Coherent PDF Toolkit for .NET

Developer's Manual

Version 1.5 (March 2010)

Coherent Graphics Ltd

For bug reports, feature requests and comments, email
contact@coherentgraphics.co.uk

Contents

1 Installation

The Coherent PDF .NET Toolkit is supplied as a Windows Installer, which installs reference copies of the DLLs and the manual in the Program Files Folder, installs the DLLs into the .NET GAC (Global Assembly Cache) and adds appropriate entries in the Registry.

1.1 Installation

Double click the .msi file and follow the instructions to install. Reference copies of the DLLs and the manual are placed in `Program Files/Coherent Graphics/Cpdflib 1.5/`.

The toolkit can be uninstalled by using "Add or Remove Programs" in the Windows Control Panel.

1.2 Visual Studio Setup

Use the "Add Reference" facility to add references in a project to:

- Coherent.PDFTools.dll
- FSharp.Core.dll

The "Add Reference" facility is located differently in each version of Visual Studio - see the documentation.

The toolkit is now ready for use.

2 Basic Usage

Function Summary

`FromFile`	Load PDF from file
`FromFileLazy`	Load PDF from file lazily
`FromFileDecrypt`	Load a PDF from file and decrypt it
`BlankDocument`	Make a blank document given page dimensions
`BlankDocumentPaper`	Make a blank document given named paper size
`IsEncrypted`	Check whether a PDF is encrypted
`DecryptPdf`	Decrypt a PDF using the user password
`DecryptPdfOwner`	Decrypt a PDF using the owner password
`ToFile`	Write a PDF to file
`ToFileEncrypted`	Write a PDF to file, encrypting it
`Pages`	Count the number of pages in a PDF
`All`	Build the page range representing all pages in a PDF
`Range`	Build a page subrange
`Even`	Build the page range of all odd pages
`Odd`	Build the page range of all even pages
`Difference`	Build the difference of two ranges
`RemoveDuplicates`	Remove duplicates from a range
`Sort`	Sort a range

The Coherent PDF Toolkit for .NET provides a wide range of facilities for modifying PDF files created by other means. There is a single library `Coherent.PDFTools.dll`. The top-level module contaning the main functions is `Cpdflib`. We use these terms interchangeably.

The rest of this manual describes the functions available in that library.

2.1 Reading PDF from Files and Memory

The type of pdf files in memory is `Pdf.pdfdoc`. To load a PDF from file, assuming it is unencrypted, call `FromFile`, giving the file name as a `string`:

Opening a File

C#
```
Pdf.pdfdoc pdf = Cpdflib.FromFile("in.pdf")
```

VB
```
Dim pdf As Pdf.pdfdoc = Cpdflib.FromFile("in.pdf")
```

(If the file is encrypted, but only with a blank user password, it will be decrypted. This is identical to Acrobat's behaviour.)

If the file is encrypted, you must decrypt it to load it fully. This requires either the user or owner password.

Opening an Encrypted File

C#
```
Pdf.pdfdoc pdf = Cpdflib.FromFileDecrypt("in.pdf", "pw")
```

VB
```
Dim pdf As Pdf.pdfdoc = Cpdflib.FromFileDecrypt("in.pdf", "pw")
```

If the operations to be performed on the PDF are limited to reading some metadata, or accessing just a portion of the file, it can be more memory- and time-efficient not to load or parse all the PDF upon opening. The only caveat is that the file must be available until the PDF object is destroyed - writing to or removing that file will result in an error.

Opening an File Lazily

C#
```
Pdf.pdfdoc pdf = Cpdflib.FromFileLazy("in.pdf")
```

VB
```
Dim pdf As Pdf.pdfdoc = Cpdflib.FromFileLazy("in.pdf")
```

No decryption is performed here (even if the password is blank), since it would involve loading the whole file, so you must explicitly check if the file is encrypted using `IsEncrypted` and decrypt it before performing any operation which requires the file to be in a decrypted state.

2.2 Creating a Blank Document

PDF Documents can also be created from scratch. To build a 20-page blank document with pages of width 500pts and height 600pts:

Making a Blank Document Given Paper Dimensions

C#
```
Pdf.pdfdoc pdf = Cpdflib.BlankDocument(500.0, 600.0, 20)
```

VB
```
Dim pdf As Pdf.pdfdoc = Cpdflib.BlankDocument(500.0, 600.0, 20)
```

To build a 20-page blank document using the US Letter paper size.

Making a Blank Document Given a Named Paper Size

C#
```
Pdf.pdfdoc pdf =
  Cpdflib.BlankDocumentPaper(Cpdflib.usletterportrait, 20)
```

VB
```
Dim pdf As Pdf.pdfdoc =
  Cpdflib.BlankDocumentPaper(Cpdflib.usletterportrait, 20)
```

Here are the standard paper sizes:

```
a0portrait        a1portrait         a2portrait
a3portrait        a4portrait         a5portrait
a0landscape       a1landscape        a2landscape
a3landscape       a4landscape        a5landscape
usletterportrait  usletterlandscape
uslegalportrait   uslegallandscape
```

2.3 Writing PDF to Files and Memory

Writing a PDF to file is achieved with the `ToFile` function. The arguments are the PDF, the filename and two booleans - the first is true if the file is to be linearized, the second if a new /ID is to be made for the file.

Writing a File

C#
```
Cpdflib.ToFile(pdf, "out.pdf", false, true)
```

VB
```
Cpdflib.ToFile(pdf, "out.pdf", false, true)
```

(In these examples, the file is not linearized, but a new /ID is made - hence `false, true`.)

To write a file encrypted, use `ToFileEncrypted`.

Encryption Methods

```
Pdfwrite.encryption_method.PDF40bit
Pdfwrite.encryption_method.PDF128bit
Pdfwrite.encryption_method.AES128bit(false)
Pdfwrite.encryption_method.AES128bit(true)
```

Here are the possible permissions:

Using any form of encryption:

`Pdfcrypt.NoEdit`	Cannot change the document
`Pdfcrypt.NoPrint`	Cannot print the document
`Pdfcrypt.NoCopy`	Cannot select or copy text or graphics
`Pdfcrypt.NoAnnot`	Cannot add or change form fields or annotations

Using 128 bit or AES encryption only:

`Pdfcrypt.NoForms`	Cannot edit form fields
`Pdfcrypt.NoExtract`	Cannot extract text or graphics
`Pdfcrypt.NoAssemble`	Cannot merge files etc.
`Pdfcrypt.NoHqPrint`	Cannot print high-quality

So to write a file encypted, we pass the PDF object, an encryption method, an array of permissions, an owner password, user password, linearize flag and the filename.

