



Essential Studio 2013 Volume 4 - v.11.4.0.26

Essential PDF



Contents

1	Overview	8
1.1	Introduction to Essential PDF	8
1.2	Prerequisites and Compatibility	10
1.3	Documentation	11
2	Installation and Deployment	14
2.1	Installation.....	14
2.2	Samples Location	14
2.3	Deployment Requirements	25
2.3.1	DLLs	26
2.3.2	Web Application deployment.....	26
3	Getting Started	28
3.1	Class Diagram	28
3.2	Creating Platform Application.....	30
3.3	Deploying Essential PDF	34
3.3.1	Windows	34
3.3.2	ASP.NET	38
3.3.3	WPF.....	42
3.3.4	Silverlight	46
3.3.5	ASP.NET MVC	50
3.4	PDF Version Compatibility.....	56
3.5	PDF Features	57
4	Concepts and Features	61
4.1	PDF Generator	61
4.1.1	Introduction.....	61
4.1.1.1	Document Object Model	61
4.1.1.2	PDF Layers.....	69
4.1.1.2.1	Use Case Scenario	69
4.1.1.2.2	Properties Tables	69
4.1.1.2.3	Creating and Embedding PDF Layers in the PDF Document	69

4.1.1.2.4	Viewing Layers Samples	71
4.1.1.3	Pen and Brush.....	75
4.1.1.4	Fonts	77
4.1.2	Drawing	79
4.1.2.1	Text	80
4.1.2.1.1	Drawing Text	80
4.1.2.1.2	Html Styled Text	83
4.1.2.1.3	Automatic Fields	85
4.1.2.1.4	Links	89
4.1.2.2	Graphics	90
4.1.2.2.1	Graphic Elements	94
4.1.2.2.2	ColorSpace	95
4.1.2.2.3	Images.....	103
4.1.2.2.4	Barcode	107
4.1.2.2.5	QR Barcode.....	117
4.1.2.2.6	Pagination.....	120
4.1.2.2.7	Page Templates	123
4.1.2.3	Tables	125
4.1.2.3.1	PdfLightTable	126
4.1.2.3.2	PdfGrid	146
4.1.2.4	Lists	165
4.1.3	Interactive Features.....	167
4.1.3.1	Actions	168
4.1.3.2	Annotation	170
4.1.3.2.1	3D Annotation.....	172
4.1.3.2.2	Hyperlinks for other external files	179
4.1.3.2.3	Free Text Annotation.....	180
4.1.3.3	Attachments.....	183
4.1.3.4	Portfolio	185
4.1.3.4.1	Creating a PDF Portfolio	186
4.1.3.4.2	Properties	189
4.1.3.4.3	Adding Custom File Fields and Attributes into a PDF Portfolio	189
4.1.4	Security.....	193
4.1.4.1	Encryption.....	193

4.1.4.2	Signing	195
4.1.5	Standards for PDF/A-1 Compliance	200
4.1.5.1	PDF/A-1b	200
4.1.5.2	PDF/X-1a	203
4.1.6	Settings	204
4.1.6.1	Document Settings	204
4.1.6.2	Custom Metadata	206
4.1.6.3	Compression	209
4.1.7	Tutorial	210
4.1.7.1	Drawing Text	211
4.1.7.1.1	Draw Simple text	211
4.1.7.1.2	Text Pagination	213
4.1.7.1.3	Draw Rich text	216
4.1.7.1.4	Draw Right-To-Left Text	218
4.1.7.2	Drawing Shapes	220
4.1.7.3	Drawing Images	223
4.1.7.4	Drawing Table	230
4.1.7.5	Stream Support	233
4.1.7.6	Security Settings	234
4.1.7.7	Page Settings	246
4.1.7.8	Headers and Footers	250
4.2	PDF Editing	253
4.2.1	Load Document	253
4.2.1.1	Bookmark	256
4.2.1.2	Adding a New Page	261
4.2.1.3	Adding an Attachment	263
4.2.1.4	Dynamic Fields	265
4.2.2	Import Pages	271
4.2.3	Replacing Images	273
4.2.4	Merge PDF	275
4.2.5	Split PDF	275
4.2.6	Transform PDF	277
4.2.7	Document Information	278
4.2.8	Booklet	279
4.2.9	OCR Support	282

4.2.9.1	Tables for Properties and Methods	282
4.2.9.2	Prerequisites.....	283
4.2.9.2.1	Dlls.....	283
4.2.9.2.2	Windows Deployment.....	283
4.2.9.2.3	Web Deployment.....	284
4.2.9.3	Performing OCR for a Complete PDF Document	284
4.2.9.4	Perform OCR for a Specific Region of PDF Document.....	285
4.2.10	Tutorial 287	
4.2.10.1	Merge PDF	287
4.2.10.2	Import Pages As Templates	289
4.2.10.3	Signature	289
4.3	PDF Form	292
4.3.1	Overview.....	293
4.3.2	Form Fields.....	297
4.3.3	Form Filling.....	308
4.4	PDF Convertor.....	316
4.4.1	HTML To PDF	316
4.4.1.1	HTML to PDF Conversion using Gecko Rendering Engine	322
4.4.1.1.1	Use Case Scenario	323
4.4.1.1.2	Prerequisites.....	323
4.4.1.1.3	Installation Steps	323
4.4.1.1.4	Conversion of HTML to PDF using Gecko Rendering Engine	323
4.4.1.2	Tagged PDF	324
4.4.2	Text Extraction.....	326
4.4.3	ImageExtraction.....	327
4.4.4	Doc To PDF.....	328
4.4.5	XPS to PDF	332
4.4.6	Tutorial.....	335
4.4.6.1	Importing.....	335
4.5	Asynchronous Support	337
4.6	Supported Elements	338
5	Frequently Asked Questions	342
5.1	PDF Generator	342
5.1.1	Common	342

5.1.1.1	How To Add Sections And Pages?	342
5.1.1.2	How To Compress a PDF Document?	343
5.1.1.3	How To Create a Borderless Table?	343
5.1.1.4	How To Create Page Labels?	344
5.1.1.5	How To Draw a SoftMask Image Into a PDF Document?	345
5.1.1.6	How To Embed 3D Files In a PDF Document?	345
5.1.1.7	How To Embed Fonts?	348
5.1.1.8	How To Find the End Position Of a Table?	349
5.1.1.9	How To Render Unicode Text [CJK Fonts] In a PDF Document?	350
5.1.1.10	How To Set Graphic Units?	351
5.1.1.11	How To Dispose The Pdf document Object?	352
5.1.1.12	How to open the generated PDF document into the browser instead of displaying the Open/Save dialog in browser?	352
5.1.1.13	How to set the default view of Navigation Pane in Viewer?	353
5.1.2	Advanced.....	354
5.1.2.1	How To Add the Tables One After Another?	355
5.1.2.2	How To Draw an Image In a Table Cell?	355
5.1.2.3	How To Implement Column Span In PdfLightTable?	356
5.1.2.4	How To Improve Performance?	358
5.1.2.5	How To Insert a Table In The Header?	359
5.1.2.6	How To Measure the String Whose End Position Is Not Known?	360
5.1.2.7	How to set margins for the PDF pages?	361
5.1.2.8	How to edit header in PdfGrid?	362
5.1.2.9	How to set width for Table?	362
5.1.2.10	How to specify bounds for Table during pagination?	363
5.2	PDF Editing	363
5.2.1	How To Access Pages In an Existing Document?	363
5.2.2	How To Add Watermarks Or Stamps In an Existing Document?	364
5.2.3	How To Convert Units In PDF / What Are the Units Of the Elements In PDF?	365
5.2.4	How To Store And Retrieve a PDF Document From the Database?	365
5.2.5	How To Enable and Disable PDF Layers In a PDF document?	366
5.2.6	How To Read Conformance Level From PDF?	367
5.3	PDF Form	367
5.3.1	How To Create a Form Which Transfers Data To the Server?	367

5.4	PDF Convertor.....	375
5.4.1	How To Convert Secure Webpages?	375
5.4.2	How To Extract Images From an Existing PDF Document?	377
5.4.3	How To Extract Text From an Existing PDF Document?	378
5.4.4	Register the Syncfusion Gecko Wrapper manually?	379
5.4.5	How can we make IE9 to render metafile?.....	379
5.5	PDF Grid.....	379
5.5.1	How to position the background image as a tile?.....	379
5.5.2	How to make background images centered, stretched and fitted within the PDF Grid?	380

1 Overview

This section covers information on the Essential PDF, key features, and prerequisites to use the control, its compatibility with various OS and browsers and finally the documentation details complimentary with the product.

1.1 Introduction to Essential PDF

Essential PDF is a .NET library with the capability to produce Adobe PDF files. It features a full-fledged object model for the easy creation of PDF files from any .NET language, providing complete control over the positioning of elements in a PDF document. It does not use any external libraries and is built from scratch in C#. It can be used on the server-side (ASP.NET or any other environment) or with Windows Forms / WPF applications. The usage is common for all the environments except for the part where the created document is saved to the disk or stream in case of Windows Forms / WPF applications, and streamed to the client browser in the case of ASP.NET applications.

 Essential PDF will be beneficial for users who want to create a PDF document with features such as, drawing text and images, inserting tables and shapes, security protection, conversion features and so on.

Real World Scenario

PDF is widely used in business, education and government, for the distribution of business letters, monthly reports, declarations or business reports. PDF can be used as the electronic data format between creators, designers, reviewers, printers and advertising companies.

Advantages of Essential PDF

Following are the advantages of using PDF in applications of different platforms:

- There are no temporary documents saved on the server in an ASP.NET application.
- The generated PDF document can also be protected using 40-Bit and 128-Bit encryption.
- The PDF file that is created using Essential PDF can be viewed using **any version of Adobe Acrobat or the free version of Acrobat Viewer** from Adobe. Find more details on creating files with different versions in PDF Version Compatibility topic.



Essential PDF

Figure 1: Essential PDF

Key Features

Some of the key features of PDF are listed below:

- Essential PDF can draw MultiPage text, formatted text etc., in pdf pages.
- It supports predefined fonts, true type fonts and CJK fonts.
- PDF can draw shapes such as rectangles, circles, arcs, ellipses and fill them with custom brushes.
- Essential PDF can load Images to pdf documents from streams and files on disk. It can draw both scalar and vector images in the documents. Soft Mask Images and Watermarks can also be drawn in the pages.
- ADO.NET tables can be imported into the pdf files; the rows and columns can be formatted. You can also insert graphic elements inside the table.
- Auto Page Breaks for large tables can be enabled in the pdf documents.
- Drawing Form Fields like buttons, text boxes, list boxes, check boxes, and so on, into the PDF document.
- Drawing Header and Footer in a PDF document.
- PDF documents are extremely secured by setting the Owner or User Password; allowing or disallowing permissions over a PDF document; 40 or 128 bit encryption is available.
- Secure your PDF document with high protection. Essential PDF enables you to secure your document with the author signature. Also, create visible and invisible signatures in the document, providing an option to preserve content or add signatures of your interest respectively.
- It has features like reading existing PDF acroforms and filling the acroform.
- HTML web page pagination is available to render large web pages into PDF without any text truncation at page breaks. Also, you can render the web pages as streams to create PDF without any intermediate files on disk.

User Guide Organization

The product comes with numerous samples as well as an extensive documentation to guide you. This User Guide provides detailed information on the features and functionalities of Essential PDF. It is organized into the following sections:

- **Overview**-This section gives a brief introduction to our product and its key features.
- **Installation and Deployment**-This section elaborates on the install location of the samples, license etc.
- **What's New**-This section lists the new features implemented for the current release.
- **Getting Started**-This section guides you on getting started with various platform application, controls etc.
- **Concepts and Features**-The features of the PDF is illustrated with use case scenarios, code examples and screen shots under this section.
- **Frequently Asked Questions**-This section illustrates the solutions for various task-based queries about Essential PDF.

Document Conventions

The conventions listed below will help you to quickly identify the important sections of information, while using the content:

Convention	Icon	Description
Note	 Note:	Represents important information
Example	Example	Represents an example
Tip		Represents useful hints that will help you in using the controls/features
Important Note		Represents additional information on the topic

1.2 Prerequisites and Compatibility

This section covers the requirements mandatory for using Essential PDF. It also lists operating systems and browsers, compatible with the product.

Prerequisites

The prerequisites details are listed below:

Development Environments	<ul style="list-style-type: none"> Visual Studio 2012 (Ultimate, Premium, Professional, and Express) Visual Studio 2010 (Ultimate, Premium, Professional, and Express) Visual Studio 2008 (Team System, Professional, Standard, and Express) Visual Studio 2005 (Professional, Standard, and Express) Silverlight 4.0
.NET Framework versions	<ul style="list-style-type: none"> .NET 4.5 .NET 4.0 .NET 3.5 SP1 .NET 2.0

Compatibility

The compatibility details are listed below:

Operating Systems	<ul style="list-style-type: none"> Windows 8 (32 bit and 64 bit) Windows Server 2008 (32 bit and 64 bit) Windows 7 (32 bit and 64 bit) Windows Vista (32 bit and 64 bit) Windows XP Windows 2003
-------------------	--

1.3 Documentation

Syncfusion provides the following documentation segments to provide all necessary information for using Essential PDF for various platforms in an efficient manner.

Type of documentation	Location
Readme	Windows Forms -[drive:]\Program Files\Syncfusion\Essential Studio\x.x.x\x\Infrastructure\Data\Release Notes\readme.htm ASP.NET -[drive:]\Program Files\Syncfusion\Essential Studio\x.x.x\x\Infrastructure\Data\asp release notes\readme.htm

	<p>WPF-[drive:]\\Program Files\\Syncfusion\\Essential Studio\\x.x.x.x\\Infrastructure\\Data\\WPF release notes\\readme.htm</p> <p>Silverlight-[drive:]\\Program Files\\Syncfusion\\Essential Studio\\x.x.x.x\\Infrastructure\\Data\\Silverlight Release Notes\\readme.htm</p> <p>ASP.NET MVC-[drive:]\\Program Files\\Syncfusion\\Essential Studio\\x.x.x.x\\Infrastructure\\Data\\MVC Release Notes\\readme.htm</p>
Release Notes	<p>Windows Forms-[drive:]\\Program Files\\Syncfusion\\Essential Studio\\x.x.x.x\\Infrastructure\\Data\\Release Notes\\Release Notes.htm</p> <p>ASP.NET-[drive:]\\Program Files\\Syncfusion\\Essential Studio\\x.x.x.x\\Infrastructure\\Data\\asp release notes\\Release Notes.htm</p> <p>WPF-[drive:]\\Program Files\\Syncfusion\\Essential Studio\\x.x.x.x\\Infrastructure\\Data\\WPF release notes\\Release Notes.htm</p> <p>Silverlight-[drive:]\\Program Files\\Syncfusion\\Essential Studio\\x.x.x.x\\Infrastructure\\Data\\Silverlight Release Notes\\Release Notes.htm</p> <p>ASP.NET MVC-[drive:]\\Program Files\\Syncfusion\\Essential Studio\\x.x.x.x\\Infrastructure\\Data\\MVC Release Notes\\Release Notes.htm</p>
User Guide (this document)	<p>Online http://help.syncfusion.com/resources (Navigate to the PDF User Guide.)</p>  <p>Note: Click Download as PDF to access a PDF version.</p> <p>Installed Documentation</p>

	Dashboard -> Documentation -> Installed Documentation.
Class Reference	Online http://help.syncfusion.com/resources (Navigate to the Reporting User Guide. Select <i>PDF</i> , and then click the Class Reference link found in the upper right section of the page.) Installed Documentation Dashboard -> Documentation -> Installed Documentation.

2 Installation and Deployment

This section covers information on the install location, samples, licensing, patches update and updation of the recent version of Essential Studio. It comprises the following sub-sections:

2.1 Installation

Refer the following path for step-by-step installation of Essential Studio set-up.

Common UG -> Installation and Deployment -> Installation topic

For More Information Refer:

For licensing, patches and information on adding or removing selective components, refer the following topics in Common UG under **Installation and Deployment**.

- Licensing
- Patches
- Add / Remove Components

2.2 Samples Location

This section covers the location of the installed samples and describes the procedure to run the samples through the sample browser and online. It also lists the location of source code.

Sample installation locations

Sample install locations for different platforms are listed below:

- **ASP.NET**-The ASP.NET samples are installed in the following location:

...My Documents\Syncfusion\EssentialStudio\Version Number\Web\pdf.web\Samples\2.0

- **ASP.NET MVC**-The ASP.NET MVC samples are installed in the following location:

...\\My Documents\\Syncfusion\\EssentialStudio\\<Version Number>\\MVC\\pdfmvcl\\samples\\3.5

- **Windows Forms**-The Windows Forms samples are installed in the following location:

*...\\My Documents\\Syncfusion\\EssentialStudio\\Version
Number\\Windows\\PDF.Windows\\Samples\\2.0*

- **WPF**-The WPF samples are installed in the following location:

...\\My Documents\\Syncfusion\\EssentialStudio\\Version Number\\WPF\\pdf.WPF\\Samples\\3.5

- **Silverlight**-The Silverlight samples are installed in the following location:

*...\\My Documents\\Syncfusion\\EssentialStudio\\Version
Number\\Silverlight\\pdf.Silverlight\\Samples\\3.5*

Viewing Samples

To view the samples:

1. Click **Start -> All Programs -> Syncfusion -> Essential Studio <x.x.x.x> -> Dashboard -> Reporting Edition.**

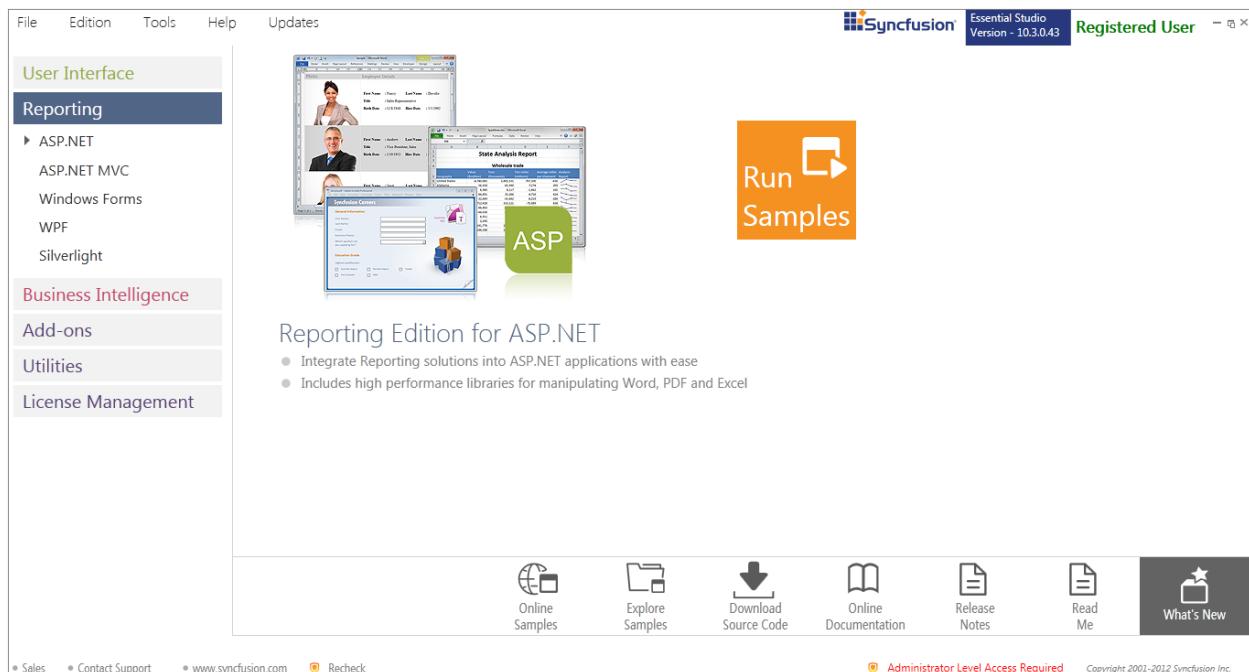


Figure 2: Syncfusion Essential Studio Dashboard Reporting Edition

The steps to view the pdf samples in various platforms are discussed below:

ASP.NET

1. Click the drop-down button of the **ASP.NET** platform. The following options are displayed and You can view the samples in the following three ways:
 - **Run Samples** - View the locally installed PDF samples for ASP.NET using the sample browser
 - **Online Samples** - View the online PDF samples for .NET
 - **Explore Samples** - Locate the samples for PDF on the disk
2. Click **Run Samples** link. Essential Studio ASP.NET - Reporting Edition sample browser is displayed.

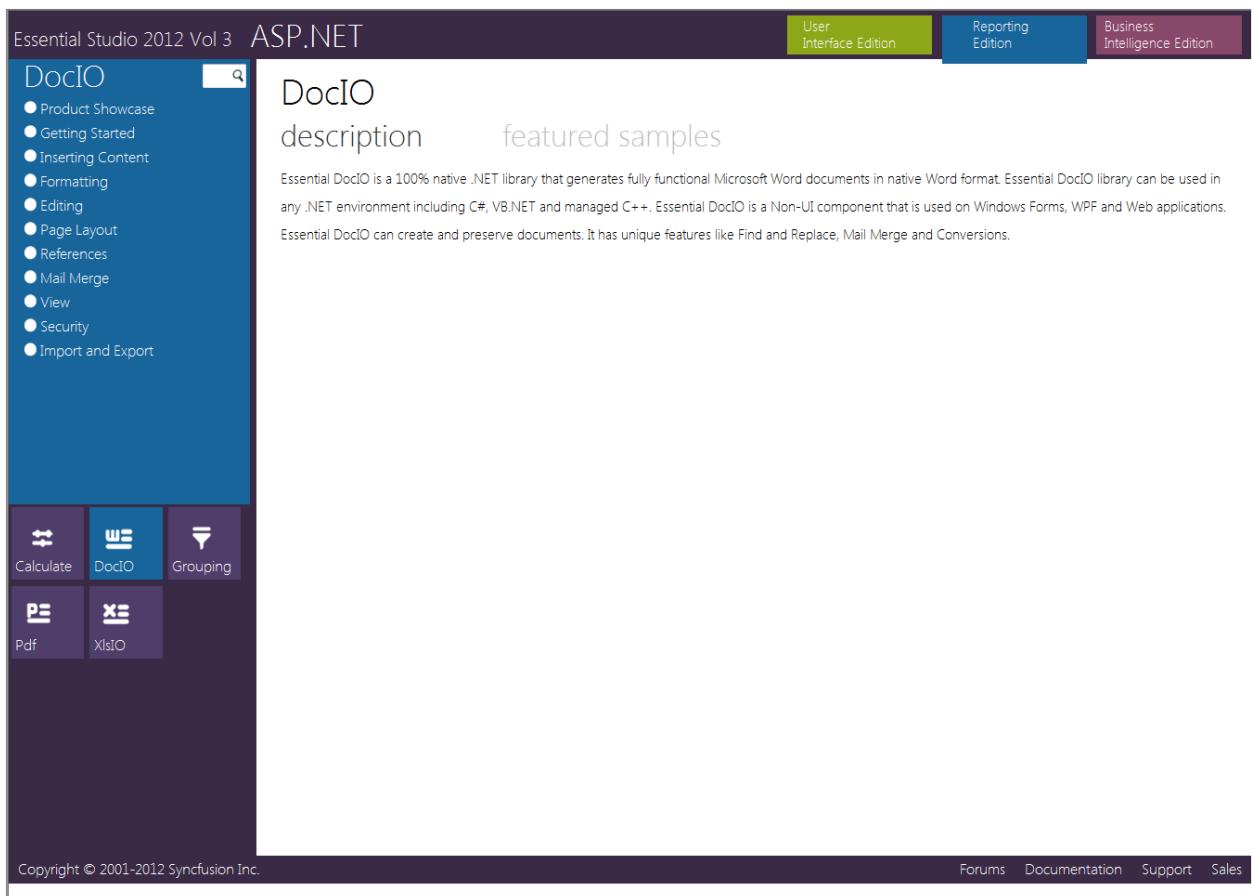


Figure 3: ASP.NET Sample Browser



Note: By default, the samples of Essential DocIO are displayed.

3. Click **Pdf** from the bottom-left pane.

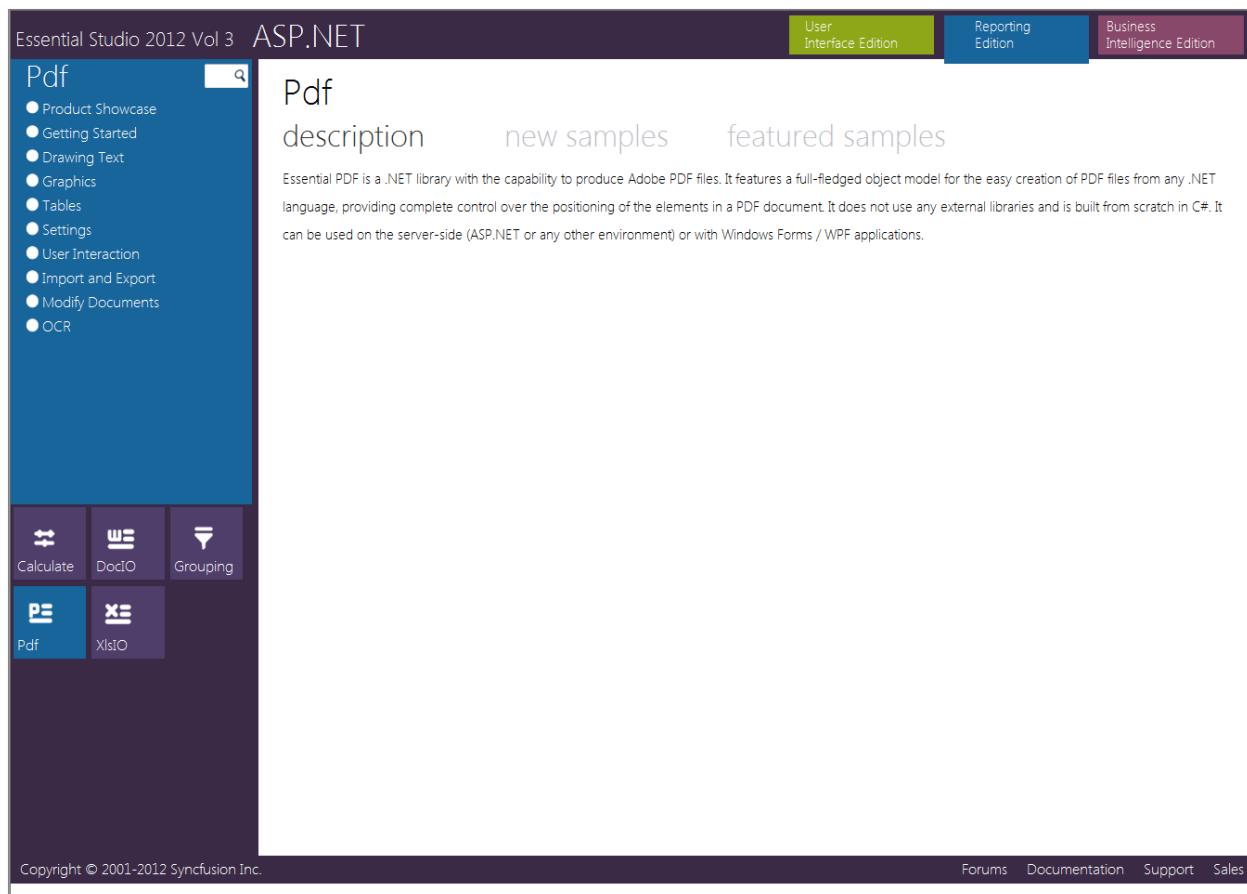


Figure 4: ASP.NET Sample Browser Showing PDF Samples

The samples will be displayed.

4. Select any sample and browse through the features.

ASP.NET MVC

To view the samples:

1. Click the drop-down button of the **ASP.NET MVC** platform. You can view the samples in any of the three options that get displayed.
2. Click **Run Samples** link. Essential Studio **ASP.NET MVC - Reporting** Edition sample browser is displayed.

The screenshot shows the ASP.NET MVC Sample Browser interface. At the top, there's a navigation bar with links for 'Essential Studio 2012 Volume 3' and 'Download Trial'. Below the navigation bar, the main content area has a title 'ASP.NET MVC' and a sidebar titled 'DocIO' containing a table of contents and a grid of icons representing various features like Grid, Tools, Chart, Gauge, Diagram, Schedule, DodO, Pdf, XlsIO, PdfViewer, and OlapGrid.

The main content area displays several examples:

- Project Status Report:** A Microsoft Word document showing a table of employee details.
- Employee Details:** A screenshot of a Microsoft Word document showing employee information.
- Featured Samples:**
 - Employee Report:** Shows a screenshot of a Microsoft Word document with the heading 'Employee Details' and a photo of Nancy Davolio.
 - Table of Contents:** Displays a table of contents for a program structure, including sections like 'Introduction', 'Hello world', 'Program structure', 'Types and variables', and 'Expressions'.
 - Clone and Merge:** Shows a process flow diagram where three 'W' icons are combined into one.
 - Forms:** Shows a screenshot of a Microsoft Word document with a form titled 'Educational Qualification' containing fields for Type, Institution, Programming Language, and Grade.
 - Update Fields:** Shows a screenshot of a Microsoft Word document with a large red exclamation mark icon.

Figure 5: ASP.NET MVC Sample Browser



Note: By default, the samples of Essential DocIO are displayed.

3. Click **Pdf** from the bottom-left pane.

The screenshot shows the ASP.NET MVC Sample Browser interface. At the top, there's a navigation bar with links for 'Essential Studio 2012 Volume 3' and 'Download Trial'. Below the navigation bar, the main content area has a sidebar on the left titled 'PDF' containing a search icon and a list of features: Product Showcase, Getting Started, Graphics, Tables, Drawing Text, Security, Settings, User Interaction, Import and Export (marked as 'updated'), Viewer, Modify Documents (marked as 'new'), and OCR.

The main content area displays several samples:

- Featured Samples:**
 - Merge Documents: Shows two document icons merging into one.
 - Overlay Documents: Shows two document icons with an overlay effect.
 - Form Fill: A table showing levels (Platinum, Gold, Silver) and fees (\$85,000, \$40,000, \$13,000).
 - Northwind Report: A screenshot of a report viewer showing customer data from the Northwind database.
- Grid:** A grid view sample.
- Tools:** A tools sample.
- Chart:** A chart sample.
- Gauge:** A gauge sample.
- Diagram:** A diagram sample.
- Schedule:** A schedule sample.
- DocIO:** A DocIO sample.
- PDF:** A PDF sample.
- XlsIO:** An XlsIO sample.
- PdfViewer:** A PdfViewer sample.
- OlapGrid:** An OlapGrid sample.

Figure 6: ASP.NET MVC Sample Browser Showing PDF Samples

4. Select any sample and browse through the features.

Windows Forms

1. Click the drop-down button of the **Windows** platform. You can view the samples in any of the three options that get displayed.

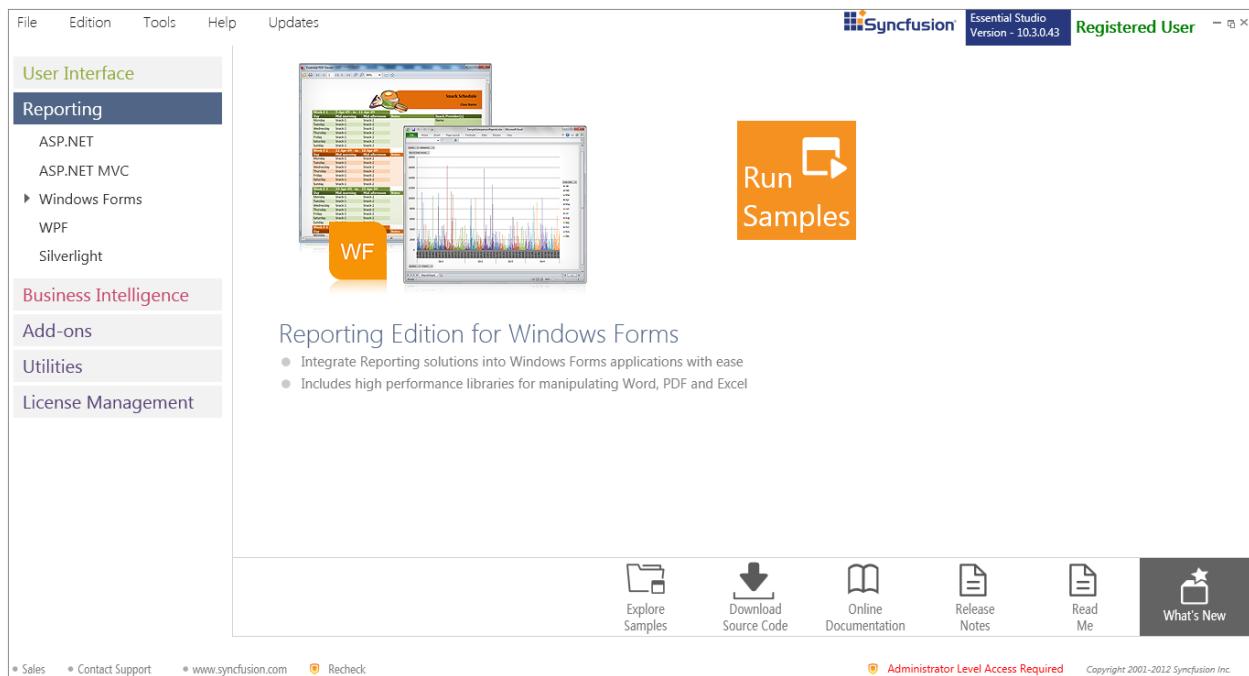


Figure 7: Windows Forms Sample browser

- Click **Run Samples** link. Essential Studio Reporting Edition Windows Forms sample browser is displayed.

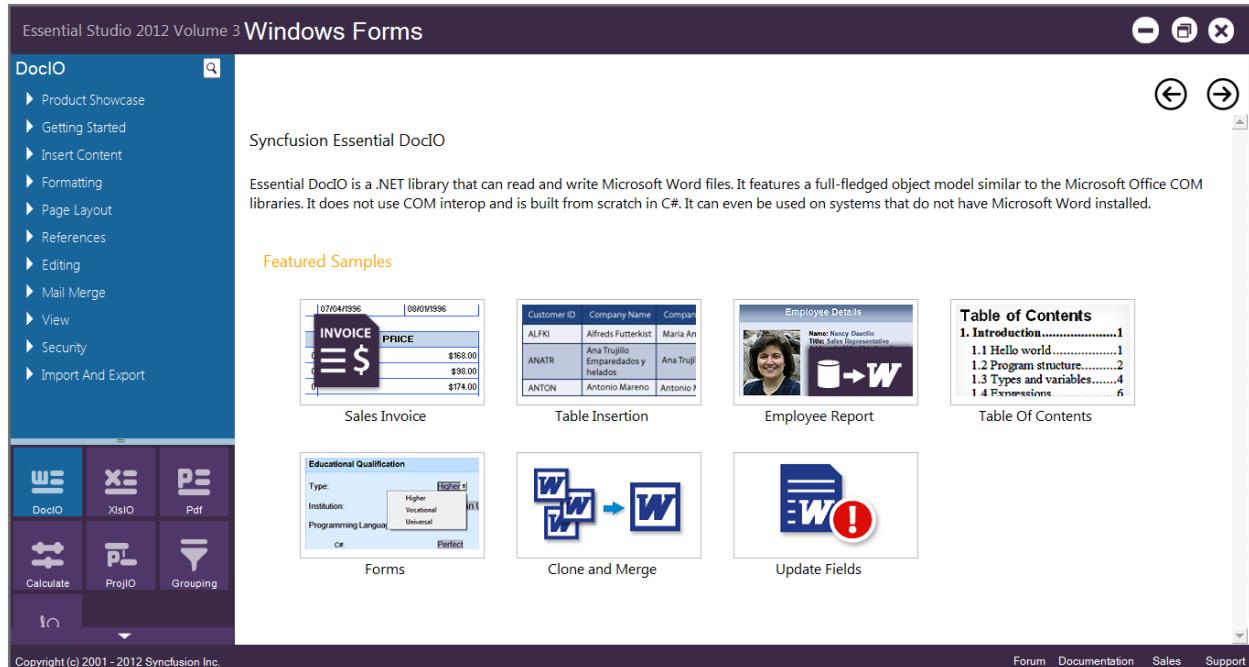


Figure 8: Windows Forms Sample browser

- Click **Pdf** form bottom-left pane.

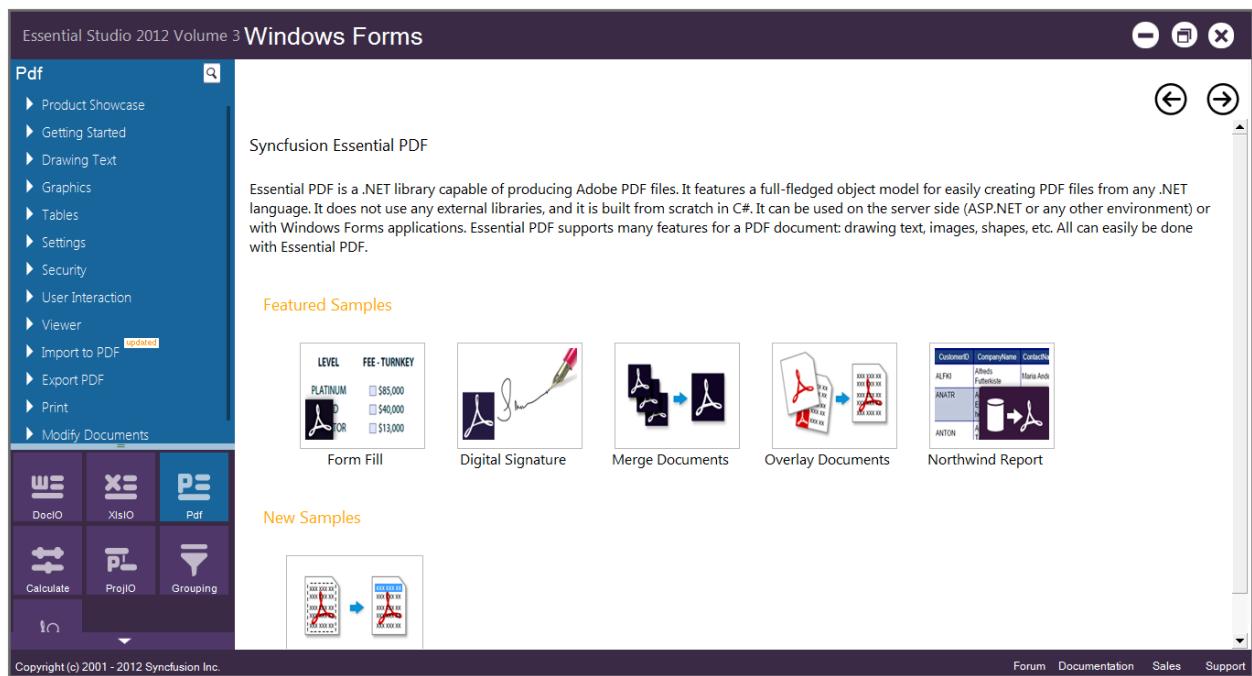


Figure 9: Windows Forms Sample Browser Displaying PDF Samples

A list of samples will be displayed on the left pane.

4. Select any sample and browse through the features.

WPF

1. Click the drop-down button of the **WPF** platform. You can view the samples in any of the three ways displayed.
2. Click **Run Samples** link. The Essential Studio - Reporting Edition sample browser is displayed.

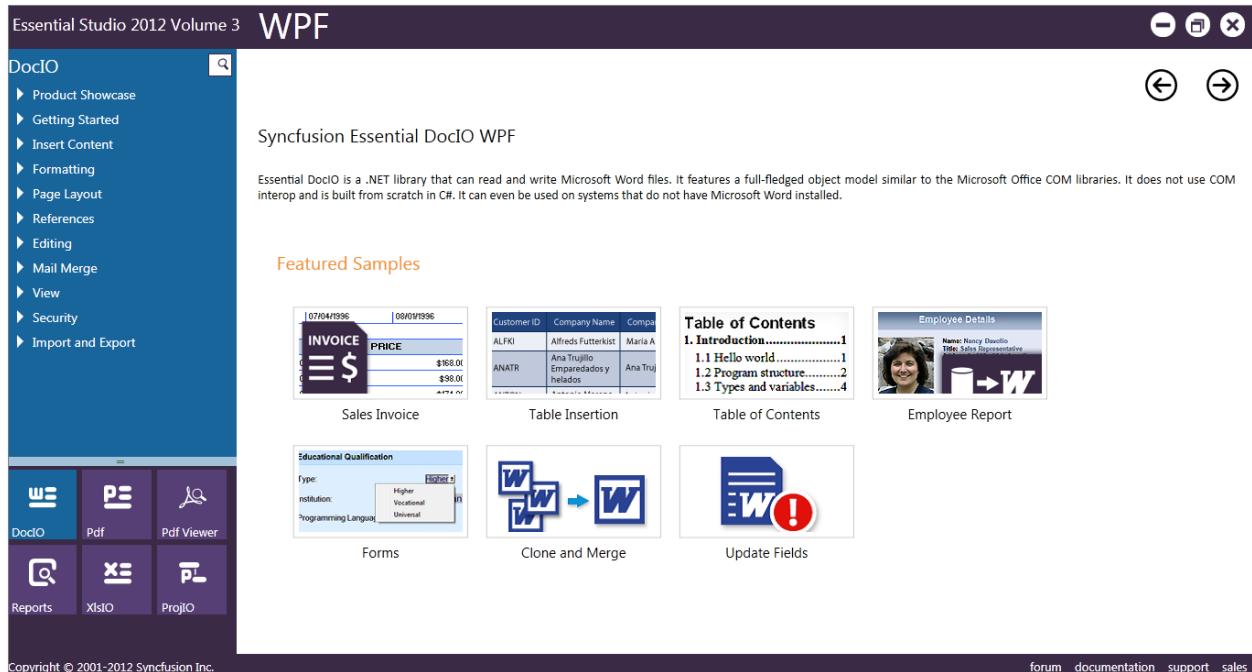


Figure 10: WPF Sample Browser

- Click **Pdf** from the bottom-left pane.

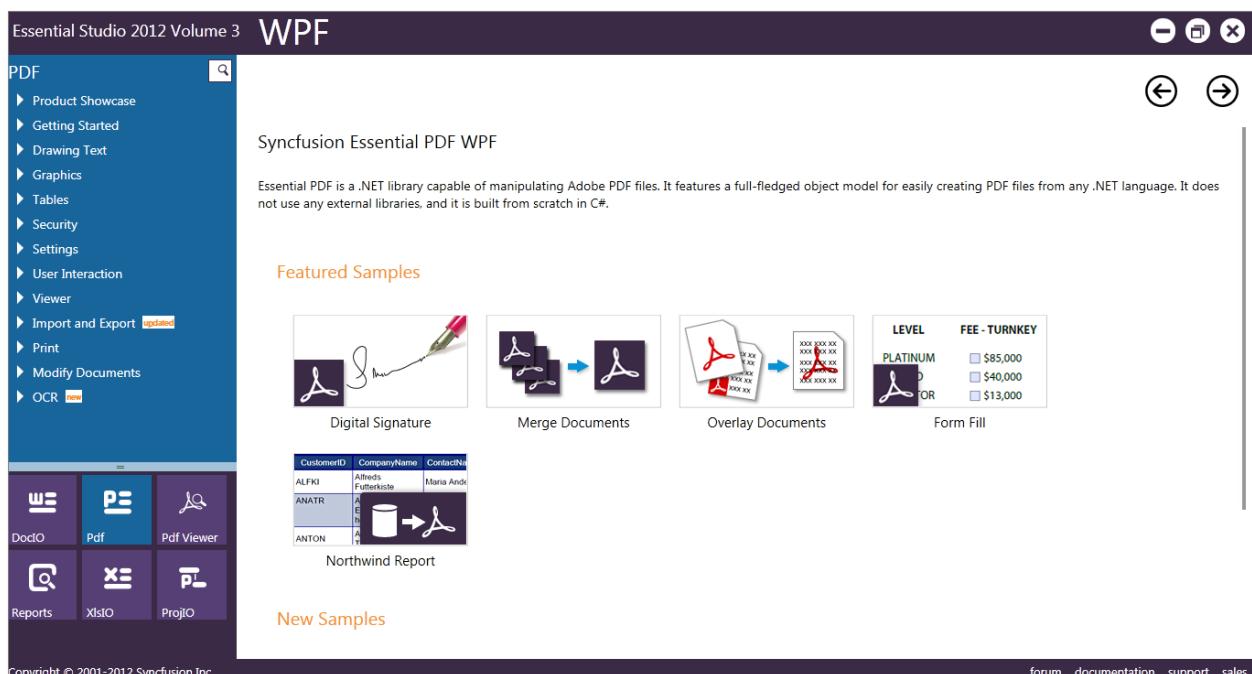


Figure 11: PDF Samples for WPF

- A list of samples will be displayed on the left pane.

- Select any sample and click **Run Sample** to browse through the sample features.

Silverlight

- Click the drop-down button of the **Silverlight** platform. The following options are displayed and you can view the samples in any of the three ways displayed.
- Click **Run Samples** link. The Essential Studio Reporting Silverlight Edition sample browser is displayed.



Figure 12: Silverlight Sample Browser



Note: By default, the samples of Essential DocIO are displayed.

- Select **Pdf** from the bottom-left pane.



Figure 13: Silverlight Sample Browser Showing PDF Samples

4. Select any sample and browse through the features.

Source Code Location

The source code for PDF under different platforms is available at the following default locations:

- Windows-[System Drive]:\Program Files\Syncfusion\Essential Studio\[Version Number]\Windows\PDF.Windows\Src
- ASP.NET-[System Drive]:\Program Files\Syncfusion\Essential Studio\[Version Number]\Web\Pdf.Web\Src
- WPF-[System Drive]:\Program Files\Syncfusion\Essential Studio\[Version Number]\WPF\Pdf.WPF\Src
- ASP.NET MVC-[System Drive]:\Program Files\Syncfusion\Essential Studio\[Version Number]\MVC\Pdf.MVC\Src

2.3 Deployment Requirements

This section discusses the deployment requirements for Essential PDF. It contains the following topics:

2.3.1 DLLs

The following assemblies need to be referenced in your application for the usage of Essential PDF.

Full Trust Mode

In full trust mode, add references to the following assemblies corresponding to the platform:

PDF (Full-Trust) – WPF, Windows Forms, ASP. NET, MVC

- Syncfusion.Core.dll
- Syncfusion.Compression.Base.dll
- Syncfusion.Pdf.Base.dll

PDF (Full - Trust) – Silverlight

- Syncfusion.Core.dll
- syncfusion.Compression.Silverlight.dll
- Syncfusion.Pdf.Silverlight.dll

Medium/Partial Trust Mode

In medium/partial trust mode, add references to the following assemblies corresponding to the platform:

ASP.NET – PDF (Partial-Trust) – Web

- Syncfusion.Core.dll
- Syncfusion.Compression.Base.dll
- Syncfusion.Pdf.Web.dll

ASP.NET MVC – PDF (Partial-Trust) – MVC

- Syncfusion.Core.dll
- Syncfusion.Compression.Base.dll
- Syncfusion.Pdf.MVC.dll

2.3.2 Web Application deployment

Web application by default is deployed in full trust mode. This section discusses the deployment in medium or partial trust scenarios.

Deploying in Medium Trust or Partial Trust Scenarios

There are two such scenarios in which Syncfusion assemblies might be deployed.

Example 1

If the Syncfusion Assemblies are in **GAC** (Global Assembly Cache), and the **Web Application** is running in **medium** trust, then the Syncfusion assemblies actually runs in full trust. Hence this scenario is fully supported and there are no additional steps necessary for deployment.

Example 2

Say, the Syncfusion Assemblies are present in the application's **bin** folder and the **Web Application** is running in **medium** trust, then the Syncfusion assemblies will run in medium trust.

Essential PDF allows to work in medium trust by using following assemblies.

- Syncfusion.Core.dll
- Syncfusion.Compression.Base.dll
- Syncfusion.Pdf.Web.dll

However following features are not currently supported under this scenario.

- Digital Signature (including) PdfSignatureField and PdfLoadedSignature Fields.
- PDF/A
- PdfHtmlTextElement
- Images
 - Metafile
 - RtfToImage
- Fonts
 - PdfTrueTypeFont
- RTL Support
- Html To PDF

3 Getting Started

This section comprises the following section:

3.1 Class Diagram

The following illustration shows the Class Diagram for Essential PDF.

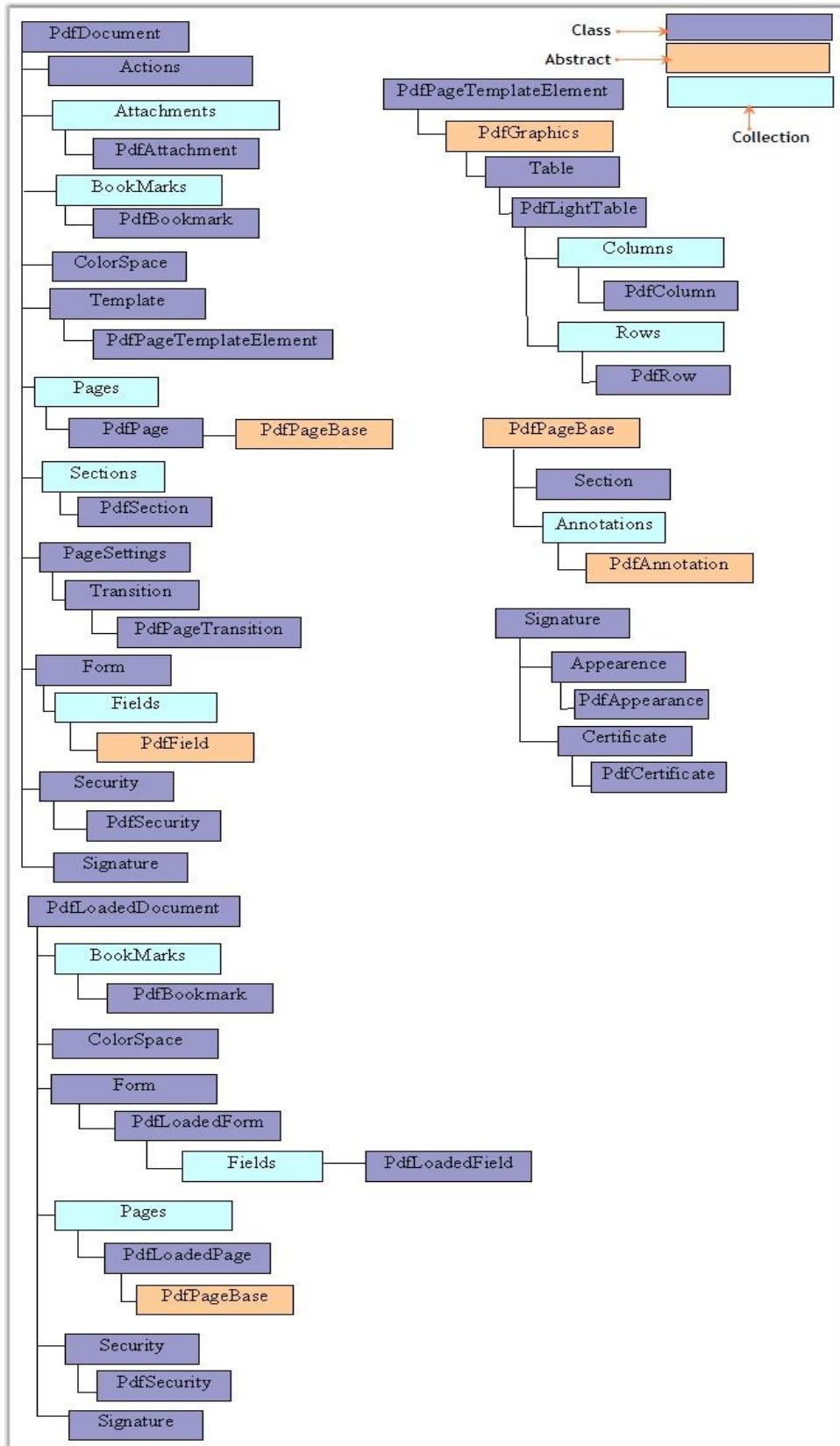


Figure 14: Class Diagram

3.2 Creating Platform Application

This section illustrates the step-by-step procedure to create the following platform applications.

Creating a Windows Application

1. Open Microsoft Visual Studio. Go to **File** menu and click **New Project**. In the **New Project** dialog, select **Windows Forms Application** template, name the project and click **OK**.

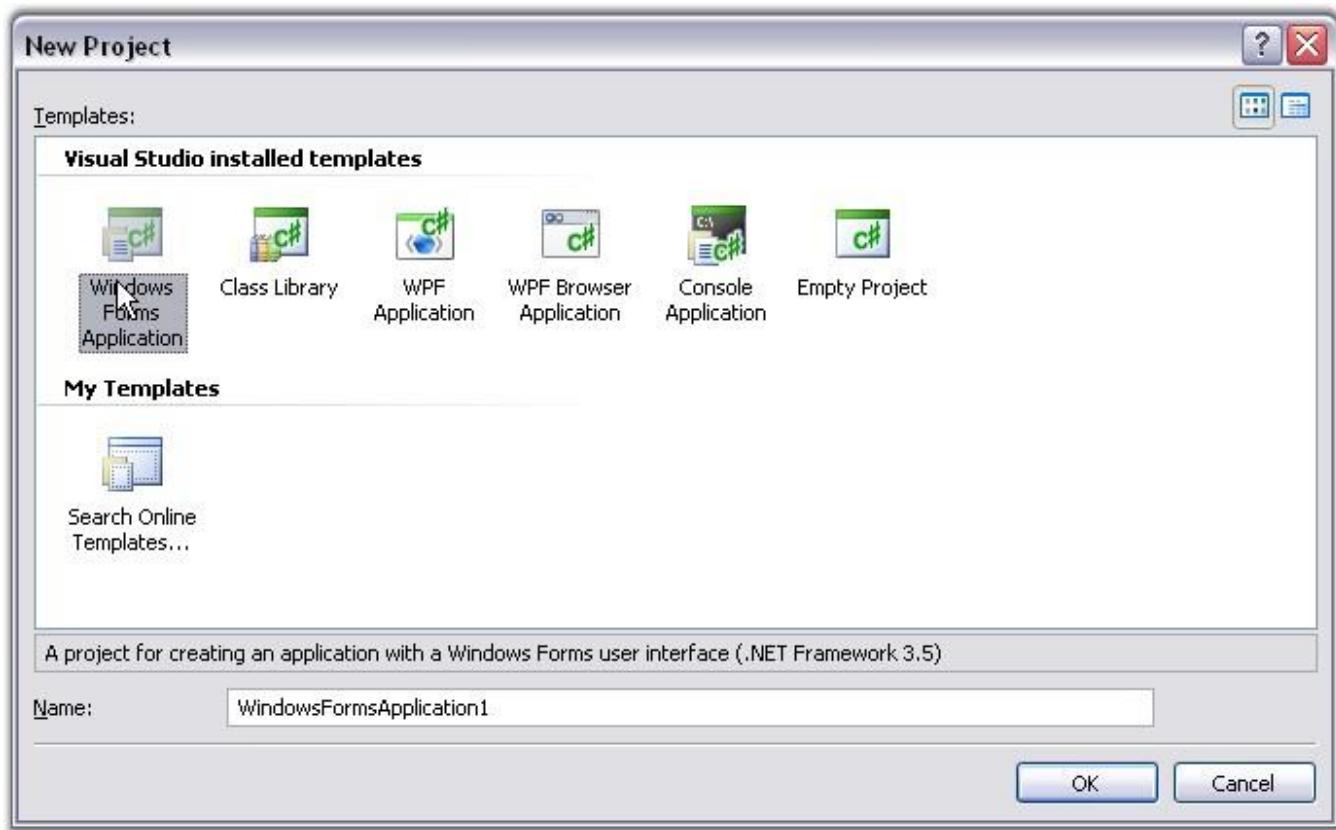


Figure 15: New Project dialog-Windows Forms Application

A windows application is created.

2. Now you need to deploy Essential PDF into this Windows application. Refer [Windows](#) topic for detailed info.

Creating an ASP.NET Application

To know how to deploy a web application, refer the ASP.NET Behind the scenes section in the Getting Started guide in the documentation.

1. Open Microsoft Visual Studio. Go to **File** menu and click **New Website**. In the **New Website** dialog, select **ASP.NET Web Site** template, name the website and click **OK**.

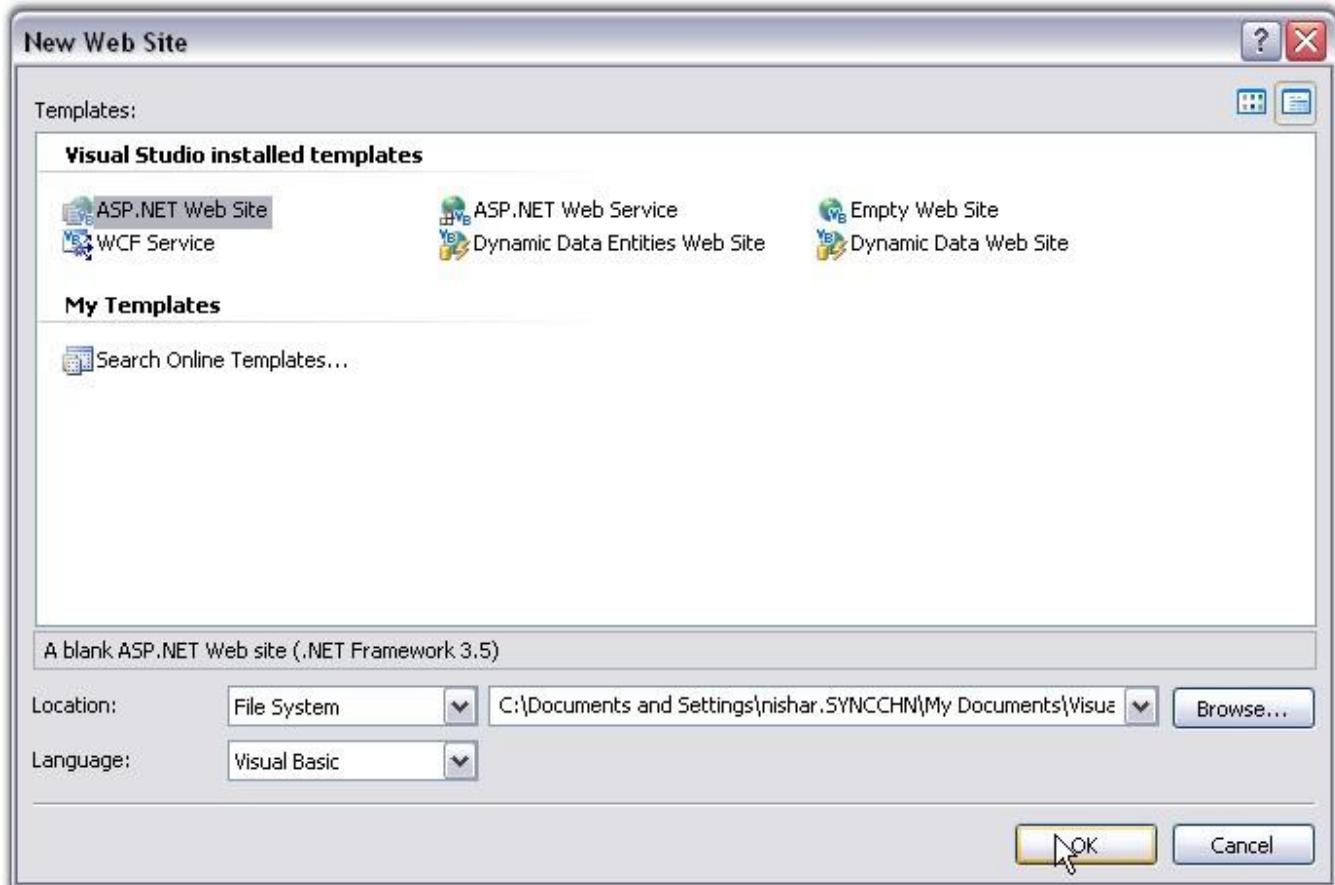


Figure 16: New Web Site dialog-ASP.NET Website

A web application is created.

2. Now you need to deploy Essential PDF into this ASP.NET application. Refer [ASP.NET](#) topic for detailed info.

Creating a WPF Application

1. Open Microsoft Visual Studio. Go to **File** menu and click **New Project**. In the **New Project** dialog, select **WPF Application** template, name the project and click **OK**.

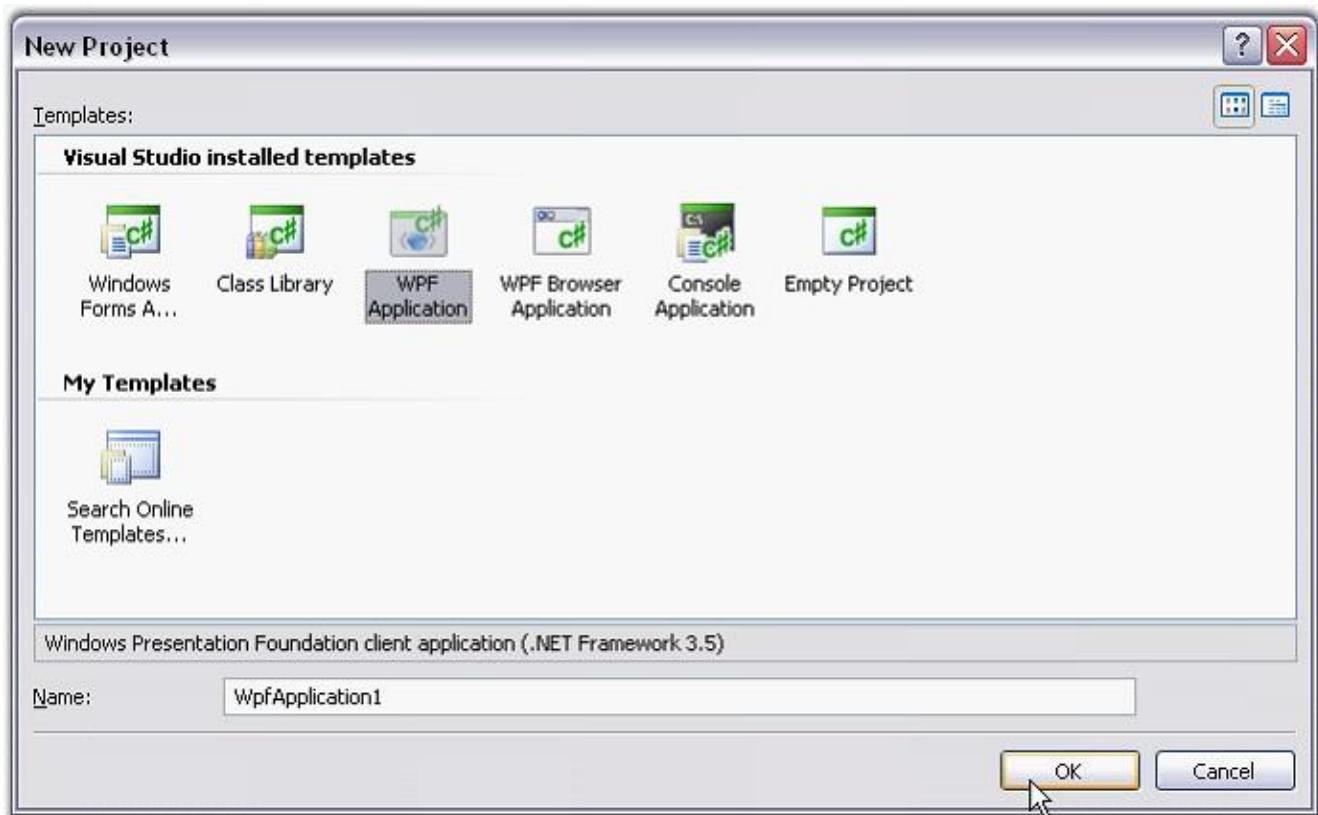


Figure 17: New Project dialog-WPF Application

A WPF application is created.

2. Now you need to deploy Essential PDF into this WPF application. Refer [WPF](#) topic for detailed info.

Creating a Silverlight Application

1. Open Microsoft Visual Studio. Go to **File** menu and click **New Project**. In the **New Project** dialog, select **Silverlight Application** template, name the project and click **OK**.

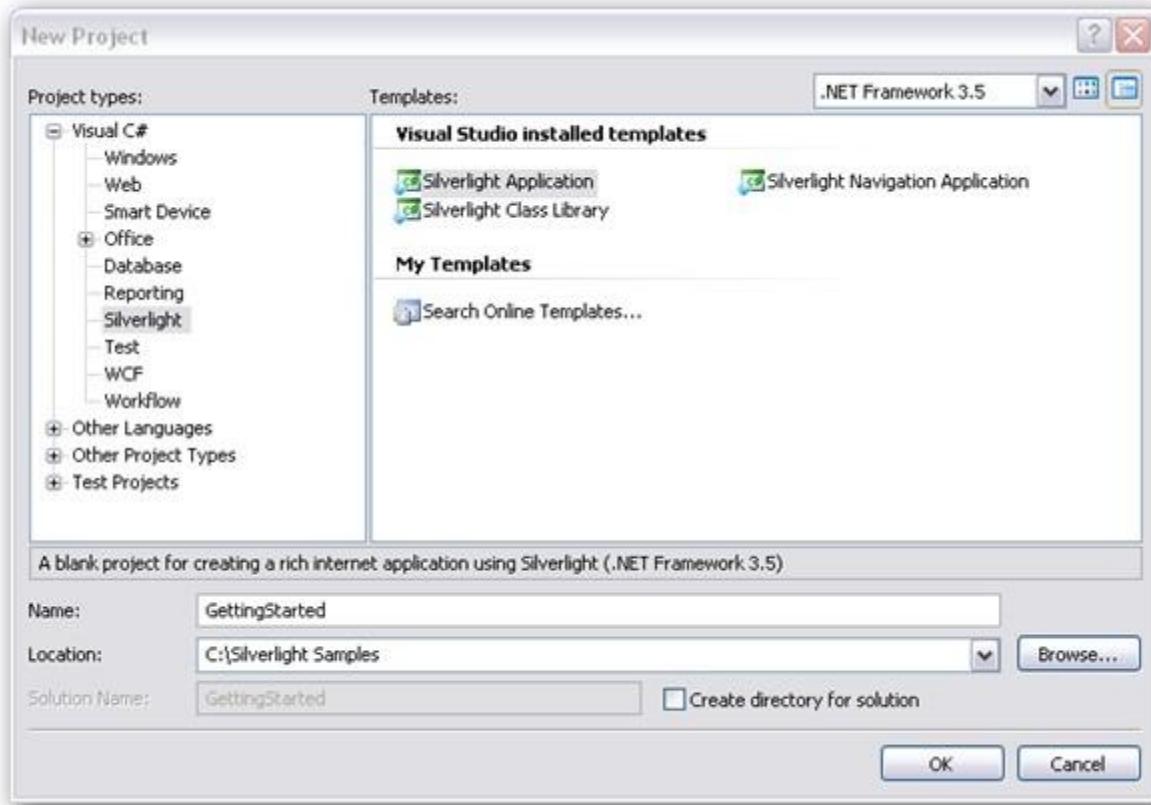


Figure 18: New Project dialog-Silverlight Application

A new Silverlight application is created.

2. Refer [Silverlight](#) topic to know how to deploy Essential PDF document with pages to the application.

Creating an ASP.NET MVC Application

Refer Grid MVC -> Getting Started -> Creating an MVC application topic to know how to create an MVC application.

To know how to add a PDF to this application, refer [ASP.NET MVC](#) topic.

