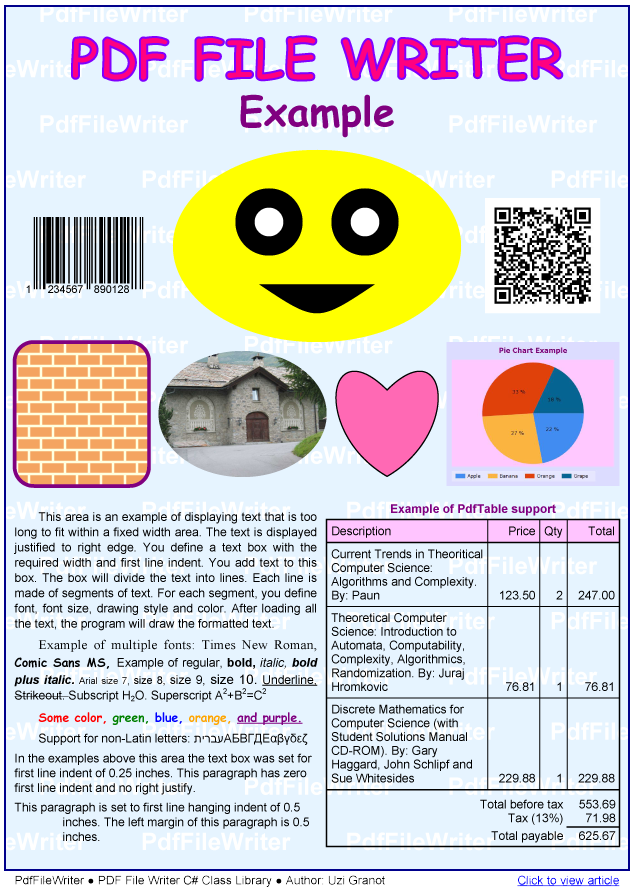
PDF File Writer C# Class Library (Version 1.15.1)

PDF File Writer is a C# .NET class library allowing applications to create PDF files. Version 1.15



1. Introduction

The PDF File Writer C# class library PdfFileWriter allows you to create PDF files directly from your .net application. The library shields you from the details of the PDF file structure. To use the library, you need to add a reference to the attached PdfFileWriter.dll class library file, add a using PdfFileWriter statement in every source file that is using the library and include the PdfFileWriter.dll with your distribution. For more details go to [2.19 Installation](http://www.codeproject.com/Articles/570682/PDF-File-Writer-Csharp-Class-Library-Version?display=Print#Installation). Alternatively, you can include the source code of the library with your application and avoid the need to distribute a separate data link library file. The minimum development requirement is .NET Framework 4.0 (Visual Studio 2010).

|  |
| --- |
| Version 1.15 Enhancements   * PDF document information dictionary. The PDF reader displays this information in the Description tab of the document properties. The information includes: Title, Author, Subject, Keywords, Created date and time, Modified date and time, the Application that produced the file, the PDF Producer. Section [2.18 Document Information Dictionary](http://www.codeproject.com/Articles/570682/PDF-File-Writer-Csharp-Class-Library-Version?display=Print#InformationDictionary). * PdfImage class was significantly modified. A new class PdfImageControl class was added to control the properties of the saved image. Image source can be an image file or .NET Image class or a Boolean array for black and white images. Saved image can be Jpeg format, indexed bitmap, gray bitmap or black and white bitmap. Bitmap format is lossless compressed image. Section [2.4. Image Support](http://www.codeproject.com/Articles/570682/PDF-File-Writer-Csharp-Class-Library-Version?display=Print#ImageSupport). |

The PDF File Writer C# class library supports the following PDF document's features:

* Graphics: drawing lines, rectangles, polygons, Bezier curves, foreground and background color, patterns and shading. Section [2.1 Coordinate System](http://www.codeproject.com/Articles/570682/PDF-File-Writer-Csharp-Class-Library-Version?display=Print#CoordinateSystem).
* Image: drawing raster (Bitmap) images and vector (Metafile) images. Section [2.4. Image Support](http://www.codeproject.com/Articles/570682/PDF-File-Writer-Csharp-Class-Library-Version?display=Print#ImageSupport).
* Text: drawing text lines and text in columns. Section [2.3 Language Support](http://www.codeproject.com/Articles/570682/PDF-File-Writer-Csharp-Class-Library-Version?display=Print#LanguageSupport).
* Barcode: support for Barcode 128, Barcode 39, Barcode interleaved 2 of 5, Barcode EAN13 and Barcode UPC-A. Section [2.5 Barcode Support](http://www.codeproject.com/Articles/570682/PDF-File-Writer-Csharp-Class-Library-Version?display=Print#BarcodeSupport).
* QR Code: support for two dimensions barcode. Section [2.8 QR Code Support](http://www.codeproject.com/Articles/570682/PDF-File-Writer-Csharp-Class-Library-Version?display=Print#QRCodeSupport).
* Encryption: support for AES-128 encryption. Section [2.6 Encryption Support](http://www.codeproject.com/Articles/570682/PDF-File-Writer-Csharp-Class-Library-Version?display=Print#EncryptionSupport).
* Web Link: Web link interactive support. Section [2.7 Web Link Support](http://www.codeproject.com/Articles/570682/PDF-File-Writer-Csharp-Class-Library-Version?display=Print#WeblinkSupport).
* Bookmark: Support for document outline. Section [2.9 Bookmark Support](http://www.codeproject.com/Articles/570682/PDF-File-Writer-Csharp-Class-Library-Version?display=Print#BookmarkSupport).
* Charts: Support for Microsoft Charting. Section [2.10 Charting Support](http://www.codeproject.com/Articles/570682/PDF-File-Writer-Csharp-Class-Library-Version?display=Print#ChartingSupport).
* Print to PDF: Create a PDF document from PrintDocument process. Section [2.11 PrintDocument Support](http://www.codeproject.com/Articles/570682/PDF-File-Writer-Csharp-Class-Library-Version?display=Print#PrintDocumentSupport).
* Display data tables. Section [2.12 Data Table Support](http://www.codeproject.com/Articles/570682/PDF-File-Writer-Csharp-Class-Library-Version?display=Print#DataTableSupport)
* Play video files. Section [2.13 Play Video Files](http://www.codeproject.com/Articles/570682/PDF-File-Writer-Csharp-Class-Library-Version?display=Print#PlayVideoFiles)
* Play sound files. Section [2.14 Play Sound Files](http://www.codeproject.com/Articles/570682/PDF-File-Writer-Csharp-Class-Library-Version?display=Print#PlaySoundFiles)
* Attach data files. Section [2.15 Attach Data Files](http://www.codeproject.com/Articles/570682/PDF-File-Writer-Csharp-Class-Library-Version?display=Print#AttachDataFiles)
* Reorder pages. Section [2.16 Reorder Pages](http://www.codeproject.com/Articles/570682/PDF-File-Writer-Csharp-Class-Library-Version?display=Print#ReorderPages)
* PDF document output to a file or to a stream. Section [2.17 Document Destination](http://www.codeproject.com/Articles/570682/PDF-File-Writer-Csharp-Class-Library-Version?display=Print#DocumentDestination).
* PDF document information dictionary. The PDF reader displays this information in the Description tab of the document properties. The information includes: Title, Author, Subject, Keywords, Created date and time, Modified date and time, the Application that produced the file, the PDF Producer. Section [2.18 Document Information Dictionary](http://www.codeproject.com/Articles/570682/PDF-File-Writer-Csharp-Class-Library-Version?display=Print#InformationDictionary).

Creating a PDF is a six steps process.

* Step 1: Create one document object PdfDocument.
* Step 2: Create resource objects such as fonts or images (i.e. PdfFont or PdfImage).
* Step 3: Create page object PdfPage.
* Step 4: Create contents object PdfContents.
* Step 5: Add text and graphics to the contents object (using PdfContents methods).
* Repeat steps 3, 4 and 5 for additional pages
* Step 6: Create your PDF document file by calling CreateFile method of PdfDocument.

Step 5 is where most of your programming effort will be spent. Adding contents is achieved by calling the methods of PdfContents class to render graphics and text. The contents class has a rich set (about 100) of methods for adding text and graphics to your document.

The demo program attached to this article is the test program developed to debug the library. The TestPdfFileWriter has six buttons on the main screen. Five buttons to produce examples of PDF files and one button to display all fonts available on your computer. The first button “Article Example” creates the PDF file displayed at the top of this article. Section [3. Development Guide by Example](http://www.codeproject.com/Articles/570682/PDF-File-Writer-Csharp-Class-Library-Version?display=Print#DevelopmentGuide).

As stated before, the PdfFileWriter C# class library shields you from the complexities of the PDF file structure. However, good understanding of PDF file is always an advantage. Adobe PDF file specification document available from Adobe website: [“PDF Reference, Sixth Edition, Adobe Portable Document Format Version 1.7 November 2006”](http://www.adobe.com/content/dam/Adobe/en/devnet/acrobat/pdfs/pdf_reference_1-7.pdf). It is an intimidating 1310 pages document. I would strongly recommend reading Chapter 4 Graphics and sections 5.2 and 5.3 of the Text chapter 5.

If you want to analyze the PDF files created by this project, or if you want to understand PDF file structure in general, you can use the demo program attached to my previous article ["PDF File Analyzer With C# Parsing Classes"](http://www.codeproject.com/Articles/450254/PDF-File-Analyzer-With-Csharp-Parsing-Classes). This article provides a concise overview of the PDF specifications.

2. PDF File Writer Library General Notes

### 2.1. Coordinate system and Unit of Measure

The PDF coordinate system origin is at the bottom left corner of the page. The X-axis is pointing to the right. The Y-axis is pointing in upward direction.

The PDF unit of measure is point. There are 72 points in one inch. The PDF File writer allows you to select your own unit of measure. All methods arguments representing position, width or height must be in your unit of measure. There are two exceptions: font size and resolution. Font size is always in points. Resolution is always in pixels per inch. The PDF File Writer converts all input arguments to points. All internal measurement values and calculations are done with double precision. At the final step when the PDF file is created, the values are converted to text strings. The conversion precision is six digits. The conversion formula used is:

// Value is Double

String Result = ((Single) Math.Round(Value, 6, MidpointRounding.AwayFromZero)).ToString();

### 2.2. Decimal separator

PDF readers such as Adobe Acrobat expect real numbers with a fraction to use period as the decimal separator. Some of the world regions use other decimal separators such as comma. Since Version 1.1 of the PDF File Writer library will use period as decimal separator regardless of regional setting of your computer.