Writing a File Encrypted

C#

```
Pdfcrypt.permission[] permissions =
 {Pdfcrypt.permission.NoEdit,
  Pdfcrypt.permission.NoAssemble};
Cpdflib.ToFileEncrypted
  (pdf,                                                  Document
   Pdfwrite.encryption_method.AES128bit(false),          Encryption
   permissions,                                          Permissions
   "fred",                                               Owner Password
   "charles",                                            User Password
   false,                                                Linearize
   false,                                                must be false
   "C:\\output.pdf");                                    Output file
```

VB

```
Dim permissions As Pdfcrypt.permission () =
 {Pdfcrypt.permission.NoEdit,
  Pdfcrypt.permission.NoAssemble};
Cpdflib.ToFileEncrypted
  (pdf,                                                  Document
   Pdfwrite.encryption_method.AES128bit(false),          Encryption
   permissions,                                          Permissions
   "fred",                                               Owner Password
   "charles",                                            User Password
   false,                                                Linearize
   false,                                                must be false
   "C:\\output.pdf");                                    Output file
```

2.4 Ranges

A *range* is an array of integers representing page numbers, commonly passed to `Cpdflib` functions. Sometimes it is treated as an unordered set, representing pages to be affected by a command, sometimes as an ordered list representing pages to be merged, for example.

All these functions return a new range, leaving inputs unaltered:

`All(pdf)`	All pages in the document, in order.
`Range(n, m)`	Pages from `n` to `m` inclusive, in order.
`Even(r)`	Second, fourth etc. pages in a range.
`Odd(r)`	First, third etc. pages in a range.
`Union(r, s)`	Pages in either range, sans duplicates, unordered.
`Difference(r, s)`	Pages in `r` which are not in `s`. Preserves order.
`RemoveDuplicates(r)`	Range without duplicates. Preserves order.
`Sort(r)`	Sort the range.

2.5 Exceptions

`Cpdflib` functions can raise one exception: `Cpdflib_error` which carries a string describing the error.

2.6 Units of Measure

The standard unit of measure in `Cpdflib` is the PDF Point, which is 1/72 of an inch. The following conversion functions are available, each taking a `double` and returning a `double`.

`PtOfCm`	Convert centimeters to points
`PtOfMm`	Convert millimeters to points
`PtOfIn`	Convert inches to points
`CmOfPt`	Convert points to centimeters
`MmOfPt`	Convert points to millimeters
`InOfPt`	Convert points to inches

2.7 Making Pages Upright

There are two kinds of rotation in PDF documents - a *viewing rotation* which can be set, making a PDF viewer such as Acrobat Reader open the docment with its pages appearing to be rotated by 90, 180 or 270 degrees, and the *actual rotation* which is how the text and graphics on a page are layed out (portrait or landscape).

The `Cpdflib` functions `Rotate` and `RotateBy` set or alter the viewing rotation, and the function `RotateContents` alters the actual rotation. The function `Upright` changes the viewing rotation to 0 and, if necessary counterrotates the actual rotation to compensate. This is useful because many `Cpdflib` functions require upright pages to work as expected.

3 Merging and Splitting

Function Summary

`MergeSimple`	Merge two or more PDFs into a single document
`Merge`	Merge PDFs, given extra options
`MergeSame`	Merge PDFs when the same PDF appears twice or more
`SelectPages`	Select some pages from a PDF into a new document
`SplitOnBookmarks`	Split one document into several, based upon bookmarks

3.1 Simple Merging

The function `MergeSimple` takes an array of PDF objects and returns a PDF object containg the result of merging them. Page numbering is retained.

Simple Merging of Documents

C#
```
Pdf.pdfdoc result = Cpdflib.MergeSimple(pdfs)
```

VB
```
Dim result As Pdf.pdfdoc = Cpdflib.MergeSimple(pdfs)
```

3.2 Merging with Options

There are two boolean options can be set to affect merging. The option `retain_numbering`, if set, keeps the page numbering of the original documents in the merged document. If clear, the pages are renumbered 1, 2, 3 etc. The option `remove_duplicate_fonts`, if set, attempts to remove any duplicate fonts in the result - this works best when merging documents which all came from the same producing program.

Merging of Documents, with Options

C#
```
Pdf.pdfdoc result = Cpdflib.Merge(pdfs, false, true)
```

VB
```
Dim result As Pdf.pdfdoc = Cpdflib.Merge(pdf, false, true)
```

(The first boolean argument is `retain_numbering`*, the second is* `remove_duplicate_fonts`*.)*

3.3 When a Document Appears Twice

When using a document twice or more in a merge (for instance, merging `A.pdf` pages 1–3 followed by `B.pdf` followed by `A.pdf` pages 4–6) `Cpdflib` needs to know that both instances of `A.pdf` refer to the same file, so it can share common objects in the output, avoiding content shared between pages (e.g fonts) being included twice.

The function `MergeSame` is equivalent to `Merge`, but an array of strings is also passed: these would typically be the filenames of the documents - but can be any strings which allow `Cpdflib` to distinguish multiple instances of the same file.

Merging When a Document Appears Twice

C#
```
Pdf.pdfdoc result = Cpdflib.MergeSame(pdfs, false, true, names)
```

VB
```
Dim result As Pdf.pdfdoc =
  Cpdflib.MergeSame(pdfs, false, true, names)
```

3.4 Selecting Pages

Any pages can be selected from a PDF with `SelectPages`, giving a range. The pages specified in the range are present in the order in which they are given, including any duplicates (so, for instance, to extract the first three pages of a document, the range would be `1 2 3`, and to duplicate the first page of a four page document, the range would be `1 1 2 3`).

Selecing Pages from a Document

C#
```
Pdf.pdfdoc result = Cpdflib.SelectPages(pdf, pages)
```

VB
```
Dim result As Pdf.pdfdoc = Cpdflib.SelectPages(pdf, pages)
```

3.5 Splitting on Bookmarks

The `SplitOnBookmarks` function can be used to split a document into chapters, sections etc according to the tree structure of its bookmarks. The `level` argument gives the level of the tree upon which to split. Level 0 indictates just top-level bookmarks, level 1 top-level and next-level-down etc.

