llPDFLib 3.0

Table of Contents

llPDFLib 3.0

1 License Agreement

THIS IS A CONTRACT. YOU SHOULD CAREFULLY READ ALL THE TERMS AND CONDITIONS BEFORE INSTALLING THIS SOFTWARE.

BY INSTALLING THE SOFTWARE, YOU ACCEPT ALL THE TERMS AND CONDITIONS OF THIS LICENSE.

This License Agreement is a legal Agreement between you, an individual or organization, and llionsoft.

Subject

The subject of the present Agreement is "llPDFLib" hereinafter mentioned as Software including the whole delivery package, namely the software itself and the covering documentation.

Accepting or rejecting the Agreement

Any usage of this product including installation of the Software or any of its parts on a computer, loading the product into operating memory (RAM) or permanent storage on a computer's disk or other storage medium, as well as any other type of usage means that you accept all the terms and conditions of the present Agreement. If you do not agree with any statement of this License, you should promptly terminate usage of the product and delete all the files referred to it, as its components, so as the results of its work, from your computer. You should also return all the existing mediums containing the Software to the place where you obtained it.

Copyright

This Software is owned by llionsoft and is protected by international copyright treaties. Any changes to the Software and its components, any additions to it, including the case of running and executing on your computer any software not built-in into your operating system and affecting the Software's operation or changing its results, as well as storage and distribution of such a changed or augmented Software, are strictly forbidden.

Usage

The Software is licensed by the present Agreement to be used for any commercial and other proper purposes.

Users can create application one of the base functionality of which PDF document creation only if they have a "PDF Application" license.

Applications built using our components are royalties free, but if you need to create a development tool (as DCU, DLL, OCX etc..) integrating this functionality you need to obtain a special license.

You acknowledge that the Software in source code form remains a confidential trade secret of llionsoft or its suppliers and therefore you agree not to modify the Software or attempt to decipher, decompile, disassemble or reverse engineer the Software. In order to obtain the source codes you should turn to llionsoft so as to purchase and register the Software's version including source codes.

Group programming projects making use of this software must purchase a copy of the software for each member of the group or purchase a site or enterprise licenses.

As "Producer" of the PDF file always "llPDFLib 3.x" will be written to the created PDF file.

Time restriction

This product has no limits in terms of its usage.

Licensed copies number restriction

Single Developer License is assigned for installation and execution on one computer assembled as a solitary system units and by one user only.

If you have an upgrade to this Software, it constitutes a single product together with the product that you upgraded, and may not be used to increase the total number of licensed installations of the Software.

Group programming projects making use of this software must purchase a copy of the software for each member of the group or purchase a site or enterprise licenses.

Site Developer License is assigned for installation and execution on unlimited numbers of computers by any user inside any buildings of organization located in one city. Any possible delivery of the product or its parts as well as registration codes or access codes you received with purchasing and/or registering this Software to any individual or organization is strictly prohibited.

Enterprise Developer License is assigned for installation and execution on unlimited numbers of computers by any user inside one organization. Any possible delivery of the product or its parts as well as registration codes or access codes you received with purchasing and/or registering this Software to any individual or organization is strictly prohibited.

Distribution restrictions

Except as provided in this License Agreement, you may not transfer, sell, rent, lease, lend, copy, modify, emulate, clone, decompile, translate, sublicense, time-share or electronically transmit or receive the Software or any of its parts or to distribute it in any other way.

Registration codes and access codes

Registration codes (e.g., serial number) and access codes (e.g. archive file password) you received with purchasing and/or registering this Software are not the subject to the

present Agreement. You may store any number of copies of such codes in any form if only you are a registered user and if you obtained them in the registration process or purchasing the Software from llionsoft or authorized persons, individuals or organizations. Any distribution or delivery of the registration and access codes to any third party as well as any form of publication is prohibited. The copyright of all registration and access codes remains the property of llionsoft which reserves the right to withhold or withdraw authorization of use of all registration codes issued to any user if there is reasonable evidence to indicate that the user is involved in a breach of the terms of this License Agreement.

Modifications to the Software

---

llionsoft reserves the right to change the executable code and to remove, add or change the Software's functions.

Warranty and guarantee

This documentation and the VCL are provided "as is" without warranty of any kind, either expressed or implied, including but not limited to the implied warranties of merchantability and/or suitability for a particular purpose. The user assumes the entire risk of any damage caused by this software.

Changing terms and conditions of this Agreement

Any variation to the terms of this Agreement shall only be valid if made and delivered by llionsoft.

If you do not agree to any clause of this Agreement you may ask for explanations from llionsoft. However, this doesn't authorize you to consider this Agreement nullified.

Web site: http://www.llion.net

E-Mail: info@llion.net

2 About llPDFLib

llPDFLib is a library of Object Pascal classes for generating Portable Document Format (PDF) files directly.

A simple pascal program may be linked with the library to produce a standalone executable that can directly generate PDF files without requiring any other tools or applications.

Contrast this with the typical methods of creating PDF files; They generally involve multi-step "work-flows" that require the use of GUI-based applications, tools to merge and customize multiple files and finally produce the PDF output via a PDFWriter print driver.

Alternatively, PDF documents may first be generated in the form of PostScript, which are then processed by Acrobat Distiller (or an equivalent program such as Ghostscript), to convert PostScript into PDF.

Such a multi-stage process is tedious, time consuming and resource intensive because Acrobat Distiller or Ghostscript is a large application that includes a full PostScript interpreter.

Perl or shell script, or AppleScript may be able to automate some of these complex processes, but such a solution is cumbersome and inherently suffers from performance and reliability problems on heavily loaded systems.

By directly producing PDF files in a one-step process, a simple pascal program with a small memory foot-print will be able to achieve the same result much more quickly.

3 Path construction

One of the key concepts of the PDF imaging model is a path which, in itself, is an invisible contour without any markings on the page. Paths must be acted upon by path-painting operators to produce markings. A path may be stroked with a certain color and width, producing an actual curve on the page. A path may also be filled with a color. It may also be used to define a clipping path. Functions in this section are used to construct paths that are subsequently used to provide various effects.

Some functions and procedures of TPDFPage (see page 37) create, modify or delete paths.

4 Path painting

Paths constructed by functions in the "Path construction (see page 5)" section are invisible, i.e., constructing a path does not produce any markings on the page. They must be stroked or filled.

TPDFPage.Clip (see page 39)

TPDFPage.EoClip (see page 41)

TPDFPage.Fill (see page 42)

TPDFPage.EoFill (see page 41)

TPDFPage.FillAndStroke (see page 42)

TPDFPage.EoFillAndStroke (see page 41)

TPDFPage.Stroke (see page 52)

5 Text procedures and functions

Text output is typically created in the following sequence of functions described in this section.

A desired font is set by SetActiveFont. After this, any number of text lines (at any orientation) may be drawn as long as the text lines share the same font attribute.

6 Graphics state procedures

A PDF viewer application maintains an internal data structure called the graphics state that holds current graphic control parameters. These parameters define the global framework within which the graphics operators execute. For example, the Fill operator implicitly uses the current color parameter, and the Stroke operator additionally uses the current line width parameter from the graphics state. The graphics state is initialized at the beginning of each page.

The current graphic state can be managed with the following procedures.

7 Symbol Reference

7.1 Classes

7.1.1 TPDFAction

Class Hierarchy

TObject
  TPDFAction

```
TPDFAction = class(TObject)
```

File

PDF

Description

All interactive operations in PDF documents (jump to page, go to URL, change state of PDF controls etc.) are made possible with the help of actions.

TPDFAction is the base class for all actions.

7.1.2 TPDFActionAnnotation

Class Hierarchy

TObject
  TPDFCustomAnnotation
    TPDFActionAnnotation

```
TPDFActionAnnotation = class(TPDFCustomAnnotation)
```

File

PDF

Description

TPDFActionAnnotation is used to create annotations in PDF Documents.

If a user clicks on an annotation in a PDF document it will execute the action linked to this annotation.

7.1.2.1 TPDFActionAnnotation.Action

```
property Action: TPDFAction;
```

Description

Action executed after click on annotation.

7.1.3 TPDFButton

Class Hierarchy

TObject
TPDFCustomAnnotation
TPDFControl
TPDFButton

```
TPDFButton = class(TPDFControl)
```

File

PDF

Description

TPDFButton creates a button in a PDF document.

A button represents an interactive control on the screen that the user can manipulate with the mouse.

7.1.3.1 TPDFButton.Caption

```
property Caption: string;
```

Description

Specifies a text string that identifies the control to the user.

7.1.4 TPDFCheckBox

Class Hierarchy

TObject
TPDFCustomAnnotation
TPDFControl
TPDFInputControl
TPDFCheckBox

```
TPDFCheckBox = class(TPDFInputControl)
```

File

PDF

Description

TPDFCheckBox is similar to TCheckBox (checkbox toggles between two states, on and off) and creates a checkbox in a PDF Document.

7.1.4.1 TPDFCheckBox.Caption

```
property Caption: string;
```

Description

Specifies a text string that identifies the control to the user.

7.1.4.2 TPDFCheckBox.Checked

```
property Checked: Boolean;
```

Description

Start state of checkbox.

7.1.5 TPDFComboBox

Class Hierarchy

TObject
TPDFCustomAnnotation
TPDFControl
TPDFInputControl
TPDFComboBox

```
TPDFComboBox = class(TPDFInputControl)
```

File

PDF

Description

TPDFComboBox is similar to TComboBox and creates a ComboBox in a PDF Document.

A combo box consisting of a dropdown list optionally accompanied by an editable text box in which the user can type a value other than the predefined choices.

7.1.5.1 TPDFComboBox.EditEnabled

```
property EditEnabled: Boolean;
```

Description

If property is set to true, the combo box includes an editable text box as well as a dropdown list; if false, it includes only a dropdown list.

7.1.5.2 TPDFComboBox.Items

```
property Items: TStringList;
```

Description

Provides access to the list of items (strings) in the list portion of the combo box.

7.1.5.3 TPDFComboBox.Text

```
property Text: string;
```

Description

A text string identifying which of the available options is currently selected.

7.1.6 TPDFControl

Class Hierarchy

TObject
  TPDFCustomAnnotation
    TPDFControl

```
TPDFControl = class(TPDFCustomAnnotation)
```

File

PDF

Description

TPDFControl is the base class for all interactive elements such as buttons, checkboxes, radiobuttons, comboboxes and edit fields.