### 2.3. Language support, fonts and character sets

The PDF File Writer library is using TrueType fonts. Please note Section 5.5 Simple Fonts of the PDF specifications. TrueType fonts are defined as simple fonts with the property "Glyphs in the font are selected by single-byte character codes obtained from a string that is shown by the text-showing operators". In other words this library is limited to one byte encoding, 0 to 255. Not all values within this range are available. Control codes 0 to 31 and 128 to 159 are excluded. The result is that this class library supports characters in the range 32 to 126 and 160 to 255. These characters are encoded using WinAnsiEncoding. It is Windows Code Page 1252, often called the “Windows ANSI” encoding. This is the standard Windows encoding for Latin text in Western writing systems.

Most TrueType fonts such as Arial Unicode MS supports character values greater than 255. The PDF File Library allows you to perform a substitution. You can use any Unicode character and map it into the available one byte range. It is performed by CharSubstitution method of the PdfFont class. Three characters within the allowable range: space (32), non breaking space (160) and non breaking dash (173) cannot be replaced. In other words you can use for substitution range 33 to 126 and range 161 to 255 except non breaking hyphen 173. Some languages such as Marathi use two Unicode characters to display one character on the screen. In these cases, substitution does not work.

PdfFont.CharSubstitution(int OriginStart, int OriginEnd, int DestStart);

For example, Unicode Greek low case letters are encoded starting with 945. To use the first 6 letters we will map them to position 161.

ArialNormal = new PdfFont(Document, "Arial", FontStyle.Regular, true);

ArialNormal.CharSubstitution(945, 950, 161);

To print them:

Contents.DrawText(ArialNormal, FontSize, TextPos, BaseLine, "&alpha;&beta;&gamma;&delta;&epsilon;&zeta;");

In versions prior to 1.4, the characters in the text string had to be recalculated. Starting with Version 1.4 you just put the Unicode string as is. If the text string contains characters that were not substituted, the program will fail with exception.

Another example, Unicode Hebrew letters are encoded in a block from 1488 to 1514. To use Hebrew letters we map them again to 161.

ArialNormal.CharSubstitution(1488, 1514, 161);

Hebrew is a right to left language. PDF readers print left to right. If we print in the same way as Greek the result will be reversed string. To get the correct result, the PDF File Writer has a new method PdfContents.ReverseString(String Text).

Contents.DrawText(ArialNormal, FontSize, TextPos, BaseLine, Contents.ReverseString( "עברית");

The support for right to left languages of this package is limited. You can print single lines. To understand the complexities of Bidi support refer to [Unicode Bidirectional Algorithm](http://www.unicode.org/reports/tr9/). You can draw text but you have to remember that PDF prints left to right and text position is to the left of the string by default. Using DrawText with TextJustify.Right option will be more natural for Hebrew.

Contents.DrawText(ArialNormal, FontSize, TextPos, BaseLine, TextJustify.Right, Contents.ReverseString( "עברית");

The ReverseString method reverses the character order. It does not know the language. In other words you cannot reverse a string that contains the letters of right to left language together with numbers or English words or even brackets ()[]<>{}. Another limitation is TextBox class cannot be used.

### 2.4. Image Support

Displaying images in the PDF document is handled by PdfImage class and by PdfImageControl class. Image source can be an image file or .NET Image derived class or a Boolean array for black and white images. Images are saved to the PDF file in Jpeg format, indexed bitmap, gray bitmap or black and white bitmap. Bitmap format is lossless compression. The PdfImageControl controls the process of saving images. Color pictures should be saved in Jpeg format. To control image data size you can reduce resolution or change image quality. Color pictures can be saved in shades of gray. Data size is cut by three but you lose the color. If the image was created programmatically as in charts and the number of colors is less than 256 the image can be saved as indexed bitmap. Each color is represented by one byte (or less) compare to 3 bytes. This can result is very significant file size reduction. The ChartExample.pdf file is reduced from 642KB to 72KB. If the image is black and white as in PdfPrintDocument images of text, the image can be saved as BWImage. In the case of PrintExample.pdf the Jpeg file is 1795KB and the black and white version is 66KB.

Three other classes within the PdfFileWriter library use the PdfImage support. The classes are: PdfChart, PdfQRCode and PdfPrintDocument.

Images are PDF resources. When you create an image resource, the software loads the image data directly into the PDF output file. In other words, the image data is not resident in memory allowing for a large number of images without reducing program memory space. There are three constructors to the PdfImage class.

// load image from file

PdfImage Image = new PdfImage(PdfDocument Document, String ImageFileName, PdfImageControl ImageControl);

// load image from memory class (Image or Bitmap)

PdfImage Image = new PdfImage(PdfDocument Document, Image MemoryImage, PdfImageControl ImageControl);

// load black and white image from Boolean array

PdfImage Image = new PdfImage(PdfDocument Document, Boolean[,] BWImage, PdfImageControl ImageControl);

The arguments are:

* Document: The PdfDocument object of your PDF document.
* ImageFileName: Image file is any file that can be loaded into .NET Image derived class. Image derived class is either a Bitmap for raster image or a Metafile image for vector image. If file name is entered and the extension is .emf or .wmf the file is loaded into a Metafile, otherwise the file is loaded into a Bitmap.
* MemoryImage: is either a Bitmap or a Metafile that was created in memory by your application.
* BWImage: is either a Boolean two dimension array. False represent black and true represent white. The PdfQRCode class is using this format to represent QR image.
* ImageControl: is an optional argument that holds image creation properties.

PdfImageControl has the following properties. All properties are optional.

* SaveAs: Save image as Jpeg, IndexedImage, GrayImage or BWImage.
* CropRect: Crop rectangle is the rectangle defines the image area to be cropped. Note: the rectangle coordinates are in .net standard. Origin is at top left and Y axis is pointing down. The crop rectangle can be Rectangle. In this case the dimensions are in pixels. The crop rectangle must be contained within the image.
* CropPercent: Crop rectangle as above but the dimensions are in percent of the image width and height.
* Resolution: Image resolution sets the image resolution provided that it is smaller than the resolution of source image. Resolution is specified in pixels per inch. Reduced resolution means smaller PDF file.
* ImageQuality: It is an integer in the range of 0 to 100 representing poor to best quality image, or DefaultQuality (-1) meaning save image with default quality. Lower image quality means smaller PDF file. If image quality is not define it is taken as default quality. The .net frame work default save method is using quality factor of 75. This number is not documented by Microsoft. It was calculated by experimentation and Internet searches.
* GrayToBWCutoff: Converting a bitmap to black and white is two step process. The first is standard color to shades of gray. The second sets the pixel to black if shade of gray is below cutoff and to white if it is above. Valid values are 1 to 99.
* ReverseBW: In Gray or BW images the color is reversed if this property is true.

Other methods of PdfImage.

The ImageSize method returns the largest rectangle with correct aspect ratio that will fit in a given area.

SizeD ImageSize(Double Width, Double Height);

The ImageSizePosition method returns the largest rectangle with correct aspect ratio that will fit in a given area and position it based on ContentAlignment enumeration.

ImageSizePos ImageSizePosition(Double Width, Double Height, ContentAlignment Alignment);

Finally to draw the image into the contents use DrawImage method. If you want the image to maintain correct aspect ratio use ImageSize or ImageSizePosition to calculate the width and height. If the ratio of width and height is not the same as the image, the image will look stretched in one of the directions.

Contents.DrawImage(Image, PosX, PosY, Width, Height);

It should be noted that in addition to PdfImage, PdfQRCode, PdfChart and PrintPdfDocument produce PDF images in the PDF document

### 2.5. Barcode Support

The code below illustrates how to include UPC-A barcode in a PDF document.

// create barcode object

BarcodeEAN13 Barcode = new BarcodeEAN13("123456789010");

// draw the barcode including text under the barcode

Contents.DrawBarcode(PosX, PosY, BarWidth, BarcodeHeight, Barcode, Font, FontSize);

In this case the class is BarcodeEAN13 with 12 digits input string. The result is UPC-A barcode.

The PDF File Writer library includes a base class Barcode. For each supported barcode one needs a derived class. The class library includes four derived classes: Barcode128, Barcode39, BarcodeInterleaved2of5 and BarcodeEAN13. The BarcodeEAN13 produces EAN-13 barcode if the input string is 13 digits and UPC-A if the input string is 12 digits. Input string with 13 digit and a leading zero is considered UPC-A.

The DrawBarcode method has a number of overloads. You specify the position of the bottom left corner of the barcode, the width of the narrow bar, the height of the barcode and the derived barcode class. There are optional arguments: justification (left, center, right) color and font to display the text. Quiet zone around the barcode is your responsibility. Optional text is displayed below the barcode. If you select color other than black you should make sure the contrast to the background is significant. Usage examples are given in [3.7 Draw Barcodes](http://www.codeproject.com/Articles/570682/PDF-File-Writer-Csharp-Class-Library-Version?display=Print#DrawBarcodes), ArticleExample.cs and OtherExample.cs.

If you want to create a derived class for another barcode, use the source code for the three included classes as an example.

### 2.6. Encryption Support

The PDF File Writer library provides support to AES 128 and Standard 128 (RC4) encryption. For more information please refer to PDF Reference sixth edition (Version 1.7) section 3.5 Encryption. The PDF File writer supports two types of encryption filters, the AES-128 and Standard 128. The Standard 128 is RC4 encryption. It is considered unsafe. For new project do not use it. It does not support public key security to encode recipient list.

To encrypt your PDF document call one of four SetEncryption methods defined in PdfDocument class:

Set Encryption with no arguments.

The PDF File Writer library will encrypt the PDF document using AES-128 encryption. The PDF reader will open the document without requesting a password. Permissions flags are set to allow all.

Document.SetEncryption();

Set Encryption with one argument.

The PDF File Writer library will encrypt the PDF document using AES-128 encryption. The argument is permissions. Permission flags are defined below. You can or together more than one permission. The PDF reference manual has full description of permissions. The PDF reader will open the document without requesting a password.

Document.SetEncryption(Permission Permissions);

Set Encryption with two arguments.

The PDF File Writer library will encrypt the PDF document using AES-128 encryption. The two arguments are user password and permissions. The PDF reader will open the document with user password. Permissions will be set as per argument.

Document.SetEncryption(String UserPassword, Permission Permissions);

Set Encryption with four arguments.

The PDF File Writer library will encrypt the PDF document using either EncryptionType.Aes128 encryption or EncryptionType.Standard128 encryption. The four arguments are user password, owner password, permissions and encryption type. If user password is null, the default password will be taken. If owner password in null, the software will generate random number password. The Standard128 encryption is considered unsafe. It should not be used for new projects.

A PDF reader such as Acrobat will accept either user or owner password. If owner password is used to open document, the PDF reader will open it with all permissions set to allow operation.

