**JPXDecode**)
* change in drawing code to avoid fine lines in images consisting of several parts (GDI+ Rounding)
- fix problem with Tw handling
- COMPDF_ScreenToClient did not provide correct information it was supposed to
COMPDF_ScreenToClient can be used to convert a screen coordinate into a logical page x, y coordinate
- ScreenToPage now works as documented

**3.2.2014: V3.12.1''**
- after rotating the first page in thumbnail view the large view was not always updated
* pdfGetInfoW will now encode CRNL inside info strings encode as "<br>". Optionally the Option 1024 can be used. In this case it will create a comma separated list for the values.

**30.1.2014: V3.12.1'**
- a certain type of cmap caused an endless loop

**23.1.2014: V3.12.1**
* If multiple rectangles are drawn they are now combined into one path if they are filled with odd-even rule.
* changed handling for transparent images which use a 2*2 bitmap as color source (avoid marquee effect)
* monochrome masks are inverted unless ImageMask=true
- fix problem with parameter rotate=-1

**17.1.2014: V3.12.0'**
+ VCL only: If the compiler symbol ENABLE_WNDRECREATE is active (=default in WPViewPDF3.PAS) the internal data is buffered before a window handle is destroyed until the object has been freed. This makes it possible to change the parent of a VCL control which implicitly destroys and recreates window handles.
* in case the AcroForm object of a PDF file specifies NeedAppearances=true, all text annotations will be rendered using the provided field data and not the appearance stream. This improves compatibility with products creating incorrect appearance streams. This mode can be disabled using command COMPDF_SetPaintMode, bit 4 (value 8)
- Mask parameter for images handled better

**11.1.2014: V3.11.9''**
* support for transparency for text paths

* support for transparency in annotation appearances (CA property)
* support for negative page rotation

**9.1.2014: V3.11.9'**
+ pdfPrint now understands the option ADDPRINTER. The mentioned printer will be added to the list of printers and then selected.
+ new ViewOption wpNoHyperlinkCursor to disable switching the mouse cursor over links. (note: wpDontUseHyperlinks can be used to disable the internal handling of links, the event can still be used to jump to destinations)

**29.12.2013: V3.11.9**
* updated support for images with transparency masks
+ message MSGPDF_SCROLLHORIZONTALLY is sent for horizontal scrolling
+ command COMPDF_GetGetHScrollSize read width and height of horizontal scroll panel
+ command COMPDF_GetGetVScrollSize read width and height of vertical scroll panel

**3.12.2013: V3.11.8**
+ added function pointer to wpview_pdfMakeImage and wpview_pdfMakeImageW to VCL
* updated Delphi "DirectDLL" demo
+ pdfGetInfoW can now also be used to read the page count and page sizes used in PDF document
- pdfMerge created incorrect error codes
- pdfPrint now creates a debug message if a file was not found (error in filename)

**3.12.2013: V3.11.7**
+ COMPDF_GetPageNumbersInView = 223; Gets a string with all the numbers of the pages which are currently displayed (at least partly). First Page is "1"
- avoid draw problems with image drawobjects with w or h <0
- modified detection for fonts which use a unicode charmap also works if charcount<255

**21.11.2013: V3.11.6''**
- handle faulty annotation streams

**13.11.2013: V3.11.6'**
- solves problem when there is a PDF % comment before PDF "trailer" object
* updated code to render stamps. They were sometimes not positioned correctly

**12.11.2013: V3.11.6 - WPViewPDF PLUS:**
**NEW AND UPDATED FUNCTIONS TO FILL FORMS INTERACTIVELY!**
+ command: **COMPDF_ACRO_MAKEDRAWOBJ** - converts fields in a PDF into DrawObjects. This makes it possible to fill a form interactively.
Example: WPViewPDF1.CommandStrEx(COMPDF_ACRO_MAKEDRAWOBJ,'',2+8+16);
+ Command: COMPDF_DrawObjectSelect Select an object with a certain ID
+ Command: COMPDF_DrawObjectDeSelectAll Deselect all objects
The ID is provided by WPViewPDF1.CommandEx(COMPDF_ACRO_GET, Cardinal(-3)) after a call to COMPDF_ACRO_GET, index to get the name
+ command COMPDF_DONTSETDEVMODE allows this values:
0 - default behaviour
1 - do not change printer parameter before printing
3 - do not set any print parameter except for page orientation
* save NeedAppearances true if fields were changed.
+ command COMPDF_SelectPaperOrientation to select the printer paper orientation.
+ command COMPDF_DrawObjectGetSelected can be used to read the draw objects which are currently selectd.