7.1.6.1 TPDFControl.Color

```
property Color: TColor;
```

Description

Specifies the background color of the control.

7.1.6.2 TPDFControl.Font

```
property Font: TPDFControlFont;
```

Description

Controls the attributes of text written on or in the control.

7.1.6.3 TPDFControl.Hint

```
property Hint: TPDFControlHint;
```

Description

Contains the text string that can appear when the user moves the mouse over the control.

7.1.6.4 TPDFControl.Name

```
property Name: string;
```

Description

Name of field used for export when the PDF document submitted.

7.1.6.5 TPDFControl.OnLostFocus

```
property OnLostFocus: TPDFAction;
```

Description

An action to be performed when the PDF control is "blurred" (loses the input focus).

7.1.6.6 TPDFControl.OnMouseDown

```
property OnMouseDown: TPDFAction;
```

Description

An action to be performed when the mouse button is pressed inside the PDF control's active area.

7.1.6.7 TPDFControl.OnMouseEnter

```
property OnMouseEnter: TPDFAction;
```

Description

An action to be performed when the cursor enters the PDF control's active area.

7.1.6.8 TPDFControl.OnMouseExit

```
property OnMouseExit: TPDFAction;
```

Description

An action to be performed when the cursor exits the PDF control's active area.

7.1.6.9 TPDFControl.OnMouseUp

```
property OnMouseUp: TPDFAction;
```

Description

An action to be performed when the mouse button is released inside the PDF control's active area.

7.1.6.10 TPDFControl.OnSetFocus

```
property OnSetFocus: TPDFAction;
```

Description

An action to be performed when the PDF control receives the input focus.

7.1.6.11 TPDFControl.ReadOnly

```
property ReadOnly: Boolean;
```

Description

If property is set to true, the user may not change the value of the field.

7.1.6.12 TPDFControl.Required

```
property Required: Boolean;
```

Description

If set, the control must have a value at the time it is exported by a submit-form action

7.1.7 TPDFControlFont

Class Hierarchy

TPDFControlFont

```
TPDFControlFont = class
```

File

PDF

Description

Font used for interactive controls.

7.1.7.1 TPDFControlFont.Color

```
property Color: TColor;
```

Description

Specifies the color of the text.

7.1.7.2 TPDFControlFont.Name

```
property Name: string;
```

Description

Identifies the typeface of the font.

7.1.7.3 TPDFControlFont.Size

```
property Size: Integer;
```

Description

Specifies the height of the font in points.

7.1.7.4 TPDFControlFont.Style

```
property Style: TFontStyles;
```

Description

Determines whether the font is normal, italic, underlined, bold etc.

7.1.8 TPDFControlHint

Class Hierarchy

TObject
TPDFControlHint

```
TPDFControlHint = class(TObject)
```

File

PDF

Description

Contains the text string that can appear when the user moves the mouse over a PDF control.

7.1.8.1 TPDFControlHint.Caption

```
property Caption: string;
```

Description

Specified text of the hint.

7.1.8.2 TPDFControlHint.Charset

```
property Charset: TFontCharset;
```

Description

Specifies the character set of the hint.

7.1.9 TPDFControls

Class Hierarchy

TPDFControls

```
TPDFControls = class
```

File

PDF

Description

List of the PDF controls. Used in various actions.

7.1.9.1 TPDFControls.Count

```
property Count: Integer;
```

Description

Count PDF controls in the list.

7.1.9.2 TPDFControls.Items

```
property Items [Index: Integer]: TPDFControl;
```

Description

Lists the PDF controls, referenced by a 0-based index.

7.1.9.3 TPDFControls.Add

```
function Add(Control: TPDFControl): Integer;
```

Description

Add PDF control to list.

7.1.9.4 TPDFControls.Clear

```
procedure Clear;
```

Description

Clear list of PDF Controls.

7.1.9.5 TPDFControls.Delete

```
procedure Delete(Control: TPDFControl);
```

Description

Delete PDF control from list.

7.1.9.6 TPDFControls.IndexOf

```
function IndexOf(Control: TPDFControl): Integer;
```

Description

Return position of PDF control in the list.

7.1.10 TPDFCustomAnnotation

Class Hierarchy

TObject
  TPDFCustomAnnotation

```
TPDFCustomAnnotation = class(TObject)
```

File

PDF

Description

TPDFCustomAnnotation is the base class of all annotations.

7.1.10.1 TPDFCustomAnnotation.BorderColor

```
property BorderColor: TColor;
```

Description

Specifies the border color of the PDF control.

7.1.10.2 TPDFCustomAnnotation.BorderStyle

```
property BorderStyle: string;
```

Description

A dash array defining a pattern of dashes and gaps to be used in drawing a dashed border.

See: TPDFPage.SetDash (see page 48)

7.1.10.3 TPDFCustomAnnotation.Box

```
property Box: TRect;
```

Description

Specifies position of the annotation on the page.

7.1.10.4 TPDFCustomAnnotation.Flags

```
property Flags: TAnnotationFlags;
```

Description

Specify the behaviour of the annotation when printed, rotated etc.

7.1.11 TPDFDocInfo

Class Hierarchy

TPersistent
  TPDFDocInfo

```
TPDFDocInfo = class(TPersistent)
```

File

PDF

Description

A PDF document may include document information containing general information such as the document's title, author, creation and modification dates.

Such global information about the document itself (as opposed to its content or structure) is called metadata and is intended to assist in cataloguing and searching for documents in external databases.

You can set this information with the TPDFDocInfo object.

7.1.11.1 TPDFDocInfo.Author

```
property Author: string;
```

Description

The name of the person who created the PDF document.

7.1.11.2 TPDFDocInfo.CreationDate

```
property CreationDate: TDateTime;
```

Description

The date the PDF document was created.

7.1.11.3 TPDFDocInfo.Creator

```
property Creator: string;
```

Description

If the document was converted to PDF from another format, the name of the application (for example, Adobe FrameMaker®) that created the original document from which it was converted.

7.1.11.4 TPDFDocInfo.Keywords

```
property Keywords: string;
```

Description

Keywords associated with the PDF document.

7.1.11.5 TPDFDocInfo.Subject

```
property Subject: string;
```

Description

The subject of the PDF document.

7.1.11.6 TPDFDocInfo.Title

```
property Title: string;
```

Description

The PDF document's title.

7.1.12 TPDFDocument

Class Hierarchy

TComponent
  TPDFDocument

```
TPDFDocument = class(TComponent)
```

File

PDF

Description

The TPDFDocument component is used to create PDF files.

TPDFDocument has the same interface as TPrinter (BeginDoc (see page 25), EndDoc (see page 26), NewPage (see page 26), Abort (see page 24) methods).

It also has a compatible Canvas (see page 19) property.

The component supports all TrueType fonts. Each font definition can be embedded in the file or not at your discretion.

7.1.12.1 TPDFDocument.Aborted

```
property Aborted: Boolean;
```

Description

This read-only property is TRUE if the generation of the PDF document was aborted using the Abort (see page 24) method.

7.1.12.2 TPDFDocument.AutoCreateURL

```
property AutoCreateURL: Boolean;
```

Description

If set to true, text written to a page via TPDFPage.TextOut (see page 53) will be parsed and if substrings 'http://','mailto:','ftp://' are located a link will be automatically appended to this URL.

7.1.12.3 TPDFDocument.AutoLaunch

```
property AutoLaunch: Boolean;
```

Description

Determines if Adobe Acrobat® should be automatically lauched to view a newly-generated file.

If OutputStream (see page 21) is assigned Adobe Acrobat® is not lauched.

7.1.12.4 TPDFDocument.Canvas

```
property Canvas: TCanvas;
```

Description

Canvas provides a drawing space for objects that must render their own images on CurrentPage (see page 19).

7.1.12.5 TPDFDocument.Compression

```
property Compression: TCompressionType;
```

Description

Determine whether PDF page streams are compressedor not. By using compression the file will be reasonably smaller.

On the other had compression will create binary data rather than ASCII data. While "deflate" produces the smallest files, "run-length" compression is compatible even with very old PDF reader programs.

7.1.12.6 TPDFDocument.CurrentPage

```
property CurrentPage: TPDFPage;
```

Description

This property determines current active page.

7.1.12.7 TPDFDocument.DocumentInfo

```
property DocumentInfo: TPDFDocInfo;
```

Description

Information about PDF document.

7.1.12.8 TPDFDocument.EMFImageAsJpeg

```
property EMFImageAsJpeg: Boolean;
```

Description

Property determine store images from parsed EMF files as JPEG or as bitmap

7.1.12.9 TPDFDocument.EmulateStandardFont

```
property EmulateStandardFont: Boolean;
```

Description

If property set to true "Arial", "Courier New" and "Times Ner Roman" will emulated via standard PDF fonts.

7.1.12.10 TPDFDocument.FileName

```
property FileName: TFileName;
```

Description

This is the filename of the PDF file which should be created.

If OutputStream (see page 21) does not equal nil the PDF document will stored in this stream.

7.1.12.11 TPDFDocument.JPEGQuality

```
property JPEGQuality: Integer;
```

Description

Quality of images which are appended to PDF document are controlled with TImageCompressinType = itcJpeg.

7.1.12.12 TPDFDocument.NonEmbeddedFont

```
property NonEmbeddedFont: TStringList;
```

Description

Specifies a list of fonts not to be embedded.

7.1.12.13 TPDFDocument.OnePass

```
property OnePass: Boolean;
```

Description

Property indicates whether to store previous pages to file after a new page is created.

7.1.12.14 TPDFDocument.OpenDocumentAction

```
property OpenDocumentAction: TPDFAction;
```

Description

An action to be performed when the document is opened.

7.1.12.15 TPDFDocument.Outlines

```
property Outlines: TPDFOutlines;
```

Description

Tree of outlines for current PDF document.

7.1.12.16 TPDFDocument.OutputStream

```
property OutputStream: TStream;
```

Description

If this property is not assigned, the PDF document is written into the file specified by the FileName (see page 20) property.

If OutputStream does not equal nil the PDF document will stored in this stream.

Default value = nil.

7.1.12.17 TPDFDocument.OwnerPassword

```
property OwnerPassword: string;
```

Description

Owner's password for current PDF document.

The password is required to edit an encrypted PDF file.

7.1.12.18 TPDFDocument.Page

```
property Page [Index: Integer]: TPDFPage;
```

Description

List the page references.

Use Page to obtain a pointer to a specific page in the current PDF document. The Index parameter indicates the index of the page, where 0 is the index of the first page, 1 is the index of the second page, and so on.