Document.SetEncryption(String UserPassword, String OwnerPassword, Permission Permissions, EncryptionType Type);

Permission flags are as follows:

// Full description is given in

// PDF Reference Version 1.7 Table 3.20

public enum Permission

{

None = 0,

LowQalityPrint = 4, // bit 3

ModifyContents = 8, // bit 4

ExtractContents = 0x10, // bit 5

Annotation = 0x20, // bit 6

Interactive = 0x100, // bit 9

Accessibility = 0x200, // bit 10

AssembleDoc = 0x400, // bit 11

Print = 0x804, // bit 12 + bit 3

All = 0xf3c, // bits 3, 4, 5, 6, 9, 10, 11, 12

}

### 2.7. Web Link Support

Starting with Version 1.7, the PDF File Writer library provides support to web linking. This feature is one of the PDF interactive features described in the PDF reference manual in Section 8 Interactive Features. It is a combination of annotation and action. Annotation associates a web link with an area on the page. When the user clicks on the area, the PDF reader will activate the default web browser navigating to the desired web page.

The annotation area is a rectangle area defined by absolute coordinates relative to the bottom left corner of the page. To add a web link call AddWebLink method of the PdfPage class.

Page.AddWebLink(Double LeftPos, Double BottomPos, Double RightPos, Double TopPos, String WebLink);

Annotations are not part of the page contents. In order or the reader of your PDF document to know where to click you need to display appropriate text or graphics in the same area on the page. In other words you need to call two methods. The AddWebLink method associated with the page and a second method associated with the contents. The second method can be a graphic object such as image or a rectangle, or text. Because AddWebLink requires coordinates relative to the bottom left corner of the page, the coordinates of your graphic object must be the same. In other words do not use translation, scaling or rotation. If you do, you need to make sure that the two areas will coincide.

The PDF File Writer has several PdfContents methods supporting text annotation.

Draw a line of text with associated web link. The text will be left justified, underlined and blue. Text position is relative to bottom left corner of the page.

// font size in points

PdfContents.DrawWebLink(PdfPage Page, PdfFont Font, Double FontSize,

Double TextAbsPosX, Double TextAbsPosY, String Text, String WebLink);

Draw a line of text with associated web link. Text position is relative to bottom left corner of the page.

// font size in points

PdfContents.DrawWebLink(PdfPage Page, PdfFont Font, Double FontSize, Double TextAbsPosX, Double TextAbsPosY,

TextJustify Justify, DrawStyle DrawStyle, Color TextColor, String Text, String WebLink);

Drawing web link within TextBox is a two step process. First you add the text and the web link string to the box using one of the AddText methods of TextBox class. Second you draw the TextBox to the page contents using one of the DrawText methods of PdfContents.

Add web link to TextBox. The text will be displayed underlined and blue.

TextBox.AddText(PdfFont Font, Double FontSize, String Text, String WebLink);

Add web link to TextBox. The text attributes are defined by DrawStyle and FontColor.

TextBox.AddText(PdfFont Font, Double FontSize, DrawStyle DrawStyle, Color FontColor, String Text, String WebLink);

Second step draw text to contents. This method assumes no extra line or paragraph spacing. Note, if you call DrawText without PdfPage argument on a TextBox with WebLink information, ApplicationException will be thrown.

// note: PosYTop is by reference.

// On exit from the method the PosYTop will have the next Y position

PdfContents.DrawText(Double PosX, ref Double PosYTop, Double PosYBottom, Int32 LineNo, TextBox Box, PdfPage Page);

This method lets you define extra line or paragraph spacing. Note, if you call DrawText without PdfPage argument on a TextBox with WebLink information, ApplicationException will be thrown.

// note: PosYTop is by reference.

// On exit from the method the PosYTop will have the next Y position

PdfContents.DrawText(Double PosX, ref Double PosYTop, Double PosYBottom, Int32 LineNo,

Double LineExtraSpace, Double ParagraphExtraSpace, Boolean FitTextToWidth, TextBox Box, PdfPage Page);

For coding examples please review [3.4 Draw Frame](http://www.codeproject.com/Articles/570682/PDF-File-Writer-Csharp-Class-Library-Version?display=Print#DrawFrame), ArticleExample.cs and OtherExample.cs source code.

### 2.8. QR Code Support

Starting with Version 1.7, the PDF File Writer library provides support to QR Code. The program supports three character sets: numeric, alphanumeric and eight bit byte. The program does not support Kanji characters. The QR Code is an image resource. The library converts the QR Code contents into an image. To display a QR Code in a document one needs to define a resource object and add this object to the contents of a page. The program automatically selects the best character set and the best matrix size. The user has to supply data string and desired error correction level. Error correction levels are: L-Low (7%), M-Medium (15%), Q-Medium-High (25%), H-High (30%). Character sets are defined as: Numeric data (digits 0-9). Alphanumeric (digits 0-9, upper case letters A-Z and nine other characters space $ % \* + - . / : ), 8-bit byte data. The program will scan the data string input and selects the most effective character set. If your data can be divided into segments with just digits or just alphanumeric characters you can create a QR Code object with array of data strings.

Create QR Code object with single data string.

PdfQRCode QRCode = new PdfQRCode(PdfDocument Document, String DataString, ErrorCorrection ErrorCorrection);

Or, create QR Code object with array of data strings. Use this form if you want to divide the data string into sections of different character sets.

PdfQRCode QRCode = new PdfQRCode(PdfDocument Document, String[] DataString, ErrorCorrection ErrorCorrection);

By default the image created will have 4 by 4 pixels per QR Code module and 16 pixels of quiet space around the image. To override the default use:

QRCode.SetModuleSize(Int32 ModuleSize, Int32 QuietZone);

To draw the QR Code call DrawQRCode method of PdfContents. The QRCode is a square. The width and height are the same.

Contents.DrawQRCode(PdfQRCode QRCode, Double OrigX, Double OrigY, Double Width);

For coding examples please review [3.7 Draw Barcodes](http://www.codeproject.com/Articles/570682/PDF-File-Writer-Csharp-Class-Library-Version?display=Print#DrawBarcodes), ArticleExample.cs and OtherExample.cs source code.

### 2.9. Bookmarks Support

Bookmarks are described in the PDF specification (section 8.2.2 Document Outline) as follows: "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 serve as a visual table of contents to display the document's structure to the user. The user can interactively open and close individual item by clicking them with the mouse."

The OtherExample.cs source code has an example of bookmarks. At one location there is a hierarchy of three levels. You can see the result in OtherExample.pdf file.

The first step in adding bookmarks to your application is:

// set the program to display bookmarks

// and get the bookmark root object

PdfBookmark BookmarkRoot = Document.GetBookmarksRoot();

This step activates bookmarks in your document and returns the root node.

Adding bookmarks is similar to adding controls to a windows form. The first level bookmarks are added to the root. Subsequent levels are added to existing bookmarks. At minimum you have to define a title, page, vertical position on the page and an open entries flag. Page is the PdfPage object of the page to go to. YPos is the vertical position relative to the bottom left corner of the page. Open entries flag is true if the lower level bookmarks are visible and false if the lower level are hidden. The first level is always visible by default.

// hierarchy example

PdfBookmark FirstLevel\_1 = BookmarkRoot.AddBookmark("Chapter 1", Page, YPos, false);

PdfBookmark SecondLevel\_11 = FirstLevel\_1.AddBookmark("Section 1.1", Page, YPos, false);

PdfBookmark SecondLevel\_12 = FirstLevel\_1.AddBookmark("Section 1.2", Page, YPos, false);

PdfBookmark ThirdLevel\_121 = SecondLevel\_12.AddBookmark("Section 1.2.1", Page, YPos, false);

PdfBookmark ThirdLevel\_122 = SecondLevel\_12.AddBookmark("Section 1.2.2", Page, YPos, false);

PdfBookmark SecondLevel\_13 = FirstLevel\_1.AddBookmark("Section 1.3", Page, YPos, false);

PdfBookmark FirstLevel\_2 = BookmarkRoot.AddBookmark("Chapter 2", Page, YPos, false);

PdfBookmark SecondLevel\_21 = FirstLevel\_2.AddBookmark("Section 2.1", Page, YPos, false);

PdfBookmark SecondLevel\_22 = FirstLevel\_2.AddBookmark("Section 2.2", Page, YPos, false);

AddBookmark() method has four overloading variations:

// basic

public PdfBookmark AddBookmark

(

String Title, // bookmark title

PdfPage Page, // bookmark page

Double YPos, // bookmark vertical position relative to bottom left corner of the page

Boolean OpenEntries // true is display children. false hide children

)

// title color and style

public PdfBookmark AddBookmark

(

String Title, // bookmark title

PdfPage Page, // bookmark page

Double YPos, // bookmark vertical position relative to bottom left corner of the page

Color Paint, // bookmark color. Coloe.Empty is display title in default color

TextStyle TextStyle, // bookmark text style: normal, bold, italic, bold-italic

Boolean OpenEntries // true is display children. false hide children

)

// XPos and zoom

public PdfBookmark AddBookmark

(

String Title, // bookmark title

PdfPage Page, // bookmark page

Double XPos, // bookmark horizontal position relative to bottom left corner of the page

Double YPos, // bookmark vertical position relative to bottom left corner of the page

Double Zoom, // Zoom factor. 1.0 is 100%. 0.0 is ignore zoom.

Boolean OpenEntries // true is display children. false hide children

)

// all options

public PdfBookmark AddBookmark

(

String Title, // bookmark title

PdfPage Page, // bookmark page

Double XPos, // bookmark horizontal position relative to bottom left corner of the page

Double YPos, // bookmark vertical position relative to bottom left corner of the page

Double Zoom, // Zoom factor. 1.0 is 100%. 0.0 is ignore zoom.

Color Paint, // bookmark color. Coloe.Empty is display title in default color

TextStyle TextStyle, // bookmark text style: normal, bold, italic, bold-italic

Boolean OpenEntries // true is display children. false hide children

)

PdfBookmark class exposes one more method GetChild. You can get any bookmark by calling GetChild with one or more integer arguments. Each argument is a zero base argument of the child position in the level. For example GetChild(2) is the third item of the first level. GetChild(2, 3) is the forth second level item of the third first level item.

### 2.10. Charting Support

The PDF specification has no specific support for charting. The PDF File Writer library provides charting support by allowing the developer to create a Microsoft Charting object and draw this object as an image into the PDF file. For more information about Microsoft Chart control note MSDN library documentation [Visual Studio 2012 Chart Controls](http://msdn.microsoft.com/en-us/library/dd456726(v=vs.110).aspx). The documentation for the charting name space is available at [Data Visualization Charting Namespace](http://msdn.microsoft.com/en-us/library/system.windows.forms.datavisualization.charting(v=vs.110).aspx). The attached ChartExample.cs has four examples of charts. If you intend to use charting, you need to add System.Windows.Forms.Visualization reference to your project. In each source module using Chart you need to add using System.Windows.Forms.DataVisualization.Charting;.

Adding a chart to a PDF document is four steps process.

* Create Chart object.
* Create PdfChart object.
* Build the chart.
* Draw the PdfChart to PdfContents.

The recommended way to create a chart is to use a static method of PdfChart object.

// Document is your PdfDocument object.

// Width and height are in user units.