23.10.2013: V3.11.5
- free text annotations were not positioned correctly
- when drawobjects (stamps) were not rendered into the PDF the display was not accurate on

rotated pages. (Position and size and rotation was not correct). This has been fixed.
- fixed problem with CCITT images which used RGB index

6.10.2013: V3.11.4
- fixed possible problem when Application.Terminate was used
+ added support for images, text and vectors using CalRGB colorspace
* added workaround for indexed images where index uses same value for all items
+ added command COMPDF_STOP which stops the rendering thread
+ use command **COMPDF_GetModified** to read modified state for PDF, it is now set by page move, deletion and rotation commands
and cleared internally when the viewer was cleared.
* added compiler switch for Delphi XE5
Note: MadExcept fixed problem with exception at 0x000014 at startup. You need to get latest version of MadExcept.

26.9.2013: V3.11.3'
- fix in unicode text detection
- fix for high memory when scrolling was done by incrementing the Page property
- fix to avoid problem with PDF files which use highly uncommon large resource dictionaries

20.9.2013: V3.11.3
+ when the user browses the document using links or bookmarks, the position is logged.
**After the backspace key was pressed or Command(COMPDF_GotoPrev, 2) the last position is located.**
- fixed problem with text in few PDF
- fix ListOutOfBounds exception on rare PDFs
- when doing fast scrolling (thumbtrack) the memory consumption went high
- fix problem when clipping was used inside of Type3 definition
* faster repaint after page rotation

12.9.2013: V3.11.2'
- fix a problem in page caching - to much memory was allocated.
- fix problem in FindText function
- HighlightText now can also work case sensitively

11.9.2013: V3.11.2
+ COMPDF_SetPageNumberStringViewer
+ COMPDF_SetPageNumberStringThumbs
- solves problem with links which point to a page with \nul coordinate values
- fixes a floatingpoint error

30.8.2013: V3.11.1
* hardening of the GDI+ interface code
* improvement in save routine to solve problem error 109 in Acrobat PRO
* fault tolerant handling of embedded images in Type3 scripts

23.8.2013: V3.11.0'
+ pdfMerge understands the option ***STAMPFILE=sometextfile.txt*** to load a stamp script.
The script uses the same syntax as the command COMPDF_StampText

23.8.2013: V3.11.0
- Destinations can now also use floating point zoomvalues /XYZ which are internally multiplied with 100
- improved threading and stability. Tested with below average computer and windows XP.
- nested clipping was not always supported
- updates scrollbar logic for single page mode

28.7.2013: V3.10.2'''
- PDF merge handles embedded objects more effectively
- transparent objects were rendered opaque
- objects were not painted correctly on empty pages

22.7.2013: V3.10.2''
- Fixed: AppendPage caused added draw objects to disappear
- Fixed: AppendPage caused rendered draw objects to disappear
- Fixed: Deleted Pages reappear after AppendPage

18.7.2013: V3.10.2'
- fix problem with embedded fonts where the names were indirectly specified
- fix problem with object selection. Now done in reverse order

16.7.2013: V3.10.2
- Fix the problem that AddImage did not work on a PDF file which was saved before with added images

7.7.2013: V3.10.1
+ pdfMerge now understands the PAGELIST option to only save part of the loaded PDF files
* The AppendPage command produced duplicates on subsequent calls. This problem has been fixed.
* update to CCITT decoding
* update to reader for in PS embedded images

12.6.2013: V3.10.0
+ COMPDF_CopyToClibrd can now copy only the text inside the drawn rectangle when called from OnSelRect event, (see Change the way the mouse works)
+ COMPDF_StampText is now able to draw lines and rectangles. It was also enhanced to easily append lines of text. It is now possible to use relative coordinates to make it easy to pre-create a stamp.
- fix problem with COMPDF_SetSecurityOwner
+ added event OnSelRect to .NET assembly

21.5.2013: V3.9.9
- the .NET assemblies now use strong naming. It is however possible to compile the .NET interface on your own.

14.5.2013: V3.9.8''
- solves a problem with hyperlink detection in combination with crop boxes
- solves a problem with 2 bit images
- Due to a problem in the GDI+ unit few applications were not closed when WPPDFViewerStop was executed.
This problem has been fixed.

23.4.2013: V3.9.8
* DrawObjects are now not deselected when moved. User can now work better with objects.