Use Page with the PageCount (see page 22) property to iterate through all of the pages in the list.

Example:

```
for I := 1 to MyPDF.PageCount - 1 do
  with MyPDF[I] do
  begin
    SetLineWidth(2);
    BeginText;
    SetRGBColor(0.5, 0.5, 0.5);
    SetTextRenderingMode(0);
    SetActiveFont('Times', [fsBold], 12);
    TextOut(Width - 5 - GetTextWidth(IntToStr(I + 1)), Height - 17, 0, IntToStr(I + 1));
```

```
    SetActiveFont('Arial', [fsBold, fsItalic], 10);
    TextOut(5, Height - 15, 0, 'http://www.llion.net');
    SetUrl(Rect(5, Height - 15, round(GetTextWidth('http://www.llion.net')) + 5,
      Height), 'http://www.llion.net');
    EndText;
    NewPath;
    MoveTo(5, Height - 15);
    LineTo(Width - 5, Height - 15);
    Stroke;
  end;
```

7.1.12.19 TPDFDocument.PageCount

```
property PageCount: Integer;
```

Description

Count of created pages in current PDF document.

7.1.12.20 TPDFDocument.PageHeight

```
property PageHeight: Integer;
```

Description

Indicate height of canvas for current page.

7.1.12.21 TPDFDocument.PageLayout

```
property PageLayout: TPageLayout;
```

Description

The page layout to be used when the document is opened.

7.1.12.22 TPDFDocument.PageMode

```
property PageMode: TPageMode;
```

Description

How the document should be displayed when opened.

7.1.12.23 TPDFDocument.PageNumber

```
property PageNumber: Integer;
```

Description

Use PageNumber to find the page number of the current page.

7.1.12.24 TPDFDocument.PageWidth

```
property PageWidth: Integer;
```

Description

Indicate width of canvas for current page

7.1.12.25 TPDFDocument.Printing

```
property Printing: Boolean;
```

Description

Printing is True when the application has called the BeginDoc (see page 25) method but the EndDoc (see page 26) method (or the Abort (see page 24) method) hasn't yet been called.

7.1.12.26 TPDFDocument.ProtectionEnabled

```
property ProtectionEnabled: Boolean;
```

Description

A PDF document can be encrypted to protect its contents from unauthorized access.

Property sets whether to encrypt the current PDF document or not.

7.1.12.27 TPDFDocument.ProtectionKeyLength

```
property ProtectionKeyLength: TPDFKeyLength;
```

Description

Length of the crypto key.

7.1.12.28 TPDFDocument.ProtectionOptions

```
property ProtectionOptions: TPDFCtiptoOptions;
```

Description

A set of flags specifying which operations are permitted when the document is opened with the user password.

7.1.12.29 TPDFDocument.Resolution

```
property Resolution: Integer;
```

Description

Set resolution for pages. Setting this property does not influence already created pages.

Default value:72

7.1.12.30 TPDFDocument.UsedDC

```
property UsedDC: HDC;
```

Description

This property return DC which used for create canvas metafiles of the each page.

7.1.12.31 TPDFDocument.UserPassword

```
property UserPassword: string;
```

Description

User password for current PDF document.

This is the password which will be used to encrypt the file.

7.1.12.32 TPDFDocument.UseScreenDC

```
property UseScreenDC: Boolean;
```

Description

If property set to true canvas metafile will created with use screen DC else metafile will created with DC of the first printer in system.

7.1.12.33 TPDFDocument.Version

```
property Version: TPDFVersion;
```

Description

Determine version of the created PDF document.

7.1.12.34 TPDFDocument.ViewerPreferences

```
property ViewerPreferences: TViewerPreferences;
```

Description

This property controls how the document is presented on the screen.

7.1.12.35 TPDFDocument.WaterMark

```
property WaterMark [Index: Integer]: TPDFPage;
```

Description

List of the created watermarks, referenced by a 0-based index.

7.1.12.36 TPDFDocument.Abort

```
procedure Abort;
```

Description

The procedure will clear all internal data and break generation of the PDF document.

7.1.12.37 TPDFDocument.AddImage

```
function AddImage(FileName: TFileName, Compression: TImageCompressionType, MaskIndex:
Integer = -1, IsMask: Boolean = False, TransparentColor: TColor = -1): Integer; overload;
```

Description

The function appends an image to a PDF document and returns the index of the image.

Example:

```
K := MyPDF.AddImage('hmb.jpg', itcjpeg);
with MyPDF.CurrentPage do
begin
  ShowImage(K, (Width - 200) / 2, 30, 200, 200, 0);
  for I := 1 to 20 do
    ShowImage(K, 200 + I * 10, 400 + I * 10, (21 - I) * 10, (21 - I) * 10, I * 10);
end;
```

7.1.12.38 TPDFDocument.AddImage

```
function AddImage(Image: TGraphic, Compression: TImageCompressionType, MaskIndex: Integer =
-1, IsMask: Boolean = False, TransparentColor: TColor = -1): Integer; overload;
```

Description

The function appends an image to a PDF document and returns the index of the image.

Example:

```
K := MyPDF.AddImage('hmb.jpg', itcjpeg);
with MyPDF.CurrentPage do
begin
  ShowImage(K, (Width - 200) / 2, 30, 200, 200, 0);
  for I := 1 to 20 do
    ShowImage(K, 200 + I * 10, 400 + I * 10, (21 - I) * 10, (21 - I) * 10, I * 10);
end;
```

7.1.12.39 TPDFDocument.AddJSFunction

```
procedure AddJSFunction(AName: string, AParams: string, ABody: string);
```

Description

Method append JavaScript function to PDF document for next use this function in actions...

7.1.12.40 TPDFDocument.BeginDoc

```
procedure BeginDoc;
```

Description

This procedure initializes creation of the PDF file. Calling the procedure sets CurrentPage (see page 19) to first page of the PDF document.

7.1.12.41 TPDFDocument.CreateAction

```
function CreateAction(CClass: TPDFActionClass): TPDFAction;
```

Description

Function create object of "CClass" class.

Example:

```
HV := PDF.CreateAction(TPDFVisibeControlAction) as TPDFVisibeControlAction;
```

7.1.12.42 TPDFDocument.CreateWaterMark

```
function CreateWaterMark: Integer;
```

Description

Function creates a watermak and returns its ID.

7.1.12.43 TPDFDocument.EndDoc

```
procedure EndDoc;
```

Description

Generate the PDF document.

7.1.12.44 TPDFDocument.GetCurrentPageIndex

```
function GetCurrentPageIndex: Integer;
```

Description

Returns Index of current page.

7.1.12.45 TPDFDocument.NewPage

```
procedure NewPage;
```

Description

Creates a new page in the PDF document and sets CurrentPage (see page 19) property to new generated page.

7.1.12.46 TPDFDocument.SetCurrentPage

```
procedure SetCurrentPage(Index: Integer);
```

Description

This function sets CurrentPage (see page 19) to page with number "Index".

7.1.13 TPDFEdit

Class Hierarchy

TObject
  TPDFCustomAnnotation
    TPDFControl
      TPDFInputControl
        TPDFEdit

```
TPDFEdit = class(TPDFInputControl)
```

File

PDF

Description

TPDFEdit is similar to TEdit and creates a control which accepts input in a PDF document.

7.1.13.1 TPDFEdit.IsPassword

```
property IsPassword: Boolean;
```

Description

If set, the field is intended for entering a secure password that should not be echoed visibly to the screen.

Characters typed from the keyboard should instead be echoed in some unreadable form, such as asterisks or bullet characters.

7.1.13.2 TPDFEdit.Justification

```
property Justification: THorJust;
```

Description

Justification of input text.

7.1.13.3 TPDFEdit.MaxLength

```
property MaxLength: Integer;
```

Description

The maximum length of the field's text, in characters.

7.1.13.4 TPDFEdit.Multiline

```
property Multiline: Boolean;
```

Description

If property is set to true, the TPDFEdit (see page 26) may contain multiple lines of text; if false, the field's text is restricted to a single line.

7.1.13.5 TPDFEdit.ShowBorder

```
property ShowBorder: Boolean;
```

Description

If property is set to false the controls border is invisible.

7.1.13.6 TPDFEdit.Text

```
property Text: string;
```

Description

Text displayed in PDF Edit control when file is created.

7.1.14 TPDFException

Class Hierarchy

Exception
  TPDFException

```
TPDFException = class(Exception)
```

File

PDF

Description

TPDFException is the exception class for errors that occur when creating PDF document

7.1.15 TPDFGoToPageAction

Class Hierarchy

TObject
  TPDFAction
    TPDFGoToPageAction

```
TPDFGoToPageAction = class(TPDFAction)
```

File

PDF

Description

A GoToPage action changes the view to a specified destination (page, location).

7.1.15.1 TPDFGoToPageAction.PageIndex

```
property PageIndex: Integer;
```

Description

Index of the page which will be displayed after the action is executed.

7.1.15.2 TPDFGoToPageAction.TopOffset

```
property TopOffset: Integer;
```

Description

Offset from top of the page which is displayed when action is executed.

7.1.16 TPDFGoToRemoteAction

Class Hierarchy

TObject
  TPDFAction
    TPDFGoToRemoteAction

```
TPDFGoToRemoteAction = class(TPDFAction)
```

File

PDF

Description

A GoToRemote action changes the view to a specified PDF document (filename, page).

7.1.16.1 TPDFGoToRemoteAction.Document

```
property Document: string;
```

Description

Name of the document which will opened after execute this action

7.1.16.2 TPDFGoToRemoteAction.InNewWindow

```
property InNewWindow: Boolean;
```

Description

A ?ag specifying whether to open the destination document in a new window. If this ?ag is false, the destination document will replace the current document in the same window. If this entry is absent, the viewer application should behave in accordance with the current user preference.

7.1.16.3 TPDFGoToRemoteAction.PageIndex

```
property PageIndex: Integer;
```

Description

Index of the page which will be displayed after the action is executed.

7.1.17 TPDFImportDataAction

Class Hierarchy

TObject
  TPDFAction
    TPDFImportDataAction

```
TPDFImportDataAction = class(TPDFAction)
```

File

PDF

Description

An import-data action imports Forms Data Format (FDF) data into a documents interactive form from a specified file.

The structure of an FDF file can be found in the Adobe PDF Reference.

7.1.17.1 TPDFImportDataAction.FileName

```
property FileName: string;
```

Description

The FDF file from which to import the data.