// Resolution is in pixels per inch.

// Resolution is optional. If not included the library will take the .net default.

// Library will create Chart object with width and height in pixels and set resolution in pixels per inch

Chart MyChart = PdfChart.CreateChart(PdfDocument Document, Double Width, Double Height, Double Resolution);

You can instantiate Chart class yourself.

Chart MyChart = new Chart();

MyChart.RenderingDpiY = 300; // example of 300 pixels per inch

MyChart.Width = 1950; // example of 6.5 inches in pixels

Mychart.Height = 1350; // example of 4.6 inches in pixels

Next you create a PdfChart from the Chart created above. Optionally, you can override the resolution.

// resolution is optional. It will override the resolution set above.

PdfChart MyPdfChart = new PdfChart(PdfDocument Document, Chart MyChart, Double Resolution);

Next you build your chart. ChartExample.cs has four examples. The documentation for building a chart is beyond the scope of this article. There plenty of examples on the Internet.

PdfChart has a CreateFont method to simplify the creation of fonts. It will calculate font size based on chart's resolution.

// FontSizeUnit is an enumeration

// Available units: pixel, point, UserUnit, Inch, cm, mm

Font CreateFont(String FontFamilyName, FontStyle Style, Double FontSize, FontSizeUnit Unit);

The last step is drawing the chart.

// Draw the chart at OrigX, OrigY in user units

// The width and height of the chart are taken from the Chart object.

// They are calculated from the size in pixels and resolution of the chart.

public void PdfContents.DrawChart(PdfChart MyPdfChart, Double OrigX, Double OrigY);

// Draw the chart at OrigX, OrigY with Width and Height as specified, all in user units.

// NOTE: Width and Height should be selected to agree with the aspect ratio of the chart object.

public void PdfContents.DrawChart(PdfChart MyPdfChart, Double OrigX, Double OrigY, Double Width, Double Height);

The PdfChart class provides some optional methods to control image positioning.

The ImageSize method returns the largest rectangle with correct aspect ratio that will fit in a given area.

SizeD ImageSize(Double Width, Double Height);

The ImageSizePosition method returns the largest rectangle with correct aspect ratio that will fit in a given area and position it based on ContentAlignment enumeration.

ImageSizePos ImageSizePosition(Double Width, Double Height, ContentAlignment Alignment);

### 2.11. Print Document Support

Print document support allows you to print a report in the same way as printing to a printer and producing a PDF document. The difference between this method of producing a PDF file and using PdfContents to produce a PDF file is the difference between raster graphics to vector graphics. Print document support creates one jpeg image per page. PrintExample.cs has an example of creating a three page document.

Normally each page is a full image of the page. If your page is letter size and the resolution is 300 pixels per inch, each pixel is 3 bytes, the bit map of the page will be 25.245MB long. PrintPdfDocument has a method CropRect that can reduce the size of the bit map significantly. Assuming a one inch margin is used, the active size of the bit map will be reduced to 15.795 MB. That is 37.4% reduction.

// main program

// Create empty document

Document = new PdfDocument(PaperType.Letter, false, UnitOfMeasure.Inch);

// create PrintPdfDocument producing an image with 300 pixels per inch

PrintPdfDocument Print = new PrintPdfDocument(Document, 300.0);

// PrintPage in the delegate method PrintPageEventHandler

// This method will print one page at a time to PrintDocument

Print.PrintPage += PrintPage;

// set margins in user units (Left, top, right, bottom)

// note the margins order are per .net standard and not PDF standard

Print.SetMargins(1.0, 1.0, 1.0, 1.0);

// crop the page image result to reduce PDF file size

// the crop rectangle is per .net standard.

// The origin is top left.

Print.CropRect = new RectangleF(0.95F, 0.95F, 6.6F, 9.1F);

// initiate the printing process (calling the PrintPage method)

// after the document is printed, add each page as an image to PDF file.

Print.AddPagesToPdfDocument();

// dispose of the PrintDocument object

Print.Dispose();

// create the PDF file

Document.CreateFile(FileName);

Example of PrintPage method

// Print each page of the document to PrintDocument class

// You can use standard PrintDocument.PrintPage(...) method.

// NOTE: The graphics origin is top left and Y axis is pointing down.

// In other words this is not PdfContents printing.

public void PrintPage(object sender, PrintPageEventArgs e)

{

// graphics object short cut

Graphics G = e.Graphics;

// Set everything to high quality

G.SmoothingMode = SmoothingMode.HighQuality;

G.InterpolationMode = InterpolationMode.HighQualityBicubic;

G.PixelOffsetMode = PixelOffsetMode.HighQuality;

G.CompositingQuality = CompositingQuality.HighQuality;

// print area within margins

Rectangle PrintArea = e.MarginBounds;

// draw rectangle around print area

G.DrawRectangle(Pens.DarkBlue, PrintArea);

// line height

Int32 LineHeight = DefaultFont.Height + 8;

Rectangle TextRect = new Rectangle(PrintArea.X + 4, PrintArea.Y + 4, PrintArea.Width - 8, LineHeight);

// display page bounds

// DefaultFont is defined somewhere else

String text = String.Format("Page Bounds: Left {0}, Top {1}, Right {2}, Bottom {3}",

e.PageBounds.Left, e.PageBounds.Top, e.PageBounds.Right, e.PageBounds.Bottom);

G.DrawString(text, DefaultFont, Brushes.Black, TextRect);

TextRect.Y += LineHeight;

// display print area

text = String.Format("Page Margins: Left {0}, Top {1}, Right {2}, Bottom {3}",

PrintArea.Left, PrintArea.Top, PrintArea.Right, PrintArea.Bottom);

G.DrawString(text, DefaultFont, Brushes.Black, TextRect);

TextRect.Y += LineHeight;

// print some lines

for(Int32 LineNo = 1; ; LineNo++)

{

text = String.Format("Page {0}, Line {1}", PageNo, LineNo);

G.DrawString(text, DefaultFont, Brushes.Black, TextRect);

TextRect.Y += LineHeight;

if(TextRect.Bottom > PrintArea.Bottom) break;

}

// move on to next page

PageNo++;

e.HasMorePages = PageNo <= 3;

return;

}

### 2.12. Data Table Support

The data table classes allow you to display data tables in your PDF document. PdfTable is the main class controlling the display of one table. A table is made out of a header row and data rows. Each row is divided into cells. PdfTableCell controls the display of one header cell or one data cell. If header is used it will be displayed at the top of the table. Optionally it will be displayed at the top of each additional page. To display data in a cell, you load the data into the Value property of PdfTableCell. Data can be: text string, basic numeric value, Boolean, Char, TextBox, image, QR code or barcode. Independently of data, you can load the cell with web link. Clicking anywhere within the cell's area will cause the PDF reader to activate web explorer and call the requested page. The display of the data is controlled by PdfTableStyle class. PdfTable class contains a default cell style and a default header style. You can override the default styles with private styles within PdfTableCell. To display a table you create a PdfTable object. Next you initialize the table, header cells, data cells and styles objects. Finally you set a loop and load the cell values of one row and then draw this row. This loop continues until all data was displayed. Below you will find the necessary sequence of steps to produce a table.

When DrawRow method is called, the software calculates the required row height. Row height is the height of the tallest cell. The row will be drawn if there is sufficient space within the table. When the available space at the bottom is too small, a new page is called and optional heading and the current row are displayed at the top of the table. If the required row height is so large that it will not fit in full empty table, an exception is raised. In order to accommodate long multi-line Strings or TextBoxes, the software can handle these cases in a flexible way. Multi-line String is converted by PdfTable into a TextBox. The PdfTableStyle class has a TextBoxPageBreakLines property. If this property is set to zero (default), the TextBox is treated as other data values. TextBox height must fit the page. If TextBoxPageBreakLines is set to a positive integer, the system will calculate cell's height as TextBox height or the height the first few lines as specified by TextBoxPageBreakLines. The system will draw the row with as many lines that fit the page. A new page will be created and the rest of the lines will be drawn. In other words, the first block of lines of a long TextBox will be at least TextBoxPageBreakLines long. TableExample.cs source contains an example of long TextBox cells.

Create a PdfTable object.

// create table

PdfTable Table = new PdfTable(Page, Contents, Font, FontSize);

Page is the current PdfPage. Contents is the current PdfContents. Font is the table default font. FontSize is the default font size in points.

Define table's area on the page.

// table's area on the page

Table.TableArea = new PdfRectangle(Left, Bottom, Right, Top);

// first page starting vertical position

Table.RowTopPosition = StartingTopPosition;

The four arguments are the four sides of the table relative to bottom left corner and in user units. If on the first page the table top position is not at the top of the page set RowTopPosition to the starting top position. On subsequent pages the table will always start at the top. If TableArea is not specified, the library will set it to default page size less 1 inch margin.

Divide the table width into columns.

// divide table area width into columns

StockTable.SetColumnWidth(Width1, Width2, Width3, ...);

The number of arguments is the number of columns. The table width less total border lines will be divided in proportion to these arguments.

Once the number of columns is set with SetColumnWidth method the library creates two PdfTableCell arrays. One array for header cells and one array for data cells.

Rows and columns of the data table can be separated by border lines. Border lines properties are defined by PdfTableBorder and PdfTableBorderStyle. There are four horizontal border lines: TopBorder, BottomBorder, HeaderHorBorder between the header row and first data row and CellHorBorder between data rows. There are two sets of vertical border lines: HeaderVertBorder array for vertical border lines within the header row, and CellVertBorder array for vertical border lines between columns within the data part of the table. Arrays size is the number of columns plus one. Array element zero is the table's left border. Array element Columns is the table's right border. All other elements are lines separating columns. Each of these lines can be defined individually. There are methods to define all border lines at once or define each individual border line.

Methods to define all border lines:

// clear all border lines

Table.Borders.ClearAllBorders();

// set all border lines to default values (no need to call)

// All frame lines are one point (1/72") wide

// All grid lines are 0.2 of one point wide

// All borders are black

Table.Borders.SetDefaultBorders();

// set all borders to same width and black color

Table.Borders.SetAllBorders(Double Width);

// set all borders to same width and a specified color

Table.Borders.SetAllBorders(Double Width, Color LineColor);

// set all borders to one width and all grid lines to another width all lines are black

Table.Borders.SetAllBorders(Double FrameWidth, Double GridWidth);

// set all borders to one width and color and all grid lines to another width and color

Table.Borders.SetAllBorders(Double FrameWidth, Color FrameColor, Double GridWidth, Color GridColor);

// set all frame borders to same width and black color and clear all grid lines

Table.Borders.SetFrame(Double Width);

// set all frame borders to same width and a specified color and clear all grid lines

Table.Borders.SetFrame(Double Width, Color LineColor);

Each horizontal border line can be cleared or set. The example is for top border line:

// clear border

Table.Borders.ClearTopBorder();

// set border with default color set to black

// Zero width means one pixel of the output device.