For More Information refer:

[Deploying Essential PDF](#)

3.3 Deploying Essential PDF

We have now created a platform application in the previous topic ([Creating Platform Application](#)). This section will guide you to deploy Essential PDF in those applications under the following topics:

- **Windows**-Step-by-step procedure to deploy PDF in Windows applications
- **ASP.NET**-Step-by-step procedure to deploy PDF in web application
- **WPF**-Step-by-step procedure to deploy PDF in WPF application
- **Silverlight**-Step-by-step procedure to deploy PDF in Silverlight application
- **ASP.NET MVC**-Step-by-step procedure to deploy PDF in ASP.NET MVC application

3.3.1 Windows

Now, you have created a Windows application (refer to [Creating Platform Application](#)). This section covers the following:

- Deploying Essential PDF in a Windows application
- Create and add a PDF document with pages to the application

Deploying Essential PDF in a Windows Application

To deploy Essential PDF:

1. Go to **Solution Explorer** of the application you have created. Right-click the **Reference** folder and then click **Add References**.
2. Add the following assemblies as references in the application.
 - Syncfusion.Core.dll
 - Syncfusion.Compression.Base.dll
 - Syncfusion.Pdf.Base.dll

 For detailed documentation on Windows Application deployment, see:
http://www.syncfusion.com/support/user/uploads/DeployingWindowsApplication_bdaf76f7.pdf.

Essential PDF is deployed in the Windows application.

Create and add a PDF document with pages to the application

Following steps will guide you to create and add PDF document to this application:

1. Create a PDF document using the following code.

[C#]

```
// Create a new PDF document. This object represents the PDF document.
// This document has one page, by default. Additional pages have to be
// added.
PdfDocument pdfDoc = new PdfDocument();
```

[VB.NET]

```
' Create a new PDF document. This object represents the PDF document.
' This document has one page, by default. Additional pages have to be added.
Dim pdfDoc As Syncfusion.Pdf.PdfDocument = New Syncfusion.Pdf.PdfDocument()
```

i The PDF document represents the document that is created in the memory. It is only the memory representation of the PDF document that is written to the disk.

A new PDF document is created.

1. Add a new page to the created document.

[C#]

```
// Add a page to the document
PdfPage page = doc.Pages.Add();
```

[VB.NET]

```
'Add a page to the document
Dim firstPage As Syncfusion.Pdf.PdfPage = doc.Pages.Add()
```

A PDF page is added to the document (doc).

2. Write the string "Hello World" on the first page in the PDF document. This task is further subdivided into the following tasks.

- Create the font object to be used for writing the "Hello World" string. You can set the font size and style in the same statement.
- Write the text using the **DrawString** method of the Graphics object.

[C#]

```
// Use the predefined fonts to draw the text.
PdfFont font = new PdfStandardFont(PdfFontFamily.Helvetica, fontSize,
fontStyle);

// Draw the string at the specified coordinates.
firstPage.Graphics.DrawString("Hello World", pdfFont, PdfBrushes.Black, 0,
0);
```

[VB.NET]

```
' Use the predefined fonts to draw the text.
Dim font As Syncfusion.Pdf.Graphics.PdfFont = New
Syncfusion.Pdf.Graphics.PdfStandardFont(PdfFontFamily.Helvetica, fontSize,
FontStyle)

' Draw the string at the specified coordinates.
firstPage.Graphics.DrawString("Hello World", pdfFont, PdfBrushes.Black, 0,0)
```

The string "Hello World" is written to the document.

3. Then write the PDF document we have created to the disk. This is achieved by using the **Save** method of the PDF document.

[C#]

```
// Save the PDF document to disk.
pdfDoc.Save("Sample.pdf");
```

[VB.NET]

```
' Save the PDF document to disk.
pdfDoc.Save("Sample.pdf")
```

You can also save the changes to the loaded document as follows.

[C#]

```
// Save the document with same name
pdfDoc.Save();
```

[VB.NET]

```
' Save the document with same name  
pdfDoc.Save()
```

The created pdf document is saved to the disk using the above code.

4. Finally close the PDF Document so that the objects are released.

[C#]

```
// Release the common resources.  
pdfDoc.Close();  
  
//(or)  
  
//Releases document stream. This releases entire document  
PdfDoc.Close(true);
```

[VB.NET]

```
' Release the common resources.  
pdfDoc.Close()  
  
'(or)  
  
'Releases document stream. This releases entire document  
PdfDoc.Close(True)
```

PDF document will be closed after saving.

The sample pdf document created through the above procedure is shown below.

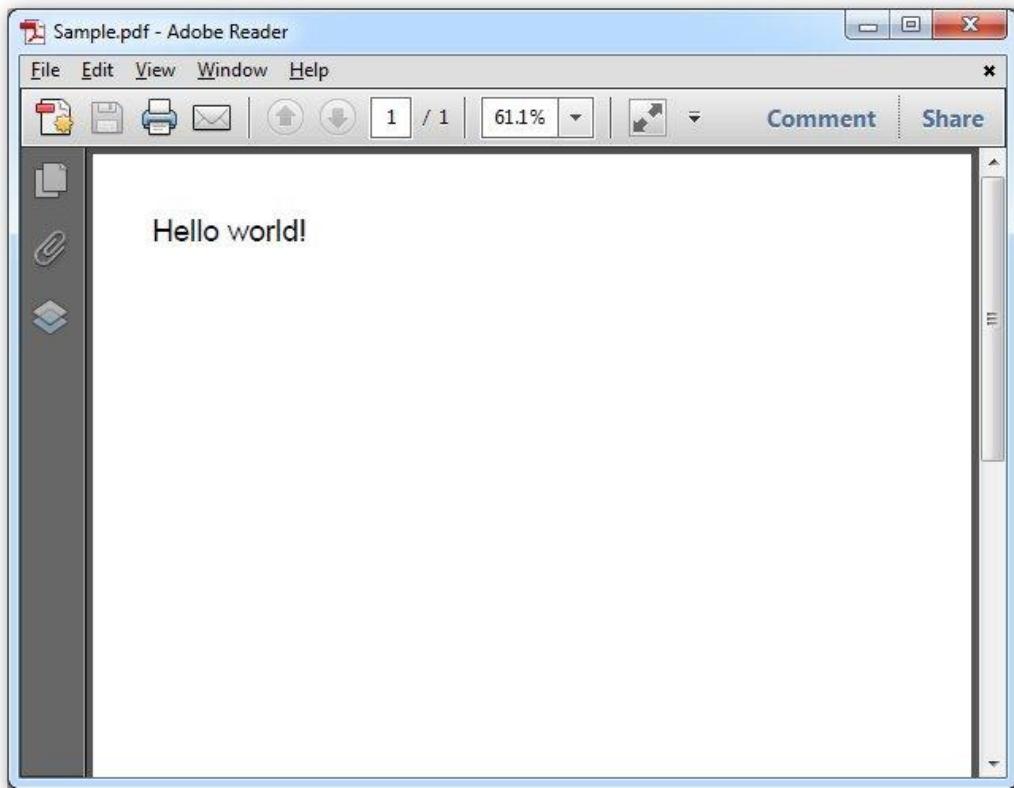


Figure 19: PDF Document with pages

A PDF document is created in the Windows application.

3.3.2 ASP.NET

Now, you have created a ASP.NET application (refer [Creating Platform Application](#)). This section covers the following:

- Deploying Essential PDF in an ASP.NET Application
- Create and add a PDF document with pages to the application

Deploying Essential PDF in an ASP.NET Application

This section provides information and instructions for deploying ASP.NET applications that use Essential PDF for ASP.NET. This is in addition to the section on Deploying Essential Studio for ASP.NET ([Common-->Deploying Essential Studio for ASP.NET](#)) in the Getting Started Guide. Essential PDF ships with .NET Framework 2.0 (Visual Studio 2005) version of the C# and VB.NET samples which are installed in 2.0 directories. During installation, application directories are created in IIS for each of the C# and VB.NET samples.

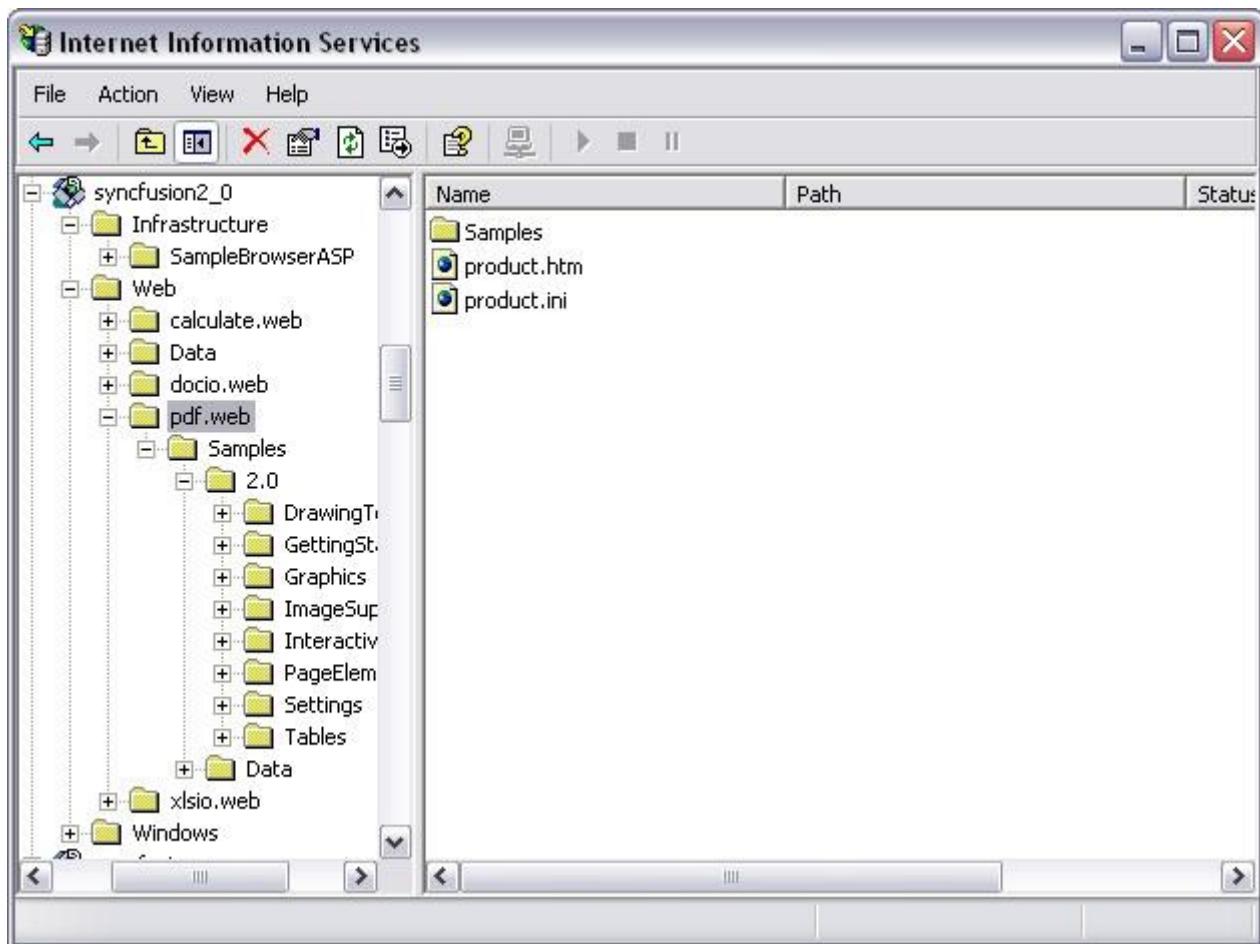


Figure 20: Sample Application Directories in IIS

The following steps will guide you to deploy Essential PDF in an ASP.NET application:

- Marking the Application directory**-The appropriate directory, usually where the aspx files are stored, must be marked as Application in IIS.
- Syncfusion Assemblies**-The Syncfusion assemblies need to be in the **bin** folder that is beside the aspx files.



Note: They can also be in the GAC, in which case, they should be referenced in Web.config file.

[ASPX]

```
<configuration>
  <system.web>
```

```

<compilation>
  <assemblies>
<add assembly="Syncfusion.Pdf.Base, Version=x.x.x.x, Culture=neutral,
PublicKeyToken=3D67ED1F87D44C89"/></assemblies>
  </compilation>
  ...
</system.web>
</configuration>

```



Note: The version number represented by x.x.x.x in the above code will vary depending on the version you are linking to.

3. **Data Files**-If you have .xml, .mdb, or other data files, ensure that they have sufficient security permission. Authenticated users should have full control over the files and the directories in order to give ASP.NET code, enough permissions to open the file during runtime.

Refer to the document in the following path, for step by step process of Syncfusion assemblies deployment in ASP.NET.

http://www.syncfusion.com/support/user/uploads/webdeployment_c883f681.pdf



Note: Application with Essential PDF needs following dependent assemblies for deployment.

- Syncfusion.Core.dll
- Syncfusion.Compression.Base.dll
- Syncfusion.Pdf.Base.dll

Essential PDF is now deployed in your ASP.NET application.

Create and add a PDF document with pages to the application

The following code snippet will guide you to create and add PDF document to this application:

[C#]

```

//Create a new document.
PdfDocument doc = new PdfDocument();

//Add a page
PdfPage page = doc.Pages.Add();

```

```
//Create Pdf graphics for the page
PdfGraphics g = page.Graphics;

//Create a solid brush
PdfBrush brush = new PdfSolidBrush(Color.Black);

float fontSize = 20f;

//Set the font
PdfFont font = new PdfStandardFont(PdfFontFamily.Helvetica, fontSize);

//Draw the text
g.DrawString("Hello world!", font, brush, new PointF(20, 20));

//Stream the output to the browser.
if (this.CheckBox1.Checked)
{
    doc.Save("Sample.pdf", Response, HttpReadType.Open);
}
else
{
    doc.Save("Sample.pdf", Response, HttpReadType.Save);
}
```

The sample pdf document created through the above procedure is shown below.

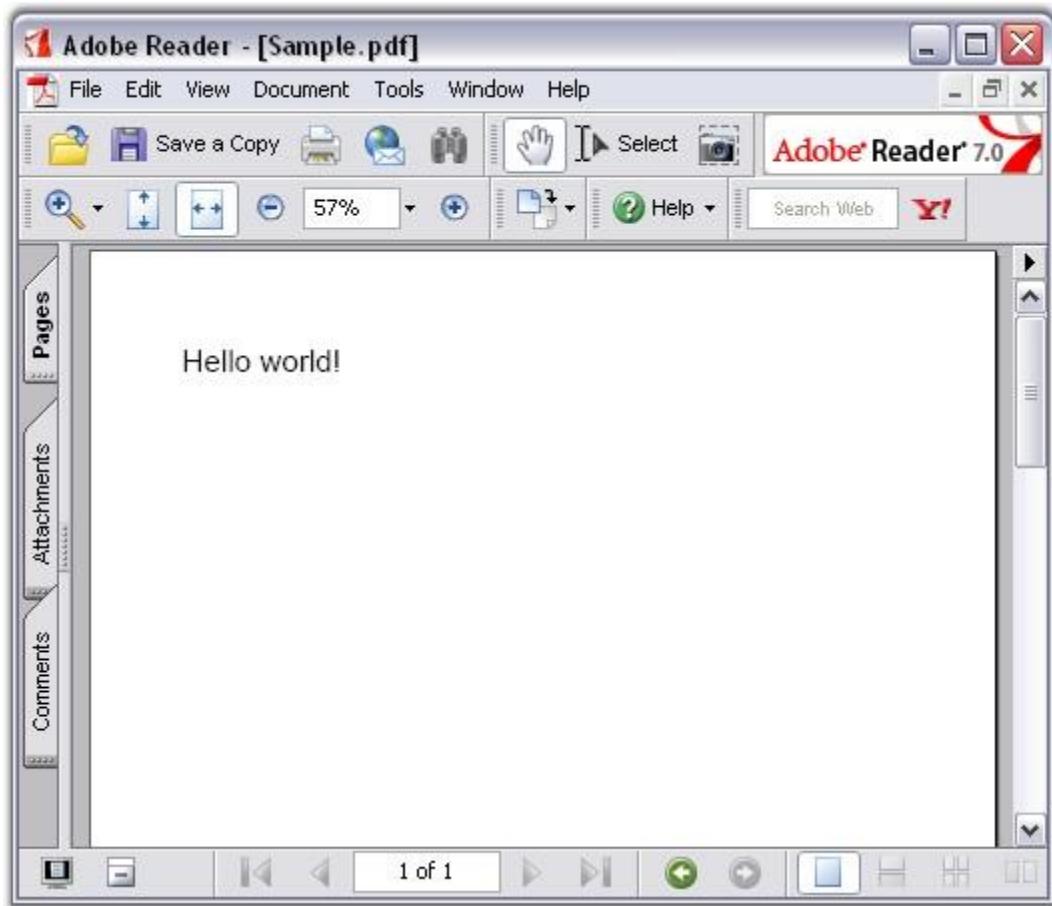


Figure 21: PDF Document with pages

A PDF document is created in the ASP.NET application.

3.3.3 WPF

Now, you have created a WPF application (refer to [Creating Platform Application](#)). This section covers the following:

- Deploying Essential PDF in a WPF Application
- Create and add a PDF document with pages to the application

Deploying Essential PDF in a WPF Application

To deploy Essential PDF:

1. Go to **Solution Explorer** of the application you have created. Right-click the **Reference** folder and then click **Add References**.
2. Add the following assemblies as references in the application:
 - Syncfusion.Core.dll
 - Syncfusion.Compression.Base.dll
 - Syncfusion.Pdf.Base.dll

 For detailed documentation on Windows Application deployment, see:
http://www.syncfusion.com/support/user/uploads/DeployingWindowsApplication_bdaf76f7.pdf.

Essential PDF is deployed in the Windows application:

Create and add a PDF document with pages to the application

The following steps will guide you to create and add a PDF document to this application:

3. Create a PDF document using the following code.

 The PDF document represents the document that is created in the memory. It is only the memory representation of the PDF document that is written to the disk.

[C#]

```
// Create a new PDF document. This object represents the PDF document.
// This document has one page, by default. Additional pages have to be
added.
PdfDocument pdfDoc = new PdfDocument();
```

[VB.NET]

```
' Create a new PDF document. This object represents the PDF document.
' This document has one page, by default. Additional pages have to be added.
Dim pdfDoc As Syncfusion.Pdf.PdfDocument = New Syncfusion.Pdf.PdfDocument()
```

A new PDF document is created.

Add a new page to the created document.

[C#]

```
// Add a page to the document
PdfPage page = doc.Pages.Add();
```

[VB .NET]

```
'Add a page to the document
Dim firstPage As Syncfusion.Pdf.PdfPage = doc.Pages.Add()
```

A PDF page is added to the document (doc).

4. Write the string "Hello World" on the first page in the PDF document. This task is further subdivided into the following tasks.

- Create the font object to be used for writing the "Hello World" string. You can set the font size and style in the same statement.
- Write the text using the **DrawString** method of the Graphics object.

[C#]

```
// Use the predefined fonts to draw the text.
PdfFont font = new PdfStandardFont(PdfFontFamily.Helvetica, fontSize,
fontStyle);

// Draw the string at the specified coordinates.
firstPage.Graphics.DrawString("Hello World", pdfFont, PdfBrushes.Black, 0,
0);
```

[VB .NET]

```
' Use the predefined fonts to draw the text.
Dim font As Syncfusion.Pdf.Graphics.PdfFont = New
Syncfusion.Pdf.Graphics.PdfStandardFont(PdfFontFamily.Helvetica, fontSize,
FontStyle)

' Draw the string at the specified coordinates.
firstPage.Graphics.DrawString("Hello World", pdfFont, PdfBrushes.Black, 0,0)
```

The string "Hello World" is written to the document.

5. Then write the PDF document we have created to the disk. This can be done by using the **Save** method of the PDF document.

[C#]

```
// Save the PDF document to disk.
```

```
pdfDoc.Save("Sample.pdf");
```

[VB.NET]

```
' Save the PDF document to disk.  
pdfDoc.Save("Sample.pdf")
```

You can also save the changes to the loaded document as follows.

[C#]

```
// Save the document with same name  
pdfDoc.Save();
```

[VB.NET]

```
' Save the document with same name  
pdfDoc.Save()
```

The created pdf document is saved to the disk using the above code.

6. Finally close the PDF Document using the following code, so that the objects are released.

[C#]

```
// Release the common resources.  
pdfDoc.Close();  
  
//(or)  
  
//Releases document stream. This releases entire document  
PdfDoc.Close(true);
```

[VB.NET]

```
' Release the common resources.  
pdfDoc.Close()  
  
'(or)  
  
'Releases document stream. This releases entire document  
PdfDoc.Close(True)
```

PDF document will be closed after saving.

The sample pdf document created through the above procedure is shown below.

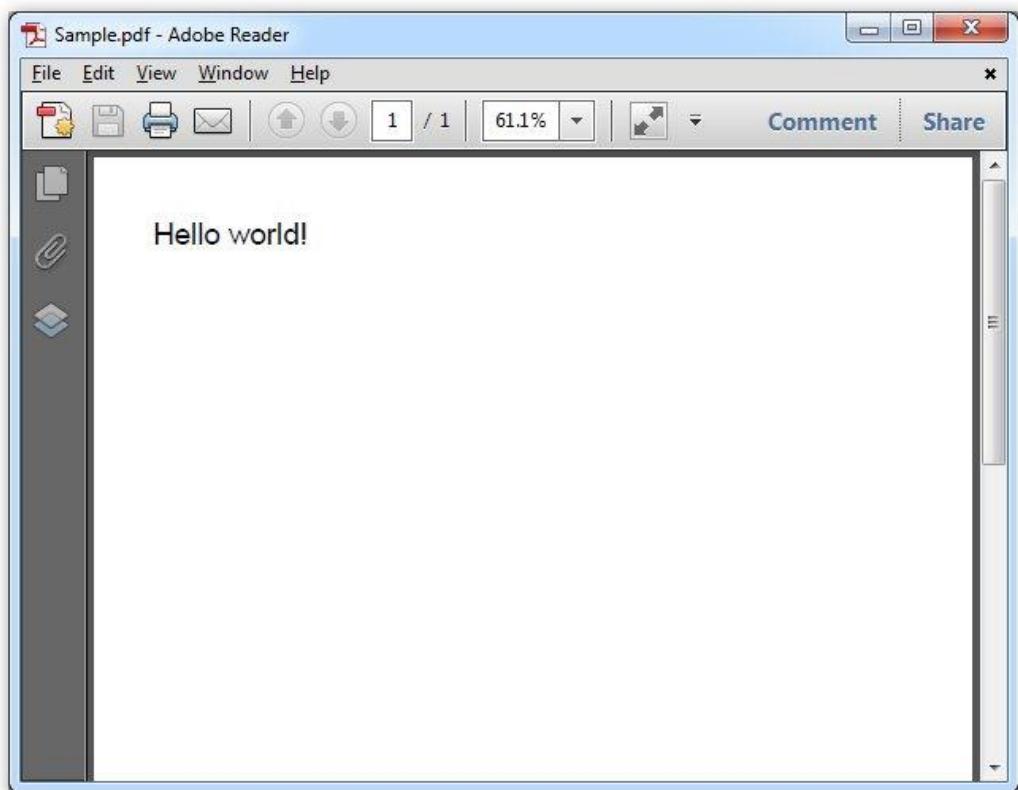


Figure 22: PDF Document with pages

A PDF document is created in the WPF application.

3.3.4 Silverlight

Now, you have created a Silverlight application (refer [Creating Platform Application](#)). This section covers the following:

- Deploying Essential PDF in a Silverlight Application
- Create and add a PDF document with pages to the application

Deploying Essential PDF in a Silverlight Application

The following steps will guide you to deploy Essential PDF:

1. Open the MainPage.xaml of the application in the designer.
2. Add the following assemblies as references in the application:
 - Syncfusion.Compression.Silverlight.dll
 - Syncfusion.Pdf.Silverlight.dll

Essential PDF is now deployed in your Silverlight application.

Create and add a PDF document with pages to the application

Following steps will guide you to create and add a PDF document to this application:

1. Create a PDF document using the following code.

i The PDF document represents the document that is created in the memory. It is only the memory representation of the PDF document that is written to the disk.

[C#]

```
// Create a new PDF document. This object represents the PDF document.  
// This document has one page, by default. Additional pages have to be  
added.  
PdfDocument pdfDoc = new PdfDocument();
```

[VB .NET]

```
' Create a new PDF document. This object represents the PDF document.  
' This document has one page, by default. Additional pages have to be added.  
Dim pdfDoc As Syncfusion.Pdf.PdfDocument = New Syncfusion.Pdf.PdfDocument()
```

A new PDF document is created.

2. Add a new page to the created document.

[C#]

```
// Add a page to the document  
PdfPage page = doc.Pages.Add();
```

[VB .NET]

```
'Add a page to the document  
Dim firstPage As Syncfusion.Pdf.PdfPage = doc.Pages.Add()
```

A PDF page is added to the document (doc).

3. Write the string "Hello World" on the first page in the PDF document. This task is further subdivided into the following tasks.

- Create the font object to be used for writing the "Hello World" string. You can set the font size and style in the same statement.
- Write the text using the **DrawString** method of the Graphics object.

[C#]

```
// Use the predefined fonts to draw the text.
PdfFont pdfFont = new PdfStandardFont(PdfFontFamily.Helvetica, fontSize,
fontStyle);

// Draw the string at the specified coordinates.
firstPage.Graphics.DrawString("Hello World", pdfFont, PdfBrushes.Black, 0,
0);
```

[VB.NET]

```
' Use the predefined fonts to draw the text.
Dim font As Syncfusion.Pdf.Graphics.PdfFont = New
Syncfusion.Pdf.Graphics.PdfStandardFont(PdfFontFamily.Helvetica, fontSize,
FontStyle)

' Draw the string at the specified coordinates.
firstPage.Graphics.DrawString("Hello World", pdfFont, PdfBrushes.Black, 0, 0)
```

The string "Hello World" is written to the document.

4. User can save the generated PDF document to their specified locations with the help of the **SaveFileDialog** class by streaming the generated PDF document.

[C#]

```
// Create instance for SaveFileDialog
SaveFileDialog sfd = new SaveFileDialog()
{
    DefaultExt = "pdf",
    Filter = "Text files (*.pdf)|*.pdf|All files (*.*)|*.*",
    FilterIndex = 1
};
```

```

        if (sfd.ShowDialog() == true)
    {
        using (Stream stream = sfd.OpenFile())
        {
            // Save the PDF document in the user specific location
            pdfDoc.Save(stream2);
        }
    };
}

```

[VB.NET]

```

'Create instance for SaveFileDialog
Dim sfd As SaveFileDialog = New SaveFileDialog()
    "Text files (*.pdf)|*.pdf|All files (*.*)|*.*", FilterIndex = 1
    "pdf", Filter = "Text files (*.pdf)|*.pdf|All files (*.*)|*.*",
FilterIndex
    DefaultExt = "pdf", Filter

If sfd.ShowDialog() = True Then
    Using stream As Stream = sfd.OpenFile()
        ' Save the PDF document in the user specific location
        pdfDoc.Save(stream2)
    End Using
End If

```

The created pdf document is saved to the disk using the above code.

5. Finally close the PDF Document using the following code, so that the objects are released.

[C#]

```

// Release the common resources.
pdfDoc.Close();

//(or)

//Releases document stream. This releases entire document
PdfDoc.Close(true);

```

[VB.NET]

```

' Release the common resources.
pdfDoc.Close()

```

```
'(or)  
  
'Releases document stream. This releases entire document  
PdfDoc.Close(True)
```

PDF document will be closed after saving.

The sample pdf document created through the above procedure is shown below.

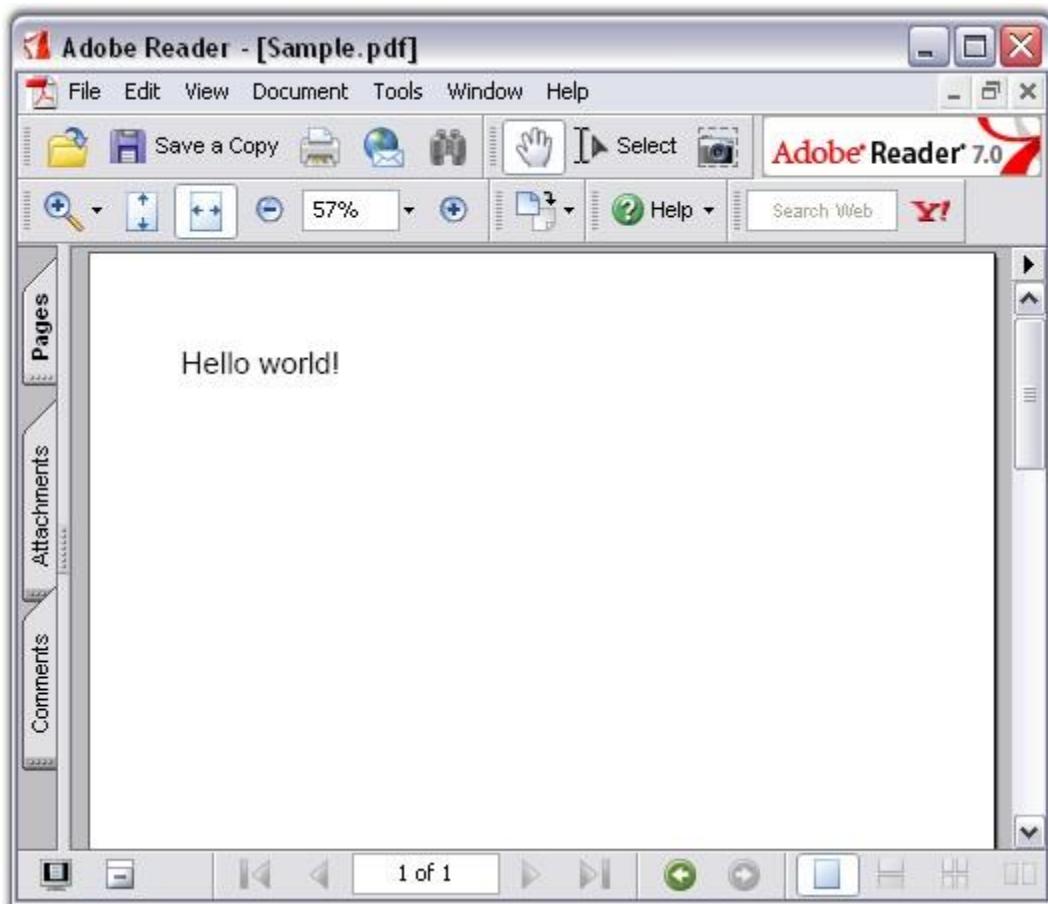


Figure 23: PDF Document with pages

A PDF document is created in the Silverlight application.

3.3.5 ASP.NET MVC

Now, you have created a ASP.NET MVC application (refer [Creating Platform Application](#)). This section covers the following:

- Deploying Essential PDF in an ASP.NET MVC Application
- Create and add a PDF document with pages to the application

Deploying Essential PDF in an ASP.NET MVC Application

The following steps will guide you to deploy Essential PDF:

1. Add the Syncfusion.Pdf.Base assembly into the **Reference** folder to deploy Essential PDF to the application.
2. Add the following assembly reference under **<compilation>** tag of Web.config file.

[ASPX]

```
<compilation debug="true">
  <assemblies>
    ...
    <add assembly="Syncfusion.Pdf.Base, Version=x.x.x.x, Culture=neutral,
      PublicKeyToken=3d67ed1f87d44c89"/>
    ...
  </assemblies>
</compilation>
```

3. Add the following under **<namespaces>** tag of Web.config file.

[ASPX]

```
<namespaces>
  ....
    <add namespace="Syncfusion.Pdf"/>
</namespaces>
```

Essential PDF is now deployed in your ASP.NET MVC application.

Create and add a PDF document with pages to the application

The following steps illustrate how to create a Word document in an MVC application.

Step 1: View

Add a **Button** to the aspx page in View as follows.

[ASPX]

```
<%Html.BeginForm("HelloWorld", "GettingStarted", FormMethod.Post);%>
<input type="submit" value="Create PDF" />
<% Html.EndForm();%>
```

Step 2: Create Custom ActionResult

In an MVC application, a controller action returns an action result. In particular, it returns an action that derives from the base ActionResult class.

But, if you want to return some other type of content to a browser, such as an image, a PDF file, or a Microsoft Excel document, you need to create your own action result. The following code example illustrates how to create a custom action result.

[C#]

```
public class PdfResult : ActionResult
{
    private string m_filename;
    private PdfDocument m_pdfDocument;
    private PdfLoadedDocument m_pdfLoadedDocument;
    private HttpResponse m_response;
    private HttpReadType m_readType;

    public string FileName
    {
        get
        {
            return m_filename;
        }
        set
        {
            m_filename = value;
        }
    }
    public PdfDocument PdfDoc
    {
```

```
get
{
    if (m_pdfDocument != null)
        return m_pdfDocument;
    return null;
}
}

public PdfLoadedDocument pdfLoadedDoc
{
    get
    {
        if (m_pdfLoadedDocument != null)
            return m_pdfLoadedDocument;
        return null;
    }
}

public HttpResponseMessage Response
{
    get
    {
        return m_response;
    }
}

public HttpReadType ReadType
{
    get
    {
        return m_readType;
    }
    set
    {
        m_readType = value;
    }
}

public PdfResult(PdfDocument pdfDocument, string filename, HttpResponseMessage response, HttpReadType type)
{
    this.m_pdfDocument = pdfDocument;
    this.m_pdfLoadedDocument = null;
    this.FileName = filename;
    this.m_response = response;
    this.ReadType = type;
}

public PdfResult(PdfLoadedDocument pdfLoadedDocument, string filename,
```

```

        HttpResponse responce, HttpReadType type)
    {
        this.m_pdfDocument = null;
        this.m_pdfLoadedDocument = pdfLoadedDocument;
        this.FileName = filename;
        this.m_responce = responce;
        this.ReadType = type;
    }

    public override void ExecuteResult(ControllerContext context)
    {
        if (context == null)
            return;
        if (pdfLoadedDoc != null)
            this.pdfLoadedDoc.Save(FileName, Response, ReadType);
        if (PdfDoc != null)
            this.PdfDoc.Save(FileName, Response, ReadType);
    }
}

```

Every action result must inherit from the base **ActionResult** class. The base ActionResult class has one method that you must implement: the **ExecuteResult** method. The ExecuteResult method is called to generate the content created by the action result. This method streams the output Word document file to the browser.

A controller action cannot return an action result directly. For example, if you want to return a view from a controller action, you don't return a ViewResult. Instead, you call the **View** method. The View method instantiates a new ViewResult and returns the new ViewResult to the browser.

An extension method named **ExportAsActionResult** has to be created to add the Controller class. The ExportAsActionResult method returns a PdfResult.

[ASPx]

```

<%Html.BeginForm("HelloWorld", "GettingStarted", FormMethod.Post);
{ %>


|                                                                                                                                                               |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <input &gt;="" <="" browser="" document="" inside="" name="Browser" open="" style="font-size:12px; font:Trebuchet MS" td="" type="checkbox" value="Browser"/> |
| <input &gt;="" <="" style="margin-right: 3px; height:27px; font-size:12px; font:Trebuchet MS" td="" type="submit" value="Create PDF"/>                        |


```

```

</td>
</tr>
</table>
<% Html.EndForm();
} %>

```

Step 3: Controller

Add the following code in the Controller's action result.

```

[C#]

public static PdfResult ExportAsActionResult(this PdfDocument PdfDoc, string
filename, HttpResponse response, HttpReadType type)
{
    return new PdfResult(PdfDoc, filename, response, type);
}

```

Step 4: Run the Application

Now try running the project by selecting **Start Debugging** option from the **Debug** menu. The following dialog box appears. It provides options to download and launch the generated file.

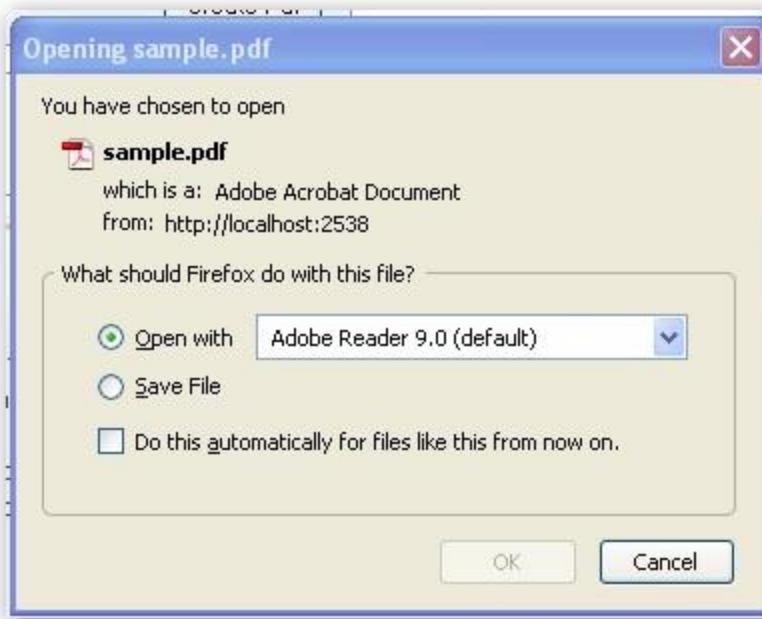


Figure 24: opening Sample.pdf

A PDF document is created in the ASP.NET MVC application.

3.4 PDF Version Compatibility

The PDF files generated by Essential PDF can be viewed using Adobe Acrobat Reader 7.x or later versions. Other conforming Readers that strictly follow Adobe's PDF specification 1.4 or later also can be used.



Note: *Adobe Reader is free software; hence the most recent version can be downloaded easily.*

Essential PDF allows you control the structure of PDF files using the FileStructure property of the document. The following settings are required to control the structure:

- **Version** (property)-Specifies the version of the PDF document.



Note: *This property marks the file by the specified version only. It does not control the compatibility of the features used in the file.*

- **CrossReferenceType** (property)-Specifies the type of the cross-reference in the file. This property allows you to present a cross-reference with a standard format, in a cross-reference table. The cross-references can also be presented as a cross-reference stream using the CrossReferenceStream property. This format is supported by the PDF 1.5 and higher versions. The files using this format are more compact, especially when compression is used. However, generating such a file takes more time.



Note: *It is suggested that you disable this option, if file generation speed is a critical factor.*

Changing PDF Version to Enable Acrobat 5.0 Compatibility

The following code example illustrates how to change the PDF version to 1.4, which will enable reading in Acrobat 5.0.

[C#]

```
doc.FileStructure.CrossReferenceType =  
    PdfCrossReferenceType.CrossReferenceTable;  
  
doc.FileStructure.Version = PdfVersion.Version1_4;
```

You have successfully changed the PDF version to 1.4.

3.5 PDF Features

Supported and non-supported elements of Essential PDF for Windows, ASP.NET, WPF, ASP.NET MVC, Silverlight, and Windows Store apps are listed in the following table.

Features	Windows Forms/WPF	ASP .NET / ASP.NET MVC (Medium Trust)	Silverlight	Windows Store Apps
Drawing Text	Yes	Yes	Yes	Yes
Text Formatting	Yes	Yes	Yes	Yes
Multilingual Support	Yes	No	No	No
Drawing RTL text	Yes	No	No	No
Text Extraction	Yes	Yes	No	No
Unicode	Yes	No	No	No
Pagination	Yes	Yes	Yes	Yes
Graphics				
Pen and Brush	Yes	Yes	Yes	Yes
Layers	Yes	Yes	Yes	Yes
Transparent Graphics	Yes	Yes	No	No
Color Spaces	Yes	Yes	Yes	Yes
Image Extraction	Yes	Yes	No	No
Enhanced Printing Support	Yes	Yes	Yes	Yes
Barcode	Yes	Yes	No	No
Document-level Operations				
Merge Documents	Yes	Yes	Yes	Yes
Split Document	Yes	Yes	Yes	Yes
Overlay Documents	Yes	Yes	Yes	Yes
Import Pages	Yes	Yes	Yes	Yes

Features	Windows Forms/WPF	ASP .NET / ASP.NET MVC (Medium Trust)	Silverlight	Windows Store Apps
Stamp	Yes	Yes	Yes	Yes
Booklet	Yes	Yes	Yes	Yes
Document Settings				
Custom Metadata	Yes	Yes	No	No
Document Properties	Yes	Yes	Yes	Yes
Page Orientation	Yes	Yes	Yes	Yes
Page Sizes	Yes	Yes	Yes	Yes
Viewer Preferences	Yes	Yes	Yes	Yes
Forms				
Fields	Yes	Yes	Yes	Yes
Form Filling	Yes	Yes	Yes	Yes
Flatten	Yes	Yes	Yes	Yes
Import Form Data	Yes	Yes	Yes	Yes
Form Export	Yes	Yes	Yes	Yes
Document Conversion				
TIFF to PDF	Yes	No	No	No
HTML to PDF	Yes	No	No	No
RTF To PDF	Yes	No	No	No
DOC To PDF	Yes	No	No	No
Excel To PDF	Yes	No	No	No
PDF OCR	Yes	No	No	No
PDF Standards				
PDF/ A-1b Compliance	Yes	Yes	Yes	Yes

Features	Windows Forms/WPF	ASP .NET / ASP.NET MVC (Medium Trust)	Silverlight	Windows Store Apps
PDF/x1a: 2001 Compliance	Yes	Yes	Yes	Yes
Fonts				
Standard Fonts	Yes	Yes	Yes	Yes
CJK Fonts	Yes	Yes	Yes	Yes
True Type Fonts	Yes	No	No	No
Unicode True Type	Yes	No	No	No
Images				
Scalar Images	Yes	Yes	Yes*	Yes*
Soft Mask	Yes	Yes	Yes	Yes
Vector Images	Yes	No	No	No
Watermarks	Yes	Yes	Yes	Yes
Tables				
ADO.Net Tables Support	Yes	Yes	No	No
Cell / Row / Column Formatting	Yes	Yes	Yes	Yes
Header	Yes	Yes	Yes	Yes
Pagination	Yes	Yes	Yes	Yes
Borders	Yes	Yes	Yes	Yes
RowSpan** and ColumnSpan	Yes	Yes	Yes	Yes
Nested**	Yes	Yes	Yes	Yes
Cell Padding and Spacing	Yes	Yes	Yes	Yes
Page Level Operations				
Headers and Footers	Yes	Yes	Yes	Yes
Page Label	Yes	Yes	Yes	Yes
Automatic Fields	Yes	Yes	Yes	Yes

Interactive Elements				
3D-Annotation	Yes	Yes	Yes	Yes
Action	Yes	Yes	Yes	Yes
Attachment	Yes	Yes	Yes	Yes
Bookmark	Yes	Yes	Yes	Yes
Hyperlink	Yes	Yes	Yes	Yes
Security				
Digital Signature	Yes	No	No	No
Encryption and Decryption	Yes	Yes	No	Yes



Note:

***Only .Jpeg format images are supported for Silverlight version.**

****Supported only in PdfGrid class**

4 Concepts and Features

The concepts and features of Essential PDF are grouped under the following sections.



Note: The concepts, features and APIs are common to Windows, Web and WPF except for the way the documents are streamed to the browser.

4.1 PDF Generator

PDF Generator is an easy-to-use and affordable solution to create a PDF file. This section demonstrates all basic elements used to create a PDF document from scratch.

4.1.1 Introduction

This section explains all the basic concepts of Essential PDF that helps to **create a simple PDF document**. It includes the following sub-sections.

- Document Object Model-Provides an overview of the Essential PDF object model, enabling you to work with PDF in a quick glance
- Pen and Brush-Describes the basic element used to draw graphic elements such as text and shapes
- Fonts-Demonstrates various fonts supported by Essential PDF, which can be used to write multilingual text

4.1.1.1 Document Object Model

PdfDocument is a top-level object in Essential PDF, which is a representation of a PDF document. The document contains a collection of sections that are represented by the **PdfSection** class, which is a logical entity containing a collection of pages and their settings. Pages (which are represented by **PdfPage class**) are the main destinations of the graphics output.



Note: A document should contain at least one section with one page.

A page does not contain any self-settings, but it inherits them from its parent section instead. It means that all the pages in the same section have the same dimensions. If a new page with different settings has to be created, then a new section should be created. A document also has capabilities to specify page settings. When a new section is created, it inherits page settings from the parent document.

You can save a document either to a file or stream through its **Save** method.

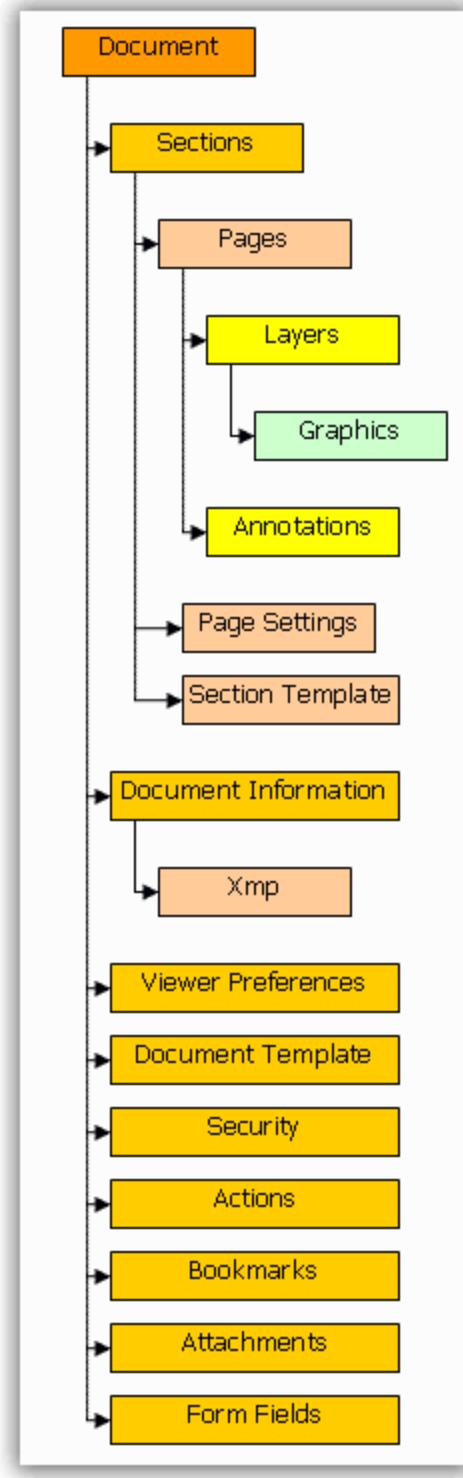


Figure 25: DOM

PDF Document

It manipulates with general document properties and stores sections. By default, an empty PDF document is created with a single page.

PdfDocument class is used to create a PDF document. The following are the public members of the PdfDocument class.



Note: To save the document asynchronously for Windows Store apps, refer to the [Asynchronous Support](#) section.

Methods

Name	Description
AddFields	Adds the fields connected to the page.
Append	Appends the specified loaded document to this one.
CheckFields	Checks what fields are connected with the page.
Clone	Creates a shallow copy of the current document.
ClonePage	Clones pages and their resource dictionaries and adds them into the document.
Close	Closes the document and releases the memory.
DisposeOnClose	Adds an object to a collection of the objects that will be disposed during document closing.
GetForm	Gets the form.
ImportPage	Imports a Page.
ImportPageRange	Imports a page range from a loaded document.
OnDocumentSaved	Raises DocumentSaved event.
OnPageSave	Called when a page is saved.
OnSaveProgress	Raises the [E:Progress] event.
PageLabelsSet	Informs the document that the page labels were set.
Save	Saves the document.

Properties

Name	Description
Actions	Gets the additional document's actions.
Attachments	Gets the attachments of the document.
Bookmarks	Gets the root of the bookmark tree in the document.
Cache	Gets collection of the cached objects.
Catalog	Gets the PDF document catalog.
ColorSpace	Gets or sets the color space for the page that will be created.
Compression	Gets or sets the desired level of stream compression.
Conformance	Gets or sets the Pdf Conformance level. Supported: PDF/A-1b - Level B compliance in Part 1.
DefaultFont	Gets the default font. It is used for complex objects when font is not explicitly defined.
DisposeObjects	Gets a list of the objects that have to be disposed after document closing.
DocumentInformation	Gets or sets document's information and properties.
FileStructure	Gets or sets the internal structure of the PDF file.
Form	Gets the interactive form of the document.
Pages	Gets the collection of the pages in the document.
PageSettings	Gets or sets default page settings of the document's sections.
Sections	Gets the collection of the sections in the document.
Security	Gets the security parameters of the document.
Template	Gets or sets a template that is applied to all pages in the document.
ViewerPreferences	Gets or sets a viewer preference object controlling the way the document is to be presented on the screen or in print.

Events

Name	Description
DocumentSaved	This event is raised when the document is saved.
SaveProgress	This event exposes the progression of the save state of the document.

Section

- Contains pages with the same parameters.
- When new pages are added to the section, it obtains parameters from the section even if it was imported from another section or document.
- When a high-level graphic object needs to be drawn on more than one page, it adds a new page to the section of the start page from which the object starts, if required.

PdfSection class allows creating and managing sections. The following are the members exposed by the PdfSection class.

Methods

Name	Description
Add	Overloaded.
Clear	Removes all the pages from the section.
Contains	Determines whether the page is within the section.
DrawTemplates	Draws page templates on the page.
IndexOf	Gets the index of the page.
Insert	Inserts a page at the specified index.
OnPageAdded	Called when the page is added.
OnPageSaving	Called when a page is being saved.
Remove	Removes the page from the section.
RemoveAt	Removes the page by its index in the section.

Properties

Name	Description
Count	Gets the count of the pages in the section.

Document	Gets the document.
Item	Gets the PdfPage at the specified index.
PageLabel	Gets or sets the page label.
Pages	Gets the pages.
PageSettings	Gets or sets the page settings of the section.
Template	Gets or sets a template for the pages in the section.

Events

Name	Description
PageAdded	This event is raised when a new page is added.

Page

Represents a rectangular area where the user can draw something, attach annotations, and so on. Each page has layers. When a page is created, it has one default layer. You can create more layers and refer them by an index. Each layer is associated with a graphics stream and has its own graphics (PdfGraphics class).

PdfPage class is used to create a page. The following are the members exposed by this class.

Methods

Name	Description
CreateTemplate	Creates a template from the page content and all annotation appearances.
ExtractImages	Extracts images from the given PDF page.
ExtractText	Extracts text from the given PDF Page.
GetAnnots	Gets the annotations array.
GetClientSize	Returns a page size reduced by page margins and page template dimensions.

GetContent	Gets the content of the page in form of a PDF template.
OnBeginSave	Raises BeginSave event.
ResetProgress	Resets the progress.
SetProgress	Sets the progress.
SetSection	Sets parent section to the page.

Properties

Name	Description
Annotations	Gets a collection of annotations of the page.
Contents	Gets array of page's content.
DefaultLayer	Gets the default layer of the page.
DefaultLayerIndex	Gets or sets index of the default layer.
Dictionary	Gets the page dictionary.
Document	Gets current document.
Graphics	Gets the graphics of the DefaultLayer.
Layers	Gets the collection of the page's layers.
Orientation	Gets the page orientation.
Rotation	Gets the page rotation.
Section	Gets the parent section of the page.
Size	Gets the size of the page.

Events

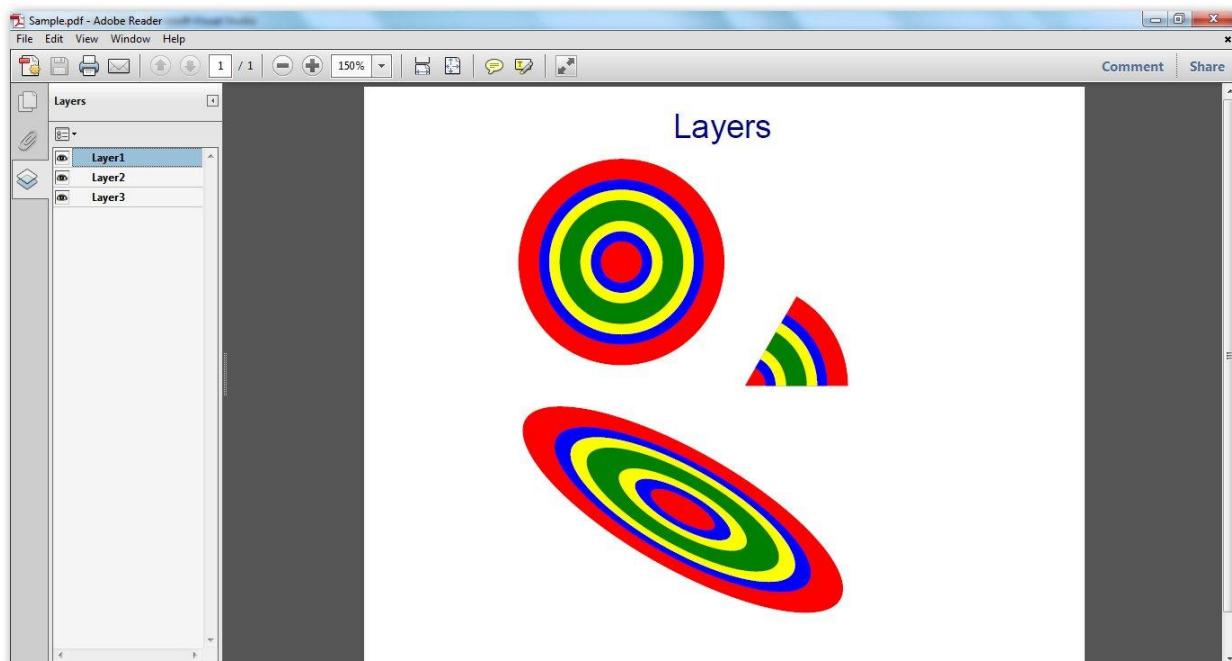
Name	Description
BeginSave	Raises before the page saves.

4.1.1.2 PDF Layers

This is a complete support for creating PDF layers, which allows creating unlimited levels of layers and sub-layers in a parent-child relationship. PDF layers provide the support for creating unlimited levels of layers and sub-layers. Through this support, the user can define optional content that can be shown or hidden by the user.

4.1.1.2.1 Use Case Scenario

It supports creating layers when working with a PDF document and making it visible and invisible dynamically.



Layers displayed in a PDF document

4.1.1.2.2 Properties Tables

Property	Description	Type	Data Type
Graphics	The graphics of the layer in the PDF document.	NA	PdfGraphics

4.1.1.2.3 Creating and Embedding PDF Layers in the PDF Document

To create PDF Layers, optional content should be implemented. The following code snippets explain the creation of optional content and the embedding of layers in a PDF document.

C#

```
//Add the layer
PdfPageLayer layer = page.Layers.Add("Layer1");
PdfGraphics graphics = layer.Graphics;
graphics.TranslateTransform(100, 60);

//Draw Arc
PdfPen pen = new PdfPen(Color.Red, 50);
RectangleF rect = new RectangleF(0, 0, 50, 50);
graphics.DrawArc(pen, rect, 360, 360);

pen = new PdfPen(Color.Blue, 30);
graphics.DrawArc(pen, 0, 0, 50, 50, 360, 360);

pen = new PdfPen(Color.Yellow, 20);
graphics.DrawArc(pen, rect, 360, 360);

pen = new PdfPen(Color.Green, 10);
graphics.DrawArc(pen, 0, 0, 50, 50, 360, 360);
```

VB

```
'Add the layer
Dim layer As PdfPageLayer = page.Layers.Add("Layer1")
Dim graphics As PdfGraphics = layer.Graphics
graphics.TranslateTransform(100, 60)

'Draw Arc
Dim pen As New PdfPen(Color.Red, 50)
Dim rect As New RectangleF(0, 0, 50, 50)
graphics.DrawArc(pen, rect, 360, 360)
```

```
pen = New PdfPen(Color.Blue, 30)
graphics.DrawArc(pen, 0, 0, 50, 50, 360, 360)

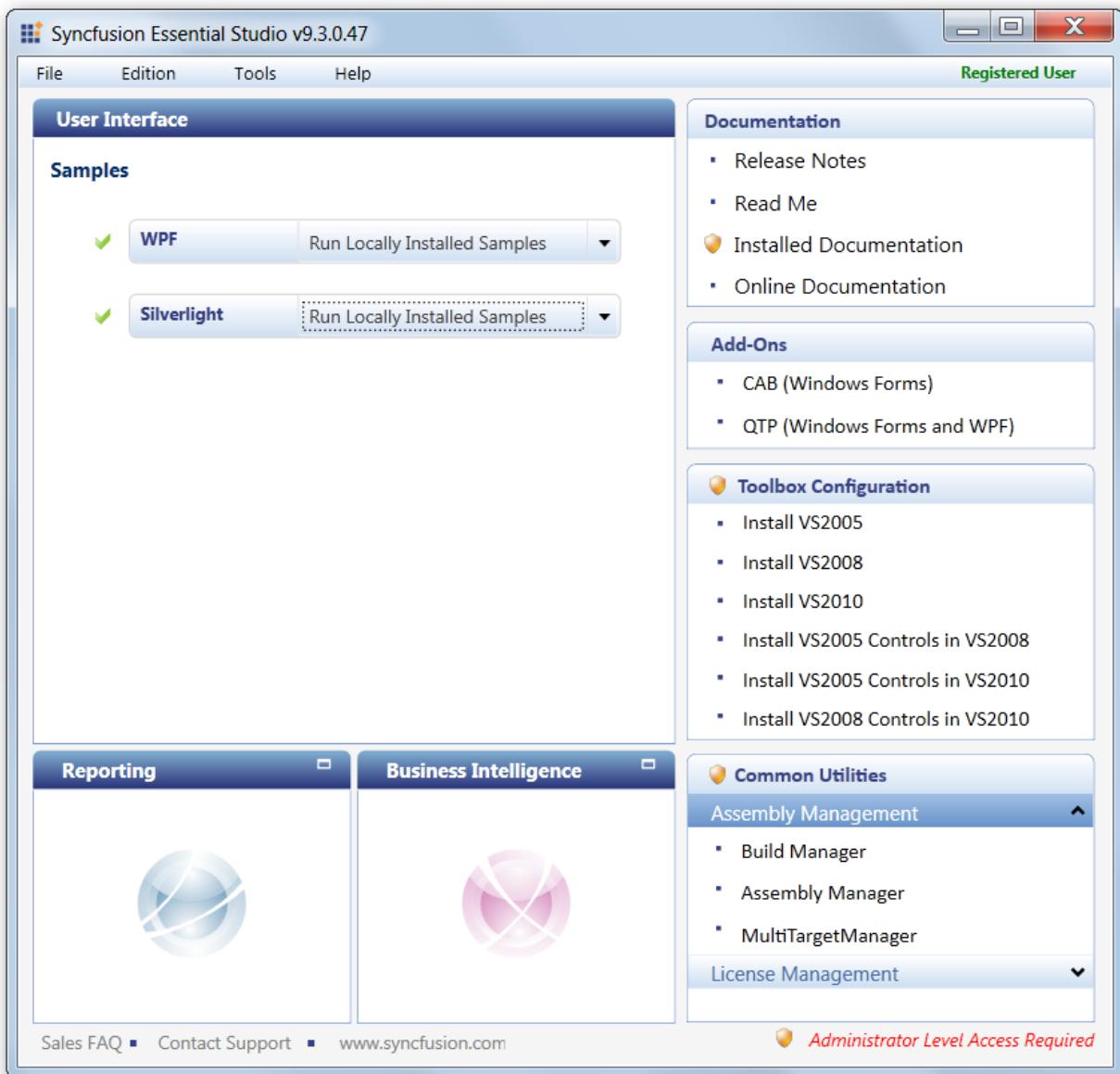
pen = New PdfPen(Color.Yellow, 20)
graphics.DrawArc(pen, rect, 360, 360)

pen = New PdfPen(Color.Green, 10)
graphics.DrawArc(pen, 0, 0, 50, 50, 360, 360)
```

4.1.1.2.4 Viewing Layers Samples

To view Layers samples:

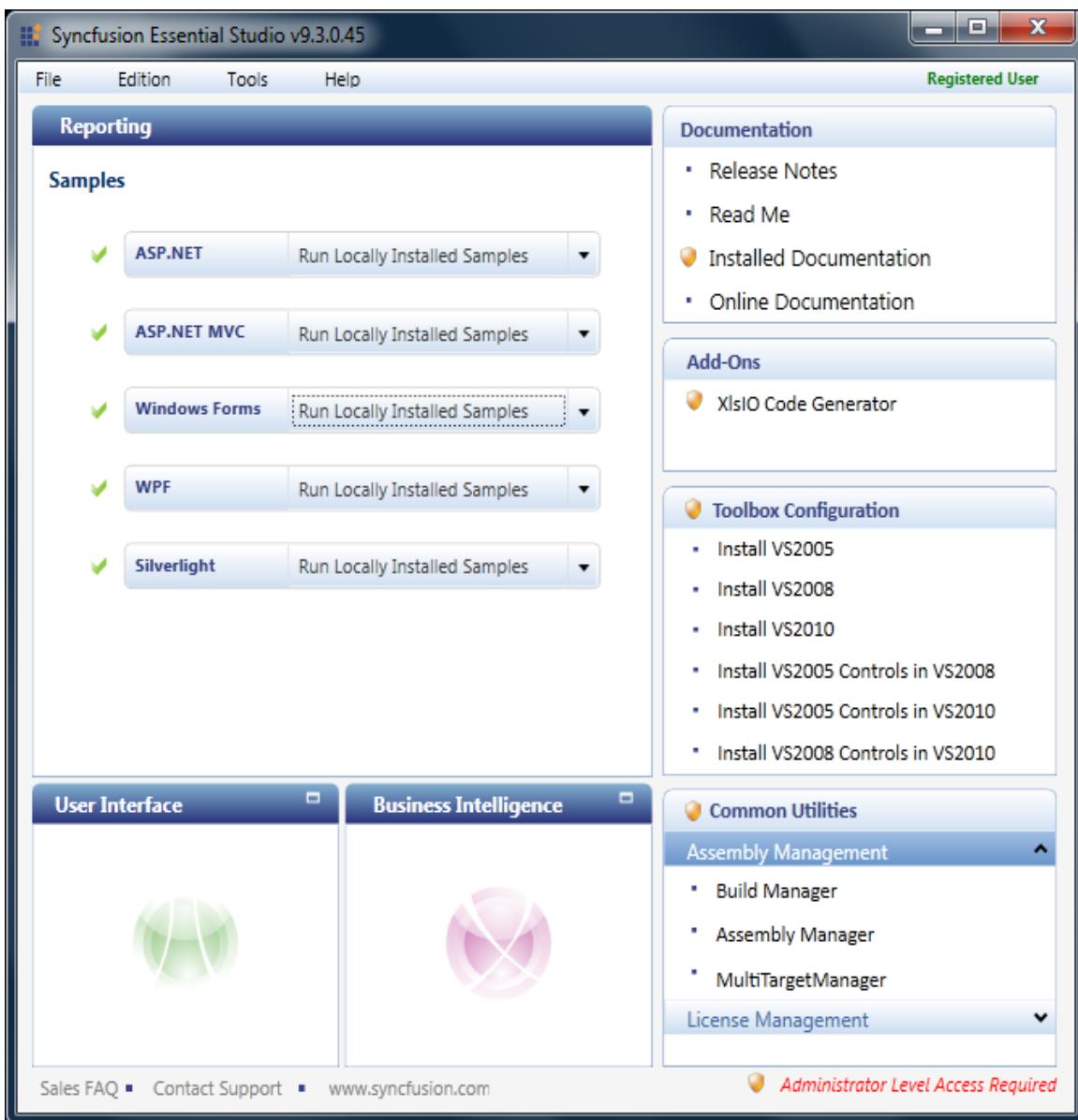
1. Open the Syncfusion Essential Studio Dashboard.
The Essential Studio Enterprise Edition window is displayed.



Essential Studio Dashboard

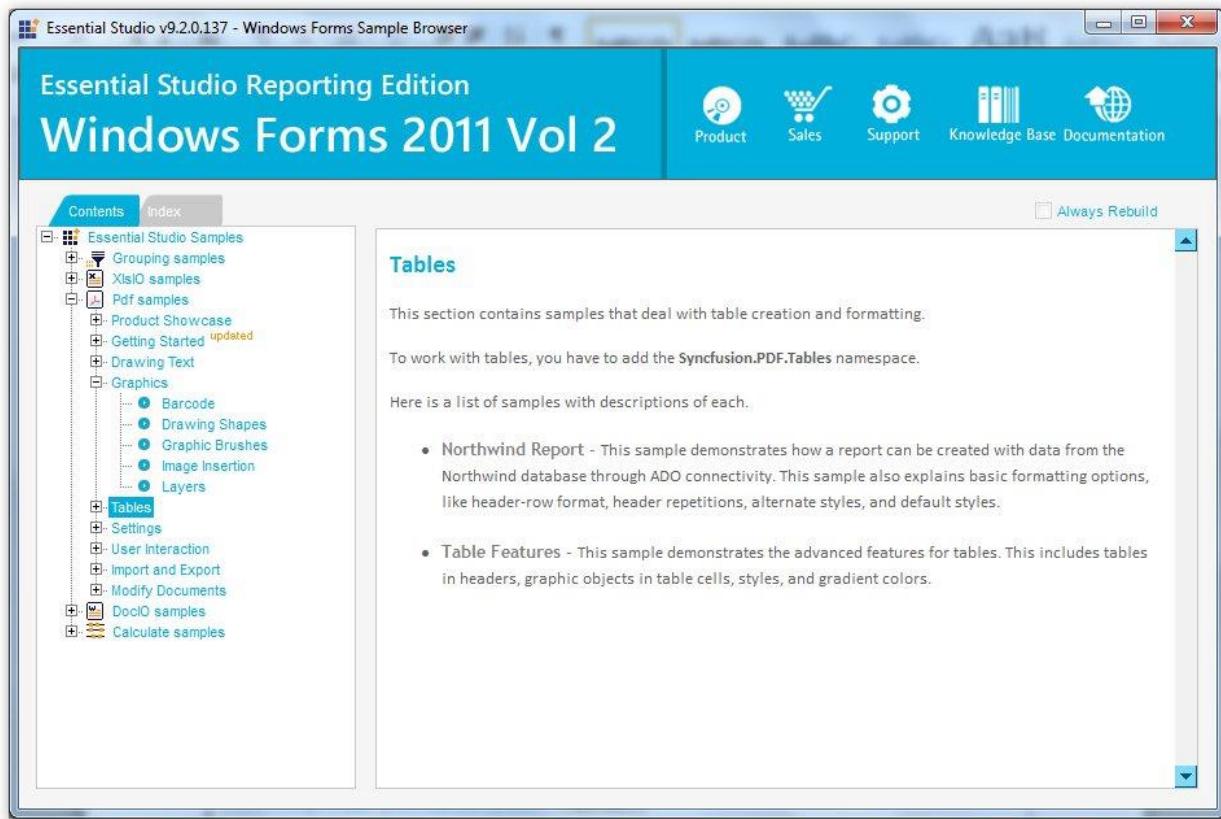
The **User Interface** edition panel is displayed by default.

2. Select **Reporting** and click the drop-down button of any platform e.g. Windows Forms, listed below **Samples**. The following options are displayed when clicking on the drop-down. You can view the samples in the following three ways:
 - **Run Locally Installed Samples** – to view the locally installed XlsIO samples using the sample browser
 - **Run Online Samples** – to view the online XlsIO samples.
 - **Explore Samples** – to locate the XlsIO samples on the disk



Syncfusion Essential Studio Dashboard

3. Choose **Run Locally Installed Samples**.
The Windows Forms Sample Browser displays.



Windows Forms Sample Browser

4. Go to **PDF samples** -> **Graphics** -> **Layers**.

For accessing ASP.NET samples:

- Go to **ASP.NET** -> **PDF samples** -> **Graphics** -> **Layers**.