7.1.18 TPDFInputControl

Class Hierarchy

TObject
TPDFCustomAnnotation
TPDFControl
TPDFInputControl

```
TPDFInputControl = class(TPDFControl)
```

File

PDF

Description

TPDFInputControl is the base class of all PDFControls where a user can input / change data.

7.1.18.1 TPDFInputControl.OnBeforeFormatting

```
property OnBeforeFormatting: TPDFJavaScriptAction;
```

Description

A JavaScript action to be performed before the field is formatted to display its current value. This allows the field's value to be modified before formatting.

7.1.18.2 TPDFInputControl.OnChange

```
property OnChange: TPDFJavaScriptAction;
```

Description

A JavaScript action to be performed when the field's value is changed. This allows the new value to be checked for validity.

7.1.18.3 TPDFInputControl.OnKeyPress

```
property OnKeyPress: TPDFJavaScriptAction;
```

Description

A JavaScript action to be performed when the user types a keystroke into a text or combo box or modifies the selection in a scrollable list.

This allows the keystroke to be checked for validity and rejected or modified.

7.1.18.4 TPDFInputControl.OnOtherControlChanged

```
property OnOtherControlChanged: TPDFJavaScriptAction;
```

Description

A JavaScript action to be performed when the value of another field changes, in order to recalculate the value of this field.

7.1.19 TPDFJavaScriptAction

Class Hierarchy

TObject
  TPDFAction
    TPDFJavaScriptAction

```
TPDFJavaScriptAction = class(TPDFAction)
```

File

PDF

Description

A JavaScript (see page 31) action causes a script to be compiled and executed by the JavaScript (see page 31) interpreter.

Depending on the nature of the script, this can cause various interactive form fields in the document to update their values or change their visual appearance.

Adobe Technical Note #5186, "Acrobat Forms JavaScript (see page 31) Object Specification" give details on the contents and effects of JavaScript (see page 31) scripts.

7.1.19.1 TPDFJavaScriptAction.JavaScript

```
property JavaScript: string;
```

Description

A string containing the JavaScript script to be executed.

7.1.20 TPDFOutlineNode

Class Hierarchy

TObject
  TPDFOutlineNode

```
TPDFOutlineNode = class(TObject)
```

File

PDF

Description

A PDF document may optionally display a document outline on the screen, allowing the user to navigate interactively from one part of the document to another.

The outline consists of a tree-structured hierarchy of outline items (sometimes called bookmarks), which serves as a "visual table of contents" to display the document's structure to the user.

The user can interactively open and close individual items by clicking them with the mouse. When an item is open, its immediate children in the hierarchy become visible on the screen; each child may in turn be opened or closed, selectively revealing or hiding further parts of the hierarchy. When an item is closed, all of its descendants in the hierarchy are hidden. Clicking the text of any visible item with the mouse activates the item, causing the viewer application to jump to a destination associated with the item.

The TPDFOutlineNode object stores destinations associated with the outline item.

7.1.20.1 TPDFOutlineNode.Action

```
property Action: TPDFAction;
```

Description

Action which will execute after puch on current node.

7.1.20.2 TPDFOutlineNode.Charset

```
property Charset: TFontCharset;
```

Description

This property determines the charset of the Title (see page 33) property. If Charset is not equal to ANSI_CHARSET then the Title (see page 33) is information stored in a PDF document as unicode strings to support national languages.

Property can be one of the following constants: ANSI_CHARSET, EASTEUROPE_CHARSET, RUSSIAN_CHARSET, GREEK_CHARSET , TURKISH_CHARSET, BALTIC_CHARSET, SHIFTJIS_CHARSET, HANGEUL_CHARSET, GB2312_CHARSET, CHINESEBIG5_CHARSET.

7.1.20.3 TPDFOutlineNode.Color

```
property Color: TColor;
```

Description

Color of the node text. Available only for 1.4 version.

7.1.20.4 TPDFOutlineNode.Count

```
property Count: Integer;
```

Description

Indicates the number of direct descendants of a node. Count includes only immediate children, and not their descendants.

7.1.20.5 TPDFOutlineNode.Expanded

```
property Expanded: Boolean;
```

Description

Determine whether node is expanded when PDF document is opened.

7.1.20.6 TPDFOutlineNode.HasChildren

```
property HasChildren: Boolean;
```

Description

Indicates whether a node has any children. HasChildren is True if the node has subnodes, or False if the node has no subnodes.

7.1.20.7 TPDFOutlineNode.Item

```
property Item [Index: Integer]: TPDFOutlineNode;
```

Description

Provides access to a child node by its position in the list of child nodes. Use Item to access a child node based on its Index property. The first child node has an index of 0, the second an index of 1, and so on.

7.1.20.8 TPDFOutlineNode.Style

```
property Style: TFontStyles;
```

Description

Style of the node text. Availablo only for 1.4 version.

7.1.20.9 TPDFOutlineNode.Title

```
property Title: string;
```

Description

The text to be displayed on the screen for this item.

7.1.20.10 TPDFOutlineNode.Delete

```
procedure Delete;
```

Description

Destroys the node and all its children. Use the Delete method to delete a node and free all associated memory.

7.1.20.11 TPDFOutlineNode.DeleteChildren

```
procedure DeleteChildren;
```

Description

Procedure Delete (see page 33) all children of current node.

7.1.20.12 TPDFOutlineNode.GetFirstChild

```
function GetFirstChild: TPDFOutlineNode;
```

Description

Returns the first child node of a node. Call GetFirstChild to access the first child node of the view node. If the node has no children, GetFirstChild returns nil.

7.1.20.13 TPDFOutlineNode.GetLastChild

```
function GetLastChild: TPDFOutlineNode;
```

Description

Returns the last immediate child node of the calling node. Call GetLastChild to find the last immediate child of a node. If the

calling node has no children, GetLastChild returns nil.

7.1.20.14 TPDFOutlineNode.GetNext

```
function GetNext: TPDFOutlineNode;
```

Description

Returns the next node after the calling node in the tree of the outline nodes.

If the calling node is the last node, GetNext returns nil. It will return the next node including child nodes. To get the next node at the same level as the calling node, use GetNextSibling (see page 34).

7.1.20.15 TPDFOutlineNode.GetNextChild

```
function GetNextChild(Node: TPDFOutlineNode): TPDFOutlineNode;
```

Description

Returns the next child node after Node. Call GetNextChild to locate the next node in the list of immediate children of the tree of the outline nodes. If the calling node has no children or there is no node after Node, GetNextChild returns nil.

7.1.20.16 TPDFOutlineNode.GetNextSibling

```
function GetNextSibling: TPDFOutlineNode;
```

Description

Returns the next node in the tree of the outline nodes at the same level as the calling node. To find the next node in the tree including child nodes, use GetNext (see page 34).

7.1.20.17 TPDFOutlineNode.GetPrev

```
function GetPrev: TPDFOutlineNode;
```

Description

Returns the previous node in the tree of the outline nodes before the calling node.

7.1.20.18 TPDFOutlineNode.GetPrevChild

```
function GetPrevChild(Node: TPDFOutlineNode): TPDFOutlineNode;
```

Description

Returns the previous child node before Node. Call GetPrevChild to locate the previous node in the list of immediate children of the tree node. If the calling node has no children or there is no node before Node, GetPrevChild returns nil.

7.1.20.19 TPDFOutlineNode.GetPrevSibling

```
function GetPrevSibling: TPDFOutlineNode;
```

Description

Returns the previous node before the calling node and at the same level.

7.1.21 TPDFOutlines

Class Hierarchy

TObject
  TPDFOutlines

```
TPDFOutlines = class(TObject)
```

File

PDF

Description

A PDF document may optionally display a document outline on the screen, allowing the user to navigate interactively from one part of the document to another.

The outline consists of a tree-structured hierarchy of outline items (sometimes called bookmarks), which serves as a "visual table of contents" to display the document's structure to the user.

The user can interactively open and close individual items by clicking them with the mouse. When an item is open, its immediate children in the hierarchy become visible on the screen; each child may in turn be opened or closed, selectively revealing or hiding further parts of the hierarchy. When an item is closed, all of its descendants in the hierarchy are hidden. Clicking the text of any visible item with the mouse activates the item, causing the viewer application to jump to a destination associated with the item.

TPDFOutlines maintains a list of outline nodes in the outline tree. Nodes can be added, deleted and inserted in the tree of the outlines.

7.1.21.1 TPDFOutlines.Count

```
property Count: Integer;
```

Description

Indicates the number of nodes maintained by the TPDFOutlines (see page 35) object.

7.1.21.2 TPDFOutlines.Item

```
property Item [Index: Integer]: TPDFOutlineNode;
```

Description

Lists all tree nodes managed by the TPDFOutlines (see page 35) object.

7.1.21.3 TPDFOutlines.Add

```
function Add(Node: TPDFOutlineNode, Title: string, Action: TPDFAction, Charset:
TFontCharset = ANSI_CHARSET): TPDFOutlineNode; overload;
```

Description

Adds a new tree node to a tree of the outline items. The node is added as the last sibling of the Node parameter. Function returns the node that has been added.

Action will execute when user clicks on created node.

7.1.21.4 TPDFOutlines.AddChild

```
function AddChild(Node: TPDFOutlineNode, Title: string, Action: TPDFAction, Charset:
TFontCharset = ANSI_CHARSET): TPDFOutlineNode; overload;
```

Description

Adds a new tree node to a tree of the outline items.The node is added as a child of the node specified by the Node parameter. It is added to the end of Node's list of child nodes. Function returns the node that has been added.

Action will execute when user clicks on created node.

7.1.21.5 TPDFOutlines.AddChildFirst

```
function AddChildFirst(Node: TPDFOutlineNode, Title: string, Action: TPDFAction, Charset:
TFontCharset = ANSI_CHARSET): TPDFOutlineNode; overload;
```

Description

Adds a new tree node to a tree of the outline items. Use AddChildFirst to insert (see page 37) a node as the first child of the node specified by the Node parameter. Function returns the node that has been added.

Action will execute when user clicks on created node.

7.1.21.6 TPDFOutlines.AddFirst

```
function AddFirst(Node: TPDFOutlineNode, Title: string, Action: TPDFAction, Charset:
TFontCharset = ANSI_CHARSET): TPDFOutlineNode; overload;
```

Description

Adds a new tree node to a tree of the outline items.The node is added as the first sibling of the node specified by the Node parameter. Function returns the node that has been added.

