GETTING STARTED GUIDE

PDF Technologies, Inc.

PDFTECHNOLOGIES
Tools & Components for Serious Developers

Overview

PDFTechLib is a .NET class library used to create, edit and otherwise manipulate PDF documents. The library does not require the use of Adobe products and is NOT a print driver.

You can create stylish PDF documents with just a few lines of code. The intuitive object model allows developers to quickly adopt and understand the wealth of code samples provided with our standard package.

You will be able to fill PDF e-forms, merge multiple documents, split existing PDF documents into multiple documents and finally create your own PDF documents using existing PDF documents as templates or from scratch.

The library is written in 100% C# managed code. This versatile library can be used in all kinds of ASP.NET applications. It can even run in restricted hosting environments.

PDFTechLib is licensed per developer and can be distributed ROYALTY FREE. Unlike other complicated and restrictive licensing plans, our simple licensing structure will allow you to quickly get started with development.

Creating a PDF document is as easy as the sample code below:

```
// If you purchased a license key, set the license like this:
// PDFDocument.License = "JRO1F2MS-2021-188-J004A";
PDFDocument doc = new PDFDocument("sample.pdf");
doc.CurrentPage.Body.SetActiveFont("Arial", PDFFontStyles.Regular,12,
Charset.ANSI_CHARSET, Color.Red);
doc.CurrentPage.Body.AddText("Wow! What a good looking PDF
Document!");
doc.Save();
```

SYSTEM REQUIREMENTS & DISTRIBUTION

PDFTechLib requires the following

- .Net Framework 4.0 or higher
- Visual Studio 2010 or higher
- Windows XP (x86) with Service Pack 3 or higher OS
- 32-Bit (x86) or 64-Bit (x64)
- 2 GB RAM or more

HOW TO INSTALL

After downloading the archive, run the installation package and follow the installation instructions.

HOW TO DISTRIBUTE

Programmers can distribute "PDFTechLib.dll" in their solutions.

The license key must be embedded in the solution code.

.Net Framework 4.0 must be present in target machines. Generally, ASP.NET Shared Hosting environments are restricted and most components cannot be run in these environments.

PDFTechLib.dll library is fully functional with minimum trust level in tightly restricted ASP.net environments. The exceptions to this are listed below:

- Signing PDF documents is not possible in restricted environments.
- EMF, HTML, XML to PDF feature cannot be used.
- Using installed system fonts is not possible because the Library requires access to the Widows/Font directory.

If you wish to use these features, the library must be given Full trust Level. If you find that your code works well on your development machine and not on the deployed target machine, more than likely the problem has to do with permissions.

ASP.NET projects can be given full trust by inserting the following section into the project's web.config file:

```
<system.web>
    <securityPolicy>
        <trustLevel name="Full" policyFile="internal"/>
    </securityPolicy>
</system.web>
```

More information on trust levels can be found here:

http://msdn.microsoft.com/en-us/library/wyts434y.aspx

For executable code or console applications, the "Run as Administrator" option must be used in restricted environments.

To automate the "UAC administrator mode" you must add the following in the "App.manifest" file.

```
<?xml version="1.0" encoding="utf-8"?>
<asmv1:assembly manifestVersion="1.0" xmlns="urn:schemas-microsoft-com:asm.v1" xmlns:asmv1="urn:schemas-microsoft-com:asm.v1" xmlns:asmv
  <trustInfo xmlns="urn:schemas-microsoft-com:asm.v2">
    <security>
      <applicationRequestMinimum>
        <defaultAssemblyRequest permissionSetReference="Custom" />
        <PermissionSet class="System.Security.PermissionSet" version="1" ID="Custom" SameSite="site" Unrestricted="true" />
      </applicationRequestMinimum>
      <requestedPrivileges xmlns="urn:schemas-microsoft-com:asm.v3">
        <!-- UAC Manifest Options
            If you want to change the Windows User Account Control level replace the
            requestedExecutionLevel node with one of the following.

        <requestedExecutionLevel  level="asInvoker" uiAccess="false" />
        <requestedExecutionLevel  level="requireAdministrator" uiAccess="false" />
        <requestedExecutionLevel  level="highestAvailable" uiAccess="false" />

            If you want to utilize File and Registry Virtualization for backward
            compatibility then delete the requestedExecutionLevel node.
        -->
        <requestedExecutionLevel level="requireAdministrator" uiAccess="false" />
      </requestedPrivileges>
    </security>
  </trustInfo>
</asmv1:assembly>
```

FEATURES

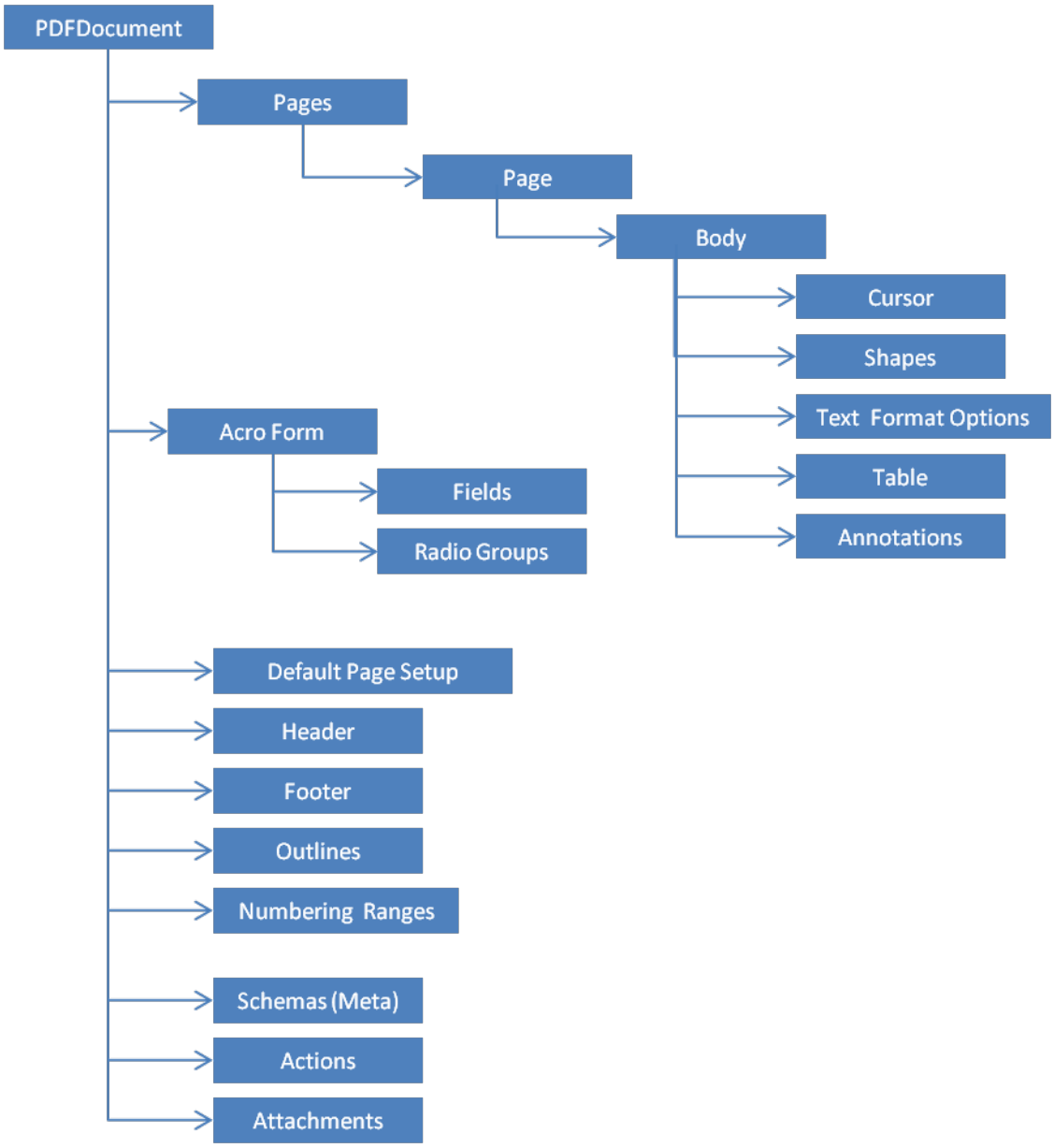
The main features of PDFTechLib are outlined below:

FONTS	Standard Fonts
	True Type Fonts
	Open Type Fonts
	CJK Fonts
	Unicode
	Various char set
	Fonts embedding
TEXT GENERATION	Text Format Options
	Multiple Columns
	Auto Text
	Justified Texts
	Text Boxes
	Tagged texts
DRAWING	Various Shapes
	Hatch Brush
	Gradient Brush
	Layers
	Barcodes
IMAGES	Vector images(EMF ,WMF support)
	Scalar Images
	Mask
	Transparency
	Stamps
	Watermarks
INTERACTIVITY	Acroforms
	Form filling
	Flatten Acroform Objects
	Form filling with XML, FDF or XFDF
	Attachments
	Annotations
	Bookmarks
	Actions

PAGE ELEMENTS	Margins
	Header and footer
	Unit of Measure
	Page Transitions
	Auto Page Numbering
	Thumbnails
	Tables
SETTINGS	Document Properties
	Custom Properties
	Page Orientation
	Page Sizes
	Viewer Preferences
SECURITY	Encryption and Decryption
	Digital Signatures
	Recover owner password
CONVERSION AND REPORTING	HTML to PDF
	XML to PDF
	Text to PDF
	Tiff to PDF with CCITT Fax compression
	Text Extraction
MISC. FEATURES	PDF Page Templates
	Split, Merge and Extract functions
	PDF/A-1b : 2005 Compatible

DEVELOPER'S GUIDE

The Object Model



Every PDF document has a pages collection and pages in that collection.

Each page in return has a body which may contain images, text, a table and annotations.

For a full list of objects, methods and their properties, please refer to our online help pages.

Getting started with your first PDF generation

Creating PDF documents is a breeze with PDFTechLib.

Following a series of easy steps is all you will need to create PDF document.

Declare your references

First, you need to declare the references to PDFTechLib. Using Visual Studio.NET, add a reference to your project to PDFTechLib.dll assembly.

Add the following line at the beginning of your code

```
using PDFTech;
```

Create the document

Now create the instance of the PDFDocument class.

```
PDFDocument doc = new PDFDocument("sample.pdf" );
```

Start adding content

After creating the page, you need to add content. To do this set the properties of the body object. By setting the Body properties you can render text and images on the page.

Here is how to accomplish this:

```
doc.CurrentPage.Body.SetActiveFont("Arial",PDFFontStyles.Regular,12,
Charset.ANSI_CHARSET, Color.Red);
doc.CurrentPage.Body.AddText("Wow. That was easy!");
//now save the document
doc.Save();
```

Creating Content

PDFTechLib contains a wide array of tools and methods to create content on a PDF page. Content is images, text, shapes, curves, tables and lines.

The text can be aligned in any direction giving you the power to create effective documents.

When creating content on a page you can rely on the default coordinates or take into account the margin system. Read more about this in the “Page Margins” section.

Creating PDF/A compliant content

PDF/A is in fact a subset of PDF, leaving out PDF features not suited to long-term archiving. In addition, the standard places requirements on software products that read PDF/A files. A "conforming reader" must follow certain rules including following color management guidelines, using embedded fonts for rendering, and making annotation content available to users.

The Standard does not define an archiving strategy or the goals of an archiving system. It identifies a "profile" for electronic documents that ensures the documents can be reproduced the exact same way in years to come. A key element to this reproducibility is the requirement for PDF/A documents to be 100 % self-contained. All of the information necessary for displaying the document in the same manner every time is embedded in the file. This includes, but is not limited to, all content (text, raster images and vector graphics), fonts, and color information. A PDF/A document is not permitted to be reliant on information from external sources (e.g. font programs and hyperlinks).

Source: Wikipedia

Sample code:

```
PDFCreationOptions options = new PDFCreationOptions();
//Set the Compliance standard for PDF/A
options.ComplianceStandard = PDFStandards.PDFA1b;
PDFDocument MyPDF = new PDFDocument("PDFA.pdf", options);
MyPDF.CurrentPage.Body.AddText("This creates a PDF/A document.");
//Incompliant actions will throw exceptions when you call save method.
Example: Encryption transparent images etc. Most of the PDF/A requirements
will be taken care of automatically by the library. Such as font embedding,
ICC Profiles etc.
MyPDF.Save();
```

Flowing Text and Column Support

Our library exposes many unique methods to allow programmers who wish to generate long and content rich documents.

PDF documents offer a unique versatility in Content Management scenarios where documents need to be generated based on complex business rules. Insurance policies, corporate documents etc. are all generated based on rules or situations.

PDFTechLib does not use the page body as a graphical canvas object. All you have to do is create the object, set page size and column count, optionally set page margins, optionally set active font and size, then start adding text tagged text or image(s) as required. All formatting, word wrapping, placing text to columns, creating paragraphs will be done by the library. You do not need to calculate any complex positioning.

or

The “AddText” and “AddTaggedText” methods allow you to add text form the cursor position.