For accessing ASP.NET MVC samples:

- Go to **ASP.NET MVC** -> **PDF samples** -> **Graphics** -> **Layers**.

For accessing WPF samples:

- Go to **WPF** -> **PDF samples** -> **Graphics** -> **Layers**.

For accessing Silverlight samples:

- Go to **Silverlight** -> **PDF samples** -> **Graphics** -> **Layers**.

4.1.1.3 Pen and Brush

Pen and Brush are two types of virtual graphics tools that are used to create objects, for example, **rectangle**, **ellipse** or **text**. Pen controls stroking operations (drawing borders and lines), while the brush controls filling operations (non-stroking).

Brush

There are four types of brushes. They are as follows. These brushes control the filling of the interior region of a shape.

- solid
- tiling
- linear gradient
- radial gradient.

1. PdfSolidBrush

This type of brush fills a shape with a single color. You may set the color while constructing the brush. The following code example illustrates this.

[C#]

```
PdfBrush brush = new PdfSolidBrush(Color.Black);
```

[VB .NET]

```
Dim brush As Syncfusion.Pdf.Graphics.PdfBrush = New  
Syncfusion.Pdf.Graphics.PdfSolidBrush(Color.Black)
```

2. PdfTilingBrush

This is one of the complex brushes available in Essential PDF. It enables you to fill the shape's interior with a repetitive pattern. To create your own pattern, use the **Graphics** property to obtain the **PdfGraphics** class instance, which allows drawing many graphics primitives.

3. PdfLinearGradientBrush

This brush is very similar to the .NET LinearGradientBrush. It has similar properties to specify blend colors and positions. Also, it requires the start and end position relative to the current origin to calculate gradient parameters and colors. Optionally, you may specify a rectangle, and a LinearGradientMode (which determines the orientation) or angle from the x-axis.

4. PdfRadialGradientBrush

This brush is similar to the above brush, the only exception being the gradient effect. Here the gradient is not linear, it's radial. This means, there are two circles with different center points and radii.

Pen

A pen controls drawing lines and shape borders. You may specify the width, different dash patterns and color of the pen. Optionally, you may create a pen from a brush which allows you to use gradients in drawing lines and curves. However, you should be careful with the coordinates of the brush.

PdfPen class defines these settings for drawing. The following are the properties of this class.

Name	Description
Brush	Gets or sets the brush, which specifies the pen behavior.
Color	Gets or sets the color of the pen.
DashOffset	Gets or sets the dash offset of the pen.
DashPattern	Gets or sets the dash pattern of the pen. The following dash patterns are available: Dash DashDot DashDotDot Dot Solid Custom
DashStyle	Gets or sets the dash style of the pen.
LineCap	Gets or sets the line cap of the pen.
LineJoin	Gets or sets the line join style of the pen.
MiterLimit	Gets or sets the miter limit.
Width	Gets or sets the width of the pen.

The following code example illustrates how to define a pen.

[C#]

```
PdfPen pen = new PdfPen(Color.Black);
```

[VB .NET]

```
Dim pen As Syncfusion.Pdf.Graphics.PdfPen = New
Syncfusion.Pdf.Graphics.PdfPen(Color.Black)
```

PdfPens and PdfBrushes

If you do not want to create pens and brushes on your own, you can use static classes that provide you with static immutable pens and brushes. Each property is named after the color of the pen or brush that it returns.

4.1.1.4 Fonts

PdfFont is the base class for all the fonts. The design of the fonts is similar to Microsoft.NET fonts. Fonts are immutable objects, i.e., they cannot be modified, once created.



Note: *There are special constructors that are used to create a new font from a prototype font, but with different settings.*

The following are the features of the PDF fonts:

- PDF font objects do not have any size; the size is set only during text printing. This provides the advantage of being able to use the same fonts with different sizes.
- There are no Underline and Strikeout font styles in PDF. Underline and Strikeout font styles are emulated by drawing a line.

As mentioned above, all the fonts derived from PdfFonts are immutable. However, there are capabilities for fonts caching. It means, if there are two similar fonts that have different sizes and Underline or Strikeout styles, just one font object will be stored in the PDF file. So, if a lot of similar fonts with different sizes, and Underline or Strikeout styles are created, there is a huge benefit from the fonts caching in terms of speed and memory usage.



Note: *Fonts caching works for all font types that are supported.*

The following are the classes derived from PdfFont.

1. PdfStandardFont

PdfStandardFont represents a font that is recognized by any Adobe Reader. It supports 14 types of fonts.

The following are some of the fonts supported by this class.

- Times-Roman (Regular, Bold, Italic, Bold Italic)
- Helvetica (Regular, Bold, Italic, Bold Italic)
- Courier (Regular, Bold, Italic, Bold Italic)
- Symbol
- ZapfDingbats



Note: Fonts that belong to this type do not support Unicode symbols. They take very less memory space, and it is suggested to use these fonts only when ASCII text has to be printed.

2. PdfTrueTypeFont

PdfTrueTypeFont fonts are created from TrueType fonts. The PdfTrueTypeFont fonts are created either from the System.Drawing.Font class or TTF file (a file containing the information about TrueType font). There are a variety of constructors that can enable to create fonts with different settings. This class is used to embed the specified font in the PDF document.



Note: There is a `unicode` parameter in some of the constructors that indicate whether the font should support unicode symbols. The fonts created from a TTF file support unicode symbols, by default. If there is no need to use Unicode symbols, it is suggested to set the `unicode` parameter to `False`. Fonts that do not support Unicode takes less memory space in the file.

- Right To Left Support

Unicode TrueType fonts created from the System.Drawing.Font class are used for RTL text output. Also, the languages with symbols substitution (like Arabic) are supported. To enable RTL and characters substitution, set **RightToLeft** property to **True** in the **PdfStringFormat** class.



- **RightToLeft property does not change the text alignment. It just enables the RTL engine and prints the text in RTL order. Use the Alignment property of PdfStringFormat to set the horizontal alignment of the text.**
- **Do not enable this feature if RTL support is not needed, because RTL support reduces the text printing speed.**

CJK Fonts

There is a set of standard PDF fonts that support Chinese, Japanese and Korean characters. These fonts are supported through the **PdfCjkStandardFont** class. Although creating such fonts is similar to the PdfStandardFont, it requires the following families in addition:

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

Text Measuring

The **MeasureString** method of the PdfFont class calculates the size of the text, number of lines that fit in the bounds, and number of characters in the text.

4.1.2 Drawing

This section explains how various drawing elements are drawn by using Essential PDF. It includes the following topics.

- Text-Describes various text drawing methods of Essential PDF.
- Graphics-Demonstrates various graphic elements in PDF.
- Tables-Explains table creation and formatting by using Essential PDF light table model.
- PDF Grid-PdfGrid is based on the cell model. It provides support for cell formatting, row and column spanning, and drawing nested tables in the PDF document.
- Lists-Describes how various lists can be drawn by using Essential PDF.

4.1.2.1 Text

This section deals with various types of text supported by Essential PDF. It includes the following topics:

- Draw Text-Provides brief description on rendering RTF text, unicode text and text flow mechanisms
- HTML Styled Text-Provides explanation on rendering HTML formatted text in PDF
- Automatic Field-Demonstrates how various dynamic fields such as date are inserted in PDF
- Links-Describes how text with links are inserted in a PDF document

4.1.2.1.1 Drawing Text

This section elaborates on various procedures for drawing the text in a PDF document.

The following are the procedures used:

- Using DrawString
- Paginating the text area
- String Formatting and
- RTF Support

Using DrawString

PdfGraphics class contains plenty of **DrawString** methods. It draws the specified text string at the specified location with the specified size, brush and font. The format of the methods is similar to the **System.Drawing.Graphics.DrawString** methods.

[C#]

```
public DrawString( string s, PdfFont font, PdfBrush brush, PointF point );
public DrawString( string s, PdfFont font, PdfBrush brush, PointF point,
PdfStringFormat format );
public DrawString( string s, PdfFont font, PdfBrush brush, float x, float y
);
public DrawString( string s, PdfFont font, PdfBrush brush, float x, float y,
PdfStringFormat format );
public DrawString( string s, PdfFont font, PdfPen pen, PointF point );
```

The following code example illustrates this.

[C#]

```

PdfLoadedDocument lDoc = new PdfLoadedDocument(filename);
page = lDoc.Pages.Add() as PdfPage;

PdfGraphics g = page.Graphics;
PdfFont font = new PdfStandardFont(PdfFontFamily.Helvetica, 14,
PdfFontStyle.Bold);
g.DrawString("Polygon", font, PdfBrushes.DarkBlue, new PointF(50, 0));

lDoc.Save(filename);
lDoc.Close();

```

[VB]

```

Dim lDoc As Syncfusion.Pdf.Parsing.PdfLoadedDocument = New
Syncfusion.Pdf.Parsing.PdfLoadedDocument(filename)
page = TryCast(lDoc.Pages.Add(), PdfPage)

Dim g As Syncfusion.Pdf.Graphics.PdfGraphics = page.Graphics
Dim font As Syncfusion.Pdf.Graphics.PdfFont = New
Syncfusion.Pdf.Graphics.PdfStandardFont(PdfFontFamily.Helvetica, 14,
PdfFontStyle.Bold)
g.DrawString("Polygon", font, PdfBrushes.DarkBlue, New PointF(50, 0))

lDoc.Save(filename)
lDoc.Close()

```

2. Paginating the text Area

PdfTextElement class represents the text area with the ability to span several pages. The **Layout** property of the **PDFLayoutFormat** class enables to break the text into multiple pages. Unicode text is also supported by this method.

[C#]

```

// Create a text element with large amount of text.
PdfTextElement element = new PdfTextElement(text, font);

// Set the properties for the text element.
element.Brush = new PdfSolidBrush(Color.Black);

// Set the string format. This can be used for setting unicode text.
element.StringFormat = format;

```

```
// Set the layout format properties to make the text flow through multiple
// pages.
PdfLayoutFormat layoutFormat = new PdfLayoutFormat();
layoutFormat.Break = PdfLayoutBreakType.FitPage;
layoutFormat.Layout = PdfLayoutType.Paginate;

// Set the bounds.
RectangleF bounds = new RectangleF(PointF.Empty, page.Graphics.ClientSize);

// Draw the text element.
element.Draw(page, bounds, layoutFormat);
```

[VB]

```
' Create a text element with large amount of text.
Dim element As Syncfusion.Pdf.Graphics.PdfTextElement = New
Syncfusion.Pdf.Graphics.PdfTextElement(Text, Font)

' Set the properties for the text element.
element.Brush = New PdfSolidBrush(Color.Black)

' Set the string format. This can be used for setting unicode text.
element.StringFormat = format

' Set the layout format properties to make the text flow through multiple
// pages.
Dim layoutFormat As Syncfusion.Pdf.Graphics.PdfLayoutFormat = New
Syncfusion.Pdf.Graphics.PdfLayoutFormat()
layoutFormat.Break = PdfLayoutBreakType.FitPage
layoutFormat.Layout = PdfLayoutType.Paginate

' Set the bounds.
Dim bounds As RectangleF = New RectangleF(PointF.Empty,
page.Graphics.ClientSize)

' Draw the text element.
element.Draw(page, bounds, layoutFormat)
```

3. String Formatting

For dedicated presentation of text in a PDF document, use a **PdfStringFormat** object. **PdfStringFormat** class provides support for the following features:

- CharacterSpacing, WordSpacing and LineSpacing
- Right-To-Left languages such as Arabic, Hebrew, Urdu, and so on
- Center, Left, Right and Justify text alignments

- Subscript and superscript modes
- ParagraphIndent customization
- WordWrapType style
- MeasureTrailingSpaces

4. RTF Support

The **Rich Text Format (RTF)** specification is a method of encoding formatted text and graphics such as bold characters and typefaces, document formatting and structures, for easy transfer between applications. Essential PDF supports drawing RTF text into a PDF document by using the **FromRtf** method in the **PdfImage** class.

You can draw RTF text by converting it into a bitmap or metafile image. Converting RTF text into a bitmap file, provides improved performance, while converting RTF text into a metafile image, provides high resolution and searchable text.

The following code example illustrates this.

[C#]

```
public PdfImage FromRtf( string rtf, float width, PdfImageType type )
public PdfImage FromRtf( string rtf, float width, float height, PdfImageType
type )
```

[VB .NET]

```
Public PdfImage FromRtf(String rtf, single width, PdfImageType type)
Public PdfImage FromRtf(String rtf, single width, single height, PdfImageType
type)
```

For More Information Refer:

[Draw Rich Text](#)

4.1.2.1.2 Html Styled Text

Essential PDF provides support to render the HTML string in a PDF document, which can flow to multiple pages by using the **PdfHTMLTextElement** class. The **PdfHTMLTextElement** class contains methods, which are used to render the specified HTML string in the PDF document. It draws the specified text string at the specified location with the specified size, brush and font. You can also align the text by using the **TextAlign** property.

The **PdfMetafileLayoutFormat** class enables to break the HTML text into multiple pages.

Supported Tags (Should be XHTML-compliant)

- Font
- B
- I
- U
- St
- sub
- sup
- BR

The following code example illustrates how to render the HTML string in a PDF document.

[C#]

```
// HtmlString
string longText = "<font color='#0000F8'>Essential PDF</font> is a
<u><i>.NET</i></u> " +
"library with the capability to produce Adobe PDF files ";

// Rendering HtmlText
PdfHTMLTextElement richTextElement = new PdfHTMLTextElement(longText, font,
brush);
richTextElement.TextAlign = TextAlign.Justify;

// Formatting Layout
PdfMetafileLayoutFormat format = new PdfMetafileLayoutFormat();
format.Layout = PdfLayoutType.OnePage;
format.Break = PdfLayoutBreakType.FitPage;

// Drawing htmlString
richTextElement.Draw(page, new RectangleF(30, 30, 600,
page.ClientSize().Height), format);
```

[VB.NET]

```
' HtmlString
Dim longText As String = "<font color='#0000F8'>Essential PDF</font> is a
<u><i>.NET</i></u> " +
"library with the capability to produce Adobe PDF files "

' Rendering HtmlText
```

```

Dim richTextElement As PdfHTMLTextElement = New PdfHTMLTextElement(longText,
font, brush)
richTextElement.TextAlign = TextAlign.Justify

' Formatting Layout
Dim format As PdfMetafileLayoutFormat = New PdfMetafileLayoutFormat()
format.Layout = PdfLayoutType.OnePage
format.Break = PdfLayoutBreakType.FitPage

' Drawing htmlString
richTextElement.Draw(page, New RectangleF(30, 30, 600,
page.GetClientSize().Height), format)

```

4.1.2.1.3 Automatic Fields

Automatic Fields are special objects that display information calculated automatically, just before the document is saved.

The following are the fields displayed:

- Page number
- Count of pages
- Author of the document
- Creation and current date
- Other document information



Note: These fields are not always evaluated at the moment of constructing the document.

To display the correct value of the field, you should specify some important properties, which are listed below.

- **Font**-Used to display the value of the field. Exception is thrown, if this property is not set.
- **Brush**-Used to print the value of the field. Exception is thrown, if this property is not set.
- **Bounds**-Specifies the bounds of the field

You can also use the **Location** and **Size** properties to define the bounds of the field.

- **Location**: Represents the location of the field. Default point is (0, 0).
- **Size**: Represents the size of the field. If it is not set, the size of the field is automatically calculated to display the value.



Note: It is not necessary to set the **Bounds** and **Size** with **Location** at the same time. **Size** and **Bounds.Size** have the same values.

Numeric fields have an additional **NumberingStyle** property. There are five possible numbering styles supported by the automatic fields. They are as follows.

- Arabic (1, 2, 3, 4, ...)
- Upper Roman (I, II, III, IV, ...)
- Roman (i, ii, iii, iv, ...)
- Upper Latin (A, B, C, D, ..., Z, AA, AB, ...)
- Latin (a, b, c, d, ..., z, aa, ab, ...)

Brief descriptions on the various numbering fields are given below.

- **PdfPageNumberField** - Specifies the number of the page on which the field has been drawn
- **PdfPageCountField** - Specifies the total number of pages in the document
- **PdfSectionPageNumberField** - Specifies the number of pages within a section
- **PdfSectionPageCountField** - Specifies the number of sections in a document
- **PdfSectionNumberField** - Specifies the number of sections within a document
- **PdfCreationDateField** - Specifies the creation date of the document

The value is taken from the **DocumentInformation.CreationDate** property.

- **PdfDateTimeField** - Specifies the current date and time
- **PdfDestinationPageNumberField** - Specifies the number of the specified destination page
- **PdfCompositeField** - Specifies the value of the field that is composed of any number of other automatic fields

PdfCreationDateField and **PdfDateTimeField** have the **DateFormatString** property, which defines the formatting string for the value of the field. This property uses the same formatting rules and specifiers as **DateTime** type of .NET. For detailed information on formatting specifiers, see [http://msdn2.microsoft.com/en-us/library/73ctwf33\(VS.80\).aspx](http://msdn2.microsoft.com/en-us/library/73ctwf33(VS.80).aspx).

You can draw the Automatic Fields on the **PdfTemplate** and set them as the document template or manually draw them on the necessary pages. The values of the fields will be automatically populated on each copy of the template.

The following code example illustrates how to display page numbers.

[C#]

```

PdfDocument document = new PdfDocument();
PdfPage page = document.Pages.Add();

PdfFont font = new PdfStandardFont(PdfFontFamily.Helvetica, 12f);
PdfBrush brush = PdfBrushes.Black;

PdfPageNumberField pageNumber = new PdfPageNumberField(font, brush);

for (int i = 0; i < 50; i++)
{
    page = document.Pages.Add();
    pageNumber.Draw(page.Graphics);
}

```

[VB.NET]

```

Dim document As PdfDocument = New PdfDocument()
Dim page As PdfPage = document.Pages.Add()

Dim font As PdfFont = New PdfStandardFont(PdfFontFamily.Helvetica, 12.0F)
Dim brush As PdfBrush = PdfBrushes.Black

Dim pageNumber As PdfPageNumberField = New PdfPageNumberField(font, brush)

For i As Integer = 0 To 49
    page = document.Pages.Add()
    pageNumber.Draw(page.Graphics)
Next i

```

The following code example illustrates how to use a composite field.

[C#]

```

PdfDocument document = new PdfDocument();
PdfPage page = document.Pages.Add();

PdfFont font = new PdfStandardFont(PdfFontFamily.Helvetica, 12f);
PdfBrush brush = PdfBrushes.Black;

PdfCompositeField compositeField = new PdfCompositeField(font, brush);

for (int i = 0; i < 50; i++)
{

```

```

PdfPage page = document.Pages.Add();
compositeField.Draw(page.Graphics);
}

```

[VB.NET]

```

Dim document As PdfDocument = New PdfDocument()
Dim page As PdfPage = document.Pages.Add()

Dim font As PdfFont = New PdfStandardFont(PdfFontFamily.Helvetica, 12.0F)
Dim brush As PdfBrush = PdfBrushes.Black

Dim compositeField As PdfCompositeField = New PdfCompositeField(font, brush)

For i As Integer = 0 To 49
Dim page As PdfPage = document.Pages.Add()
    compositeField.Draw(page.Graphics)
Next i

```

When an automatic field is used as a component of the composite field, it is not necessary to specify its Font, Brush and Bounds properties. Just call its constructor without parameters.



Note: You must specify the preceding properties for the composite field.

The following code example illustrates how to use automatic fields in templates.

[C#]

```

PdfDocument document = new PdfDocument();
PdfPage page = document.Pages.Add();

PdfFont font = new PdfStandardFont(PdfFontFamily.Helvetica, 12f);
PdfBrush brush = PdfBrushes.Black;

PdfTemplate template = new PdfTemplate(15, 15);

PdfDateTimeField dateField = new PdfDateTimeField(font, brush);
dateField.DateFormatString = "dd'/'MMMM'/'yyyy";

dateField.Draw(template.Graphics);

for (int i = 0; i < 50; i++)
{

```

```

PdfPage page = document.Pages.Add();
page.Graphics.DrawPdfTemplate(template, new Point(50, 50));
}

```

[VB.NET]

```

Dim document As PdfDocument = New PdfDocument()
Dim page As PdfPage = document.Pages.Add()

Dim font As PdfFont = New PdfStandardFont(PdfFontFamily.Helvetica, 12.0F)
Dim brush As PdfBrush = PdfBrushes.Black

Dim template As PdfTemplate = New PdfTemplate(15, 15)

Dim dateField As PdfDateTimeField = New PdfDateTimeField(font, brush)
Dim dateField.DateFormatString = "dd'/'MMMM'/'yyyy"

dateField.Draw(template.Graphics)

For i As Integer = 0 To 49
    Dim page As PdfPage = document.Pages.Add()
    page.Graphics.DrawPdfTemplate(template, New Point(50, 50))
Next i

```

4.1.2.1.4 Links

A hyperlink, which is more commonly called a link, is an electronic connection between one web page and other web pages on the same web site, or web pages located on another web site. More specifically, a hyperlink is a connection between one page of a hypertext document to another.

You can create hyperlinks in a PDF document by using the **PdfTextWebLink** class. The **DrawTextWebLink** method is used to draw hyperlinks in PDF pages.

The following code example illustrates how to draw hyperlinks.

[C#]

```

// Create the Text Web Link
PdfTextWebLink textLink = new PdfTextWebLink();
textLink.Url = "http://www.google.com";

```

```
textLink.Text = "Google Search";
textLink.Brush = brush;
textLink.Font = font;
textLink.Pen = PdfPens.Brown;
textLink.DrawTextWebLink(page, new PointF(10, 40));
```

[VB .NET]

```
' Create the Text Web Link
Dim textLink As PdfTextWebLink = New PdfTextWebLink()
textLink.Url = "http://www.google.com"
textLink.Text = "Google Search"
textLink.Brush = brush
textLink.Font = font
textLink.Pen = PdfPens.Brown
textLink.DrawTextWebLink(page, New PointF(10, 40))
```

4.1.2.2 Graphics

Primitives

The general class **PdfGraphics** enables to draw a wide range of primitives like lines, curves, paths and text. For each such operation there is a set of methods called **Draw<primitive>()** (e.g. **DrawLine**). Each set of methods accepts parameters specific to each primitive type (for example: pen, brush, boundaries, etc). If pen is used, the primitive is drawn, and if brush is used, the primitive is filled. You can also use Null value instead of pen or brush.

The following are the public members exposed by the **PdfGraphics** class.

Methods

Name	Description
CheckCorrectLayoutRectangle	Creates laid
ClipTranslateMargins	Sets the drawing area and translates the origin
DrawArc	Draws an arc
DrawBezier	Draws a Bezier curve
DrawEllipse	Draws an ellipse

DrawImage	Inserts an image
DrawLine	Draws a line
DrawPath	Draws a path
DrawPdfTemplate	Draws a PDF template
DrawPie	Draws a Pie
DrawPolygon	Draws a Polygon shape
DrawRectangle	Draws a rectangle shape
DrawString	Draws the text
DrawStringLayoutResult	Draws a layout result
GetBezierArcPoints	Gets the Bezier points for arc constructing
GetLineBounds	Returns bounds of the line info
GetTextVerticalAlignShift	Calculates shift value if the text is vertically aligned
InitializeCoordinates	Initializes coordinate system
MultiplyTransform	Multiplies the world transformation of the Graphics and specifies the Matrix
PutComment	Puts a comment line
Reset	Clears instances
Restore	Restores the state of the Graphics to the state represented by a GraphicsState or to the last state
RotateTransform	Rotates coordinate system in counterclockwise direction
Save	Saves the current state of the Graphics and identifies the saved state with a GraphicsState
ScaleTransform	Scales coordinates by specified coordinates
SetBBox	Sets the BBox entry of the graphics dictionary
SetClip	Modifying the current clipping path by intersecting it with the current path

SetLayer	Sets the layer for the graphics
SetTransparency	Sets transparency
SkewTransform	Skews coordinate system axes
TranslateTransform	Translates coordinates by specified coordinates

Properties

Name	Description
AutomaticFields	Gets the automatic fields
ClientSize	Gets a SizeF structure that binds the region of the Graphics reduced by margins and page templates
ColorSpace	Gets or sets the current color space
Layer	Gets the layer for the graphics, if exists
Matrix	Gets the matrix reflecting current transformation
Page	Gets the page for this graphics, if exists
Size	Gets the size of the canvas
StreamWriter	Gets the stream writer

Units, Size and Co-ordinate System

The co-ordinate system is either Top or Left. Origin is translated depending on the margins and page templates. The measure units are points (1/72 inch). The **PdfUnitConvertor** class enables to convert different measure units.

Size property of PdfGraphics returns the size of the canvas. Also, **ClientSize** property returns a client area of the canvas, which might be smaller. Any output of the client area will not be visible.

Graphics State and Co-ordinate System Manipulation

Also, PdfGraphics class allows manipulating with graphics state (save, restore) and coordinate system (rotate, translate, etc). **Save** and **Restore** methods can be used for manipulating with graphics state, while TranslateTransform, RotateTransform, etc., can be used for co-ordinates manipulating. Also, clip regions can be set using the **SetClip** method.

You may save the current graphics state, translate the origin, rotate and draw some primitives, restore the graphics state and continue drawing with the restored coordinate system.

Transparency

You may specify transparency for pen operations (for example: drawing lines), brush operations (for example: filling shapes), and for both of them simultaneously.

Also, you may specify the method of the resulting color calculation. Transparency is set by using the **SetTransparency** method of PdfGraphics. It includes the following blend modes:

- Normal
- Multiply
- Screen
- Overlay
- Darken
- Lighten
- ColorDodge
- ColorBurn
- HardLight
- SoftLight
- Difference
- Exclusion

Text Output

There are plenty of **DrawString** methods in PdfGraphics that allow text printing. The format of the methods are similar to the System.Drawing.Graphics.DrawString methods.

PdfPen as well as **PdfBrush** are used to print the text. You can use either object, or even both pen and brush for the text output. PdfPen sets the text boundaries while PdfBrush fills the internal area of the text.

If the coordinates are used for the text output only, it will be printed despite the graphics boundaries. New lines symbols split the text by lines only. If the bound (RectangleF structure) of the text is set, the text will be laid out to fit the boundaries. If the width of the boundaries is set to 0 or less, the text will not be limited horizontally. If the height of the boundaries is set to 0 or less, the text will be limited by the boundaries of the PdfGraphics.

Text Output Settings

To apply different settings to text output, **PdfStringFormat** class is used. It contains variety of properties that allow to set different text output settings.

Note: You must add the **Syncfusion.Pdf.Graphics** namespace to work with Graphics and graphic elements.

4.1.2.2.1 Graphic Elements

These include the basic functionality of drawing elements on the canvas (PdfGraphics). As a result, you can draw such objects on pages or any other object that has graphics context (PdfTemplate etc). Graphics elements are simple and does not span several pages.

Layout

PdfLayoutElement class provides an ability to draw contents on several pages. This functionality is described in the [Pagination](#) section.

Shapes

PdfShapeElement class provides the basic functionality of simple graphics primitives (like lines, rectangles, etc.). It is derived from LayoutElement, which enables every shape to span several pages. The basic graphics primitives are as follows:

- Line
- Rectangle
- Polygon
- Arc
- Bezier Curve
- Ellipse
- Path
- PdfTemplate
- Pie
- Image

Each shape can be drawn by its own **Draw** method or by using an appropriate method of the **PdfGraphics** class (like DrawLine, DrawRectangle, etc.). Each shape has its own coordinate system (which is equal to a page coordinate system). Coordinates of the shape are set in that coordinate system. When the shape is going to be drawn by using its **Draw** method, its coordinate system is translated by the coordinates set to the Draw method. So, whenever the shape is going to be drawn, take its own coordinate system into consideration.

The following screen shot illustrates the shape drawing behavior.

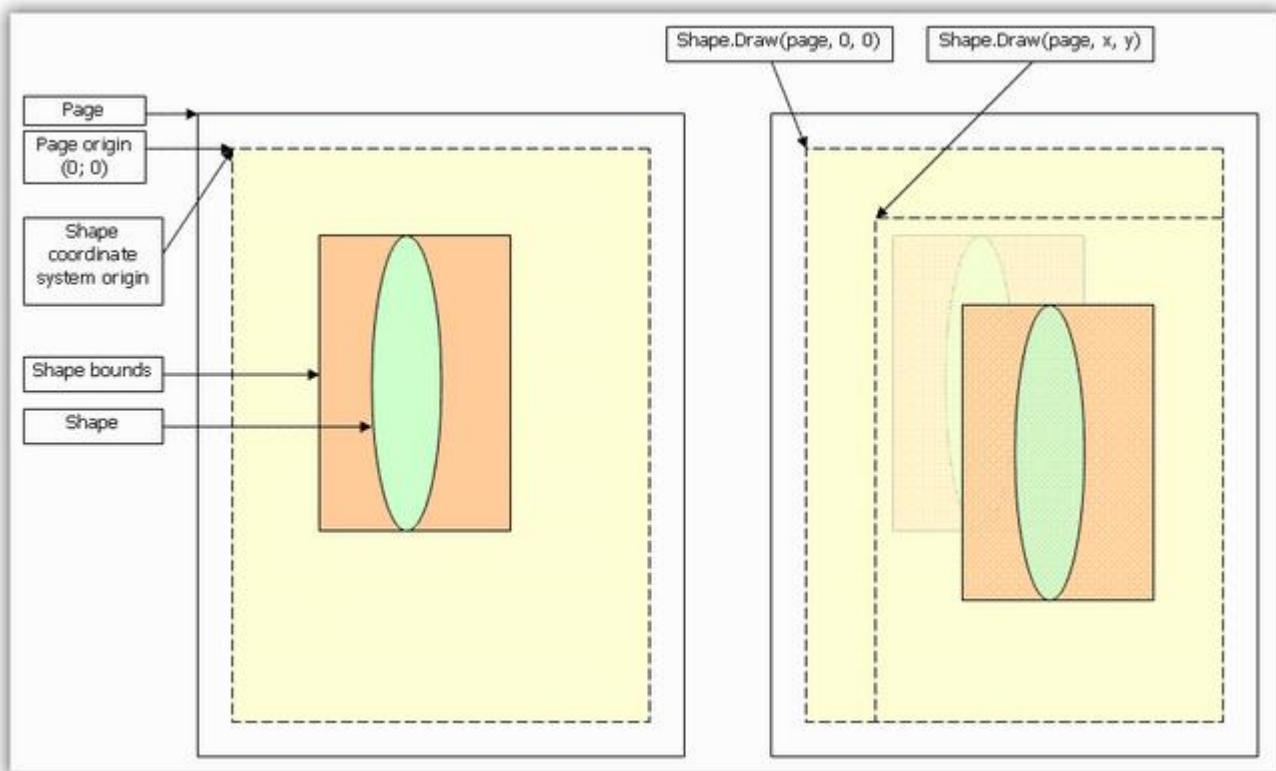


Figure 26: Shape drawing behavior

4.1.2.2.2 ColorSpace

While preparing your document, you may need colors in different color spaces. You may control it in the following two ways.

- Document level and
- Graphics level

Document Level

The **PdfDocument** class has the **ColorSpace** property, which is used to choose color spaces for new pages. Color spaces can be classified into three color space families.

Essential PDF now supports the following color spaces in each family.

Device Color Spaces

- DeviceGray
- DeviceRGB
- DeviceCMYK

CIE-based Color Spaces

- CalGray
- CalRGB
- Lab

ICC-based Color Spaces

- Special color spaces
- Indexed
- Separation

[C#]

```
// page1 has default color space.
PdfPage page1 = doc.Pages.Add();

doc.ColorSpace = PdfColorSpace.CMYK;

// page2 has CMYK color space.
PdfPage page2 = doc.Pages.Add();

page = doc.Pages.Add();
g = page.Graphics;

// Create CalRGB color space
PdfCalRGBColorSpace calRgbCS = new PdfCalRGBColorSpace();

// Update color values
calRgbCS.Gamma = new double[] { 1.6, 1.1, 2.5 };
calRgbCS.Matrix = new double[] { 1, 0, 0, 0, 1, 0, 0, 0, 1 };
calRgbCS.WhitePoint = new double[] { 0.2, 1, 0.8 };
PdfCalRGBColor red = new PdfCalRGBColor(calRgbCS);
red.Red = 0;
red.Green = 1;
red.Blue = 0;

PdfBrush brush2 = new PdfSolidBrush(red);
RectangleF rect = new RectangleF(20, 30, 30, 30);
```

```
// Draw using the PdfBrush
g.DrawRectangle(brush2, rect);

// Create CalGray color space
PdfCalGrayColorSpace calGrayCS = new PdfCalGrayColorSpace();

// Update color values
calGrayCS.Gamma = 0.7;
calGrayCS.WhitePoint = new double[] { 0.2, 1, 0.8 };
PdfCalGrayColor red1 = new PdfCalGrayColor(calGrayCS);
red1.Gray = 0.1;
brush2 = new PdfSolidBrush(red1);
rect = new RectangleF(100, 30, 30, 30);

// Draw using the PdfBrush
g.DrawRectangle(brush2, rect);

// Create Lab color space
PdfLabColorSpace calGrayCS1 = new PdfLabColorSpace();

// Update color values
calGrayCS1.Range = new double[] { 0.2, 1, 0.8, 23.5 };
calGrayCS1.WhitePoint = new double[] { 0.2, 1, 0.8 };
PdfLabColor red2 = new PdfLabColor(calGrayCS1);
red2.L = 90;
red2.A = 0.5;
red2.B = 20;

brush2 = new PdfSolidBrush(red2);
rect = new RectangleF(180, 30, 30, 30);

// Draw using the PdfBrush
g.DrawRectangle(brush2, rect);

// Creating ICCBased color space
PdfCalRGBColorSpace calRgbCS3 = new PdfCalRGBColorSpace();
calRgbCS3.Gamma = new double[] { 7.6, 5.1, 8.5 };
calRgbCS3.Matrix = new double[] { 1, 0, 0, 0, 1, 0, 0, 0, 1 };
calRgbCS3.WhitePoint = new double[] { 0.7, 1, 0.8 };

// Read the ICC profile.
FileStream fs = new FileStream(@"rgb.icc", FileMode.Open, FileAccess.Read);
byte[] profileData = new byte[fs.Length];
fs.Read(profileData, 0, profileData.Length);
fs.Close();
```

```
// Instantiate ICC color space
PdfICCCColorSpace IccBasedCS4 = new PdfICCCColorSpace();
IccBasedCS4.ProfileData = profileData;
IccBasedCS4.AlternateColorSpace = calRgbCS3;
IccBasedCS4.ColorComponents = 3;
IccBasedCS4.Range = new double[] { 0.0, 1.0, 0.0, 1.0, 0.0, 1.0 };

PdfICCCColor red4 = new PdfICCCColor(IccBasedCS4);
red4.ColorComponents = new double[] { 1, 0, 1 };
rect = new RectangleF(20, 110, 30, 30);
brush2 = new PdfSolidBrush(red4);

// Draw using the PdfBrush
g.DrawRectangle(brush2, rect);

// Create Separation color space
PdfExponentialInterpolationFunction function = new
PdfExponentialInterpolationFunction(true);
float[] numArray = new float[3] { 90f, 0.5f, 20f };
function.C1 = numArray;

PdfLabColorSpace calLabCS8 = new PdfLabColorSpace();
calLabCS8.Range = new double[] { 0.2, 1, 0.8, 23.5 };
calLabCS8.WhitePoint = new double[] { 0.2, 1, 0.8 };

// Instantiate Separation color space
PdfSeparationColorSpace colorspace8 = new PdfSeparationColorSpace();
colorspace8.AlternateColorSpaces = calLabCS8;
colorspace8.TintTransform = function;
colorspace8.Colorant = "PANTONE Orange 021 C";
PdfSeparationColor color8 = new PdfSeparationColor(colorspace8);
color8.Tint = 0.7;

brush2 = new PdfSolidBrush(color8);
rect = new RectangleF(100, 110, 30, 30);

// Draw using the PdfBrush
g.DrawRectangle(brush2, rect);

// Create Indexed color space
PdfIndexedColorSpace colorspace7 = new PdfIndexedColorSpace();

// Updated color values
colorspace7.BaseColorSpace = new PdfDeviceColorSpace(PdfColorSpace.RGB);
colorspace7.MaxColorIndex = 3;
```

```

colorspace7.IndexedColorTable = new byte[] { 150, 0, 222, 255, 0, 0, 0, 255,
0, 0, 0, 255 };

PdfIndexedColor color7 = new PdfIndexedColor(colorspace7);
color7.SelectColorIndex = 3;

brush2 = new PdfSolidBrush(color7);
rect = new RectangleF(180, 110, 30, 30);

// Draw using the PdfBrush
g.DrawRectangle(brush2, rect);

```

[VB.NET]

```

' page1 has default color space.
Dim page1 As Syncfusion.Pdf.PdfPage = doc.Pages.Add()

doc.ColorSpace = PdfColorSpace.CMYK

' page2 has CMYK color space.
Dim page2 As Syncfusion.Pdf.PdfPage = doc.Pages.Add()

page = doc.Pages.Add()
g = page.Graphics

' Create CalRGB color space
Dim calRgbCS As PdfCalRGBColorSpace = New PdfCalRGBColorSpace()

' Update color values
calRgbCS.Gamma = New Double() { 1.6, 1.1, 2.5 }
calRgbCS.Matrix = New Double() { 1, 0, 0, 0, 1, 0, 0, 0, 1 }
calRgbCS.WhitePoint = New Double() { 0.2, 1, 0.8 }
Dim red As PdfCalRGBColor = New PdfCalRGBColor(calRgbCS)
red.Red = 0
red.Green = 1
red.Blue = 0

Dim brush2 As PdfBrush = New PdfSolidBrush(red)
Dim rect As RectangleF = New RectangleF(20, 30, 30, 30)

' Draw using the PdfBrush
g.DrawRectangle(brush2, rect)

' Create CalGray color space
Dim calGrayCS As PdfCalGrayColorSpace = New PdfCalGrayColorSpace()

```

```

' Update color values
calGrayCS.Gamma = 0.7
calGrayCS.WhitePoint = New Double() { 0.2, 1, 0.8 }
Dim red1 As PdfCalGrayColor = New PdfCalGrayColor(calGrayCS)
red1.Gray = 0.1
brush2 = New PdfSolidBrush(red1)
rect = New RectangleF(100, 30, 30, 30)

' Draw using the PdfBrush
g.DrawRectangle(brush2, rect)

' Create Lab color space
Dim calGrayCS1 As PdfLabColorSpace = New PdfLabColorSpace()

' Update color values
calGrayCS1.Range = New Double() { 0.2, 1, 0.8, 23.5 }
calGrayCS1.WhitePoint = New Double() { 0.2, 1, 0.8 }
Dim red2 As PdfLabColor = New PdfLabColor(calGrayCS1)
red2.L = 90
red2.A = 0.5
red2.B = 20

brush2 = New PdfSolidBrush(red2)
rect = New RectangleF(180, 30, 30, 30)

' Draw using the PdfBrush
g.DrawRectangle(brush2, rect)

' Creating ICCBased color space
Dim calRgbCS3 As PdfCalRGBColorSpace = New PdfCalRGBColorSpace()
calRgbCS3.Gamma = New Double() { 7.6, 5.1, 8.5 }
calRgbCS3.Matrix = New Double() { 1, 0, 0, 0, 1, 0, 0, 0, 1 }
calRgbCS3.WhitePoint = New Double() { 0.7, 1, 0.8 }

' Read the ICC profile.
Dim fs As FileStream = New FileStream("rgb.icc", FileMode.Open,
FileAccess.Read)
Dim profileData As Byte() = New Byte(fs.Length - 1) {}
fs.Read(profileData, 0, profileData.Length)
fs.Close()

' Instantiate ICC color space
Dim IccBasedCS4 As PdfICCCColorSpace = New PdfICCCColorSpace()
IccBasedCS4.ProfileData = profileData
IccBasedCS4.AlternateColorSpace = calRgbCS3

```

```

IccBasedCS4.ColorComponents = 3
IccBasedCS4.Range = New Double() { 0.0, 1.0, 0.0, 1.0, 0.0, 1.0 }

Dim red4 As PdfICCCColor = New PdfICCCColor(IccBasedCS4)
red4.ColorComponents = New Double() { 1, 0, 1 }
rect = New RectangleF(20, 110, 30, 30)
brush2 = New PdfSolidBrush(red4)

' Draw using the PdfBrush
g.DrawRectangle(brush2, rect)

' Create Separation color space
Dim exFunction As PdfExponentialInterpolationFunction = New
PdfExponentialInterpolationFunction(True)
Dim numArray As Single() = New Single(2) {90.0F, 0.5F, 20.0F}
exFunction.C1 = numArray

Dim calLabCS8 As PdfLabColorSpace = New PdfLabColorSpace()
calLabCS8.Range = New Double() { 0.2, 1, 0.8, 23.5 }
calLabCS8.WhitePoint = New Double() { 0.2, 1, 0.8 }

' Instantiate Separation color space
Dim colorspace8 As PdfSeparationColorSpace = New PdfSeparationColorSpace()
colorspace8.AlternateColorSpaces = calLabCS8
colorspace8.TintTransform = exFunction
colorspace8.Colorant = "PANTONE Orange 021 C"
Dim color8 As PdfSeparationColor = New PdfSeparationColor(colorspace8)
color8.Tint = 0.7

brush2 = New PdfSolidBrush(color8)
rect = New RectangleF(100, 110, 30, 30)

' Draw using the PdfBrush
g.DrawRectangle(brush2, rect)

' Create Indexed color space
Dim colorspace7 As PdfIndexedColorSpace = New PdfIndexedColorSpace()

' Updated color values
colorspace7.BaseColorSpace = New PdfDeviceColorSpace(PdfColorSpace.RGB)
colorspace7.MaxColorIndex = 3
colorspace7.IndexedColorTable = New Byte() { 150, 0, 222, 255, 0, 0, 0, 255,
0, 0, 0, 255 }

Dim color7 As PdfIndexedColor = New PdfIndexedColor(colorspace7)
color7.SelectColorIndex = 3

```

```

brush2 = New PdfSolidBrush(color7)
rect = New RectangleF(180, 110, 30, 30)

' Draw using the PdfBrush
g.DrawRectangle(brush2, rect)

```



Note: This property will not convert the entire document into different color spaces. Also, it will not have any impact on pages that have been created before the property value has changed. It just chooses the color space for the pages created after the alteration.

Graphics Level

The **PdfGraphics** class has its own **ColorSpace** property, which chooses the color space for the objects that will be drawn next. This property alternates the document settings and enables to change the color space as many times as you wish. **Save** and **Restore** methods saves and restores the color space, along with other graphics state parameters respectively.

[C#]

```

PdfPen pen = new PdfPen(Color.Red);
PdfBrush brush = new PdfSolidBrush(Color.Blue);

// Default color space.
graphics.DrawRectangle(pen, brush, rectangle);

graphics.Save();

// GrayScale color space.
graphics.ColorSpace = PdfColorSpace.GrayScale;
graphics.DrawRectangle(pen, brush, rectangle);

// CMYK color space.
graphics.ColorSpace = PdfColorSpace.CMYK;
graphics.DrawRectangle(pen, brush, rectangle);
graphics.Restore();

// Default color space.
graphics.DrawRectangle(pen, brush, rectangle);

```

[VB .NET]

```

Dim pen As Syncfusion.Pdf.Graphics.PdfPen = New
Syncfusion.Pdf.Graphics.PdfPen(Color.Red)

```

```

Dim brush As Syncfusion.Pdf.Graphics.PdfBrush = New
Syncfusion.Pdf.Graphics.PdfSolidBrush(Color.Blue)

' Default color space.
graphics.DrawRectangle(pen, brush, rectangle)

graphics.Save()

' GrayScale color space.
graphics.ColorSpace = PdfColorSpace.GrayScale
graphics.DrawRectangle(pen, brush, rectangle)

' CMYK color space.
graphics.ColorSpace = PdfColorSpace.CMYK
graphics.DrawRectangle(pen, brush, rectangle)
graphics.Restore()

' Default color space.
graphics.DrawRectangle(pen, brush, rectangle)

```



Note: You may change the color space as many times as you wish, however, you cannot alternate the color space for objects that have been saved before.

4.1.2.2.3 Images

Essential PDF supports both raster and vector images. It supports the following image formats:

- Bmp
- Jpeg
- Gif
- Png
- Tif
- Wmf
- Emf and
- Emf+

Images are supported through the **PdfImage** class, which is an abstract base class that provides the common functionality for **PdfBitmap** and **PdfMetafile** classes. There are static methods in **PdfImage** providing the capability to create a **PdfImage** instance from different sources.

PdfImage as well as graphics elements support layouting multiple pages (see [Pagination](#)).

Base Properties

- **Height**-Specifies image height in pixels
- **Width**-Specifies image width in pixels
- **HorizontalResolution**-Specifies horizontal image resolution, which is also known as DpiX
- **VerticalResolution**-Specifies vertical image resolution, also known as DpiY
- **PhysicalDimension**-Specifies image dimension in points
- **Quality**-Gets or sets the quality



Note: *Image quality is 100 by default, which increases the resultant file size and quality. Reducing the quality will reduce the file size.*

Bitmaps

PdfBitmap class provides functionality of raster images described above. Masks and alpha channels are supported. There are two different kinds of masks: color mask, which is implemented by the **PdfColorMask** class, and image mask, which is implemented by the **PdfImageMask** class. Masks are set by using the **Mask** property of the PdfBitmap object.

The active frame is chosen if the image is a multiframe image (Gif, Tif). The **FrameCount** and **ActiveFrame** properties enable to control the active frame.

Metafiles

All three types of windows metafiles are supported by Essential PDF through the **PdfMetafile** class. Additionally, **Rich Text Format** (RTF) is supported through the PdfMetafile class. PdfMetafile supports multipage layout as well as graphic elements. Additionally, it supports splitting of text lines through the page breaks, if the text line is shared by pages. **PdfMetafileLayoutFormat** should be used when the **Draw** method is called for handling this feature.

While rendering a large meta file with images and text in a PDF document, which has page breaks, you will notice that the images and text are not split across the page breaks. This is achieved by disabling the **SplitTextLines** and **SplitImages** properties of the **PdfMetafileLayoutFormat** class as follows.

[C#]

```
PdfMetafileLayoutFormat format = new PdfMetafileLayoutFormat();
format.SplitTextLines = false;
```

```
format.SplitImages = false;
```

[VB .NET]

```
Dim format As New PdfMetafileLayoutFormat()
format.SplitTextLines = False
format.SplitImages = False
```

The following are the public properties of the PdfMetafileLayoutFormat class.

Name	Description
Break	Gets or sets break type of the element.
Layout	Gets or sets layout type of the element.
PaginateBounds	Gets or sets the bounds on the next page.
SplitImages	Gets or sets the value indicating whether the images should be split between the pages or not.
SplitTextLines	Gets or sets the value indicating whether the text line should be split between the pages or not.
UsePaginateBounds	Gets a value that indicates whether PaginateBounds should be used or not.

Color Spaces

Images retain their original color space. The supported color spaces are as follows.

- **RGB** - Images with 24-bit color space
- **CMYK** - Images with 48-bit color space
- **Grayscale** - Images with 8-bit color space
- **Indexed** - Images with 8-bit indexed color space

Transparency

Transparency of PdfBitmap images are provided by the abstract PdfMask base class.

PdfImageMask

Using grayscale or monochrome images in PdfImageMask enables to create transparent images depending on the pixel format.

- Soft Mask

A soft mask specifies a transparency level for each pixel of the image. You can create these masks from a grayscale image. The level of gray indicates the level of transparency.

- Hard Mask

A hard mask classifies pixels based on their transparency. You can create these masks from a monochrome image.



Note: *Image masks should be of the same width and height as the base image.*

PdfColorMask

This mask enables masking colors (making them transparent) by specifying the range of colors. All colors that exist between the start color and the end color will be transparent.



Note: *According to PDF References, it is recommended not to use JPEG images with color key masking.*

Image Quality

The **Quality** property is used to specify the quality value for images. An image can have its quality set from 0 through 100; 100 the default value increases the resultant file size and quality. Reducing the quality of an image will reduce the corresponding file size.

This property is also used as an encoding parameter while saving an image to PDF format.

[C#]

```
//Setting the quality of the image.
PdfBitmap image=new PdfBitmap(fileName);
image.Quality=80;
```

[VB .NET]

```
'Setting the quality of the image.
Dim image As New PdfBitmap(fileName)
image.Quality=80
```

4.1.2.2.4 Barcode

Bar codes provide a simple and inexpensive method of encoding text information that is easily read by inexpensive electronic readers. A bar code consists of a series of parallel, adjacent bars and spaces. Predefined bar and space patterns or "symbologies" are used to encode small strings of character data into a printed symbol.

The basic structure of a bar code consists of a leading and trailing quiet zone, a start pattern, one or more data characters, optionally one or two check characters, and a stop pattern. Essential PDF supports 1D / linear barcodes and 2D barcode.

1D / Linear Barcodes

Following codes are the 1D / linear barcodes. This section also includes details of each code with coding examples.

- Code39
- Code39Extended
- Code11
- Codabar
- Code32
- Code93
- Code93Extended
- Code128
- Code128A
- Code128B
- Code128C

Code39

The Code 39 character set includes the digits 0-9, the letters A-Z (upper case only), and the following symbols: space, minus (-), plus (+), period (.), dollar sign (\$), slash (/), and percent (%). A special start / stop character is placed at the beginning and end of each barcode. The barcode may be of any length, although more than 25 characters really begin to push the bounds. Code 39 is just about the only type of barcode in common use that does not require a checksum.

The following code example illustrates how to draw Code39 Barcode.

```
[C#]

// Drawing Code39 barcode
PdfCode39Barcode barcode = new PdfCode39Barcode();
```

```
// Setting height of the barcode
barcode.BarHeight = 45;
barcode.Text = "CODE39$";

// Printing barcode on to the Pdf.
barcode.Draw(page, new PointF(25, 70));
```

[VB.NET]

```
' Drawing Code39 barcode
Dim barcode As PdfCode39Barcode = New PdfCode39Barcode()

' Setting height of the barcode
barcode.BarHeight = 45
barcode.Text = "CODE39$"

' Printing barcode on to the Pdf.
barcode.Draw(page, New PointF(25, 70))
```

ExtendedCode39

Code 39 Extended is an extended version of Code 39 that supports the ASCII character set. So with Code 39 Extended, you can also code the 26 lower letters (a-z) and the special characters you have on your keyboard.

The following code example illustrates how to draw Code39Extended barcode.

[C#]

```
// Drawing Code39Extended barcode
PdfCode39ExtendedBarcode barcodeExt = new PdfCode39ExtendedBarcode();
barcodeExt.TextAlignment = PdfBarcodeTextAlignment.Left;
barcodeExt.Text = "CODE39Ext";
//Printing barcode on to the Pdf.
barcodeExt.Draw(page, new PointF(25, 200));
```

[VB.NET]

```
' Drawing Code39Extended barcode
Dim barcodeExt As PdfCode39ExtendedBarcode = New PdfCode39ExtendedBarcode()
barcodeExt.TextAlignment = PdfBarcodeTextAlignment.Left
barcodeExt.Text = "CODE39Ext"
```

```
'Printing barcode on to the Pdf.
barcodeExt.Draw(page, New PointF(25, 200))
```

Code 11

Code 11 is used primarily for labeling telecommunications equipment. The character set includes the digits 0 to 9, a dash (-), and a start / stop code. Each character is encoded with three bars and two spaces. Of these five elements, there may be two wide and three narrow elements, or one wide and four narrow elements.

The following code example illustrates how to draw Code 11 Barcode.

[C#]

```
// Drawing Code 11 barcode
PdfCode11Barcode barcode11 = new PdfCode11Barcode();
barcode11.Text = "012345678";
barcode11.EncodeStartStopSymbols = true;
//Printing barcode on to the Pdf.
barcode11.Draw(page, new PointF(25, 300));
```

[VB.NET]

```
' Drawing Code 11 barcode
Dim barcode11 As PdfCode11Barcode = New PdfCode11Barcode()
barcode11.Text = "012345678"
barcode11.EncodeStartStopSymbols = True
'Printing barcode on to the Pdf.
barcode11.Draw(page, New PointF(25, 300))
```

CodaBar

CodaBar is a variable length symbology that performs encoding of the following 20 characters:

0123456789-\$/.+ABCD.

CodaBar uses the characters, A, B, C and D, only as start and stop characters. Codabar is used in libraries, blood banks, the overnight package delivery industry, and a variety of other information processing applications.

The following code example illustrates how to draw Codabar barcode.

[C#]

```
// Drawing Codabar barcode
PdfCodabarBarcode codabar = new PdfCodabarBarcode();
codabar.Text = "0123";
//Printing barcode on to the Pdf.
codabar.Draw(page, new PointF(25, 400));
```

[VB.NET]

```
' Drawing Codabar barcode
Dim codabar As PdfCodabarBarcode = New PdfCodabarBarcode()
codabar.Text = "0123"
'Printing barcode on to the Pdf.
codabar.Draw(page, New PointF(25, 400))
```

Code 32

It is mainly used for coding pharmaceuticals, cosmetics and dietetics. Code 32 is mainly used to encode pharmaceutical products in Italy. Code 32 is used to encode Italian Pharmacode, which has the following structure:

- 'A' character (ASCII 65), which is not really encoded
- 8 digits for Pharmacode (It generally begins / is prefixed with 0)
- 1 digit for Checksum module 10, which is automatically calculated by Barcode Professional

The value to be encoded (that is passed to Barcode Professional), must be 8 digits pharmacode (prefix it with '0' if necessary), because the 9th digit (the checksum) is automatically calculated by Barcode Professional products.

[C#]

```
PdfCode32Barcode code32 = new PdfCode32Barcode();

code32.Font = font;

// Setting height of the barcode
code32.BarHeight = 45;
code32.Text = "01234567";
code32.TextDisplayLocation = TextLocation.Bottom;
code32.EnableCheckDigit = true;
code32.ShowCheckDigit = true;
```

```
//Printing barcode on to the Pdf.
code32.Draw(page, new PointF(25, 500));
```

[VB.NET]

```
Dim code32 As New PdfCode32Barcode()

code32.Font = font

' Setting height of the barcode
code32.BarHeight = 45
code32.Text = "01234567"
code32.TextDisplayLocation = TextLocation.Bottom
code32.EnableCheckDigit = True
code32.ShowCheckDigit = True

'Printing barcode on to the Pdf.
code32.Draw(page, New PointF(25, 500))
```

Code93

Code 93 was designed to complement and improve upon Code 39. It can represent the full ASCII character set by using combinations of 2 characters. Code 93 is a continuous, variable-length symbology and produces denser code.

- The Standard Mode (default implementation) can encode uppercase letters (A through Z), digits (0 through 9), and special characters like the *, -, \$, %, (Space), ., /, and +.
- The Full ASCII Mode or Extended Version can encode all 128 ASCII characters.

The asterisk (*) is not a true encodable character, but is the start and stop 'symbol' for Code 93.

[C#]

```
PdfCode93Barcode code93 = new PdfCode93Barcode();
// Setting height of the barcode
code93.BarHeight = 45;
code93.Text = "ABC 123456789";
//Printing barcode on to the Pdf.
code93.Draw(page, new PointF(25, 600));
```

[VB .NET]

```
Dim code93 As New PdfCode93Barcode()
' Setting height of the barcode
code93.BarHeight = 45
code93.Text = "ABC 123456789"
'Printing barcode on to the Pdf.
code93.Draw(page, New PointF(25, 600))
```

Code93Extended**[C#]**

```
PdfCode93ExtendedBarcode code93ext = new PdfCode93ExtendedBarcode();
//Setting height of the barcode
code93ext.BarHeight = 45;
code93ext.EncodeStartStopSymbols = true;
code93ext.Text = "(abc) 123456789";

//Printing barcode on to the Pdf.
page = doc.Pages.Add();
code93ext.Draw(page, new PointF(25, 50));
```

[VB .NET]

```
Dim code93ext As New PdfCode93ExtendedBarcode()
'Setting height of the barcode
code93ext.BarHeight = 45
code93ext.EncodeStartStopSymbols = True
code93ext.Text = "(abc) 123456789"

'Printing barcode on to the Pdf.
page = doc.Pages.Add()
code93ext.Draw(page, New PointF(25, 50))
```

Code128

Code 128 is a variable length, high density, alphanumeric, linear bar code symbology, capable of encoding the full 128-character ASCII character set and extended character sets. This Symbology includes a checksum digit for verification, and the barcode may also be verified character-by-character verifying the parity of each data byte.

Code 128 Code Sets

- Code Set A (or Chars Set A) includes all of the standard upper case U.S. alphanumeric keyboard characters and punctuation characters together with the control characters, (i.e. characters with ASCII values from 0 to 95 inclusive), and seven special characters.
- Code Set B (or Chars Set B) includes all of the standard upper case alphanumeric keyboard characters and punctuation characters together with the lower case alphabetic characters (i.e. characters with ASCII values from 32 to 127 inclusive), and seven special characters.
- Code Set C (or Chars Set C) includes the set of 100 digit pairs from 00 to 99 inclusive, as well as three special characters. This allows numeric data to be encoded as two data digits per symbol character, at effectively twice the density of standard data.

Code 128 Special characters

The last seven characters of Code Sets A and B (character values 96 - 102) and the last three characters of Code Set C (character values 100 - 102) are special non-data characters with no ASCII character equivalents, which have particular significance to the bar code reading device.



Note: If you specify that the data must be encoded by using Char Set C, then the number of characters after it must be even.

Code128A

[C#]

```
PdfCode128ABarcode barcode128A = new PdfCode128ABarcode();
// Setting height of the barcode
barcode128A.BarHeight = 45;
barcode128A.Text = "ABCD 12345";
barcode128A.EnableCheckDigit = true;
barcode128A.EncodeStartStopSymbols = true;
barcode128A.ShowCheckDigit = true;

//Printing barcode on to the Pdf.
barcode128A.Draw(page, new PointF(25, 135));
```

[VB .NET]

```
Dim barcode128A As New PdfCode128ABarcode()
' Setting height of the barcode
barcode128A.BarHeight = 45
barcode128A.Text = "ABCD 12345"
barcode128A.EnableCheckDigit = True
barcode128A.EncodeStartStopSymbols = True
barcode128A.ShowCheckDigit = True
```

```
'Printing barcode on to the Pdf.
barcode128A.Draw(page, New PointF(25, 135))
```

Code128B**[C#]**

```
PdfCode128BBarcode barcode128B = new PdfCode128BBarcode();
// Setting height of the barcode
barcode128B.BarHeight = 45;
barcode128B.Text = "12345 abcd";
barcode128B.EnableCheckDigit = true;
barcode128B.EncodeStartStopSymbols = true;
barcode128B.ShowCheckDigit = true;
```

[VB .NET]

```
Dim barcode128B As New PdfCode128BBarcode()
' Setting height of the barcode
barcode128B.BarHeight = 45
barcode128B.Text = "12345 abcd"
barcode128B.EnableCheckDigit = True
barcode128B.EncodeStartStopSymbols = True
barcode128B.ShowCheckDigit = True
```

Code128C**[C#]**

```
PdfCode128CBarcode barcode128C = new PdfCode128CBarcode();
// Setting height of the barcode
barcode128C.BarHeight = 45;
barcode128C.Text = "001122334455";
barcode128C.EnableCheckDigit = true;
barcode128C.EncodeStartStopSymbols = true;
barcode128C.ShowCheckDigit = true;
```

[VB .NET]

```
Dim barcode128C As New PdfCode128CBarcode()
' Setting height of the barcode
```

```
barcode128C.BarHeight = 45
barcode128C.Text = "001122334455"
barcode128C.EnableCheckDigit = True
barcode128C.EncodeStartStopSymbols = True
barcode128C.ShowCheckDigit = True
```

2D Barcode

DataMatrix Barcode

DataMatrix barcode is a two dimensional barcode that consists of a grid of dark and light dots or blocks forming square or rectangular symbol. The data encoded in the barcode can either be number or alphanumeric. The **PdfDataMatrixBarcode** class available in Syncfusion.Pdf.Barcode namespace sets the suitable encoding type and size for the input data. However, the size, encoding type and dimension of individual blocks can also be set using the properties.



By default, the width of the quiet zone on all four sides of the barcode is equal to the dimension of the blocks.

Use case scenario

The DataMatrix bar codes are widely used in printed media such as labels and letters. It can be read easily by a bar code reader and also by mobile phones.

Properties, Methods and Events

Properties

Table 1: Properties Table

Property	Description	Data Type
Encoding	Gets or sets the encoding type.	PdfDataMatrixEncoding
Size	Gets or sets the size of the symbol.	PdfDataMatrixSize
Text	Gets or sets the data.	String
XDimension	Gets or sets the dimension of the blocks	float

Methods

Table 2:Methods Table

Method	Description	Parameters	Return Type
Draw	Draws barcode in PdfPage	(PdfPage, PointF)	Void

Tolimage	Converts barcode to Image	None	Image
----------	---------------------------	------	-------

The following code snippet illustrates how to draw a DataMatrix barcode:

[C#]

```
// Create a DataMatrix barcode.
PdfDataMatrixBarcode dataMatrix = new PdfDataMatrixBarcode("Syncfusion");

// Set the dimension.
dataMatrix.XDimension = 3;

// Set the encoding.
dataMatrix.Encoding = PdfDataMatrixEncoding.ASCII;

// Choose the matrix size.
dataMatrix.Size = PdfDataMatrixSize.Size12x12;

// Draw the barcode on PdfPage.
dataMatrix.Draw(page, PointF.Empty);
```

[VB.NET]

```
' Create a DataMatrix barcode
Dim dataMatrix As New PdfDataMatrixBarcode("Syncfusion")

' Set the dimension
dataMatrix.XDimension = 3

' Set the encoding
dataMatrix.Encoding = PdfDataMatrixEncoding.ASCII

' Choose the matrix size
dataMatrix.Size = PdfDataMatrixSize.Size12x12
```

```
' Draw the barcode on PdfPage
dataMatrix.Draw(page, PointF.Empty)
```

Barcode as Image

Barcode that is generated by using Essential PDF is simultaneously converted to an image. The following code example illustrates this.

[C#]

```
Image img = barcode.ToImage();
img.Save("Code38Barcode.png", ImageFormat.Png);
```

[VB.NET]

```
Private img As Image = barcode.ToImage()
img.Save("Code38Barcode.png", ImageFormat.Png)
```

4.1.2.2.5 QR Barcode

A QR Barcode is a two-dimensional barcode that consists of a grid of dark and light dots or blocks that form a square. The data encoded in the barcode can be numeric, alphanumeric, or JIS8 characters. The **PdfQRBarcode** class available in Syncfusion.Pdf.Barcode namespace sets the suitable version, Error correction level and size for the input data. However, the version, Error correction level and size of individual blocks can also be set using the properties.

4.1.2.2.5.1 Use Case Scenarios

The QR Barcodes are widely used in printed media such as labels and letters. It can be read easily by a bar code reader and also by mobile phones.

4.1.2.2.5.2 Properties

Table 3: Properties Table

Property	Description	Data Type
Text	Gets or sets the data.	String

Version	Gets or sets the QR barcode version.	QRCodeVersion
ErrorCorrectionLevel	Gets or sets the error correction level.	PdfErrorCorrectionLevel
InputMode	Gets or sets the mode of the input text.	InputMode
XDimension	Gets or sets the dimension of the blocks.	Float

4.1.2.2.5.3 Methods

Table 4: Methods Table

Method	Description	Parameters	Return Type
Draw	Draws barcode in PdfPage.	(PdfPage, PointF)	Void
ToImage	Converts barcode to image.	None	Image

4.1.2.2.5.4 Version

The QR Barcode uses version levels numbered from 1 to 40. Version 1 measures 21 modules x 21 modules, Version 2 measures 25 modules x 25 modules, and so on. The number of modules increases in steps of 4 modules per side up to Version 40, which measures 177 modules x 177 modules. Each version has its own capacity. By default the QR Version is Auto, which will automatically set the version according to the length of the input text.

4.1.2.2.5.5 Error correction level

The QR Barcode employs error correction to generate a series of error correction codewords which are added to the data codeword sequence. This enables the symbol to withstand damage without loss of data. There are four user-selectable levels of error correction, as shown in the table, offering the capability of recovery from the following amounts of damage. By default, the Error correction level is Low (L).

Table 3: Error Correction Level Table

Error Correction Level	Recovery Capacity % (approx.)
------------------------	-------------------------------

L	7
M	15
Q	25
H	30

4.1.2.2.5.6 Input mode

There are three modes for the input as defined in the table below. Each mode supports the specific set of input characters. The user may select the most suitable input mode. The default input mode is Binary Mode.

Table 4: Input Mode Table

Input Mode	Possible characters
Numeric Mode	0,1,2,3,4,5,6,7,8,9
Alphanumeric Mode	0–9, A–Z (upper-case only), space, \$, %, *, +, -, ., /, :
Binary Mode	Shift JIS characters

4.1.2.2.5.7 Adding a QR Barcode to an application

[C#]

```
//Create a QR Barcode.
PdfQRBarcode QRBarcode = new PdfQRBarcode();

//Set the dimension.
QRBarcode.XDimension = 8;

//Set the input mode.
QRBarcode.InputMode = InputMode.BinaryMode;

//Set the error correction level.
QRBarcode.ErrorCorrectionLevel =
PdfErrorCorrectionLevel.High;
QRBarcode.Text = "Syncfusion";

//Set the barcode version.
QRBarcode.Version = QRCodeVersion.Auto;
```

```
//Draw the barcode on PdfPage.
QRBarcode.Draw(page, PointF.Empty);
```

[VB.NET]

```
'Create a QR Barcode.
Dim QRBarcode As New PdfQRBarcode()

'Set the dimension.
QRBarcode.XDimension = 8

'Set the input mode.
QRBarcode.InputMode = InputMode.BinaryMode

'Set the error correction level.
QRBarcode.ErrorCorrectionLevel = PdfErrorCorrectionLevel.High
QRBarcode.Text = "Syncfusion"

'Set the barcode version.
QRBarcode.Version = QRCodeVersion.Auto

'Draw the barcode on PdfPage.
QRBarcode.Draw(page, PointF.Empty)
```

4.1.2.2.6 Pagination

Pagination is the capability of the elements to span more than one page (to be shared by more than one page). The base class for all such elements is the **PdfLayoutElement** class. All the primitives that are derived from that class support page layouting in their own way. The following events are raised by this class.

Name	Description
BeginPageLayout	This event is raised before the element is printed on the page.
EndPageLayout	This event is raised after the element is printed on the page.

The layouting is accomplished by using the following methods.

[C#]

```
public PdfLayoutResult Draw( PdfPage page, PointF location, PdfLayoutFormat format );
public PdfLayoutResult Draw( PdfPage page, RectangleF layoutRectangle,
PdfLayoutFormat format );
```

[VB .NET]

```
Public PdfLayoutResult Draw(PdfPage page, PointF location, PdfLayoutFormat format)
Public PdfLayoutResult Draw(PdfPage page, RectangleF layoutRectangle, PdfLayoutFormat
format)
```

The parameters define the page where the layouting should start, a location or bounds on the page, and layouting settings.



Note: If, only the location is set, or the width or height is less than or equal to zero, the bounds are calculated, as the remaining space on the page where the location specified is a Left or Top of the bounds.

The **PdfLayoutFormat** class specifies basic layout settings, such as the type of page break or bounds on another page. The following properties belong to the **PdfLayoutFormat** class.

Name	Description
Break	Gets or sets break type of the element.
Layout	Gets or sets layout type of the element.
PaginateBounds	Gets or sets the bounds on the next page.
UsePaginateBounds	Gets a value that indicates whether PaginateBounds should be used or not.