Action will execute when user clicks on created node.

7.1.21.7 TPDFOutlines.Clear

```
procedure Clear;
```

Description

Deletes all tree nodes contained from the list managed by TPDFOutlines (see page 35).

7.1.21.8 TPDFOutlines.Delete

```
procedure Delete(Node: TPDFOutlineNode);
```

Description

Removes a node from the tree of the outline items.

7.1.21.9 TPDFOutlines.GetFirstNode

```
function GetFirstNode: TPDFOutlineNode;
```

Description

Returns the first tree node in the tree of the outline items.

7.1.21.10 TPDFOutlines.Insert

```
function Insert(Node: TPDFOutlineNode, Title: string, Action: TPDFAction, Charset:
TFontCharset = ANSI_CHARSET): TPDFOutlineNode; overload;
```

Description

Inserts a tree node into the tree of the outline items before the node specified by the Node parameter.

Function returns the node that has been added.

Action will execute when user clicks on created node.

7.1.22 TPDFPage

Class Hierarchy

TObject
  TPDFPage

```
TPDFPage = class(TObject)
```

File

PDF

Description

This class consist of information about one page of the PDF document.

The class supports drawing and filling a variety of shapes and lines, writing text and rendering graphic images.

7.1.22.1 TPDFPage.Height

```
property Height: Integer;
```

Description

Height of the page.

7.1.22.2 TPDFPage.Orientation

```
property Orientation: TPDFPageOrientation;
```

Description

Use Orientation to determine if a page is landscape or portrait.

7.1.22.3 TPDFPage.PageRotate

```
property PageRotate: TPDFPageRotate;
```

Description

Determines number of degrees by which the page should be rotated clockwise when displayed or printed.

7.1.22.4 TPDFPage.Size

```
property Size: TPDFPageSize;
```

Description

Determines size of the page in standard size pages.

Example:

```
MyPDF.CurrentPage.Size:=psLetter;
```

7.1.22.5 TPDFPage.Thumbnail

```
property Thumbnail: Integer;
```

Description

A PDF document may define thumbnail images representing the contents of its pages in miniature form. A viewer application can then display these images on the screen, allowing the user to navigate to a page by clicking its thumbnail image with the mouse.

Property indicates what image will show as a thumbnail. (Value for this property is returned from the TPDFDocument.AddImage (see page 24) function)

7.1.22.6 TPDFPage.WaterMark

```
property WaterMark: Integer;
```

Description

This property indicates the watermak of the current page (Value of WaterMark is returned by TPDFDocument.CreateWatermark (see page 26) function).

7.1.22.7 TPDFPage.Width

```
property Width: Integer;
```

Description

Width of the page.

7.1.22.8 TPDFPage.Arc

```
function Arc(X1: Extended, Y1: Extended, x2: Extended, y2: Extended, BegAngle: Extended,
EndAngle: Extended): TDoublePoint; overload;
```

Description

Use Arc to create an elliptically curved path. The arc traverses the perimeter of an ellipse (see page 40) that is bounded by the points (X1,Y1) and (X2,Y2). The arc is drawn following the perimeter of the ellipse (see page 40), counterclockwise, from the starting point to the ending point. The starting point is defined by the intersection of the ellipse (see page 40) and a line defined by BegAngle and EndAngle, specified in degrees.

7.1.22.9 TPDFPage.Arc

```
function Arc(X1: Extended, Y1: Extended, x2: Extended, y2: Extended, x3: Extended, y3:
Extended, x4: Extended, y4: Extended): TDoublePoint; overload;
```

Description

Use Arc to create an elliptically curved path. The arc traverses the perimeter of an ellipse (see page 40) that is bounded by the points (X1,Y1) and (X2,Y2). The arc is drawn following the perimeter of the ellipse (see page 40), counterclockwise, from the starting point to the ending point. The starting point is defined by the intersection of the ellipse (see page 40) and a line defined by the center of the ellipse (see page 40) and (X3,Y3). The ending point is defined by the intersection of the ellipse (see page 40) and a line defined by the center of the ellipse (see page 40) and (X4, Y4).

7.1.22.10 TPDFPage.Circle

```
procedure Circle(X: Extended, Y: Extended, R: Extended);
```

Description

Advanced graphic operation

This procedure creates a circular path centered at (x, y) with radius "r" in the counter-clock-wise direction.

If you need a circle drawn in the clock-wise direction, please use Arc (see page 38)(x, y, x+r,y+r, 360.0); ClosePath (see page 40);.

This function performs a move to angle 0 (right edge) of the circle. Current point will also be at the same location after the call.

7.1.22.11 TPDFPage.Clip

```
procedure Clip;
```

Description

Path operation

This procedure install the current paths as the boundary for clipping subsequent drawing. The use of the clip operator may require some care, because clip and eoclip (see page 41) operators do not consume the current path.

Also note that there is no practical way of removing a clipping path, except by "GStateRestore (see page 43)"-ing a graphical state before clipping is imposed.

Example:

```
with MyPDF.CurrentPage do
begin
  GStateSave;
  NewPath;
  Rectangle( x, y,x+width,y+height);
  Clip;
  NewPath;
  GStateRestore;
end;
```

See: Path painting (see page 6)

7.1.22.12 TPDFPage.ClosePath

```
procedure ClosePath;
```

Description

This closes a path by connecting the first and the last point in the path currently being constructed. Call to this procedure is often needed to avoid a notch in a stroked path, and to make "line join" work correctly in joining the first and the last points.

7.1.22.13 TPDFPage.Comment

```
procedure Comment(st: string);
```

Description

Insert comment into page stream.

7.1.22.14 TPDFPage.CreateControl

```
function CreateControl(CClass: TPDFControlClass, ControlName: string, Box: TRect):
TPDFControl;
```

Description

Function creates object of "CClass" class.

ControlName is name of PDF control.

Box specifies position of the PDF control on the page.

Example:

```
Btn1 := CreateControl(TPDFButton, 'Btn1', Rect(20, 90, 65, 105)) as TPDFButton;
```

7.1.22.15 TPDFPage.Curveto

```
procedure Curveto(X1: Extended, Y1: Extended, X2: Extended, Y2: Extended, X3: Extended, Y3:
Extended);
```

Description

This procedure adds a Bezier cubic curve segment to the path starting at the current point as (x0, y0), using two points (x1, y1) and (x2, y2) as control points, and terminating at point (x3, y3). The new current point will be (x3, y3). If there is no current point, an error will result.

See: Path construction (see page 5)

7.1.22.16 TPDFPage.Ellipse

```
procedure Ellipse(X1: Extended, Y1: Extended, X2: Extended, Y2: Extended);
```

Description

This procedure create an ellipse path specified by top left point at pixel coordinates (X1, Y1) and the bottom right point at

(X2, Y2) in the counter-clock-wise direction. If you need a ellipse drawn in the clock-wise direction, please use Arc (see page 38)(x1, y1, x2,y2, 360.0); ClosePath (see page 40); This function performs a move to angle 0 (right edge) of the circle (see page 39). Current point will also be at the same location after the call.

See: Path construction (see page 5)

7.1.22.17 TPDFPage.EoClip

```
procedure EoClip;
```

Description

This procedure installs the current paths as the boundary for clipping subsequent drawing and uses the "even-odd" rule for defining the "inside" that shows through the clipping window. The use of the clip (see page 39) operator may require some care, because clip (see page 39) and eoclip operators do not consume the current path. Also note that there is no practical way of removing a clipping path, except for by "GStateRestore (see page 43)"-ing a graphical state before clipping is imposed.

Example:

```
with MyPDF.CurrentPage do
begin
  GStateSave;
  NewPath;
  Rectangle( x, y,x+width,y+height);
  Clip;
  NewPath;
  GStateRestore;
end;
```

See: Path painting (see page 6)

7.1.22.18 TPDFPage.EoFill

```
procedure EoFill;
```

Description

This procedure uses the current path as the boundary for color filling and uses the "evenodd" rule for defining an "inside" that is painted.

7.1.22.19 TPDFPage.EoFillAndStroke

```
procedure EoFillAndStroke;
```

Description

This function is used to first fill (see page 42) the inside with the current fill (see page 42) color, and then stroking the path with the current stroke (see page 52) color. PDF's graphics state maintains separate colors for fill (see page 42) and stroke (see page 52) operations, thus these combined operators are made available.

See: Path painting (see page 6), TPDFPage.EoFill (see page 41)

7.1.22.20 TPDFPage.ExtTextOut

```
procedure ExtTextOut(X: Extended, Y: Extended, Orientation: Extended, TextStr: string, Dx: PExt);
```

Description

Extended version of TPDFPage.TextOut (see page 53)

Last parameter is pointer to array of width between characters.

7.1.22.21 TPDFPage.ExtUnicodeTextOut

```
procedure ExtUnicodeTextOut(X: Extended, Y: Extended, Orientation: Extended, Text: PWord, Len: Integer, DX: PExt);
```

Description

Unicode version of TPDFPage.ExtTextOut (see page 42)

Last parameter is pointer to array of width between characters.

7.1.22.22 TPDFPage.Fill

```
procedure Fill;
```

Description

This procedure uses the current path as the boundary for color filling and uses the "non-zero winding number" rule.

See: Path painting (see page 6)

7.1.22.23 TPDFPage.FillAndStroke

```
procedure FillAndStroke;
```

Description

This function is used to first fill (see page 42) the inside with the current fill (see page 42) color, and then stroking the path with the current stroke (see page 52) color. PDF's graphics state maintains separate colors for fill (see page 42) and stroke (see page 52) operations, thus these combined operators are made available.

See: Path painting (see page 6), TPDFPage.Fill (see page 42)

7.1.22.24 TPDFPage.GetCurrentFontSize

```
function GetCurrentFontSize: Extended;
```

Description

Return size of the current font in points.

7.1.22.25 TPDFPage.GetTextRowCount

```
function GetTextRowCount(BoxWidth: Integer, TextStr: string): Integer;
```

Description

Return count of the rows need for display string in the box with BoxWidth

7.1.22.26 TPDFPage.GetTextWidth

```
function GetTextWidth(Text: string): Extended;
```

Description

This function return size of the 'Text' in points for the active font. SetActiveFont (see page 47) must be called before calling this function.