C#
```
Pdf.pdfdoc [] results = Cpdflib.SplitOnBookmarks(pdf, 0)
```

VB
```
Dim results As Pdf.pdfdoc () = Cpdflib.SplitOnBookmarks(pdf, 0)
```

4 Pages

Function Summary

`ScalePages`	Scale pages and their contents
`ScaleToFit`	Scale pages to fit the given dimensions
`ScaleToFitPaper`	Scale pages to fit a given paper size
`ScaleContents`	Scale the contents of a page without scaling the page
`ShiftContents`	Shift the contents of a page in x and/or y
`Rotate`	Change the viewing rotation, in absolute terms
`RotateBy`	Change the viewing rotation, in relative terms
`RotateContents`	Rotate the page contents by a free angle
`Upright`	Reconcile real and viewing rotations
`HFlip`	Flip pages horizontally
`VFlip`	Flip pages vertically
`Crop`	Add or change a cropping rectangle
`RemoveCrop`	Remove any cropping rectangle
`SetMediabox`	Set the media box (page size)

4.1 Page Sizes

In some of the following functions, standard page sizes can be given. For convenience, here is a list of supported standard page sizes:

```
a0portrait        a1portrait         a2portrait
a3portrait        a4portrait         a5portrait
a0landscape       a1landscape        a2landscape
a3landscape       a4landscape        a5landscape
usletterportrait  usletterlandscape
uslegalportrait   uslegallandscape
```

4.2 Scale Pages

The `ScalePages` function scales the page dimensions and its content by the x and y factors given.

Double the width and height of all the pages in a PDF

C#
```
Cpdflib.ScalePages(pdf, Cpdflib.All(pdf), 2.0, 2.0)
```

VB
```
Cpdflib.ScalePages(pdf, Cpdflib.All(pdf), 2.0, 2.0)
```

The `ScaleToFitPaper` function scales the page dimensions and its content to fit the given page dimensions or paper size. If the aspect ratios of the two page sizes are unequal, the page is scaled to the largest size which will fit within the new page dimensions, and centered therein.

Scale pages to fit US Letter paper

C#
```
Cpdflib.ScaleToFit(pdf, Cpdflib.All(pdf), 500.0, 400.0)
Cpdflib.ScaleToFitPaper(pdf, Cpdflib.All(pdf), Cpdflib.usletter)
```

VB
```
Cpdflib.ScaleToFit(pdf, Cpdflib.All(pdf), 500.0, 400.0)
Cpdflib.ScaleToFitPaper(pdf, Cpdflib.All(pdf), Cpdflib.usletter)
```

4.3 Shift Page Contents

The `ShiftPages` function moves the content of a given page in x and/or y directions the given number of points. The origin is at the bottom left of the page. Any page rotation is not taken into account.

Move all pages in a PDF 5pts to the right

C#
```
Cpdflib.ShiftPageContents(pdf, Cpdflib.All(pdf), 5.0, 0.0)
```

VB
```
Cpdflib.ShiftPageContents(pdf, Cpdflib.All(pdf), 5.0, 0.0)
```

4.4 Rotate Pages

There are two kinds of rotation in PDF documents - a *viewing rotation* which can be set, making a PDF viewer such as Acrobat Reader open the docment with its pages appearing to be rotated by 90, 180 or 270 degrees, and the *actual rotation* which is how the text and graphics on a page are layed out (portrait or landscape).

The `Cpdflib` functions `Rotate` and `RotateBy` set or alter the viewing rotation, and the function `RotateContents` alters the actual rotation. The function `Upright` changes the viewing rotation to 0 and, if necessary counterrotates the actual rotation to compensate. This is useful because many `Cpdflib` functions require upright pages to work as expected.

Rotate all pages to 90 degrees

C#
```
Cpdflib.Rotate(pdf, Cpdflib.All(pdf), 90)
```

VB
```
Cpdflib.Rotate(pdf, Cpdflib.All(pdf), 90)
```

Rotate all pages by 90 degrees

C#
```
Cpdflib.RotateBy(pdf, Cpdflib.All(pdf), 90)
```

VB
```
Cpdflib.RotateBy(pdf, Cpdflib.All(pdf), 90)
```

Rotate the contents of all pages by 30 degrees

C#
```
Cpdflib.RotateContents(pdf, Cpdflib.All(pdf), 30.0)
```

VB
```
Cpdflib.RotateContents(pdf, Cpdflib.All(pdf), 30.0)
```

4.5 Flip Pages

The functions `HFlip` and `VFlip` flip content horizontally and vertically respectively.

Flip pages horizontally

C#
```
Cpdflib.HFlip(pdf, Cpdflib.All(pdf))
```

VB
```
Cpdflib.HFlip(pdf, Cpdflib.All(pdf))
```

Flip pages vertically

C#
```
Cpdflib.VFlip(pdf, Cpdflib.All(pdf))
```

VB
```
Cpdflib.VFlip(pdf, Cpdflib.All(pdf))
```

4.6 Page Size and Page Cropping

All PDF files contain a *media box* for each page, giving the dimensions of the paper. To change these dimensions (without altering the page contents in any way), use the `SetMediabox` function.

Set media box (page size)

C#
```
Cpdflib.SetMediabox
  (pdf, Cpdflib.All(pdf), 0.0, 0.0, 100.0, 200.0)
```

VB
```
Cpdflib.SetMediabox
  (pdf, Cpdflib.All(pdf), 0.0, 0.0, 100.0, 200.0)
```

The four floating point arguments are minimum x, minimum y, maximum x, maximum y. The origin is at (0, 0) with x coordinates increasing to the right, y coordinates increasing upwards.

PDF file can also optionally contain a *crop box* for each page, defining to what extent the page is cropped before being displayed or printed. A crop box can be set, changed and removed, without affecting the underlying media box. To set or change the crop box use `Crop`. To remove any existing crop box, use `RemoveCrop`.

Set crop box

C#
```
Cpdflib.Crop
  (pdf, Cpdflib.All(pdf), 0.0, 0.0, 100.0, 200.0)
```

VB
```
Cpdflib.Crop
  (pdf, Cpdflib.All(pdf), 0.0, 0.0, 100.0, 200.0)
```

The four floating point arguments are minimum x, minimum y, maximum x, maximum y. The origin is at (0, 0) with x coordinates increasing to the right, y coordinates increasing upwards.