Sample:

```
PDFCreationOptions options = new PDFCreationOptions();
options.PageSize = PDFPageSize.A4;
options.SetMargin(50, 70, 50, 70);
PDFDocument MyPDF = new PDFDocument("Sample.pdf", options);
MyPDF.CurrentPage.Body.SetColumnCount(ColumnCount.TwoColumn);
MyPDF.CurrentPage.Body.SetActiveFont("Arial", PDFFontStyles.Underline, 12);
MyPDF.CurrentPage.Body.AddText("This is a flowing text. This can be as long
as you wish. Line breaks are interpreted as a paragraph.");
MyPDF.CurrentPage.Body.AddTaggedText("Here is <b>the</b> tagged text sample
<a href=\"http://www.pdf-technologies.com\"><font color='Navy'><u>Please
visit us.</u></font></a>");
MyPDF.CurrentPage.Body.AddImage(new PDFImage("sample.bmp"));
MyPDF.Save();
```

TrueType and OpenType Fonts

PDFTechLib supports the use of both TrueType and Open Type fonts.

The SetActiveFont method allows you to switch between any types of font throughout a document. PDFTechLib can automatically use the system fonts.

You can easily embed the fonts you wish into the final document so that the document looks exactly the way it was intended in its final destination.

How to use a TrueType font:

```
doc.CurrentPage.Body.SetActiveFont("verdana",PDFFontStyles.Regular, 12);
doc.CurrentPage.Body.AddText("This text uses Verdana font, size 12 and the font is embedded");
```

**** If a font needs to be bold or italic that font is searched for and embedded into the PDF document, if the font cannot be found then the font is imitated.**

Specifying an alternate font directory name

Make sure that you use the notation below to set the location of the alternate font directory. You have to use the "Server.MapPath" notation to ensure that you have access to this directory. Make sure that you have access to the entire path not just the folder at the end. This is especially useful in restricted environments where your code does not have access to the Windows/Fonts directory.

Here is how this can be done:

```
PDFCreationOptions options = new PDFCreationOptions();
options.LeftMargin = 20;
options.RightMargin = 0;
string fontDirectory = Server.MapPath(@"fonts");
options.FontDirectory = fontDirectory;
PDFDocument MyPDF = new PDFDocument("test.pdf", options);
MyPDF.DocumentInfo.Title = "test pdf";
MyPDF.AutoLaunch = true;

string fontName = "MyriadPro";
MyPDF.PageHeader.SetActiveFont(fontName, PDFFontStyles.Bold, 18);
MyPDF.PageHeader.SetTextAlignment(PDFTech.TextAlign.Center);
MyPDF.PageHeader.AddText("PDF Document Generation Sample");

MyPDF.CurrentPage.Body.SetActiveFont(fontName, PDFFontStyles.Bold, 28);
MyPDF.CurrentPage.Body.AddText("MyriadPro -  o O y Y");

MyPDF.CurrentPage.Body.AddText("Lorem ipsum dolor sit amet, consectetur
adipiscing elit. Ut id arcu sit amet massa malesuada sagittis. ");

MyPDF.Save();
```

Tagged Text

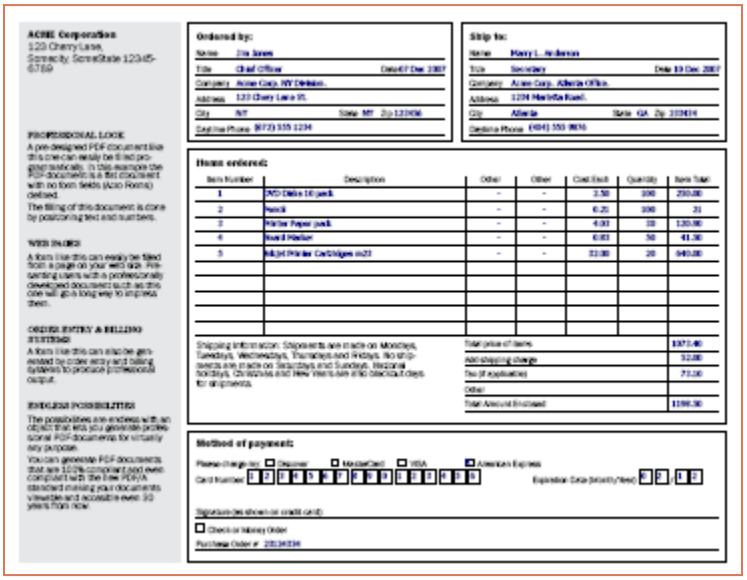
PDFTechLib provides support for a subset of HTML tags. The usage of XHTML is required.

The following methods support html formatted tagged text AddTaggedText, TaggedTextBox, Table.Cell.Value

Tag Name	Description
<b>	Bold
<i>	Italic
<st>	Strikethrough
 	new line
<u>	Underline
<font face="" size="" color="">	Font
<sup>	Superscript
<sub>	Subscript
<a>	Hyperlink
<p>	Paragraph
<hr>	Horizontal line

TextOut

Text Out is one of the basic functions of the library. You can place text anywhere on the page by specifying coordinates.



Sample:

```
PDFDocument MyPdf = new PDFDocument("Textoutsample.pdf");
MyPdf.CurrentPage.Body.TextOut(13, 133, 0, "Jim Jones", HorJust.Left,
VertJust.Top);
...
...
MyPdf.Save();
```

TABLES

One of the most common scenarios of usage for our library is data reporting. PDFTechLib supports cascading data styling in addition to basic features to support the generation of tables for reporting.

You may merge cells horizontally or vertically to create flexible table structures.