See: Text procedures and functions (see page 7)

7.1.22.27 TPDFPage.GetUnicodeTextRowCount

```
function GetUnicodeTextRowCount(BoxWidth: Integer, Text: PWord, Len: Integer): Integer;
```

Description

Unicode version of the TPDFPage.GetTextRowCount (see page 43)

7.1.22.28 TPDFPage.GetUnicodeWidth

```
function GetUnicodeWidth(Text: PWord, Len: Integer): Extended;
```

Description

Unicode version of TPDFPage.GetTextWidth (see page 43)

7.1.22.29 TPDFPage.GStateRestore

```
procedure GStateRestore;
```

Description

Graphic state operation

A well-structured PDF document typically contains many graphical elements that are essentially independent of each other and sometimes nested to multiple levels. The graphics state stack allows these elements to make local changes to the graphics state without disturbing the graphics state of the surrounding environment.

This procedure restores the entire graphics state to its former value by popping it from the stack.

See: Graphics state procedures (see page 8)

7.1.22.30 TPDFPage.GStateSave

```
procedure GStateSave;
```

Description

A well-structured PDF document typically contains many graphical elements that are essentially independent of each other and sometimes nested to multiple levels. The graphics state stack allows these elements to make local changes to the graphics state without disturbing the graphics state of the surrounding environment.

This procedure pushes a copy of the entire graphics state onto the stack.

See: Graphics state procedures (see page 8)

7.1.22.31 TPDFPage.LineTo

```
procedure LineTo(X: Extended, Y: Extended);
```

Description

Graphic primitive operation

This procedure adds a line segment to the path, starting at the current point and ending at point (x, y). Current point set to (x,y).

See: Path construction (see page 5)

7.1.22.32 TPDFPage.MoveTo

```
procedure MoveTo(X: Extended, Y: Extended);
```

Description

This procedure moves the current point to the location specified by (x, y).

See:Path construction (see page 5)

7.1.22.33 TPDFPage.NewPath

```
procedure NewPath;
```

Description

Clears the current path. Current point becomes undefined.

See: Path construction (see page 5)

7.1.22.34 TPDFPage.NoDash

```
procedure NoDash;
```

Description

This procedure resets the dash pattern back to none, i.e., solid line.

See: Graphics state procedures (see page 8)

7.1.22.35 TPDFPage.Pie

```
procedure Pie(X1: Extended, Y1: Extended, x2: Extended, y2: Extended, BegAngle: Extended,
EndAngle: Extended); overload;
```

Description

Use Pie to append a pie-shaped wedge on the path. The wedge is defined by the ellipse (see page 40) bounded by the rectangle (see page 46) determined by the points (X1, Y1) and X2, Y2). The section drawn is determined by BegAngle and EndAngle, specified in degrees.

Current point is center of the wedge.

See: Path construction (see page 5)

7.1.22.36 TPDFPage.Pie

```
procedure Pie(X1: Extended, Y1: Extended, x2: Extended, y2: Extended, x3: Extended, y3:
Extended, x4: Extended, y4: Extended); overload;
```

Description

Use Pie to append a pie-shaped wedge on the path. The wedge is defined by the ellipse (see page 40) bounded by the rectangle (see page 46) determined by the points (X1, Y1) and X2, Y2). The section drawn is determined by two lines radiating from the center of the ellipse (see page 40) through the points (X3, Y3) and (X4, Y4)

Current point is center of the wedge.

See: Path construction (see page 5)

7.1.22.37 TPDFPage.PlayMetaFile

```
procedure PlayMetaFile(MF: TMetaFile); overload;
```

Description

Procedure paints context of the metafile on the page.

7.1.22.38 TPDFPage.PlayMetaFile

```
procedure PlayMetaFile(MF: TMetafile, x: Extended, y: Extended, XScale: Extended, YScale:
Extended); overload;
```

Description

Procedure paints context of the metafile on the page.

x,y - top level coner for paint

xscale (see page 46), yscale - scale (see page 46) of height and width when painting on the page.

7.1.22.39 TPDFPage.Rectangle

```
procedure Rectangle(X1: Extended, Y1: Extended, X2: Extended, Y2: Extended);
```

Description

This function draws a rectangle with one corner at (x1, y1) and second at (x2,y2).

See:Path construction (see page 5)

7.1.22.40 TPDFPage.RectRotated

```
procedure RectRotated(X: Extended, Y: Extended, W: Extended, H: Extended, Angle: Extended);
```

Description

This function draws a rectangle (see page 46) of size (w, h) with one corner at (x,y), with an orientation argument, angle, specified in degrees.

See: Path construction (see page 5)

7.1.22.41 TPDFPage.Rotate

```
procedure Rotate(Angle: Extended);
```

Description

This procedure rotates the coordinate system by the angle given in degrees (positive is clock wise).

See: Graphics state procedures (see page 8)

7.1.22.42 TPDFPage.RoundRect

```
procedure RoundRect(X1: Integer, Y1: Integer, X2: Integer, Y2: Integer, X3: Integer, Y3:
Integer);
```

Description

Add a rectangle (see page 46) with rounded corners to path. The rectangle (see page 46) will have edges defined by the points (X1,Y1), (X2,Y1), (X2,Y2), (X1,Y2), but the corners will be shaved to create a rounded appearance. The curve of the rounded corners matches the curvature of an ellipse (see page 40) with width X3 and height Y3

See: Path construction (see page 5)

7.1.22.43 TPDFPage.Scale

```
procedure Scale(SX: Extended, SY: Extended);
```

Description

Transform operation

This procedure scales the coordinate system by scaling factors supplied for X and Y dimensions.

See: Graphics state procedures (see page 8)

7.1.22.44 TPDFPage.SetAction

```
function SetAction(ARect: TRect, Action: TPDFAction): TPDFCustomAnnotation;
```

Description

Procedure creates object of TPDFCustomAnnotation (see page 16).

7.1.22.45 TPDFPage.SetActiveFont

```
procedure SetActiveFont(FontName: string, FontStyle: TFontStyles, FontSize: Extended,
FontCharset: TFontCharset = ANSI_CHARSET);
```

Description

Text operation

This procedure sets the active font for text operations.

llPDFLib emulates fsUnderLine and fsStrikeOut style. If the font does not have fsBold or fsItalic style then llPDFLib will emulate it as well.

See: Text procedures and functions (see page 7)

7.1.22.46 TPDFPage.SetAnnotation

```
function SetAnnotation(ARect: TRect, Title: string, Text: string, Color: TColor, Flags:
TAnnotationFlags, Opened: Boolean, Charset: TFontCharset = ANSI_CHARSET):
TPDFCustomAnnotation;
```

Description

Annotation

This function inserts a text annotation into the current page in area bounded by Rect. Strings Title and Text sets the title and content of the annotation.

Color will be used for the following purposes:

- The background of the annotation's icon when closed
- The title bar of the annotation's pop-up window

If Charset not equal to ANSI_CHARSET then Title and Text information is stored in the PDF document as unicode string to support national languages.

Charset can be one of the following constants:ANSI_CHARSET, EASTEUROPE_CHARSET, RUSSIAN_CHARSET, GREEK_CHARSET , TURKISH_CHARSET, SHIFTJIS_CHARSET, HANGEUL_CHARSET, GB2312_CHARSET, CHINESEBIG5_CHARSET and BALTIC_CHARSET.

7.1.22.47 TPDFPage.SetCharacterSpacing

```
procedure SetCharacterSpacing(Spacing: Extended);
```

Description

This procedure sets the additional space (in points) that should be inserted between characters.

See: Text procedures and functions (see page 7)

7.1.22.48 TPDFPage.SetCMYKColor

```
procedure SetCMYKColor(C: Extended, M: Extended, Y: Extended, K: Extended);
```

Description

This procedure set both stroke (see page 52) and fill (see page 42) colors to the color values specified.

Note: All arguments will take values in the range [0 .. 1].

7.1.22.49 TPDFPage.SetCMYKColorFill

```
procedure SetCMYKColorFill(C: Extended, M: Extended, Y: Extended, K: Extended);
```

Description

This procedure set the fill (see page 42) color to the values specified.

Note: All arguments will take values in the range [0 .. 1].

7.1.22.50 TPDFPage.SetCMYKColorStroke

```
procedure SetCMYKColorStroke(C: Extended, M: Extended, Y: Extended, K: Extended);
```

Description

This procedure set the stroke (see page 52) color to the values specified.

Note: All arguments will take values in the range [0 .. 1].

7.1.22.51 TPDFPage.SetDash

```
procedure SetDash(DashSpec: string);
```

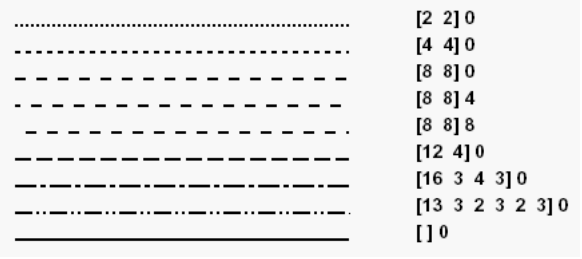
Description

The line dash pattern controls the pattern of dashes and gaps used to stroke (see page 52) paths.

Before beginning to stroke (see page 52) a path, the dash array is cycled through, adding up the lengths of dashes and gaps. When the accumulated length equals the value specified by the dash phase, stroking of the path begins, using the dash array cyclically from that point onward.

See: Graphics state procedures (see page 8)

Example:



7.1.22.52 TPDFPage.SetFlat

```
procedure SetFlat(FlatNess: integer);
```

Description

The flatness tolerance controls the maximum permitted distance in device pixels between the mathematically correct path and an approximation constructed from straight line segments.

7.1.22.53 TPDFPage.SetGray

```
procedure SetGray(Gray: Extended);
```

Description

This procedure sets both stroke (see page 52) and fill (see page 42) colors to the gray value specified.

Note: All arguments will take values in the range [0 .. 1].

7.1.22.54 TPDFPage.SetGrayFill

```
procedure SetGrayFill(Gray: Extended);
```

Description

This procedure sets fill (see page 42) colors to the gray value specified.

Note: All arguments will take values in the range [0 .. 1].

7.1.22.55 TPDFPage.SetGrayStroke

```
procedure SetGrayStroke(Gray: Extended);
```

Description

This procedure sets stroke (see page 52) colors to the gray value specified.

Note: All arguments will take values in the range [0 .. 1].

7.1.22.56 TPDFPage.SetHorizontalScaling

```
procedure SetHorizontalScaling(Scale: Extended);
```

Description

This procedure sets the horizontal scaling factor in a percentage. This essentially expands or compresses the horizontal dimension of the string. The default value for this parameter is 100 (%).