Remove any crop box in place

C#
```
Cpdflib.RemoveCrop(pdf, Cpdflib.All(pdf))
```

VB
```
Cpdflib.RemoveCrop(pdf, Cpdflib.All(pdf))
```

5 Compression

Function Summary

`Compress`	Compress the data streams within a document
`Decompress`	Decompress the data streams within a document

`Cpdflib` provides basic facilities for compressing and decompressing PDF streams.

5.1 Compressing a Document

To compress the streams in a file, use the `Compress` function. It uses the `/FlateDecode` compression method.

C#
```
Cpdflib.Compress(pdf)
```

VB
```
Cpdflib.Compress(pdf)
```

5.2 Decompressing a Document

The `Decompress` function decompresses the streams in a document. If `Cpdflib` finds a compression method it can't deal with, the stream is left compressed.

C#
```
Cpdflib.Decompress(pdf)
```

VB
```
Cpdflib.Decompress(pdf)
```

6 Bookmarks

Function Summary

`AddBookmarks`	Add or replace bookmarks
`MakeBookmark`	Make a bookmark
`GetBookmarks`	Get a list of the bookmarks in a document
`RemoveBookmarks`	Remove bookmarks from a document

PDF Bookmarks (properly called the *document outline*) represent a tree of references to parts of the file, typically displayed at the side of the screen. The user can click on one to move to the specified place. `Cpdflib` provides facilities to list, add, and remove bookmarks. The format used by the list and add operations is the same, so you can feed the output of one into the other, for instance to copy bookmarks.

Each bookmark has four elements:

`level`	Integer	Level of bookmark (0 = top level, 1 = next level etc.)
`text`	String	Bookmark text e.g ”Section 1B”
`target`	Integer	The destination page number
`isopen`	Boolean	If true this bookmark’s children are visible

For example:

`level`	`string`	`pagenumber`	`isopen`
0	”Part 1”	1	true
1	”Part 1A”	2	false
1	”Part 1B”	3	false
0	”Part 2”	8	false
1	”Part 2a”	9	false

If the page number is 0, it indicates that clicking on that entry doesn’t move to a page.

To add bookmarks, build a suitable bookmark array using `MakeBookmarks` and call `AddBookmarks`.

Make and add bookmarks to a document

C#
```
Pdfmarks.bookmark [] marks =
  {Cpdflib.MakeBookmark(0, "Part 1", 1, true),
   Cpdflib.MakeBookmark(1, "Part 1A", 2, false),
   Cpdflib.MakeBookmark(1, "Part 1B", 3, false),
   Cpdflib.MakeBookmark(0, "Part 2", 8, false),
   Cpdflib.MakeBookmark(1, "Part 2a", 9, false)};
Cpdflib.AddBookmarks(pdf, marks);
```

VB
```
Dim marks As Pdfmarks.bookmark () =
  {Cpdflib.MakeBookmark(0, "Part 1", 1, true),
   Cpdflib.MakeBookmark(1, "Part 1A", 2, false),
   Cpdflib.MakeBookmark(1, "Part 1B", 3, false),
   Cpdflib.MakeBookmark(0, "Part 2", 8, false),
   Cpdflib.MakeBookmark(1, "Part 2a", 9, false)}
Cpdflib.AddBookmarks(pdf, marks)
```

It's important that the bookmarks passed to `AddBookmarks` are consistent - that is, that the levels represent a proper tree form.

The function `GetBookmarks` returns an array of all the bookmarks (if any) in a document.

Get the bookmarks from a PDF

C#
```
Pdfmarks.bookmark [] marks = Cpdflib.GetBookmarks(pdf)
```

VB
```
Dim marks As Pdfmarks.bookmark () = Cpdflib.GetBookmarks(pdf)
```

The bookmark data structure can then be inspected:

Write the title of a bookmark 'mark' to screen

C#
```
Console.WriteLine(mark.text)
```

VB
```
Console.WriteLine(mark.text)
```

The bookmarks in a document can be removed with `RemoveBookmarks`.

Remove bookmarks

C#
```
Cpdflib.RemoveBookmarks(pdf)
```

VB
```
Cpdflib.RemoveBookmarks(pdf)
```

7 Presentations

Function Summary

`AddPresentation`	Make a PDF into a Powerpoint-style presentation
`RemovePresentation`	Remove such a presentation

The PDF file format, starting at Version 1.1, provides for simple slide-show presentations in the manner of Microsoft Powerpoint. These can be played in Acrobat and possibly other PDF viewers, typically started by entering full-screen mode. The `AddPresentation` operation allows such a presentation to be built from any PDF file.

There are several transition styles:

Split Two lines sweep across the screen, revealing the new page. By default the lines are horizontal. If the `vertical` flag is set, vertical lines are used instead.

Blinds Multiple lines sweep across the screen, revealing the new page. By default the lines are horizontal. If the `vertical` flag is set, vertical lines are used instead.

Box A rectangular box sweeps inward from the edges of the page. Set the `outward` argument to make it sweep from the center to the edges.

Wipe A single line sweeps across the screen from one edge to the other in a direction specified by the `direction` argument (see below).

Dissolve The old page dissolves gradually to reveal the new one.

Glitter The same as **Dissolve** but the effect sweeps across the page in the direction specified by the `direction` argument.

It is the transition *from* each page named which is altered. The `effect_duration` option specifies the length of time in seconds for the transition itself. The `duration` option specifies the maximum time in seconds that the page is displayed before the presentation automatically advances.

For no automatic advancement, specify a negative number.

The `direction` argument (for **Wipe** and **Glitter** styles only) specifies the direction of the effect. The following values are valid:

0 Left to right

90 Bottom to top (**Wipe** only)

180 Right to left (**Wipe** only)

270 Top to bottom

315 Top-left to bottom-right (**Glitter** only)

Make a PDF presentation

C# / VB

```
Cpdflib.AddPresentation
  (pdf,                              Document
   Cpdflib.All(pdf),                 Range
   Cpdflib.transition.Blinds,        Transition
   1.0,                              Duration
   False,                            Vertical Flag
   False,                            Outward Flag
   0,                                Direction
   0.5)                              Effect Duration
```

To remove a transition style currently applied to the selected pages, use `RemovePresentation`.