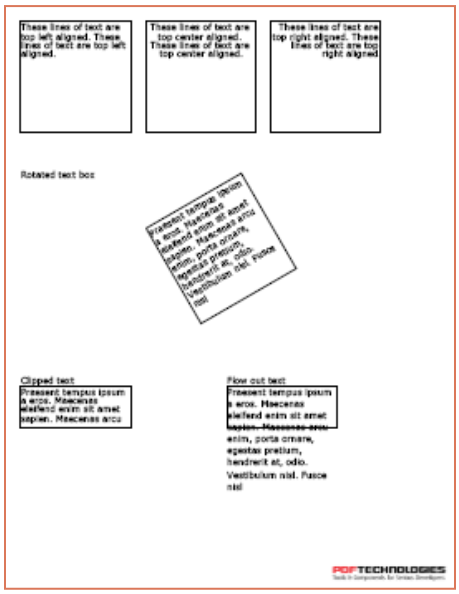
- Automatic column width calculations
- Automatic Row Height calculations
- Column header replication
- Text alignment in cells (left, right, center, up, down, middle)
- Flexible cell content (Text, Tagged text, image)
- Support for sophisticated styling for table and cell content (table, header, column header, row, column and cell based)
- Easy to understand object structure

Sample basic code:

```
PDFDocument MyPDF = new PDFDocument("Tables.pdf");
Table table = new Table(2,2);
table.DisplayHeader = true;
table.column(0).header.SetValue("Name");
table.column(1).header.SetValue("Surname");
table.cell(0, 0).SetValue ( "John");
table.cell(0, 1).SetValue("Smith");
table.cell(1, 0).SetValue("Joe");
table.cell(1, 1).SetValue("Doe");
MyPDF.CurrentPage.Body.DrawTable(table);
MyPDF.Save();
```

Text Boxes (Text Areas)

Text boxes are a powerful mechanism. You may position a text box anywhere on a page and fill it with text or tagged text. Word wrapping, Text alignment and positioning in this rectangle are automatically handled by the library. You may also create a body style for text boxes.



Sample:

```
PDFDocument MyPdf = new PDFDocument("TextBox.pdf");
MyPdf.CurrentPage.Body.AddTextArea(new RectangleF(27, 27, 200, 200), "These
lines of text are top left aligned. These lines of text are top left
aligned.", true);
MyPdf.Save();
```

USING TEMPLATES

PDFTechLib allows you to use pre-existing PDF documents as templates for new document creation. By using templates you can avoid having to code for certain visual aspects of the final output.

A good example is using a letterhead template for letters. Because the letterhead components such as the logo, header and footer are likely to stay the same, you can create the template using popular layout tools such as Adobe In-design and use the template over and over again to fill it with letter text.

Sample:

```
PDFCreationOptions options = new PDFCreationOptions();
options.TemplateFileName = "template1.pdf";
options.TemplatePageNumber = 1;
options.SetMargin(105, 200, 105, 133);
PDFDocument MyPdf = new PDFDocument("mydoc.pdf", options);
MyPdf.CurrentPage.Body.SetColumnCount(ColumnCount.TwoColumn);
MyPdf.CurrentPage.Body.AddText("What a nice template.");
MyPdf.Save();
```

Supported Page Sizes

PDFTechLib allows the creation of PDF documents using different page sizes. The size of the paper is controlled through its width and height properties. The orientation can also be controlled.

The following page sizes are supported:

Page Size	Pixel Height	Pixel Width
Letter	792	612
A4	842	595
A3	1190	842
Legal	1008	612
B5	728	516
C5	649	459
8x11	792	595
B4	1031	728
A5	595	419
Folio	936	612
Executive	756	522
EnvB4	1031	728
EnvB5	708	499
EnvC6	459	323
EnvDL	623	312
EnvMonarch	540	279
Env9	639	279
Env10	684	297
Env11	747	324

Unit of Measure

You can specify the precise position of text and images on a page using a variety of units of measure such as **inches, millimeters, centimeters,** and **pixels**. The default is pixels.

Example:

```
PDFCreationOptions options = new PDFCreationOptions();
options.UnitOfMeasure = UnitOfMeasure.Inch;
PDFDocument doc = new PDFDocument("sample.pdf" ,options);
doc.CurrentPage.Body.Shapes.DrawLine(new PDFPen(), 1, 1, 5, 1);
doc.Save();
```

Page margins

PDFTechLib allows you to define page margins. All positioning then becomes relative to these margins. To that effect, an image or text positioned at (0, 0) would actually be positioned at the top left corner defined by the page margins.

Sample:

```
PDFCreationOptions options = new PDFCreationOptions();
options.UnitOfMeasure = UnitOfMeasure.Inch;
options.LeftMargin = 0.5;
options.TopMargin = 1;
options.RightMargin = 0.5;
options.BottomMargin = 1;

PDFDocument doc = new PDFDocument("sample.pdf", options);

//X is relative to page margins. 0 means the position on the left margin.

int posX = 0;
//Y is relative to page margins. 0 means the position on the top margin.
int posY = 0;
//Orientation can be 0 to 360 degree
int textOrientation = 0;
doc.CurrentPage.Body.TextOut(posX, posY, textOrientation, "Wow, what a
wonderful margin.");
doc.Save();
```

UNICODE SUPPORT

Unicode characters are supported using a simple parameter.

Supported CJK fonts (Far East)

- MonotypeHeiMedium
- MonotypeSungLight
- SinoTypeSongLight
- HeiseiKakuGothicW5
- HeiseiMinchoW3
- HanyangSystemsGothicMedium
- HanyangSystemsShinMyeongJoMedium

Sample:

```
PDFDocument MyPDF = new PDFDocument("Unicode.pdf");
MyPDF.CurrentPage.Body.SetActiveFont("Tahoma", PDFFontStyles.Regular, 12,
Charset.GREEK_CHARSET);
MyPDF.CurrentPage.Body.AddText("εγγράφων σχετικά με τη μύγα από οποιαδήποτε.
εφαρμογή δικτύου. Η βιβλιοθήκη δεν");
MyPDF.Save();
```

Importing existing PDF files

Existing PDF files can be updated to include new content. You can insert new pages into an existing PDF document at any position.

Manipulation of PDF documents

Existing PDF documents can be manipulated using PDFTechLib. You may merge multiple PDF files or to split a PDF file to individual pages. You may extract text out of PDF documents and alter the document security. Extraction of individual pages from existing PDF documents is also possible.

Merging PDF files

Multiple PDF files can be with a single line of code.

Sample:

```
PDFDocument doc = new PDFDocument("merge.pdf");
doc.LoadPdf("sample.pdf","");
doc.LoadPdf("document.pdf", "");
doc.Save();
```

Splitting PDF files

A PDF file can be split into multiple individual pages. Each page can then be saved as an individual PDF document.

Use the following source to automatically split a PDF file into multiple pages:

Sample:

```
PDFOperation operation = new PDFOperation("sample.pdf", "");
operation.SplitFile("target.pdf");
```

Sample.pdf is the original file to be split. Target.pdf is the name that will be used for all individual files that will be created as a result of the split operation.

TEXT EXTRACTION

PDF documents with text content can be scanned for text and the document text can be extracted.

Text extraction from PDF documents is a very important feature that can be very useful in certain scenarios. In cases where documents need to be searched for a specific text or abstract needs to be generated from one or more PDF documents, this feature is priceless.

The library provides many different options to extract specific pages. It also provides an option to extract text in flowing or physical format.

The following example extracts all pages in the “sample.pdf” and writes to the “sample.txt” using flowing style.