Table.Borders.SetTopBorder(Double LineWidth);

// set border

Table.Borders.SetTopBorder(Double LineWidth, Color LineColor);

Each vertical border line can be cleared or set. The example is for cell's vertical border lines:

// clear border

Table.Borders.ClearCellVertBorder(Int32 Index);

// set border with default color set to black

Table.Borders.SetCellVertBorder(Int32 Index, Double LineWidth);

// set border

Table.Borders.SetCellVertBorder(Int32 Index, Double LineWidth, Color LineColor);

Set other optional table properties. The values given in the example below are the defaults.

// header on each page

HeaderOnEachPage = true;

// minimum row height

MinRowHeight = 0.0;

Table information is processed one row at a time. Each row is made of cells. One cell per column. The display of cell's information is controlled by PdfTableStyle class. There are about 20 style properties. For the complete list view the source code or the help file. Some of these styles are specific to the type of information to be displayed. Here is an example

// make some changes to default header style

Table.DefaultHeaderStyle.Alignment = ContentAlignment.BottomRight;

// create private style for header first column

Table.Header[0].Style = Table.HeaderStyle;

Table.Header[0].Style.Alignment = ContentAlignment.MiddleLeft;

// load header value

Table.Header[0].Value = "Date";

// make some changes to default cell style

Table.DefaultCellStyle.Alignment = ContentAlignment.MiddleRight;

Table.DefaultCellStyle.Format = "#,##0.00";

// create private style for date column

Table.Cell[0].Style = StockTable.CellStyle;

Table.Cell[0].Style.Alignment = ContentAlignment.MiddleLeft;

Table.Cell[0].Style.Format = null;

After initialization is done it is time to display the data. The example below is from TableExample.cs. It is a table of stock prices. There are 6 columns.

// open stock daily price

StreamReader Reader = new StreamReader("SP500.csv");

// ignore header

Reader.ReadLine();

// read all daily prices

for(;;)

{

String TextLine = Reader.ReadLine();

if(TextLine == null) break;

String[] Fld = TextLine.Split(new Char[] {','});

Table.Cell[ColDate].Value = Fld[ColDate];

Table.Cell[ColOpen].Value = Double.Parse(Fld[ColOpen], NFI.PeriodDecSep);

Table.Cell[ColHigh].Value = Double.Parse(Fld[ColHigh], NFI.PeriodDecSep);

Table.Cell[ColLow].Value = Double.Parse(Fld[ColLow], NFI.PeriodDecSep);

Table.Cell[ColClose].Value = Double.Parse(Fld[ColClose], NFI.PeriodDecSep);

Table.Cell[ColVolume].Value = Int32.Parse(Fld[ColVolume]);

StockTable.DrawRow();

}

StockTable.Close();

The DrawRow(NewPage) method has an optional argument Boolean NewPage = false. The default value is false. If you want the next row to be printed at the top of the next page, set the argument to true.

Web link Example.

// set cell number 6 with QR code and web link

BookList.Cell[6].Value = new PdfQRCode(Document, WeblinkString, ErrorCorrection.M);

BookList.Cell[6].WebLink = WebLinkString;

For more examples of data table source code look into ArticleExample.cs and TableExample.cs. For more detail documentation of PdfTable, PdfTableCell, PdfTableStyle and PdfTableBorder classes look into PdfFileWriter.chm help file.

### 2.13. Play Video Files

The PdfFileWriter supports embedding video files in the PDF document. Full examples of playing video files are given in the page 7 of OtherExample.cs. Adding a video file requires the use of three classes. First you need to embed the video file in the PDF document.

// save the video in embedded media file

PdfEmbeddedFile VideoFile = new PdfEmbeddedFile(Document, "LooneyTunes.mp4");

Second you need to define how the video is to be played. The PdfDisplayMedia class has a number of methods to control the video display. Please refer to the class' source code and the documentation help file. For example: RepeatCount or ScaleMedia. If you want to play the video in a floating window you must use SetMediaWindow method.

// create display media object

PdfDisplayMedia DisplayMedia = new PdfDisplayMedia(VideoFile);

Third you need to define the area on the PDF page that the user must click in order to activate the video. If you want to activate the video when the annotation area is visible, use ActivateActionWhenPageIsVisible.

// create annotation object

// AnnotRect is the rectangular area on the page that the user will click

// to activate the video and the video will be displayed in

PdfAnnotation Annotation = new PdfAnnotation(Page, AnnotRect, DisplayMedia);

// define X Object to paint the annotation area when the video is not playing

// in this example we draw one line of text. You can draw any graphics, text or display image

// just like you draw to the contents of a page

PdfXObject XObject = new PdfXObject(Document, AnnotRect.Width, AnnotRect.Height);

XObject.DrawText(ArialNormal, 14.0, 0.5 \* AnnotRect.Width, 0.5 \* AnnotRect.Height, TextJustify.Center, "Click here to play the video");

// attach the XObject to the annotation object

Annotation.Appearance(XObject);

### 2.14. Play Sound Files

The PdfFileWriter supports embedding sound files in the PDF document. Full example of playing sound file is given in the page 7 of OtherExample.cs. Embedding sound files is essentially the same as video files. The only obvious difference is that there is nothing to display.

// save sound in embedded media file

PdfEmbeddedFile EmbeddedMediaFile = new PdfEmbeddedFile(Document, "Ring01.wav");

// create play sound display media object

// for sound media set MediaWindow to hidden

PdfDisplayMedia DisplayMedia = new PdfDisplayMedia(EmbeddedMediaFile);

DisplayMedia.SetMediaWindow(MediaWindow.Hidden);

// display text area to activate the sound

// this text area is part of current contents

Double TextWidth = Contents.DrawText(ArialNormal, 12.0, TextPosX, TextPosY, TextJustify.Center, "Click this text to play sound");

// define a rectangular area around the text to click on in order to play the sound

PdfRectangle AnnotRect = new PdfRectangle(TextPosX - 0.5 \* TextWidth, TextPosY - ArialNormal.DescentPlusLeading(12.0),

TextPosX + 0.5 \* TextWidth, TextPosY + ArialNormal.AscentPlusLeading(12.0));

// create annotation object

PdfAnnotation Annotation = new PdfAnnotation(Page, AnnotRect, DisplayMedia);

### 2.15. Attach Data Files

The PdfFileWriter supports embedding data files in the PDF document. Full example of embedding file is given in the page 7 of OtherExample.cs. The user can save the files or display the files.

// save attachment file in embedded media file

EmbeddedFile = new PdfEmbeddedFile(Document, "BookList.txt");

// annotation

PdfRectangle AnnotRect = new PdfRectangle(IconPosX, IconPosY, IconPosX + IconWidth, IconPosY + IconHeight);

PdfAnnotation Annotation = new PdfAnnotation(Page, AnnotRect, EmbeddedFile, FileAttachIcon.Paperclip);

### 2.16. Reorder Pages

The PdfFileWriter appends new pages to the end of the page list. If you want to move a page from its current position to a new position use the following method.

// Source and destination index are zero based.

// Source must be 0 to PageCount - 1.

// Destination must be 0 to PageCount.

// If destination index is PageCount, the page will be the last page

// PageCount is a property of PdfDocument.

Document.MovePage(Int32 SourceIndex, Int32 DestinationIndex);

### 2.17. Document Destination

The PdfFileWriter creates PDF documents. The main class PdfDocument constructor gives you two choices to save the document. The first choice is to save the PDF file to a disk file. In this case you provide the constuctor with a file name. At the end of file creation you call PdfDocument.CreateFile. This method writes the PDF to the file and closes the file.

// create main class

PdfDocument Document = new PdfDocument(PaperType.Letter, false, UnitOfMeasure.Inch, FileName);

// terminate

Document.CreateFile();

The second choice is a stream. You create a stream, either memory stream or a file stream, and you pass the stream as an argument to the PdfDocument constructor. After CreateFile method is executed, your stream contains the PDF document. Extract the document from the stream as appropriate to your application. You must close the stream in your application.

// create memory stream

MemoryStream PdfStream = new MemoryStream();

// create main class

PdfDocument Document = new PdfDocument(PaperType.Letter, false, UnitOfMeasure.Inch, PdfStream);

// terminate

Document.CreateFile();

// save the memory stream to a file

FileStream FS = new FileStream(FileName, FileMode.Create);

PdfStream.WriteTo(FS);

PdfStream.Close();

FS.Close();

### 2.18. Document Information Dictionary

The PDF document information dictionary is displayed by the PDF reader in the Description tab of the document properties. The information includes: Title, Author, Subject, Keywords, Created date and time, Modified date and time, the Application that produced the file, the PDF Producer. Including document information in your application is done as follows:

PdfInfo Info = new PdfInfo(Document);

Info.Title("Article Example");

Info.Author("Uzi Granot Granotech Limited");

Info.Keywords("PDF, .NET, C#, Library, Document Creator");

Info.Subject("PDF File Writer C# Class Library (Version 1.15.0)");

When PdfInfo object is created, four additional fields are added to the dictionary. You can override all of them in your code.

// set creation and modify dates

DateTime LocalTime = DateTime.Now;

Info.CreationDate(LocalTime);

Info.ModDate(LocalTime);

// set creator and producer

Info.Creator("PdfFileWriter C# Class Library Version " + PdfDocument.RevisionNumber);

Info.Producer("PdfFileWriter C# Class Library Version " + PdfDocument.RevisionNumber);

### 2.19. Installation

Integrating PdfFileWriter to your application requires the following steps. Install the attached PdfFileWriter.dll file in your development area. Start the Visual C# program and open your application. Go to the Solution Explorer, right click on References and select Add Reference. Select the Browse tab and navigate your file system to the location of the PdfFileWriter.dll. When your application is published, the code>PdfFileWriter.dll must be included.

Source code documentation is available in as a help file PdfFileWriter.chm. The file is produced by Sandcastle. The result looks like Microsoft documentation pages.

If you want access to the source code of the PdfFileWriter project, install the PdfFileWriter project in your development area. The PdfFileWriter.dll will be in PdfFileWriter\bin\Release directory.