Remove any presentation

C#
```
Cpdflib.RemovePresentation(pdf)
```

VB
```
Cpdflib.RemovePresentation(pdf)
```

8 Logos, Watermarks and Stamps

Function Summary

`StampOn`	Stamp a page over another one
`StampUnder`	Stamp a page under another one
`CombinePages`	Combine the contents of pages together
`AddText`	Add text, dates, stamps, times and bates numbers
`RemoveText`	Remove text added with `AddText`

8.1 Adding a Logo or Watermark

The `StampOn` and `StampUnder` functions stamp the first page of a source PDF onto or under each page in the given range of a given document.

Stamp a source PDF on or under some pages of a document

C#
```
Cpdflib.StampOn(sourcepdf, pdf, range)
Cpdflib.StampUnder(sourcepdf, pdf, range)
```

VB
```
Cpdflib.StampOn(sourcepdf, pdf, range)
Cpdflib.StampUnder(sourcepdf, pdf, range)
```

The `CombinePages` function takes two PDF documents and stamps each page of one over the corresponding page in the other. Page attributes (such as the display rotation) are taken from the "under" file. For best results, remove any rotation differences in the two files using `Upright` and `SetMediabox` first.

Combine pages

C#
```
Pdf.pdfdoc result =
  Cpdflib.CombinePages(pdf_over, pdf_under)
```

VB
```
Dim result As Pdf.pdfdoc =
  Cpdflib.CombinePages(pdf_over, pdf_under)
```

8.2 Stamp Text, Dates and Times

The `AddText` function stamps text, dates, or times over one or more pages of a document. The position, color, font and size may be customised.

Here is the basic usage. We describe each of the options below.

Stamp Text

C# / VB

```
Cpdflib.AddText
  (pdf,                                  Document
   Cpdflib.All(pdf),                     Range
   "This is Page %Page",                 Text
   Cpdf.position.TopLeft(10.0),          Position
   1.0,                                  Line spacing
   0,                                    Starting Bates Number
   Pdftext.standard_font.Courier,        Font
   24.0,                                 Font Size
   Cpdflib.Red,                          Text Color
   false,                                If set, stamp on shorter side
   false,                                If set, stamp underneath
   true)                                 If set, stamp relative to cropbox
```

Several special formatting codes (similar to those found in the `printf` function in most programming languages) are available.

Page Numbers

`%Page`	Page number in arabic notation (1, 2, 3...)
`%roman`	Page number in lower-case roman notation (i, ii, iii...)
`%Roman`	Page number in upper-case roman notation (I, II, III...)
`%EndPage`	Last page of document in arabic notation

For example, the format `"Page %Page of %EndPage"` might become ”Page 5 of 17”.

Date and Time Formats

`%a`	Abbreviated weekday name (Sun, Mon etc.)
`%A`	Full weekday name (Sunday, Monday etc.)
`%b`	Abbreviated month name (Jan, Feb etc.)
`%B`	Full month name (January, February etc.)
`%d`	Day of the month (01–31)
`%e`	Day of the month (1–31)
`%H`	Hour in 24-hour clock (00–23)
`%I`	Hour in 12-hour clock (01–12)
`%j`	Day of the year (001–366)
`%m`	Month of the year (01–12)
`%M`	Minute of the hour (00–59)
`%p`	”a.m” or ”p.m”
`%S`	Second of the minute (00–61)

`%T`	Same as %H:%M:%S
`%u`	Weekday (1–7, 1 = Monday)
`%w`	Weekday (0–6, 0 = Monday)
`%Y`	Year (0000–9999)
`%%`	The % character.

Bates Numbers

Unique page identifiers can be specified by putting `%Bates` in the string. The starting point can be set with the `bates` argument. For example:

```
"Page ID: %Bates"
```

when `bates` is 45 would number each page 45, 46, 47…

Position

The position of the text may be specified in absolute terms:

```
PosCentre(200.0, 200.0)
```

Position the center of the baseline text at (200pt, 200pt)

```
PosLeft(200.0, 200.0)
```

Position the left of the baseline of the text at (200pt, 200pt)

```
PosRight(200.0, 200.0)
```

Position the right of the baseline of the text at (200pt, 200pt)

Position relative to certain common points can also be set:

`Top(10)`	Center of baseline 10 pts down from the top center
`TopLeft(10)`	Left of baseline 10 pts down and in from top left
`TopRight(10)`	Right of baseline 10 pts down and left from top right
`Left(10)`	Left of baseline 10 pts in from center left
`BottomLeft(10)`	Left of baseline 10 pts in and up from bottom left
`Bottom(10)`	Center of baseline 10 pts up from bottom center
`BottomRight(10)`	Right of baseline 10 pts up and in from bottom right
`Right(10)`	Right of baseline 10 pts in from the center right

No attempt is made to take account of the page rotation, so you might like to use `-upright` (see §4.4) first.

Font and Size

The font may be set with the `font` argument. The 14 Standard PDF fonts are available:

TimesRoman
TimesBold
TimesItalic
TimesBoldItalic
Helvetica
HelveticaBold
HelveticaOblique
HelveticaBoldOblique
Courier
CourierBold
CourierOblique
CourierBoldOblique
Symbol
ZapfDingbats

The font size can be set with the `fontsize` option, which specifies the size in points.

Colors

The `color` argument allows the color of the text to be set. The following values are predefined (components range between 0 and 1):

Color	R, G, B
white	1, 1, 1
black	0, 0, 0
red	1, 0, 0
green	0, 1, 0
blue	0, 0, 1

The function `Rgb(r, g, b)` can be used to build a color from red, green and blue components.

Multi-line Text

The code `\n` can be included in the text string to move to the next line. In this case, the vertical position refers to the baseline of the first line of text (if the position is at the top, top left or top right of the page) or the baseline of the last line of text (if the position is at the bottom, bottom left or bottom right).

The `linespacing` option can be used to increase or decrease the line spacing, where a spacing of 1 is the standard.

9 Multipage Facilities

Function Summary

`TwoUp`	Impose a document two-up
`TwoUpStack`	Impose a document two-up, with no scaling
`PadBefore`	Add blank pages before some pages
`PadAfter`	Add blank pages after some pages

9.1 Two-up

The `TwoUp` function puts two logical pages on each physical page, rotating them 90 degrees and scaling them down to do so. The `TwoUpStack` variant does the same, but with no scaling, the output page size being twice the input page size.