Sample:

```
string Error = "";
//open the pdf document
PDFOperation operation = new PDFOperation("c:\\sample.pdf", "");

//retrieve page count
int pageCount = operation.GetPageCount(out Error);

if (Error == "")
{
      if (!operation.ExtractText(0, pageCount, LayoutType.Flowing,
EncodingType.UTF8, false, true, false, "C:\\sample.txt", out Error))
      Console.WriteLine(Error);
}
else
{
      Console.WriteLine(Error);
}
```

TEXT EXTRACTION FROM A SPECIFIC LOCATION OF A PDF DOCUMENT

Text can be extracted from a specific location of the PDF document. The rectangle denotes the coordinates on the page.

```
PDFOperation op = new PDFOperation("sample.pdf", "");
int pageNumber = 1;
string text = op.ExtractText(pageNumber, new RectangleF(880, 740, 120, 20),
UnitOfMeasure.Pixel);
op.Close();
Console.WriteLine(text);
```

NUMBERING PAGES

You may choose from many page numbering styles. The following sample demonstrates, page numbering applied to the first three pages as Roman Style (I,II,III…) . Then starting from page 4, page numbering is restarted from 1 as regular style (1,2,3..)

The last page numbering style will continue to the end of the document.

Sample:

```
//Add Roman style Page Numbering range for the intro pages
PageNumberingRange pnrintro = MyPDF.AddNumberingRange(0, 1); ;
pnrintro.NumberingStyle = NumberingStyle.LowerCaseRoman;

//Add Regular Page Numbering range starting from page 4
PageNumberingRange pnrsections = MyPDF.AddNumberingRange(3, 1); ;
pnrintro.NumberingStyle = NumberingStyle.Regular;
```

Links

You may embed URL links in PDF documents. Supported link types are URL links, File Links, Internal and External document links also called web links.

Sample:

```
PDFDocument MyPDF = new PDFDocument("Link.pdf");
MyPDF.CurrentPage.Body.AddTaggedText("<a href=\"http://www.pdf-
technologies.com\"><font color='Navy'><u>Please visit us.</u></font></a>");
MyPDF.Save();
```

Creating Bookmarks

A document outline can be generated to assist in document navigation. The document outline (also referred to as the bookmarks tree) is a hierarchy of master and child items in a document. Each page created is automatically added to this collection.

Sample:

```
PDFCreationOptions options = new PDFCreationOptions();
options.Viewer.PageMode = PageMode.UseOutlines;
PDFDocument MyPDF = new PDFDocument("Bookmarks.pdf", options);
for (int i = 0; i < 10; i++)
{
      if (i != 0)
            MyPDF.NewPage();
}
PDFGoToPageAction act = MyPDF.CreateGoToPageAction(0, 0);
PDFOutlineNode root = MyPDF.Outlines.Add(null, "Bookmarks", act,
Charset.ANSI_CHARSET);

root.Expanded = true;
for (int i = 0; i < MyPDF.Pages.Count; i++)
{
      // Create child bookmark
      act = MyPDF.CreateGoToPageAction(i, 0);
      PDFOutlineNode chapter = MyPDF.Outlines.AddChild(root, "Page " + (i +
1).ToString(), act, Charset.ANSI_CHARSET);
      chapter.Expanded = true;
      chapter.Color = Color.Blue;
      chapter.Style = FontStyle.Italic;
}
MyPDF.Save();
```

PDF Security and Encryption

Using PDFTechLib you may secure any PDF document. You can set the user and the owner passwords to restrict access to PDF documents.

The following access rights can be set on a PDF document:

- Allow Printing
- Allow Modify Contents
- Allow Copy
- Allow Modify Annotations
- Allow Fill In
- Allow Screen Readers
- Allow Assembly
- Allow Degraded Printing

PDFTechLib supports both the **40 bit** and the **128 bit** keys to encrypt PDF documents.

The following sample shows how to set the security over a PDF file:

```
PDFCreationOptions options = new PDFCreationOptions();
options.OwnerPassword = "topSecret";
options.UserPassword = "topSecret";
options.ProtectionOptions = PDFCryptoOptions.AllowPrinting;
options.EncryptionKeySize = EncryptionKeySize.Use128BitKey;
PDFDocument doc = new PDFDocument("sample.pdf", options);
```

If you would like to set multiple crypto options, use the **pipe (|)** character to separate out the different options:

The different options must all be declared on the same line, otherwise the last one will take effect.

```
PDFCreationOptions options = new PDFCreationOptions();
options.OwnerPassword = "topSecret";
options.UserPassword = "topSecret";

options.ProtectionOptions = PDFCryptoOptions.AllowPrinting |
PDFCryptoOptions.AllowCopy | PDFCryptoOptions.AllowFillIn;

options.EncryptionKeySize = EncryptionKeySize.Use128BitKey;
PDFDocument doc = new PDFDocument("sample.pdf", options);
```

PDF Actions

PDFTechLib supports the following actions.

- Go to Page
- Import data
- Java script
- Launch
- Remote go to
- Reset form
- Submit form
- URL
- Visible control

Annotations

Ability to place any type of annotation to PDF files.

Examples of annotation types that can be used are:

- PDFTextAnnotation
- PdfRubberStampAnnotation
- PdfLineAnnotation
- PdfFileAttachmentAnnotation
- PdfMarkupAnnotation
- PDFCircleAnnotation

Sample:

```
PDFDocument MyPDF = new PDFDocument("Annotations.pdf");
MyPDF.CurrentPage.Body.AddTextAnnotation("Comment", "Comment Annotation",
TextAnnotationIcon.Comment, new RectangleF(0, 0, 100, 100));
MyPDF.CurrentPage.Body.AddText("Sample comment");
MyPDF.Save();
```

Barcode Support

PDFTechLib supports the creation of 1D and 2D barcodes. These are:

EAN Family

Symbologies used to mark retail products and publications worldwide.

- Ean13
- Ean8
- Isbn
- Ismn
- Issn
- Jan13
- Jan8
- Scc14
- Sscc18
- UccEan128
- Upc_E
- UpcA

128 Full ASCII Family

Highly dense and general purpose full ASCII range symbologies used in the shipping and distribution industry.

- Code_128
- Telepen

3 of 9 Family

Symbologies used in automotive and defense industry.

- Code32
- Code39
- Code93
- NumlyNumber
- Pharmacode
- Pzn

2 of 5 Family

General purpose numeric symbologies used in warehouse, airline ticketing, medical, and automotive industry.

- Industrial2of5
- Interleaved2of5
- Itf14
- Opc

Plessy Family

Symbologies used for marking retail shelves.