 Note:

- *PdfLayoutFormat contains the PaginateBounds property that specifies the bounds of the element on the pages that follows. If this property is set, the element will use it for laying out the next pages, otherwise the element will be laid out according to the bounds used on the first page (layoutRectangle parameter or calculated as was mentioned above), but with the Y coordinate set to zero (0).*
- *Each element may implement its own laying out settings depending on its own structure and specifications.*
- *The objects supporting page laying out can be drawn on simple graphics or by using the following methods.*

[C#]

```
public void Draw( PdfGraphics graphics );
public void Draw( PdfGraphics graphics, PointF location );
```

[VB .NET]

```
Public void Draw(PdfGraphics graphics)
Public void Draw(PdfGraphics graphics, PointF location)
```

After laying out, each element returns a lay out result, depending on the element and the settings. The base result is described by the **PdfLayoutResult** class.

The basic information stored in result is the page where the element ends, and the bounds of the element on that page, which might be helpful for further drawing operations on this page.

The following are the properties of the **PdfLayoutResult** class.

Name	Description
Bounds	Gets the bounds of the element on the last page where it was drawn.
Page	Gets the last page where the element was drawn.

For example, **PdfTextLayoutResult** provides a text that was not laid out and a width of the last laid out line. **PdfMetafile** as well as graphics primitives also supports multipage layout. Additionally, if it is required to eliminate the splitting of text lines between the pages, the **PdfMetafileLayoutFormat** is used as the input parameter of the **Draw** method. This class contains a property, which allows to enable or disable splitting of text lines.

There are two events provided by the laid out elements: **BeginPageLayout** and **EndPageLayout**. They provide an option to track the current state of the layout, and specify the custom settings for the layout process.

4.1.2.2.7 Page Templates

Page Templates define similar graphics primitives for a range of pages. Each page has two sources of the templates (template on the document holding the section of the page, and template on the section containing the page) that apply sequentially.

Each page template has four properties for Left, Right, Top and Bottom docked templates, and a collection of the additional template elements (stamps). Also, there are eight additional templates that you can add to the odd / even pages (EvenTop, OddTop, etc.). If one of these eight templates is set, it overrides its usual template (OddTop overrides Top, etc.); Otherwise, the usual template is used.



Note: A **PdfPageTemplateElement** is added as one template. It can be assigned to **Left**, **Right**, **Top** or **Bottom** only once.

Using the Page Templates

If you want to define some graphics content to all pages of the document, use the **Template** property of the **PdfDocument** class. You can define Left, Top, Right, Bottom templates, as well as arbitrary quantity of the other templates (stamps) that can be used for water marking or stamping of the pages.

If you have decided to use some custom templates for a specified range of the page, use **Template** property of the **PdfSection** class containing these pages. It involves the same functionality as in the PdfDocument class. Additionally, you can disable or enable the document templates from the section.

Default Behavior

Document templates are enabled, by default.



Note: Section template, which is printed over the parent template, does not replace document templates. If you want to insert a watermark or stamp on the page, use **Stamps** property of the **PdfDocumentTemplate** class.

Behavior

PdfPageTemplateElement class has the functionality of aligning (use **Alignment** property) and docking (use **Dock** property) of this class. Docking to the Left, Top, Right, Bottom is implemented similar to Windows Forms Docking functionality (Top and Bottom have priority). Docking stamp elements do not have any priorities and the appearance depends on their order in the collection.



Note: *Alignment has higher priority than Docking in the template element, but Docking resets the Alignment.*

Each template element which is docked, sticks to its appropriate side of the page. It stretches itself according to the dimensions of the page and resets the alignment. Avoid printing any content that can be stretched in cases where the pages have different sizes. Also, define the size of the template elements according to its docking style on the page. But, if you want to use some template element as Left, Top, Right or Bottom, but do not want the element to be stretched, you can set the **Alignment** property, once you assigned the template element (or set Dock property) to any of the mentioned properties. In this case, the template will stick to the appropriate position but will not be stretched.



Note: *In this scenario, you can set the Alignment property to the appropriate side only (depending on the Dock style). For example, if you set some template element as Top, the allowed values for Alignment are: TopLeft, TopCenter and TopRight. You cannot set any other value attributing to the possible inconsistency with Docking style.*

Z-Order of the Layers

Each page can contain page templates from the document and from the parent section. Also, it can contain its own layers. The order of the layers is as follows (from back to top):

1. Document page templates (Left, Top, Right, Bottom) which have Background property set to **True**.
2. Document stamp elements which have Background property set to **True**.
3. Section page templates (Left, Top, Right, Bottom) which have Background property set to **True**.
4. Section stamp elements which have Background property set to **True**.
5. Page layers.
6. Section page templates (Left, Top, Right, Bottom) which have Foreground property set to **True**.
7. Section stamp elements which have Foreground property set to **True**.
8. Document page templates (Left, Top, Right, Bottom) which have Foreground property set to **True**.
9. Document stamp elements which have Foreground property set to **True**.

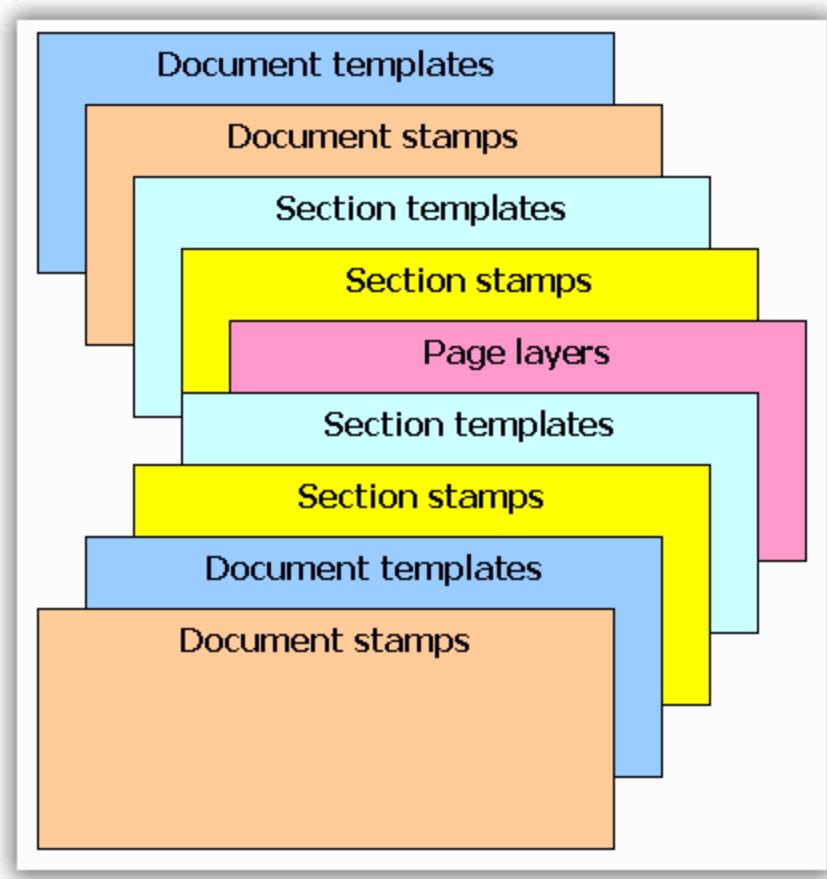


Figure 27: Layers

4.1.2.3 Tables

Tables are useful for presenting a large quantity of information clearly and concisely. In PDF, tables are drawn as a series of rectangles with text and image correctly positioned within them. Using Essential PDF, it is drawn with the help of PdfBrush, PdfPen and other PdfGraphics elements. Essential PDF also offers two classes to achieve them without any hassle. They are:

- [PdfLightTable](#)
- [PdfGrid](#)

PdfLightTable

It allows the creation of table with inputs from DataTable, arrays or any other entity class. It allows you to perform simple formatting using events. As this class allows minimal customization options, rendering will be faster than PDF Grid and is recommended to draw a simple table. Check the following comparison table for more details.

PdfGrid

It is based on cell model that offers rich API for formatting and layout options. It can take input from DataTable, arrays or any other entity class. Formatting can be done to all levels of PdfGrid. More features like Nested Table, Row and Column Spanning are also supported. It offers full control over the appearance and is recommended to draw complex table structures. Check the following comparison table for more details.

Comparison between PdfLightTable and PdfGrid

Platforms	PdfLightTable	PdfGrid
Windows Forms	Yes	Yes
WPF	Yes	Yes
ASP.NET	Yes	Yes
ASP.NET MVC	Yes	Yes
Silverlight	Yes	Yes

Features	PdfLightTable	PdfGrid
Formatting		
Row	No direct API, possible through events	Yes
Column	Yes (StringFormat)	Yes (StringFormat)
Cell	No direct API for single cell formatting, possible through events	Yes
Others		
Row span	No	Yes
Column span	No direct API, possible through events	Yes
Nested Grid	Possible through events	Yes
Layout Events	BeginCellLayout, BeginPageLayout, BeginRowLayout, EndCellLayout, EndPageLayout, EndRowLayout	BeginPageLayout, EndPageLayout

4.1.2.3.1 PdfLightTable

The PdfLightTable class represents simple tables that are used for publishing structured data from arrays, data tables or data columns. There are no real cells or rows, and all the data is taken from the data source (DataSource property).

Using PdfLightTable, any type of table can be created using its event handlers. PdfLightTable in Silverlight supports IEnumerable and TableDirect data sources and not arrays, data tables or data columns. This section explains how to draw PdfLightTable elements using Essential PDF. It includes the following topics:

4.1.2.3.1.1 Properties, Methods and Events

Properties

Name	Description	Data Type
AllowRowBreakAcrossPages	Gets or sets a value indicating the row break is to be made or not	Boolean
Columns	Gets the columns	PdfColumnCollection
DataMember	Gets or sets the data member	String
DataSource	Gets or sets the data source	Object
DataSourceType	Gets or sets the data source type of the PdfLightTable	PdfLightTableDataSourceType
IgnoreSorting	Gets or sets a value indicating whether PdfLightTable should ignore sorting in the DataTable	Boolean
Rows	Gets the rows	PdfRowCollection
Style	Gets or sets the properties	PdfLightTableStyle

Methods

Method	Description	Parameters	Return Type
Draw	Draws PdfLightTable	Overloads: (PdfGraphics graphics)	Void
		(PdfPage page, PointF location)	PdfLightTableLayoutResult
		(PdfPage page, RectangleF bounds)	PdfLightTableLayoutResult
		(PdfGraphics graphics, PointF location)	Void
		(PdfGraphics graphics, RectangleF bounds)	Void

	(PdfPage page, float x, float y)	PdfLightTableLayoutResult
	(PdfPage page, PointF location, PdfLightTableLayoutFormat format)	PdfLightTableLayoutResult
	(PdfPage page, PointF location, PdfLayoutFormat format)	PdfLayoutResult
	(PdfPage page, RectangleF bounds, PdfLightTableLayoutFormat format)	PdfLightTableLayoutResult
	(PdfPage page, PointF location, PdfLayoutFormat format)	PdfLayoutResult
	(PdfGraphics graphics, float x, float y)	void
	(PdfGraphics graphics, PointF location, float width)	void
	(PdfPage page, float x, float y, float width)	PdfLightTableLayoutResult
	(PdfPage page, float x, float y, PdfLightTableLayoutFormat format)	PdfLightTableLayoutResult
	(PdfPage page, float x, float y, PdfLayoutFormat format)	PdfLayoutResult
	(PdfGraphics graphics, float x, float y, float width)	void
	(PdfPage page, float x, float y, float width, PdfLightTableLayoutFormat format)	PdfLightTableLayoutResult

Events

Name	Description
BeginCellLayout	This event is raised on starting cell layout.
BeginPageLayout	This event is raised before the element is printed on the page. (Inherited from PdfLayoutElement.)
BeginRowLayout	This event is raised on starting row layout.
EndCellLayout	This event is raised on having finished cell layout.
EndPageLayout	This event is raised after the element is printed on the page. (Inherited from PdfLayoutElement.)

EndRowLayout	This event is raised on finishing row layout.
QueryColumnCount	This event is raised when the column number is requested.
QueryNextRow	This event is raised when the next row data is requested.
QueryRowCount	This event is raised when the row number is requested.

4.1.2.3.1.2 PdfLightTable Creation

Note: You should add the `Syncfusion.Pdf.Tables` namespace to work with `PdfLightTable`.

You may create a `PdfLightTable` simply by specifying a new operator with the proper constructor. After assigning the data source, it can be drawn using one of the overloads of `Draw` method as follows:

[C#]

```
// Creating a PdfLightTable.
PdfLightTable pdfLightTable = new PdfLightTable();

// Assigning data source.
pdfLightTable.DataSource = dataSource;

// Drawing PdfLightTable.
pdfLightTable.Draw(graphics);
```

[VB.NET]

```
' Creating a PdfLightTable.
Dim pdfLightTable As New PdfLightTable()

' Assigning data source.
pdfLightTable.DataSource = dataSource

' Drawing PdfLightTable.
pdfLightTable.Draw(graphics)
```

Different types of data can be set to `PdfLightTable`. Also, the `draw` method facilitates overloads that would help you to layout the `PdfLightTable` as required. The topics that discuss them are:

- [Data](#)
- [Layout](#)

Ignore Sorting

If there is a DataTable assigned as a data source, you may specify whether you need sorted or unsorted data. If this property is set to True, the DataTable sorting will be ignored.

Note: *Sorting is disabled, by default. This is because sorted data takes more time.*

AllowRowBreakAcrossPages

This property allows changing the row split behavior across pages. By default, this Boolean property is set to True, which allows splitting the row across pages when the row cannot accommodate within the bounds of the page. If set to false, the entire row will be shifted to the next page.

Note: *If the row height is greater than the page height, it will be forcibly split across pages ignoring this property.*

4.1.2.3.1.2.1 Data

You can initialize the data to the PdfLightTable with the help of the DataSource property. This is achieved through:

- External DataSource: By assigning the External data source
- Table Direct : By adding columns and rows to the PdfLightTable
- Event Handlers : By adding columns and rows by using the event handlers

4.1.2.3.1.2.1.1 External DataSource

Data source is an object that might be an array (two-dimensional, one-dimensional or nested), a DataTable, DataColumn, DataView or DataSet. To draw an external data source, you must set DataSourceType property to PdfLightTableDataSourceType.External.

The following code example illustrates how to assign external data source to PdfLightTable:

[C#]

```
pdfLightTable.DataSourceType = PdfLightTableDataSourceType.External;
pdfLightTable.DataSource = dataTable;
```

[VB .NET]

```
pdfLightTable.DataSourceType = PdfLightTableDataSourceType.External
pdfLightTable.DataSource = dataTable
```

Note: External data source is the default data source.

4.1.2.3.1.2.1.2 Table Direct

You can directly add rows and columns to PdfLightTable. To achieve this, set DataSourceType property to PdfLightTableDataSourceType.TableDirect. The following code example illustrates this:

[C#]

```
PdfLightTable pdfLightTable = new PdfLightTable();

// Setting the DataSourceType as Direct
pdfLightTable.DataSourceType = PdfLightTableDataSourceType.TableDirect;

// Creating Columns
pdfLightTable.Columns.Add(new PdfColumn("Roll Number"));
pdfLightTable.Columns.Add(new PdfColumn("Name"));
pdfLightTable.Columns.Add(new PdfColumn("Class"));

// Adding Rows
pdfLightTable.Rows.Add(new object[] {"111", "Maxim", "III"});

// Drawing the PdfLightTable
pdfLightTable.Draw(page, PointF.Empty);
```

[VB .NET]

```
Dim pdfLightTable As PdfLightTable = New PdfLightTable()

' Setting the DataSourceType as Direct
pdfLightTable.DataSourceType = PdfLightTableDataSourceType.TableDirect

' Creating Columns
pdfLightTable.Columns.Add(New PdfColumn("Roll Number"))
pdfLightTable.Columns.Add(New PdfColumn("Name"))
pdfLightTable.Columns.Add(New PdfColumn("Class"))

' Adding Rows
pdfLightTable.Rows.Add(New Object() {"111", "Maxim", "III"})
```

```
' Drawing the PdfLightTable
pdfLightTable.Draw(page, PointF.Empty)
```

4.1.2.3.1.2.1.2.1 IEnumarable

PdfLightTable in Silverlight platform can take input from IEnumarable objects. The following is the code snippet:

[C#]

```
//Creating PdfLightTable.
PdfLightTable pdfLightTable = new PdfLightTable();

//Creating IEnumarable source.
Dictionary<string, string> dictionary = new Dictionary<string,
string>();
dictionary.Add("AAA", "111");
dictionary.Add("BBB", "112");
dictionary.Add("CCC", "113");

pdfLightTable.DataSource = dictionary;

//Draw.
pdfLightTable.Draw(page, PointF.Empty);
```

[VB.NET]

```
'Creating PdfLightTable.
Dim pdfLightTable As New PdfLightTable()

'Creating IEnumarable source.
Dim dictionary As New Dictionary(Of String, String)()
dictionary.Add("AAA", "111")
dictionary.Add("BBB", "112")
dictionary.Add("CCC", "113")

pdfLightTable.DataSource = dictionary

'Drawing the PdfLightTable
pdfLightTable.Draw(page, PointF.Empty)
```

4.1.2.3.1.2.1.3 Event Handlers

Data to PdfLightTable can also be set using the following three events:

- QueryColumnCount – Sets the number of columns.
- QueryRowCount – Sets the number of rows.
- QueryNextRow – Sets data to the PdfLightTable.

Note: These events will act only when the **DataSource** property is not set.

The following code snippet illustrates this:

```
[C#]

public string[][] datastring = new string[2][];

// Giving it some column arrays
datastring[0] = new string[] { "111", "Maxim", "100" };
datastring[1] = new string[] { "222", "Calvin", "95" };

// Creating PdfLightTable
PdfLightTable pdfLightTable = new PdfLightTable();

pdfLightTable.QueryColumnCount += new
QueryColumnCountEventHandler(pdfLightTable_QueryColumnCount);
pdfLightTable.QueryNextRow += new
QueryNextRowEventHandler(pdfLightTable_QueryNextRow);
pdfLightTable.QueryRowCount += new
QueryRowCountEventHandler(pdfLightTable_QueryRowCount);

// Drawing the PdfLightTable
pdfLightTable.Draw(page, PointF.Empty);

// Getting the number of columns
void pdfLightTable_QueryColumnCount(object sender,
QueryColumnCountEventArgs args)
{
    args.ColumnCount = 3;
}

// Getting the number of rows
void pdfLightTable_QueryRowCount(object sender, QueryRowCountEventArgs
args)
{
    args.RowCount = 2;
}

// Getting the row data
```

```

void pdfLightTable_QueryNextRow(object sender, QueryNextRowEventArgs
args)
{
    if (args.RowIndex < datastring.Length)
        args.RowData = new string[] { datastring[args.RowIndex][0],
datastring[args.RowIndex][1], datastring[args.RowIndex][2] };
}

```

[VB.NET]

```

Public datastring(1)() As String

' Giving it some column arrays
Private datastring(0) = New String() { "111", "Maxim", "100" }
Private datastring(1) = New String() { "222", "Calvin", "95" }

' Creating PdfLightTable
Private pdfLightTable As New PdfLightTable()

pdfLightTable.QueryColumnCount += New
QueryColumnCountEventHandler(AddressOf pdfLightTable_QueryColumnCount)
pdfLightTable.QueryNextRow += New QueryNextRowEventHandler(AddressOf
pdfLightTable_QueryNextRow)
pdfLightTable.QueryRowCount += New QueryRowCountEventHandler(AddressOf
pdfLightTable_QueryRowCount)

' Drawing the PdfLightTable
pdfLightTable.Draw(page, PointF.Empty)

' Getting the number of columns
Private Sub pdfLightTable_QueryColumnCount(Object sender,
QueryColumnCountEventArgs args)
    args.ColumnCount = 3
End Sub

' Getting the number of rows
Private Sub pdfLightTable_QueryRowCount(Object sender,
QueryRowCountEventArgs args)
    args.RowCount = 2
End Sub

' Getting row data
Private Sub pdfLightTable_QueryNextRow(Object sender,
QueryNextRowEventArgs args)
    If args.RowIndex < datastring.Length Then
        args.RowData = New String() { datastring(args.RowIndex)(0),

```

```

datastring(args.RowIndex) (1), datastring(args.RowIndex) (2) }
End If
End Sub

```

4.1.2.3.1.2.2 Layout

PdfLightTableLayoutFormat

Layouting PdfLightTable can be done using the PdfLightTableLayoutFormat class. Overloads accepting pages can accept standard formats as other layouting elements. However, they treat the PdfLayoutBreakType.FitElement value of Format.Break property as one, for a single row and not for the entire PdfLightTable.

Properties

Name	Description	Data Type
Break	Gets or sets the break type	PdfLayoutBreakType
EndColumnIndex	Gets or sets the end column index	Integer
Layout	Gets or sets the layout type	PdfLayoutType
PaginateBounds	Gets or sets the PdfLightTable bounds for the following page	RectangleF
StartColumnIndex	Gets or sets the start column index	Integer

Also, you may select a range of column using the PdfLightTableLayoutFormat properties, StartColumnIndex and EndColumnIndex. You should pass an instance of this class to one of the Draw overloads, instead of the PdfLayoutFormat class instance.

[C#]

```

PdfLightTableLayoutFormat format = new PdfLightTableLayoutFormat();
format.StartColumnIndex = 0;
format.EndColumnIndex = 3;

// Draws the PdfLightTable from the first to the fourth column
pdfLightTable.Draw(page, PointF.Empty, format);

```

[VB .NET]

```

Dim format As PdfLightTableLayoutFormat = New PdfLightTableLayoutFormat()
()
format.StartColumnIndex = 0
format.EndColumnIndex = 3

'Drawing the PdfLightTable from the first to the fourth column
pdfLightTable.Draw(page, PointF.Empty, format)

```

The PdfLayoutType class is used to specify the type of pagination. The Paginate LayoutType draws the PdfLightTable on the (immediate) following pages, if the element exceeds the page. The OnePage layout draws the element only on one page. The following code example illustrates this:

[C#]

```

PdfLightTableLayoutFormat format = new PdfLightTableLayoutFormat();
format.Layout = PdfLayoutType.Paginate;
format.Break = PdfLayoutBreakType.FitElement;
format.StartColumnIndex = 1;
format.EndColumnIndex = 2;

// Drawing the PdfLightTable with the layout format
pdfLightTable.Draw(page, PointF.Empty, format);

```

[VB.NET]

```

Dim format As PdfLightTableLayoutFormat
= New PdfLightTableLayoutFormat()
format.Layout = PdfLayoutType.Paginate
format.Break = PdfLayoutBreakType.FitElement
format.StartColumnIndex = 1
format.EndColumnIndex = 2

' Drawing the PdfLightTable with the layout format
pdfLightTable.Draw(page, PointF.Empty, format)

```

PdfLightTableLayoutResult

You can get the layout settings for the drawn PdfLightTable with the help of PdfLightTableLayoutResult class. Also, you can get the bounds and the last page where the PdfLightTable is drawn using the Bounds and Page properties. This is mainly used to write some text or any other element below the large table that shows the number of pages.

Properties

Name	Description	Data Type
Bounds	Gets the bounds in the last page where it was drawn	RectangleF
LastRowIndex	Gets the index of the last row	Integer
Page	Gets the last page where PdfLightTable was drawn	PdfPage

[C#]

```
// Drawing the PdfLightTable.
PdfLightTableLayoutResult result =
pdfLightTable.Draw(page, PointF.Empty, format);

// Returning the rectangle value for the last page.
Console.WriteLine(result.Bounds.ToString());
```

[VB.NET]

```
' Drawing the PdfLightTable.
Dim result As PdfLightTableLayoutResult =
pdfLightTable.Draw(page, PointF.Empty, format)

' Returning the rectangle value for the last page.
Console.WriteLine(result.Bounds.ToString())
```

4.1.2.3.1.3 PdfLightTable Formatting

This section talks about the most direct and indirect formatting options (through events) that are possible with PdfLightTable. The PdfLightTableStyle class, accessed through Style property of PdfLightTable instance has a number of properties that allow formatting the entire PdfLightTable or parts of it. Border and few other properties are discussed below. Header, Cell, Row and Column are discussed in the following links:

- Header
- Row
- Column
- Cell

Border

Border for the entire PdfLightTable can be set using the BorderPen property. Also, border styles applied to individual cells might override this value. You can specify the [PdfPen](#) to be used to draw border with any color.

[C#]

```
pdfLightTable.Style.BorderPen = PdfPens.Khaki;
```

[VB .NET]

```
pdfLightTable.Style.BorderPen = PdfPens.Khaki
```

BorderOverlapStyle

This property decides whether the cell border overlaps with neighboring cells or if it should be drawn inside cell.

Note: This property applies for all cells in the PdfLightTable. You need to be careful when using overlapping borders, because they may produce bad results if they are not of the same width and color.

[C#]

```
pdfLightTable.Style.BorderOverlapStyle = PdfBorderOverlapStyle.Overlap;
```

[VB .NET]

```
pdfLightTable.Style.BorderOverlapStyle = PdfBorderOverlapStyle.Overlap
```

Padding and Spacing

You can also specify the cell spacing (distance between cells) and cell padding (distance between cell text and border) for all cells.

[C#]

```
pdfLightTable.Style.CellPadding = 4;
pdfLightTable.Style.CellSpacing = 10;
```

[VB .NET]

```
pdfLightTable.Style.CellPadding = 4
pdfLightTable.Style.CellSpacing = 10
```

4.1.2.3.1.3.1 Header

Header is a set of rows that repeat on each page and has its own style. Rows for the header might be taken either from column captions or from ordinary rows. In the latter case, the rows are treated as headers and do not appear in the body of the PdfLightTable.

[C#]

```
// Header from column captions
pdfLightTable.Style.ShowHeader = true;
pdfLightTable.Style.HeaderSource = PdfHeaderSource.ColumnCaptions;
pdfLightTable.Style.RepeateHeader = true;
pdfLightTable.Style.HeaderStyle = headerStyle;

// Header from rows
pdfLightTable.Style.ShowHeader = true;
pdfLightTable.Style.HeaderSource = PdfHeaderSource.Rows;
pdfLightTable.Style.RepeatHeader = true;
pdfLightTable.Style.HeaderRowCount = 3;
pdfLightTable.Style.HeaderStyle = headerStyle;
```

[VB .NET]

```
' Header from column captions
pdfLightTable.Style.ShowHeader = True
pdfLightTable.Style.HeaderSource = PdfHeaderSource.ColumnCaptions
pdfLightTable.Style.RepeatHeader = True
pdfLightTable.Style.HeaderStyle = headerStyle

' Header from rows
pdfLightTable.Style.ShowHeader = True
pdfLightTable.Style.HeaderSource = PdfHeaderSource.Rows
pdfLightTable.Style.RepeatHeader = True
pdfLightTable.Style.HeaderRowCount = 3
pdfLightTable.Style.HeaderStyle = headerStyle
```

The headerStyle is an instance of [PdfCellStyle](#) that can be set to header row.

4.1.2.3.1.3.2 Row

The values of existing row, entered with `DataSourceType` as `PdfLightTableDataSourceType.TableDirect` can be edited using the `Values` property. The following is the code:

[C#]

```
pdfLightTable.Rows[1].Values = new string[] { "333", "John", "234" };
```

[VB .NET]

```
pdfLightTable.Rows(1).Values = New String() { "333", "John", "234" }
```

The minimum height of a row in `PdfLightTable` can be set using [BeginRowLayout](#) and [EndRowLayout](#) events.

4.1.2.3.1.3.3 Column

Name	Description	Data Type
ColumnName	Gets or sets the column name	String
StringFormat	Gets or sets the string format for the column	PdfStringFormat
Width	Gets or sets the width of the column	float

ColumnName

By default, `PdfLightTable` displays the column text as the `DataSource` column name. You can change the column text with the help of the `ColumnName` property. The following code snippet illustrates this:

[C#]

```
// Specifying Column name
pdfLightTable.Columns[2].ColumnName = "Student Name";
```

[VB .NET]

```
' Specifying Column name
pdfLightTable.Columns(2).ColumnName = "Student Name"
```

StringFormat

The format of the data for a single column can be changed using the StringFormat property. Check String Formatting in [DrawingText](#) for more details.

Width

By default, all the columns in a PdfLightTable have equal width, and the columns automatically fill the entire width of the PdfLightTable. If the width of any of the column(s) is increased or decreased, the width of other columns changes appropriately.

To customize initial column widths, you can invoke the Width property for each column of the PdfLightTable. The following code snippet illustrates this:

[C#]

```
// Setting width for third column
pdfLightTable.Columns[2].Width = 10;
```

[VB .NET]

```
' Setting width for third column
pdfLightTable.Columns(2).Width = 10
```

Note: The unit of the Width property is always points. You can set the PDF units only as points. Also, you can use the [PdfUnitConvertor](#) class to convert the other units to points.

Column Span

You can set column span in PdfLightTable using BeginRowLayout event. Check the following link for more details.

[How To Implement Column Span In PdfLightTable?](#)

4.1.2.3.1.3.4 Cell

You can specify the default cell style by using the DefaultStyle property. The style for the header cells is set by using the HeaderStyle property.

Also, you can specify an alternate style by using the `AlternateStyle` property. This property is used to customize the appearance of the odd row cells.

Properties

Name	Description	Data Type
BackgroundBrush	Gets or sets the brush with which, the background will be drawn	PdfBrush
Border	Gets or sets the pen with which, the border will be drawn	PdfPen
Font	Gets or sets the font	PdfFont
StringFormat	Gets or sets the string format of the text	PdfStringFormat
TextBrush	Gets or sets the brush, which will be used to draw font	PdfBrush
TextPen	Gets or sets the pen, which will be used to draw text outlines	PdfPen

The `Style` property enables you to specify the font along with its appearance (brush, pen and string format), and border along with the background of cells. Although there are fixed properties for styles, you can specify different styles using [BeginCellLayout](#) and [EndCellLayout](#) events.

[C#]

```

PdfCellStyle altStyle = new PdfCellStyle(font, PdfBrushes.White,
PdfPens.Green);
altStyle.BackgroundBrush = PdfBrushes.DarkGray;

PdfCellStyle headerStyle = new PdfCellStyle(font, PdfBrushes.White,
PdfPens.Brown);
headerStyle.BackgroundBrush = PdfBrushes.Red;

pdfLightTable.Style.AlternateStyle = altStyle;
pdfLightTable.Style.HeaderStyle = headerStyle;

```

[VB.NET]

```

Dim altStyle As Syncfusion.Pdf.Tables.PdfCellStyle
= NewSyncfusion.Pdf.Tables.PdfCellStyle(Font, PdfBrushes.White,

```

```

PdfPens.Green)
altStyle.BackgroundBrush = PdfBrushes.DarkGray

Dim headerStyle As Syncfusion.Pdf.Tables.PdfCellStyle
= NewSyncfusion.Pdf.Tables.PdfCellStyle(Font, PdfBrushes.White,
PdfPens.Brown)
headerStyle.BackgroundBrush = PdfBrushes.Red

pdfLightTable.Style.AlternateStyle = altStyle
pdfLightTable.Style.HeaderStyle = headerStyle

```

4.1.2.3.1.4 PdfLightTable Customization

PdfLightTable offers a set of events that help to change the look and feel in the PDF. The following are the list of events and links that discuss them:

4.1.2.3.1.4.1 BeginPageLayout

This event is raised before layout starts on a page. The arguments of this event are as follows.

- Page (read-only): Page on which layout should be performed
- Bounds: Size of the PdfLightTable part, which should be laid out on the page
- Cancel: Enables to cancel layout

4.1.2.3.1.4.2 EndPageLayout

This event is raised when layout on a page finishes. The arguments of this event are as follows:

- Result (read-only): Layout result for the current page
- NextPage: Page on which layout should continue

4.1.2.3.1.4.3 BeginRowLayout

This event is raised when row layout starts. The arguments of this event are as follows:

- RowIndex (read-only): Index of the row (zero based)
- [CellStyle](#) : Style of the cells within the row
- [ColumnSpanMap](#) : Array of integers specifying column span
- Cancel: Enables to cancel layout
- IgnoreColumnFormat: Gets or sets a value indicating whether column string format should be ignored.

- Skip: Enables to skip the entire row
- MinimalHeight: Enables to specify the minimal row height, which is used to preserve space for images

The following code example illustrates how to set the row height.

[C#]

```
//Subscribing the event
pdfLightTable.BeginRowLayout
+= new BeginRowLayoutEventHandler(pdfLightTable_BeginRowLayout);

//Setting row height for the second row
void pdfLightTable_BeginRowLayout(object sender, BeginRowLayoutEventArgs args)
{
    if (args.RowIndex == 1)
    {
        args.MinimalHeight = 25;
    }
}
```

[VB.NET]

```
'Subscribing the event
Private pdfLightTable.BeginRowLayout
+= NewBeginRowLayoutEventHandler(pdfLightTable_BeginRowLayout)

'Setting rowheight for the second row
Private Sub pdfLightTable_BeginRowLayout(ByVal sender As Object, ByVal args As BeginRowLayoutEventArgs)
    If args.RowIndex = 1 Then
        args.MinimalHeight = 25
    End If
End Sub
```

4.1.2.3.1.4.4 EndRowLayout

This event is raised when row layout finishes. The arguments of this event are as follows:

- RowIndex (read-only): Index of the row (zero based)
- Cancel: Enables to cancel layout
- LayoutCompleted (read-only): Gets a value indicating whether the row was drawn completely.

- Bounds (read-only): Bounds of the row on the page

4.1.2.3.1.4.5 BeginCellLayout

This event is raised when cell layout starts. The arguments of this event are as follows:

- RowIndex (read-only): Index of the current row
- CellIndex (read-only): Index of the current cell within the row
- Value (read-only): Text value of the cell
- Bounds (read-only): Bounds of the cell
- [Graphics](#) : Graphics on which the cell should be drawn
- Skip: Indicates if the cell should be skipped

The following code example illustrates how to draw the graphics elements inside the cell.

[C#]

```
//Subscribing the event
pdfLightTable.BeginCellLayout += new
BeginCellLayoutEventHandler(pdfLightTable_BeginCellLayout);

// Drawing ellipse inside the cell
void pdfLightTable_BeginCellLayout(object sender, BeginCellLayoutEventArgs
args)
{
    if (args.RowIndex == 0 && args.CellIndex == 2)
    {
        args.Graphics.DrawEllipse(PdfBrushes.Red, args.Bounds);
    }
}
```

[VB .NET]

```
'Subscribing the event
Private pdfLightTable.BeginCellLayout += New
BeginCellLayoutEventHandler(pdfLightTable_BeginCellLayout)

' Drawing ellipse inside the cell
Private Sub pdfLightTable_BeginCellLayout(ByVal sender As Object, ByVal
args As BeginCellLayoutEventArgs)
    If args.RowIndex = 0 AndAlso args.CellIndex = 2 Then
        args.Graphics.DrawEllipse(PdfBrushes.Red, args.Bounds)
    End If
End Sub
```

End Sub

4.1.2.3.1.4.6 EndCellLayout

This event is raised when cell layout finishes. The arguments of this event are as follows:

- RowIndex (read-only): Index of the current row
- CellIndex (read-only): Index of the current cell within the row
- Value (read-only): Text value of the cell
- Bounds (read-only): Bounds of the cell
- [Graphics](#): Graphics on which the cell should be drawn

4.1.2.3.2 PdfGrid

PdfGrid class, based on cell model, helps to draw tables of complex structures. Data from DataTables, arrays or other entity classes can be given as input. In Silverlight platform, IEnumerable data objects can be set as data source. Formatting can be done at all levels and it provides direct API for this. It also supports row, column spanning and drawing of nested tables. This section explains how a table can be drawn using PdfGrid. It includes the following sections:

4.1.2.3.2.1 Properties, Methods and Events

Properties

Name	Description	Data Type
AllowRowBreakAcrossPages	Gets or sets whether to split or move rows that overflow a page.	Boolean
Columns	Gets the columns.	PdfGridColumnCollection
DataMember	Gets or sets the data member.	String
DataSource	Gets or sets the data source.	Object
Headers	Gets the headers.	PdfGridHeaderCollection
RepeatHeader	Gets or set a value indicating whether to repeat header	Boolean
Rows	Gets the rows.	PdfGridRowCollection
Style	Gets or sets the style.	PdfGridStyle

Methods

Method	Description	Parameters	Return Type
Draw	Draws PdfGrid	Overloads: (PdfGraphics graphics) (PdfPage page, PointF location) (PdfPage page, RectangleF bounds) (PdfGraphics graphics, PointF location) (PdfGraphics graphics, RectangleF bounds) (PdfPage page, float x, float y) (PdfPage page, PointF location, PdfGridLayoutFormat format) (PdfPage page, PointF location, PdfLayoutFormat format) (PdfPage page, RectangleF bounds, PdfGridLayoutFormat format) (PdfPage page, PointF location, PdfLayoutFormat format) (PdfGraphics graphics, float x, float y) (PdfGraphics graphics, PointF location, float width) (PdfPage page, float x, float y, float width) (PdfPage page, float x, float y, PdfGridLayoutFormat format) (PdfPage page, float x, float y, PdfLayoutFormat format) (PdfGraphics graphics, float x, float y, float width) (PdfPage page, float x, float y, float width, PdfGridLayoutFormat format)	Void PdfGridLayoutResult PdfGridLayoutResult Void Void PdfGridLayoutResult PdfGridLayoutResult PdfLayoutResult PdfGridLayoutResult PdfLayoutResult void void PdfGridLayoutResult PdfGridLayoutResult PdfLayoutResult void PdfGridLayoutResult

Events

Name	Description
BeginPageLayout	This event is raised before the element should be printed on the page. (Inherited from PdfLayoutElement.)
EndPageLayout	This event is raised after the element is printed on the page. (Inherited from PdfLayoutElement.)

4.1.2.3.2.2 PdfGrid Creation

Note: You must add **Syncfusion.Pdf.Grid** namespace to work with **PdfGrid**.

You can create a PdfGrid by simply specifying the new operator with a proper constructor. After assigning data source it can be drawn using one of the overloads of Draw method.

[C#]

```
// Create a PdfGrid.
PdfGrid pdfGrid = new PdfGrid();

// Assign data source.
pdfGrid.DataSource = dataSource;

// Draw PdfGrid.
pdfGrid.Draw(graphics);
```

[VB .NET]

```
' Create a PdfGrid.
Dim pdfGrid As New PdfGrid()

' Assign data source.
pdfGrid.DataSource = dataSource

' Draw PdfGrid.
pdfGrid.Draw(graphics)
```

Data to PdfGrid can be entered manually or taken from an external data source. Also, the draw method helps to control the layout of the PdfGrid and returns information to user after drawing completes. The following topics discuss them.

- Data
- Layout

AllowRowBreakAcrossPages

This property allows changing the row split behavior across pages. By default, this Boolean property is set to True, which allows splitting the row across pages when the row cannot accommodate within the bounds of the page. If set to false, the entire row will be shifted to the next page.

Note: *If the row height is greater than the page height, the row will be split or cut based on the True and False value of this property.*

4.1.2.3.2.2.1 Data

External Data Source

You can bind data to a PdfGrid by associating it with an external data source. You can set the external data source by using the DataSource property. The following code example illustrates this.

[C#]

```
DataTable dt = new DataTable();
dt.Columns.Add("ID");
dt.Columns.Add("Name");

dt.Rows.Add(new object[] { "E01", "Clay" });
dt.Rows.Add(new object[] { "E02", "Thomas" });

// Create a PdfGrid.
PdfGrid pdfGrid = new PdfGrid();

// dt is an object that can be an array (two-dimensional, one-dimensional or nested), a DataTable, DataColumn, DataView or DataSet.
pdfGrid.DataSource = dt;
```

[VB.NET]

```
Dim dt As DataTable = New DataTable()
dt.Columns.Add("ID")
dt.Columns.Add("Name")

dt.Rows.Add(New Object() { "E01", "Clay"})
dt.Rows.Add(New Object() { "E02", "Thomas" })

' Create a PdfGrid.
```

```
Dim pdfGrid As New PdfGrid()

' dt is an object that can be an array (two-dimensional, one-
dimensional or nested), a DataTable, DataColumn, DataView or DataSet.
pdfGrid.DataSource = dt
```

Direct Rows and Columns

Alternatively, you can bind data to a PdfGrid without setting any data source. This is achieved using the PdfGridRow and PdfGridColumn classes. The following code example illustrates this.

[C#]

```
PdfPage pdfPage = pdfDocument.Pages.Add();

// Create a new PdfGrid.
PdfGrid pdfGrid = new PdfGrid();

// Add three columns.
pdfGrid.Columns.Add(3);

// Add header.
pdfGrid.Headers.Add(1);
PdfGridRow pdfGridHeader = pdfGrid.Headers[0];
pdfGridHeader.Cells[0].Value = "Employee ID";
pdfGridHeader.Cells[1].Value = "Employee Name";
pdfGridHeader.Cells[2].Value = "Salary";

// Add rows.
PdfGridRow pdfGridRow = pdfGrid.Rows.Add();
pdfGridRow.Cells[0].Value = "E01";
pdfGridRow.Cells[1].Value = "Clay";
pdfGridRow.Cells[2].Value = "$10,000";

// Draw the PdfGrid.
pdfGrid.Draw(pdfPage, PointF.Empty);
```

[VB.NET]

```
Dim pdfPage As PdfPage = pdfDocument.Pages.Add()

' Create a new PdfGrid.
Dim pdfGrid As New PdfGrid()
```

```

' Add three columns.
pdfGrid.Columns.Add(3)

' Add header.
pdfGrid.Headers.Add(1)
Dim pdfGridHeader As PdfGridRow = pdfGrid.Headers(0)
pdfGridHeader.Cells(0).Value = "Employee ID"
pdfGridHeader.Cells(1).Value = "Employee Name"
pdfGridHeader.Cells(2).Value = "Salary"

' Add rows.
Dim pdfGridRow As PdfGridRow = pdfGrid.Rows.Add()
pdfGridRow.Cells(0).Value = "E01"
pdfGridRow.Cells(1).Value = "Clay"
pdfGridRow.Cells(2).Value = "$10,000"

' Draw the PdfGrid.
pdfGrid.Draw(pdfPage, PointF.Empty)

```

4.1.2.3.2.2.2 Layout

This section speaks about layouting options and the two events associated with PdfGrid.

PdfGridLayoutFormat

Layouting PdfGrid can be done using the PdfGridLayoutFormat class. Overloads accepting pages can accept standard formats as other layouting elements. However, they treat the PdfLayoutBreakType.FitElement value of Format.Break property as one, for a single row and not for the entire PdfGrid.

Properties

Name	Description	Data Type
Break	Gets or sets the break type	PdfLayoutBreakType
Layout	Gets or sets the layout type	PdfLayoutType
PaginateBounds	Gets or sets the PdfGrid bounds for the following page	RectangleF

The PdfLayoutType class is used to specify the type of pagination. The Paginate LayoutType draws the PdfGrid to the next following pages, if the element exceeds the page. The OnePage layout draws the element only on one page. The following code example illustrates this.

[C#]

```
PdfGridLayoutFormat format = new PdfGridLayoutFormat();
format.Layout = PdfLayoutType.Paginate;
format.Break = PdfLayoutBreakType.FitElement;

// Draws the PdfGrid with the layout format
pdfGrid.Draw(pdfPage, PointF.Empty, format);
```

[VB .NET]

```
Dim format As PdfGridLayoutFormat = New PdfGridLayoutFormat()
format.Layout = PdfLayoutType.Paginate
format.Break = PdfLayoutBreakType.FitElement

' Draws the PdfGrid with the layout format
pdfGrid.Draw(pdfPage, PointF.Empty, format)
```

PdfGridLayoutResult

You can get the layout settings for the drawn PdfGrid with the help of PdfGridLayoutResult class. Also, you can get the bounds and the last page where the PdfGrid was drawn using the Bounds and Page properties. This is mainly used to write some text or any other element below the large table that shows number of pages.

Properties

Name	Description	Data Type
Bounds	Gets the bounds in the last page where it was drawn	RectangleF
Page	Gets the last page where PdfLightTable was drawn	PdfPage

[C#]

```
// Draws the PdfGrid
```

```

PdfGridLayoutResult result = pdfGrid.Draw(pdfPage, PointF.Empty,
format);

// Returns the rectangle value for the last page.
Console.WriteLine(result.Bounds.ToString());

```

[VB .NET]

```

' Draws the PdfGrid.
Dim result As PdfGridLayoutResult = pdfGrid.Draw(pdfPage, PointF.Empty,
format)

' Returns the rectangle value for the last page.
Console.WriteLine(result.Bounds.ToString())

```

Events

The following events are associated with PdfGrid. The functionalities of these events are common for both PdfGrid and PdfLightTable.

- [BeginPageLayout](#)
- [EndPageLayout](#)

4.1.2.3.2.3 PdfGrid Formatting

This section explains the most direct options available to format PdfGrid. The PdfGridStyle class, accessible through Style property of PdfGrid provides options to format entire PdfGrid or parts of it. Formatting applicable for entire PdfGrid using PdfGridStyle class is discussed in this section. Header, Row, Column and Cell are discussed in the following links:

- [Header](#)
- [Row](#)
- [Column](#)
- [Cell](#)

Note: If the style properties are applied to both PdfGridCell and PdfGridRow, PdfGridCell takes over the precedence. Following is an example for the exact order of precedence.

```

PdfBrush backgroundBrush = Cell.BackgroundBrush ?? Row.Style.BackgroundBrush ?? 
Row.Grid.Style.BackgroundBrush

```

Properties

Name	Description	Data Type
------	-------------	-----------

AllowHorizontalOverflow	Gets or sets a value indicating whether to allow horizontal overflow.	Boolean
BackgroundBrush	Gets or sets background brush.	PdfBrush
BorderOverlapStyle	Gets or sets border overlap style.	PdfBorderOverlapStyle
CellPadding	Gets or sets cell padding.	PdfPaddings
CellSpacing	Gets or sets cell spacing.	Float
Font	Gets or sets the font.	PdfFont
HorizontalOverflowType	Gets or sets the type of horizontal overflow.	PdfHorizontalOverflowType
TextBrush	Gets or sets the text brush.	PdfBrush
TextPen	Gets or sets the text pen.	PdfPen

AllowHorizontalOverflow

If set to True, the columns exceeding the current page width would be wrapped and drawn in the next or last page. The default value is false. It should be used along with HorizontalOverflowType property. The default value of HorizontalOverflowType is PdfHorizontalOverflowType.LastPage.

[C#]

```
pdfGrid.Style.AllowHorizontalOverflow = true;
pdfGrid.Style.HorizontalOverflowType =
PdfHorizontalOverflowType.NextPage;
```

[VB .NET]

```
pdfGrid.Style.AllowHorizontalOverflow = True
pdfGrid.Style.HorizontalOverflowType =
PdfHorizontalOverflowType.NextPage
```

BorderOverlapStyle

This property decides if the cell border should overlap with neighboring cells or to draw the interior of cell.

Note: This property applies for all cells in the PdfGrid. Be careful while using overlapping borders, because they may produce bad results if they are not of the same width and color.

[C#]

```
pdfGrid.Style.BorderOverlapStyle = PdfBorderOverlapStyle.Overlap;
```

[VB .NET]

```
pdfGrid.Style.BorderOverlapStyle = PdfBorderOverlapStyle.Overlap
```

CellPadding

The distance between text and border inside a cell otherwise known as CellPadding, can be set to all cells in the PdfGrid. The PdfPaddings class allows setting padding to individual or all sides.

[C#]

```
// Padding will be applied for all four sides of cells in PdfGrid.  
pdfGrid.Style.CellPadding.All = 0.3f;  
  
// Padding will be applied only at the top of all cells in PdfGrid.  
pdfGrid.Style.CellPadding.Top = 0.3f;
```

[VB .NET]

```
' Padding will be applied for all four sides of cells in PdfGrid.  
pdfGrid.Style.CellPadding.All = 0.3f  
  
' Padding will be applied only at the top for all cells in PdfGrid.  
pdfGrid.Style.CellPadding.Top = 0.3f
```

CellSpacing

The distance between the cells otherwise known as CellSpacing, can be set to all cells in PdfGrid using CellSpacing property.

[C#]

```
pdfGrid.Style.CellSpacing = 0.5f;
```

[VB .NET]

```
pdfGrid.Style.CellSpacing = 0.5f
```

4.1.2.3.2.3.1 Header

Header is a set of rows that can be optionally repeated on each page and has its own style. You can add header as follows:

- Directly from column captions.
- By using Add method of the PdfGridHeaderCollection class.

Note: When you bind data source to PdfGrid, column captions will be automatically added to header collection. It can be removed at any time using Clear method of PdfGridHeaderCollection.

The following code example illustrates how to add headers to PdfGrid by using the Add method.

[C#]

```
// Add a new header to PdfGrid.
pdfGrid.Headers.Add(1);

// Get the first header row.
PdfGridCellCollection collection = pdfGrid.Headers[0].Cells;

// Set the header names.
collection[0].Value = "Header1";
collection[1].Value = "Header2";
collection[2].Value = "Header3";
collection[3].Value = "Header4";
```

[VB .NET]

```
' Add a new header to PdfGrid.
pdfGrid.Headers.Add(1)

' Get the first header row.
Dim collection As PdfGridCellCollection = pdfGrid.Headers(0).Cells
```

```
' Set the header names.
collection(0).Value = "Header1"
collection(1).Value = "Header2"
collection(2).Value = "Header3"
collection(3).Value = "Header4"
```

RepeatHeader

Header can be set to repeat on each page where PdfGrid is paginated. RepeatHeader property should be set to true to achieve this.

[C#]

```
pdfGrid.RepeatHeader = true;
```

[VB .NET]

```
pdfGrid.RepeatHeader = True
```

Style

You can specify the header style for the PdfGrid by using the PdfGridRowStyle or PdfGridCellStyle classes. The style applied at the collection will be applied to all rows in the header. Following code example illustrates how to specify the header style.

[C#]

```
// Applying header style.
pdfGrid.Headers.ApplyStyle(style);
```

[VB .NET]

```
' Applying header style.
pdfGrid.Headers.ApplyStyle(style)
```

Note: Styles for each PdfGridRow in Header can be individually applied using PdfGridRowStyle class.

Refer to the following topics for more details:

[PdfGridRowStyle](#)

[PdfGridCellStyle](#)

4.1.2.3.2.3.2 Row

Height

The Height property of the PdfGridRow class is used to specify the row height for the PdfGrid rows. The following code example illustrates how to set this property.

[C#]

```
// Access the row in PdfGrid.
PdfGridRow pdfGridRow = pdfGrid.Rows[0];

// Set the height for a row.
pdfGridRow.Height = 20;
```

[VB .NET]

```
' Access the row in PdfGrid.
Dim pdfGridRow As PdfGridRow = pdfGrid.Rows(0)

' Set the height for a row.
pdfGridRow.Height = 20
```

Note: The unit of the Height property is always points. You can set the PDF units only as points. Also, you can use the [PdfUnitConvertor](#) class to convert the other units to points.

Row Span

PdfGrid enables you to merge cells within a row. You can specify the number of cells to be merged using the RowSpan property of PdfGridCell class. The following code example illustrates this.

[C#]

```
// Merging row cells.
pdfGridRow.Cells[0].RowSpan = 2;
```

[VB .NET]

```
' Merging row cells.
pdfGridRow.Cells(0).RowSpan = 2
```

Style

The PdfGridRowStyle class, accessed through Style property of PdfGridRow class is used to specify the row style for the PdfGrid rows.

Note: If the style properties are applied to both PdfGridCell and PdfGridRow, PdfGridCell takes over the precedence. Following is an example for the exact order of precedence.

PdfBrush backgroundBrush = Cell.BackgroundBrush ?? Row.Style.BackgroundBrush ??
Row.Grid.Style.BackgroundBrush

The following code example illustrates how to specify the row style for the PdfGrid rows:

[C#]

```
// Create an instance of PdfGridRowStyle
PdfGridRowStyle pdfGridRowStyle = new PdfGridRowStyle();
pdfGridRowStyle.BackgroundBrush = PdfBrushes.LightYellow;
pdfGridRowStyle.Font = new PdfStandardFont(PdfFontFamily.Courier, 10);
pdfGridRowStyle.TextBrush = PdfBrushes.Blue;
pdfGridRowStyle.TextPen = PdfPens.Pink;

// Set style for the PdfGridRow.
pdfGrid.Rows[0].Style = pdfGridRowStyle;
```

[VB .NET]

```
' Create an instance of PdfGridRowStyle
Dim pdfGridRowStyle As New PdfGridRowStyle()
pdfGridRowStyle.BackgroundBrush = PdfBrushes.LightYellow
pdfGridRowStyle.Font = New PdfStandardFont(PdfFontFamily.Courier, 10)
pdfGridRowStyle.TextBrush = PdfBrushes.Blue
pdfGridRowStyle.TextPen = PdfPens.Pink

' Set style for the PdfGridRow.
pdfGrid.Rows(0).Style = pdfGridRowStyle
```

You may also apply PdfGridCellStyle to a PdfGridRow using the ApplyStyle property. The following code snippet illustrates this:

[C#]

```
// Set style for the PdfGridRow.
pdfGrid.Rows[0].ApplyStyle(pdfGridCellStyle);
```

[VB .NET]

```
// Set style for the PdfGridRow.
pdfGrid.Rows(0).ApplyStyle(pdfGridCellStyle)
```

All rows in the PdfGrid can be set with same style using the `ApplyStyle` method of `PdfGridRowCollection`. This style can be a `PdfGridRowStyle` or `PdfGridCellStyle`. The following is the code snippet:

[C#]

```
// Set style for all rows in PdfGrid.
pdfGrid.Rows.ApplyStyle(style);
```

[VB .NET]

```
// Set style for all rows in PdfGrid.
pdfGrid.Rows.ApplyStyle(style)
```

Refer to the following topic for more details:

[PdfGridCellStyle](#)

4.1.2.3.2.3.3 Column

Width

By default, all the columns in PdfGrid have equal width, and the columns automatically fill the entire width of the PdfGrid. If the width of the PdfGrid is increased or decreased, the column width also changes appropriately.

You can specify the width for a particular column by using the `Width` property. The following code example illustrates how to set the width.

[C#]

```
// Set Width for first column.
pdfGrid.Columns[0].Width = 20f;
```

[VB.NET]

```
' Set Width for first column.
pdfGrid.Columns(0).Width = 20f
```

Note: The unit of the **Width** property is always points. You can set the PDF units only as points. Also, you can use the [PdfUnitConvertor class](#) to convert the other units to points.

Column Span

PdfGrid enables you to merge cells within a column. You can specify the number of cells to be merged by using the **ColumnSpan** property **PdfGridCell** class. The following code example illustrates this.

[C#]

```
// Merging column cells.
pdfGrid.Rows[0].Cells[0].ColumnSpan = 2;
```

[VB.NET]

```
' Merging column cells.
pdfGrid.Rows(0).Cells(0).ColumnSpan = 2
```

Format

You can specify the content format for the PdfGrid columns by using the **Format** property. Check String Formatting in [DrawingText](#) for more details.

4.1.2.3.2.3.4 Cell

Properties

Name	Description	Data Type
------	-------------	-----------

ColumnSpan	Gets or set the column span.	Integer
Height	Gets the height.	Float
ImagePosition	Gets or sets the image alignment type of the background image.	PdfGridImagePosition
RowSpan	Gets or sets the row span	Integer
StringFormat	Gets or sets the string format.	PdfStringFormat
Style	Gets or sets the cell style.	PdfGridCellStyle
Value	Gets or sets the value.	Object
Width	Gets the width.	float

Cell Size

The width and height cannot be modified for a single cell, but for the entire column or row. Please check [PdfGridColumn](#) and [PdfGridRow](#) for more details.

Value

You can specify the value for an individual cell using the Value property. Also, you can specify another PdfGrid as the cell value to make a nested table. The following code snippet illustrates this.

[C#]

```
// Set the value to the specific cell.
parentPdfGrid.Rows[0].Cells[0].Value = "Nested Table";
parentPdfGrid.Rows[0].Cells[1].RowSpan = 2;
parentPdfGrid.Rows[0].Cells[1].ColumnSpan = 2;

PdfGrid childPdfGrid = new PdfGrid();
childPdfGrid.Columns.Add(5);
for (int i = 0; i < 5; i++)
{
    PdfGridRow row = childPdfGrid.Rows.Add();
    for (int j = 0; j < 5; j++)
```

```

    {
        row.Cells[j].Value = String.Format("Cell [{0} {1}]", j, i);
    }
}

// Set the value as another PdfGrid in a cell.
parentGrid.Rows[0].Cells[1].Value = childPdfGrid;

```

[VB.NET]

```

' Set the value to the specific cell.
parentPdfGrid.Rows(0).Cells(0).Value = "Nested Table"
parentPdfGrid.Rows(0).Cells(1).RowSpan = 2
parentPdfGrid.Rows(0).Cells(1).ColumnSpan = 2

Dim childPdfGrid As New PdfGrid()
childPdfGrid.Columns.Add(5)
For i As Integer = 0 To 4
    Dim row As PdfGridRow = childPdfGrid.Rows.Add()
    For j As Integer = 0 To 4
        row.Cells(j).Value = String.Format("Cell [{0} {1}]", j, i)
    Next j
Next i

' Set the value as another PdfGrid in a cell.
parentGrid.Rows(0).Cells(1).Value = childPdfGrid

```

Style

PdfGrid provides various options to customize the cell content, text color, background color, and so on. The following properties can be used for this purpose.

Properties

Name	Description	Data Type
BackgroundBrush	Gets or sets background brush for the cell.	PdfBrush
BackgroundImage	Gets or sets background image for the cell.	PdfImage
Borders	Gets or sets the borders	PdfBorders

Font	Gets or sets the font.	PdfFont
StringFormat	Gets or sets the string format	PdfStringFormat
TextBrush	Gets or sets the text brush.	PdfBrush
TextPen	Gets or sets the text pen.	PdfPen

The following code example illustrates how to customize the cell content.

[C#]

```
//Specify the style for the PdfGridCell.
PdfGridCellStyle pdfGridCellStyle = new PdfGridCellStyle();
pdfGridCellStyle.BackgroundImage = new PdfBitmap("pdf_button.png");
pdfGridCellStyle.TextPen = PdfPens.Red;
pdfGridCellStyle.Borders.All = PdfPens.Red;

PdfGridCell pdfGridCell = pdfGrid.Rows[0].Cells[0];

// Apply style
pdfGridCell.Style = pdfGridCellStyle;

// Set image position for the background image in the style.
pdfGridCell.ImagePosition = PdfGridImagePosition.Fit;
```

[VB .NET]

```
'Specify the style for the PdfGridCell.
Dim pdfGridCellStyle As New PdfGridCellStyle()
pdfGridCellStyle.BackgroundImage = New PdfBitmap("pdf_button.png")
pdfGridCellStyle.TextPen = PdfPens.Red
pdfGridCellStyle.Borders.All = PdfPens.Red

Dim pdfGridCell As PdfGridCell = pdfGrid.Rows(0).Cells(0)

' Apply style
pdfGridCell.Style = pdfGridCellStyle

' Set image position for the background image in the style.
pdfGridCell.ImagePosition = PdfGridImagePosition.Fit
```

For more details on PdfGridImagePosition, check the following FAQ:

[PdfGridCell background image](#)

4.1.2.4 Lists

Lists in the Essential PDF are used to list out the items in a collection in some order to provide readability. There are two kinds of lists. They are:

- **Ordered list**, which is represented by the PdfOrderedList class
- **Unordered list**, which is represented by the PdfUnorderedList class

Base class for the preceding classes is the **PdfList** class, which contains an item collection represented by the **PdfListCollection** class. The items from the collection are represented by the **PdfListItem** class.

Ordered List

PdfOrderedList class represents an ordered list.

Initialize Lists

You can create new instances of the PdfOrderedList class by using the following constructors.

- **PdfOrderedList()**: Creates list with default settings
- **PdfOrderedList(PdfListCollection items)**: Creates list with the specified collection of items
- **PdfOrderedList(PdfOrderedMarker marker)**: Creates list with the specified marker
- **PdfOrderedList(PdfListCollection items, PdfOrderedMarker marker)**: Creates list with the specified items collection and marker
- **PdfOrderedList(string text)**: creates list from the specified text. It splits text by using the "\n" symbol and creates a collection of items.
- **PdfOrderedList(string text, PdfOrderedMarker marker)**: Creates list from the specified text and with specified marker. It splits text by using the "\n" symbol and creates a collection of items.

List Marker

Ordered list has ordered markers that are represented by the **PdfOrderedMarker** class. To create a new instance of the ordered marker, use the following constructors.

- **PdfOrderedMarker(PdfNumberStyle style, PdfFont font)**: Creates marker by using the PdfNumberStyle and specified font
- **PdfOrderedMarker (PdfNumberStyle style, string finalizer, PdfFont font)**: Creates marker with number style, font, finalizer, and the specified symbol that follows after number. Default value for finalizer is '.'.
- **PdfOrderedMarker (PdfNumberStyle style, string delimiter, string finalizer, PdfFont font)**: Creates marker with the number style, font, finalizer, delimiter, and the specified symbol located between numbers. It is used when the **MarkerHierarchy** property of the PdfOrderedList class is set **True**. Default value for delimiter is '.'.

Default list marker has **Number** style.

Features

Also, PdfOrderedList enables you to use numbering hierarchy, which means if you have an ordered list and one of its item has an ordered sublist, marker of it will contain the number of its parent item. This number is split by the delimiter.

To use numbering hierarchy, just set the **MarkerHierarchy** property of the PdfOrderedList class to **True**. Default value is **False**.

Unordered List

PdfUnorderedList class represents an unordered list.

Initialize Lists

You can create a new instance of the PdfUnorderedList class by using the following constructors.

- **PdfUnorderedList()**: Creates list with default settings
- **PdfUnorderedList(PdfListItemCollection items)**: Creates list with the specified collection of items
- **PdfUnorderedList(PdfUnorderedMarker marker)**: Creates list with the specified marker
- **PdfUnorderedList(PdfListItemCollection items, PdfUnorderedMarker marker)**: Creates list with the specified items collection and marker
- **PdfUnorderedList (string text)**: Creates list from the specified text. It splits the text by using the "\n" symbol and creates a collection of items.
- **PdfUnorderedList (string text, PdfUnorderedMarker marker)**: Creates list from the specified text and with the specified marker. It splits text by using the "\n" symbol and creates a collection of items.

List Marker

Unordered list has an unordered marker that is represented by the **PdfUnorderedMarker** class. Unordered marker has the marker style represented by the **PdfUnorderedMarkerStyle** class. The following marker styles are supported.

- None
- Disk
- Square
- Asterisk
- Circle
- CustomString
- CustomImage
- CustomTemplate

Default list marker has **Disk** style.

To use the **CustomString**, **CustomImage** or **CustomTemplate** style, you must set the **Text**, **Image** or **Template** property of the **PdfUnorderedMarker** class respectively.

Drawing Lists

There are a lot of **Draw** overloads that enable you to draw lists on a series of pages or on a **PdfGraphics** page.

Events

Each list raises the following four events:

- **BeginPageLayout** event is raised when the list starts layouting on page
- **EndPageLayout** event is raised when the list completes layouting on the page
- **BeginItemLayout** event is raised when the item starts layouting
- **EndItemLayout** event is raised when the item completes layouting



Note: You should add the **Syncfusion.Pdf.List** namespace to work with lists.

4.1.3 Interactive Features

This section demonstrates various user interactive features and how it can be implemented with Essential PDF. It includes the following topics.

- **Action**—Demonstrates various file actions and form actions supported by Essential PDF.
- **Annotation**—Demonstrates various annotation types supported by Essential PDF.
- **Attachments**—Explains how various file attachments can be carried out by using Essential PDF.
- **Portfolio**—Demonstrates how to create a PDF Portfolio to add multiple files of different formats, created in different applications.



Note: You must add the `Syncfusion.Pdf.Interactive` namespace to work with interactive features.

4.1.3.1 Actions

Essential PDF supports different actions triggered by different events and user interaction. There are a lot of possible actions such as playing a particular sound or movie, launching an application or URI, and so on.

Essential PDF supports the following types of actions.

- **PdfSoundAction**, which plays the specified music file
- **PdfUriAction** that launches the specified URI
- **PdfGoToAction** that goes to the specified page of the document
- **PdfJavaScriptAction**, which executes specified PDF javascript code
- **PdfLaunchAction** that launches the application or opens the document
- **PdfNamedAction**, which goes to the named destination: next, previous, first or last page
- **PdfSubmitAction**, which submits the data that is entered into the PDF form
- **PdfResetAction** that resets the fields of the PDF form

[C#]

```

PdfUriAction uriAction = new PdfUriAction("http://www.google.com");

PdfDestination dest = new PdfDestination(page, new Point(0, 100));

PdfGoToAction goToAction = new PdfGoToAction(page);
goToAction.Destination = dest;
uriAction.Next = goToAction;

PdfJavaScriptAction javaAction = new PdfJavaScriptAction("app.alert(\"Hello
\\\")");
goToAction.Next = javaAction;

document.Actions.AfterOpen = soundAction;

```

[VB .NET]

```

Dim uriAction As Syncfusion.Pdf.Interactive.PdfUriAction = New
Syncfusion.Pdf.Interactive.PdfUriAction("http://www.google.com")

Dim dest As Syncfusion.Pdf.Interactive.PdfDestination = New
Syncfusion.Pdf.Interactive.PdfDestination(page, New Point(0, 100))

Dim goToAction As Syncfusion.Pdf.Interactive.PdfGoToAction = New
Syncfusion.Pdf.Interactive.PdfGoToAction(page)
goToAction.Destination = dest
uriAction.Next = goToAction

Dim javaAction As Syncfusion.Pdf.Interactive.PdfJavaScriptAction = New
Syncfusion.Pdf.Interactive.PdfJavaScriptAction("app.alert(\"Hello \")")
goToAction.Next = javaAction

document.Actions.AfterOpen = soundAction

```



Note: *Next property of an action is used to specify the queue of actions as illustrated in the preceding example.*

1. Following are the actions for Document Object:

- **AfterOpen** action to be performed after the document is opened
- **AfterPrint** action to be performed after the document is printed
- **AfterSave** action to be performed after the document is saved
- **BeforeClose** action to be performed before the document is closed
- **BeforePrint** action to be performed before the document is printed
- **BeforeSave** action to be performed before the document is saved

2. Following are the actions for Annotation Object:

- **GotFocus** action to be performed when the annotation gets focus
- **LostFocus** action to be performed when the annotation loses focus
- **MouseEnter** action to be performed when the cursor enters the active area of the annotation
- **MouseLeave** action to be performed when the cursor leaves the active area of the annotation
- **MouseDown** action to be performed when the mouse button is pressed inside the active area of the annotation
- **MouseUp** action to be performed when the mouse button is released inside the active area of the annotation

3. Following are the actions for Form Field Object:

- **Calculate** javascript action to be performed to recalculate the value of the field
- **Format** javascript action to be performed before the field is formatted to display the value
- **Validate** javascript action to be performed when the value of the field is changed
- **KeyPressed** javascript action to be performed when the user types, key-stroke into a text field or combo box, or modifies the selection in a scrollable list box

Additionally, the fields support all the actions that are supported by annotations.

4.1.3.2 Annotation

An annotation associates an object such as a note, sound, or movie with a location on the page of a PDF document, or provides a way to interact with the user, by means of the mouse and keyboard.

An annotation is activated when you click the left mouse button.