Impose pages two-up

C#
```
Cpdflib.TwoUp(pdf)
```

VB
```
Cpdflib.TwoUp(pdf)
```

9.2 Inserting Blank Pages

Sometimes, for instance to get a printing arrangement right, it's useful to be able to insert blank pages into a PDF file. `Cpdflib` can add blank pages before (`PadBefore`) a given page or pages, or after (`PadAfter`). The pages in question are specified by a range.

Here's an example with `PadAfter`:

Insert blank pages after existing pages one, two and three

C#
```
int[] r = {1, 2, 3}
Cpdflib.PadAfter(pdf, r)
```

VB
```
Dim r As Integer() r = {1, 2, 3}
Cpdflib.PadAfter(pdf, r)
```

10 Annotations

Function Summary

`ListAnnotations`	Get the textual content of existing annotations
`CopyAnnotations`	Copy annotations from one document to another
`RemoveAnnotations`	Remove the annotations from a document

An annotation consists of a page number and a string. The two components can be accessed using the dot notation. For instance, if the annotation variable is `annot`:

To access page number	`annot.annotation_page`
To access annotation contents	`annot.annotation_content`

To list the annotations in a document, use `ListAnnotations`:

List annotations in a document

C#
```
Cpdflib.annotation [] annots = Cpdflib.ListAnnotations(pdf)
```

VB
```
Dim annots As Cpdflib.annotation () =
  Cpdflib.ListAnnotations(pdf)
```

To copy annotations from one document to another, use `CopyAnnotations`:

Copy annotations from one document to another

C#
```
Cpdflib.CopyAnnotations(source_pdf, target_pdf)
```

VB
```
Cpdflib.CopyAnnotations(source_pdf, target_pdf)
```

To remove annotations from a documents, use `RemoveAnnotations`:

Remove annotations from a document

C#
```
Cpdflib.RemoveAnnotations(pdf)
```

VB
```
Cpdflib.RemoveAnnotations(pdf)
```

11 Document Information

Function Summary

`ListFonts`	List the fonts in a document
`GetVersion`	Get the PDF version number
`GetTitle`	Get the title of a document
`GetAuthor`	Get the author of a document
`GetSubject`	Get the subject of a document
`GetKeywords`	Get the keyword list from a document
`GetCreator`	Get the creator name from a document
`GetProducer`	Get the producer name from a document
`GetCreationDate`	Get the creation date of a document
`GetModificationDate`	Get the modification date of a document
`SetVersion`	Set the PDF version number
`SetTitle`	Set the title of a document
`SetAuthor`	Set the author of a document
`SetSubject`	Set the subject of a document
`SetKeywords`	Set the keyword list in a document
`SetCreator`	Set the creator name in a document
`SetProducer`	Set the producer name in a document
`SetCreationDate`	Set the creation date in a document
`SetModifcationDate`	Set the modification date in a document
`MarkTrapped`	Mark a PDF as trapped
`MarkUntrapped`	Mark a PDF as untrapped
`GetPageInfo`	Get information for each page
`SetPageLayout`	Set the page layout
`SetPageMode`	Set the page mode
`HideToolbar`	Hide or reveal the viewer's toolbar
`HideMenubar`	Hide or reveal the viewer's menubar
`FitWindow`	Set or unset Fit-to-Window
`CenterWindow`	Set or unset Center-Window
`DisplayDocTitle`	Set or unset display of document title

11.1 Listing Fonts

The `ListFonts` returns an array of `font_info` structures, each member of which can be accessed with the dot notation:

```
font_pagenumber  integer
font_name        string
font_subtype     string
font_basename    string
font_encoding    string
```

For example:

List fonts

C#
```
Cpdflib.font_info [] fonts = Cpdflib.ListFonts(pdf)
```

VB
```
Dim fonts As Cpdflib.font_info () = Cpdflib.ListFonts(pdf)
```

might return the array:

Page	Name	Type	Base Name	Encoding
1	/F245	/Type0	/Cleargothic-Bold	/Identity-H
1	/F247	/Type0	/ClearGothicSerialLight	/Identity-H
1	/F248	/Type1	/Times-Roman	/WinAnsiEncoding
1	/F250	/Type0	/Cleargothic-RegularItalic	/Identity-H
2	/F13	/Type0	/Cleargothic-Bold	/Identity-H
2	/F16	/Type0	/Arial-ItalicMT	/Identity-H
2	/F21	/Type0	/ArialMT	/Identity-H
2	/F58	/Type1	/Times-Roman	/WinAnsiEncoding
2	/F59	/Type0	/ClearGothicSerialLight	/Identity-H
2	/F61	/Type0	/Cleargothic-BoldItalic	/Identity-H
2	/F68	/Type0	/Cleargothic-RegularItalic	/Identity-H
3	/F47	/Type0	/Cleargothic-Bold	/Identity-H
3	/F49	/Type0	/ClearGothicSerialLight	/Identity-H
3	/F50	/Type1	/Times-Roman	/WinAnsiEncoding
3	/F52	/Type0	/Cleargothic-BoldItalic	/Identity-H
3	/F54	/Type0	/TimesNewRomanPS-BoldItalicMT	/Identity-H
3	/F57	/Type0	/Cleargothic-RegularItalic	/Identity-H
4	/F449	/Type0	/Cleargothic-Bold	/Identity-H
4	/F451	/Type0	/ClearGothicSerialLight	/Identity-H
4	/F452	/Type1	/Times-Roman	/WinAnsiEncoding

The first column gives the page number, the second the internal unique font name, the third the type of font (Type1, TrueType etc), the fourth the PDF font name, the fifth the PDF font encoding.

11.2 Getting Document Information

There are a number of functions for getting document information. Each function takes the document as an argument and returns an `integer` or `string`.