- Msi

Codabar

Discrete and self-checking symbology used in blood banks, photo labs, and on FedEx air bills.

- Codabar
- Code11

Postal

Symbologies used by postal services for sorting mails.

- AustraliaPost
- DeutschePostIdentcode
- DeutschePostLeitcode
- Planet
- Postnet
- RoyalMail
- Kix
- SingaporePost
- SwissPostParcel
- UspsFim
- UspsHorizontalBars
- IntelligentMail
- UspsPicUccEan128
- UspsSackLabel
- UspsTrayLabel

2D Family

Encode more data in less space with 2D symbologies. These symbologies also offer better error correction mechanisms.

- AztecCode
- Code16k
- CompactPdf417
- DataMatrix
- MacroPdf417
- MicroPdf417
- PDF417
- QRCode
- Semacode

Sample code for Linear Barcodes:

```
PDFDocument doc = new PDFDocument("barcode.pdf");
doc.AutoLaunch = true;
PDFTech.Barcodes.Barcode1D.Code_128 code128 = new
PDFTech.Barcodes.Barcode1D.Code_128("1234567890", UnitOfMeasure.Pixel);
code128.CharSet = PDFTech.Barcodes.Code128.Auto;
if (code128.IsCodeValid())
{
      doc.CurrentPage.Body.Add1DBarcode(code128, 130, 100);
}
if (!doc.Save())
Console.WriteLine(doc.Error);
```

Sample code for 2D Barcodes:

```
PDFDocument doc = new PDFDocument("barcode.pdf");
doc.AutoLaunch = true;
PDFTech.Barcodes.Barcode2D.DataMatrix dataMatrix = new
PDFTech.Barcodes.Barcode2D.DataMatrix("123456PDFTech",UnitOfMeasure.Pixel);
dataMatrix.SymbolSize = PDFTech.Barcodes.DataMatrixSymbolSize.Size40X40;
if (dataMatrix.IsCodeValid())
{
      doc.CurrentPage.Body.Add2DBarcode(dataMatrix, 130, 100);
}
if (!doc.Save())
Console.WriteLine(doc.Error);
```

Acroform Filling

Acroforms allow the collection of form based information from users. Forms created using the Adobe design tools are fully supported by PDFTechLib. You can fill field values in PDF and extract values out of filled forms. You can also flatten the fields.

Sample:

```
PDFDocument doc = new PDFDocument("formfill.pdf");
string formFile = ("fw9.pdf");
doc.Pages.Delete(doc.CurrentPage);
doc.LoadPdf(formFile, "");

(doc.AcroForm.Fields["f1_01(0)"] as PDFEdit).Text = "Jim Smith";
(doc.AcroForm.Fields["f1_02(0)"] as PDFEdit).Text = "Acme Corp. Inc.";

(doc.AcroForm.Fields["c1_01(0)"] as PDFCheckBox).Checked = true;

doc.AcroForm.FlattenFormFields = true;
doc.Save();
```

Creating Acroforms

You can also create Acroforms on the fly. PDFTechLib supports the following field types when you create the form.

- PDFButton
- PDFCheckBox
- PDFEdit
- PDFRadioButton
- PDFComboBox
- PDFListBox

Sample:

```
PDFDocument MyPDF = new PDFDocument("AcroForm.pdf");
MyPDF.CurrentPage.Body.TextOut(50, 50, 0, "State :");
PDFComboBox cmbState = MyPDF.AcroForm.AddPDFComboBox("cmbState", new
RectangleF(100, 50, 300, 20), false);
cmbState.Items.Add(new PDFListItem("Georgia", "GA"));
cmbState.Items.Add(new PDFListItem("NewYork", "NY"));
cmbState.Items.Add(new PDFListItem("Washington", "WA"));
cmbState.Items.Add(new PDFListItem("Florida", "FL"));
MyPDF.Save();
```

Conversion and Reporting

HTML to PDF

Using PDFTechLib programmers can convert HTML documents to PDF documents.

HTML is one of the most common document formats. Converting HTML to PDF is a very sophisticated process. We use MS internet Explorer API on the back-end to achieve this conversion.

The following are the restrictions.

- DTD formats cannot be supported.
- Gradient Styled objects cannot be converted.
- Pages might be cropped, if the server's screen resolution is smaller than the page size.

Sample:

```
PDFDocument MyDocument = new PDFDocument("HTMLSample.pdf");
MyDocument.ImportHTML("http://www.samplesite.com/mypage.htm");
MyDocument.Save();
```

XML to PDF

Using PDFTechLib programmers can convert XML- XSL documents to PDF documents.

Sample:

```
PDFDocument MyDocument = new PDFDocument("XMLSample.pdf");
string path=Environment.CurrentDirectory;
MyDocument.ImportXml("sample.xml", "sample.xsl", path);
MyDocument.Save();
```

Tiff to PDF with CCITT Fax compression

Using PDFTechLib you can convert Tiff files to PDF documents with CCITT fax compression.

Tiff documents are generally multi paged scanner or fax outputs. Processing this image type is a complex operation. Our library exposes many functions to support the manipulation of TIFF documents.

The following sample opens a TIFF image and splits it into multiple frames. It then places each image on a separate page. Each image is automatically resized to fit the page.

Sample:

```
PDFDocument MyDocument = new PDFDocument("TiffSample.pdf");
MyDocument.TiffToPDF("sample.tiff");
MyDocument.Save();
```

PDFTECHNOLOGIES
Tools & Components for Serious Developers

How to save Acroform Fields using a different name.

You can read an existing PDF document and extract the Acroform fields to create a new PDF document.