Add the following statement to all source modules using this library.

using PdfFileWriter;

If you intend to use charting, you need to add reference to: System.Windows.Forms.Visualization. In each source module using Chart you need to add

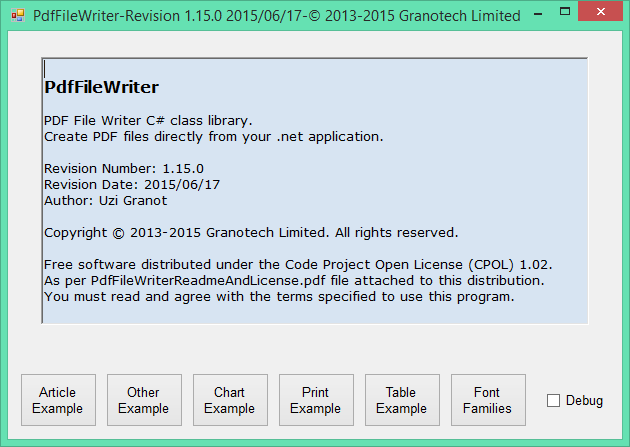
using System.Windows.Forms.DataVisualization.Charting;

3. Development Guide by Example

This section describes the integration of the PDF File Writer C# class library to your application. The test program TestPdfFileWriter program is a simulation of your own application. When you press on the “Article Example” button, the program executes the code in ArticleExample.cs source file. The image above displays the resulted PDF file. This method demonstrates the creation of one page document with some text and graphics. After going through this example, you should have a good understanding of the process. The other example buttons produce a variety of PDF documents. In total, practically every feature of this library is demonstrated by these examples.

The Debug check box, if checked, will create a PDF file that can be viewed with a text editor but cannot be loaded to a PDF reader. The resulted file is not compressed and images and font file are replaced with text place holder. The Debug check box should be used for debugging only.

The TestPdfFileWriter program was developed using Microsoft Visual C# 2012. It was tested for Windows XP, Vista, 7 and 8.



### \*-3.1. Document Creation Overview

The Test method below demonstrates the six steps described in the introduction for creating a PDF file. The method will be executed when you press on the “Article Example” button of the demo program. The following subsections describe in detail each step.

// Create &ldquo;article example&rdquo; test PDF document

public void Test(Boolean Debug, String FileName)

{

// Step 1: Create empty document

// Arguments: page width: 8.5&rdquo;, page height: 11&rdquo;, Unit of measure: inches

// Return value: PdfDocument main class

Document = new PdfDocument(PaperType.Letter, false, UnitOfMeasure.Inch, FileName);

// for encryption test

// Document.SetEncryption(Permission.All & ~Permission.Print);

// Debug property

// By default it is set to false. Use it for debugging only.

// If this flag is set, PDF objects will not be compressed, font and images will be replaced

// by text place holder. You can view the file with a text editor but you cannot open it with PDF reader.

Document.Debug = Debug;

// Step 2: create resources

// define font resources

DefineFontResources();

// define tiling pattern resources

DefineTilingPatternResource();

// Step 3: Add new page

Page = new PdfPage(Document);

// Step 4:Add contents to page

Contents = new PdfContents(Page);

// Step 5: add graphics and text contents to the contents object

DrawFrameAndBackgroundWaterMark();

DrawTwoLinesOfHeading();

DrawHappyFace();

DrawBarcode();

DrawBrickPattern();

DrawImage();

DrawHeart();

DrawChart();

DrawTextBox();

DrawBookOrderForm();

// Step 6: create pdf file

// argument: PDF file name

Document.CreateFile();

// start default PDF reader and display the file

Process Proc = new Process();

Proc.StartInfo = new ProcessStartInfo(FileName);

Proc.Start();

// exit

return;

}

### 3.2. Font Resources

The DefineFontResources method creates all the font resources used in this example. In addition, you will find an example of one character substitution. Arial character 9679 (decimal) is a full circle glyph is being mapped into character 164 (decimal). The full circle glyph is used as a separator within the text at the bottom left of the PDF example document. To see all the characters available for any font, press the third button “Font Families”. Select a family and view the glyphs defined for each character. To view individual glyph press view or double click.

// Define Font Resources

private void DefineFontResources()

{

// Define font resources

// Arguments: PdfDocument class, font family name, font style, embed flag

// Font style (must be: Regular, Bold, Italic or Bold | Italic) All other styles are invalid.

// Embed font. If true, the font file will be embedded in the PDF file.

// If false, the font will not be embedded

String FontName1 = "Arial";

String FontName2 = "Times New Roman";

ArialNormal = new PdfFont(Document, FontName1, FontStyle.Regular, true);

ArialBold = new PdfFont(Document, FontName1, FontStyle.Bold, true);

ArialItalic = new PdfFont(Document, FontName1, FontStyle.Italic, true);

ArialBoldItalic = new PdfFont(Document, FontName1, FontStyle.Bold | FontStyle.Italic, true);

TimesNormal = new PdfFont(Document, FontName2, FontStyle.Regular, true);

Comic = new PdfFont(Document, "Comic Sans MS", FontStyle.Bold, true);

// substitute one character for another

// this program supports characters 32 to 126 and 160 to 255

// if a font has a character outside these ranges that is required by the application,

// you can replace an unused character with this character

// Note: space (32) and non breaking space (160) cannot be replaced

ArialNormal.CharSubstitution(9679, 9679, 161); // euro

ArialNormal.CharSubstitution(1488, 1514, 162); // hebrew

ArialNormal.CharSubstitution(1040, 1045, 189); // russian

ArialNormal.CharSubstitution(945, 950, 195); // greek

return;

}

### 3.3. Tiling Pattern Resource

The DefineTilingPatternResource method defines background pattern resource for the example area. The pattern is the word “PdfFileWriter” in white over light blue background. The pattern is made of two lines of repeating the key word. The two lines are skewed by half word length.

If you want to find interesting patterns, search the internet for catalogs of companies making floor or wall tiles.

// Define Tiling Pattern Resource

private void DefineTilingPatternResource()

{

// create empty tiling pattern

WaterMark = new PdfTilingPattern(Document);

// the pattern will be PdfFileWriter layed out in brick pattern

String Mark = "PdfFileWriter";

// text width and height for Arial bold size 18 points

Double FontSize = 18.0;

Double TextWidth = ArialBold.TextWidth(FontSize, Mark);

Double TextHeight = ArialBold.LineSpacing(FontSize);

// text base line

Double BaseLine = ArialBold.DescentPlusLeading(FontSize);

// the overall pattern box (we add text height value as left and right text margin)

Double BoxWidth = TextWidth + 2 \* TextHeight;

Double BoxHeight = 4 \* TextHeight;

WaterMark.SetTileBox(BoxWidth, BoxHeight);

// save graphics state

WaterMark.SaveGraphicsState();

// fill the pattern box with background light blue color

WaterMark.SetColorNonStroking(Color.FromArgb(230, 244, 255));

WaterMark.DrawRectangle(0, 0, BoxWidth, BoxHeight, PaintOp.Fill);

// set fill color for water mark text to white

WaterMark.SetColorNonStroking(Color.White);

// draw PdfFileWriter at the bottom center of the box

WaterMark.DrawText(ArialBold, FontSize, BoxWidth / 2, BaseLine, TextJustify.Center, Mark);

// adjust base line upward by half height

BaseLine += BoxHeight / 2;

// draw the right half of PdfFileWriter shifted left by half width

WaterMark.DrawText(ArialBold, FontSize, 0.0, BaseLine, TextJustify.Center, Mark);

// draw the left half of PdfFileWriter shifted right by half width

WaterMark.DrawText(ArialBold, FontSize, BoxWidth, BaseLine, TextJustify.Center, Mark);

// restore graphics state

WaterMark.RestoreGraphicsState();

return;

}

### 3.4. Draw Frame with Background Pattern

The DrawFrameAndBackgroundWaterMark method draws a frame around the example area with background water mark pattern. The pattern resource was define in the previous subsection.

// Draw frame around example area

private void DrawFrameAndBackgroundWaterMark()

{

// save graphics state

Contents.SaveGraphicsState();

// Draw frame around the page

// Set line width to 0.02"

Contents.SetLineWidth(0.02);

// set frame color dark blue

Contents.SetColorStroking(Color.DarkBlue);

// use water mark tiling pattern to fill the frame

Contents.SetPatternNonStroking(WaterMark);

// rectangle position: x=1.0", y=1.0", width=6.5", height=9.0"

Contents.DrawRectangle(1.0, 1.0, 6.5, 9.0, PaintOp.CloseFillStroke);

// restore graphics sate

Contents.RestoreGraphicsState();

// draw article name under the frame

// Note: the \u00a4 is character 164 that was substituted during Font resource definition

// this character is a solid circle it is normally unicode 9679 or \u25cf in the Arial family

Contents.DrawText(ArialNormal, 9.0, 1.1, 0.85, "PdfFileWriter \u25cf PDF File Writer C# Class Library \u25cf Author: Uzi Granot");

// draw web link to the article

Contents.DrawWebLink(Page, ArialNormal, 9.0, 7.4, 0.85, TextJustify.Right, DrawStyle.Underline, Color.Blue, "Click to view article",

"http://www.codeproject.com/Articles/570682/PDF-File-Writer-Csharp-Class-Library-Version");

return;

}

### 3.5. Draw Two Lines of Heading

The DrawTwoLinesOfHeading method draws two heading lines at the center of the page. The first line is drawing text with outline special effect.

// Draw heading

private void DrawTwoLinesOfHeading()

{

// page heading

// Arguments: Font: ArialBold, size: 36 points, Position: X = 4.25", Y = 9.5"

// Text Justify: Center (text center will be at X position)

// Stoking color: R=128, G=0, B=255 (text outline)

// Nonstroking color: R=255, G=0, B=128 (text body)

Contents.DrawText(Comic, 40.0, 4.25, 9.25, TextJustify.Center, 0.02, Color.FromArgb(128, 0, 255), Color.FromArgb(255, 0, 128), "PDF FILE WRITER");

// save graphics state

Contents.SaveGraphicsState();

// change nonstroking (fill) color to purple

Contents.SetColorNonStroking(Color.Purple);

// Draw second line of heading text

// arguments: Handwriting font, Font size 30 point, Position X=4.25", Y=9.0"

// Text Justify: Center (text center will be at X position)

Contents.DrawText(Comic, 30.0, 4.25, 8.75, TextJustify.Center, "Example");

// restore graphics sate (non stroking color will be restored to default)

Contents.RestoreGraphicsState();

return;

}

### 3.6. Draw Happy Face

The DrawHappyFace method is an example of drawing oval and constructing path from a line and Bezier curve.