12.4.2012: V3.9.7'
+ ViewOption wpHideFocusRectThumbnails
+ when writing PDF the PageMode can be set using COMPDF_SetPageMode = 360
+ otherwise the PageMode of first loaded PDF file is preserved and written to new file

10.4.2012: V3.9.7
* the zoom tool (zoom to rectangle) (example) now centers the selected rectangle

- FIX bug: PDF did not load when "Creator" was not used
+ COMPDF_SelectMode can now also set modes for thumbnail view
+ The VCL now defined the function MouseMode for easier access to this feature
* any page selection with mouse is now disabled if the ViewOption does not set wpPageSelection
* ViewOption wpInteractiveThumbnails will only activate the page moving in thumbnail view, not the selection.
* ViewOption wpPageMultiSelection is now required for multi page selection
* ViewOption wpShowPageSelection is used to enable the selection
by keyboard (Shift, Ctrl + Page Up/Down, Home, End)
+ new ViewOption: wpHidePageSelectionThumbnails

7.4.2012: V3.9.6
+ new: COMPDF_GetBookmarkXML - to read current outline tree in XML format
+ new: COMPDF_SetBookmarkXML - to set new outline tree for next save operation
- fix problem with some outlined texts
* when saving PDF files named destinations are now also copied (see COMPDF_SetSaveMode)
- fix problems with scaled signature stamps
* PLUS.SaveTo... will save compressed PDF data also for changed streams
- fix for AddDrawObject function in .NET C# wrapper
- .NET wrapper can now supports "AnyCPU" - it loads 64bit WPViewPDF engine when in 64bit mode
- Text draw objects did not render correctly when TTF DLL was not available

22.3.2012: V3.9.5
+ it is now possible to implement a zoom tool (zoom to rectangle) (see example)
- drawing a rectangle with mouse (frameline) on a rotated PDF page did not show correct position
- copying to bitmap on rotated bitmap did not copy correct area
+ ViewOption wpThumbnailAtozoomToSquareWH. If used, the thumbnails will be sized to make them fit into the window wether they are rotated or not. This helps to avoid change of zoom when pages are rotated in the thumbnail window.
+ COMPDF_ClientToScreenPage, COMPDF_ClientToScreenXY to get screen point corresponding to a PDF page point
+ VCL function: ScreenToPage and PageToScreen
* modified calculation of current page which takes into account how much of a page is being displayed.
- delete selection now removes selection
- selection under program control also updates thumbview
- solve exception when thumbnails were not displayed

18.3.2012: V3.9.4
+ sometimes inline images caused problems - this had been fixed.
* graphics are rendered in higher quality mode
+ possibility to change the zoom level in thumbnail view. Use command COMPDF_ZoomThumbnails, value 1..9 to increase, value -1..-9 do decrease or absolute value.
- in single page mode first page was not automatically displayed after load
- the DLL will now unload quicker

8.3.2012: V3.9.3
- fix problem with font names in PS code
- clipping operations built using many small rectangles did not work
- fix problem with update of scrollbars

27.2.2013: V3.9.2'
* fix problem: on Chinese systems text was sometimes not displayed correctly

27.2.2013: V3.9.2

+ command: COMPDF_SetSaveMode
+ **Singlepage Mode**: COMPDF_SinglepageMode 1 / 0
+ COMPDF_GetHWND can be used to move thumbnail viewer to a different parent panel
- fixed problem that sometimes a page was blank in scroller.
- fixed some problems which occur with threading
- highlighting searched text did not work

17.2.2013: V3.9.0
+ new ViewOption **wpInteractiveThumbnails** to make it possible to select and move pages in thumbnail view.
+ VCL: propertry PopupMenuThumbnails which is used for right click on thumbnails
+ command COMPDF_GetClickElement to check if the x,y position is the viewer or the thumbnail window.
- improved command COMPDF_SaveBMPToClipboard (copy rectangle or complete page)
+ page text can now be exported as simple XML data. Only the tags page, text, table, tr and td are used.
* some optimations to threaded paint

1.2.2013: V3.8.3
* faster display of certain scanned documents
- solves character spacing problem with Chinese text
- solves problem with PDF which use \0 as character code

18.1.2013: V3.8.2
* fix a problem when saving compressed PDF files
+ pdfPrint now supports the option LIMITA3 to force any pages larger than A3 to be scaled down to A3 (Printing (on paper)

7.1.2013: V3.8.1
+ COMPDF_CopyToClibrd, 4 will copy the complete page as bitmap. The resolution can be passed as highword
+ WriteBitmap now understands "clipboard" as file name to create a Bitmap in the clipboard

17.12.2012: V 3.8'
- 32 bit DLLs compiled with different compiler to reduce the DLL size
- text of a few PDF files with undefined fonts was rendered wrong the first time it was displayed.
- fixed a problem with masked images

14.12.2012: V 3.8
**+ added 64bit edition of WPViewPDF. Please use the type1 DLL "wp_type1ttf64.dll" for 64 bit applications.**
(Functionality is the same as 32 bit, except for TIFF support. TIFF support is not possible in 64 bit edition)
* the info dialog now display the version number from the version resource of the engine DLL
+ pdfPrint DLL call understands "DIALOG=1" to display a print dialog
+ show hand point cursor for links
- handling of COLLATE with pdfPrint was improved.