Sample:

```
PDFCreationOptions options = new PDFCreationOptions();
options.Viewer.ViewerPreferences = ViewerPreference.CenterWindow;
PDFDocument doc = new PDFDocument(@"FormFill.pdf", options);
doc.AutoLaunch = true;
string formFile = (@"C:\SampleApp\SampleApp\Resource\fw9.pdf");
doc.Pages.Delete(doc.CurrentPage);
doc.LoadPdf(formFile, "");
(doc.AcroForm.Fields["f1_01(0)"] as PDFEdit).Text = "Jim Smith";
(doc.AcroForm.Fields["f1_02(0)"] as PDFEdit).Text = "Acme Corp. Inc.";
(doc.AcroForm.Fields["f1_03(0)"] as PDFEdit).Text = "Lorem dolor ";
PDFEdit pdfEdit = (doc.AcroForm.Fields["f1_04(0)"] as PDFEdit);
pdfEdit.Text = "New control value.";
pdfEdit.Name = "NewControlName";
doc.AcroForm.FlattenFormFields = true;



PDFCreationOptions options2 = new PDFCreationOptions();
options2.LeftMargin = 0;
options2.RightMargin = 0;
PDFDocument MyPDF = new PDFDocument(@"AcroForm.pdf", options2);
MyPDF.DocumentInfo.Title = "AcroForm creation";
MyPDF.AutoLaunch = true;
MyPDF.PageHeader.SetTextAlignment(TextAlign.Center);
MyPDF.PageHeader.SetActiveFont("Arial", PDFFontStyles.Bold, 18);
MyPDF.PageHeader.AddText("AcroForm Generation Sample");
MyPDF.PageHeader.SetTextAlignment(TextAlign.Left);
PDFPage page = MyPDF.CurrentPage;
int captionLeft = 50;
int valueLeft = 250;
int ycounter = 100;
int yinterval = 30;
int crtlHeight = 20;
page.Body.SetActiveFont("Arial", PDFFontStyles.Bold, 14);
//Draw for Caption
page.Body.TextOut(captionLeft, ycounter, 0, "Personal information form");
```

```
ycounter += yinterval;
page.Body.Shapes.DrawLine(new PDFTech.PDFPen(), 50, ycounter,
page.Body.Dimension.Width - 50, ycounter);
page.Body.SetActiveFont("Arial", PDFFontStyles.Bold, 12);

ycounter += yinterval;
page.Body.TextOut(captionLeft, ycounter, 0, "First Name :");
PDFEdit txtFirstName = MyPDF.AcroForm.AddPDFEdit("txtFirstName", new
RectangleF(valueLeft, ycounter, 300, crtlHeight), "Jim");
txtFirstName.Font.Style = PDFFontStyles.Regular;
txtFirstName.Required = true;
ycounter += yinterval;
page.Body.TextOut(captionLeft, ycounter, 0, "Last Name :");

PDFEdit txtLastName = MyPDF.AcroForm.AddPDFEdit("txtLastName", new
RectangleF(valueLeft, ycounter, 300, crtlHeight), "Jones");
txtLastName.Font.Style = PDFFontStyles.Regular;

ycounter += yinterval;
page.Body.TextOut(captionLeft, ycounter, 0, pdfEdit.Name);
PDFEdit txtaddress = MyPDF.AcroForm.AddPDFEdit(pdfEdit.Name, new
RectangleF(valueLeft, ycounter, 300, crtlHeight), pdfEdit.Text);

txtaddress.Font.Style = PDFFontStyles.Regular;

MyPDF.Save();
```

Image compression

You can compress the images used in PDF documents.

Sample:

```
PDFDocument MyDocument = new PDFDocument("Sample.pdf");
PDFImage image = new PDFImage("extrelogo.bmp");
image.Compression = ImageCompressionType.Flate;
MyDocument.CurrentPage.Body.AddImage(image, 0, 13);
MyDocument.Save();
```

Adding an image to cover the entire page

The important factor here is to calculate the size of the image before placing the image on the page. PDFTechLib will look at the actual image size when adding it to a page.

Sample:

```
PDFCreationOptions options = new PDFCreationOptions(); options.UnitOfMeasure =
UnitOfMeasure.Mm;

options.TopMargin = 0;
options.LeftMargin = 0;
options.RightMargin = 0;
options.BottomMargin = 0;
PDFDocument MyDocument = new PDFDocument(@"fullsize.pdf", options);
MyDocument.AutoLaunch = true;
int imageWidth = 516;
int imageHeight = 728;
Bitmap resizedImage = new Bitmap(imageWidth, imageHeight);
Bitmap originalImage = new Bitmap(@"test.bmp");
using (Graphics graphics = Graphics.FromImage((Image)resizedImage))
{
graphics.DrawImage(originalImage, 0, 0, imageWidth, imageHeight);
}

PDFImage image = new PDFImage(resizedImage);
MyDocument.CurrentPage.Body.AddImage(image);
MyDocument.Save();
```

How to get the Width and Height of a PDF document

You can get the width and height of a PDF document using the methods below.

Sample:

```
PDFDocument MyDocument = new PDFDocument("Sample.pdf");
PDFDocInfo Info = MyPDF.DocumentInfo;
MyPDF.CurrentPage.PageSetupInfo.PageWidth;
MyPDF.CurrentPage.PageSetupInfo.PageHeight;
MyDocument.Save();
```

How to add a rotated text to a PDF document

You can add a rotated text to a PDF document using the method below:

Sample:

```
PDFDocument MyDocument = new PDFDocument("Sample.pdf");
MyDocument.CurrentPage.Body.TextOut(220, 293, 0, "Rotated text box");
MyDocument.CurrentPage.Body.SetTextAlignment(TextAlign.Left);
MyDocument.CurrentPage. Body.AddTextArea(new RectangleF(253, 400, 200, 200),
"These lines of text are left aligned and rotated towards the left.", true,
false, 30);
MyDocument.CurrentPage.Body.TextOut(77, 665, 0, "Clipped text");
MyDocument.Save();
```

How to add a password to a PDF document

You can add a password to an existing PDF document. You must make sure that the User Password and Owner Password is different. They cannot be the same.

Sample:

```
PDFCreationOptions options = new PDFCreationOptions();
options.UserPassword = "topsecret";
options.OwnerPassword = "ownerpassword";
options.EncryptionKeySize = EncryptionKeySize.Use128BitKey;
PDFDocument MyDoc = new PDFDocument("Encryption.pdf", options);
MyDoc.AutoLaunch = true;
MyDoc.PageHeader.SetTextAlignment(TextAlign.Center);
MyDoc.PageHeader.AddText("Encryption Sample");
MyDoc.PageHeader.Shapes.DrawLine(new PDFPen());
MyDoc.PageFooter.SetActiveFont("Helvetica", PDFFontStyles.Regular, 8,
Charset.ANSI_CHARSET, Color.Black);
MyDoc.PageFooter.Shapes.DrawLine(new PDFPen());
MyDoc.PageFooter.AddText("Encryption Sample");
PDFPage page = MyDoc.CurrentPage;
page.Body.SetTextAlignment(TextAlign.Justified);
page.Body.SetActiveFont("Tahoma",PDFFontStyles.Regular, 12);
page.Body.AddText(File.ReadAllText("short-lorem.txt"));
page.Body.Shapes.SetTransparency(50);
page.Body.SetActiveFont("Helvetica", PDFFontStyles.Bold, 80,
Charset.ANSI_CHARSET, Color.Red);
page.Body.TextOut(0, 300, 20, "Top Secret", HorJust.Left, VertJust.Center);
page.Body.Shapes.SetTransparency(0);
MyDoc.Save();
```

PDFTECHNOLOGIES
Tools & Components for Serious Developers

How to add a link to a PDF document

Using the method below, you can add links to a PDF document.