See: Text procedures and functions (see page 7)

7.1.22.57 TPDFPage.SetLineCap

```
procedure SetLineCap(LineCap: TPDFLineCap);
```

Description

Other

The line cap style specifies the shape to be used at the ends of open subpaths (and dashes, if any) when they are stroked.

7.1.22.58 TPDFPage.SetLineJoin

```
procedure SetLineJoin(LineJoin: TPDFLineJoin);
```

Description

The line join style specifies the shape to be used at the corners of paths that are stroked.

7.1.22.59 TPDFPage.SetLineWidth

```
procedure SetLineWidth(lw: Extended);
```

Description

This procedure sets the current linewidth to the value specified in points

See: Graphics state procedures (see page 8)

7.1.22.60 TPDFPage.SetLinkToPage

```
function SetLinkToPage(ARect: TRect, PageIndex: Integer, TopOffset: Integer):
TPDFCustomAnnotation;
```

Description

This procedure places an active hyper-link area for the rectangle (see page 46) bounded by Rect. Destination is a page of the current PDF document with 'PageIndex' index.

Page with 'PageIndex' index must be created on time of call this procedure.

"Top" parameter depends on the location of the display window on that page (offset from top of the page).

7.1.22.61 TPDFPage.SetMiterLimit

```
procedure SetMiterLimit(MiterLimit: Extended);
```

Description

The miter limit imposes a maximum on the ratio of the miter length to the line width. When the limit is exceeded, the join is converted from a miter to a bevel.

See: Graphics state procedures (see page 8)

7.1.22.62 TPDFPage.SetRGBColor

```
procedure SetRGBColor(R: Extended, G: Extended, B: Extended);
```

Description

This procedure sets both stroke (see page 52) and fill (see page 42) colors to the color values specified.

Note: All arguments will take values in the range [0 .. 1].

7.1.22.63 TPDFPage.SetRGBColorFill

```
procedure SetRGBColorFill(R: Extended, G: Extended, B: Extended);
```

Description

This procedure sets the fill (see page 42) color to the values specified.

Note: All arguments will take values in the range [0 .. 1].

7.1.22.64 TPDFPage.SetRGBColorStroke

```
procedure SetRGBColorStroke(R: Extended, G: Extended, B: Extended);
```

Description

This procedure sets the stroke (see page 52) color to the values specified.

Note: All arguments will take values in the range [0 .. 1].

7.1.22.65 TPDFPage.SetTextRenderingMode

```
procedure SetTextRenderingMode(Mode: integer);
```

Description

This procedure sets the mode that determines how the character outline is used. By default, the character outline is used for filling operations by which inside of the outline path is painted solidly with the current fill (see page 42) color. This may be changed by calling this function.

Mode Description

0 Fill (see page 42) text.

1 Stroke (see page 52) text.

2 Fill (see page 42), then stroke (see page 52), text.

3 Neither fill (see page 42) nor stroke (see page 52) text (invisible).

4 Fill (see page 42) text and add to path for clipping.

5 Stroke (see page 52) text and add to path for clipping.

6 Fill (see page 42), then stroke (see page 52), text and add to path for clipping.

7 Add text to path for clipping.

7.1.22.66 TPDFPage.SetUrl

```
function SetUrl(ARect: TRect, URL: string): TPDFCustomAnnotation;
```

Description

This procedure places an active hyper-link area for the rectangle (see page 46) bounded by Rect.

The "URL" should be a string specifying a standard URL (Uniform Resource Locator) used in World Wide Web address specification (e.g., "http://www.llion.net/" or "mailto:team@llion.net").

7.1.22.67 TPDFPage.SetWordSpacing

```
procedure SetWordSpacing(Spacing: Extended);
```

Description

This procedure sets the additional space (in points) that should be inserted between words, i.e., for every space character found in the text string.

See: Text procedures and functions (see page 7)

7.1.22.68 TPDFPage.ShowImage

```
procedure ShowImage(ImageIndex: Integer, x: Extended, y: Extended, w: Extended, h:
Extended, angle: Extended);
```

Description

Picture

This procedure place the image data of size (w, h) with one corner at (x,y), and angle, specified in degrees (ImageIndex is returned by the TPDFDocument.AddImage (see page 24) function) into the current content stream for the page

7.1.22.69 TPDFPage.Stroke

```
procedure Stroke;
```

Description

This function strokes the current paths by the current stroke color and current linewidth.

See: Path painting (see page 6)

7.1.22.70 TPDFPage.TextBox

```
procedure TextBox(Rect: TRect, Text: string, Hor: THorJust, Vert: TVertJust);
```

Description

Show 'Text' string inside area bounded Rect with horizontal justify (Hor) and vertical justify (Vert).

See: Text procedures and functions (see page 7)

7.1.22.71 TPDFPage.TextFromBaseLine

```
procedure TextFromBaseLine(BaseLine: Boolean);
```

Description

Out text from baseline of the font.

7.1.22.72 TPDFPage.TextOut

```
procedure TextOut(X: Extended, Y: Extended, Orientation: Extended, TextStr: string);
```

Description

{$IFDEF CB} procedure TextOutput(X, Y, Orientation (see page 37): Extended; TextStr: string;); {$ELSE}

This procedure draws a text line "textstr" using the current font starting at location (x, y) at "orientation" (in degrees).

See: Text procedures and functions

7.1.22.73 TPDFPage.TextOutBox

```
function TextOutBox(LTCornX: integer, LTCornY: integer, Interval: integer, BoxWidth:
integer, BoxHeight: integer, TextStr: string): integer;
```

Description

Out text in the box.

LTCornX, LTCornY : top left coner of the box

BoxWidth, BoxHeight : width and height of the box

Interval: interval between rows

TextStr :string for out

7.1.22.74 TPDFPage.Translate

```
procedure Translate(XT: Extended, YT: Extended);
```

Description

This procedure shifts the origin of the coordinate system by the (xt, yt) specified.

7.1.22.75 TPDFPage.UnicodeTextOut

```
procedure UnicodeTextOut(X: Extended, Y: Extended, Orientation: Extended, Text: PWord, Len: Integer);
```

Description

ENDIF

7.1.22.76 TPDFPage.UnicodeTextOutBox

```
function UnicodeTextOutBox(LTCornX: integer, LTCornY: integer, Interval: integer, BoxWidth: integer, BoxHeight: integer, Text: PWord, Len: Integer): integer;
```

Description

Unicode version of TPDFPage.TextOutBox (see page 53)

7.1.23 TPDFRadioButton

Class Hierarchy

TObject
  TPDFCustomAnnotation
    TPDFControl
      TPDFInputControl
        TPDFRadioButton

```
TPDFRadioButton = class(TPDFInputControl)
```

File

PDF

Description

TPDFRadioButton is similar to TRadioButton and creates a radiobutton in a PDF document.

To create a set of radiobuttons create TPDFRadiobuttons with equal names and differences export values.

7.1.23.1 TPDFRadioButton.Checked

```
property Checked: Boolean;
```

Description

Start state of radiobutton.

7.1.23.2 TPDFRadioButton.ExportValue

```
property ExportValue: string;
```

Description

Value which will be sent to a URL when PDF document is submited

Must have only alphabet and numeric characters.

7.1.24 TPDFResetFormAction

Class Hierarchy

TObject
  TPDFAction
    TPDFResetFormAction

```
TPDFResetFormAction = class(TPDFAction)
```

File

PDF

Description

A reset-form action resets selected PDF controls to their default values. (Default value is the value of the control at time of PDF document creation).

7.1.24.1 TPDFResetFormAction.ResetFields

```
property ResetFields: TPDFControls;
```

Description

List of the fields which will be reset after action is executed.

If list is empty will reset all controls.

7.1.25 TPDFSubmitFormAction

Class Hierarchy

TObject
  TPDFAction
    TPDFSubmitFormAction

```
TPDFSubmitFormAction = class(TPDFAction)
```

File

PDF

Description

A submit-form action transmits the names and values of selected PDF Controls to a specified uniform resource locator (URL (see page 56)), typically the address of a World Wide Web server that will process them and send back a response.

7.1.25.1 TPDFSubmitFormAction.IncludeFields

```
property IncludeFields: TPDFControls;
```

Description

List of the fields which will submited after action is executed.

If list is empty will submit all controls.

7.1.25.2 TPDFSubmitFormAction.SendEmpty

```
property SendEmpty: Boolean;
```

Description

If this property is set to false the name of the controls without names will not be submited, else will send only names of the controls to URL (see page 56).

7.1.25.3 TPDFSubmitFormAction.SubmitType

```
property SubmitType: TPDFSubmitType;
```

Description

This property sets the type of submit.

7.1.25.4 TPDFSubmitFormAction.URL

```
property URL: string;
```

Description

Uniform resource locator (URL) of the script at the Web server that will process the submission.

7.1.26 TPDFTextAnnotation

Class Hierarchy

TObject
  TPDFCustomAnnotation
    TPDFTextAnnotation

```
TPDFTextAnnotation = class(TPDFCustomAnnotation)
```

File

PDF

Description

A text annotation represents a "sticky note" attached to a point in the PDF document.

When closed, the annotation appears as an icon; when open, it displays a pop-up window containing the text of the note, in a font and size chosen by the viewer application.

7.1.26.1 TPDFTextAnnotation.Caption

```
property Caption: string;
```

Description

Property specifies title of the annotation.

7.1.26.2 TPDFTextAnnotation.Charset

```
property Charset: TFontCharset;
```

Description

Property specifies charset of the text annotation.

7.1.26.3 TPDFTextAnnotation.Opened

```
property Opened: Boolean;
```

Description

A property specifying whether the annotation should initially be displayed open.

7.1.26.4 TPDFTextAnnotation.Text

```
property Text: string;
```

Description

The text to be displayed in the pop-up window when the annotation is opened. Carriage returns may be used to separate the text into paragraphs.

7.1.26.5 TPDFTextAnnotation.TextAnnotationIcon

```
property TextAnnotationIcon: TTextAnnotationIcon;
```

Description

Determine icon for closed annotation.

7.1.27 TPDFURLAction

Class Hierarchy

TObject
  TPDFAction
    TPDFURLAction

```
TPDFURLAction = class(TPDFAction)
```

File

PDF

Description

A uniform resource locator (URL (see page 57)) is a string that identifies (resolves to) a resource on the Internet—typically a file that is the destination of a hypertext link, although it can also resolve to a query or other entity.