- PdfSoundAnnotation plays sound file
- PdfAttachmentAnnotation opens attached file
- PdfUriAnnotation navigates to specified URI
- PdfActionAnnotation performs specified action
- PdfFileLinkAnnotation opens file with the specified path
- PdfDocumentLinkAnnotation navigates to specified destination within the document
- PdfPopupAnnotation displays the text in a pop-up window for entry and editing
- PdfLineAnnotation displays a single straight line on the page

[C#]

```
RectangleF popupAnnotationRectangle = new RectangleF(0, 50, 50, 50);
PdfPopupAnnotation popupAnnotation = new
PdfPopupAnnotation(popupAnnotationRectangle, "Test popup annotation");
popupAnnotation.Border.Width = 4;
popupAnnotation.Icon = PopupIcon.NewParagraph;
page.Annotations.Add(popupAnnotation);

RectangleF uriAnnotationRectangle = new RectangleF(0, 100, 80, 20);
PdfUriAnnotation uriAnnotation = new
PdfUriAnnotation(uriAnnotationRectangle, "http://www.google.com");
page.Annotations.Add(uriAnnotation);

RectangleF docLinkAnnotationRectangle = new RectangleF(0, 200, 80, 20);
PdfDocumentLinkAnnotation documentAnnotation = new
PdfDocumentLinkAnnotation(docLinkAnnotationRectangle);
documentAnnotation.Text = "Document link annotation";
```

```

documentAnnotation.Color = new PdfColor(Color.Navy);
documentAnnotation.Border.Width = 3;
documentAnnotation.Border.HorizontalRadius = 25;
documentAnnotation.AnnotationFlags = AnnotationFlags.NoRotate;
PdfPage page2 = document.Pages.Add();
documentAnnotation.Destination = new PdfDestination(page2);
documentAnnotation.Destination.Location = new Point(0, 0);
documentAnnotation.Destination.Zoom = 5;
page.Annotations.Add(documentAnnotation);

```

[VB.NET]

```

Dim popupAnnotationRectangle As RectangleF = New RectangleF(0, 50, 50, 50)
Dim popupAnnotation As Syncfusion.Pdf.Interactive.PdfPopupAnnotation = New
Syncfusion.Pdf.Interactive.PdfPopupAnnotation(popupAnnotationRectangle,
"Test popup annotation")
popupAnnotation.Border.Width = 4
popupAnnotation.Icon = PopupIcon.NewParagraph
page.Annotations.Add(popupAnnotation)

Dim uriAnnotationRectangle As RectangleF = New RectangleF(0, 100, 80, 20)
Dim uriAnnotation As Syncfusion.Pdf.Interactive.PdfUriAnnotation = New
Syncfusion.Pdf.Interactive.PdfUriAnnotation(uriAnnotationRectangle,
"http://www.google.com")
page.Annotations.Add(uriAnnotation)

Dim docLinkAnnotationRectangle As RectangleF = New RectangleF(0, 200, 80,
20)
Dim documentAnnotation As
Syncfusion.Pdf.Interactive.PdfDocumentLinkAnnotation = New
Syncfusion.Pdf.Interactive.PdfDocumentLinkAnnotation(docLinkAnnotationRectan-
gle)
documentAnnotation.Text = "Document link annotation"
documentAnnotation.Color = New PdfColor(Color.Navy)
documentAnnotation.Border.Width = 3
documentAnnotation.Border.HorizontalRadius = 25
documentAnnotation.AnnotationFlags = AnnotationFlags.NoRotate
Dim page2 As Syncfusion.Pdf.PdfPage = document.Pages.Add()
documentAnnotation.Destination = New PdfDestination(page2)
documentAnnotation.Destination.Location = New Point(0, 0)
documentAnnotation.Destination.Zoom = 5
page.Annotations.Add(documentAnnotation)

```

Getting Annotation from existing PDF Document

Essential PDF now supports reading annotations from the existing PDF document. The following code example illustrates this.

[C#]

```
PdfLoadedDocument ldoc = new PdfLoadedDocument("Sample.pdf");
PdfPageBase lpage = ldoc.Pages[1];
PdfLoadedAttachmentAnnotation attAnnot = lpage.Annotations[0] as
PdfLoadedAttachmentAnnotation;
attAnnot.Color = new PdfColor(Color.Red);
attAnnot.Text = "New Annotation";
attAnnot.Icon = PdfAttachmentIcon.PushPin;
```

[VB .NET]

```
Dim ldoc As PdfLoadedDocument = New PdfLoadedDocument("Sample.pdf")
Dim lpage As PdfPageBase = ldoc.Pages(1)
Dim attAnnot As PdfLoadedAttachmentAnnotation = TryCast(lpage.Annotations(0),
PdfLoadedAttachmentAnnotation)
attAnnot.Color = New PdfColor(Color.Red)
attAnnot.Text = "New Annotation"
attAnnot.Icon = PdfAttachmentIcon.PushPin
```

4.1.3.2.1 3D Annotation

3D Annotations are the means by which 3D artwork is represented in a PDF document. Essential PDF can embed the 3D files (u3d) in PDF files. It provides a way to interact with the user by using the mouse and keyboard.

You can control the annotation with the help of the following classes.

- Pdf3DAnnotation
- Pdf3DView
- Pdf3DProjection
- Pdf3DActivation
- Pdf3DBackground
- Pdf3DRenderMode
- Pdf3DLighting
- Pdf3DCrossSection

Pdf3DAnnotation

Pdf3DAnnotation specifies parameters to be applied to the virtual camera associated with a 3D annotation. These parameters include orientation and position of the camera, details regarding the projection of camera coordinates onto the target coordinate system of the annotation, and a description of the background on which the artwork is to be drawn.

The following code example illustrates how to embed a 3D file.

[C#]

```
// Pdf 3D Annotation  
Pdf3DAnnotation annotation = new Pdf3DAnnotation(new RectangleF(10, 270,  
270, 150), @"..\..\Data\box.u3d");
```

[VB .NET]

```
' Pdf 3D Annotation  
Dim annotation As Syncfusion.Pdf.Interactive.Pdf3DAnnotation = New  
Syncfusion.Pdf.Interactive.Pdf3DAnnotation(New RectangleF(10, 270, 270,  
150), "...\\Data\\box.u3d")
```

Pdf3DView

Pdf3DView class specifies parameters to be applied to the virtual camera associated with a 3D annotation. These parameters include orientation and position of the camera, details regarding the projection of camera coordinates onto the target coordinate system of the annotation, and a description of the background on which the artwork is to be drawn.

[C#]

```
// Pdf 3D View  
Pdf3DView view = new Pdf3DView();  
view.ExternalName = "Default View";
```

[VB .NET]

```
' Pdf 3D View  
Dim view As Syncfusion.Pdf.Interactive.Pdf3DView = New  
Syncfusion.Pdf.Interactive.Pdf3DView()  
view.ExternalName = "Default View"
```

Pdf3DProjection

Pdf3DProjection defines the mapping of 3D camera coordinates onto the target coordinate system of the annotation. Using this class, you can specify the type of Projection which determines how objects are projected onto the near plane and scaled. The possible values are **Orthographic** projection and **Perspective** projection.

Pdf3DProjection supports both near and far clipping. This type of clipping defines a near plane and far plane. Objects or parts of objects that are beyond the far plane or closer to the camera than the near plane, are not drawn.

[C#]

```
Pdf3DView view = new Pdf3DView();

Pdf3DProjection projection = new Pdf3DProjection();
projection.ProjectionType = Pdf3DProjectionType.Perspective;
projection.FieldOfView = 10;
projection.ClipStyle = Pdf3DProjectionClipStyle.ExplicitNearFar;
projection.NearClipDistance = 10;

view.Projection = projection;
```

[VB .NET]

```
Dim view As Syncfusion.Pdf.Interactive.Pdf3DView = New
Syncfusion.Pdf.Interactive.Pdf3DView()

Dim projection As Syncfusion.Pdf.Interactive.Pdf3DProjection = New
Syncfusion.Pdf.Interactive.Pdf3DProjection()
projection.ProjectionType = Pdf3DProjectionType.Perspective
projection.FieldOfView = 10
projection.ClipStyle = Pdf3DProjectionClipStyle.ExplicitNearFar
projection.NearClipDistance = 10

view.Projection = projection
```



Note: You can set the near or far distance by using the **NearClipDistance** property. If you want to set the clipping distance explicitly, you have to set the **ClipStyle** property to **ExplicitNearFar**.

Pdf3DActivation

Pdf3DActivation class determines when a 3D annotation is active or when it is inactive. 3D artwork is activated in one of the following three states with the help of Pdf3DActivationMode class.

- **ExplicitActivation**-Initial state of the annotation is deactivated. If you want to activate the annotation by clicking it, then you can use the explicit annotation.
- **PageOpen**-Annotation is activated while opening the pdf document page
- **PageVisible**-Annotation is activated when the page is visible

You can show or hide the toolbar by using the **ShowToolbar** property.

[C#]

```
// Creating instance for Pdf3DActivation class
Pdf3DActivation activation = new Pdf3DActivation();

// Setting Activation Mode as PageVisible
activation.ActivationMode = Pdf3DActivationMode.PageVisible;

// Showing the Toolbar
activation.ShowToolbar = true;

// Setting Deactivation Mode as PageVisible
activation.DeactivationMode = Pdf3DDeactivationMode.ExplicitDeactivation;

// Assigning the Activation to the Pdf3DAnnotation
annotation.Activation = activation;
```

[VB.NET]

```
' Creating instance for Pdf3DActivation class
Dim activation As Syncfusion.Pdf.Interactive.Pdf3DActivation = New
Syncfusion.Pdf.Interactive.Pdf3DActivation()

' Setting Activation Mode as PageVisible
activation.ActivationMode = Pdf3DActivationMode.PageVisible

' Showing the Toolbar
activation.ShowToolbar = True

' Setting Deactivation Mode as PageVisible
activation.DeactivationMode = Pdf3DDeactivationMode.ExplicitDeactivation

' Assigning the Activation to the Pdf3DAnnotation
```

```
annotation.Activation = activation
```



Note: You can also set the **deactivation mode** by using the **Pdf3DDeactivationMode** class.

Pdf3DBackground

Pdf3DBackground defines the background over which the 3D artwork is to be drawn. You can apply background color to the entire annotation by enabling the **ApplyToEntireAnnotation** property. If set to **False**, the background should apply only to the rectangle that is specified by the 3D view box of the annotation. Default value is **False**.

The following code example illustrates this.

[C#]

```
PdfColor color = new PdfColor(Color.Silver);

Pdf3DBackground background = new Pdf3DBackground();
background.Color = color;
background.ApplyToEntireAnnotation = true;

// Setting Background color to current view.
view.Background = background;
```

[VB .NET]

```
Dim color As Syncfusion.Pdf.Graphics.PdfColor = New
Syncfusion.Pdf.Graphics.PdfColor(color.Silver)

Dim background As Syncfusion.Pdf.Interactive.Pdf3DBackground = New
Syncfusion.Pdf.Interactive.Pdf3DBackground()
background.Color = color
background.ApplyToEntireAnnotation = True

' Setting Background color to current view.
view.Background = background
```

Pdf3DRenderMode

You can specify the rendering style for the 3D artwork by using the Pdf3DRenderMode class. For example, surfaces may be filled with opaque colors, they may be stroked as a "wireframe", or the artwork may be rendered with special lighting effects.

The following are the rendering styles supported by the Pdf3DRenderMode class.

- Solid
- SolidWireframe
- Transparent
- TransparentWireframe
- BoundingBox
- TransparentBoundingBox
- TransparentBoundingBoxOutline
- Wireframe
- ShadedWireframe
- HiddenWireframe
- Vertices
- ShadedVertices
- Illustration
- SolidOutline
- ShadedIllustration

Apart from using the Style property, you can also change the rendering style by using the following properties.

- **AuxiliaryColor:** PdfColor name that specifies the auxiliary color to be used when rendering the 3D image. The first entry in the array is a color space; the subsequent entries are values specifying color values in that color space.
- **FaceColor:** PdfColor name that specifies the face color to be used when rendering the 3D image. This entry is relevant only when the Style property is set to Illustration.
- **Opacity:** Number specifying the opacity of the added transparency applied by some render modes using a standard additive blend. Default value is **0.5**.
- **CreaseValue:** Number specifying the angle in degrees to be used as the crease value while determining silhouette edges. Default value is **45**.

The following code example illustrates this.

[C#]

```
Pdf3DRendermode rendermode = new Pdf3DRendermode();
rendermode.Style = Pdf3DRenderStyle.Solid;
view.RenderMode = renderMode;
```

[VB .NET]

```
Dim rendermode As Syncfusion.Pdf.Interactive.Pdf3DRendermode = New
Syncfusion.Pdf.Interactive.Pdf3DRendermode()
rendermode.Style = Pdf3DRenderStyle.Solid
view.RenderMode = renderMode
```

Pdf3DLighting

Pdf3DLighting class specifies the lighting to be applied to the 3D artwork. The following lighting effects are supported by the **Pdf3DLighting** class.

- Artwork
- None
- White
- Day
- Night
- Hard
- Primary
- Blue
- Red
- Cube
- CAD
- Headlamp

[C#]

```
Pdf3DLighting lighting = new Pdf3DLighting();
lighting.Style = Pdf3DLightingStyle.CAD;
view.LightingScheme = lightingScheme;
```

[VB .NET]

```
Dim lighting As Syncfusion.Pdf.Interactive.Pdf3DLighting = New
Syncfusion.Pdf.Interactive.Pdf3DLighting()
lighting.Style = Pdf3DLightingStyle.CAD
view.LightingScheme = lightingScheme
```

Pdf3DCrossSection

Pdf3DCrossSection specifies how a portion of the 3D artwork is clipped for the purpose of showing artwork cross sections. The following code example illustrates this.

[C#]

```
Pdf3DCrossSection csection = new Pdf3DCrossSection();
csection.IntersectionColor = color;
csection.IntersectionIsVisible = 50;
view.CrossSections = csection;
```

[VB.NET]

```
Dim csection As Syncfusion.Pdf.Interactive.Pdf3DCrossSection = New
Syncfusion.Pdf.Interactive.Pdf3DCrossSection()
csection.IntersectionColor = color
csection.IntersectionIsVisible = 50
view.CrossSections = csection
```

4.1.3.2.2 Hyperlinks for other external files

External files can be hyperlinked in a pdf document using the Annotation feature. An annotation can associate objects such as note, sound or a movie by specifying the corresponding object location in the document.

PdfFileLinkAnnotation class of the Essential PDF can be used to associate an external file to the document as follows:

[C#]

```
RectangleF flAnnotationRectangle = new RectangleF(0, 100, 80, 20);

//Create PdfFileLinkAnnotation to link external file
PdfFileLinkAnnotation flAnnotation = new
PdfFileLinkAnnotation(flAnnotationRectangle, filename);

//Add the Annotation to the page
page.Annotations.Add(flAnnotation);
```

[VB.NET]

```
Dim flAnnotationRectangle As New RectangleF(0, 100, 80, 20)

'Create PdfFileLinkAnnotation to link external file
```

```

Dim flAnnotation As New PdfFileLinkAnnotation(f1AnnotationRectangle,
filename)

'Add the Annotation to the page
Page.Annotations.Add(f1Annotation)

```

 Where `filename` = A string value specifying the full path of the file to be embedded.

Example for the location of the external file

```

PdfFileLinkAnnotation fileLinkAnnotation = new
PdfFileLinkAnnotation(fileLinkAnnotationRectangle,
@"..\..\..\Common\Images\PDF\logo.png");

```

Sample Location

A sample which illustrates this feature is available in the following sample installation location:

<Install Location>\Windows\Pdf.Windows\Samples\2.0\User Interaction\Interactive Features

4.1.3.2.3 Free Text Annotation

This feature enables the user to display the text directly on the page. Unlike an ordinary text annotation, a free text annotation has no open or closed state; the text is not displayed in the pop-up window. If the user wants to add a comment directly, without placing it in a pop-up window, FreeTextAnnotation can be used. Free Text Annotations can be included anywhere in the PDF Document by specifying the location and the CallOutLine points. While creating the project, the Interactive namespace has to be added in the project to enable this feature.

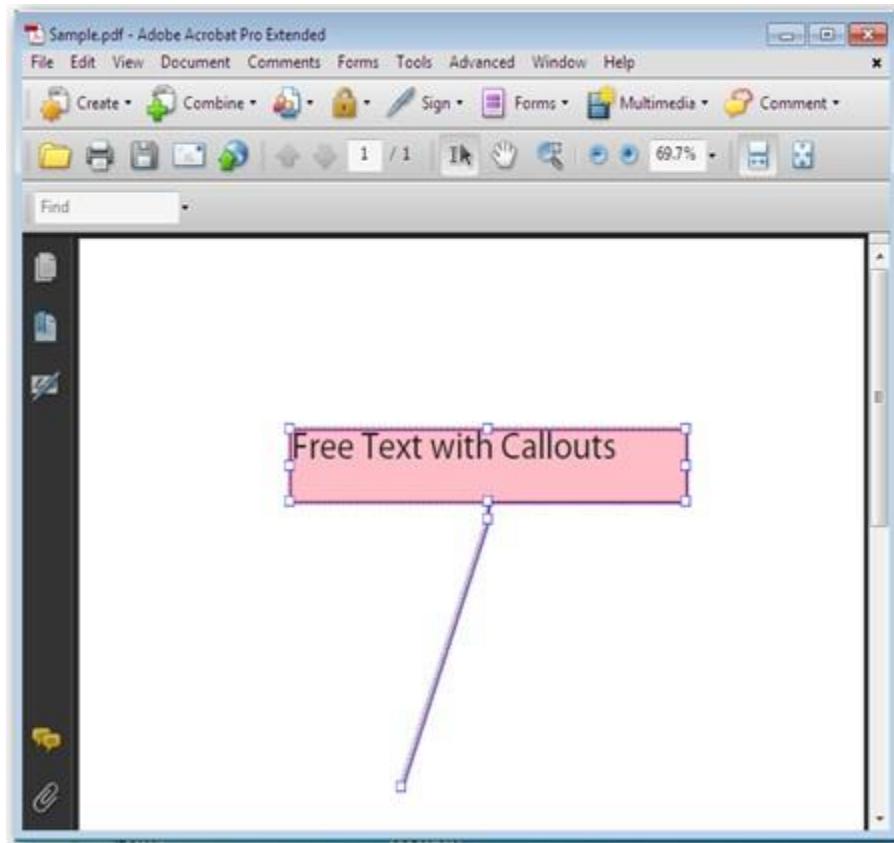


Figure 28: Free Text Annotation

List of Properties

The following table lists the properties available.

Property	Type	Value It Accepts	Description
MarkupText	String	String	Allows you to set the comment text.
TextMarkupColor	Color	PdfColor	Allows you to set the color of the comment text.
Font	Font	PdfStandardFont/PdfTrueTypeFont	Allows you to set the font type for the comment text.
Color	Color	PdfColor	Allows you to set the background color of the Annotation box.
BorderColor	Color	PdfColor	Allows you to set the border color of the Annotation box.

Property	Type	Value It Accepts	Description
Border	float	PdfAnnotationBorder	Allows you to set the border type of the Annotation box.
LineEndingStyle	lineStyle	PdfLineEndingStyle	Allows you to set the line ending style for the callout line.
AnnotationFlags	PdfAnnotationFlag	PdfAnnotationFlags	Allows you to set annotation flags.
Opacity	opacity	Float	Allows you to set the opacity of the Annotation box.
CalloutLines	Points	PointF[]	Allows you to set the starting and ending coordinates of the callout line.

List of Methods

The following table lists the methods available.

Method	Parameters of the Method	Return Type	Purpose
PdfFreeTextAnnotation	PdfFreeTextAnnotation(System.Drawing.RectangleF)	Annotations	Creates annotation to be added to the PDF document.

Creating a Free Text Annotation

The following code snippet illustrates the creation of Free Text Annotation in PDF.

[C#]

```

PdfFreeTextAnnotation annot = new PdfFreeTextAnnotation(new RectangleF(50,
100, 100, 50));

annot.MarkupText = "Free Text with Callouts";

annot.TextMarkupColor = new PdfColor(Color.Black);
annot.Font = new PdfStandardFont(PdfFontFamily.Helvetica, 7f);
annot.Color = new PdfColor(Color.Yellow);

```

```
annot.BorderColor = new PdfColor(Color.Red);
annot.Border = new PdfAnnotationBorder(.5f);
annot.LineEndingStyle = PdfLineEndingStyle.OpenArrow;
annot.AnnotationFlags = PdfAnnotationFlags.Default;
annot.Text = "Free Text";
annot.Opacity = 0.5f;
PointF[] points = { new PointF(100, 400), new PointF(100, 450)
};

annot.CalloutLines = points;

page.Annotations.Add(annot);
```

Run the code. You have successfully created a Free Text Annotation box.

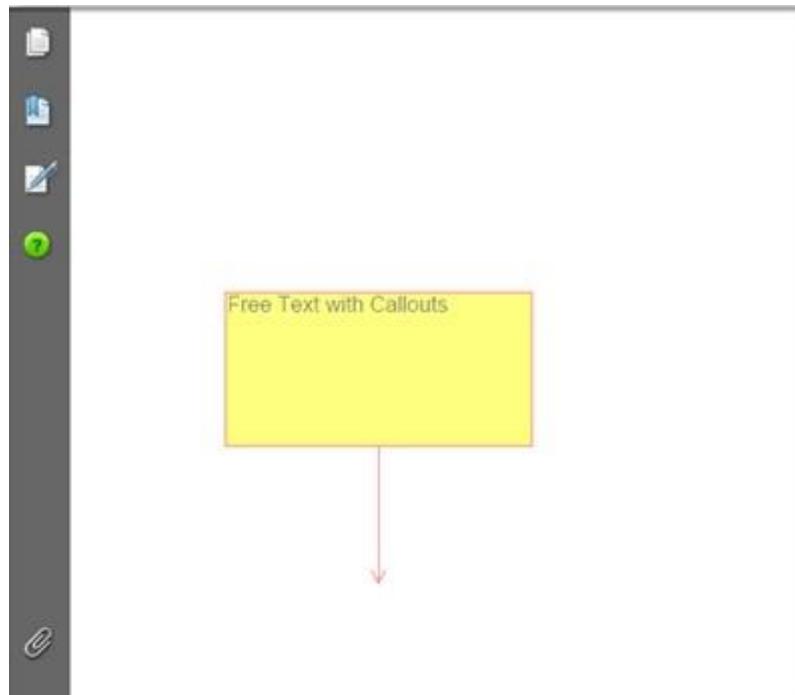


Figure 29: Free Text Annotation

4.1.3.3 Attachments

PDF document can contain any number of attached files. They are displayed on the Attachments navigation panel. All the data of the attached file is embedded into the document. To add a new attachment to the document, you can use the **PdfAttachment** class.

The following code example illustrates how to add an attachment to the document.

[C#]

```
PdfAttachment attachment = new PdfAttachment(@"..\..\Images\jpg\logo.jpg");
attachment.ModificationDate = DateTime.Now;
attachment.Description = "Syncfusion Logo";
attachment.MimeType = "application/jpeg";
document.Attachments.Add(attachment);
```

[VB .NET]

```
Dim attachment As PdfAttachment = New
PdfAttachment("../..\\Images\\jpg\\logo.jpg")
attachment.ModificationDate = DateTime.Now
attachment.Description = "Syncfusion Logo"
attachment.MimeType = "application/jpeg"
document.Attachments.Add( attachment )
```

Be sure to add the attachment to the attachment collection of the document. You can specify some additional parameters such as CreationDate, ModificationDate, MimeType or Description for the attachment.

It is possible to create attachments from data contained in a stream or data array. In this case, the file name passed to the constructor will determine the title of the attachment displayed on the attachments panel.

The following code example illustrates how to attach stream data to the PDF document.

[C#]

```
using (FileStream fileStream = new FileStream(@"..\..\Images\jpg\image.jpg",
FileMode.Open))
{
    PdfAttachment streamAttachment = new PdfAttachment("mouse.jpg",
fileStream);
    streamAttachment.MimeType = "application/jpeg";
    document.Attachments.Add(streamAttachment);
}
```

[VB .NET]

```
Dim fileStream As IO.FileStream = New  
IO.FileStream("../..\\Images\\jpg\\image.jpg", FileMode.Open)  
Dim streamAttachment As Syncfusion.Pdf.Interactive.PdfAttachment = New  
Syncfusion.Pdf.Interactive.PdfAttachment("mouse.jpg", fileStream)  
streamAttachment.MimeType = "application/jpeg"  
document.Attachments.Add(streamAttachment)  
  
Dim disp As IDisposable = fileStream  
disp.Dispose()
```

4.1.3.4 Portfolio

Essential PDF Portfolio provides support to add multiple files of different formats, created in different applications. This feature enables you to enhance the presentation of files stored in the PDF by providing support to define your own fields for attachments.

Use Case Scenarios

- You can add any number of documents into PDF Portfolio.
- You can attach files; and add custom file fields and attributes to provide your own file description.



Figure 30: PDF Portfolio

4.1.3.4.1 Creating a PDF Portfolio

The **PdfPortfolioInformation** class is used to create a PDF Portfolio document. The following code example illustrates how to create a simple PDF Portfolio.

[C#]

```
// Create a new instance of the PdfDocument class.
PdfDocument document = new PdfDocument();
```

```
// Create a new portfolio.  
document.PortfolioInformation = new PdfPortfolioInformation();  
  
// Set the view mode of the portfolio.  
document.PortfolioInformation.ViewMode = PdfPortfolioViewMode.Tile;  
  
// Create the attachment.  
PdfAttachment pdfFile = new  
PdfAttachment(GetFullTemplatePath("CorporateBrochure.pdf"));  
pdfFile.FileName = "CorporateBrochure.pdf";  
  
// Create the attachement.  
PdfAttachment wordfile = new  
PdfAttachment(GetFullTemplatePath("Stock.docx"));  
wordfile.FileName = "Stock.docx";  
  
// Create the attachement.  
PdfAttachment excelfile = new  
PdfAttachment(GetFullTemplatePath("Chart.xlsx"));  
excelfile.FileName = "Chart.xlsx";  
  
// Set the startup document for the Portfolio.  
document.PortfolioInformation.StartupDocument = pdfFile;  
  
// Add the attachment into the document.  
document.Attachments.Add(pdfFile);  
document.Attachments.Add(wordfile);  
document.Attachments.Add(excelfile);  
  
// Add a new page into the document.  
document.Pages.Add();  
  
// Save and close the document.  
document.Save("Sample.pdf");  
document.Close(true);
```

[VB.NET]

```
'Create a new instance of the PdfDocument class.  
Dim document As New PdfDocument()  
  
'Create a new portfolio.  
document.PortfolioInformation = New PdfPortfolioInformation()  
  
'Set the view mode of the portfolio.  
document.PortfolioInformation.ViewMode = PdfPortfolioViewMode.Tile  
  
'Create the attachment.  
Dim pdfFile As New PdfAttachment(GetFullPath("CorporateBrochure.pdf"))  
pdfFile.FileName = "CorporateBrochure.pdf"  
  
'Create the attachment.  
Dim wordfile As New PdfAttachment(GetFullPath("Stock.docx"))  
wordfile.FileName = "Stock.docx"  
  
'Create the attachment.  
Dim excelfile As New PdfAttachment(GetFullPath("Chart.xlsx"))  
excelfile.FileName = "Chart.xlsx"  
  
'Set the startup document for the Portfolio.  
document.PortfolioInformation.StartupDocument = pdfFile  
  
'Add the attachment into the document.  
document.Attachments.Add(pdfFile)  
document.Attachments.Add(wordfile)  
document.Attachments.Add(excelfile)  
  
'Add a new page into the document.  
document.Pages.Add()
```

```
'Save and close the document.

document.Save("Sample.pdf")
document.Close(True)
```

4.1.3.4.2 Properties

Property	Description	Type	Data Type
Schema	Gets or sets the schema field of the portfolio.	NA	PdfPortfolioSchema
ViewMode	Gets or sets the view mode of the portfolio.	NA	PdfPortfolioViewMode
StartupDocument	Gets or sets the startup document of portfolio.	NA	PdfAttachment

4.1.3.4.3 Adding Custom File Fields and Attributes into a PDF Portfolio

Custom File fields and attributes enable you to store any type of information about the attached files. You can add custom file fields and attributes into a PDF Portfolio by using the **PdfPortfolioSchema** and **PdfPortfolioAttributes** classes. The following code example illustrates how to add custom file fields and attributes into a PDF Portfolio.

[C#]

```
// Create a PDF document.
PdfDocument document = new PdfDocument();

// Create a new PDF Portfolio.
document.PortfolioInformation = new PdfPortfolioInformation();
document.PortfolioInformation.ViewMode = PdfPortfolioViewMode.Tile;

// Create a new Portfolio Schema.
PdfPortfolioSchema schema = new PdfPortfolioSchema();

// Create custom file fields.
```

```
PdfPortfolioSchemaField field = new PdfPortfolioSchemaField();
field.Name = "Price";
field.Order = 1;
field.Type = PdfPortfolioSchemaFieldType.String;
field.Visible = true;
schema.AddSchemaField(field);

field = new PdfPortfolioSchemaField();
field.Name = "Author Name";
field.Order = 2;
field.Type = PdfPortfolioSchemaFieldType.String;
field.Visible = true;
schema.AddSchemaField(field);

field = new PdfPortfolioSchemaField();
field.Name = "Year";
field.Order = 3;
field.Type = PdfPortfolioSchemaFieldType.String;
field.Visible = true;
schema.AddSchemaField(field);

// Add the Schema into the Portfolio.
document.PortfolioInformation.Schema = schema;

// Create the file attachment with custom file attributes.
PdfAttachment pdfFile = new PdfAttachment("CorporateBrochure.pdf");
pdfFile.FileName = "CorporateBrochure.pdf";
pdfFile.PortfolioAttributes = new PdfPortfolioAttributes();
pdfFile.PortfolioAttributes.AddAttributes("Price", "250");
pdfFile.PortfolioAttributes.AddAttributes("Author Name", "Syncfusion");
pdfFile.PortfolioAttributes.AddAttributes("Year", "1981");

// Create the file attachment with custom file attributes.
PdfAttachment wordFile = new PdfAttachment("Stock.docx");
```

```
wordFile.FileName = "Stock.docx";
wordFile.PortfolioAttributes = new PdfPortfolioAttributes();
wordFile.PortfolioAttributes.AddAttributes("Price", "500");
wordFile.PortfolioAttributes.AddAttributes("Author Name",
"Syncfusion");
wordFile.PortfolioAttributes.AddAttributes("Year", "1991");

document.Attachments.Add(pdfFile);
document.Attachments.Add(wordFile);
document.Pages.Add();

document.Save("sample.pdf");
```

[VB.NET]

```
'Create a PDF document.

Dim document As New PdfDocument()

'Create a new PDF Portfolio.

document.PortfolioInformation = New PdfPortfolioInformation()
document.PortfolioInformation.ViewMode = PdfPortfolioViewMode.Tile

'Create a new Portfolio Schema.

Dim schema As New PdfPortfolioSchema()

'Create custom file fields.

Dim field As New PdfPortfolioSchemaField()
field.Name = "Price"
field.Order = 1
field.Type = PdfPortfolioSchemaFieldType.String
field.Visible = True
schema.AddSchemaField(field)

field = New PdfPortfolioSchemaField()
field.Name = "Author Name"
```

```
field.Order = 2
field.Type = PdfPortfolioSchemaFieldType.String
field.Visible = True
schema.AddSchemaField(field)

field = New PdfPortfolioSchemaField()
field.Name = "Year"
field.Order = 3
field.Type = PdfPortfolioSchemaFieldType.String
field.Visible = True
schema.AddSchemaField(field)

'Add the Schema into the Portfolio.
document.PortfolioInformation.Schema = schema

'Create the file attachment with custom file attributes.
Dim pdfFile As New PdfAttachment("CorporateBrochure.pdf")
pdfFile.FileName = "CorporateBrochure.pdf"
pdfFile.PortfolioAttributes = New PdfPortfolioAttributes()
pdfFile.PortfolioAttributes.AddAttributes("Price", "250")
pdfFile.PortfolioAttributes.AddAttributes("Author Name", "Syncfusion")
pdfFile.PortfolioAttributes.AddAttributes("Year", "1981")

'Create the file attachment with custom file attributes.
Dim wordFile As New PdfAttachment("Stock.docx")
wordFile.FileName = "Stock.docx"
wordFile.PortfolioAttributes = New PdfPortfolioAttributes()
wordFile.PortfolioAttributes.AddAttributes("Price", "500")
wordFile.PortfolioAttributes.AddAttributes("Author Name", "Syncfusion")
wordFile.PortfolioAttributes.AddAttributes("Year", "1991")

document.Attachments.Add(pdfFile)
document.Attachments.Add(wordFile)
document.Pages.Add()
```

```
document.Save("sample.pdf")
```

4.1.4 Security

Security features of Adobe enables to authenticate and authorize any person reviewing their PDF reports. This section demonstrates various security features supported by Essential PDF. It includes the following topics.

- Encryption - This topic demonstrates how the PDF file is encrypted and decrypted with password by using Essential PDF
- Signing - This topic demonstrates various signatures supported by Essential PDF and how they can be applied to a PDF document

Note: You must add the `Syncfusion.Pdf.Security` namespace to work with interactive features.

4.1.4.1 Encryption

The PDF document can be encrypted with 40, 128 and 256 bit key to protect its contents from unauthorized access. Document permissions are managed with owner and user passwords and access rights of the document.

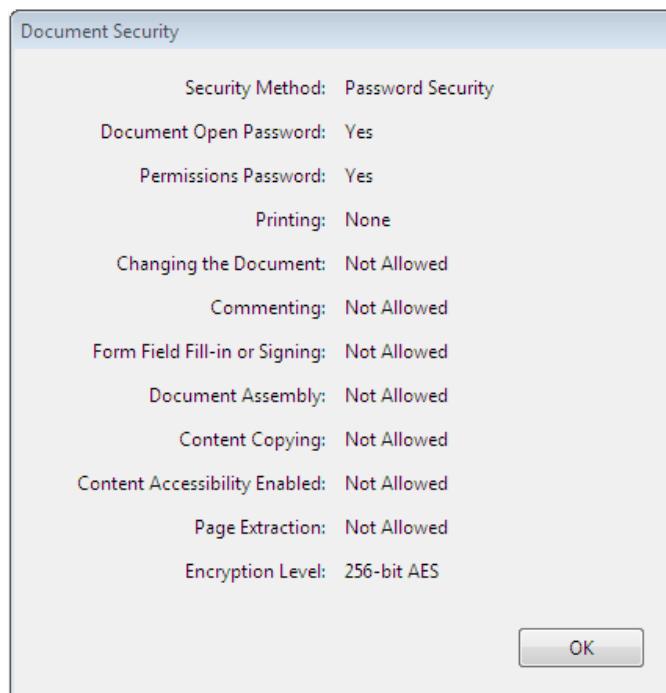


Figure 31: Encryption Level

PdfSecurity

PdfSecurity is a class that enables to manage security properties.

Owner Password

Opening the document with the correct owner password (assuming it is not the same as the user password), allows full access to the document. This unlimited access includes the ability to change the document's passwords and access permissions.

User Password

Opening the document with the correct user password (or opening a document that does not have a user password), allows additional operations to be performed according to the user access permissions, specified in the document's permissions.

Permissions

Essential PDF provides support to set restricted document operations on the generated PDF document. This ensures that the end-user is granted restricted permission to interact with the generated PDF document. This includes:

- Modifying the document's contents.
- Copying or otherwise extracting text and graphics from the document, including extraction for accessibility purposes.
- Adding or modifying text annotations and interactive form fields.
- Printing the document.

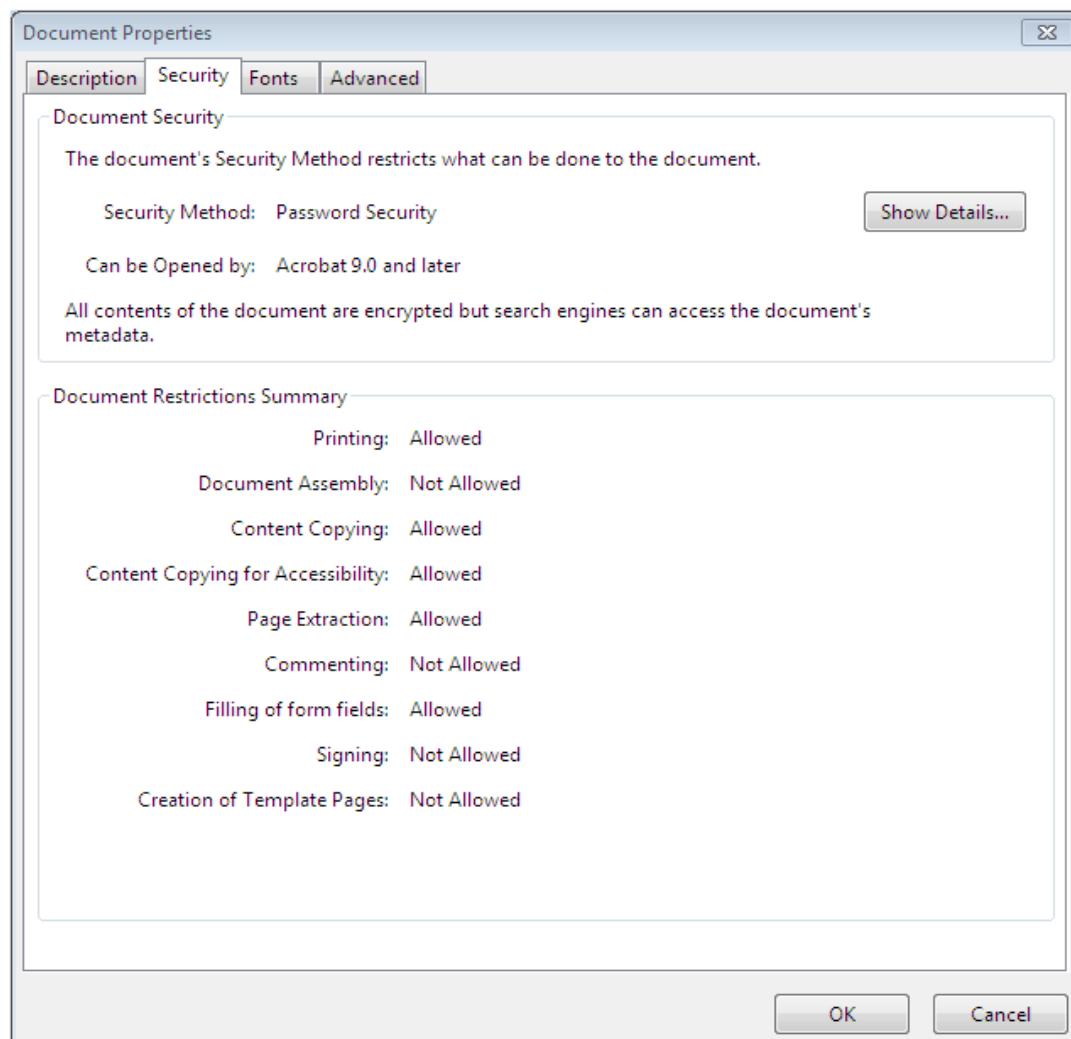


Figure 32: Restricted Access

4.1.4.2 Signing

A digital signature is an electronic signature that is used to authenticate the identity of the sender of a message or the signer of a document, and to ensure that the original content of the message or document that has been sent is unchanged.

Digital signatures are easily transportable, cannot be imitated by someone else, and can be automatically time-stamped. It has the ability to ensure that once the original signed message is received, the sender cannot easily repudiate it later.

A digital signature is used to authenticate the identity of a user and the document's contents. It stores information about the signer and the state of the document at the moment of signing.

PdfSignature

It is a base class that provides the functionality to create either visible or invisible signature on the page. To create an invisible signature, just set the signature size to zero. Later, the visibility of the signature is verified by using the `Visible` property. `PdfSignature` contains information about signer, signing location, signing reason, and so on. The following table lists the public properties of this class.

Name	Description
Appearance	Gets the signature appearance.
Bounds	Gets or sets bounds of signature.
Certificate	Gets signing certificate.
Certificated	Gets or sets a value indicating certificate document or not. Note: Works only with Adobe Reader 7.0.8 or higher.
ContactInfo	Gets or sets information provided by the signer to enable a recipient to contact the signer to verify the signature; for example, a phone number.
DocumentPermissions	Gets or sets the permission for certificated document.
Field	Gets pdf signature field.
Location	Gets or sets signature location on the page.
LocationInfo	Gets or sets the physical location of the signing.
Reason	Gets or sets the reason of signing.
Visible	Gets a value indicating whether the signature is visible or not.

TimeStampServer	Sets the timestamp for the signature.
-----------------	---------------------------------------

Features

- **Appearance:** PdfAppearance allows you to draw and create custom appearance on the PdfSignature field.
- **Certificated:** Allows document recipients to know, if any changes have been made, contrary to the author's intent.
- **DocumentPermissions:** Allows you to set permissions on certificated document with help of the PdfCertificationFlags.
- **Visible / Invisible Signature:** Allows you to create visible or invisible signatures by enabling the **Visible** property.
- **TimeStampServer :** Allows you to include timestamp for the digital signature.

PdfCertificate

When you use digital signatures, each user is given a digital certificate. This certificate is actually a small file on a disk or on another device, such as a smart card. Each certificate contains a unique code, and the certificate imprints this code on each signature you create with it. This means that all of your signatures can be traced back to your certificate, and the certificate itself can be traced back to you. In this way, digital signatures identify you through a clear chain of ownership.

It is a class that provides the functionality to use certificates for PdfSignature from PFX files or local Certification Storage. Certificates in local storage are found by using static methods **FindByIssuer**, **FindBySubject**, **FindBySerialId**. Also, there is the **GetCertificates** static method, which allows to get an array of all certificates from the local storage.

Standard Signature

PdfCertificate class is used for getting the certificates from disk and **PdfSignature** class is used to sign a document with the given certificate. PdfSignature class has methods and properties that allow to set the signature information such as reason, location information, bounds where the signature has to be placed, and contact information.

The following code example illustrates this.

[C#]
<pre>PdfCertificate pdfCert = new PdfCertificate("PDF.pfx", "111"); // Find certificate by Issuer.</pre>

```

PdfCertificate findByIssuer = PdfCertificate.FindByIssuer(sType,
pdfCert.IssuerName);

// Draw the signature.
PdfSignature signature = new PdfSignature(page, pdfCert, "Signature");
signature.Bounds = new RectangleF(0, 0, 250, 100);

// Set Signature info.
signature.Reason = "I am author of this document.";

```

[VB.NET]

```

Dim pdfCert As PdfCertificate = New PdfCertificate("PDF.pfx", "111")

' Find certificate by Issuer.
Dim findByIssuer As PdfCertificate = PdfCertificate.FindByIssuer(sType,
pdfCert.IssuerName)

' Draw the signature.
Dim signature As PdfSignature = New PdfSignature(page, pdfCert, "Signature")
signature.Bounds = New RectangleF(0, 0, 250, 100)

' Set Signature info.
signature.Reason = "I am author of this document."

```

Author Signature

By default, documents are signed with standard signature types. **Certificated** property of PdfSignature is used to create an author signature. When signed with this type of signature, any modification after signing will be detected, and hence does not provide support to add multiple signatures.

Note: This implementation of certification will only work in Acrobat 7 and higher versions.

The following code example illustrates this.

[C#]

```

PdfSignature signature = new PdfSignature(page, pdfCert, "Signature");
// Setting the certified signature.
signature.Certificated = true;

```

[VB .NET]

```
Dim signature As PdfSignature = New PdfSignature(page, pdfCert, "Signature")
' Setting the certified signature.
signature.Certificated = True
```

Timestamp in digital signature

Essential PDF supports addition of timestamp in digital signatures. The date and time at which the document is signed can be added as a part of the signature. Timestamps are easier to verify when they're associated with a timestamp authority's trusted certificate. Including a timestamp helps to establish exactly when the document is signed and reduces the chances of an invalid signature. The timestamp can be obtained from a third-party timestamp authority or from the certificate authority that issued the digital ID.

Timestamps appear in the signature field and in the Signature Properties dialog box. If the timestamp is included, the certificate will appear in the **Date/Time** tab of the **Signature Properties** dialog box. If no timestamp is added the signature field displays the local time of the computer at the moment of signing.

To apply timestamp using Essential PDF, the **TimeStampServer** property of the **PdfSignature** class has to be used. The parameters for the TimeStampMethod are the URI of digital server, username and password.

The following code illustrates the method for adding timestamp in the digital signature.

[C#]

```
//Get certificate.
PdfCertificate pdfCert = new PdfCertificate(@"..\..\Data\PDF.pfx",
"syncfusion");

//Sign the document with timestamp.
PdfSignature signature = new PdfSignature(page, pdfCert, "Signature");

//Add time stamp using the server URI and credentials.
signature.TimeStampServer = new TimeStampServer(
    new Uri("http://digistamp.syncfusion.com"), "user",
"123456");
```

[VB]

```
'Get certificate.
Dim pdfCert As New PdfCertificate("../..\Data\PDF.pfx", "syncfusion")

'Sign the document with timestamp.
Dim signature As New PdfSignature(page, pdfCert, "Signature")
```

```
'Add time stamp using the server URI and credentials.
signature.TimeStampServer = New TimeStampServer(New
Uri("http://digistamp.syncfusion.com"), "user", "123456")
```

Multiple Signatures

You can add multiple signatures to the document with incremental updates by using standard signatures.

Note: Currently only self-created and third-party .pfx certificates are supported.

4.1.5 Standards for PDF/A-1 Compliance

PDF elements are standardized under ISO for several constituencies. This section deals with following standards that are supported by Essential PDF.

- PDF/A-This topic demonstrates PDF/A-1b standard, which is used for archiving in environments like corporate, government, and library.
- PDF/X-This topic discusses the PDF/X-1a standard that is mainly available for standardizing printing and graphics.

4.1.5.1 PDF/A-1b

PDF/A

The PDF/A formats specified in the ISO 19005 standards strive to provide a mechanism for representing electronic documents. These documents are represented in a manner that preserves their visual appearance over time, independent of the tools and systems used for creating, storing, or rendering the files. A key element to this reproducibility is the requirement for PDF/A documents to be 100 percent self-contained.

All of the information necessary for displaying the document in the same manner every time is embedded in the file. This includes all visible content like text, raster images, vector graphics, fonts, and color information. The standard is based on PDF 1.4, and imposes some restrictions regarding the use of color, fonts, annotations, and other elements.

There are two types of PDF/A-1:

- **ISO 19005-1 Level B conformance (PDF/A-1b)** ensures that the visual appearance of a document is preservable over the long term. PDF/A-1b ensures that the document will look the same, when it is processed sometime in the future.
- **ISO 19005-1 Level A conformance (PDF/A-1a):** It is based on level B, but adds crucial properties from »Tagged PDF«. It requires structure information and reliable text semantics in order to preserve the document's logical structure and natural reading order. PDF/A-1a not only ensures that the document will look the same when it is used in the future, but also ensures that its contents can be reliably interpreted and accessed by physically impaired users.



Note: *PDF/A-1a and PDF/A-1b differ primarily with respect to text extraction.*

PDF/A-1b

Creating a PDF/A-1b document is very simple. You must set PdfConformanceLevel to PdfA1B while creating an instance to the PDF document. PDF/A standard imposes some restrictions regarding the usage of color, fonts, annotations, and other elements. These restrictions are:

- The use of JavaScript is forbidden.
- Attaching any file to a PDF document is forbidden.
- Hyperlinking to a non-PDF file is forbidden.
- Security features are forbidden.
- The use of Form Fields is forbidden.
- Text containing HTML tags is forbidden.
- Supports the use of TrueType fonts only, does not support Type1 font.
- Supports the use of RGB color, does not support CMYK color.

Validating PDF/A1-b

Adobe Acrobat Preflight tool is used to verify the compliance of a PDF document with the PDF/A standard.

You can verify the compliance of a PDF file by using the Preflight tool. Using the menu options select Advanced > Preflight > PDF/A compliance > Verify compliance with PDF/A-1b.

The following code example illustrates how to create PDF/A-1b compliant output:

[C#]

```
//Create a new document with PDF/A standard.
PdfDocument doc = new PdfDocument(PdfConformanceLevel.Pdf_A1B);
```

```
//Add a page.
PdfPage page = doc.Pages.Add();

//Create Pdf graphics for the page.
PdfGraphics g = page.Graphics;

//Create a solid brush.
PdfBrush brush = new PdfSolidBrush(Color.Black);
float fontSize = 20f;
Font f = new Font("Helvetica", fontSize, FontStyle.Regular);

//Set the font.
PdfFont font = new PdfTrueTypeFont(f, true);

//Draw the text.
g.DrawString("Hello world!", font, brush, new PointF(20, 20));
doc.Save("Sample.pdf");
```

[VB.NET]

```
'Create a new document with PDF/A standard.
Dim doc As PdfDocument = New PdfDocument(PdfConformanceLevel.Pdf_A1B)

'Add a page.
Dim page As PdfPage = doc.Pages.Add()

>Create Pdf graphics for the page.
Dim g As PdfGraphics = page.Graphics

>Create a solid brush.
Dim brush As PdfBrush = New PdfSolidBrush(Color.Black)
Dim fontSize As Single = 20f
Dim f As Font = New Font("Helvetica", fontSize, FontStyle.Regular)

'Set the font.
Dim font As PdfFont = New PdfTrueTypeFont(f, True)

'Draw the text.
g.DrawString("Hello world!", font, brush, New PointF(20, 20))
doc.Save("Sample.pdf")
```

4.1.5.2 PDF/X-1a

PDF/X is a subset of the Adobe Portable Document Format (PDF) specification, which exhibits best practices in graphic arts file exchange. PDF/X-1a restricts the content in a PDF document that does not directly serve the purpose of high-quality print production output, such as annotations, Java Actions, and embedded multimedia.

PDF/X-1a also eliminates the most common errors in file preparation. Sending the document as a PDF/X-1a file, will guarantee that font errors do not occur. This is because a file declared as complying with the PDF/X-1a standard meets the following requirements:

- All fonts and images must be embedded.
- All elements must be encoded as CMYK.

In addition,

- MediaBox and TrimBox or ArtBox must be defined; BleedBox is optional
- The output intent must be specified either by stating a Characterized Printing Condition or identifying an ICC output profile.

Advantages of PDF/X-1a

By using a PDF/X-1a workflow,

- Print-ready files will reproduce as you intended.
- You will save time and money.

The following code snippet illustrates how to create a pdf document that complies with the above standard.

[C#]

```
//Create the document.
PdfDocument doc = new PdfDocument(PdfConformanceLevel.Pdf_X1A2001);

//Set the color space. Should be CMYK.
doc.ColorSpace = PdfColorSpace.CMYK;

//Save and close the document.
doc.Save("sample.pdf");
```

[VB .NET]

```
'Create the document.
Dim doc As New PdfDocument(PdfConformanceLevel.Pdf_X1A2001)

'Set the color space. Should be CMYK.
doc.ColorSpace = PdfColorSpace.CMYK

'Save and close the document.
doc.Save("sample.pdf")
```

4.1.6 Settings

This section demonstrates various document settings and compression settings available in Essential PDF.

- Document Settings-This topic provides details on setting various document properties and meta data writing.
- Compression-This topic explains how a document is compressed by using Essential PDF.

4.1.6.1 Document Settings

The Document settings help in storing information about the document. Extensible Metadata Platform (XMP) is a technology that enables to embed metadata.

 Metadata is the data that describes a file into the file itself.

It uses XML as the syntax for metadata description. XMP is provided with the following schemas:

- Basic Schema
- Dublin Core Schema
- Rights Management Schema
- Basic Job Ticket Schema
- Paged-Text Schema
- PDF Schema

The document properties of Adobe are set by using either the **XMP's PDF schema** or the **DocumentInformation** method of the PdfDocument. The properties that can be set are as follows.

- Author
- CreationDate
- Keywords
- Producer
- Subject
- Title

The following code snippet illustrates setting the document properties such as **Title**, **Author** and **Keywords**.

[C#]

```
// Setting various Document Properties.
pdfDoc.DocumentInformation.Title = "Document Properties Information";
pdfDoc.DocumentInformation.Author = "Syncfusion";
pdfDoc.DocumentInformation.Keywords = "PDF";

// XMP Basic Schema.
BasicSchema basic = xmp.BasicSchema;
basic.Advisory.Add("advisory");
basic.BaseURL = new Uri("http://google.com");
```

[VB .NET]

```
' Setting various Document Properties.
pdfDoc.DocumentInformation.Title = "Document Properties Information"
pdfDoc.DocumentInformation.Author = "Syncfusion"
pdfDoc.DocumentInformation.Keywords = "PDF"

' XMP Basic Schema.
Dim basic As Syncfusion.Pdf.Xmp.BasicSchema = xmp.BasicSchema
basic.Advisory.Add("advisory")
basic.BaseURL = New Uri("http://google.com")
```

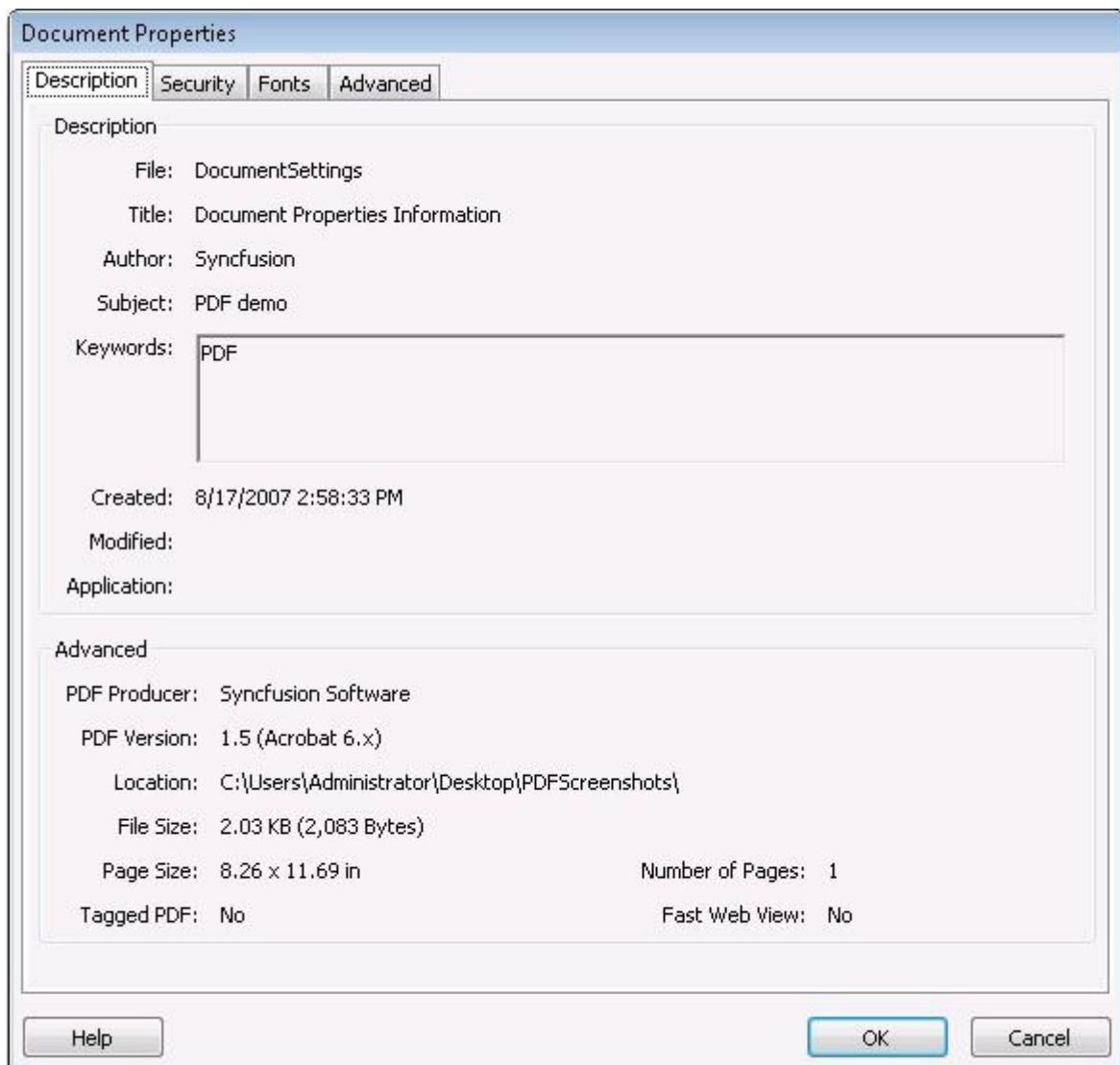


Figure 33: Document Properties

4.1.6.2 Custom Metadata

Essential PDF allows you to add required metadata (custom metadata) to a PDF document. Custom metadata can be information about the document which cannot be fit in the predefined metadata fields. For example: If a metadata field "Link" is available, you can only provide a link there. But, Essential PDF allows you to add additional information like Author, date of creation etc. about the link.

This feature allows you to add as many metadata fields as you want. Only new metadata fields can be added. You cannot add metadata fields under the predefined metadata fields.

How to add a Custom Metadata Field

To add a custom metadata field,

- Add namespace in the project
- Create an XML document container
- Create a custom schema

Adding Namespace in the Project

The user needs to add the namespace Syncfusion.Pdf.Xmp in the project to enable addition of custom metadata fields in the PDF document.

Add the namespace using the following code.

[C#]

```
using Syncfusion.Pdf.Xmp;
```

Creating an XML Document Container

The custom metadata to be created has to be stored and linked to the PDF document in use. Here an XML document is used as a container to save the custom metadata fields for the PDF document.

Add the following code to create an XML document to store custom metadata fields.

[C#]

```
XmpMetadata xmp = new  
XmpMetadata(pdfDoc.DocumentInformation.XmpMetadata.XmlData);
```

The following table provides more information on the code.

Property	Type	Value It Accepts
XmlData	XmlElement	XmlElement

Creating a Custom Schema

A custom schema defines the structure of the customized information records. You can use the `CustomSchema` class to:

- Define custom metadata files; and,
- Add them to the PDF document

Add the following code to define a custom schema.

[C#]

```
//create a custom schema
CustomSchema cs = new CustomSchema(xmp, "custom", "http://www.syncfusion.com");
cs["Author"] = "Syncfusion";
cs["creationDate"] = DateTime.Now.ToString();
cs["DOCID"] = "SYNCSAM001";
```

The code snippet above, illustrates creation of custom schema or first-level metadata field `www.syncfusion.com`, which is a link. The second-level metadata fields under the link are Author, creation date and DocID.

On running the code, the values assigned to these fields are reflected as data on expanding the first-level metadata field as shown in the following screenshot.

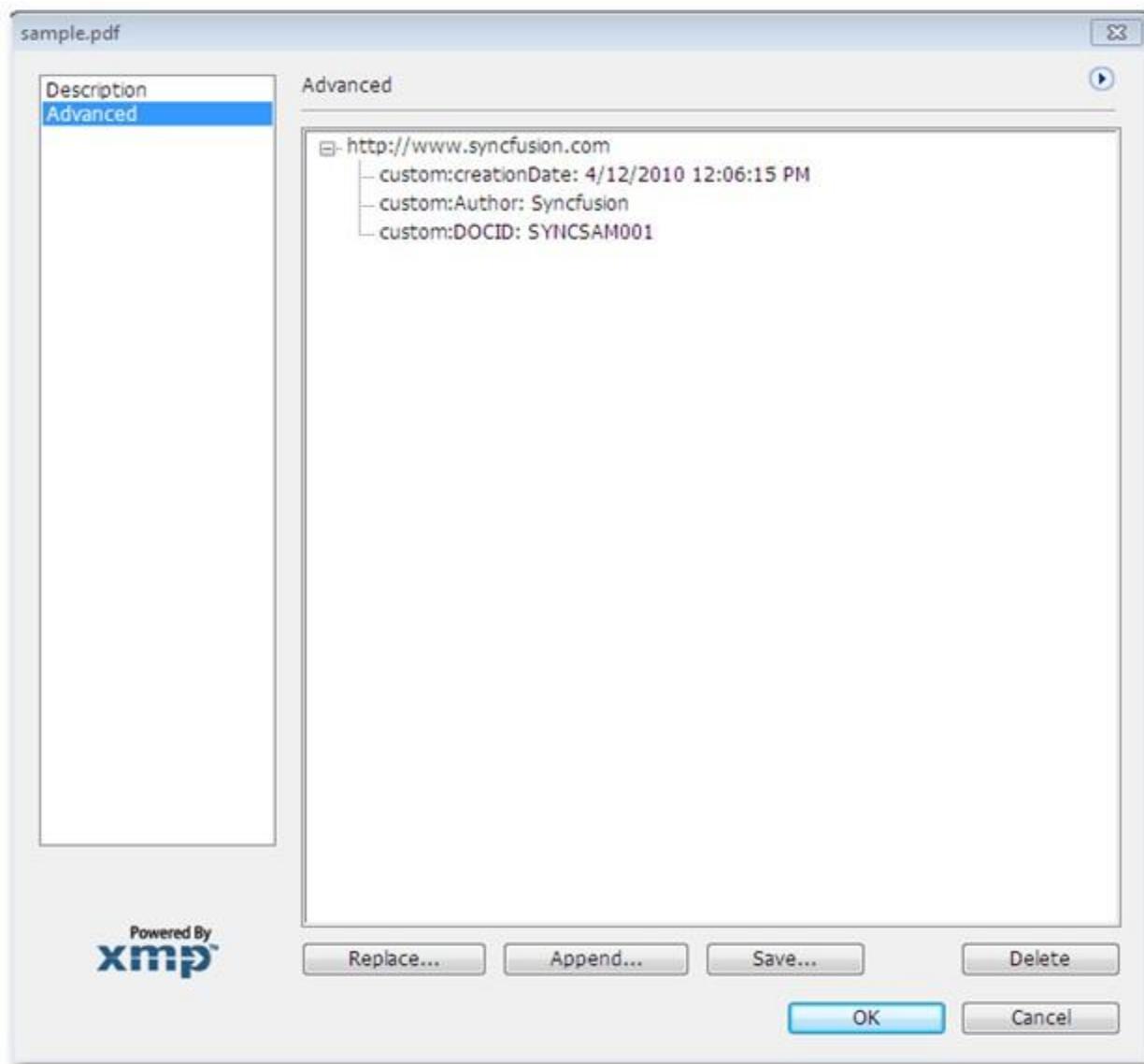


Figure 34: Custom Metadata Field

You have successfully created a custom metadata field in the PDF document.

4.1.6.3 Compression

Compression is the process of reducing the size of data in order to save space or transmission time.

For data transmission, compression can be performed on any of the following depending on a number of factors:

- Just the data content, or
- Entire transmission unit.

Content compression

Content compression involves following:

- Removing all extra space characters
- Inserting a single repeat character to indicate a string of repeated characters
- Substituting smaller bit strings for frequently occurring characters

Advantages of Content compression

- Reduces a text file to 50 percent of its original size



Note: Compression is performed by a program that uses a formula or algorithm to determine how to compress or decompress data. The algorithm is one of the critical factors to determine the compression quality. This is elaborated below.

Controlling the Compression Levels

Essential PDF controls the compression level of the document by using the **PdfCompressionLevel** class with the help of the LZW and zlib/deflate compression algorithms. Both LZW and Flate algorithms compress either binary data or ASCII text and always produces the binary data.

The following compression levels are supported by Essential PDF:

- **None**-Packs without compression
- **BestSpeed**-Performs high speed compression; reduction of data size is lesser
- **BelowNormal**-Performs compression that is rated between Normal and BestSpeed compressions
- **Normal**-Performs compression at normal speed; normal reduction of data size
- **AboveNormal**-Performs enhanced compression compared to the normal compression; time consumption exceeds the normal compression
- **Best**-Performs the best compression; time consuming

4.1.7 Tutorial

This tutorial will show you how easy it is to get started using Essential PDF. It will give you a basic introduction to the concepts you need to know before getting started with the product and some tips and ideas on how to implement PDF into your projects. The lessons in this tutorial are meant to introduce you to PDF with simple step-by-step procedures.

Note: Refer to the [Class Reference documentation](#) for comprehensive information on the classes.

The features are discussed under the following sections

4.1.7.1 Drawing Text

This section focuses on demonstrating how the text can be drawn with fewer lines of code. It comprises the following topics.

4.1.7.1.1 Draw Simple text

Drawing Text in a PDF document is made simpler and similar to .NET GDI. This section demonstrates how a string is drawn in a PDF page by using Essential PDF.