You can add a Text link, text web page link or GoToPage link.

Sample:

```
PDFDocument MyPDF = new PDFDocument("Links.pdf");
PDFPage page = MyPDF.CurrentPage;

Table table = new Table(5, 2);
table.column(1).style.paddingLeft = 26;
table.column(0).style.paddingLeft = 26;
table.style.borderColor = Color.Black;
table.style.borderType = borderType.solid;
table.rowHeight = 100;
table.style.textAlign = TextAlignment.left;

table.cell(0, 0).SetValue("Rectangular area links");
table.cell(0, 0).style.backgroundColor = Color.Gray;
table.cell(0, 1).SetValue("Text links");
table.cell(0, 1).style.backgroundColor = Color.Gray;
table.row(0).height = 50;

table.cell(1, 0).SetValue("Text web link");
PDFAction act = MyPDF.CreatePDFURLAction("http://www.pdf-technologies.com");
table.cell(1, 0).SetAction(act);
table.cell(3, 0).SetValue("Text in document link ");
act = MyPDF.CreateGoToPageAction(1, 0);
table.cell(3, 0).SetAction(act);
table.cell(4, 0).SetValue("Text external pdf document link");
act = MyPDF.CreatePdfRemoteGoToAction("taggedText.pdf", true, 0);
table.cell(4, 0).SetAction(act);
table.cell(1, 1).SetValue("<a href='http://www.pdf-technologies.com'><u>Text
web link</u></a>", true);
MyPDF.CurrentPage.Body.DrawTable(table);
MyPDF.Save();
```

Adding an Annotation to a PDF document

You can use the sample code below to add a Text or Rubber stamp annotation to an existing PDF document.

Sample:

```
PDFDocument MyDocument = new PDFDocument("Sample.pdf");
PDFPage page = MyPDF.CurrentPage;
page.Body.SetActiveFont("Helvetica", PDFFontStyles.Bold, 10);
page.Body.TextOut(26, 60, 0, "Text Annotations");

//Text annotations
page.Body.AddTextAnnotation("Comment", "Comment Annotation",
TextAnnotationIcon.Comment, new RectangleF(26, 80, 100, 100));
page.Body.SetActiveFont("Helvetica", PDFFontStyles.Regular, 8);
page.Body.TextOut(26, 106, 0, "Comment");
page.Body.AddTextAnnotation("Insert", "Insert Annotation",
TextAnnotationIcon.Insert, new RectangleF(239, 80, 100, 100));
page.Body.TextOut(239, 106, 0, "Insert");
page.Body.AddTextAnnotation("Key", "Key Annotation", TextAnnotationIcon.Key,
new RectangleF(346, 80, 100, 100));
page.Body.TextOut(346, 106, 0, "Key");

//Rubber stamp annotations
page.Body.SetActiveFont("Helvetica", PDFFontStyles.Bold, 10);
PDFRubberStampAnnotation rsa = page.Body.AddRubberStampAnnotation("Approved",
"Approved annotation", RubberStampAnnotationIcon.SBApproved, new
RectangleF(26, 186, 100, 30));
rsa.Opacity = 50;
page.Body.SetActiveFont("Helvetica", PDFFontStyles.Regular, 8);
page.Body.TextOut(26, 240, 0, "Approved");
page.Body.AddRubberStampAnnotation("Completed", "Completed annotation",
RubberStampAnnotationIcon.SBCompleted, new RectangleF(160, 186, 100, 30));
page.Body.TextOut(160, 240, 0, "Completed");
page.Body.AddRubberStampAnnotation("Confidential", "Confidential annotation",
RubberStampAnnotationIcon.SBConfidential, new RectangleF(293, 186, 100, 30));
page.Body.TextOut(293, 240, 0, "Confidential");
MyDocument.Save();
```

Custom Watermarks

The sample below shows how a watermark can be added to an existing PDF document.

Sample:

```
PDFCreationOptions options = new PDFCreationOptions();
options.Watermark.SetText("Approved");
PDFDocument MyDocument = new PDFDocument("Watermarks.pdf",options);
MyDocument.PageHeader.SetTextAlignment(TextAlign.Center);
MyDocument.PageHeader.AddText("Watermarks Sample");
MyDocument.PageHeader.Shapes.DrawLine(new PDFPen());
MyDocument.PageFooter.Shapes.DrawLine(new PDFPen());
MyDocument PDF.PageFooter.AddText("Watermarks Sample");
MyDocument.AutoLaunch = true;
MyDocument.DocumentInfo.Title = "Demo [Watermarks]";
MyDocument.Save();
```

How to create a PDF document using a stream:

The sample below shows how a PDF document can be created using a stream.

Sample:

```
PDFDocument MyDocument = new PDFDocument("TestDocument.pdf");
MyDocument.AutoLaunch = true;
FileStream fileStream = new FileStream("presentation.pdf", FileMode.Open,
FileAccess.Read, FileShare.Read);
MyDocument.LoadPdf(fileStream, "");
fileStream.Close();
MyDocument.Save();
```

Extracting attachments to a folder:

Use the code below to extract PDF atachments to a folder. Please make sure that the folder is not protected or secured.

sample:

```
string sourceFileName = @"C:\pdf\Attachments.pdf";
string outputDirectory = @"c:\pdf\folder";
PDFDocument.ExtractAttachments(sourceFileName, outputDirectory);
```

Accessing attached documents in PDF:

Using the code below, you can access the attaced documents in a PDF document. The GetPDFAttachments method will return a collection. You can access the name and contents of the attachment from this collection.

sample:

```
string sourceFileName = @"C:\temp\Attachments.pdf";
string outputDirectory = @"c:\temp\ext";
List <PDFAttachmentFileContent> attachments = PDFDocument.GetPDFAttachments(
sourceFileName );

for (int i = 0; i < attachments.Count; i++){
    PDFAttachmentFileContent attachment = attachments[i];
    String name = attachment.Name;
    Byte[] content = attachment.Content;
}
```

Printing the specified pdf document:

Use the code below to print the specified PDF document.

sample:

```
ProcessStartInfo info = new ProcessStartInfo();
info.Verb = "print";
info.FileName = @"c:\output.pdf";
info.CreateNoWindow = true;
info.WindowStyle = ProcessWindowStyle.Hidden;

Process p = new Process();
p.StartInfo = info;
p.Start();

p.WaitForInputIdle();
System.Threading.Thread.Sleep(3000);
if (false == p.CloseMainWindow())
    p.Kill();
```