A URL (see page 57) action causes a URL (see page 57) to be resolved.

7.1.27.1 TPDFURLAction.URL

```
property URL: string;
```

Description

Property specifies URL to be resolved after action is executed.

7.1.28 TPDFVisibeControlAction

Class Hierarchy

TObject
  TPDFAction
    TPDFVisibeControlAction

```
TPDFVisibeControlAction = class(TPDFAction)
```

File

PDF

Description

A hide action hides or shows one or more PDF controls on the screen by setting or clearing their Hidden flags.

7.1.28.1 TPDFVisibeControlAction.Controls

```
property Controls: TPDFControls;
```

Description

List of the PDF controls which have their Hidden flags changed.

7.1.28.2 TPDFVisibeControlAction.Visible

```
property Visible: Boolean;
```

Description

State which Hidden flags are set to.

7.2 Structs, Records, Enums

7.2.1 TAnnotationFlag

```
TAnnotationFlag = (afInvisible, afHidden, afPrint, afNoZoom, afNoRotate, afNoView,
afReadOnly);
```

File

PDF

Members

Members	Description
afInvisible	If set, do not display the annotation if it does not belong to one of the standard annotation types and no annotation handler is available.
afHidden	If set, do not display or print the annotation or allow it to interact with the user, regardless of its annotation type or whether an annotation handler is available.
afPrint	If set, print the annotation when the page is printed. If clear, never print the annotation, regardless of whether it is displayed on the screen. This can be useful, for example, for annotations representing interactive pushbuttons, which would serve no meaningful purpose on the printed page.
afNoZoom	If set, do not scale the annotation's appearance to match the magnification of the page. The location of the annotation on the page (defined by the upper-left corner of its bounding box) remains fixed, regardless of the page magnification.

afNoRotate	If set, do not rotate the annotation's appearance to match the rotation of the page. The upper-left corner of the annotation's bounding box remains in a fixed location on the page, regardless of the page rotation.
afNoView	If set, do not display the annotation on the screen or allow it to interact with the user. The annotation may be printed (depending on the setting of the Print flag), but should be considered hidden for purposes of onscreen display and user interaction.
afReadOnly	If set, do not allow the annotation to interact with the user. The annotation may be displayed or printed (depending on the settings of the NoView and Print flags), but should not respond to mouse clicks or change its appearance in response to mouse motions.

Description

Flags specifying various characteristics of the annotation.

7.2.2 TCompressionType

```
TCompressionType = (ctNone, ctFlate);
```

File

PDF

Members

Members	Description
ctNone	Not compress contents of the pages
ctFlate	Compress contents of the pages with deflate compression

Description

Determine whether PDF page streams are compressedor not.

7.2.3 THorJust

```
THorJust = (hjLeft, hjCenter, hjRight);
```

File

PDF

Members

Members	Description
hjLeft	Aling text relatively left of the box
hjCenter	Aling text relatively center of the box
hjRight	Aling text relatively right of the box

Description

Determinate horizontal align of the text.

7.2.4 TImageCompressionType

```
TImageCompressionType = (itcFlate, itcJpeg, itcCCITT3, itcCCITT32d, itcCCITT4);
```

File

PDF

Members

Members	Description
itcFlate	Flate compression (Used for b/w and color images)
itcJpeg	Jpeg compression (Used for b/w and color images)

itcCCITT3	CCITT 3 compression (Used for b/w images only)
itcCCITT32d	CCITT 3 (2D) compression (Used for b/w images only)
itcCCITT4	CCITT 4 compression (Used for b/w images only)

Description

Method which used for compress images in PDF document

7.2.5 TPageLayout

```
TPageLayout = (plSinglePage, plOneColumn, plTwoColumnLeft, plTwoColumnRight);
```

File

PDF

Members

Members	Description
plSinglePage	Display one page at a time.
plOneColumn	Display the pages in one column.
plTwoColumnLeft	Display the pages in two columns, with odd-numbered pages on the left.
plTwoColumnRight	Display the pages in two columns, with odd-numbered pages on the right.

Description

Specifying the page layout to be used when the document is opened

7.2.6 TPageMode

```
TPageMode = (pmUseNone, pmUseOutlines, pmUseThumbs, pmFullScreen);
```

File

PDF

Members

Members	Description
pmUseNone	Neither document outline nor thumbnail images visible
pmUseOutlines	Document outline visible
pmUseThumbs	Thumbnail images visible
pmFullScreen	Full-screen mode, with no menu bar, window controls, or any other window visible

Description

Specifying how the document should be displayed when opened.

7.2.7 TPDFCriptoOption

```
TPDFCriptoOption = (coPrint, coModifyStructure, coCopyInformation, coModifyAnnotation,
coPrintHi, coFillAnnotation, coExtractInfo, coAssemble);
```

File

PDF

Members

Members	Description
coPrint	Print the document
coModifyStructure	Modify the contents of the document
coCopyInformation	Copy or otherwise extract text and graphics from the document, including extracting text and graphics (in support of accessibility to disabled users or for other purposes).
coModifyAnnotation	Add or modify text annotations, fill in interactive form fields.
coPrintHi	Print the document with high quality
coFillAnnotation	Fill in existing interactive form fields (including signature fields)
coExtractInfo	Extract text and graphics
coAssemble	Assemble the document (insert, rotate, or delete pages and create bookmarks or thumbnail images)

Description

Encrypted PDF document may be restricted from certain operations.

7.2.8 TPDFKeyLength

```
TPDFKeyLength = (kl40, kl128);
```

File

PDF

Members

Members	Description
kl40	Length of the key is 40 bit
kl128	Length of the key is 128 bit

Description

Determinate length of the key for crypted documents.

7.2.9 TPDFLineCap

```
TPDFLineCap = (lcButtEnd, lcRound, lcProjectingSquare);
```

File

PDF

Members

Members	Description
lcButtEnd	The stroke is squared off at the endpoint of the path. There is no projection beyond the end of the path.
lcRound	A semicircular arc with a diameter equal to the line width is drawn around the endpoint and filled in.
lcProjectingSquare	The stroke continues beyond the endpoint of the path for a distance equal to half the line width and is then squared off.

Description

The line cap style specifies the shape to be used at the ends of open subpaths (and dashes, if any) when they are stroked.

7.2.10 TPDFLineJoin

```
TPDFLineJoin = (ljMiter, ljRound, ljBevel);
```

File

PDF

Members

Members	Description
`ljMiter`	The outer edges of the strokes for the two segments are extended until they meet at an angle, as in a picture frame. If the segments meet at too sharp an angle (as defined by the miter limit parameter— see "Miter Limit," below), a bevel join is used instead.
`ljRound`	A circle with a diameter equal to the line width is drawn around the point where the two segments meet and is filled in, producing a rounded corner.
`ljBevel`	The two segments are finished with butt caps (see TPDFLineCap (see page 61)) and the resulting notch beyond the ends of the segments is filled with a triangle

Description

The line join style specifies the shape to be used at the corners of paths that are stroked.

7.2.11 TPDFPageOrientation

```
TPDFPageOrientation = (poPagePortrait, poPageLandscape);
```

File

PDF

Members

Members	Description
`poPagePortrait`	Portrait Orientation
`poPageLandscape`	Landscape Orientation

Description

Determinate orientation of the page.

7.2.12 TPDFPageRotate

```
TPDFPageRotate = (pr0, pr90, pr180, pr270);
```

File

PDF

Members

Members	Description
`pr0`	0 Degree
`pr90`	90 Degree
`pr180`	180 Degree
`pr270`	270 Degree

Description

Rotate page in PDF document viewer.

7.2.13 TPDFPageSize

```
TPDFPageSize = (psUserDefined, psLetter, psA4, psA3, psLegal, psB5, psC5, ps8x11, psB4,
psA5, psFolio, psExecutive, psEnvB4, psEnvB5, psEnvC6, psEnvDL, psEnvMonarch, psEnv9,
psEnv10, psEnv11);
```

File

PDF

Description

Determinate size of the pages.

7.2.14 TPDFSubmitType

```
TPDFSubmitType = (stGet, stPost, stFDF);
```

File

PDF

Members

Members	Description
stGet	Control names and values will submited using an HTTP GET request.
stPost	Control names and values will submited using an HTTP POST request.
stFDF	Control names and values will submited in Forms Data Format (FDF).

Description

Type of submit.

7.2.15 TPDFVersion

```
TPDFVersion = (v13, v14);
```

File

PDF

Members

Members	Description
v13	Version 1.3
v14	Version 1.4

Description

Determinate version of the created PDF document

7.2.16 TVertJust

```
TVertJust = (vjUp, vjCenter, vjDown);
```

File

PDF

Members

Members	Description
vjUp	Aling text relatively top of the box
vjCenter	Aling text relatively center of the box
vjDown	Aling text relatively bottom of the box

Description

Determinate vertical align of the text.

7.2.17 TViewerPreference

```
TViewerPreference = (vpHideToolBar, vpHideMenuBar, vpHideWindowUI, vpFitWindow,
vpCenterWindow);
```

File

PDF

Members

Members	Description
vpHideToolBar	A flag specifying whether to hide the viewer application's tool bars when the document is active.
vpHideMenuBar	A flag specifying whether to hide the viewer application's menu bar when the document is active.
vpHideWindowUI	A flag specifying whether to hide user interface elements in the document's window (such as scroll bars and navigation controls), leaving only the document's contents displayed.
vpFitWindow	A flag specifying whether to resize the document's window to fit the size of the first displayed page.
vpCenterWindow	A flag specifying whether to position the document's window in the center of the screen.

Description

Determinate how the document is presented on the screen.

7.3 Types

7.3.1 TAnnotationFlags

```
TAnnotationFlags = set of TAnnotationFlag;
```

File

PDF

Description

This is type TAnnotationFlags.

7.3.2 TPDFActionClass

```
TPDFActionClass = class of TPDFAction;
```

File

PDF

Description

This is type TPDFActionClass.

7.3.3 TPDFControlClass

```
TPDFControlClass = class of TPDFControl;
```

File

PDF

Description

This is type TPDFControlClass.

7.3.4 TPDFCtiptoOptions

```
TPDFCtiptoOptions = set of TPDFCriptoOption;
```

File

PDF

Description

This is type TPDFCtiptoOptions.

7.3.5 TViewerPreferences

```
TViewerPreferences = set of TViewerPreference;
```

File

PDF

Description

This is type TViewerPreferences.

Index