The process is very similar to drawing any other object on the PDF page. The string is drawn by using the **DrawString** method of the **Graphics** class. You also need to specify the font and brush with which you want the string to be drawn.

```
[C#]

//Creates a new PDF document.
PdfDocument doc = new PdfDocument();

//Adds a page to the document.
PdfPage page = doc.Pages.Add();

//Creates Pdf graphics for the page
PdfGraphics g = page.Graphics;

//Creates a solid brush.
PdfBrush brush = new PdfSolidBrush(Color.Black);

//Sets the font.
PdfFont font = new PdfStandardFont(PdfFontFamily.Helvetica, fontSize);
```

```
//Draws the text.  
g.DrawString("Hello world!", font, brush, new PointF(20, 20));  
  
//Saves the document.  
doc.Save("Sample.pdf");
```

[VB.NET]

```
'Creates a new PDF document.  
Dim doc As PdfDocument = New PdfDocument()  
  
'Adds a page to the document.  
Dim page As PdfPage = doc.Pages.Add()  
  
'Creates Pdf graphics for the page  
Dim g As PdfGraphics = page.Graphics  
  
'Creates a solid brush  
Dim brush As PdfBrush = New PdfSolidBrush(Color.Black)  
  
'Sets the font  
Dim font As PdfFont = New PdfStandardFont(PdfFontFamily.Helvetica, fontSize)  
  
'Draws the text  
g.DrawString("Hello world!", font, brush,new PointF(20,20))  
  
'Saves the document.  
doc.Save("Sample.pdf")
```



Figure 35: Simple Text

4.1.7.1.2 Text Pagination

Text in a PDF document can flow through multiple pages. You can specify different formats for the text element using **PDFStringFormat** class of Essential PDF. The following code snippet illustrates how to draw the text element of a PDF document with the custom formats.

```
[C#]  
  
//Create a new PDF document.  
PdfDocument doc = new PdfDocument();  
  
//Add a page to the document.  
PdfPage page = doc.Pages.Add();
```

```

//Read the text from the text file
string path = "../../../../../Data/SampleText.txt";
StreamReader reader = new StreamReader(path, Encoding.ASCII);
string text = reader.ReadToEnd();
reader.Close();

//Set the formats for the text
PdfStringFormat format = new PdfStringFormat();
format.Alignment = PdfTextAlignment.Justify;
format.LineAlignment = PdfVerticalAlignment.Top;
format.ParagraphIndent = 15f;

//Create a text element
PdfTextElement element = new PdfTextElement(text, font);
element.Brush = new PdfSolidBrush(Color.Black);
element.StringFormat = format;
element.Font = new PdfStandardFont(PdfFontFamily.Helvetica, 12);

//Set the properties to paginate the text.
PdfLayoutFormat layoutFormat = new PdfLayoutFormat();
layoutFormat.Break = PdfLayoutBreakType.FitPage;
layoutFormat.Layout = PdfLayoutType.Paginate;

//Draw the text element with the properties and formats set.
PdfTextLayoutResult result = element.Draw(page, bounds, layoutFormat);

//Save the document.
doc.Save("Sample.pdf");

```

[VB.NET]

```

'Create a new PDF document.
Dim doc As PdfDocument = New PdfDocument()

'Set compression level
doc.Compression = PdfCompressionLevel.None

'Add a page to the document.
Dim page As PdfPage = doc.Pages.Add()

'Read the text from the text file
Dim path As String = "../../../../../Data/SampleText.txt"
Dim reader As StreamReader = New StreamReader(path, Encoding.ASCII)
Dim text As String = reader.ReadToEnd()
reader.Close()

```

```
'Set the formats for the text
Dim format As PdfStringFormat = New PdfStringFormat()
format.Alignment = PdfTextAlignment.Justify
format.LineAlignment = PdfVerticalAlignment.Top
format.ParagraphIndent = 15f

'Create a text element
Dim element As PdfTextElement = New PdfTextElement(text, font)
element.Brush = New PdfSolidBrush(Color.Black)
element.StringFormat = format
element.Font = New PdfStandardFont(PdfFontFamily.Helvetica, 12)

'Set the properties to paginate the text.
Dim layoutFormat As PdfLayoutFormat = New PdfLayoutFormat()
layoutFormat.Break = PdfLayoutBreakType.FitPage
layoutFormat.Layout = PdfLayoutType.Paginate
Dim bounds As RectangleF = New RectangleF(PointF.Empty,
page.Graphics.ClientSize)

'Draw the text element with the properties and formats set.
Dim result As PdfTextLayoutResult = element.Draw(page, bounds, layoutFormat)

'Save the document.
doc.Save("Sample.pdf")
```

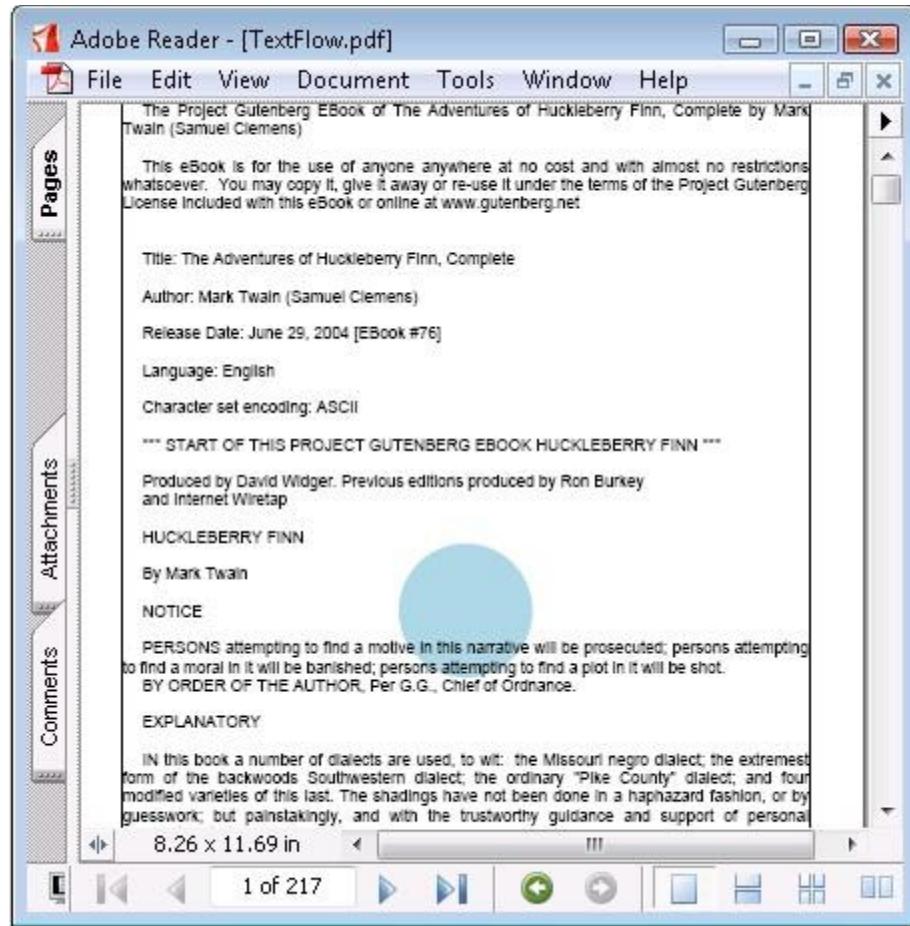


Figure 36: Text Pagination

4.1.7.1.3 Draw Rich text

RTF (Rich Text Format) is a standard specification for the formatting of documents. They are ASCII files with special commands to indicate formatting information such as margins and fonts.

Essential PDF supports drawing an RTF text into a PDF document by converting it as a bitmap or metafile image.

- Converting RTF text into a bitmap file, provides improved performance
- Converting RTF text into a metafile image provides high resolution and searchable text.

The following code illustrates how to draw an RTF text into bitmap and metafile formats.

[C#]

```
//Draw RTF as metafile
PdfMetafile metafile = ( PdfMetafile )PdfImage.FromRtf( text, bounds.Width,
PdfImageType.Metafile );
PdfMetafileLayoutFormat format = new PdfMetafileLayoutFormat();

//Allow pagination without any breaks at page breaks.
format.SplitTextLines = false;

//Draw the image.
metafile.Draw( page, 0, 0, format );

//Draw RTF as Bitmap
bitmap = PdfImage.FromRtf( text, bounds.Width, PdfImageType.Bitmap );
bitmap.Draw(page, 0, 0, format)
```

[VB.NET]

```
'Convert it as metafile image.
Dim metafile As PdfMetafile = CType(PdfImage.FromRtf(text, bounds.Width,
PdfImageType.Metafile), PdfMetafile)
Dim format As PdfMetafileLayoutFormat = New PdfMetafileLayoutFormat()

'Allow the text to flow multiple pages without any breaks.
format.SplitTextLines = False

'Draw the image.
metafile.Draw(page, 0, 0, format)

'Draw RTF as Bitmap
Dim bitmap As PdfImage = PdfImage.FromRtf(text, bounds.Width,
PdfImageType.Bitmap)
bitmap.Draw(page, 0, 0, format)
```

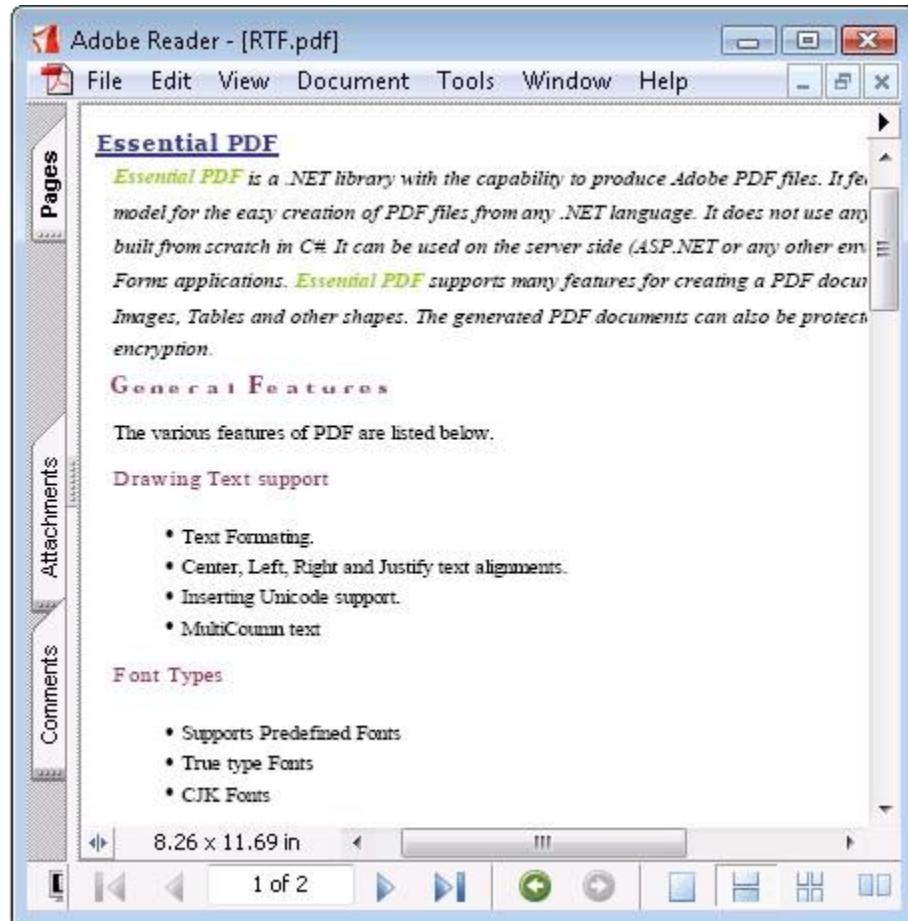


Figure 37: RTF Support

4.1.7.1.4 Draw Right-To-Left Text

Essential PDF provides support for drawing RTL languages into the PDF document.

Middle eastern languages such as Hebrew and Arabic are written predominantly from right-to-left. Numbers are written with the most significant digit left-most, just as in European or other left-to-right text. Languages written in left-to-right scripts are often mixed; hence the complete document is bidirectional in nature; a mix of both right-to-left (RTL) and left-to-right (LTR) writing. Text written in the Hebrew and Arabic languages are often referred to as bidirectional or "bidi" in short.

[C#]

```
//Set the font with unicode option
Font f = new Font("Arial", 14);
```

```
PdfFont font = new PdfTrueTypeFont(f, true);

//Set the format for string
PdfStringFormat format = new PdfStringFormat();

//Set the format as RTL type
format.RightToLeft = true;

//Set the alignment
format.Alignment = PdfTextAlignment.Right;

//Draw the RTL text
page.Graphics.DrawString(text, font, brush, rect, format);
```

[VB.NET]

```
'Set the font with unicode option
Dim f As Font = New Font("Arial", 14)
Dim font As PdfFont = New PdfTrueTypeFont(f, True)

'Set the format for string
Dim format As PdfStringFormat = New PdfStringFormat()

'Set the format as RTL type
format.RightToLeft = True

'Set the alignment
format.Alignment = PdfTextAlignment.Right

'Draw the RTL text
page.Graphics.DrawString(text, font, brush, rect, format)
```

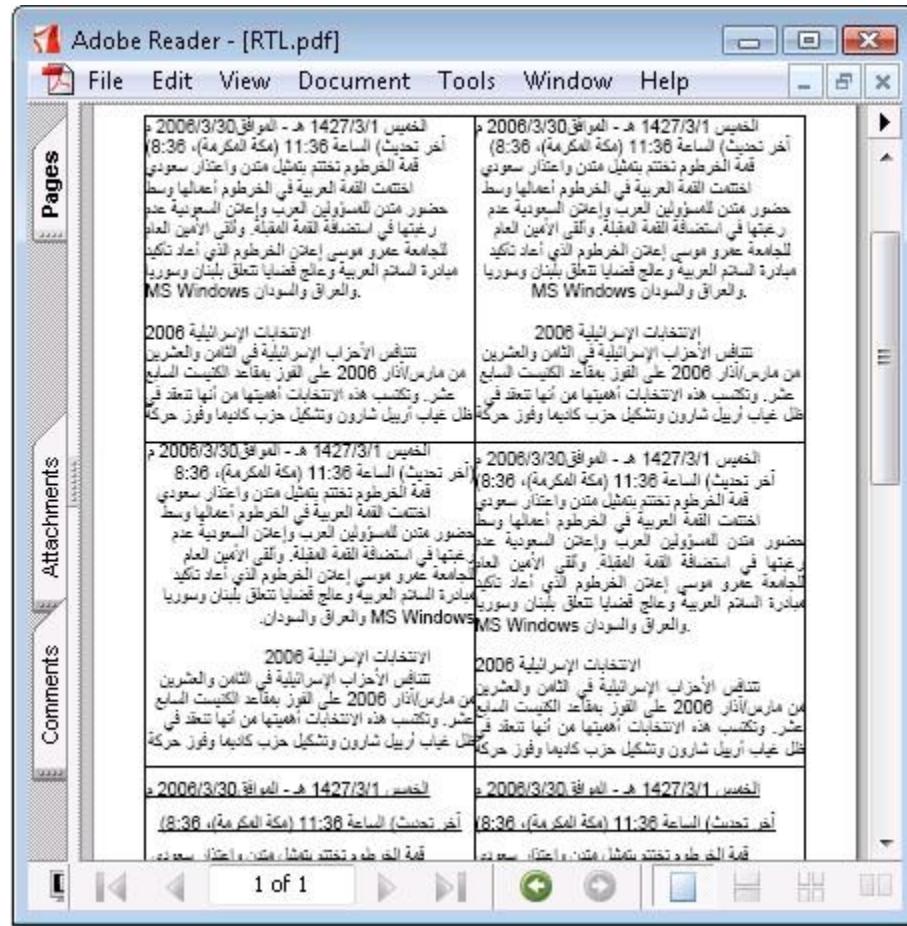


Figure 38: PDF with RTL Support

4.1.7.2 Drawing Shapes

Essential PDF has a comprehensive set of APIs that can be used for drawing a variety of shapes such as,

- Rectangles
- Circles
- Arcs
- Curves and so on.

Shapes are filled by using different types of brushes like gradient brush, Tiling brush, and so on. Essential PDF supports drawing of shapes with different color spaces. Transparency of shapes can also be set.

PdfGraphics class allows drawing a wide range of primitives like

- Lines
- Curves
- Paths
- Text

For each such operation there is a set of methods like `Draw<primitive>()` (for example: `DrawLine`).

Each set of methods accepts parameters specific to each primitive type (for example: pen, brush, boundaries, etc.).

- If pen is used, the primitive will be drawn
- If brush is used, the primitive will be filled.



Note: You must add the `Syncfusion.Pdf.Graphics` namespace to work with graphic objects.

The following code example illustrates how to draw shapes.

[C#]

```
//Draws polygon with pen and brush.
PdfGraphics g = page.Graphics;
PdfPen pen = new PdfPen(Color.Brown);
PdfSolidBrush brush = new PdfSolidBrush(Color.Green);
g.DrawPolygon(pen, brush, points);
```

[VB .NET]

```
'Draws polygon with pen and brush.
Dim g As PdfGraphics = page.Graphics
Dim pen As PdfPen = New PdfPen(Color.Brown)
Dim brush As PdfSolidBrush = New PdfSolidBrush(Color.Green)
g.DrawPolygon(pen, brush, points)
```

You can paginate the element as follows.

[C#]

```
PdfEllipse ellipse = new PdfEllipse(rect);

//Set layout property to make the element break across the pages.
PdfLayoutFormat format = new PdfLayoutFormat();
format.Break = PdfLayoutBreakType.FitPage;
format.Layout = PdfLayoutType.Paginate;
ellipse.Brush = PdfBrushes.Brown;

//Draw ellipse.
ellipse.Draw(page, 20, 20, format);
```

[VB.NET]

```
Dim ellipse As PdfEllipse = New PdfEllipse(rect)

' Set layout property to make the element break across the pages.
Dim format As PdfLayoutFormat = New PdfLayoutFormat()
format.Break = PdfLayoutBreakType.FitPage
format.Layout = PdfLayoutType.Paginate
ellipse.Brush = PdfBrushes.Brown

'Draw ellipse.
ellipse.Draw(page, 20, 20, format)
```

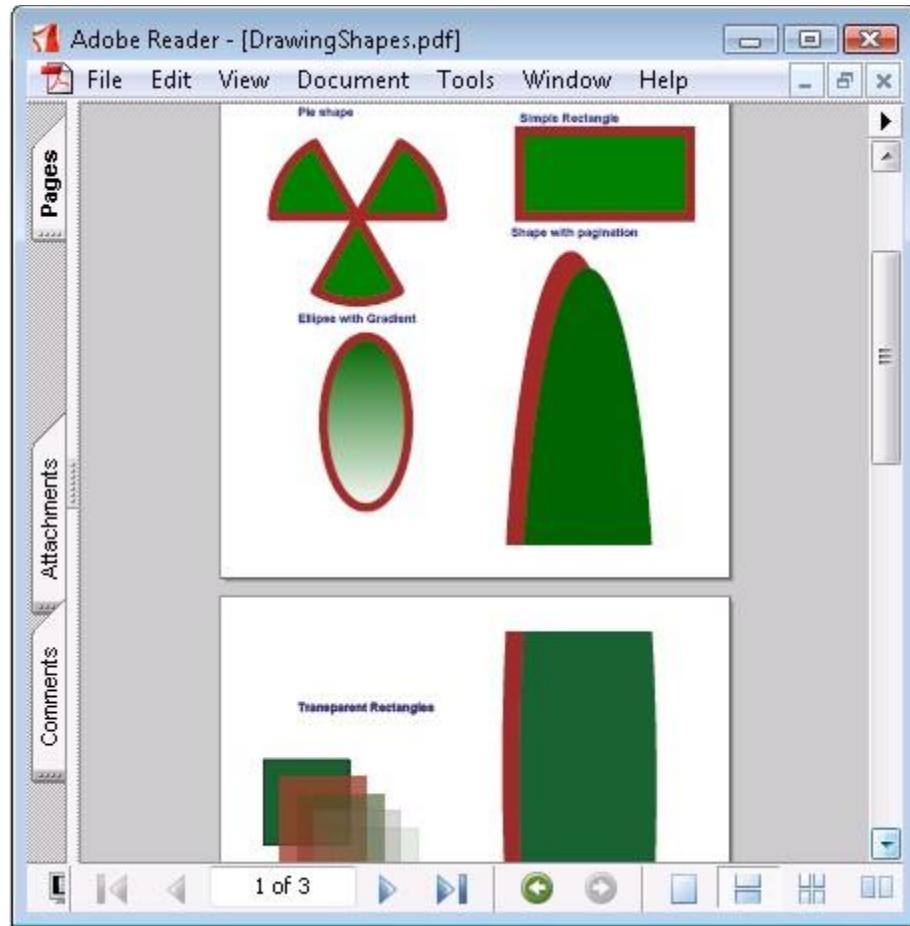


Figure 39: PDF Document drawn with Shapes

4.1.7.3 Drawing Images

Essential PDF consists of a comprehensive set of APIs that are used to create Adobe PDF documents with text and rich graphics including custom drawing and images. It also has support for loading images from the following objects:

- Streams
- Files on disk
- System.Drawing.Bitmap

You can resize and insert images into the PDF document at the required size, and specify the image quality by using the **Quality** property.

1 Advanced features like transparency and soft masking are supported.

The following code snippets illustrate drawing images of different formats in a PDF document.

[C#]

```
//Draw bitimage image with image size
PdfImage image = new PdfBitmap(pngImage);
g.DrawImage(image, 0, 0, image.Width, image.Height);

//Draw metafile
PdfMetafile metafile = new PdfMetafile(emfImage);
g.DrawImage(metafile, new PointF(0, 50));

//Image pagination jpg
image = new PdfBitmap(jpgImage);

//Set layout format for paginate the image to multiple pages.
PdfLayoutFormat format = new PdfLayoutFormat();
format.Layout = PdfLayoutType.Paginate;
PointF location = new PointF(0, 400);
SizeF size = new SizeF(400, -1);
RectangleF rect = new RectangleF(location, size);
image.Draw(page, rect, format);

//Multiframe Tiff image
PdfBitmap tiffImage = new PdfBitmap(multiframeImage);
int frameCount = tiffImage.FrameCount;
for (int i = 0; i < frameCount; i++)
{
    page = section.Pages.Add();
    section.PageSettings.Margins.All = 0;
    g = page.Graphics;
    tiffImage.ActiveFrame = i;
    g.DrawImage(tiffImage, 0, 0, page.GetClientSize().Width,
    page.GetClientSize().Height);
}

//Specify the quality of bitmap image
PdfBitmap image1 = new PdfBitmap(Image.FromFile("image.bmp"));
image1.Quality = 50;
page.Graphics.DrawImage( image1, new PointF( 0, 0 ));

//Specify the quality of Metafile image
// Create metafile image
```

```

PdfMetafile metafile = (PdfMetafile) PdfImage.FromImage(img);

// Specify the quality of the metafile
metafile.Quality = 50;

```

[VB.NET]

```

'Draw bitmap image with image size
Dim image As PdfImage = New PdfBitmap(pngImage)
g.DrawImage(image, 0, 0, image.Width, image.Height)

'Draw metafile
Dim metafile As PdfMetafile = New PdfMetafile(emfImage)
g.DrawImage(metafile, New PointF(0,50))

'Image pagination jpg
Dim image = New PdfBitmap(jpgImage)

'Set layout format for paginate the image to multiple pages.
Dim format As PdfLayoutFormat = New PdfLayoutFormat()
format.Layout = PdfLayoutType.Paginate
Dim location As PointF = New PointF(0, 400)
Dim size As SizeF = New SizeF(400, -1)
Dim rect As RectangleF = New RectangleF(location, size)
image.Draw(page, rect, format)

'Multiframe Tiff image
Dim tiffImage As PdfBitmap = New PdfBitmap(multiframeImage)
Dim frameCount As Integer = tiffImage.FrameCount
Dim i As Integer = 0
Do While i < frameCount
    page = section.Pages.Add()
    section.PageSettings.Margins.All = 0
    g = page.Graphics
    tiffImage.ActiveFrame = i
    g.DrawImage(tiffImage, 0,
    0,page.GetClientSize().Width,page.GetClientSize().Height)
    i += 1
Loop

'Specify the quality of bitmap image
Dim image1 As New PdfBitmap(Image.FromFile("image.bmp"))
image1.Quality = 50
page.Graphics.DrawImage(image1, New PointF(0, 0))

```

```
'Specify the quality of Metafile image
' Create metafile image
Dim metafile As PdfMetafile = DirectCast(PdfImage.FromImage(img),
PdfMetafile)
' Specify the quality of the metafile
metafile.Quality = 50
```



Figure 40: Drawing PNG Image

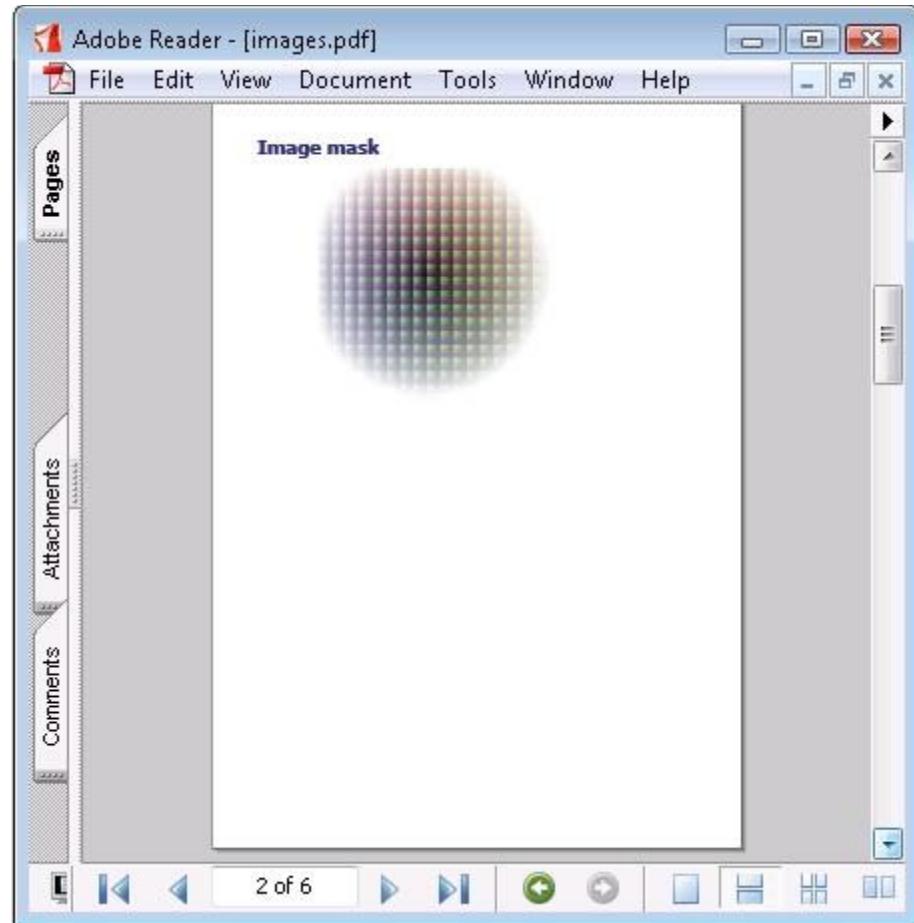


Figure 41: Drawing Image Mask

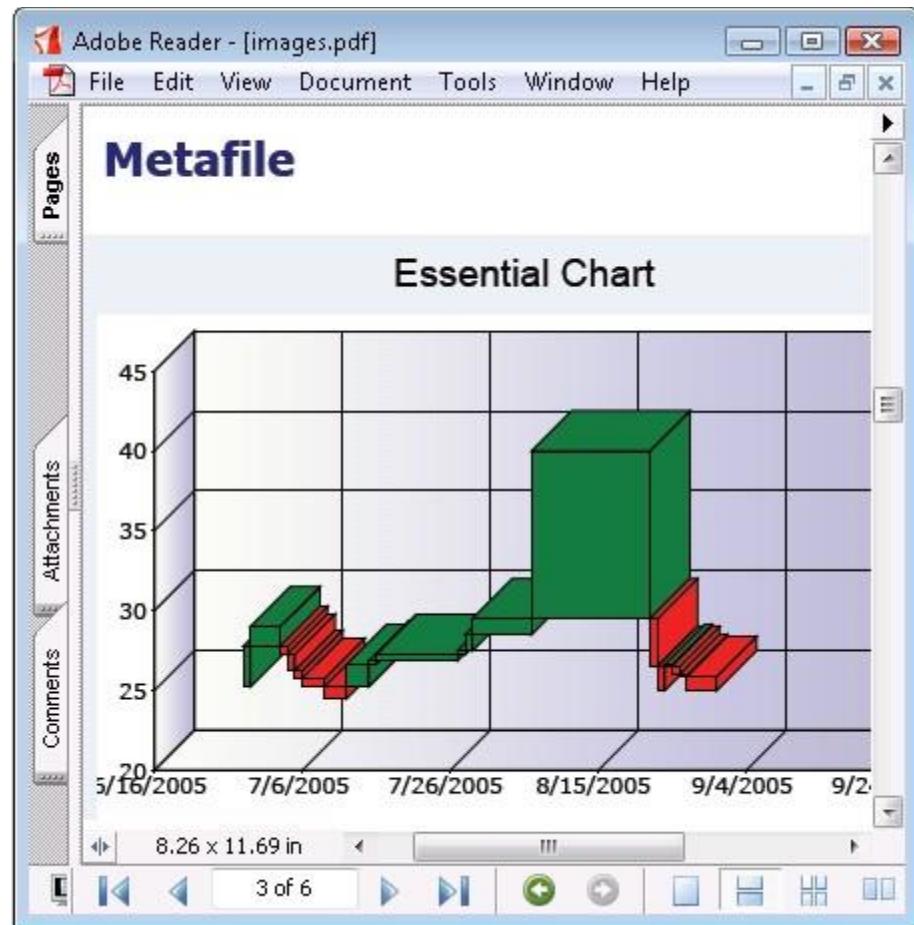


Figure 42: Drawing Metafile Image



Figure 43: Illustrates Image Pagination

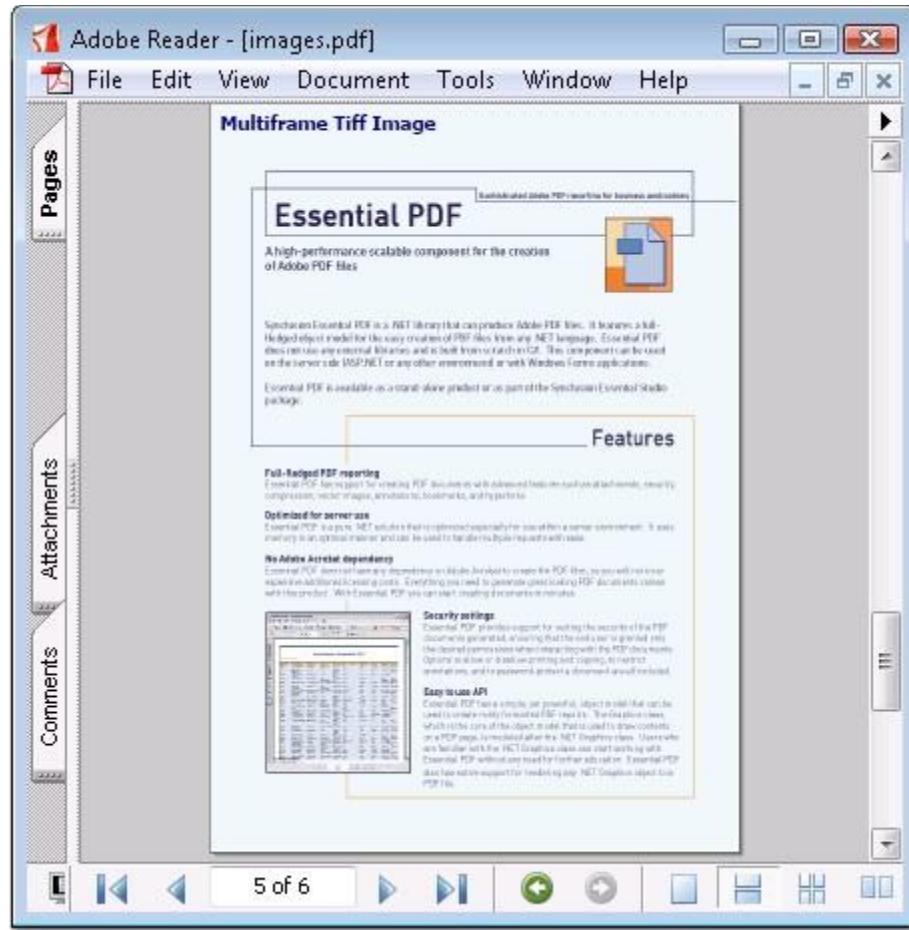


Figure 44: Drawing TIFF Image

4.1.7.4 Drawing Table

Essential PDF provides support for inserting tables into the PDF page. This feature enables the following:

- Drawing tabular data into the PDF page
- Built-in support for importing an ADO.NET DataTable into a table in the PDF page

PdfLightTable class represents simple tables that are used for publishing structured data from arrays, data tables or data columns. There are no real cells or rows in these tables, and all the data is taken from the data source by using the **DataSource** property.



Note: You must add the **Syncfusion.Pdf.Tables** namespace to work with Tables.

The following code example illustrates how to create and format tables.

[C#]

```
//Create Pdf table.  
PdfLightTable table = new PdfLightTable();  
  
//Set Data source.  
table.DataSource = dataTable;  
  
//Set table alternate row style.  
table.Style.AlternateStyle = altStyle;  
  
//Set header row style.  
table.Style.HeaderStyle = headerStyle;  
  
//Show the header row.  
table.Style.ShowHeader = true;  
  
//Repeat header in all the pages.  
table.Style.RepeatHeader = true;  
  
//Set header data from column caption.  
table.Style.HeaderSource = PdfHeaderSource.ColumnCaptions;  
  
//Set layout properties.  
PdfLayoutFormat format = new PdfLayoutFormat();  
format.Break = PdfLayoutBreakType.FitElement;  
format.Layout = PdfLayoutType.Paginate;  
  
//Draw table.  
table.Draw(page, PointF.Empty, format);
```

[VB.NET]

```
'Create Pdf table.  
Dim table As PdfLightTable = New PdfLightTable()  
  
'Set Data source.  
table.DataSource = dataTable  
  
'Set table alternate row style.  
table.Style.AlternateStyle = altStyle
```

```
'Set header row style.  
table.Style.HeaderStyle = headerStyle  
  
'Show the header row.  
table.Style.ShowHeader = True  
  
'Repeat header in all the pages.  
table.Style.RepeatHeader = True  
  
'Set header data from column caption.  
table.Style.HeaderSource = PdfHeaderSource.ColumnCaptions  
  
'Set layout properties.  
Dim format As PdfLayoutFormat = New PdfLayoutFormat()  
format.Break = PdfLayoutBreakType.FitElement  
format.Layout = PdfLayoutType.Paginate  
  
'Draw table.  
table.Draw(page, PointF.Empty, format)
```

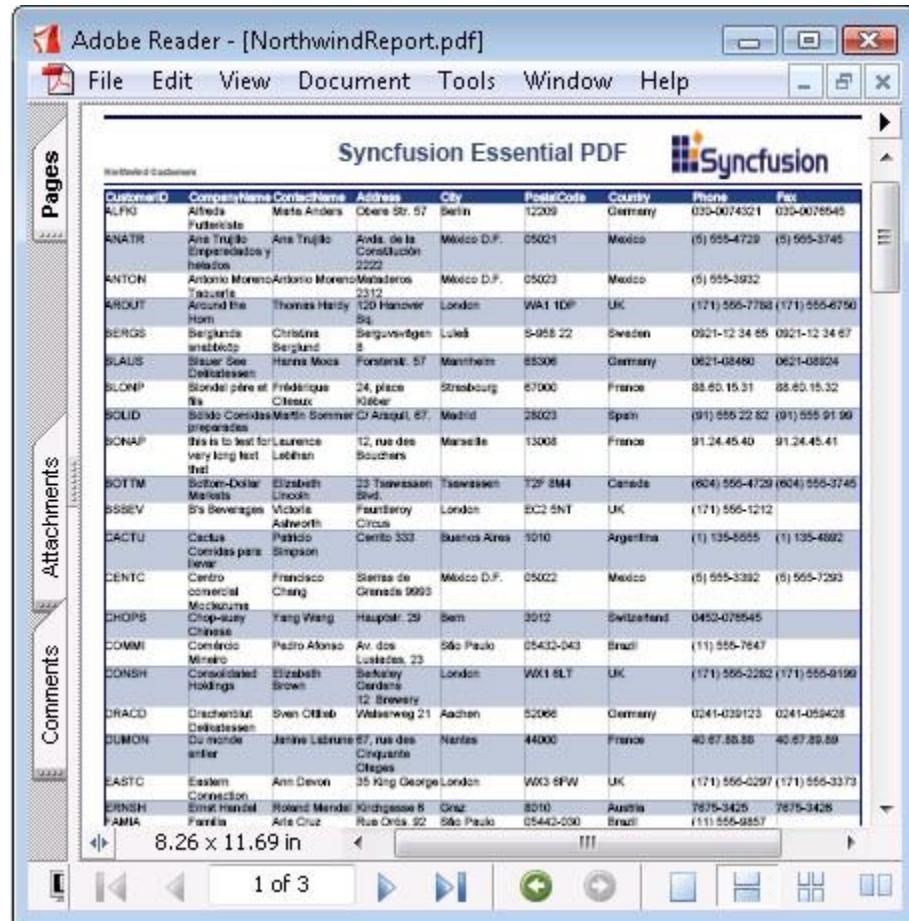


Figure 45: Drawing Table in the PDF page

4.1.7.5 Stream Support

Essential PDF provides support to open or save the created PDF document into a memory stream or file stream. `MemoryStream` and `FileStream` classes can be used for this purpose.

This feature enables the following:

- Open or save the PDF document in the database
- Make changes to your document without saving it to disk

The following code illustrates how to create a memory stream or file stream and save the pdf document into those streams.

[C#]

```
//Create a Memory Stream.
System.IO.MemoryStream memStream = new MemoryStream();

//Save the Stream to memory.
document.Save(memStream);
memStream.Seek(0, SeekOrigin.Begin);

//Create a file Stream
System.IO.FileStream fs = new FileStream("Sample.pdf", FileMode.Create);
memStream.WriteTo(fs);

//Close the Streams.
memStream.Close();
fs.Close();
```

[VB.NET]

```
'Create a Memory Stream.
Dim memStream As MemoryStream = New MemoryStream()

'Save the Stream to memory.
document.Save (memStream, FormatType.Doc)
memStream.Seek (0, SeekOrigin.Begin)

'Create a file Stream
Dim fs As FileStream = New FileStream("Sample.doc", FileMode.Create)
memStream.WriteTo(fs)

'Close the Streams.
memStream.Close()
fs.Close()
```



Note: To open or save the document asynchronously for Windows Store apps, refer to the [Asynchronous Support](#) section.

4.1.7.6 Security Settings

Adobe provides options for securing a PDF document from unauthorized access, and for restricting the access for some user. This section demonstrates various options provided by Essential PDF to secure a PDF document.

The following techniques are used to protect a PDF document:

- Encryption and
- Digital Signature

Encryption

A PDF document is encrypted to protect its contents from unauthorized access. Encryption applies to all strings and streams in the document. Essential PDF provides support for 40, 128 and 256-bit encryption. Essential PDF also provides support for restricted document operations like:

- Allow EditContent
- Allow Copy Content
- Allow Edit Annotations
- Allow AccessibilityCopyContent
- Allow AssembleDocument
- Allow Print
- Allow FullQualityPrint

You can also protect a document with user and owner password.



Note: You must add the `Syncfusion.Pdf.Security` namespace to work with security settings.

Encryption Algorithms

Adobe supports Advanced Encryption Standard (AES) in Adobe version 7.0 and later. Essential PDF also supports strong encryption using 128 and 256-bit AES algorithm. In order to achieve this, specify the type of encryption algorithm in the **Algorithm** property of the Security class.

The following code snippet illustrates this.

[C#]

```
// Set 128 bit AES encryption mode.
doc.Security.KeySize = PdfEncryptionKeySize.Key128Bit;
doc.Security.Algorithm = PdfEncryptionAlgorithm.AES;
```

 **Note:** 40-bit encryption will use RC4 algorithm by default and 256-bit encryption uses AES algorithm.

The following code example illustrates the protection of documents with user and owner password.

[C#]

```
// Set encryption key.
doc.Security.KeySize = PdfEncryptionKeySize.Key128Bit;
doc.Security.Algorithm = PdfEncryptionAlgorithm.AES;

// Setting username and password to protect the document.
doc.Security.OwnerPassword = "syncfusion";
doc.Security.UserPassword = "password";

// Allow print with full quality.
doc.Security.Permissions = PdfPermissionsFlags.Print |
PdfPermissionsFlags.FullQualityPrint;
```

[VB.NET]

```
' Set encryption key.
doc.Security.KeySize = PdfEncryptionKeySize.Key128Bit
doc.Security.Algorithm = PdfEncryptionAlgorithm.AES

' Setting username and password to protect the document.
doc.Security.OwnerPassword = "syncfusion"
doc.Security.UserPassword = "password"

' Allow print with full quality.
doc.Security.Permissions = PdfPermissionsFlags.Print Or
PdfPermissionsFlags.FullQualityPrint
```

The following screen shot shows how to open an encrypted document.



Figure 46: Opening an Encrypted document

The following screen shot shows a security dialog of encrypted documents with restrictions.

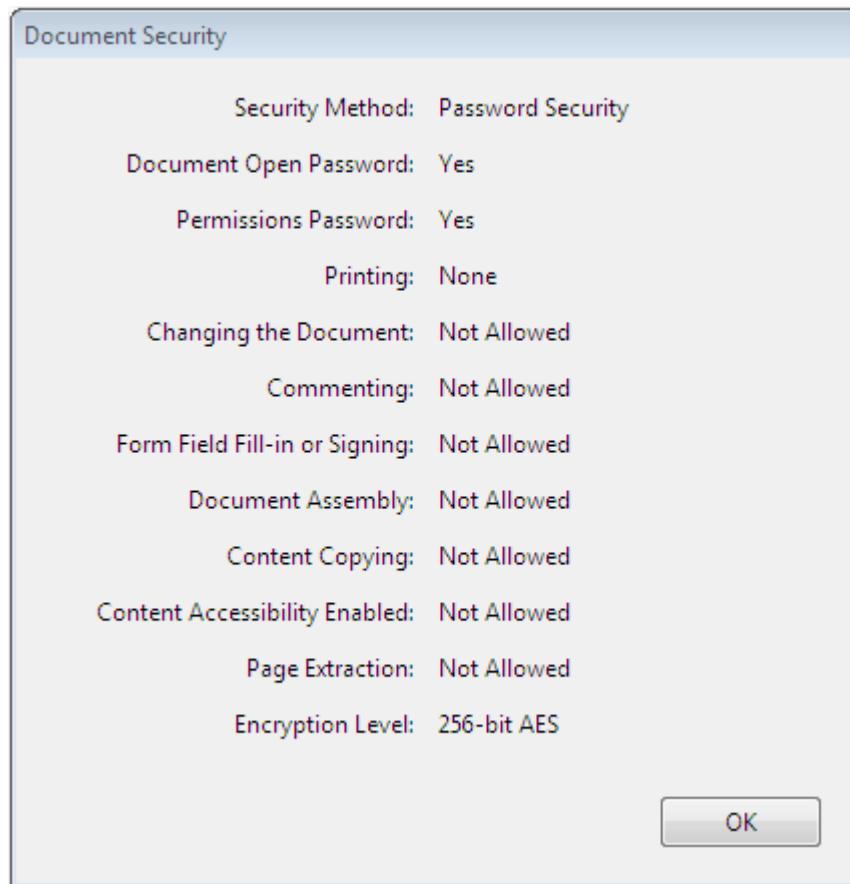


Figure 47: Document Security Dialog box

The following screen shot shows a sample encrypted document.

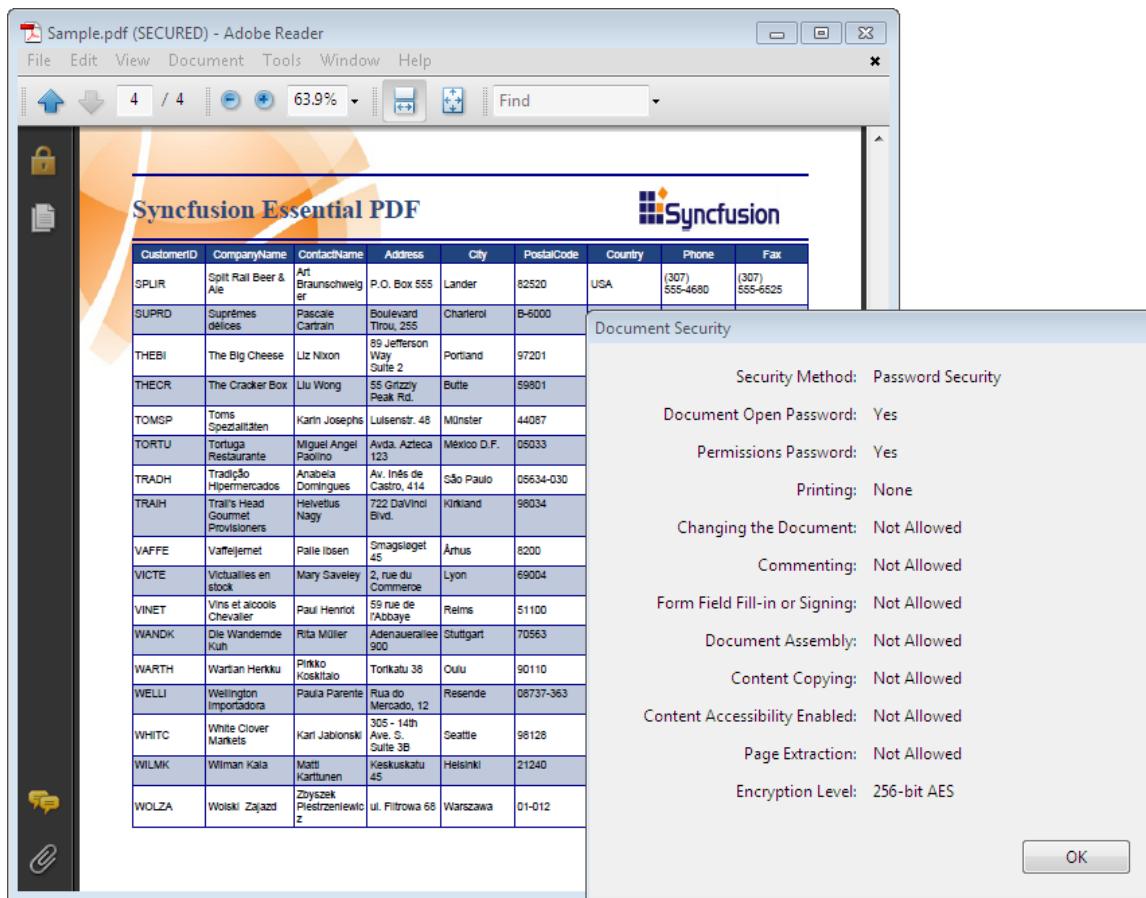


Figure 48: A Simple encrypted document

Digital Signature

In general, digital signatures are used to authenticate the identity of a user and the document's content. It stores information about the signer and the state of the document, i.e. when it was signed.

When enterprises distribute documents electronically, it is often important that recipients can verify:

- Whether the content has not been altered. (integrity)
- Whether the document is coming from the actual person who sent it. (authenticity)
- Whether an individual who has signed the document cannot deny the signature. (non-repudiation)

Digital signatures address these security requirements by providing greater assurances of document integrity, authenticity and non-repudiation.

A PDF document may contain the following standard types of signatures:

One or more document (or ordinary) signatures. These signatures are sometimes referred to as **recipient signatures**. If a signed document is modified and saved by incremental update, the data corresponding to the byte range of the original signature is preserved. If the signature is valid, it is possible to recreate the state of the document as it existed at the time of signing.

At most one Modification Detection and Prevention (MDP) signature. This signature is also referred to as an **author** or **certifying** signature.

The following screenshot shows an author signature prompt.

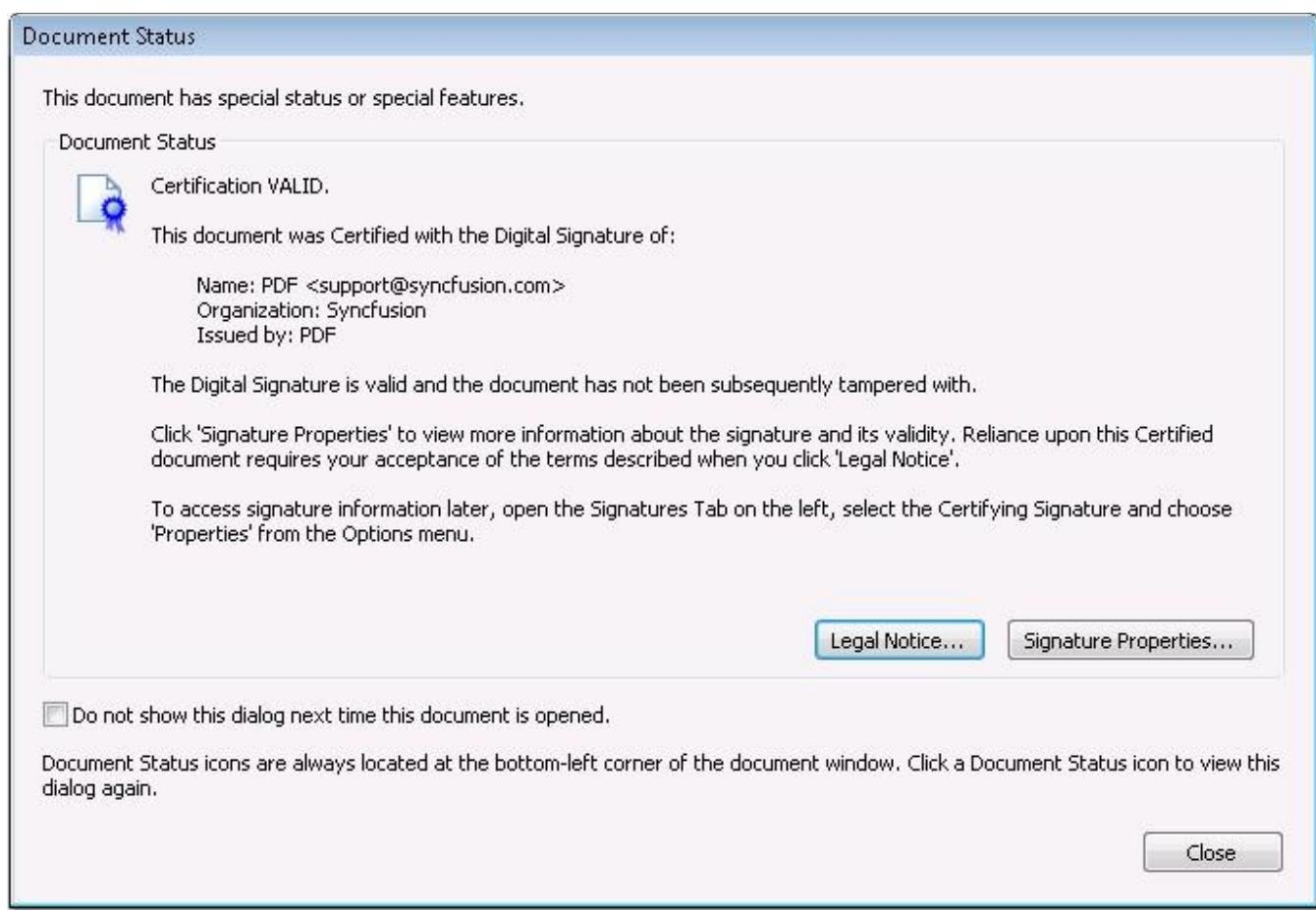


Figure 49: Author Signature Prompt

The following screenshot shows an author signature.

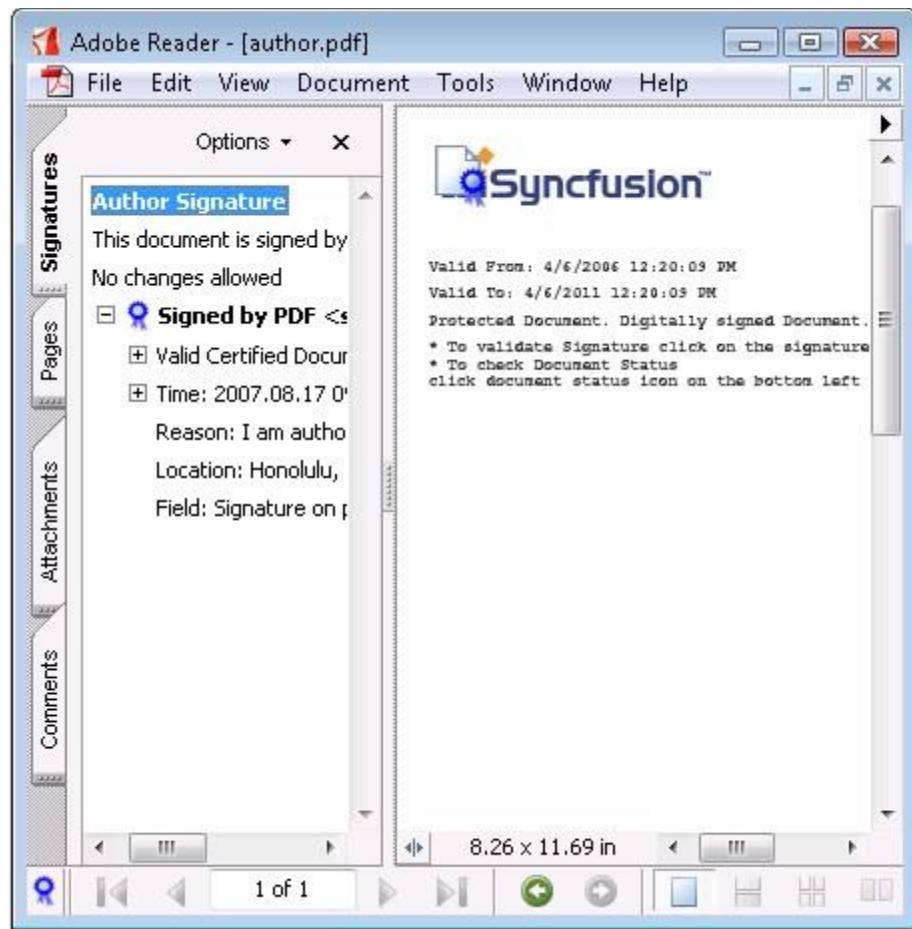


Figure 50: Author Signature

The following screen shot shows a standard signature.

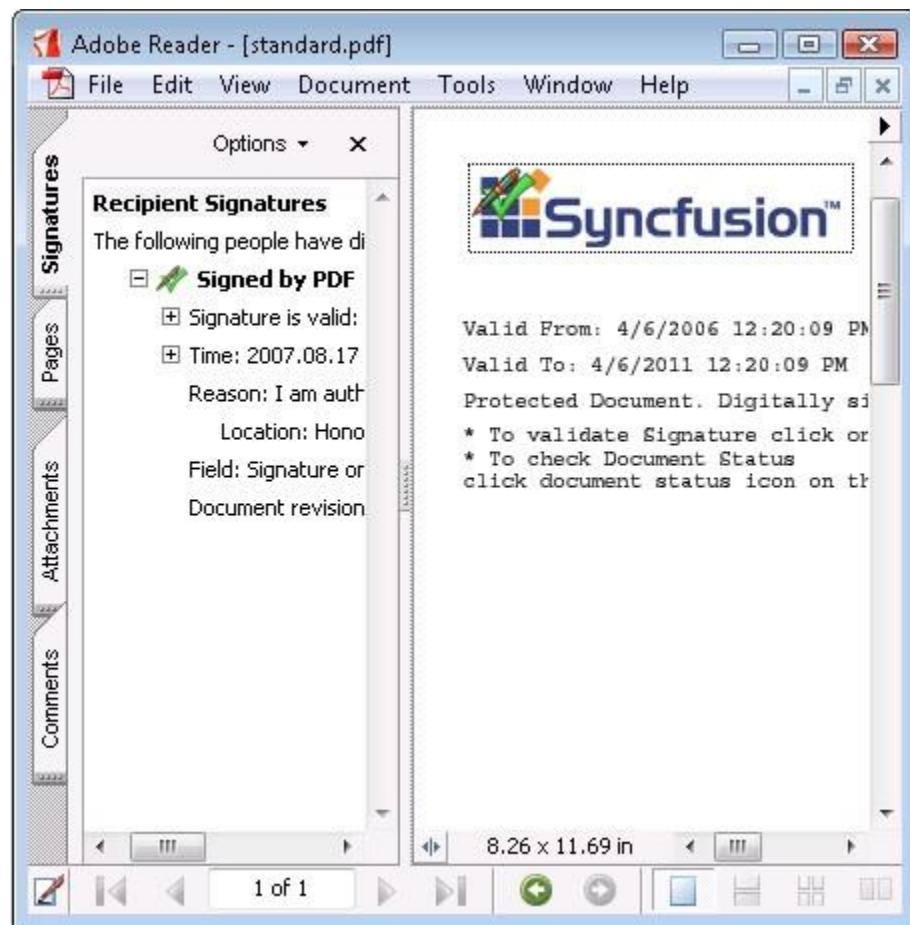


Figure 51: Standard Signature

The following screen shot shows the dialog that appears during verification.



Figure 52: Signature Validation Status

The following code example illustrates the signing of a document with the author's signature.

[C#]

```
//Map the path of the certificate store.

//Get certificate.
PdfCertificate pdfCert = new PdfCertificate(@"..\..\Data\PDF.pfx",
"syncfusion");

//Sign the document in the image.
signature = new PdfSignature(page, pdfCert, "Signature");
bmp = new PdfBitmap(@"..\..\Data\syncfusion_logo.gif");
signature.Bounds = new RectangleF(new PointF(5, 5), bmp.PhysicalDimension);

//Set the Author signature.
signature.Certificated = true;

//Set signature display properties.

// Set signature Info.
signature.ContactInfo = "johndoe@owned.us";
signature.LocationInfo = "Honolulu, Hawaii";
signature.Reason = "I am author of this document.;"
```

[VB]

```
'Map the path of the certificate store.

'Get certificate.
Dim pdfCert As PdfCertificate = New PdfCertificate("../..\Data\PDF.pfx",
"syncfusion")

'Sign the document in the image.
signature = New PdfSignature(page, pdfCert, "Signature")
bmp = New PdfBitmap("../..\Data\syncfusion_logo.gif")
signature.Bounds = New RectangleF(New PointF(5, 5), bmp.PhysicalDimension)

'Set the Author signature.
Private signature.Certificated = True

' Set signature display properties.

'Set signature Info.
signature.ContactInfo = "johndoe@owned.us"
```

```
signature.LocationInfo = "Honolulu, Hawaii"  
signature.Reason = "I am author of this document."
```



Note: Currently only self-created and third-party .pfx certificates are supported.

Timestamp in digital signature

Essential PDF supports addition of timestamp in digital signatures. The date and time at which the document is signed can be added as a part of the signature. Timestamps are easier to verify when they are associated with a timestamp authority's trusted certificate. Including a timestamp, helps to establish exactly when the document is signed and reduces the chances of an invalid signature. The timestamp can be obtained from a third-party timestamp authority or from the certificate authority that issued the digital ID.

Timestamps appear in the signature field and in the Signature Properties dialog box. If the timestamp is included, the certificate will appear in the **Date/Time** tab of the **Signature Properties** dialog box. If no timestamp is added, the signature field displays the local time of the computer at the moment of signing.

The following figure shows the timestamp properties of the digital signature.

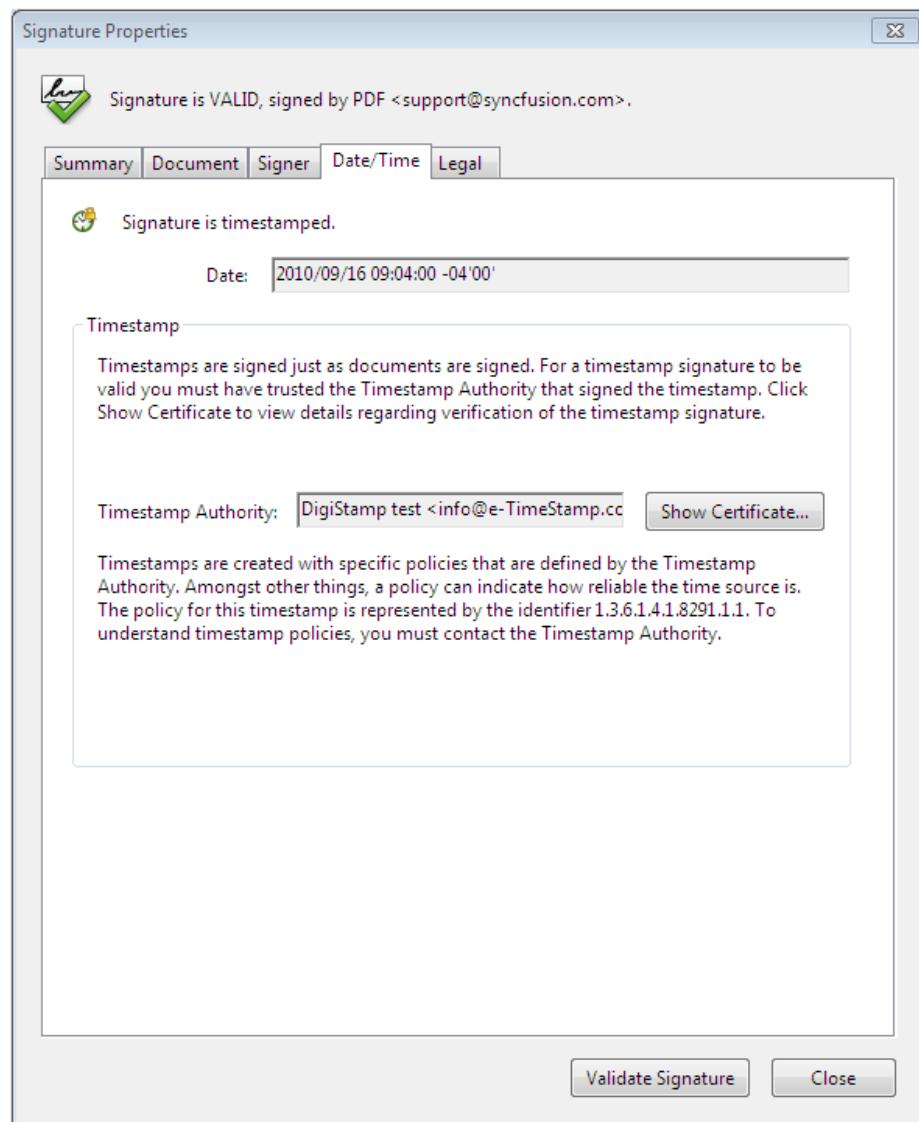


Figure 53: Timestamp properties of the digital signature

To apply timestamp using Essential PDF, the **TimeStampServer** property of the **PdfSignature** class has to be used. The parameters for the TimeStampMethod are the URI of digital server, username and password.

The following code illustrates the method for adding timestamp in the digital signature.

```
[C#]
//Get certificate.
PdfCertificate pdfCert = new PdfCertificate(@"..\..\Data\PDF.pfx",
"syncfusion");
```

```
//Sign the document with timestamp.  
PdfSignature signature = new PdfSignature(page, pdfCert, "Signature");  
  
//Add time stamp using the server URI and credentials.  
signature.TimeStampServer = new TimeStampServer(  
    new Uri("http://digistamp.syncfusion.com"), "user",  
    "123456");
```

[VB]

```
'Get certificate.  
Dim pdfCert As New PdfCertificate("../Data\PDF.pfx", "syncfusion")  
  
'Sign the document with timestamp.  
Dim signature As New PdfSignature(page, pdfCert, "Signature")  
  
'Add time stamp using the server URI and credentials.  
signature.TimeStampServer = New TimeStampServer(New  
Uri("http://digistamp.syncfusion.com"), "user", "123456")
```

The following shows a document signed with the timestamp.

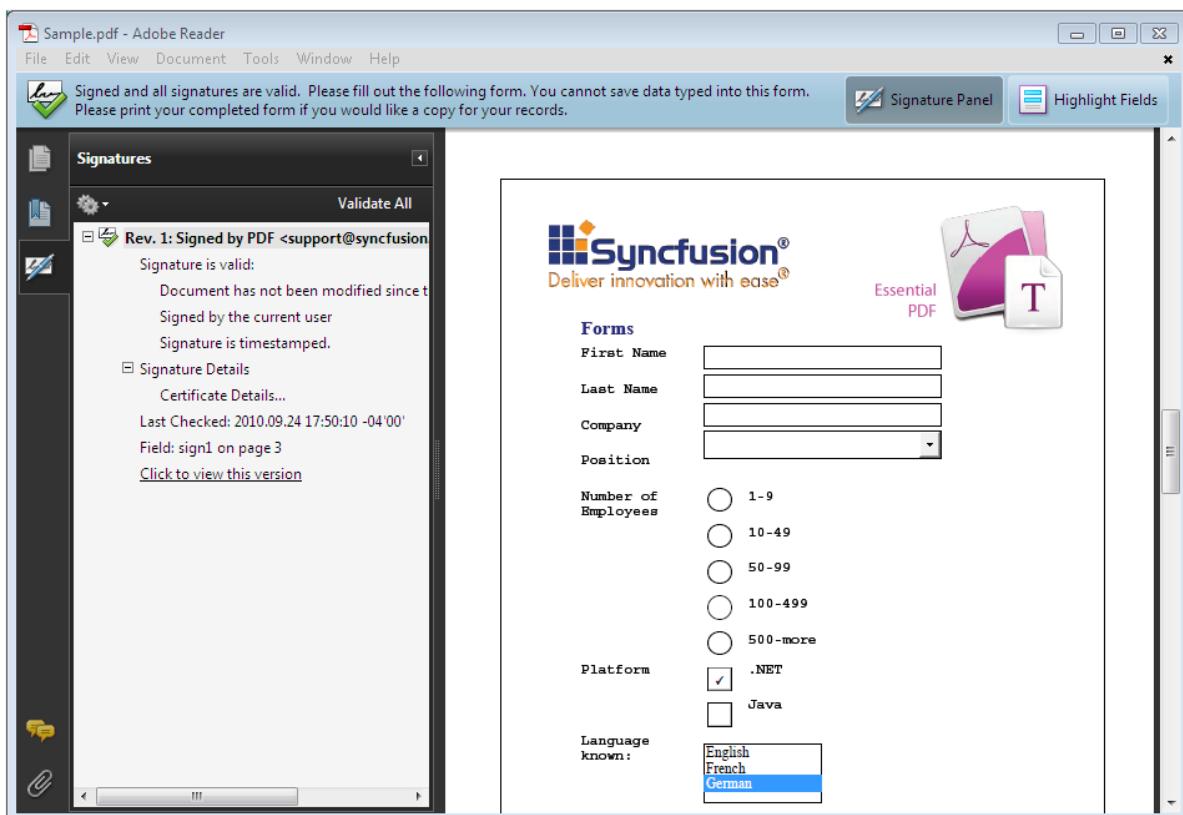


Figure 54: Document signed with timestamp

4.1.7.7 Page Settings

Essential PDF supports various page settings options that control the page display. They are as follows.

- Page orientation
- Page size
- Page layout
- Page mode
- Page scale
- Page transition

Some of the page sizes supported are as follows:

- A0 to A9
- B0 to B5
- ArchA to ArchE

- Half Letter
- Ledger
- Letter
- Legal
- Note
- Letter11x17

Some of the page layouts supported are as follows:

- OneColumn
- SinglePage
- TwoColumnLeft
- TwoColumnRight
- TwoPageRight
- TwoPageLeft

Some of the page modes supported are as follows:

- Full screen
- UseAttachment.UseNone
- UseOC
- UseOutlines
- UseThumbs

When a print dialog is displayed for a document, the values to be selected for the page scaling option are as follows:

- **None**-Indicates that the print dialog should reflect no page scaling
- **AppDefault**-Indicates that applications should use the current print scaling.



Note: If this entry has an unrecognized value, applications should use the current print scaling. The default value is AppDefault.

The following code snippets illustrate the various page settings.

[C#]

```
// To set landscape page orientation.  
doc.PageSettings.Orientation = PdfPageOrientation.Landscape;
```

```
// To set UseOC page Mode.  
doc.ViewerPreferences.PageMode = PdfPageMode.UseOC;  
  
// Setting pagescale option as None  
doc.ViewerPreferences.PageScaling = PageScalingMode.None;  
  
// To set two column left page layout.  
doc.ViewerPreferences.PageLayout = PdfPageLayout.TwoColumnLeft;  
  
//Sets page transition.  
doc.PageSettings.Transition.PageDuration = 1;  
doc.PageSettings.Transition.Duration = 1;
```

[VB.NET]

```
' To set landscape page orientation.  
doc.PageSettings.Orientation = PdfPageOrientation.Landscape  
  
' To set UseOC page Mode.  
doc.ViewerPreferences.PageMode = PdfPageMode.UseOC  
  
' Setting pagescale option as None  
doc.ViewerPreferences.PageScaling = PageScalingMode.None  
  
' To set two column left page layout.  
doc.ViewerPreferences.PageLayout = PdfPageLayout.TwoColumnLeft  
  
'Sets page transition.  
doc.PageSettings.Transition.PageDuration = 1  
doc.PageSettings.Transition.Duration = 1
```

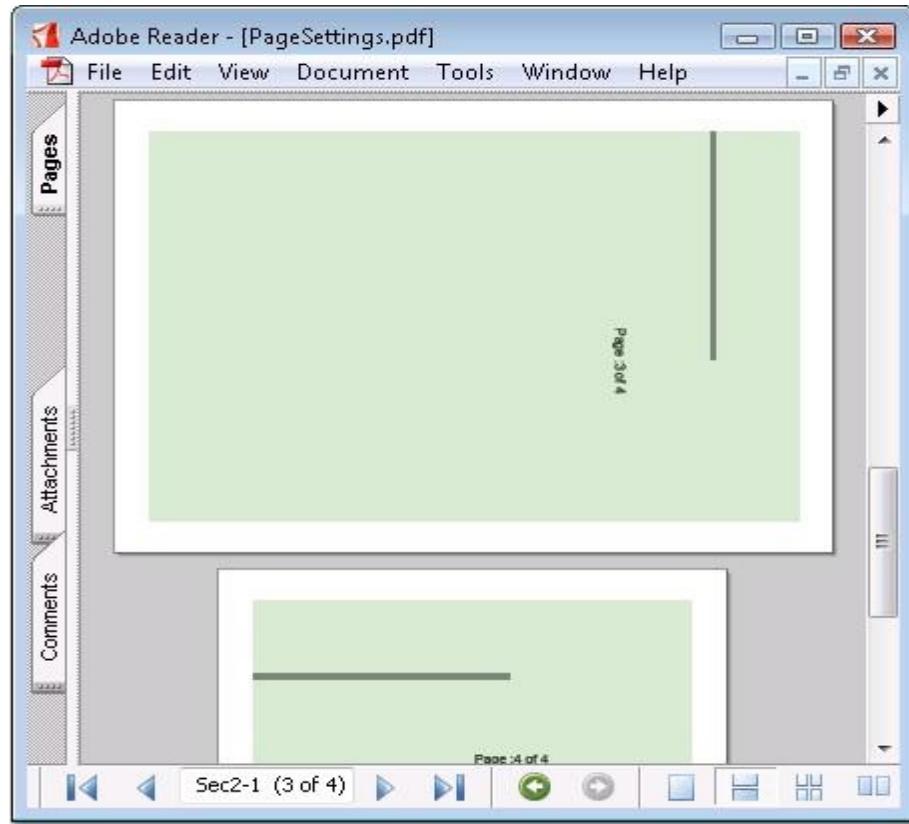


Figure 55: Page Settings

Viewer Preference Settings

Essential PDF supports Viewer Preference options for the pdf pages such as hiding toolbar, hiding menubar and hiding window UI. The **HideToolbar**, **HideMenuBar** and **HideWindowUI** properties can be used for enabling these features. The following code example illustrates this.