12.12.2012: V 3.07.1
- some signature bitmaps were not visible

6.12.2012: V 3.07.0
- fix for some JBIG2 bitmaps which appeared inverted. (Requires external JBIG2 support)
+ AcroField support now supports Choice Fields (Ch). Writing of appearance streams was improved.
* improved compatibility with certain PDF files which reference string or name properties as objects

23.11.2012: V3.06.9'
- fixes problem with some PDF files which uses compressed XREF tables.
- don't select first page after load operation

20.11.2012: V3.06.9
+ COMPDF_SetSaveMode allows it to remove information from the PDF on next save operation
+ COMPDF_AppendPage=325 can be used to append an empty page to the current view. You can pass the width and height encoded in high and low word of the integer parameter or 0, to use the last page width and height.
+ Improved Prediction code to work around problem in certain scanner files

13.11.2012: V3.06.8'
- do not nest q Q commands in BT ET elements.
* improved rendering code, fixes problem when v, y and re commands were combined
+ new commands: COMPDF_BEGIN_SELECTION = 1300, COMPDF_END_SELECTION = 1301 can be used to wrap selection commands to avoid additional calls the selection change event and redraw.

31.10.2012: V3.06.7
+ command COMPDF_SelectPrintColorMode = 350 - select the color mode for printing. 0=default, 1=monochrome, 2=color
* PLUS: optimation in save method - now faster for certain PDF files which use long strings

16.10.2012: V3.06.6
+ command COMPDF_GetLoadedPortfolio - check if a pdf portfolio was loaded (only dummy page is displayed)
+ command COMPDF_SetProhibitPortfolios - use 1 to disable loading of portfolios
* ViewPDF03.ocx has been updated to allow more than one control in VB6. (ViewerStart must be called with the same DLL path.)

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

# 12 Changes to Version 2

WPViewPDF V3 and V4 are 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 as integer type, 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 V4.0 this events do not work yet:
OnHyperlinkWWW
OnHyperlinkPage
OnError
OnMailMergeGetText

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

e) GetPageText now expects an optional 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. It is possible to create draw objects which trigger the merge event

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

# 13 License



**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.** You may distribute the WPViewPDF3 runtime with Your application if all developers who were working (anywhere) on the project have a license for WPViewPDF 3. If your application is modular and only a few persons work on the PDF viewing part, you still need license for all the developers to have the right to include our component with your application.

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. JBIG2 support requires the DLLs wpdecodejp.dll and, for 64 bit, wpdecodejp64.dll.

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.

**WPViewPDF MakeImage License**

With this license you can convert PDF data from file or stream to a bitmapfile or stream.

# 14 Credits

## 14.1 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 PDF Engine further uses the public zlib, the Independent JPEG Group's JPEG and RC4 routines. It also uses the LZW algorithm for decompression.

## 14.2 LibTIFF Credits

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.

LibTIFF is also used by the tiff-to-pdf functionality of pdfMerge

## 14.3 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.

## 14.4 AES

Advanced Encryption Standard (AES), Delphi implementation

**ElAES**

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
Copyright (c) 2001, EldoS, Alexander Ionov

## 14.5 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.

## 14.6 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
//
```

## 14.7 JPEG 2000

JPX decoded images can be decoded with 32 bit Version of the engine.

uses jpeg2000-for-pascal http://code.google.com/p/pasjpeg2000

Based on openJPEG: http://www.openjpeg.org/

```
 * Copyright (c) 2002-2007, Communications and Remote Sensing Laboratory,
Universite catholique de Louvain (UCL), Belgium
 * Copyright (c) 2002-2007, Professor Benoit Macq
 * Copyright (c) 2001-2003, David Janssens
 * Copyright (c) 2002-2003, Yannick Verschueren
 * Copyright (c) 2003-2007, Francois-Olivier Devaux and Antonin Descampe
 * Copyright (c) 2005, Herve Drolon, FreeImage Team
 * Copyright (c) 2006-2007, Parvatha Elangovan
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS `AS IS'
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED
TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
PURPOSE
```

```
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE COPYRIGHT OWNER OR
CONTRIBUTORS BE
 * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
 * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT
OF
 * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
BUSINESS
 * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
WHETHER IN
 * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR
OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED
OF THE
 * POSSIBILITY OF SUCH DAMAGE.
```

## 14.8 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

## 14.9 JBIG2

The JBIG2 decoding DLL wpdecodejp was built with the help of PDFIUM:



## 14.10 JPEG support

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.