Function Summary

Function	Returns	Description
`GetVersion`	`integer`	Get the PDF version number
`GetTitle`	`string`	Get the title of a document
`GetAuthor`	`string`	Get the author of a document
`GetSubject`	`string`	Get the subject of a document
`GetKeywords`	`string`	Get the keyword list from a document
`GetCreator`	`string`	Get the creator name from a document
`GetProducer`	`string`	Get the producer name from a document
`GetCreationDate`	`date`	Get the creation date of a document
`GetModificationDate`	`date`	Get the modification date of a document

(Dates are a special kind of string - see Appendix A). For example, to return the author of a document:

C#
```
string author = Cpdflib.GetAuthor(pdf)
```

VB
```
Dim author As String = Cpdflib.GetAuthor(pdf)
```

11.3 Setting Document Information

There are a number of functions for setting document information. Each function takes the document as an argument, followed by a single other argument.

Function	Argument	Description
`SetVersion`	`integer`	Set the PDF version number
`SetTitle`	`string`	Set the title of a document
`SetAuthor`	`string`	Set the author of a document
`SetSubject`	`string`	Set the subject of a document
`SetKeywords`	`string`	Set the keyword list in a document
`SetCreator`	`string`	Set the creator name in a document
`SetProducer`	`string`	Set the producer name in a document
`SetCreationDate`	`date`	Set the creation date in a document
`SetModifcationDate`	`date`	Set the modification date in a document

(Dates are a special kind of string - see Appendix A). For example, to set the title of a document, and set its version number to PDF 1.2:

C#
```
Cpdflib.SetVersion(pdf, 2)
Cpdflib.SetTitle(pdf, "Forces of Nature")
```

VB
```
Cpdflib.SetVersion(pdf, 2)
Cpdflib.SetTitle(pdf, "Forces of Nature")
```

11.4 Mark Trapped / Untrapped

These two functions can be used to mark a PDF trapped or untrapped. They take one argument—the document.

11.5 Get and Set Page Layout and Mode

The `SetPageLayout` function specifies the page layout to be used when a document is opened in, for instance, Acrobat. The possible values are provided in the `layout` object in the `Cpdflib` module:

`SinglePage`	Display one page at a time
`OneColumn`	Display the pages in one column
`TwoColumnLeft`	Display the pages in two columns, odd numbered pages on the left
`TwoColumnRight`	Display the pages in two columns, even numbered pages on the left
`TwoPageLeft`	(PDF 1.5 and above) Display the pages two at a time, odd numbered pages on the left
`TwoPageRight`	(PDF 1.5 and above) Display the pages two at a time, even numbered pages on the left

For instance:

Set document to open with pages in Two Columns, even numbered pages on the left.

C#
```
Cpdflib.SetPageLayout(pdf, Cpdflib.layout.TwoColumnRight)
```

VB
```
Cpdflib.SetPageLayout(pdf, Cpdflib.layout.TwoColumnRight)
```

The *page mode* in a PDF file defines how a viewer should display the document when first opened and can be set with the `SetPageMode` function. Possible values, in the `pagemode` object in `Cpdflib` are:

`UseNone`	Neither document outline nor thumbnail images visible
`UseOutlines`	Document outline (bookmarks) visible
`UseThumbs`	Thumbnail images visible
`FullScreen`	Full-screen mode (no menu bar, window controls, or anything but the document visible)
`UseOC`	(PDF 1.5 and above) Optional content group panel visible
`UseAttachments`	(PDF 1.5 and above) Attachments panel visible

For instance:

Set document to open with bookmarks visible

C#
```
Cpdflib.SetPageMode(pdf, Cpdflib.pagemode.UseOutlines)
```

VB
```
Cpdflib.SetPageMode(pdf, Cpdflib.pagemode.UseOutlines)
```

Page information (media box, crop box) can be returned with the function `GetPageInfo`, which takes a document,and returns an array of `page_info` objects, one for each page in the document.

Each object has two members `mediabox` and `cropbox`, each of which is a `box` object. A box object contains four floating point objects (`minx`, `maxx`, `miny`, `maxy`) describing the box in question. For example:

Read and print the minimum x coordinate of the media box of the first page

C#
```
Cpdflib.page_info [] pageinfo = Cpdflib.GetPageInfo(pdf)
Console.WriteLine(pageinfo[0].mediabox.minx)
```

VB
```
Dim pageinfo As Cpdflib.page_info() = Cpdflib.GetPageInfo(pdf)
Console.WriteLine(pageinfo(0).mediabox.minx)
```

11.6 Other View Settings

The following functions take a document, and a boolean, to indicate if the option is to be set or unset.

`HideToolbar`	Hide the viewer's toolbar
`HideMenubar`	Document outline (bookmarks) visible
`HideWindowUi`	Hide the viewer's scroll bars
`FitWindow`	Resize the document's windows to fit size of first page
`CenterWindow`	Position the document window in the center of the screen
`DisplayDocTitle`	Display the document title instead of the file name in the title bar

For instance:

Set document to open resized to fit first page

C#
```
Cpdflib.FitWindow(pdf)
```

VB
```
Cpdflib.FitWindow(pdf)
```

12 Document Metadata

Function Summary

`SetMetadataFromFile`	Set the Document Metadata from a file
`SetMetadataFromByteArray`	Set the Document Metadata from a byte array
`GetMetadata`	Get the document metadata as a byte array
`RemoveMetadata`	Remove the metadata from a document

PDF files can contain a piece of arbitrary metadata, often in XML format. This is typically stored in an uncompressed stream, so that other applications can read it without having to decode the whole PDF. To set the metadata from a file:

Add metadata from a file

C#
```
Cpdflib.SetMetadataFromFile(pdf, "C:\\data.xml")
```

VB
```
Cpdflib.SetMetadataFromFile(pdf, "C:\\data.xml")
```

To set from a byte array:

Set Metadata from Byte Array

C#
```
Cpdflib.SetMetadataFromByteArray(pdf, bytes)
```

VB
```
Cpdflib.SetMetadataFromByteArray(pdf, bytes)
```

Return the metadata as a byte array:

C#
```
byte [] bytes = Cpdflib.GetMetadata(pdf)
```

VB
```
Dim bytes As Byte () = Cpdflib.GetMetadata(pdf)
```

To remove any metadata from a file:

C#
```
Cpdflib.RemoveMetadata(pdf)
```

VB
```
Cpdflib.RemoveMetadata(pdf)
```

13 File Attachments

Function Summary

`AttachFile`	Attach a file to the document
`RemoveAttachedFiles`	Remove all attached files from a document

PDF supports adding attachments (files of any kind, including other PDFs) to an existing file. The `Cpdflib` library supports adding and removing *top-level attachments* — that is, ones which are associated with the document as a whole rather than with an individual page.