// Draw Happy Face

private void DrawHappyFace()

{

// save graphics state

Contents.SaveGraphicsState();

// translate coordinate origin to the center of the happy face

Contents.Translate(4.25, 7.5);

// change nonstroking (fill) color to yellow

Contents.SetColorNonStroking(Color.Yellow);

// draw happy face yellow oval

Contents.DrawOval(-1.5, -1.0, 3.0, 2.0, PaintOp.Fill);

// set line width to 0.2" this is the black circle around the eye

Contents.SetLineWidth(0.2);

// eye color is white with black outline circle

Contents.SetColorNonStroking(Color.White);

Contents.SetColorStroking(Color.Black);

// draw eyes

Contents.DrawOval(-0.75, 0.0, 0.5, 0.5, PaintOp.CloseFillStroke);

Contents.DrawOval(0.25, 0.0, 0.5, 0.5, PaintOp.CloseFillStroke);

// mouth color is black

Contents.SetColorNonStroking(Color.Black);

// draw mouth by creating a path made of one line and one Bezier curve

Contents.MoveTo(-0.6, -0.4);

Contents.LineTo(0.6, -0.4);

Contents.DrawBezier(0.0, -0.8, 0, -0.8, -0.6, -0.4);

// fill the path with black color

Contents.SetPaintOp(PaintOp.Fill);

// restore graphics sate

Contents.RestoreGraphicsState();

return;

}

### 3.7. Draw Barcodes

The DrawBarcode method is an example of drawing two barcodes EAN-13 and Code-128

// Draw Barcode

private void DrawBarcode()

{

// save graphics state

Contents.SaveGraphicsState();

BarcodeEAN13 Barcode1 = new BarcodeEAN13("1234567890128");

Contents.DrawBarcode(1.3, 7.05, 0.012, 0.75, Barcode1, ArialNormal, 8.0);

PdfQRCode QRCode = new PdfQRCode(Document, "http://www.codeproject.com/Articles/570682/PDF-File-Writer-Csharp-Class-Library-Version", ErrorCorrection.M);

Contents.DrawQRCode(QRCode, 6.0, 6.8, 1.2);

// define a web link area coinciding with the qr code

Page.AddWebLink(6.0, 6.8, 7.2, 8.0, "http://www.codeproject.com/Articles/570682/PDF-File-Writer-Csharp-Class-Library-Version");

// restore graphics sate

Contents.RestoreGraphicsState();

return;

}

### 3.8. Draw Rectangle with Rounded Corners and Filled with Brick Pattern

The DrawBrickPattern method draws a rectangle with rounded corners. The area is filled with a brick pattern. The PdfTillingPattern class is a general class to define tiling patterns. The class has two static methods to create specific patterns. SetBrickPattern to draw a brick wall pattern and SetWeavePattern to draw a weaving pattern.

// Draw rectangle with rounded corners and filled with brick pattern

private void DrawBrickPattern()

{

// Define brick tilling pattern resource

// Arguments: PdfDocument class, Scale factor (0.25), Stroking color (lines between bricks), Nonstroking color (brick)

// Return value: tilling pattern resource

PdfTilingPattern BrickPattern = PdfTilingPattern.SetBrickPattern(Document, 0.25, Color.LightYellow, Color.SandyBrown);

// save graphics state

Contents.SaveGraphicsState();

// set outline width 0.04"

Contents.SetLineWidth(0.04);

// set outline color to purple

Contents.SetColorStroking(Color.Purple);

// set fill pattern to brick

Contents.SetPatternNonStroking(BrickPattern);

// draw rounded rectangle filled with brick pattern

Contents.DrawRoundedRectangle(1.1, 5.0, 1.4, 1.5, 0.2, PaintOp.CloseFillStroke);

// restore graphics sate

Contents.RestoreGraphicsState();

return;

}

### 3.9. Draw Image and Clip it

The DrawImage method is an example of drawing an image. The PdfFileWriter support drawing images stored in all image files supported by Bitmap class and Metafile class. The ImageFormat class defines all image types. The JPEG image file type is the native image format of the PDF file. If you call the PdfImage constructor with JPEG file, the program copies the file as is into the PDF file. If you call the PdfImage constructor with any other type of image file, the program converts it into JPEG file. In order to keep the PDF file size as small as possible, make sure your image file resolution is not unreasonably high.

The PdfImage class loads the image and calculates maximum size that can fit a given image size in user coordinates and preserve the original aspect ratio. Before drawing the image we create an oval clipping path to clip the image.

// Draw image and clip it

private void DrawImage()

{

// define local image resources

// resolution 96 pixels per inch, image quality 50%

PdfImage Image1 = new PdfImage(Document, "TestImage.jpg", 96.0, 50);

// save graphics state

Contents.SaveGraphicsState();

// translate coordinate origin to the center of the picture

Contents.Translate(2.6, 5.0);

// adjust image size and preserve aspect ratio

ImageSizePos NewSize = Image1.ImageSizePosition(1.75, 1.5, ContentAlignment.MiddleCenter);

// clipping path

Contents.DrawOval(NewSize.DeltaX, NewSize.DeltaY, NewSize.Width, NewSize.Height, PaintOp.ClipPathEor);

// draw image

Contents.DrawImage(Image1, NewSize.DeltaX, NewSize.DeltaY, NewSize.Width, NewSize.Height);

// restore graphics state

Contents.RestoreGraphicsState();

return;

}

### 3.10. Draw Heart

The DrawHeart method displays a heart shape by defining the two ends of the center line. The DrawHeart method of the PdfContents class is a special case of drawing two symmetric Bezier curves forming a path.

// Draw heart

private void DrawHeart()

{

// save graphics state

Contents.SaveGraphicsState();

// draw heart

// The first argument are the coordinates of the centerline of the heart shape.

// The DrawHeart is a special case of DrawDoubleBezierPath method.

Contents.SetColorNonStroking(Color.HotPink);

Contents.DrawHeart(new LineD(new PointD(4.98, 5.1), new PointD(4.98, 6.0)), PaintOp.CloseFillStroke);

// restore graphics state

Contents.RestoreGraphicsState();

return;

}

### 3.11. Draw Pie Chart

The DrawChart method is an example of defining a chart and drawing it to the PDF document.

// Draw pie chart

private void DrawChart()

{

// save graphics state

Contents.SaveGraphicsState();

// create chart

Chart PieChart = PdfChart.CreateChart(Document, 1.8, 1.5, 300.0);

// create PdfChart object with Chart object

PdfChart PiePdfChart = new PdfChart(Document, PieChart);

// make sure we have good quality image

PieChart.AntiAliasing = AntiAliasingStyles.All;

// set colors

PieChart.BackColor = Color.FromArgb(220, 220, 255);

PieChart.Palette = ChartColorPalette.BrightPastel;

// default font

Font DefaultFont = MyPdfChart.CreateFont("Verdana", FontStyle.Regular, 0.05, FontSizeUnit.UserUnit);

Font TitleFont = MyPdfChart.CreateFont("Verdana", FontStyle.Bold, 0.07, FontSizeUnit.UserUnit);

// title (font size is 0.25 inches)

Title Title1 = new Title("Pie Chart Example", Docking.Top, TitleFont, Color.Purple);

PieChart.Titles.Add(Title1);

// legend

Legend Legend1 = new Legend();

PieChart.Legends.Add(Legend1);

Legend1.BackColor = Color.FromArgb(230, 230, 255);

Legend1.Docking = Docking.Bottom;

Legend1.Font = DefaultFont;

// chart area

ChartArea ChartArea1 = new ChartArea();

PieChart.ChartAreas.Add(ChartArea1);

// chart area background color

ChartArea1.BackColor = Color.FromArgb(255, 200, 255);

// series 1

Series Series1 = new Series();

PieChart.Series.Add(Series1);

Series1.ChartType = SeriesChartType.Pie;

Series1.Font = DefaultFont;

Series1.IsValueShownAsLabel = true;

Series1.LabelFormat = "{0} %";

// series values

Series1.Points.Add(22.0);

Series1.Points[0].LegendText = "Apple";

Series1.Points.Add(27.0);

Series1.Points[1].LegendText = "Banana";

Series1.Points.Add(33.0);

Series1.Points[2].LegendText = "Orange";

Series1.Points.Add(18.0);

Series1.Points[3].LegendText = "Grape";

Contents.DrawChart(MyPdfChart, 5.6, 5.0);

// restore graphics state

Contents.RestoreGraphicsState();

return;

}

### 3.12. Draw Text Box

The DrawTextBox method is an example of using the TextBox class. The TextBox class formats text to fit within a column. The text can be drawn using a verity of font's styles and sizes.

// Draw example of a text box

private void DrawTextBox()

{

// save graphics state

Contents.SaveGraphicsState();

// translate origin to PosX=1.1" and PosY=1.1" this is the bottom left corner of the text box example

Contents.Translate(1.1, 1.1);

// Define constants

// Box width 3.25"

// Box height is 3.65"

// Normal font size is 9.0 points.

const Double Width = 3.15;

const Double Height = 3.65;

const Double FontSize = 9.0;

// Create text box object width 3.25"

// First line indent of 0.25"

TextBox Box = new TextBox(Width, 0.25);

// add text to the text box

Box.AddText(ArialNormal, FontSize,

"This area is an example of displaying text that is too long to fit within a fixed width " +

"area. The text is displayed justified to right edge. You define a text box with the required " +

"width and first line indent. You add text to this box. The box will divide the text into " +

"lines. Each line is made of segments of text. For each segment, you define font, font " +

"size, drawing style and color. After loading all the text, the program will draw the formatted text.\n");

Box.AddText(TimesNormal, FontSize + 1.0, "Example of multiple fonts: Times New Roman, ");

Box.AddText(Comic, FontSize, "Comic Sans MS, ");

Box.AddText(ArialNormal, FontSize, "Example of regular, ");

Box.AddText(ArialBold, FontSize, "bold, ");

Box.AddText(ArialItalic, FontSize, "italic, ");

Box.AddText(ArialBoldItalic, FontSize, "bold plus italic. ");

Box.AddText(ArialNormal, FontSize - 2.0, "Arial size 7, ");

Box.AddText(ArialNormal, FontSize - 1.0, "size 8, ");

Box.AddText(ArialNormal, FontSize, "size 9, ");

Box.AddText(ArialNormal, FontSize + 1.0, "size 10. ");

Box.AddText(ArialNormal, FontSize, DrawStyle.Underline, "Underline, ");

Box.AddText(ArialNormal, FontSize, DrawStyle.Strikeout, "Strikeout. ");

Box.AddText(ArialNormal, FontSize, "Subscript H");

Box.AddText(ArialNormal, FontSize, DrawStyle.Subscript, "2");

Box.AddText(ArialNormal, FontSize, "O. Superscript A");

Box.AddText(ArialNormal, FontSize, DrawStyle.Superscript, "2");

Box.AddText(ArialNormal, FontSize, "+B");

Box.AddText(ArialNormal, FontSize, DrawStyle.Superscript, "2");

Box.AddText(ArialNormal, FontSize, "=C");

Box.AddText(ArialNormal, FontSize, DrawStyle.Superscript, "2");

Box.AddText(ArialNormal, FontSize, "\n");

Box.AddText(Comic, FontSize, Color.Red, "Some color, ");

Box.AddText(Comic, FontSize, Color.Green, "green, ");

Box.AddText(Comic, FontSize, Color.Blue, "blue, ");

Box.AddText(Comic, FontSize, Color.Orange, "orange, ");

Box.AddText(Comic, FontSize, DrawStyle.Underline, Color.Purple, "and purple.\n");

Box.AddText(ArialNormal, FontSize, "Support for non-Latin letters: ");

Box.AddText(ArialNormal, FontSize, Contents.ReverseString( "עברית"));

Box.AddText(ArialNormal, FontSize, "АБВГДЕ");

Box.AddText(ArialNormal, FontSize, "&alpha;&beta;&gamma;&delta;&epsilon;&zeta;");

Box.AddText(ArialNormal, FontSize, "\n");

// Draw the text box

// Text left edge is at zero (note: origin was translated to 1.1")

// The top text base line is at Height less first line ascent.