[C#]

```
//To hide the viewer application's Tool bar  
doc.ViewerPreferences.HideToolbar = true;  
  
//To hide the viewer application's Menu bar  
doc.ViewerPreferences.HideMenubar = true;  
  
//To hide the user interface elements such as Scroll bar, navigation controls  
doc.ViewerPreferences.HideWindowUI = true
```

[VB.NET]

```
'To hide the viewer application's Tool bar
doc.ViewerPreferences.HideToolbar = True

'To hide the viewer application's Menu bar
doc.ViewerPreferences.HideMenubar = True

'To hide the user interface elements such as Scroll bar, navigation controls
doc.ViewerPreferences.HideWindowUI = True
```

Sample Location

A sample which demonstrates the PDF Page Settings is available in the following sample installation location:

<Install Location>\Windows\Pdf.Windows\Samples\2.0\Settings\Page Settings

4.1.7.8 Headers and Footers

Headers and footers can be placed in the pages of your PDF document.

Follow the below procedure to place a header:

1. Create a template object for the header. **PdfPageTemplateElement** class can be used for creating a template object.
2. Assign the created template header to PDF document header.

The same procedure can be followed to create a footer. Page numbers on the footer of a document are set by using automatic fields.

You can dock the header or footer to any position.

The following code example illustrates how to create a Header and Footer.

[C#]

```
//Create a header and draw the image.
RectangleF rect = new RectangleF(0, 0, doc.Pages[0].GetClientSize().Width,
50);
```

```

PdfPageTemplateElement header = new PdfPageTemplateElement(rect);
PdfImage img = new PdfBitmap(@"..\..\Data\logo.png");

//Draw the image in the Header.
header.Graphics.DrawImage(img, imageLocation, imageSize);

//Add the header at the top
doc.Template.Top = header;

// Footer.
// Create a Template that can be used as a footer.
//Create a page template
PdfPageTemplateElement footer = new PdfPageTemplateElement(rect);

//Create page number field
PdfPageNumberField pageNumber = new PdfPageNumberField(font, brush);

//Create page count field
PdfPageCountField count = new PdfPageCountField(font, brush);

//Add the fields in composite fields
PdfCompositeField compositeField = new PdfCompositeField(font, brush, "Page
{0} of {1}", pageNumber, count);
compositeField.Bounds = footer.Bounds;

//Draw the composite field in footer
compositeField.Draw(footer.Graphics, new PointF(470, 40));

//Add the footer template at the bottom
doc.Template.Bottom = footer;

```

[VB.NET]

```

'Create a header and draw the image.
Dim rect As RectangleF = New RectangleF(0, 0,
doc.Pages(0).GetClientSize().Width, 50)
Dim header As PdfPageTemplateElement = New PdfPageTemplateElement(rect)
Dim img As PdfImage = New PdfBitmap("../..\Data\logo.png")

'Draw the image in the Header.
header.Graphics.DrawImage(img, imageLocation, imageSize)

'Add the header at the top
doc.Template.Top = header

```

```
' Footer.  
' Create a Template that can be used as a footer.  
'Create a page template  
Dim footer As PdfPageTemplateElement = New PdfPageTemplateElement(rect)  
  
'Create page number field  
Dim pageNumber As PdfPageNumberField = New PdfPageNumberField(font, brush)  
  
'Create page count field  
Dim count As PdfPageCountField = New PdfPageCountField(font, brush)  
  
'Add the fields in composite fields  
Dim compositeField As PdfCompositeField = New PdfCompositeField(font, brush,  
"Page {0} of {1}", pageNumber, count)  
compositeField.Bounds = footer.Bounds  
  
'Draw the composite field in footer  
compositeField.Draw(footer.Graphics, New PointF(470, 40))  
  
'Add the footer template at the bottom  
doc.Template.Bottom = footer
```

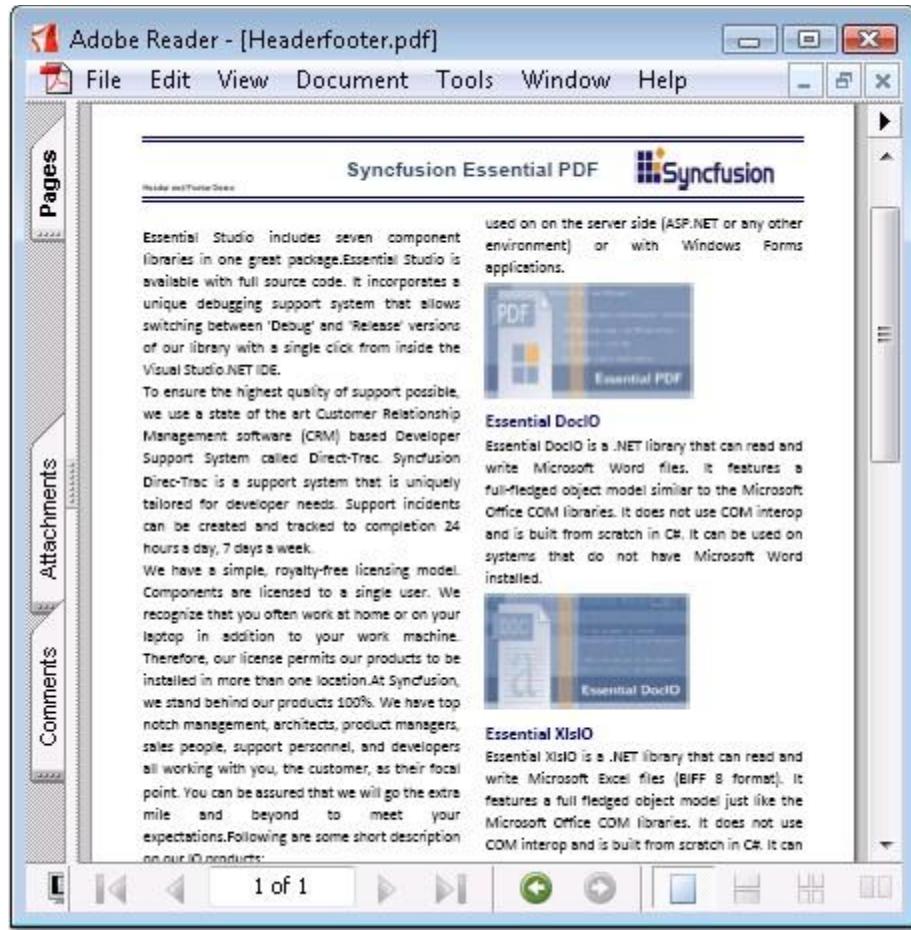


Figure 56: Header and Footer in PDF page

4.2 PDF Editing

The previous [PDF Generator](#) section elaborated on the creation of pdf document and also the various features of the document.

This section will describe the various PDF editing capabilities of the created document under the following topics:

4.2.1 Load Document

An existing PDF document can be loaded and customized. To open an existing PDF document for further manipulations, use the **PdfLoadedDocument** class. Its constructor allows you to specify the file name, stream or byte array as the source of the document data and the password for the encrypted documents.

[C#]

```
PdfLoadedDocument ldDoc = new PdfLoadedDocument(filename);
PdfLoadedDocument ldDoc = new PdfLoadedDocument(memoryStream);
PdfLoadedDocument ldDoc = new PdfLoadedDocument(byte);
PdfLoadedDocument ldDoc = new PdfLoadedDocument(filename,password);
PdfLoadedDocument ldDoc = new PdfLoadedDocument(memoryStream,password);
PdfLoadedDocument ldDoc = new PdfLoadedDocument(byte,password);
```

[VB .NET]

```
Dim ldDoc As PdfLoadedDocument = New PdfLoadedDocument(filename)
Dim ldDoc As PdfLoadedDocument = New PdfLoadedDocument(memoryStream)
Dim ldDoc As PdfLoadedDocument = New PdfLoadedDocument(byte)
Dim ldDoc As PdfLoadedDocument = New PdfLoadedDocument(filename,password)
Dim ldDoc As PdfLoadedDocument = New
PdfLoadedDocument(memoryStream,password)
Dim ldDoc As PdfLoadedDocument = New PdfLoadedDocument(byte,password)
```

Note: You must add the **Syncfusion.Pdf.Parsing** namespace to work with the loaded documents.

In the loaded document, you can do the following:

- Access the bookmarks
- Add a new page
- Add an Attachment
- Load dynamic fields

Go the above individual links for detailed information.

Public Members

The following table lists the public members of the **PdfLoadedDocument** class.

Methods

Name	Description
AddFields	Adds the fields connected to the page.
Append	Appends the specified loaded document to this one.
Clone	Creates a shallow copy of the current document.
Close	Releases the document stream.
CreateBookmarkRoot	Creates the bookmark root.
CreateForm	Creates a new form.
Dispose	Performs application-defined tasks associated with freeing, releasing, or resetting unmanaged resources.
DisposeOnClose	Adds an object to a collection of the objects that will be disposed during document closing. (Inherited from PdfDocumentBase.)
ImportPage	Overloaded.
ImportPageRange	Imports a page range from a loaded document. (Inherited from PdfDocumentBase.)
OnDocumentSaved	Raises DocumentSaved event. (Inherited from PdfDocumentBase.)
Save	Saves the document.
Split	Splits a PDF file to many PDF files; each of them consists of one page from the source file.

Properties

Name	Description
Bookmarks	Gets the bookmarks.
Compression	Gets or sets the desired level of stream compression.
Conformance	Gets the conformance level applied in the PDF.
DocumentInformation	Gets or sets document's information and properties.
FileStructure	Gets or sets the internal structure of the PDF file.

Form	Gets the loaded form.
Pages	Gets the pages.
Security	Gets the security parameters of the document.
ViewerPreferences	Gets or sets a viewer preferences object, controlling the way the document is to be presented on the screen, or in print.

Cloning a document

You can clone a PDF document in order to copy the document without saving it. This can be done by using the **PdfLoadedDocument.Clone** method.

[C#]

```
PdfLoadedDocument ldoc = new PdfLoadedDocument("sample.pdf");
PdfLoadedDocument doc = lDoc.Clone() as PdfLoadedDocument;
```

[VB .NET]

```
Dim ldoc As New PdfLoadedDocument(sample.pdf)
Dim doc As PdfLoadedDocument = TryCast(lDoc.Clone(), PdfLoadedDocument)
```



Note: To open and save the document asynchronously for Windows Store apps, refer to the [Asynchronous Support](#) section.

4.2.1.1 Bookmark

While loading an existing document, the library loads all bookmarks of the document. Each loaded bookmark is represented by the **PdfLoadedBookmark** class, inherited from the **PdfBookmark** class. You can access the root collection of document bookmarks by using the **Bookmark** property of the **PdfLoadedDocument** class. This collection is represented by the **PdfBookmarkBase** class.

The following code example illustrates how to access a loaded bookmark.

[C#]

```
const string filename = "...";
PdfLoadedDocument ldDoc = new PdfLoadedDocument(filename);

PdfBookmarkBase rootCollection = ldDoc.Bookmarks;

PdfLoadedBookmark bookmark = rootCollection[0] as PdfLoadedBookmark;

ldDoc.Save(newFileName);
ldDoc.Close();
```

[VB .NET]

```
const string filename = "...";
PdfLoadedDocument ldDoc = new PdfLoadedDocument(filename);

PdfBookmarkBase rootCollection = ldDoc.Bookmarks;

PdfLoadedBookmark bookmark = rootCollection(0) as PdfLoadedBookmark;

ldDoc.Save( newFileName );
ldDoc.Close();
```

Bookmark Manipulation

The following manipulations can be made to the bookmarks:

- Modifying bookmarks
- Adding Actions to the bookmark

1. Modifying bookmarks

Bookmarks can be modified in the following ways:

- Change the bookmark style, color, title and destination
- Add or insert new bookmarks into the root collection
- Add or insert new bookmarks as a child of another bookmark
- Assign the destination of the added bookmarks to a loaded page or a new page of the document

The following code example illustrates how to modify the bookmark style and destination.

[C#]

```

const string filename = "...";
PdfLoadedDocument ldDoc = new PdfLoadedDocument(filename);
bookmarPdfLoadedPage ldPage = loadedDoc.Pages[1] as PdfLoadedPage;

PdfBookmarkBase rootCollection = ldDoc.Bookmarks;

PdfLoadedBookmark bookmark = rootCollection[0] as PdfLoadedBookmark;
bookmark.Destination = new PdfDestination(ldPage);
bookmark.Color = Color.Green;
bookmark.TextStyle = PdfTextStyle.Bold;
bookmark.Title = "Changed title";

ldDoc.Save(newFileName);
ldDoc.Close();

```

[VB .NET]

```

Const filename As String = "..."
Dim ldDoc As New PdfLoadedDocument(filename)
Dim ldPage As bookmarPdfLoadedPage = TryCast(loadedDoc.Pages(1),
PdfLoadedPage)

Dim rootCollection As PdfBookmarkBase = ldDoc.Bookmarks

Dim bookmark As PdfLoadedBookmark = TryCast(rootCollection(0),
PdfLoadedBookmark)
bookmark.Destination = New PdfDestination(ldPage)
bookmark.Color = Color.Green
bookmark.TextStyle = PdfTextStyle.Bold
bookmark.Title = "Changed title"

Const newFileName As String = "..."
ldDoc.Save(newFileName)
ldDoc.Close()

```

2. Adding Actions to Bookmark

You can perform actions by clicking the bookmarks at run time. To add custom actions to the bookmarks use the following classes.

Class Name	Description
PdfLaunchAction	Launches an application, opens or prints a document.

PdfUriAction	Acts as a unique resource identifier.
PdfJavaScriptAction	Performs a javascript action in the PDF document.
PdfDestination	Represents an anchor in the document where bookmarks and annotations can direct when clicked.
PdfGoToAction	This action goes to a destination in the current document.

[C#]

```
//Create new Bookmark
PdfBookmark bookmarkaction = doc.Bookmarks.Add("Annotations");

PdfLaunchAction action = new PdfLaunchAction(@"..\..\Data\Book.txt");

//launch file when we click the Bookmark
bookmarkaction.Action = action;

PdfBookmark bookmarkaction1 = doc.Bookmarks.Add("Uriaction");

//Create uri action
PdfUriAction uriAction = new PdfUriAction("http://www.google.com");

//Set uri action. Clicking the bookmark will move to the corresponding uri
bookmarkaction1.Action = uriAction;

PdfBookmark bookmarkaction2 = doc.Bookmarks.Add("Scriptaction");

//Create Java action
PdfJavaScriptAction javaAction = new PdfJavaScriptAction("app.alert(\"You
are looking at Java script action of PDF \")");
bookmarkaction2.Action = javaAction;

PdfBookmark bookmarkaction3 = doc.Bookmarks.Add("Pagelocation");
PdfDestination dest = new PdfDestination(doc.Pages[1], new Point(0, 100));
PdfGoToAction goToAction = new PdfGoToAction(doc.Pages[1]);
goToAction.Destination = dest;

// Will move to a particular location of this page
bookmarkaction3.Action = goToAction;
```

[VB .NET]

```
'Create new Bookmark
Dim bookmarkaction As PdfBookmark = doc.Bookmarks.Add("Annotations")

Dim action As New PdfLaunchAction("../..\\Data\\Book.txt")

'launch file when we click the Bookmark
bookmarkaction.Action = action

Dim bookmarkaction1 As PdfBookmark = doc.Bookmarks.Add("Uriaction")

'Create url action
Dim uriAction As New PdfUriAction("http://www.google.com")

'Set uri action. Clicking the bookmark will move to the corresponding uri
bookmarkaction1.Action = uriAction

Dim bookmarkaction2 As PdfBookmark = doc.Bookmarks.Add("Scriptaction")

'Create Java action
Dim javaAction As New PdfJavaScriptAction("app.alert(\"You are looking at
Java script action of PDF \")")
bookmarkaction2.Action = javaAction

Dim bookmarkaction3 As PdfBookmark = doc.Bookmarks.Add("Pagelocation")
Dim dest As New PdfDestination(doc.Pages(1), New Point(0, 100))
Dim goToAction As New PdfGoToAction(doc.Pages(1))
goToAction.Destination = dest

' Will move to this page particular location
bookmarkaction3.Action = goToAction
```



Figure 57: JavaScript alert message displayed using bookmark action

4.2.1.2 Adding a New Page

A new page can be created in the existing pdf document or an existing page can be removed from the pdf document. This section discusses the following:

1. Creating a page
2. Removing a page

Creating a page

To add a new page to a PDF document that was created by an anonymous user, do the following:

1. Pass the path of that particular document to the **PdfLoadedDocument** constructor

2. Call the **doc.Pages.Add** method.

This will create an empty page with the default parameters.

 You can also specify the size and margins for the new page.

The following code snippet illustrates how to create a new page in the pdf document.

[C#]

```
PdfLoadedDocument lDoc = new PdfLoadedDocument(filename);
page = lDoc.Pages.Add() as PdfPage;

g = page.Graphics;
text = "Page 2";
g.DrawString(text, font, PdfBrushes.Black, PointF.Empty);

filename = OutputPath + "AddNewPages.pdf";
lDoc.Save(filename);
lDoc.Close();
```

[VB .NET]

```
Dim lDoc As New PdfLoadedDocument(filename)
page = TryCast(lDoc.Pages.Add(), PdfPage)

g = page.Graphics
text = "Page 2"
g.DrawString(text, font, PdfBrushes.Black, PointF.Empty)

filename = OutputPath + "AddNewPages.pdf"
lDoc.Save(filename)
lDoc.Close()
```



Note: You can use the page's **Graphics**, but should not use the graphics objects that require the page to layout.

A new page is added to the pdf document.

Removing a page

You can also remove pages from the existing PDF document by using the following methods of the **PdfLoadedPageCollection** class.

- Remove
- RemoveAt

The following code snippet illustrates how to remove an existing page from the pdf document.

[C#]

```
PdfLoadedDocument doc = new PdfLoadedDocument("Sample.pdf");

// Removes the page by passing the PDF page
doc.Pages.Remove(doc.Pages[1]);

// Removes the page by specifying the page index
doc.Pages.RemoveAt(2);
```

[VB .NET]

```
Dim doc As PdfLoadedDocument = New PdfLoadedDocument("Sample.pdf")

' Removes the page by passing the PDF page
doc.Pages.Remove(doc.Pages(1))

' Removes the page by specifying the page index
doc.Pages.RemoveAt(2)
```

4.2.1.3 Adding an Attachment

An attachment can be easily added to a PDF document using the **PdfAttachment** class. The following code example illustrates this.

[C#]

```
PdfLoadedDocument doc1 = new PdfLoadedDocument(@"..\..\Data\Sample.pdf");

if(doc1.Attachments == null)
doc1.CreateAttachment();

PdfAttachment attachment = new PdfAttachment(@"..\..\Data\Manual.txt");
attachment.ModificationDate = DateTime.Now;
attachment.Description = @"..\..\Data\Manual.txt";
attachment.MimeType = "application/txt";

doc1.Attachments.Add(attachment);
```

[VB .NET]

```
Dim doc1 As New PdfLoadedDocument("..\\..\\Data\\Sample.pdf")

If doc1.Attachments Is Nothing Then
    doc1.CreateAttachment()
End If

Dim attachment As New PdfAttachment("..\\..\\Data\\Manual.txt")
attachment.ModificationDate = DateTime.Now
attachment.Description = "..\\..\\Data\\Manual.txt"
attachment.MimeType = "application/txt"

doc1.Attachments.Add(attachment)
```

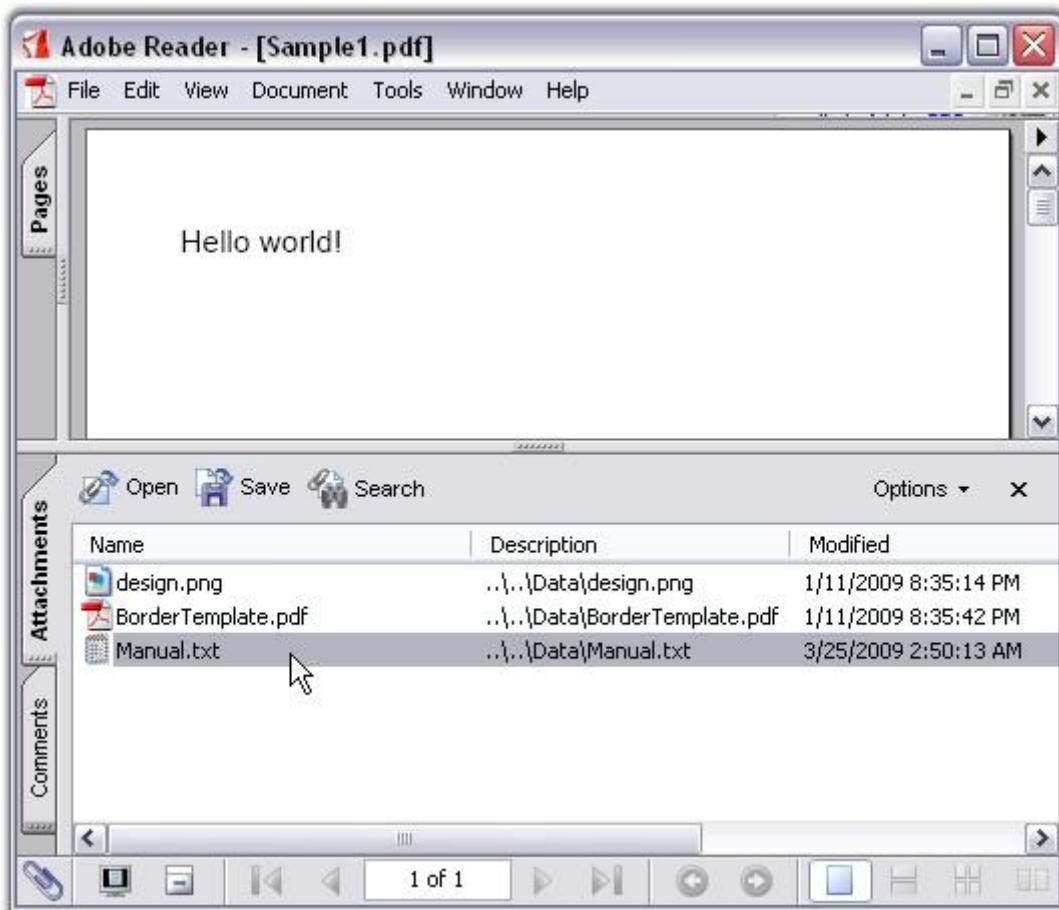


Figure 58: "Manual.txt" document attached to the PDF Document

Manual.txt file is attached to the pdf document.

4.2.1.4 Dynamic Fields

Automatic Fields or Dynamic Fields are special objects that display information calculated automatically, just before the document is saved. \

The fields display the following:

- Page number
- Count of pages
- Author of the document
- Creation and current date

It also displays other information of the document, which is not always evaluated at the moment of constructing the document.

To display the correct value of the field, you should specify the following important properties.

- **Font**-Font used to display the value of the field. Exception is thrown if this property is not set.
- **Brush**-Brush used to print the value of the field. Exception is thrown if this property is not set.
- **Bounds**-Specifies the bounds of the field.

You can also use the **Location** and **Size** properties to define the bounds of the field.

- **Location**-Location of the field. Default point is (0, 0).
- **Size**-Size of the field. If it is not set, the size of the field is automatically calculated to display the value.



Note: It is not necessary to set the **Bounds** and **Size** with **Location** at the same time. **Size** and **Bounds.Size** have the same values.

Numeric fields have an additional **NumberingStyle** property. There are five possible numbering styles supported by the automatic fields:

- **Arabic (1, 2, 3, 4, ...)**
- **Upper Roman (I, II, III, IV, ...)**
- **Roman (i, ii, iii, iv, ...)**
- **Upper Latin (A, B, C, D, ..., Z, AA, AB, ...)**
- **Latin (a, b, c, d, ..., z, aa, ab, ...)**

A brief description on various numbering fields is given below:

- **PdfPageNumberField**-Number of the page on which the field has been drawn
- **PdfPageCountField**-Total number of pages in the document
- **PdfSectionPageNumberField**-Number of pages within a section
- **PdfSectionPageCountField**-Number of sections in a document
- **PdfSectionNumberField**-Number of sections within a document
- **PdfCreationDateField**-Creating date of the document; the value is taken from the **DocumentInformation.CreationDate** property
- **PdfDateTimeField**-Current date and time
- **PdfDestinationPageNumberField**-Number of the specified destination page

- **PdfCompositeField**-Value of the field is composed of any number of other automatic fields

PdfCreationDateField and PdfDateTimeField have the **DateFormatString** property, which defines the formatting string for the value of the field. This property uses the same formatting rules and specifiers as DateTime type of .NET. For detailed information on formatting specifiers, see [http://msdn2.microsoft.com/en-us/library/73ctwf33\(VS.80\).aspx](http://msdn2.microsoft.com/en-us/library/73ctwf33(VS.80).aspx).

You can draw the Automatic Fields on the **PdfTemplate** and set them as the document template or manually draw them on the necessary pages. The values of the fields will be automatically populated on each copy of the template.

The following code example illustrates how to insert dynamic fields such as page number, count, datetime, composite fields, and so on, into the existing document.

[C#]

```

PdfLoadedDocument doc = new PdfLoadedDocument(@"../../Sample.pdf");
PdfFont font = new PdfStandardFont(PdfFontFamily.Helvetica, 12);

//Create page number field
PdfPageNumberField pageNumber = new PdfPageNumberField(font,
PdfBrushes.Black);

//Create page count field
PdfPageCountField count = new PdfPageCountField(font, PdfBrushes.Black);
PdfDateTimeField datetimelfield = new PdfDateTimeField(font,
PdfBrushes.Black);
PdfCompositeField compositeField = new PdfCompositeField(font,
PdfBrushes.Black, "Page {0} of {1}{2}", pageNumber, count, datetimelfield);
compositeField.Bounds = new RectangleF(0,0,250,100);
compositeField.Draw(doc.Pages[0].Graphics, new PointF(0, 0));

PdfCreationDateField datefield = new PdfCreationDateField(font,
PdfBrushes.Black);

```

[VB .NET]

```

Dim doc As New PdfLoadedDocument("../../Sample.pdf")
Dim font As PdfFont = New PdfStandardFont(PdfFontFamily.Helvetica, 12)

'Create page number field
Dim pageNumber As New PdfPageNumberField(font, PdfBrushes.Black)

```

```
'Create page count field
Dim count As New PdfPageCountField(font, PdfBrushes.Black)
Dim datetimefield As New PdfDateTimeField(font, PdfBrushes.Black)
Dim compositeField As New PdfCompositeField(font, PdfBrushes.Black, "Page
{0} of {1}{2}", pageNumber, count, datetimefield)
compositeField.Bounds = New RectangleF(0, 0, 250, 100)
compositeField.Draw(doc.Pages(0).Graphics, New PointF(0, 0))

Dim datefield As New PdfCreationDateField(font, PdfBrushes.Black)
```

When an automatic field is used as a component of the composite field, it is not necessary to specify its Font, Brush and Bounds properties. Just call its constructor without parameters.



Note: You must specify the preceding properties for the composite field.

The following code example illustrates how to use automatic fields in templates.

[C#]

```
PdfLoadedDocument doc = new PdfLoadedDocument(@"../../Sample.pdf");

PdfFont font = new PdfStandardFont(PdfFontFamily.Helvetica, 12f);
PdfBrush brush = PdfBrushes.Black;

PdfTemplate template = new PdfTemplate(15, 15);

PdfDateTimeField dateField = new PdfDateTimeField(font, brush);
dateField.DateFormatString = "dd/MMMM/yyyy";

dateField.Draw(template.Graphics);

for (int i = 0; i < 50; i++)
{
    PdfPage page = document.Pages.Add();
    page.Graphics.DrawPdfTemplate(template, new Point(50, 50));
}
```

[VB .NET]

```
Dim doc As PdfLoadedDocument = New PdfLoadedDocument()

Dim font As PdfFont = New PdfStandardFont(PdfFontFamily.Helvetica, 12.0F)
```

```

Dim brush As PdfBrush = PdfBrushes.Black

Dim template As PdfTemplate = New PdfTemplate(15, 15)

Dim dateField As PdfDateTimeField = New PdfDateTimeField(font, brush)
Dim dateField.DateFormatString = "dd'/'MMMM'/'yyyy"

dateField.Draw(template.Graphics)

For i As Integer = 0 To 49
    Dim page As PdfPage = document.Pages.Add()
    page.Graphics.DrawPdfTemplate(template, New Point(50, 50))
Next i

```

PDF Page Label

A PdfPageLabel object specifies a new numbering range to be applied to the document sections. The following code example illustrates how to set the numbering range to the PDF document sections.

[C#]

```

for (int k = 0, i1=0; k < ldoc.Section.Count; k++)
{
    PdfPageLabel label = new PdfPageLabel();
    label.StartNumber = 1;
    if (k == 0)
    {
        label.NumberStyle = PdfNumberStyle.Numeric;
    }
    else if (k == 1)
    {
        label.NumberStyle = PdfNumberStyle.LowerLatin;
    }
    else if (k == 2)
    {
        label.NumberStyle = PdfNumberStyle.UpperLatin;
    }

    label.Prefix = i1 + "-";
    ldoc.LoadedPageLabel = label;
    i1++;
}

```

[VB .NET]

```
Dim k As Integer = 0, i1 As Integer = 0
While k < ldoc.Section.Count
Dim label As New PdfPageLabel()
    label.StartNumber = 1
    If k = 0 Then
        label.NumberStyle = PdfNumberStyle.Numeric
    ElseIf k = 1 Then
        label.NumberStyle = PdfNumberStyle.LowerLatin
    ElseIf k = 2 Then
        label.NumberStyle = PdfNumberStyle.UpperLatin
    End If

    label.Prefix = i1 & "-"
    ldoc.LoadedPageLabel = label
    i1 += 1
    k += 1
End While
```

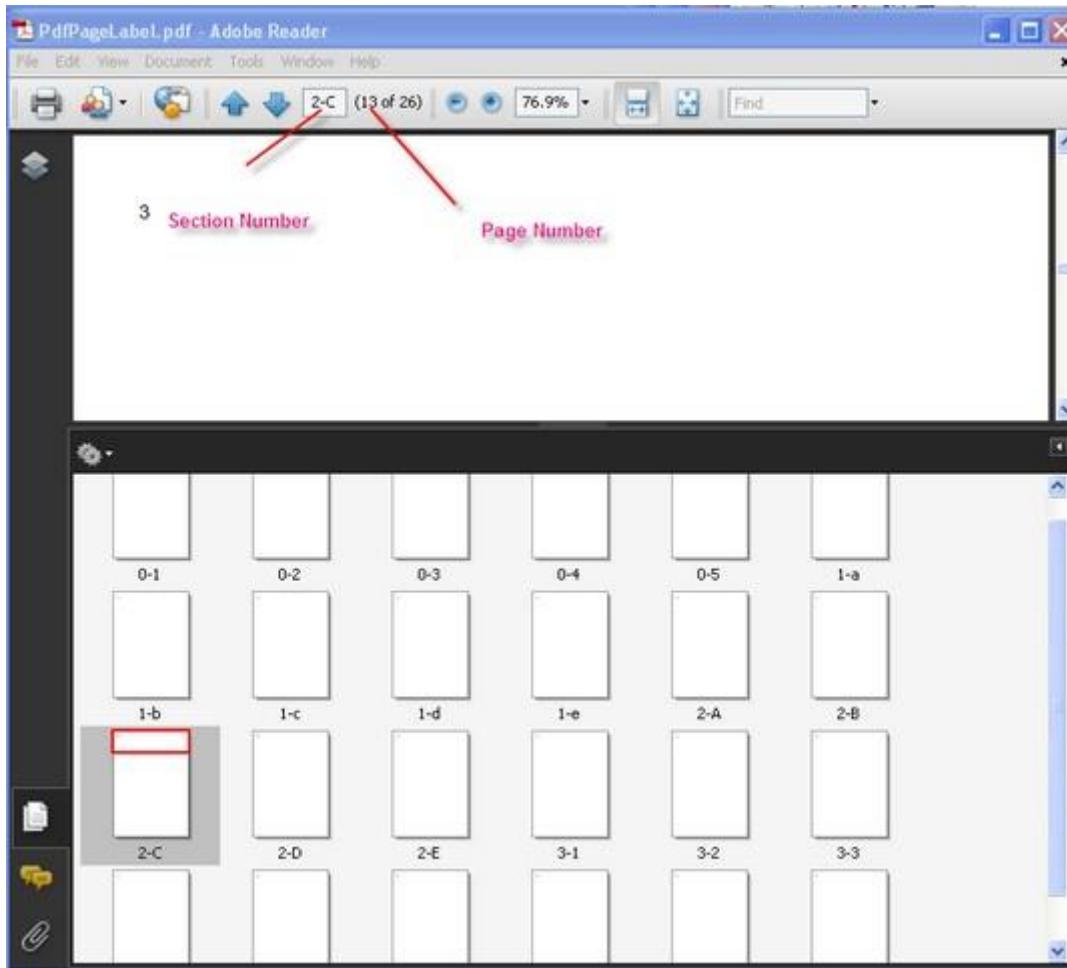


Figure 59: Setting the numbering Range for PDF document

4.2.2 Import Pages

Pages from other document can be imported to the existing document using the **ImportPage** method. The following code example illustrates this method.

[C#]

```
doc.ImportPage(ldDoc, page);
doc.ImportPage(ldDoc, pageIndex);
doc.ImportPageRange(ldDoc, startPageIndex, endPageIndex);
```

[VB .NET]

```
doc.ImportPage(ldDoc, page)
doc.ImportPage(ldDoc, pageIndex)
doc.ImportPageRange(ldDoc, startPageIndex, endPageIndex)
```



Note: The first two methods in the above code are just shortcuts to the last one, which is a powerful tool appending not just pages, but annotations and forms as well.

The parameters included are as follows.

- **IdDoc:** Loaded PDF document
- **Page:** Page that should be appended (not a page itself, but its valid representation)
- **PageIndex:** Index of the page
- **StartPageIndex, endPageIndex:** Indices specifying the page range

The following code snippets illustrate how to import a page to the existing document.

[C#]

```
PdfLoadedDocument ldDoc = new PdfLoadedDocument( filename );
int startIndex = 0;
int endIndex = ldDoc.Pages.Count - 1;
newDoc.ImportPageRange( ldDoc, startIndex, endIndex );
newDoc.Save( newFilename );
newDoc.Close();
ldDoc.Close();
```

[VB .NET]

```
Dim ldDoc As PdfLoadedDocument = New PdfLoadedDocument(filename)
Dim startIndex As Integer = 0
Dim endIndex As Integer = ldDoc.Pages.Count - 1
NewDoc.ImportPageRange(ldDoc, startIndex, endIndex)
NewDoc.Save(NewFilename)
NewDoc.Close()
ldDoc.Close()
```

Implementation Note

The importing is done by converting the page content to the PdfTemplate object, which means that the new page will not inherit the possibly complex layer structure, so you will see just one default layer. Obviously, you will be able to place something beneath that layer. However you will not be able to manipulate the "old" layers, because they won't exist.

This conversion is performed in order to avoid an incomplete page, harming further user output.

Restrictions

- All pages are appended to the host document to make easier bookmarks (outlines) merging. Bookmarks are organized as a complex tree, as it is hard to calculate where the new items should be placed.
- Outlines (Bookmarks) will be copied to the target document along with the pages. The library will try to rebuild the bookmark tree with those bookmarks that have the destination pointing to any of the imported pages.
- The outline tree might look a bit weird if just part of it was copied, as it is hard to recreate the tree with part of the bookmarks.
- Some of the contents are usually imported from the original document to the final document during saving process. Hence, the original document has to be closed only after the final document is saved.



Note: To import the document asynchronously for Windows Store apps, refer to the [Asynchronous Support](#) section.

4.2.3 Replacing Images

Essential PDF supports extracting image locations from an existing document, replace with new images and then save them in the same locations.

This feature is implemented using the following APIs.

Replacing Images

This is done using the ReplaceImage method.

[C#]

```
doc.Pages[0].ReplaceImage(1, new PdfBitmap(@"Water_lilies.jpg"));
```

Extracting Image Location

[C#]

```
foreach (PdfLoadedPage lpage in loadedPages)
{
    PdfImageInfo[] info = lpage.ImagesInfo;
    foreach (PdfImageInfo information in info)
    {
        RectangleF location=information.Bounds.
    }
}
```

The following code snippet illustrates the use case for the above APIs.

[C#]

```
PdfLoadedDocument doc = new PdfLoadedDocument(@"imageDoc.pdf");
PdfBitmap bmp = new PdfBitmap(@"Water lilies.jpg");
doc.Pages[0].ReplaceImage(1, bmp);
doc.Save("Replace Sample.pdf");
System.Diagnostics.Process.Start("Replace Sample.pdf");
```

[C#]

```
// Load an existing PDF
PdfLoadedDocument ldoc = new PdfLoadedDocument(txtUrl.Text);

// Loading Page collections
PdfLoadedPageCollection loadedPages = ldoc.Pages;
int page = 0;

// Extract Image from PDF document pages
foreach (PdfLoadedPage lpage in loadedPages)
{
    PdfImageInfo[] info = lpage.ImagesInfo;

    if (info != null)
        foreach (PdfImageInfo information in info)
            information.Image.Save("Image" + page.ToString() +
information.Bounds.ToString() + ".png",     ImageFormat.Png);
    page++;
Image image = info[0].Image;
image.Save("test.png");
```

```
System.Diagnostics.Process.Start("test.png");
}
```

4.2.4 Merge PDF

Merging feature in the Essential PDF enables appending all documents to the target document. While merging, all bookmarks and attachments will be copied, additionally to **ImportPageRange** behavior.

There are various overloads of the **Merge** method that allow specifying different parameters. Some important parameters are:

- Array of strings
- Array of PdfDocumentBase instances

If the target document is null (in overload that accepts PdfDocumentBase class as a target), a new instance will be created.

You can also merge the documents in the following ways:

- Append all the documents one after the another by using the **Append** method.
- Import the pages from different documents by using the **ImportPageRange** or **ImportPage** method.
- Use **Insert** method to insert the pages one by one.

For more details, see [Merge PDF](#).



Note: To merge or append the document asynchronously for Windows Store apps, refer to the [Asynchronous Support](#) section.

4.2.5 Split PDF

Splitting operation is used to generate a set of PDF documents, each of which is made of one page from the base document. Each new document is saved with a unique name which is generated from the pattern specified.

The pattern should be in .NET format (for example: "myfile{0:000}.pdf") or just a pdf name. In the latter case, the unique name will have the number before ".pdf".

[C#]

```
PdfLoadedDocument ldDoc = new PdfLoadedDocument(defDocumentPath);
const string destFilePattern = OutputPath + "split{0:00}.pdf";
ldDoc.Split(destFilePattern);
```

[VB .NET]

```
Dim ldDoc As PdfLoadedDocument = New PdfLoadedDocument(defDocumentPath)
const String destFilePattern = OutputPath + "split{0:00}.pdf"
ldDoc.Split(destFilePattern)
```



Note: Splitting algorithm uses the Import Page methods. So the result would be similar to it.

Essential PDF also allows to split the pages as per the user's need. The following code example illustrates this.

[C#]

```
// To load an existing document which needs to be split.
PdfLoadedDocument ldDoc = new PdfLoadedDocument(filename);

// Create a pdf document.
PdfDocument doc1 = new PdfDocument();
PdfDocument doc2 = new PdfDocument();

// To add page 9 into pdf document1.
doc1.ImportPage(ldDoc, 9);

// To add page 10 into pdf document1.
doc1.ImportPage(ldoc, 10);

// To add page 5 into pdf document2.
doc2.ImportPage(ldoc, 5);

// To add page 6 into pdf document2.
doc2.ImportPage(ldoc, 6);

// Save pdf document1.
doc1.Save("Document1.pdf");

// Save pdf document1.
doc2.Save("Document2.pdf");
```

[VB.NET]

```
' To load an existing document which needs to be split.
Dim ldDoc As PdfLoadedDocument = New PdfLoadedDocument(filename)

' Create a pdf document.
Dim doc1 As PdfDocument = New PdfDocument()
Dim doc2 As PdfDocument = New PdfDocument()

' To add page 9 into pdf document1.
doc1.ImportPage(ldDoc, 9)

' To add page 10 into pdf document1.
doc1.ImportPage(ldoc,10)

' To add page 5 into pdf document2.
doc2.ImportPage(ldoc, 5 )

' To add page 6 into pdf document2.
doc2.ImportPage(ldoc, 6 )

' Save pdf document1.
doc1.Save("Document1.pdf")

' Save pdf document1.
doc2.Save("Document2.pdf")
```

4.2.6 Transform PDF

The pdf pages can be converted to PdfTemplate object if you want to create a [booklet](#) or just place a few pages onto a single page like an image. The template can be created using the below code.

[C#]

```
PdfTemplate template = lpage.CreateTemplate();
```

[VB.NET]

```
Dim template As PdfTemplate = lpage.CreateTemplate()
```

 **Note:** This template can be scaled, rotated, placed at different coordinates, and so on.

Restrictions

This above process can convert annotations also but with some limitations as follows

- It does not pay attention to the fields
- It takes the first appearance stream from the annotation's normal appearance dictionary, (if it isn't a stream)
- It places the appearance stream as a template on the page according to its states, say, on, off or some other states.
- This may lead to unexpected results.

For more details, see [Import Pages as Templates](#).

4.2.7 Document Information

Essential PDF enables the user to access the following information of the existing document with the help of the **PdfDocumentInformation** class.

- Author
- Creator
- Keywords
- Producer
- Subject
- Title and so on

The following code example illustrates how to access the document information.

[C#]

```
PdfLoadedDocument doc = new PdfLoadedDocument(filename);

// Accessing document information
authorBox.Text = doc.DocumentInformation.Author;
titleBox.Text = doc.DocumentInformation.Title;
subjectBox.Text = doc.DocumentInformation.Subject;
kwBox.Text = doc.DocumentInformation.Keywords;
creatorBox.Text = doc.DocumentInformation.Creator;
prodBox.Text = doc.DocumentInformation.Producer;
```

[VB .NET]

```
Dim doc As PdfLoadedDocument = New PdfLoadedDocument(filename)

' Accessing document information
authorBox.Text = doc.DocumentInformation.Author
titleBox.Text = doc.DocumentInformation.Title
subjectBox.Text = doc.DocumentInformation.Subject
kwBox.Text = doc.DocumentInformation.Keywords
creatorBox.Text = doc.DocumentInformation.Creator
prodBox.Text = doc.DocumentInformation.Producer
```



Note: You can write the document information with the newly created document, but you cannot overwrite the existing meta data information.

4.2.8 Booklet

Booklets are documents with multiple pages arranged on sheets of paper. When folded, the paper will represent the correct page order. Essential PDF provides support for creating booklets, which produces the resulting PDF document that can be printed and stapled in the center to form a booklet.

For example, assume that you have a 13 page document. Creating a booklet of the document will result in a PDF file with 7 pages((page 1, null), (page2, page13), (page3, page12), ...((page7, page8).

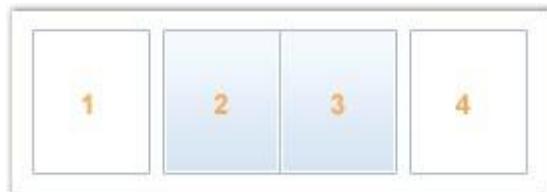


Figure 60: Pages Arranged in PDF

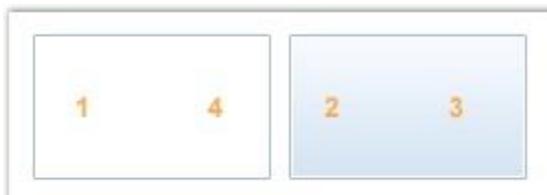


Figure 61: Pages Arranged in Booklet Layout

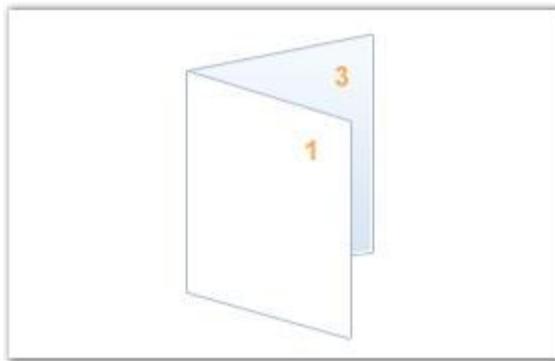


Figure 62: Pages Printed and Folded into New Booklet

PdfBookletCreator class is used for creating Booklets. The following code example illustrates how to create the Booklet.

[C#]

```
// Load a PDF document.
PdfLoadedDocument ldoc = new PdfLoadedDocument("SamplePDF.pdf");

// Create booklet with two sides.
PdfDocument doc = PdfBookletCreator.CreateBooklet(ldoc, new SizeF(500, 500),
true);

// Save the document.
doc.Save("Sample.pdf");
```

[VB .NET]

```
' Load a PDF document.
Dim ldoc As PdfLoadedDocument = New PdfLoadedDocument("SamplePDF.pdf")

' Create booklet with two sides.
Dim doc As PdfDocument = PdfBookletCreator.CreateBooklet(ldoc, New
SizeF(500, 500), True)

' Save the document.
doc.Save("Sample.pdf")
```

The following code example illustrates the overloads of the **CreateBooklet** method.

[C#]

```
CreateBooklet(PdfLoadedDocument, SizeF)
CreateBooklet(PdfLoadedDocument, SizeF, Boolean)
```

```
CreateBooklet(String, String, SizeF)
CreateBooklet(PdfLoadedDocument, SizeF, Boolean, PdfMargins)
CreateBooklet(String, String, SizeF, Boolean)
```

You can also apply margins to the booklets at the time of creating the booklet by using one of the preceding overloads.

The following code example illustrates how to create a booklet with the following overload:
CreateBooklet(PdfLoadedDocument, SizeF, Boolean, PdfMargins).

[C#]

```
// Load a PDF document.
PdfLoadedDocument ldoc = new PdfLoadedDocument("SamplePDF.pdf");

// Specify the margin.
PdfMargins margin = new PdfMargins();
margin.All = 10;

// Create booklet with two sides.
PdfDocument doc = PdfBookletCreator.CreateBooklet(ldoc, new SizeF(500, 500),
true, margin);

// Save the document.
doc.Save("Sample.pdf");
```

[VB.NET]

```
' Load a PDF document.
Dim ldoc As PdfLoadedDocument = New PdfLoadedDocument("SamplePDF.pdf")

' Specify the margin.
Dim margin As PdfMargins = New PdfMargins()
margin.All = 10

' Create booklet with two sides.
Dim doc As PdfDocument = PdfBookletCreator.CreateBooklet(ldoc, New
SizeF(500, 500), True, margin)

' Save the document.
doc.Save("Sample.pdf")
```

4.2.9 OCR Support

The Tesseract optical character recognition (OCR) engine was originally developed by Hewlett-Packard. It was one of the top three engines in the 1995 UNLV accuracy test and is probably one of the most accurate open-source OCR engines available. It has been extensively revised with sponsorship from Google. Essential PDF uses the Tesseract OCR engine to perform OCR on a PDF file. Essential PDF eliminates the 32-bit restriction of Tesseract and allows you to work in either x86-bit or x64-bit platforms without any deployment changes.

Use Case Scenarios

- This will convert an unsearchable PDF document to a searchable PDF document.
- It allows users to search, select, and copy text from images found in the PDF document.

4.2.9.1 Tables for Properties and Methods

Properties

Table 5: Properties Table

Property	Description
Settings	Gets or sets the settings for performing OCR.
Paginate	Gets or sets image pagination in the PDF document.
Regions	Gets or sets page regions for performing OCR.
Language	Gets or sets the OCR language to process.
PageIndex	Gets or sets page index to process OCR.
PageRegions	Gets or sets a rectangular array for the page to process the OCR in specific regions.
Performance	Gets or sets the performance of the OCR.
BlackList	Gets or sets the black-list values.

WhiteList	Gets or sets the white-list values.
-----------	-------------------------------------

Methods

Table 6: Method Table

Method	Description
PerformOCR	Performs OCR on images in the loaded PDF document.

4.2.9.2 Prerequisites

This section covers the mandatory requirements for using OCR support.

4.2.9.2.1 DLLs

The following assemblies need to be referenced in your application to use the Tesseract OCR engine.

Syncfusion DLLs:

- Syncfusion.Core.dll
- Syncfusion.Compression.Base.dll
- Syncfusion.Pdf.Base.dll
- Syncfusion.OcrProcessor.dll

Tesseract DLLs:

- SyncfusionTesseract.dll (Tesseract version 3.0)
- liblept168.dll (Leptonica image processing library used by the Tesseract OCR engine since version 3)

4.2.9.2.2 Windows Deployment

Use the following steps to organize Windows deployment:

1. Place the **SyncfusionTesseract.dll** and **liblept168.dll** assemblies in the **Bin** folder.
2. Refer to the following assemblies to your project:
 - Syncfusion.Core.dll
 - Syncfusion.Compression.Base.dll
 - Syncfusion.Pdf.Base.dll
 - Syncfusion.OCRProcessor.Base.dll

4.2.9.2.3 Web Deployment

Use the following steps to organize web deployment:

1. Place the **SyncfusionTesseract.dll** and **libliblept168.dll** assemblies in the local system and provide an assembly path to the OCR processor.
2. Refer to the following assemblies to your project:
 - Syncfusion.Core.dll
 - Syncfusion.Compression.Base.dll
 - Syncfusion.Pdf.Base.dll
 - Syncfusion.OCRProcessor.Base.dll

4.2.9.3 Performing OCR for a Complete PDF Document

The following code example illustrates how to perform OCR for a complete PDF document.

[C#]

```
//Initialize the OCR processor
using (OCRProcessor processor = new
OCRProcessor(ocrBinariesPath))
{
    //Load a PDF document
    PdfLoadedDocument lDoc = new PdfLoadedDocument(fileName);
    //Set OCR language to process
    processor.Settings.Language = Languages.English;
    //Process OCR by providing the PDF document, data
    dictionary and language
    processor.PerformOCR(lDoc, tessDataPath);
    //Save the OCR processed PDF document in a disk
    lDoc.Save("Sample.pdf"));
    lDoc.Close(true);
}
```

[VB]

```
'Initialize the OCR processor
Using processor As New OCRProcessor(ocrBinariesPath)
'Load a PDF document
Dim lDoc As New PdfLoadedDocument(fileName)
```

```

'Set OCR language to process
processor.Settings.Language = Languages.English

'Process OCR by providing the PDF document, data dictionary, and
language
processor.PerformOCR(lDoc, tessDataPath)

'Save the OCR processed PDF document in a disk
lDoc.Save("Sample.pdf")
lDoc.Close(True)
End Using

```

4.2.9.4 Perform OCR for a Specific Region of PDF Document

The following code example explains how to perform OCR for a specific region of the PDF document.

[C#]

```

//Initialize the OCR processor
using (OCRProcessor processor = new OCRProcessor(ocrBinariesPath))
{
    //Load a PDF document
    PdfLoadedDocument lDoc = new PdfLoadedDocument(fileName);

    //Set OCR language to process
    processor.Settings.Language = Languages.English;
    RectangleF rect = new RectangleF(0, 100, 950, 150);
    //Assign rectangles to the page
    PageRegion region = new PageRegion();
    List<PageRegion> pageRegions = new List<PageRegion>();
    regionPageIndex = 1;
    region.PageRegions = new RectangleF[] { rect };
    pageRegions.Add(region);
    processor.Settings.OCRPageRegion = pageRegions;
    //Process OCR by providing the PDF document, data dictionary, and
language
    processor.PerformOCR(lDoc, tessDataPath);
}

```

```
//Save the OCR processed PDF document in a disk
lDoc.Save("Sample.pdf");
lDoc.Close(true);
}
```

[VB]

```
Using processor As New OCRProcessor(ocrBinariesPath)
'Load a PDF document
Dim lDoc As New PdfLoadedDocument(fileName)
'Set OCR language to process
processor.Settings.Language = Languages.English
Dim rect As New RectangleF(0, 100, 950, 150)
'Assign rectangles to the page
Dim region As New PageRegion()
Dim pageRegions As New List(Of PageRegion)()
regionPageIndex = 1
region.PageRegions = New RectangleF() {rect}
pageRegions.Add(region)
processor.Settings.OCRPageRegion = pageRegions
'Process OCR by providing the PDF document, data dictionary, and
language
processor.PerformOCR(lDoc, tessDataPath)
'Save the OCR processed PDF document in a disk
lDoc.Save("Sample.pdf")
lDoc.Close(True)
End Using
```



Note: The Tesseract binaries, namely **SyncfusionTesseract.dll**, **libliblept168.dll**, and language pack (tessdata), will be available in the following location.

<<Installation Location>>\Syncfusion\Essential Studio\<<Version Number>>\OCRProcessor

4.2.10 Tutorial

This tutorial will show you how easy it is to get started using Essential PDF. It will give you a basic introduction to the concepts you need to know before getting started with the product and some tips and ideas on how to implement PDF into your projects. The lessons in this tutorial are meant to introduce you to PDF with simple step-by-step procedures.

Features

4.2.10.1 Merge PDF

Essential PDF supports the merging of multiple PDF documents. It can merge multiple documents from stream as well as the files stored on the disk.

The following merging techniques are discussed in this section:

- Merging Multiple Documents from Disk
- Merging Multiple Documents from Stream
- Merging Two Files using Append method
- Merging pages of different Documents

Merging Multiple Documents from Disk

The following code example illustrates how to merge multiple documents.

[C#]

```
// Create a string array of source files which are to be merged.
string[] source = { source1, source2 };

// Merge PDFDocument.
PdfDocument.Merge(destination, source);
```

[VB.NET]

```
' Create a string array of source files which are to be merged.
Dim source As String() = { source1, source2 }

' Merge PDFDocument.
PdfDocument.Merge(destination, source)
```

Merge Multiple Documents from Stream

It is also possible to merge multiple PDF documents from stream. The following code example illustrates this.

[C#]

```
Stream[] streams = { stream1, stream2 };
PdfDocumentBase.Merge(doc, streams);
doc.Save("sample.pdf");
```

[VB .NET]

```
Dim streams As Stream() = {stream1, stream2}
PdfDocumentBase.Merge(doc, streams)
doc.Save("sample.pdf")
```

Merging two Files using Append method

You can also merge two files, by appending one file after another. The following code example illustrates this.

[C#]

```
// Append PDFDocument.
doc1.Append(doc2);
```

[VB .NET]

```
' Append PDFDocument.
doc1.Append(doc2)
```

Merging pages of different Documents

Yet another way of merging will be, to import all the pages from one document to another. The following code example illustrates this.

[C#]

```
// Import all the pages to another document
doc2.ImportPageRange(doc2, 0, doc.Pages.Count);
```

[VB .NET]

```
' Import all the pages to another document
doc2.ImportPageRange(doc2, 0, doc.Pages.Count)
```



Note: To merge or append the document asynchronously for Windows Store apps, refer to the [Asynchronous Support](#) section.

4.2.10.2 Import Pages As Templates

You can create a booklet or just place few pages onto a single one by converting the pages into a PdfTemplate object. This template can be scaled, rotated, placed at different coordinates, and so on. It enables you to customize the page representation as per your need.

The following code example illustrates how to import a page as a template.

[C#]

```
PdfPage page = doc1.Pages.Add();
PdfGraphics g = page.Graphics;

PdfPageBase lpage = doc2.Pages[0];
PdfTemplate template = lpage.CreateTemplate();

g.DrawPdfTemplate(template, PointF.Empty, page.GetClientSize());
```

[VB .NET]

```
Dim doc As PdfLoadedDocument = New
PdfLoadedDocument("../Data/sample.pdf")
Dim page As PdfPage = doc1.Pages.Add()
Dim g As PdfGraphics = page.Graphics

Dim lpage As PdfPageBase = doc2.Pages(0)
Dim template As PdfTemplate = lpage.CreateTemplate()

g.DrawPdfTemplate(template, PointF.Empty, page.GetClientSize())
```

4.2.10.3 Signature

You can add signatures to an existing document by using the **PdfSignature** class, and also create a document with **multiple signatures**. You have to create a new instance for the document after saving the document.

The following code example illustrates how to create a document with multiple signatures.

[C#]

```
//Append shows empty fields
PdfDocument doc = new PdfDocument();
PdfPageBase page = doc.Pages.Add();

// Creating Certificate
PdfCertificate pdfCert = new PdfCertificate(@"..\..\test.pfx", "111");

// Adding the signature
PdfSignature signature = new PdfSignature(doc, page, pdfCert, "Signature 1");
PdfBitmap bmp = new PdfBitmap(@"..\..\PDFDemo.jpg");

// Setting the appearance of the signature
signature.Bounds = new RectangleF(new PointF(5, 5), bmp.PhysicalDimension);
signature.ContactInfo = "johndoe@owned.us";
signature.LocationInfo = "Honolulu, Hawaii";
signature.Reason = "I am author of this document.";
string validto = "Valid To: " + signature.Certificate.ValidTo.ToString();
string validfrom = "Valid From: " +
signature.Certificate.ValidFrom.ToString();

PdfSolidBrush brush = new PdfSolidBrush(new PdfColor(1, 1, 255));
PdfPen pen = new PdfPen(brush, 1);
PdfFont font = new PdfStandardFont(PdfFontFamily.Courier, 12,
PdfFontStyle.Regular);
PdfGraphics g = signature.Appearence.Normal.Graphics;
g.DrawImage(bmp, 0, 0);
g.DrawString(validfrom, font, pen, brush, 10, 30);
g.DrawString(validto, font, pen, brush, 10, 10);

// Storing the document
doc.Save("Sample.pdf");
doc.Close(true);

// Load the signed document
PdfLoadedDocument document = new PdfLoadedDocument("Sample.pdf");
page = document.Pages.Add();

// Adding a signature
```

```

PdfSignature signature1 = new PdfSignature(document, page, pdfCert,
"Signature 2");
PdfBitmap bmp1 = new PdfBitmap(@"..\..\Image.jpg");
signature1.Bounds = new RectangleF(new PointF(5, 5), bmp.PhysicalDimension);
signature1.ContactInfo = "Roase@owned.us";
signature1.LocationInfo = "London, UAE";
signature1.Reason = "I am second author of this document.";
validto = "Valid To: " + signature1.Certificate.ValidTo.ToString();
validfrom = "Valid From: " + signature1.Certificate.ValidFrom.ToString();

brush = new PdfSolidBrush(new PdfColor(1, 1, 255));
pen = new PdfPen(brush, 1);
font = new PdfStandardFont(PdfFontFamily.Courier, 12, PdfFontStyle.Regular);
g = signature1.Appearance.Normal.Graphics;
g.DrawImage(bmp1, 100, 400);
g.DrawString(validfrom, font, pen, brush, 10, 30);
g.DrawString(validto, font, pen, brush, 10, 10);

// Store the document
document.Save("Sample1.pdf");

```

[VB]

```

'Append shows empty fields
Dim doc As PdfDocument = New PdfDocument()
Dim page As PdfPageBase = doc.Pages.Add()

' Creating Certificate
Dim pdfCert As PdfCertificate = New PdfCertificate("../..\test.pfx", "111")

' Adding the signature
Dim signature As PdfSignature = New PdfSignature(doc, page, pdfCert,
"Signature 1")
Dim bmp As PdfBitmap = New PdfBitmap("../..\PDFDemo.jpg")

' Setting the appearance of the signature
signature.Bounds = New RectangleF(New PointF(5, 5), bmp.PhysicalDimension)
signature.ContactInfo = "johndoe@owned.us"
signature.LocationInfo = "Honolulu, Hawaii"
signature.Reason = "I am author of this document."
    Dim validto As String = "Valid To: " &
signature.Certificate.ValidTo.ToString()
    Dim validfrom As String = "Valid From: " &
signature.Certificate.ValidFrom.ToString()

    Dim brush As PdfSolidBrush = New PdfSolidBrush(New PdfColor(1, 1,

```

```

255))
    Dim pen As PdfPen = New PdfPen(Brush, 1)
    Dim font As PdfFont = New PdfStandardFont(PdfFontFamily.Courier, 12,
PdfFontStyle.Regular)
    Dim g As PdfGraphics = signature.Appearance.Normal.Graphics
g.DrawImage(bmp, 0, 0)
g.DrawString(validfrom, font, pen, brush, 10, 30)
g.DrawString(validto, font, pen, brush, 10, 10)

' Storing the document
doc.Save("Sample.pdf")
doc.Close(True)

' Load the signed document
Dim document As PdfLoadedDocument = New PdfLoadedDocument("Sample.pdf")
page = document.Pages.Add()

' Adding a signature
Dim signature1 As PdfSignature = New PdfSignature(document, page, pdfCert,
"Signature 2")
Dim bmp1 As PdfBitmap = New PdfBitmap("../..\Image.jpg")
signature1.Bounds = New RectangleF(New PointF(5,5), bmp.PhysicalDimension)
signature1.ContactInfo = "Roase@owned.us"
signature1.LocationInfo = "London, UAE"
signature1.Reason = "I am second author of this document."
validto = "Valid To: " & signature1.Certificate.ValidTo.ToString()
validfrom = "Valid From: " & signature1.Certificate.ValidFrom.ToString()

brush = New PdfSolidBrush(New PdfColor(1, 1, 255))
pen = New PdfPen(brush, 1)
font = New PdfStandardFont(PdfFontFamily.Courier, 12, PdfFontStyle.Regular)
g = signature1.Appearance.Normal.Graphics
g.DrawImage(bmp1, 100,400)
g.DrawString(validfrom, font, pen, brush, 10, 30)
g.DrawString(validto, font, pen, brush, 10, 10)

' Store the document
document.Save("Sample1.pdf")

```

4.3 PDF Form

In the previous sections we have seen the procedure to create a pdf document, its properties and various editing options of the loaded document. In this section, we will see how to create and work with forms by using Essential PDF under the following topics.

4.3.1 Overview

AcroForms are the PDF files that contain form fields. Data can be entered into these fields by the end-user or the author of the form. Internally, AcroForms are annotations or fields applied to a PDF document. While loading an existing document, the AcroForm is also loaded if available.

Creating Form

The loaded form is represented by the **PdfLoadedForm** class. It is accessed by using the **Form** property of the **PdfLoadedDocument** class. If the document does not contain an AcroForm, you can create it by using the **CreateForm** method of the **PdfLoadedDocument** class.

The following code example illustrates this.

[C#]

```
// Loading Existing Document
PdfLoadedDocument loadedDoc = new PdfLoadedDocument(filename);

// Creates a form
loadedDoc.CreateForm();

// Adding a new Page
PdfPageBase page = loadedDoc.Pages.Add();

// Creating a new Field
PdfButtonField bt = new PdfButtonField(page, "Submit");
bt.Bounds = new RectangleF(0, 0, 100, 100);
bt.Text = "Submit";

// Adding the Field
loadedDoc.Form.Fields.Add(bt);
```

[VB]

```
' Loading Existing Document
Dim loadedDoc As PdfLoadedDocument = New PdfLoadedDocument(filename)

' Creates a form
loadedDoc.CreateForm()

' Adding a new Page
```

```

Dim page As PdfPageBase = loadedDoc.Pages.Add()

' Creating a new Field
Dim bt As PdfButtonField = New PdfButtonField(page, "SumBit")
bt.Bounds = New RectangleF(0, 0, 100, 100)
bt.Text = "Submit"

' Adding the Field
loadedDoc.Form.Fields.Add(bt)

```

Accessing Form

PdfLoadedForm class contains the collection of loaded fields represented by the **PdfLoadedFormFieldCollection** class, and inherited from the **PdfFieldCollection** class. The base class for each loaded field is represented by the **PdfLoadedField** class, and inherited from the **PdfField** class.

You can change the form's properties, add new fields or remove the existing fields.

The following code example illustrates how to use a loaded form.

[C#]

```

PdfLoadedDocument ldDoc = new PdfLoadedDocument(filename);

PdfLoadedForm form = ldDoc.Form;
PdfPage page = ldDoc.Pages.Add() as PdfPage;

PdfTextBoxField textField = new PdfTextBoxField(page, "textBox");
textField.Bounds = new RectangleF(0, 0, 100, 100);
textField.Text = "New text field";
form.Fields.Add(textField);

ldDoc.Save(newFileName);
ldDoc.Close();

```

[VB]

```

Dim ldDoc As PdfLoadedDocument = New PdfLoadedDocument(filename)

Dim form As PdfLoadedForm = ldDoc.Form
Dim page As PdfPage = TryCast(ldDoc.Pages.Add(), PdfPage)

```

```

Dim textField As PdfTextBoxField = New PdfTextBoxField(page, "textBox")
textField.Bounds = New RectangleF(0, 0, 100, 100)
textField.Text = "New text field"
form.Fields.Add(textField)

ldDoc.Save(newFileName)
ldDoc.Close()

```

Flattening

The library enables to flatten the loaded field by using the Flatten property of the PdfLoadedField class. A particular field or the whole form can be flattened using this class. The following code snippet illustrates this.

[C#]

```

//For Whole form flattening
PdfLoadedDocument ldDoc = new PdfLoadedDocument(filename);
PdfLoadedForm form = ldDoc.Form;
form.Flatten = true;
ldDoc.Save(newFileName);
ldDoc.Close();

//Flattening the first field
form.Fields[0].Flatten = true;

```

[VB .NET]

```

'For Whole form flattening
Dim ldDoc As New PdfLoadedDocument(filename)
Dim form As PdfLoadedForm = ldDoc.Form
form.Flatten = True
ldDoc.Save(newFileName)
ldDoc.Close()

'Flattening the first field
form.Fields(0).Flatten = True

```

Disable AutoFormat

This property allows you to disable the formatting applied to form fields. Formats can be disabled for either selected fields or for the complete form.

Use Case Scenarios

This property helps when filling fields in different cultures. For example, a currency text field might get a number as an input in different formats, based on the culture in use. If it is preset with a culture that may affect other culture inputs, this property will help to remove it and set the input text with the new formatting.

Adding Disable AutoFormat to an Application

Adding the Disable AutoFormat feature to the application is described in the following code snippets:

[C#]

```
// Disable format for the selected field
PdfLoadedTextBoxField field = loadedDocument.Form.Fields["Price"] as
PdfLoadedTextBoxField;
field.Text = "$1,000.23";
field.DisableAutoFormat = true;

// Disable format for all fields in the form
loadedDocument.Form.DisableAutoFormat = true;
```

[VB]

```
' Disable format for the selected field
Dim field As PdfLoadedTextBoxField =
TryCast(loadedDocument.Form.Fields("Price"), PdfLoadedTextBoxField)
field.Text = "$1,000.23"
field.DisableAutoFormat = True
' Disable format for all fields in the form
loadedDocument.Form.DisableAutoFormat = True
```



Note: XFA Form is currently supported with the following limitations:

- User rights will not be preserved
- Only XFA static forms are supported

Modifying loaded form actions

Essential PDF provides support for modifying the various actions of the loaded forms. The following code example illustrates this.