To add an attachment, use `AttachFile`. Multiple files can be attached by multiple invocations of `AttachFile`, and will appear in Acrobat's list in the order added.

Attach the file `sheet.xls` *to a document*

C#
```
Cpdflib.AttachFile(pdf, "C:\\sheet.xls")
```

VB
```
Cpdflib.AttachFile(pdf, "C:\\sheet.xls")
```

To remove all attached files, use `RemoveAttachedFiles`:

Remove attached files

C#
```
Cpdflib.RemoveAttachedFiles(pdf)
```

VB
```
Cpdflib.RemoveAttachedFiles(pdf)
```

14 Miscellaneous

Function Summary

`BlackText`	Blacken text
`BlackLines`	Blacken lines
`BlackFills`	Blacken fills
`ThinLines`	Make all lines at least a certain width
`Draft`	Remove images for draft printing
`CopyId`	Copy the unique document ID from one document to another

14.1 Graphical Alterations

Sometimes PDF output from an application (for instance, a web browser) has text in colors which would not print well on a grayscale printer. The `BlackText` operation blackens all text on the given pages so it will be readable when printed. This will not work on text which has been converted to outlines, nor on text which is part of a form.

The `BlackLines` function does the same for lines and `BlackFills` for fill colors.

Blacken lines in a document

C#
```
Cpdflib.BlackLines(pdf)
```

VB
```
Cpdflib.BlackLines(pdf)
```

Quite often, applications will use very thin lines, or even the value of 0, which in PDF means "The thinnest possible line on the output device". This might be fine for on-screen work, but when printed on a high resolution device, such as by a commercial printer, they may be too faint, or disappear altogether. The `ThinLines` function prevents this by changing all lines thinner than the given minimal thickness to the given width. For example:

Thicken lines to at least 2pt

C#
```
Cpdflib.ThinLines(pdf, Cpdflib.All(pdf), 2.0)
```

VB
```
Cpdflib.ThinLines(pdf, Cpdflib.All(pdf), 2.0)
```

14.2 Draft Documents

The `Draft` function removes bitmap (photographic) images from a file, so that it can be printed with less ink. Optionally, the the boxes argument can be set, filling the spaces left blank with a crossed box denoting where the image was. This is not guaranteed to be fully visible in all cases (the bitmap may be have been partially covered by vector objects or clipped in the original). For example:

Remove images, replacing with crossed boxes

C#
```
Cpdflib.Draft(pdf, Cpdflib.All(pdf), true)
```

VB
```
Cpdflib.Draft(pdf, Cpdflib.All(pdf), true
```

14.3 Document Identification

The `CopyId` option copies the ID from the given file to the input, writing to the output. If there is no ID in the source file, the operation fails.

Remove attached files

C#
```
Cpdflib.CopyId(from_pdf, to_pdf)
```

VB
```
Cpdflib.CopyId(from_pdf, to_pdf)
```

A Dates

Dates in PDF are specified according to the following format:

D:YYYYMMDDHHmmSSOHH'mm'

where:

- YYYY is the year;
- MM is the month;
- DD is the day (01-31);
- HH is the hour (00-23);
- mm is the minute (00-59);
- SS is the second (00-59);
- O is the relationship of local time to Universal Time (UT), denoted by '+', '-' or 'Z';
- HH is the absolute value of the offset from UT in hours (00-23);
- mm is the absolute value of the offset from UT in minutes (00-59).

A contiguous prefix of the parts above can be used instead, for lower accuracy dates. For example:

D:2010 (2010)
D:20100103 (3rd March 2010)
D:201001031854-08'00' (3rd March 2010, 6:54PM, US Pacific Standard Time)

B Example Program in C#

This program loads a file from disk, adds page numbers, and then writes the file encrypted to disk.

```
namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
          //Load the (unencrypted) input file
          Pdf.pdfdoc pdf = Cpdflib.FromFile("C:\\input.pdf");

          //Resolve any rotation of pages
          Cpdflib.Upright(pdf, Cpdflib.All(pdf));

          //Add page numbers
          Cpdflib.AddText
            (pdf,                                  //Document
             Cpdflib.All(pdf),                     //Range
             "Page %Page of %EndPage",             //Text
             Cpdf.position.Top(20.0),              //Position
             1.0,                                  //Line spacing
             0,                                    //Bates number
             Pdftext.standard_font.TimesBold,      //Font
             12.0,                                 //Font Size
             Cpdflib.Black,                        //Text Color
             false,                                //If set, stamp on shorter side
             false,                                //If set, stamp underneath
             true);                                //If set, stamp relative to cropbox

          //Write to file, encrypted with AES encryption
          Pdfcrypt.permission[] permissions =
            {Pdfcrypt.permission.NoEdit,
             Pdfcrypt.permission.NoAssemble};
          Cpdflib.ToFileEncrypted
             (pdf,                                          //Document
              Pdfwrite.encryption_method.AES128bit(false),  //Encryption method
              permissions,                                  //Permissions
              "fred",                                       //Owner Password
              "charles",                                    //User Password
              false,                                        //Linearize
              "C:\\output.pdf");                            //Output filename
        }
```

```
        }
}
```

C Example Program in VB.NET

This program loads a file from disk, adds page numbers, and then writes the file encrypted to disk.

```
Module Module1

    Sub Main()
        'Load the (unencrypted) input file
        Dim pdf As Pdf.pdfdoc = Cpdflib.FromFile("C:\\input.pdf")

        'Resolve any rotation of pages
        Cpdflib.Upright(pdf, Cpdflib.All(pdf))

        'Add page numbers
        Cpdflib.AddText _
          (pdf, Cpdflib.All(pdf), "Page %Page of %EndPage", _
          Cpdf.position.Top(20.0), 1.0, 0, _
          Pdftext.standard_font.TimesBold, _
          12.0, Cpdflib.Black, False, False, True)

        'Write to file, encrypted with AES encryption
        Dim permissions As Pdfcrypt.permission() = _
          {Pdfcrypt.permission.NoEdit, Pdfcrypt.permission.NoAssemble}
        Cpdflib.ToFileEncrypted _
          (pdf, _
           Pdfwrite.encryption_method.AES128bit(False), _
           permissions, "fred", "charles", _
           False, "C:\\output.pdf")
    End Sub

End Module
```