// Text drawing is limited to vertical coordinate of zero.

// First line to be drawn is line zero.

// After each line add extra 0.015".

// After each paragraph add extra 0.05"

// Stretch all lines to make smooth right edge at box width of 3.15"

// After all lines are drawn, PosY will be set to the next text line after the box's last paragraph

Double PosY = Height;

Contents.DrawText(0.0, ref PosY, 0.0, 0, 0.015, 0.05, TextBoxJustify.FitToWidth, Box);

// Create text box object width 3.25"

// No first line indent

Box = new TextBox(Width);

// Add text as before.

// No extra line spacing.

// No right edge adjustment

Box.AddText(ArialNormal, FontSize,

"In the examples above this area the text box was set for first line indent of " +

"0.25 inches. This paragraph has zero first line indent and no right justify.");

Contents.DrawText(0.0, ref PosY, 0.0, 0, 0.01, 0.05, TextBoxJustify.Left, Box);

// Create text box object width 2.75

// First line hanging indent of 0.5"

Box = new TextBox(Width - 0.5, -0.5);

// Add text

Box.AddText(ArialNormal, FontSize,

"This paragraph is set to first line hanging indent of 0.5 inches. " +

"The left margin of this paragraph is 0.5 inches.");

// Draw the text

// left edge at 0.5"

Contents.DrawText(0.5, ref PosY, 0.0, 0, 0.01, 0.05, TextBoxJustify.Left, Box);

// restore graphics state

Contents.RestoreGraphicsState();

return;

}

### 3.13. Draw Book Order Form

The DrawBookOrderForm method is an example of an order entry form or an invoice. It is an example for data table support. It demonstrate the use of PdfTable, PdfTableCell and PdfTableStyle classes.

// Draw example of order form

private void DrawBookOrderForm()

{

// Define constants to make the code readable

const Double Left = 4.35;

const Double Top = 4.65;

const Double Bottom = 1.1;

const Double Right = 7.4;

const Double FontSize = 9.0;

const Double MarginHor = 0.04;

const Double MarginVer = 0.04;

const Double Border = 0.015;

const Double Grid = 0.01;

// column widths

Double ColWidthPrice = ArialNormal.TextWidth(FontSize, "9999.99") + 2.0 \* MarginHor;

Double ColWidthQty = ArialNormal.TextWidth(FontSize, "Qty") + 2.0 \* MarginHor;

Double ColWidthDesc = Right - Left - Border - 3 \* Grid - 2 \* ColWidthPrice - ColWidthQty;

// define table

PdfTable Table = new PdfTable(Page, Contents, ArialNormal, FontSize);

Table.TableArea = new PdfRectangle(Left, Bottom, Right, Top);

Table.SetColumnWidth(new Double[] {ColWidthDesc, ColWidthPrice, ColWidthQty, ColWidthPrice});

// define all borders

Table.Borders.SetAllBorders(FrameWidth, GridWidth);

// margin

PdfRectangle Margin = new PdfRectangle(MarginHor, MarginVer);

// default header style

Table.DefaultHeaderStyle.Margin = Margin;

Table.DefaultHeaderStyle.BackgroundColor = Color.FromArgb(255, 196, 255);

Table.DefaultHeaderStyle.Alignment = ContentAlignment.MiddleRight;

// private header style for description

Table.Header[0].Style = Table.HeaderStyle;

Table.Header[0].Style.Alignment = ContentAlignment.MiddleLeft;

// table heading

Table.Header[0].Value = "Description";

Table.Header[1].Value = "Price";

Table.Header[2].Value = "Qty";

Table.Header[3].Value = "Total";

// default style

Table.DefaultCellStyle.Margin = Margin;

// description column style

Table.Cell[0].Style = Table.CellStyle;

Table.Cell[0].Style.MultiLineText = true;

// qty column style

Table.Cell[2].Style = Table.CellStyle;

Table.Cell[2].Style.Alignment = ContentAlignment.BottomRight;

Table.DefaultCellStyle.Format = "#,##0.00";

Table.DefaultCellStyle.Alignment = ContentAlignment.BottomRight;

Contents.DrawText(ArialBold, FontSize, 0.5 \* (Left + Right), Top + MarginVer + Table.DefaultCellStyle.FontDescent,

TextJustify.Center, DrawStyle.Normal, Color.Purple, "Example of PdfTable support (new for 1.10.0)");

// reset order total

Double Total = 0;

// loop for all items in the order

// Order class is a atabase simulation for this example

foreach(Order Book in Order.OrderList)

{

Table.Cell[0].Value = Book.Title + ". By: " + Book.Authors;

Table.Cell[1].Value = Book.Price;

Table.Cell[2].Value = Book.Qty;

Table.Cell[3].Value = Book.Total;

Table.DrawRow();

// accumulate total

Total += Book.Total;

}

Table.Close();

// save graphics state

Contents.SaveGraphicsState();

// form line width 0.01"

Contents.SetLineWidth(FrameWidth);

Contents.SetLineCap(PdfLineCap.Square);

// draw total before tax

Double[] ColumnPosition = Table.ColumnPosition;

Double TotalDesc = ColumnPosition[3] - MarginHor;

Double TotalValue = ColumnPosition[4] - MarginHor;

Double PosY = Table.RowTopPosition - 2.0 \* MarginVer - Table.DefaultCellStyle.FontAscent;

Contents.DrawText(ArialNormal, FontSize, TotalDesc, PosY, TextJustify.Right, "Total before tax");

Contents.DrawText(ArialNormal, FontSize, TotalValue, PosY, TextJustify.Right, Total.ToString("#.00"));

// draw tax (Ontario Canada HST)

PosY -= Table.DefaultCellStyle.FontLineSpacing;

Contents.DrawText(ArialNormal, FontSize, TotalDesc, PosY, TextJustify.Right, "Tax (13%)");

Double Tax = Math.Round(0.13 \* Total, 2, MidpointRounding.AwayFromZero);

Contents.DrawText(ArialNormal, FontSize, TotalValue, PosY, TextJustify.Right, Tax.ToString("#.00"));

// draw total line

PosY -= Table.DefaultCellStyle.FontDescent + 0.5 \* MarginVer;

Contents.DrawLine(ColumnPosition[3], PosY, ColumnPosition[4], PosY);

// draw final total

PosY -= Table.DefaultCellStyle.FontAscent + 0.5 \* MarginVer;

Contents.DrawText(ArialNormal, FontSize, TotalDesc, PosY, TextJustify.Right, "Total payable");

Total += Tax;

Contents.DrawText(ArialNormal, FontSize, TotalValue, PosY, TextJustify.Right, Total.ToString("#.00"));

PosY -= Table.DefaultCellStyle.FontDescent + MarginVer;

Contents.DrawLine(ColumnPosition[0], Table.RowTopPosition, ColumnPosition[0], PosY);

Contents.DrawLine(ColumnPosition[0], PosY, ColumnPosition[4], PosY);

Contents.DrawLine(ColumnPosition[4], Table.RowTopPosition, ColumnPosition[4], PosY);

// restore graphics state

Contents.RestoreGraphicsState();

return;

}

4. References

* Adobe PDF file specification document available from Adobe website: [“PDF Reference, Sixth Edition, Adobe Portable Document Format Version 1.7 November 2006”](http://www.adobe.com/content/dam/Adobe/en/devnet/acrobat/pdfs/pdf_reference_1-7.pdf).
* Information about OpenType font specifications can be found at [Microsoft Typography - OpenType Specification](http://www.microsoft.com/typography/otspec/).
* Source for a decompression class matching the PDF Deflate compression class is available at ["PDF File Analyzer With C# Parsing Classes"](http://www.codeproject.com/Articles/450254/PDF-File-Analyzer-With-Csharp-Parsing-Classes) article.
* The original source code for both compression and decompression classes is taken from [“Processing Standard ZIP Files with C# Compression/Decompression Classes”](http://www.codeproject.com/Articles/359758/Processing-Standard-Zip-Files-with-Csharp-compress) article by Uzi Granot published in CodeProject.com website.

5. Other Open Source Software by This Author

* [Android Color Selector for Programmers](https://play.google.com/store/apps/details?id=com.granotech.colorselector) is a free App available on Google Play.
* [Google Play In-App Billing Demo App](http://www.codeproject.com/Articles/720303/Google-Play-In-App-Billing-Demo-App) This article is an example of Google Play In-App Billing Version 3.

6. History

* 2013/04/01: Version 1.0 Original Version.
* 2013/04/09: Version 1.1 Support for countries with decimal separator other than period.
* 2013/07/21: Version 1.2 The original revision supported image resources with jpeg file format only. Version 1.2 support all image files acceptable to Bitmap class. See ImageFormat class. The program was tested with: Bmp, Gif, Icon, Jpeg, Png and Tiff. See Section 2.3 and Section 3.8 above.
* 2014/02/07: Version 1.3 Fix bug in PdfContents.DrawBezierNoP2(PointD P1, PointD P3).
* 2014/03/01: Version 1.4 Improved support for character substitution. Improved support for image inclusion. Some fixes related to PdfXObject.
* 2014/05/05: Version 1.5 Barcode support without use of fonts. Four barcodes are included: Code-128, Code-39, UPC-A and EAN-13. See Section 2.5 and Section 3.7.
* 2014/07/09: Version 1.6 (1) The CreateFile method resets the PdfDocument to initial condition after file creation. (2) The PdfFont object releases the unmanaged code resources properly.
* 2014/08/25: Version 1.7 Support for document encryption, web-links and QR Code.
* 2014/09/12: Version 1.8 Support for bookmarks.
* 2014/10/06: Version 1.9 Support for charting, PrintDocument and image Metafiles.
* 2014/10/12: Version 1.9.1 Fix to ChartExample. Parse numeric fields in regions with decimal separator other than period.
* 2014/12/02: Version 1.10.0 Support for data tables. Add source code documentation. Increase maximum number of images per document.
* 2015/01/12: Version 1.11.0 Support for video, sound and attachment files. Add support for Interleave 2 of 5 barcode.
* 2015/04/13: Version 1.12.0 Support for reordering pages and enhance data table border lines support.
* 2015/05/05: Version 1.13.0 PDF document output to a stream. PDF table insert page break. Image quality enhancement. Support for Standard-128 (RC4) encryption.
* 2015/06/08: Version 1.14.0 Support for long text blocks or TextBox within PDF Table.
* 2015/06/09: Version 1.14.1 one line change to Copy method of PdfTableStyle class.
* 2015/06/17: Version 1.15.0 Document information dictionary. PdfImage rewrite. Additional image saving options.