[C#]

```
//Load the Document.
PdfLoadedDocument ldoc1 = new PdfLoadedDocument("Form_Action.pdf");

//Create a PdfLodedForm.
PdfLoadedForm form = ldoc1.Form;

PdfJavaScriptAction javaAction1 = new PdfJavaScriptAction("app.alert(\"You
are looking at Java script action of PDF (PdfLoadedCheckBoxField)\")");

(form.Fields[".NET"] as PdfLoadedCheckBoxField).LostFocus = javaAction1;
```

[VB .NET]

```
'Load the Document.
Dim ldoc1 As New PdfLoadedDocument("Form_Action.pdf")

'Create a PdfLodedForm.
Dim form As PdfLoadedForm = ldoc1.Form

Dim javaAction1 As New PdfJavaScriptAction("app.alert(\"You are looking at
Java script action of PDF (PdfLoadedCheckBoxField)\")")

TryCast(form.Fields(".NET"), PdfLoadedCheckBoxField).LostFocus = javaAction1
```

4.3.2 Form Fields

An interactive form is a collection of fields for gathering information interactively from the user. A PDF document may contain any number of fields appearing on any combination of pages, all of which makes up a single global interactive form spanning the entire document. Arbitrary subsets of these fields can be imported or exported from the document.

You can change the form's properties, add new fields or remove existing fields. Also you can flatten the loaded field by using the Flatten property.

This section covers the following:

- Form Fields Creation
- Editing Form Fields

Form Fields Creation

The following classes are used to create Form Fields.

Class	Description
PdfButtonField	Creates Button (Submit Button and Reset button can be created).
PdfCheckBoxField	Creates Check Box.
PdfComboBoxField	Creates Combo Box.
PdfListFieldItem	Creates list items from Combo Box and List Box.
PdfListBoxField	Creates List Box.
PdfTextBoxField	Creates Text Box.
PdfRadioButtonListField	Creates Radio button list.
PdfRadioButtonListItem	Creates Radio button list item.

Button field

A button field represents an interactive control on the screen that the user can manipulate using the mouse. PdfButtonField class is used to create Buttons fields.

The most important feature of PDF form is the ability to send entered data to a server. To perform this action you should create an action of SubmitAction type, and specify a valid data processing script URL. The action must be assigned to a MouseUp action of the submit button.

 **1** ResetAction is used to restore the default values of the fields or simply clear them.

The following code example illustrates how to create a button field.

```
[C#]

// Creating a Button
PdfButtonField button = new PdfButtonField(page, "Click");
button.Bounds = new RectangleF(0, 420, 90, 20);
button.Text = "Click";
loadedDoc.Form.Fields.Add(button);

// Creating Submit action button
PdfSubmitAction submitAction = new
PdfSubmitAction("http://stevex.net/dump.php");
submitAction.DataFormat = SubmitDataFormat.Html;
```

```
//Create submit button to transfer the values in the form
PdfButtonField submitButton = new PdfButtonField(page, "submitButton");
submitButton.Bounds = new RectangleF(100, 420, 90, 20);
submitButton.Font = font;
submitButton.Text = "Submit";

// Assigning the submit action
submitButton.Actions.MouseUp = submitAction;

// Adding the Field
loadedDoc.Form.Fields.Add(submitButton);
```

[VB.NET]

```
' Creating a Button
Dim button As PdfButtonField = New PdfButtonField(page, "Click")
button.Bounds = New RectangleF(0, 420, 90, 20)
button.Text = "Click"
loadedDoc.Form.Fields.Add(button)

' Creating Submit action button
Dim submitAction As PdfSubmitAction = New
PdfSubmitAction("http://stevex.net/dump.php")
submitAction.DataFormat = SubmitDataFormat.Html

'Create submit button to transfer the values in the form
Dim submitButton As PdfButtonField = New PdfButtonField(page,
"submitButton")
submitButton.Bounds = New RectangleF(100, 420, 90, 20)
submitButton.Font = font
submitButton.Text = "Submit"

' Assigning the submit action
submitButton.Actions.MouseUp = submitAction

' Adding the Field
loadedDoc.Form.Fields.Add(submitButton)
```

Essential PDF enables to add a Print Button to the form. Clicking the print button, initializes a print dialog. The following code example illustrates this.

[C#]

```
// Create a print button
```

```
PdfButtonField print = new PdfButtonField(page, "print");
print.Bounds = new RectangleF(200, 25, 90, 15);
print.Text = "Print";
print.AddPrintAction();
```

[VB.NET]

```
'Create a print button
Dim print As PdfButtonField = New PdfButtonField(page, "print")
print.Bounds = New RectangleF(200, 25, 90, 15)
print.Text = "Print"
print.AddPrintAction()
```

Check Box Field

A check box field represents one or more check boxes that toggle between two states, ON and OFF. The check box state is manipulated by the user using the mouse or keyboard.

PdfCheckBoxField class is used to create a check box in PDF forms. You can customize the check box style by using properties such as **BorderStyle**, **HighlightMode**, **BorderWidth** and so on.

[C#]

```
//Create a check box
PdfCheckBoxField checkBox = new PdfCheckBoxField(page, ".NET");

//Set properties
checkBox.Bounds = new RectangleF(100, 290, 20, 20);
checkBox.HighlightMode = PdfHighlightMode.Push;
checkBox.BorderStyle = PdfBorderStyle.Beveled;

//Set the value for the check box
checkBox.Checked = true;
g.DrawString(".NET", font, brush, new RectangleF(150, 290, 180, 20));
loadedDoc.Form.Fields.Add(checkBox);
```

[VB.NET]

```
'Create a check box
Dim checkBox As PdfCheckBoxField = New PdfCheckBoxField(page, ".NET")

'Set properties
checkBox.Bounds = New RectangleF(100, 290, 20, 20)
```

```

checkBox.HighlightMode = PdfHighlightMode.Push
checkBox.BorderStyle = PdfBorderStyle.Beveled

'Set the value for the check box
checkBox.Checked = True
g.DrawString(".NET", font, brush, New RectangleF(150,290,180,20))
loadedDoc.Form.Fields.Add(checkBox)

```

PdfComboBox Field

A combo box represents a drop-down list, optionally accompanied by an editable text box in which the user can type a value other than the predefined choices. PdfComboBoxField class is used to create a combo box field in PDF forms. You can add list of items to the combo box by using the PdfListFieldItem class.

The following code example illustrates this.

[C#]

```

//Create a combo box
PdfComboBoxField positionComboBox = new PdfComboBoxField(page,
"positionComboBox");

//Set properties
positionComboBox.Bounds = new RectangleF(100, 115, 200, 20);
positionComboBox.Font = font;

// Setting the combobox as editable
positionComboBox.Editable = true;

//Add combobox to document
loadedDoc.Form.Fields.Add(positionComboBox);

//Create the field item to be added in the combobox
PdfListFieldItem item1 = new PdfListFieldItem("Developer", "Developer");
PdfListFieldItem item2 = new PdfListFieldItem("Accountant", "Accountant");

//Add the items in combo box.
positionComboBox.Items.Add(item1);
positionComboBox.Items.Add(item2);

```

[VB .NET]

```

'Create a combo box
Dim positionComboBox As PdfComboBoxField = New PdfComboBoxField(page,

```

```

"positionComboBox")

' Set properties
positionComboBox.Bounds = New RectangleF(100, 115, 200, 20)
positionComboBox.Font = font

' Setting the combobox as editable
positionComboBox.Editable = True

' Add combobox to document
loadedDoc.Form.Fields.Add(positionComboBox)

' Create the field item to be added in the combobox
Dim item1 As PdfListFieldItem = New PdfListFieldItem("Developer",
"Developer")
Dim item2 As PdfListFieldItem = New PdfListFieldItem("Accountant",
"Accountant")

' Add the items in combobox.
positionComboBox.Items.Add(item1)
positionComboBox.Items.Add(item2)

```

PdfListBoxField

A scrollable List Box contains several text items, one or more of which may be selected as the field value. **PdfListBoxField** is used to create the ListBox field in PDF forms.

[C#]

```

//Create list box
PdfListBoxField listBox = new PdfListBoxField(page, "list1");

//Set the properties.
listBox.Bounds = new RectangleF(100, 350, 100, 50);
listBox.HighlightMode = PdfHighlightMode.Outline;

//Add the items to the list box
listBox.Items.Add(new PdfListFieldItem("English", "English"));
listBox.Items.Add(new PdfListFieldItem("French", "French"));
listBox.Items.Add(new PdfListFieldItem("German", "German"));

//Select the item
listBox.SelectedIndex = 2;

//Set the multiselect option
listBox.MultiSelect = true;

```

```
loadedDoc.Form.Fields.Add(listBox);
```

[VB .NET]

```
'Create list box
Dim listBox As PdfListBoxField = New PdfListBoxField(page, "list1")

'Set the properties.
listBox.Bounds = New RectangleF(100, 350, 100, 50)
listBox.HighlightMode = PdfHighlightMode.Outline

'Add the items to the list box
listBox.Items.Add(New PdfListFieldItem("English", "English"))
listBox.Items.Add(New PdfListFieldItem("French", "French"))
listBox.Items.Add(New PdfListFieldItem("German", "German"))

'Select the item
listBox.SelectedIndex = 2

'Set the multiselect option
listBox.MultiSelect = True
loadedDoc.Form.Fields.Add(listBox)
```



Note: You can create multiple options from the ListBox by setting the MultiSelect option to True.

TextBox field

A text field is a box or space in which the user can enter text through the keyboard. The text can be restricted to a single line or permitted to span multiple lines, depending on the setting of the Multiline flag. **PdfTextBoxField** class is used to create a textbox field in PDF forms. This class also provides support to create password and multilined text boxes.

The following code example illustrates this.

[C#]

```
//Create a text box
PdfTextBoxField firstNameTextBox = new PdfTextBoxField(page,
"firstNameTextBox");

//Set properties
firstNameTextBox.Bounds = new RectangleF(100, 20, 200, 20);
firstNameTextBox.Font = font;
firstNameTextBox.Password = true;
```

```

firstNameTextBox.Multiline = true;

//Add the text box in document
loadedDoc.Form.Fields.Add(firstNameTextBox);

```

[VB.NET]

```

'Create a text box
Dim firstNameTextBox As PdfTextBoxField = New PdfTextBoxField(page,
"firstNameTextBox")

'Set properties
firstNameTextBox.Bounds = New RectangleF(100, 20, 200, 20)
firstNameTextBox.Font = font
firstNameTextBox.Password = True
firstNameTextBox.Multiline = True

'Add the text box in document
loadedDoc.Form.Fields.Add(firstNameTextBox)

```

Radio Button field

Radio button fields contain a set of related buttons that can each be set to ON or OFF. Typically, at most, one radio button in a set can be ON at any given time, and selecting any one of the buttons automatically de-selects all the others. PdfRadioButtonListField class is used to create a radio button in the PDF Forms. You can create the radio button list items by using the PdfRadioButtonListItem class.

The following code example illustrates how to create radio buttons.

[C#]

```

//Create a Radio button
PdfRadioButtonListField employeesRadioList = new
PdfRadioButtonListField(page, "employeesRadioList");
loadedDoc.Form.Fields.Add(employeesRadioList);

//Create radio button items
PdfRadioButtonListItem radioItem1 = new PdfRadioButtonListItem("1-9");
radioItem1.Bounds = new RectangleF(100, 140, 20, 20);
g.DrawString("1-9", font, brush, new RectangleF(150, 145, 180, 20));

PdfRadioButtonListItem radioItem2 = new PdfRadioButtonListItem("10-49");
radioItem2.Bounds = new RectangleF(100, 170, 20, 20);
g.DrawString("10-49", font, brush, new RectangleF(150, 175, 180, 20));

```

```
//add the items to radio button group
employeesRadioList.Items.Add(radioItem1);
employeesRadioList.Items.Add(radioItem2);
```

[VB.NET]

```
'Create a Radio button
Dim employeesRadioList As PdfRadioButtonListField = New
PdfRadioButtonListField(page, "employeesRadioList")
loadedDoc.Form.Fields.Add(employeesRadioList)

'Create radio button items
Dim radioItem1 As PdfRadioButtonListFormItem = New PdfRadioButtonListFormItem("1-9")
radioItem1.Bounds = New RectangleF(100, 140, 20, 20)
g.DrawString("1-9", font, brush, New RectangleF(150, 145, 180, 20))

Dim radioItem2 As PdfRadioButtonListFormItem = New PdfRadioButtonListFormItem("10-
49")
radioItem2.Bounds = New RectangleF(100, 170, 20, 20)
g.DrawString("10-49", font, brush, New RectangleF(150, 175, 180, 20))

'add the items to radio button group
employeesRadioList.Items.Add(radioItem1)
employeesRadioList.Items.Add(radioItem2)
```

Signature fields

A signature field is a form field that contains a digital signature. **PdfSignatureField** class is used to create signature fields in PDF forms. **PdfSignature** class enables to sign the signature field with the given certificate.

[C#]

```
PdfSignatureField field = new PdfSignatureField(page, "Signature");
field.Bounds = new RectangleF(0, 0, 90, 20);
field.BackColor = new PdfColor(Color.Red);
field.BorderColor = new PdfColor(Color.Red);
field.Signature = new PdfSignature();
field.Signature.Certificate = certificate;
field.Signature.Reason = "Reason";
document.Form.Fields.Add(field);
```

[VB.NET]

```

Dim field As New PdfSignatureField(page, "Signature")
field.Bounds = New RectangleF(0, 0, 90, 20)
field.BackColor = New PdfColor(Color.Red)
field.BorderColor = New PdfColor(Color.Red)
field.Signature = New PdfSignature()
field.Signature.Certificate = certificate
field.Signature.Reason = "Reason"
document.Form.Fields.Add(field)

```



Note: For more details on public members, see class reference documentation for pdf.

Editing Form Fields

PdfLoadedForm class contains collection of loaded fields, represented by the **PdfLoadedFormFieldCollection** class, and inherited from the **PdfFieldCollection** class. The base class for each loaded field is represented by the **PdfLoadedField** class and inherited from the **PdfField** class.

The following loaded fields are supported in the library:

- Button fields
 - Push button field, represented by **PdfLoadedButtonField** class
 - Check Box field, represented by **PdfLoadedCheckBoxField** class
 - Radio button field, represented by **PdfLoadedRadioButtonListField** class
- Text fields
 - Text field, represented by **PdfLoadedTextBoxField** class
- Choice fields
 - List Box field, represented by **PdfLoadedListBoxField** class
 - Combo Box field, represented by **PdfLoadedComboBoxField** class
- Signature fields
 - Signature field, represented by **PdfLoadedSignatureField** class

You can access each field by using its index or field name. The following code example illustrates this.

[C#]

```

PdfLoadedTextBoxField field = form.Fields["fieldname"] as
PdfLoadedTextBoxField;
PdfLoadedTextBoxField ldField = form.Fields[ 0 ] as PdfLoadedTextBoxField;

```

[VB .NET]

```

Dim field As PdfLoadedTextBoxField = form.Fields["fieldname"] as
PdfLoadedTextBoxField
Dim field As PdfLoadedTextBoxField = form.Fields( 0 ) as
PdfLoadedTextBoxField

```

Essential PDF enables you to retrieve the bounds and value of the field, change the field location and size, and modify its value. Also you can get or set another available property.

The following code example illustrates how to change the bounds and value of the field.

[C#]

```

PdfLoadedDocument ldDoc = new PdfLoadedDocument(filename);
PdfLoadedForm form = ldDoc.Form;

PdfLoadedTextBoxField ldField = form.Fields[ 0 ] as PdfLoadedTextBoxField;

RectangleF newBounds = new RectangleF( 100, 100, 50, 50 );

ldField.Bounds = newBounds;
ldField.SpellCheck = true;
ldField.Text = "New text of the field.";
ldField.Password = false;

ldDoc.Save( newFileName );
ldDoc.Close();

```

[VB.NET]

```

Dim ldDoc As PdfLoadedDocument = New PdfLoadedDocument(filename)
Dim form As PdfLoadedForm = ldDoc.Form

Dim ldField As PdfLoadedTextBoxField = form.Fields( 0 ) as
PdfLoadedTextBoxField

Dim NewBounds As RectangleF = New RectangleF(100, 100, 50, 50)

ldField.Bounds = NewBounds
ldField.SpellCheck = True
ldField.Text = "New text of the field."
ldField.Password = False

ldDoc.Save (NewFileName)

```

```
ldDoc.Close()
```

4.3.3 Form Filling

Essential PDF provides support to fill AcroForm fields. You can fill the form field value either by using its field name or field index.

The following code example illustrates how to fill various loaded fields by using Essential PDF.

Filling the Text Box Field

The following code illustrates how to fill the Text Box Field.

[C#]

```
PdfLoadedTextBoxField ldField = form.Fields[0] as PdfLoadedTextBoxField;
RectangleF newBounds = new RectangleF(100, 100, 50, 50);
ldField.Bounds = newBounds;
ldField.SpellCheck = true;
ldField.Text = "New text of the field.";
ldField.Password = false;
ldField.BorderStyle = PdfBorderStyle.Dashed;
```

[VB.NET]

```
Dim ldField As PdfLoadedTextBoxField = form.Fields( 0 ) as
PdfLoadedTextBoxField

Dim NewBounds As RectangleF = New RectangleF(100, 100, 50, 50)

ldField.Bounds = NewBounds
ldField.SpellCheck = True
ldField.Text = "New text of the field."
ldField.Password = False
ldField.BorderStyle = PdfBorderStyle.Dashed
```

Formatting the text box

The following table lists some of the properties of the TextBoxField.

TextBoxField Property	Description
BackColor	Gets or sets the back color of the field.
BorderColor	Gets or sets the border color for the field.
BorderStyle	Gets or sets the border style for the field.
Border	Gets or sets the width of the field border.
ForeColor	Gets or sets the fore color for the field.
HighlightMode	Gets or sets the highlight mode of the field. It includes the following options. <i>Invert</i> <i>Outline</i> <i>Push</i> <i>NoHighlighting</i>
TextAlignment	Gets or sets the alignment of the text in the field. It includes the following options. <i>Center</i> <i>Left</i> <i>Right</i> <i>Justify</i>

[C#]

```

PdfLoadedTextBoxField txt = frm.Fields[i] as PdfLoadedTextBoxField;
txt.BorderColor = Color.SteelBlue;
txt.BorderStyle = PdfBorderStyle.Solid;
txt.BorderWidth = 1;
txt.BackColor = new PdfColor(Color.AliceBlue );
txt.ForeColor = new PdfColor(Color.Navy );
txt.HighlightMode = PdfHighlightMode.Invert;
txt.TextAlignment = PdfTextAlignment.Right;
Font f = new Font("Arial", 18f);
PdfTrueTypeFont txtfnt = new PdfTrueTypeFont(f, false);

```

```
txt.Font = txtfnt;
```

[VB.NET]

```
Dim txt As PdfLoadedTextBoxField = TryCast(frm.Fields(i),  
PdfLoadedTextBoxField)  
txt.BorderColor = Color.SteelBlue  
txt.BorderStyle = PdfBorderStyle.Solid  
txt.BorderWidth = 1  
txt.BackColor = New PdfColor(Color.AliceBlue)  
txt.ForeColor = New PdfColor(Color.Navy)  
txt.HighlightMode = PdfHighlightMode.Invert  
txt.TextAlignment = PdfTextAlignment.Right  
Dim f As New Font("Arial", 18.0F)  
Dim txtfnt As New PdfTrueTypeFont(f, False)  
  
txt.Font = txtfnt
```



Figure 63: *BorderColor = "SteelBlue"; BorderStyle = "Solid"; BorderWidth = "1"; BackColor = "AliceBlue"; ForeColor = "Navy"*

Filling the Combo Box Field

The following code illustrates how to fill the ComboBox Field.

[C#]

```
//fill combo box  
PdfLoadedComboBoxField combo = (PdfLoadedComboBoxField)doc.Form.Fields[1];  
combo.SelectedIndex = 3;
```

[VB.NET]

```
'fill combo box  
Dim combo As PdfLoadedComboBoxField = CType(doc.Form.Fields(1),  
PdfLoadedComboBoxField)  
combo.SelectedIndex = 3
```

Filling the Radio Button Field

The following code illustrates how to fill the Radio Button Field.

[C#]

```
//Fill radio button
PdfLoadedRadioButtonListField radio =
(PdfLoadedRadioButtonListField)doc.Form.Fields[2];
radio.SelectedIndex = 1;
```

[VB .NET]

```
'Fill radio button
Dim radio As PdfLoadedRadioButtonListField = CType(doc.Form.Fields(2),
PdfLoadedRadioButtonListField)
radio.SelectedIndex = 1
```

Filling the List Box Field

The following code illustrates how to fill the List Box Field.

[C#]

```
//fill list box
PdfLoadedListBoxField list = (PdfLoadedListBoxField)doc.Form.Fields[3];
list.SelectedIndex = 2;
```

[VB .NET]

```
'fill list box
Dim list As PdfLoadedListBoxField = CType(doc.Form.Fields(3),
PdfLoadedListBoxField)
list.SelectedIndex = 2
```

Filling the Check Box Field

The following code illustrates how to fill the Check Box Field.

[C#]

```
//Fill check box
PdfLoadedCheckBoxField loadField = form.Fields[4] as PdfLoadedCheckBoxField;

//Check the first item in the checkbox group
loadField.Items[0].Checked = true;

//check the checkbox if it is not grouped.
```

```
loadField.Checked = true;
```

[VB.NET]

```
'Fill check box
Dim loadField As PdfLoadedCheckBoxField = form.Fields(4) as PdfLoadedCheckBoxField

'Check the first item in the checkbox group
loadField.Items(0).Checked = True

'check the checkbox if it is not grouped.
loadField.Checked = True
```

Filling the Signature Field

The following code illustrates how to fill the Signature Field.

[C#]

```
PdfLoadedSignatureField sigField = ldoc.Form.Fields[0] as PdfLoadedSignatureField;
sigField.Signature = new PdfSignature();
sigField.Signature.Certificate = certificate;
sigField.Signature.Reason = "Reason";
```

[VB.NET]

```
Dim sigField As PdfLoadedSignatureField = TryCast(ldoc.Form.Fields(0),
PdfLoadedSignatureField)
sigField.Signature = New PdfSignature()
sigField.Signature.Certificate = certificate
sigField.Signature.Reason = "Reason"
```

You can also enumerate the fields and fill them. The following code example illustrates how to enumerate the text fields.

[C#]

```
PdfLoadedForm form = doc.Form;

PdfLoadedFormFieldCollection fields = form.Fields;

for ( i = 0; i < doc.Form.Fields.Count; i++)
```

```
{
    if (doc.Form.Fields[i] is PdfLoadedTextBoxField)
    {
        PdfLoadedTextBoxField textBox =
(PdfLoadedTextBoxField)doc.Form.Fields[i];
        textBox.Text = "Text";
    }
}
```

[VB .NET]

```
Dim form As PdfLoadedForm = doc.Form

Dim fields As PdfLoadedFormFieldCollection = form.Fields

For i = 0 To doc.Form.Fields.Count - 1 Step i + 1
    If TypeOf doc.Form.Fields(i) Is PdfLoadedTextBoxField Then
Dim textBox As PdfLoadedTextBoxField = CType(doc.Form.Fields(i),
PdfLoadedTextBoxField)
        textBox.Text = "Text"
    End If
Next
```

Form Fields**TryGetField:**

Loaded field from the PdfLoadedFormFieldCollection provides TryGetField method to obtain the form fields. It is used to get the field value from the given field name. It specifies whether the particular field is loaded or not by returning the boolean value.

Syntax:**[C#]**

```
public bool TryGetField(string fieldName, out PdfLoadedField field)
```

[VB .NET]

```
Public Function TryGetField(fieldName As String, ByRef field As PdfLoadedField)
As Boolean
```

Example:

The following code example illustrates how to get the form field with the given field name.

[C#]

```
// Load the document.
PdfLoadedDocument doc = new PdfLoadedDocument(@"..\..\Data\Form.pdf");
// Load the form from the loaded document.
PdfLoadedForm form = doc.Form;
// Load the form field collections from the form.
PdfLoadedFormFieldCollection field = new
PdfLoadedFormFieldCollection(form);
PdfLoadedField m_field = null;
string fieldValue = string.Empty;
// TryGetField Method.
if (field.TryGetField("f1-1", out m_field))
{
    (m_field as PdfLoadedTextBoxField).Text = "1";
}
```

[VB .NET]

```
' Load the document.
Dim doc As New PdfLoadedDocument("../Data/Form.pdf")
' Load the form from the loaded document.
Dim form As PdfLoadedForm = doc.Form
' Load the form field collections from the form.
Dim field As New PdfLoadedFormFieldCollection(Form)
Dim m_field As PdfLoadedField = Nothing
Dim fieldValue As String = String.Empty
' TryGetField Method.
If field.TryGetField("f1-1", m_field) Then
    TryCast(m_field, PdfLoadedTextBoxField).Text = "1"
End If
```

TryGetValue:

Loaded field from the PdfLoadedFormFieldCollection provides TryGetValue method to obtain the field values. It is used to get the field value from the given field name. It specifies whether the particular field returns true or false value.

Example - For a check box field, it returns a value whether it is checked or not.

Syntax:**[C#]**

```
public bool TryGetValue (string fieldName, out string fieldValue)
```

[VB .NET]

```
Public Function TryGetValue(fieldName As String, ByRef fieldValue As
string) As Boolean
```

Example:

The following code example illustrates how to get the form field with the given field name.

[C#]

```
// Load the document.
PdfLoadedDocument doc = new PdfLoadedDocument(@"Form.pdf");
// Load the form from the loaded document.
PdfLoadedForm form = doc.Form;
// Load the form field collections from the form.
PdfLoadedFormFieldCollection field = new
PdfLoadedFormFieldCollection(form);
PdfLoadedField m_field = null;
string fieldValue = string.Empty;
// TryGetValue Method.
if (field.TryGetValue("f1-2", out fieldValue) && fieldValue == "")
{
    (form.Fields["f1-2"] as PdfLoadedTextBoxField).Text = "2";
}
```

[VB .NET]

```
' Load the document.
Dim doc As New PdfLoadedDocument("Form.pdf")
' Load the form from the loaded document.
Dim form As PdfLoadedForm = doc.Form
' Load the form field collections from the form.
Dim field As New PdfLoadedFormFieldCollection(form)
Dim m_field As PdfLoadedField = Nothing
Dim fieldValue As String = String.Empty
```

```
' TryGetValue Method.
If field.TryGetValue("f1-2", fieldValue) AndAlso fieldValue = "" Then
    TryCast(form.Fields("f1-2"), PdfLoadedTextBoxField).Text = "2"
End If
```

4.4 PDF Convertor

This section contains detailed description on various Essential PDF conversion capabilities under the following topics.

4.4.1 HTML To PDF

WebPages and HTML pages can be imported to PDF using the *HtmlConverter*. The *HTMLConverter* converts the HTML web page to a *Bitmap* or *Metafile* image using *MSHTML*.

 MSHTML is a rendering library that is used to render HTML documents. The MSHTML library is like an engine that is used to drive the Internet Explorer.

Essential PDF renders the converted image into the PDF. You can convert a web page to PDF, either as Bitmap or Metafile types.

- Rendering web pages as Bitmap provides reasonable performance
- Rendering web pages as Metafile provides high resolution

This section covers the following:

- Converting Methods
- HTMLConvertor Options

Converting Methods

HTML documents can be converted to PDF through the following methods:

- ConvertToImage
- FromString

1. ConvertToImage

The `ConvertToImage` method converts the URL into an image. It recognizes tables, images, lists, and so on. The URL parameter can be a HTTP or HTTPS address such as "`http://www.server.com/path/file.html`", or a local physical path such as "`c:\path\file.html`".



Note: If you want to open a dynamically generated document such as .asp or aspx file, you need to invoke it through HTTP even if this file is local to your own script.

The overloaded `ConvertToImage` method enables to convert a HTML page to an image with `AspectRatio`, to maintain the ratio of the image dimension. This prevents text truncation at the corners.

[C#]

```
Image img = html.ConvertToImage("http://www.google.com", ImageType.Metafile, (int)width, -1,
AspectRatio.KeepWidth);
```

[VB.NET]

```
Dim img As Image = html.ConvertToImage("http://www.google.com", ImageType.Metafile, CInt(width), -1,
AspectRatio.KeepWidth)
```



Note: HTML To PDF conversion allows text selection and search within the generated document. However, in machines where IE9 is installed, the document would contain Bitmap image of the converted page / file there by restricting text selection and search. But this behavior can be changed for certain webpages by changing the registry value. For more details, refer the [Frequently Asked Questions section](#).

Authentication

You can use the `ConvertToImage` method to access the authenticated web pages, by passing its user credential values as arguments. The following code example illustrates this:

[C#]

```
Image img = html.ConvertToImage("http://www.google.com", ImageType.Metafile,
(int)width, -1, AspectRatio.KeepWidth, "UserName", "Password");
```

[VB .NET]

```
Dim img As Image = html.ConvertToImage("http://www.google.com",
ImageType.Metafile, CInt(width), -1, AspectRatio.KeepWidth, "UserName",
"Password")
```

2. FromString

FromString method renders HTML from the string to the image. The following code example illustrates this:

[C#]

```
public Image FromString( string html, ImageType type );
public Image FromString( string html, ImageType type, int width );
public Image FromString( string html, ImageType type, int width, int height );
public Image FromString( string html, ImageType type, int width, int height,
AspectRatio aspectRatio );
```

[VB.NET]

```
Public Image FromString(String html, ImageType type)
Public Image FromString(String html, ImageType type, Integer width)
Public Image FromString(String html, ImageType type, Integer width, Integer height)
Public Image FromString(String html, ImageType type, Integer width, Integer height, AspectRatio aspectRatio)
```

Sample code:

[C#]

```
//initialze the html converter
HtmlConverter converter = new HtmlConverter();
//convert html to pdf
Image result = converter.FromString(html, ImageType.Metafile,
(int)imgWidth, -1, AspectRatio.KeepWidth);

//Render the image in the Pdf document
PdfImage pdfImage=PdfImage.FromImage(result);
pdfImage.Draw(pdfPage, PointF.Empty, format);
```

[VB.NET]

```
'Initialize the HTML Converter
Dim converter As New HtmlConverter

'Convert the HTML file to Image
Dim result As Image = converter.FromString(htmlFilePath,
ImageType.Metafile, CType(imgWidth, Integer), -1,
AspectRatio.KeepWidth)

'Render the image in the Pdf document
```

```
Dim pdfImage As PdfImage = pdfImage.FromImage(result)
pdfImage.Draw(pdfPage, PointF.Empty, format)
```

 **Note:** Both `ConvertToImage()` and `FromString()` methods are used to convert the HTML pages whose height is less than 32767 pixels as image and the options like `EnableHyperlinks`, `EnableJavascript` and `AutoDetectPageBreak` has no effect.

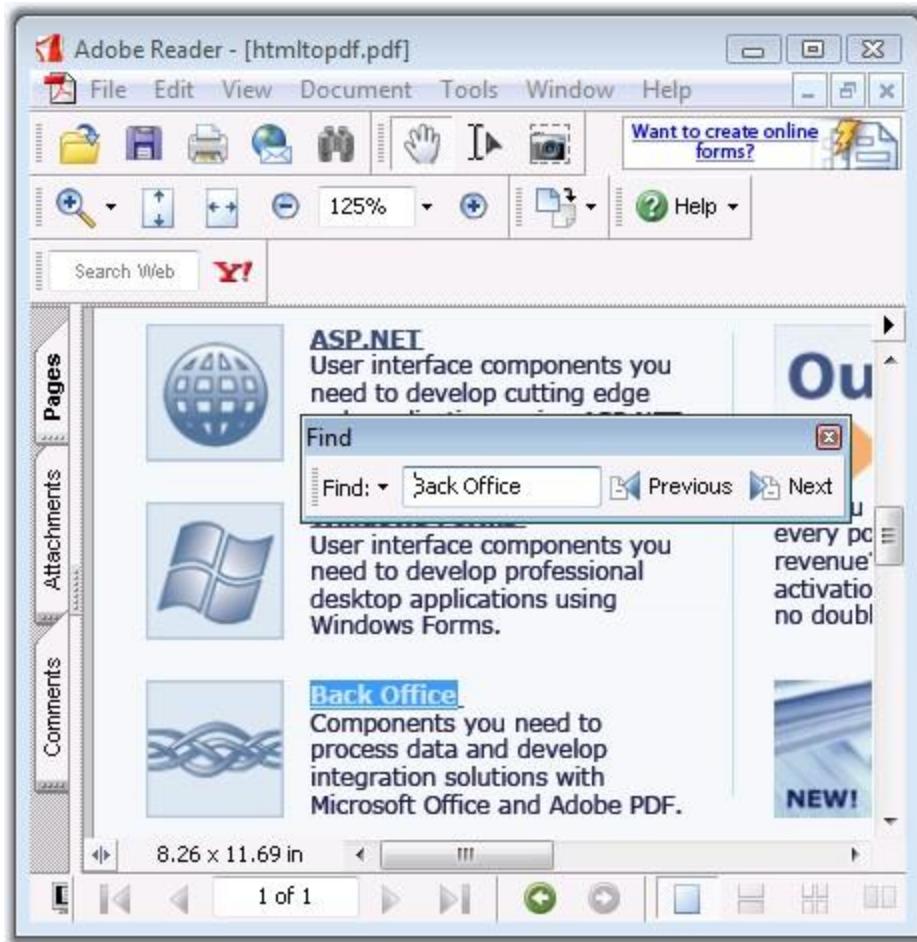


Figure 64: HTML to PDF Conversion Through `FromString` Method

HtmlConverter Options

HtmlConverter provides the following options to control HtmToPDF conversions.

- `EnableJavaScript`
- `AutoDetectPageBreak`
- `Enable Hyperlinks`

EnableJavaScript

You can control the JavaScript by using the *EnableJavaScript* property of the `HtmlConverter` class library. By default this property are set to *False*. So the JavaScript code is disabled during conversion. Set the *EnableJavaScript* property to *True* to activate the JavaScript code during conversion.

The following code example illustrates this:

[C#]

```
HtmlConverter html = new HtmlConverter();

//Activating JavaScript
html.EnableJavaScript = true;
```

[VB.NET]

```
Dim html As HtmlConverter = New HtmlConverter()

'Activating JavaScript
html.EnableJavaScript = True
```



Note: If JavaScript code is not executed by setting the *EnableJavaScript* property, it means the Internet Security Settings on the server does not allow the JavaScript execution.

Enable Hyperlink

Essential PDF supports enabling/ disabling live hyperlinks in PDF while converting web pages to PDF. The following code example illustrates this:

[C#]

```
HtmlConverter html = new HtmlConverter();

// Enabling Hyperlink
html.EnableHyperlinks = true;
```

[VB.NET]

```
Dim html As New HtmlConverter()

'Enabling Hyperlink
```

```
html.EnableHyperlinks = True
```

AutoDetectPageBreak

The *HtmlConverter* supports custom page breaks with standard CSS styles like *page-break-before:always* and *page-break-after:always* that can be applied to any HTML object. You can enable custom page breaks by setting the *AutoDetectPageBreak* property to *True*.

The following code example illustrates this:

[C#]

```
HtmlConverter html = new HtmlConverter();

// Activate the Page Break
html.AutoDetectPageBreak = true;

// Convert the Html file as an image
HtmlToPdfResult result = html.Convert(txtUrl.Text, ImageType.Metafile, (int)width, -1,
AspectRatio.KeepWidth);

// Specify the quality of the metafile
PdfMetafile mf = new PdfMetafile(result.RenderedImage as Metafile);
mf.Quality = 100;

PdfMetafileLayoutFormat format = new PdfMetafileLayoutFormat();
format.Break = PdfLayoutBreakType.FitPage;
format.Layout = PdfLayoutType.Paginate;

//Render the image in PDF document
mf.Draw(page, new PointF(0, 0), format);
```

[VB.NET]

```
Dim html As HtmlConverter = New HtmlConverter()

' Activate the Page Break
html.AutoDetectPageBreak = True

' Convert the Html file as an image
Dim result As HtmlToPdfResult = html.Convert(txtUrl.Text, ImageType.Metafile, CInt(Fix(width)), -1,
AspectRatio.KeepWidth)

' Specify the quality of the metafile
```

```

Dim mf As PdfMetafile = New PdfMetafile(TryCast(result.RenderedImage, Metafile))
mf.Quality = 100

Dim format As PdfMetafileLayoutFormat = New PdfMetafileLayoutFormat()
format.Break = PdfLayoutBreakType.FitPage
format.Layout = PdfLayoutType.Paginate

' Render the image in PDF document
mf.Draw(page, New PointF(0, 0), format)

```



Note: The custom page breaks is supported only when converting *MetaFile* to PDF. They are not supported when converting *Bitmap* to PDF.

Rendering HTML page without Splitting

To avoid the images and text split across the page breaks, while rendering a large meta file with images and text in a PDF document, disable the *SplitTextLines* and *SplitImages* properties of the *PdfMetafileLayoutFormat* class. The following code illustrates this:

[C#]

```

PdfMetafileLayoutFormat format = new PdfMetafileLayoutFormat();
format.SplitTextLines = false;
format.SplitImages = false;

```

[VB.NET]

```

Dim format As New PdfMetafileLayoutFormat()
format.SplitTextLines = False
format.SplitImages = False

```

4.4.1.1 HTML to PDF Conversion using Gecko Rendering Engine

Gecko is a free and open source layouting engine used in many applications developed by Mozilla Foundation and the Mozilla Corporation (notably the Firefox web browser), as well as in many other open source software projects. Essential PDF also supports converting Webpages to PDF using Mozilla's Gecko rendering engine.

4.4.1.1.1 Use Case Scenario

Starting with Internet Explorer 9, Microsoft has made a series of core-architectural changes to Internet Explorer. One of them was to use DirectX (a.k.a, D2D) to render webpage to achieve full-hardware acceleration development to support HTML5 standard features instead of GDI based rendering. As our HTML to PDF conversion depends on IE's GDI based rendering during conversion, our HTML Converter will not be able to generate PDF documents that contain selectable text. Hence, if the machine has IE9 or later installed, then you should consider using our Gecko based rendering. This approach will reliably generate PDF documents that are text selectable and printer friendly.

4.4.1.1.2 Prerequisites

Dlls	<ul style="list-style-type: none"> • Syncfusion.Core.dll • Syncfusion.Compression.Base.dll • Syncfusion.Pdf.Base.dll • Syncfusion.HtmlConverter.Base.dll • Syncfusion.GeckoHtmlRenderer.dll • Syncfusion.GeckoWrapper.dll
Software Development Kit (SDK)	<ul style="list-style-type: none"> • XulRunner-SDK 2.0

4.4.1.1.3 Installation Steps

To facilitate conversion, our HTML Converter depends on our Active-X Wrapper control which talks to GECKO APIs during conversion. While installing the Essential Studio, the assembly manager will register the Syncfusion.GeckoWrapper.dll in the machine. To register our Active-X wrapper control manually:

1. Copy the Syncfusion.GeckoWrapper.dll to the Bin folder of Gecko SDK (a.k.a, XulRunner-SDK 2.0.) which can download from this [location](#).
2. Register the Syncfusion.GeckoWrapper.dll using the following command in the command prompt:

`regsvr32 Syncfusion.GeckoWrapper.dll`

4.4.1.1.4 Conversion of HTML to PDF using Gecko Rendering Engine

The following code snippets explain the conversion of HTML to PDF using the Gecko Rendering Engine.

[C#]

```
//Initializing the Gecko Rendering Engine
GeckoHtmlRendererControl renderer = new GeckoHtmlRendererControl();
```

```
//Initialzing the html converter
HtmlConverter converter = new HtmlConverter(renderer);

//Converting html to pdf
HtmlToPdfResult result = converter.Convert(txtUrl.Text,
ImageType.Metafile, width, height, AspectRatio.KeepWidth);

//Rendering the image in the PDF document
result.Render(document);
```

Limitations

1. Formatting / styles created using dynamic scripts will not be rendered in the resultant PDF.
2. Other features in HTML to PDF conversion such as hyperlinks will not be available for conversion using Gecko rendering engine. However, the page breaks are supported but we can't explicitly control the page break.

4.4.1.2 Tagged PDF

HTML to PDF conversion handled using MSHTML rendering library can now generate tagged PDF documents.

Tagged PDF is a stylized use of PDF that builds on the logical structure framework. It defines a set of standard structure types and attributes that allow page content (text, graphics, and images) to be extracted and reused. The contents are accessible to users with visual impairments.

HTML documents can be converted to tagged PDFs using the **ConvertToTaggedPDF** method. It returns **PdfLayoutResult** which provides the end rectangle bounds and PDF page after the conversion.

[C#]

```
public PdfLayoutResult ConvertToTaggedPDF(PdfDocument document, string url);

public PdfLayoutResult ConvertToTaggedPDF(PdfDocument document, string url, string userName, string password);

public PdfLayoutResult ConvertToTaggedPDF(PdfDocument document, string html, string baseURL);
```

[VB.NET]

```
public PdfLayoutResult ConvertToTaggedPDF(PdfDocument document, String url)

public PdfLayoutResult ConvertToTaggedPDF(PdfDocument document, String
```

```
url, String userName, String password)
public PdfLayoutResult ConvertToTaggedPDF(PdfDocument document, String
html, String baseURL)
```

A tagged PDF can be converted from a Web page or HTML string by using the following code snippet:

[C#]

```
// Create a new PdfDocument.
PdfDocument document = new PdfDocument();
PdfLayoutResult result = null;

// Create a new instance of HtmlConverter class.
using (HtmlConverter html = new HtmlConverter())
{
    // Turn on or off various options.
    html.EnableJavaScript = true;
    html.EnableActiveXContents = true;

    // Convert to Tagged PDF.
    result = html.ConvertToTaggedPDF(document, url);
}

// Save and close the document.
document.Save(@"Sample.pdf");
document.Close(true);
```

[VB .NET]

```
' Create a new PdfDocument.
Dim document As New PdfDocument()
Dim result As PdfLayoutResult = Nothing

' Create a new instance of HtmlConverter class.
Using html As New HtmlConverter()
    ' Turn on or off various options.
```

```

    html.EnableJavaScript = True
    html.EnableActiveXContents = True

    ' Convert to Tagged PDF.
    result = html.ConvertToTaggedPDF(document, url)

End Using

' Save and close the document.
document.Save("Sample.pdf")
document.Close(True)

```



Note: The HTML to PDF conversion, which creates a metafile during the evolution, would interpret the content as either text or an image. The outcome of this PDF would contain only paragraph and figure tags; hyperlinks are not supported.

4.4.2 Text Extraction

A PDF file represents an ordered sequence of fixed pages. The planned appearance of everything that each page contains is completely specified down to the smallest detail. All the graphics, images, and text are specified to appear at precise spots on the page, in a particular color, of a given and fixed size, and so on.

Essential PDF provides support to extract text from an existing PDF document. By using the **ExtractText** method, you can extract the text, page by page.

The following code example illustrates this.

[C#]

```

// Load an existing PDF
PdfLoadedDocument ldoc = new PdfLoadedDocument("Sample.pdf");

// Loading first page
PdfLoadedPage page = ldoc.Pages[0];

// Extract text from first page
string s = page.ExtractText();

```

[VB .NET]

```
' Load an existing PDF
Dim ldoc As PdfLoadedDocument = New PdfLoadedDocument("Sample.pdf")

' Loading first page
Dim page As PdfLoadedPage = ldoc.Pages(0)

' Extract text from first page
Dim s As String = page.ExtractText()
```



Note: The text will be extracted in the order in which it is written in the document stream. It is not in the order in which it is viewed in the PDF viewer.

4.4.3 Image Extraction

Essential PDF provides support to extract images from an existing PDF document. You can extract images by using the **ExtractImages** method in the **PdfLoadedPage** class.

The following code example illustrates how to extract images from a PDF document.

[C#]

```
// Load an existing PDF
PdfLoadedDocument ldoc = new PdfLoadedDocument("Sample.pdf");

// Loading first page
PdfLoadedPage page = ldoc.Pages[0];

// Extract images from first page
Image[] img = page.ExtractImages();
```

[VB .NET]

```
' Load an existing PDF
Dim ldoc As PdfLoadedDocument = New PdfLoadedDocument("Sample.pdf")

' Loading first page
Dim page As PdfLoadedPage = ldoc.Pages(0)

' Extract images from first page
Dim img As Image() = page.ExtractImages()
```

Limitations

Image extraction does not work with the following constraints:

- If the image has multiple filters in the PDF document.
- You cannot extract the image which is placed on the Xobject, also known as **PdfTemplate**.

4.4.4 Doc To PDF

Essential DocIO enables to export the Word document into a PDF document. By using the **ConvertToPDF** method of the **DocToPDFConverter** class, you can convert the Word document to PDF, and save the PDF document.



Note: You need to have Essential PDF and Essential DocIO installed in your system. This is because "Syncfusion.DocToPDFConverter.Base.dll" is conditionally shipped when both DocIO.Base and Pdf.Base is installed.

This section covers the following:

- Assemblies Dependent for this Conversion
- Supported Elements and Limitations
- UnSupported Elements and Limitation

Assembly Dependency for this Conversion

- Syncfusion.DocToPDFConverter.Base.dll
- Syncfusion.DocIO.Base.dll
- Syncfusion.Pdf.Base.dll
- Syncfusion.Core.dll
- Syncfusion.Compression.Base.dll

The following code illustrates how to convert a word document, say, "sample.doc" to a PDF document.

[C#]

```
WordDocument wordDoc = new WordDocument("sample.doc");
DocToPDFConverter converter = new DocToPDFConverter();
```

```
//Convert word document into PDF document
PdfDocument pdfDoc = converter.ConvertToPDF(wordDoc);

//Save the pdf file
pdfDoc.Save("DoctoPDF.pdf");
```

[VB .NET]

```
Dim wordDoc As New WordDocument("sample.doc")
Dim converter As New DocToPDFConverter()

'Convert word document into PDF document
Dim pdfDoc As PdfDocument = converter.ConvertToPDF(wordDoc)

'Save the pdf file
pdfDoc.Save("DoctoPDF.pdf")
```

Supported Elements

This feature provides support for the following elements.

- Paragraph and Character Formatting
- MultiColumn Text
- Headers and Footers
- Bulleted, Numbered and MultiLevel Lists
- Images
- Tables (both simple and nested)
- Breaks (page, section, linebreak, etc.)
- OLEObject
- Text Box
- Page Settings and Background Image
- Document Properties

Paragraph and Character Formatting

This feature supports almost all the paragraph formatting options except Full-Justification. The supported paragraph formatting options are as follows.

- Paragraph and Character Fonts
- Font styles (Bold, Italic, Underline and Strike through)
- Subscript and Superscript
- Paragraph and Text Highlighting
- Indents, tabs and spaces

- Line Spacing
- Left, Right and Center Justification

Known Limitations

- Borders around paragraphs.
- Full Justification.

- **MultiColumn Text**

Word document containing multicolumn text is supported.

Known Limitations - But the output may look different in case full-justification formatting is applied onto the columns.

- **Headers and Footers**

Page headers and footers are supported and can contain images, text and page number fields.

- **Bulleted, Numbered and MultiLevel Lists**

Bulleted, numbered and multilevel lists are supported with proper indentation and alignments as represented in the Word document.

Known Limitations - In some case, the image bullets which is set on document may be replaced by the following symbol in the generated document. For eg :

 Essential DocIO. Essential XlsIO. Essential PDF.	==> Will be replaced as ==>	<ul style="list-style-type: none"> • Essential DocIO. • Essential XlsIO. • Essential PDF.
---	---	--

Figure 65: Image replacement

- **Images**

The images present in the document are supported along with their corresponding positions and sizes.

Known Limitations - However, the images placed inside a shape will not be preserved in the generated PDF document.

- **Tables**

Both simple and nested tables are supported with proper preservation of text formatting and images present inside the table cell.

Known Limitations

- Tables making use of patterns and 3D borders will not be retained in the output document.
- Absolutely positioned tables are not supported.

- **Breaks**

Columns, section, line and page breaks are fully supported.

- **OLEObject**

OLEObjects are partially supported, i.e., the image, which represents a particular document that will be available in the generated PDF document. But the object associated with the object will not be converted into the generated document.

- **Text Box**

The text value present in the text box will be rendered as text in its actual position in the generated PDF document.

- **Page Settings**

The actual page settings will be preserved in the generated PDF document which includes page size, orientation, page borders and its background image, if available.

- **Document Properties**

The document properties present in the Word document will also be preserved in the generated PDF Document.

Unsupported Elements

The following are the list of unsupported elements, which will not be preserved in the generated PDF document.

- Shapes and AutoShapes
- Comments
- Hyperlinks
- Bookmarks
- Footnotes and Endnotes
- Dynamic Fields
- Charts
- Table of Contents

Known Limitations - Pagination

Pagination

Essential DocIO, when generating the PDF document, makes sensible decisions while laying out the text and its supported elements. However, pagination is not guaranteed with all the documents.

4.4.5 XPS to PDF

An XPS (XML Paper Specification) document, standardized by Ecma International, can be now converted to PDF.

The XPS document format consists of XML structured markup that defines the layout of a document and the visual appearance of each page, along with rendering rules for distributing, archiving, rendering, processing, and printing the documents. Similar to PDF, XPS is also a fixed-layout document format which helps to preserve document fidelity and to achieve device-independent document appearance.

XPS documents can be converted to PDF using the **Convert** method of the **XPSToPdfConverter** class.

 **Note:** You need to add the **Syncfusion.XPS** namespace to work with the **XPSToPdfConverter** class.

[C#]

```
public PdfDocument Convert(byte[] file);
public PdfDocument Convert(string fileName);
public PdfDocument Convert(Stream file);
```

[VB.NET]

```
public PdfDocument Convert(byte[] file)

public PdfDocument Convert(String fileName)

public PdfDocument Convert(Stream file)
```

An XPS document can be converted to PDF using the following code snippet:

[C#]

```
// Create converter class.
XPSToPdfConverter converter = new XPSToPdfConverter();

// Convert the XPS to PDF
PdfDocument document = converter.Convert(fileName);

// Save and close the document
document.Save("Sample.pdf");
document.Close(true);
```

[VB.NET]

```
' Create converter class.
Dim converter As New XPSToPdfConverter()

' Convert the XPS to PDF
Dim document As PdfDocument = converter.Convert(fileName)

' Save and close the document
document.Save("Sample.pdf")
document.Close(True)
```

Supported Elements

Element	Convert to PDF
---------	----------------

ArcSegment	Yes
Canvas	Yes
DocumentOutline	No
DocumentReference	No
FigureStructure	No
FixedPageResources	Yes
Glyphs	Yes
Gradient	Yes
ImageBrush	Yes
Intent	Yes
LinkTarget	Yes
ListItemStructure	Yes
ListStructure	Yes
MatrixTransform	Yes
NamedElement	No
OutlineEntry	No
PageContent	Yes
PageContentLinkTargets	No
ParagraphStructure	No
Path	Yes
PolyBezierSegment	Yes
PolyLineSegment	Yes
PolyQuadraticBezierSegment	Yes
ResourceDictionary	Yes
SectionStructure	No
SignBy	No
SignatureDefinition	No

SignatureDefinitions	No
SigningLocation	No
SolidColorBrush	Yes
SpotLocation	No
Story	No
TableStructure	No
VisualBrush	No

4.4.6 Tutorial

This tutorial will show you how easy it is to get started using Essential PDF. It will give you a basic introduction to the concepts you need to know before getting started with the product and some tips and ideas on how to implement PDF into your projects. The lessons in this tutorial are meant to introduce you to PDF with simple step-by-step procedures.



Note: Refer to the Class Reference documentation for comprehensive information on the classes.

4.4.6.1 Importing

The integrated HTML to PDF Converter is implemented by using the **HtmlConverter** class. It basically includes the functionality of the HTML to PDF Converter product, and additionally offers the possibility to specify the position and the size of the PDF content.

The following code example illustrates this.

[C#]

```
// Convert web page into image.
HtmlConverter html = new HtmlConverter();
Image img= html.ConvertToImage("www.syncfusion.com", ImageType.Metafile,
1024)

// Draw the image into the PDF document as metafile
// Create metafile image
PdfMetafile metafile = (PdfMetafile)PdfImage.FromImage(img);

// Specify the quality of the metafile
metafile.Quality = 100;
```

```

// Set the layout format
PdfMetafileLayoutFormat format = new PdfMetafileLayoutFormat();
format.Break = PdfLayoutBreakType.FitPage;
format.Layout = PdfLayoutType.Paginate;

// Prevent text getting break at the page breaks
format.SplitTextLines = false;

// Draw the converted image to PDF
metafile.Draw(page, new RectangleF(0, 0, img.Width, img.Height), format);

// Draw the image into the PDF document as bitmap
PdfImage image = new PdfBitmap(img);
PdfLayoutFormat format = new PdfLayoutFormat();
format.Break = PdfLayoutBreakType.FitPage;
format.Layout = PdfLayoutType.Paginate;
image.Draw(page, new RectangleF(0, 0, pageSize.Width, pageSize.Height),
format);

```

[VB.NET]

```

' Convert web page into image.
Dim html As HtmlConverter = New HtmlConverter()
Dim img As Image = html.ConvertToImage("www.syncfusion.com",
ImageType.Metafile, 1024)

' Draw the image into the PDF document as metafile
' Create metafile image
Dim metafile As PdfMetafile = CType(PdfImage.FromImage(img), PdfMetafile)

' Specify the quality of the metafile
metafile.Quality = 100

' Set the layout format
Dim format As PdfMetafileLayoutFormat = New PdfMetafileLayoutFormat()
format.Break = PdfLayoutBreakType.FitPage
format.Layout = PdfLayoutType.Paginate

' Prevent text getting break at the page breaks
format.SplitTextLines = False

' Draw the converted image to PDF
metafile.Draw(page, New RectangleF(0, 0, img.Width, img.Height), format)

```

```
' Draw the image into the PDF document as bitmap
Dim image As PdfImage = New PdfBitmap(img)
Dim format As PdfLayoutFormat = New PdfLayoutFormat()
format.Break = PdfLayoutBreakType.FitPage
format.Layout = PdfLayoutType.Paginate
image.Draw(page, New RectangleF(0, 0, pageSize.Width, pageSize.Height),
format)
```

4.5 Asynchronous Support

Essential PDF can read, write, and merge PDF files using asynchronous methods. This is a new approach introduced in Visual Studio 2012 that enables applications to use asynchronous programming. The following list of public APIs in Essential PDF support asynchronous programming.



Note: Asynchronous support is applicable only to Windows Store apps.

Method	Description	Parameters	Return Type
OpenAsync	Opens an existing PDF document.	async Task<bool> OpenAsync(StorageFile stFile)	Boolean
		async Task<bool> OpenAsync(StorageFile stFile, string password)	Boolean
		async Task<bool> OpenAsync(byte[] bytes)	Boolean
		async Task<bool> OpenAsync(byte[] bytes, string password)	Boolean
		async Task<bool> OpenAsync(Stream stream)	Boolean
		async Task<bool> OpenAsync(Stream stream, string password)	Boolean
SaveAsync	Saves the document to a stream in asynchronous mode.	async Task<bool> SaveAsync(Stream stream)	Boolean
SaveAsync	Saves the PDF file to a storage file.	async Task<bool> SaveAsync(StorageFile stFile)	Boolean
Save	Saves the modified document.	async Task<bool> Save()	Boolean

ImportPageAsync	Imports a page in asynchronous mode.	<code>async Task<PdfPageBase></code> ImportPageAsync(<code>PdfLoadedDocument</code> <code>IdDoc</code> , <code>int pageIndex</code>)	<code>PdfPageBase</code>
		<code>async Task<PdfPageBase></code> ImportPageAsync(<code>PdfLoadedDocument</code> <code>IdDoc</code> , <code>PdfPageBase</code> <code>page</code>)	<code>PdfPageBase</code>
ImportPageRange Async	Imports page range in asynchronous mode.	<code>async Task<PdfPageBase></code> ImportPageRangeAsync(<code>PdfLoadedDocument</code> <code>IdDoc</code> , <code>int startIndex</code> , <code>int endIndex</code>)	<code>PdfPageBase</code>
AppendAsync	Appends documents in asynchronous mode.	<code>async Task<bool></code> AppendAsync(<code>PdfLoadedDocument</code> <code>IdDoc</code>)	<code>Boolean</code>
MergeAsync	Merges documents in asynchronous mode.	<code>async static Task<PdfDocumentBase></code> MergeAsync(<code>PdfDocumentBase</code> <code>dest</code> , <code>PdfLoadedDocument</code> <code>src</code>)	<code>PdfDocumentBase</code>

4.6 Supported Elements

Supported and non-supported Excel elements of Essential PDF for Windows, ASP.NET, WPF, ASP.NET MVC, Silverlight, and Windows Store apps are listed in the following table.

Features	Windows Forms/WPF	ASP .NET/ ASP.NET MVC (Medium Trust)	ASP .NET/ ASP.NET MVC (Full Trust)	Silverlight	Windows Store Apps
Drawing Text	Yes	Yes	Yes	Yes	Yes
Text Formatting	Yes	Yes	Yes	Yes	Yes
Multilingual Support	Yes	No	Yes	No	No
Drawing RTL text	Yes	No	Yes	No	No
Text Extraction	Yes	Yes	Yes	No	No
Unicode	Yes	No	Yes	No	No
Pagination	Yes	Yes	Yes	Yes	Yes
Graphics					
Pen and Brush	Yes	Yes	Yes	Yes	Yes
Layers	Yes	Yes	Yes	Yes	Yes

Features	Windows Forms/WPF	ASP .NET/ ASP.NET MVC (Medium Trust)	ASP .NET/ ASP.NET MVC (Full Trust)	Silverlight	Windows Store Apps
Transparent Graphics	Yes	Yes	Yes	No	No
Color Spaces	Yes	Yes	Yes	Yes	Yes
Image Extraction	Yes	Yes	Yes	No	No
Enhanced Printing Support	Yes	Yes	Yes	Yes	Yes
Barcode	Yes	Yes	Yes	No	No
Document-level Operations					
Merge Documents	Yes	Yes	Yes	Yes	Yes
Split Document	Yes	Yes	Yes	Yes	Yes
Overlay Documents	Yes	Yes	Yes	Yes	Yes
Import Pages	Yes	Yes	Yes	Yes	Yes
Stamp	Yes	Yes	Yes	Yes	Yes
Booklet	Yes	Yes	Yes	Yes	Yes
Document Settings					
Custom Metadata	Yes	Yes	Yes	No	No
Document Properties	Yes	Yes	Yes	Yes	Yes
Page Orientation	Yes	Yes	Yes	Yes	Yes
Page Sizes	Yes	Yes	Yes	Yes	Yes
Viewer Preferences	Yes	Yes	Yes	Yes	Yes
Forms					
Fields	Yes	Yes	Yes	Yes	Yes
Form Filling	Yes	Yes	Yes	Yes	Yes
Flatten	Yes	Yes	Yes	Yes	Yes

Features	Windows Forms/WPF	ASP .NET/ ASP.NET MVC (Medium Trust)	ASP .NET/ ASP.NET MVC (Full Trust)	Silverlight	Windows Store Apps
Import Form Data	Yes	Yes	Yes	Yes	Yes
Form Export	Yes	Yes	Yes	Yes	Yes
Document Conversion					
TIFF to PDF	Yes	No	Yes	No	No
HTML to PDF	Yes	No	Yes	No	No
RTF To PDF	Yes	No	Yes	No	No
DOC To PDF	Yes	No	Yes	No	No
Excel To PDF	Yes	No	Yes	No	No
PDF OCR	Yes	No	Yes	No	No
PDF Standards					
PDF/ A-1b Compliance	Yes	Yes	Yes	Yes	Yes
PDF/x1a: 2001 Compliance	Yes	Yes	Yes	Yes	Yes
Fonts					
Standard Fonts	Yes	Yes	Yes	Yes	Yes
CJK Fonts	Yes	Yes	Yes	Yes	Yes
True Type Fonts	Yes	No	Yes	No	No
Unicode True Type	Yes	No	Yes	No	No
OTF Font	No	No	No	No	No
Images					
Scalar Images	Yes	Yes	Yes	Yes*	Yes*
Soft Mask	Yes	Yes	Yes	Yes	Yes
Vector Images	Yes	No	Yes	No	No
Watermarks	Yes	Yes	Yes	Yes	Yes
Tables					

Features	Windows Forms/WPF	ASP .NET/ ASP.NET MVC (Medium Trust)	ASP .NET/ ASP.NET MVC (Full Trust)	Silverlight	Windows Store Apps
ADO.Net Tables Support	Yes	Yes	Yes	No	No
Cell / Row / Column Formatting	Yes	Yes	Yes	Yes	Yes
Header	Yes	Yes	Yes	Yes	Yes
Pagination	Yes	Yes	Yes	Yes	Yes
Borders	Yes	Yes	Yes	Yes	Yes
RowSpan** and ColumnSpan	Yes	Yes	Yes	Yes	Yes
Nested**	Yes	Yes	Yes	Yes	Yes
Cell Padding and Spacing	Yes	Yes	Yes	Yes	Yes
Page Level Operations					
Headers and Footers	Yes	Yes	Yes	Yes	Yes
Page Label	Yes	Yes	Yes	Yes	Yes
Automatic Fields	Yes	Yes	Yes	Yes	Yes



Note:

*Only .Jpeg format images are supported for Silverlight version.

**Supported only in PdfGrid class

5 Frequently Asked Questions

This section illustrates the solutions for various task-based queries about Essential PDF.

5.1 PDF Generator

This section shows some specific tasks that are supported in a PDF Generator.

5.1.1 Common

This section shows some common tasks in a PDF Generator.

5.1.1.1 How To Add Sections And Pages?

Essential PDF enables to add any number of sections and pages. A PDF document should contain atleast one section. Each section can have any number of pages in it. PdfSection class is used to create sections and PdfPage class is used to add pages.

[C#]

```
PdfDocument document = new PdfDocument();

//Adds a section to the document
PdfSection section = doc.Sections.Add();

//Adds a page to the section
PdfPage = section.Pages.Add();
```

[VB.NET]

```
Dim document As PdfDocument = New PdfDocument()

'Adds a section to the document
Dim section As PdfSection = doc.Sections.Add()

'Adds a page to the section
Dim page As PdfPage = section.Pages.Add()
```

5.1.1.2 How To Compress a PDF Document?

Compression is used for reducing the size of the created PDF document. Essential PDF controls the compression level of the document by using the **PdfCompressionLevel** class with the help of the LZW and zlib/deflate compression algorithms. Both LZW and Flate methods compress either binary data or ASCII text, but always produce binary data, even if the original data is text.

The following compression levels are supported by Essential PDF.

- Best
- BestSpeed
- BelowNormal
- AboveNormal
- None

[C#]

```
// Compressing PDF document
doc.Compression = PdfCompressionLevel.Normal;
```

[VB .NET]

```
' Compressing PDF document
doc.Compression = PdfCompressionLevel.Normal
```

5.1.1.3 How To Create a Borderless Table?

You can create a borderless table by making use of the **Style** property of the **PdfLightTable** class. This is achieved by setting the border color of the table to **Transparent**. The following code example illustrates this.

[C#]

```
// Creating Border with transparent color.
PdfPen borderPen = new PdfPen(Color.Transparent);
borderPen.Width = 1.0f;

// Assigning the border pen to table cell.
PdfCellStyle defStyle = new PdfCellStyle();
defStyle.Font = font;
defStyle.BorderPen = borderPen;
table.Style.DefaultStyle = defStyle;
```

[VB.NET]

```
' Creating Border with transparent color.
Dim borderPen As PdfPen = New PdfPen(Color.Transparent)
borderPen.Width = 1.0F

' Assigning the border pen to table cell.
Dim defStyle As PdfCellStyle = New PdfCellStyle()
defStyle.Font = font
defStyle.BorderPen = borderPen
table.Style.DefaultStyle = defStyle
```

5.1.1.4 How To Create Page Labels?

You will be able to find a Pages tab in Adobe Reader which contains small page images. Also you will be able to find text underneath each image. You can edit that text by using the **PdfPageLabel** class. This class allows specifying the prefix, numbering style, and starting number for a page group, which is a section. You can create and initialize an instance of this class and assign it to the **Section** property.

[C#]

```
// Create a new document class object.
PdfDocument doc = new PdfDocument();

// Create few sections with few pages in each.
for (int i = 0; i < 3; ++i)
{
    PdfSection section = doc.Sections.Add();
    PdfPageLabel label = new PdfPageLabel();
    label.Prefix = "Sec" + i + "-";
    section.PageLabel = label;

    PdfPage page;
    for (int j = 0; j < 10; ++j)
    {
        page = section.Pages.Add();
    }
}
doc.Save(filename);
```

[VB.NET]

```
' Create a new document class object.
Private doc As PdfDocument = New PdfDocument()

' Create few sections with few pages in each.
For i As Integer = 0 To 2
Dim section As PdfSection = doc.Sections.Add()
Dim label As PdfPageLabel = New PdfPageLabel()
    label.Prefix = "Sec" & i & "-"
    section.PageLabel = label

Dim page As PdfPage
    For j As Integer = 0 To 9
        page = section.Pages.Add()
    Next j
Next i
doc.Save(filename)
```

5.1.1.5 How To Draw a SoftMask Image Into a PDF Document?

Essential PDF supports drawing mask images over other images. The **Mask** property of the **PdfBitmap** class is used for drawing masked images.

[C#]

```
// Bitmap with Tiff image mask.
PdfBitmap image = new PdfBitmap(tifImage);
(image as PdfBitmap).Mask = new PdfImageMask(new PdfBitmap(bmpImage));
page.Graphics.DrawString("Image mask", font, brush, new PointF(10, 350));
g.DrawImage(image, 10, 450);
```

[VB.NET]

```
' Bitmap with Tiff image mask.
Dim image As PdfBitmap = New PdfBitmap(tifImage)
(TryCast(image, PdfBitmap)).Mask = New PdfImageMask(New PdfBitmap(bmpImage))
page.Graphics.DrawString("Image mask", font, brush, New PointF(10, 350))
g.DrawImage(image, 10, 450)
```

5.1.1.6 How To Embed 3D Files In a PDF Document?

Essential PDF provides support to embed collections of three-dimensional objects (such as those used by CAD software) in PDF files with the help of the **Pdf3DAnnotation** class.

The following code example illustrates how to embed a U3D file.

[C#]

```
// Pdf 3D Annotation
Pdf3DAnnotation annot = new Pdf3DAnnotation(new RectangleF(10, 70, 150,
150), @"..\..\Data\AQuick Example.u3d");

// Create font, font style and brush.
Font f = new Font("Calibri", 11, FontStyle.Regular);
PdfFont font = new PdfTrueTypeFont(f, false);
PdfBrush brush = new PdfSolidBrush(Color.DarkBlue);
PdfBrush bgbrush = new PdfSolidBrush(Color.WhiteSmoke);
PdfColor color = new PdfColor(Color.Thistle);

page.Graphics.DrawRectangle(bgbrush, new RectangleF(10, 270, 150, 150));

// Pdf 3D Appearance
annot.Appearance = new PdfAppearance(annot);
annot.Appearance.Normal.Graphics.DrawString("Click to activate", font,
brush, new PointF(40, 40));
annot.Appearance.Normal.Draw(page, new PointF(annot.Location.X,
annot.Location.Y));

Pdf3DProjection projection = new Pdf3DProjection();
projection.ProjectionType = Pdf3DProjectionType.Perspective;

projection.FieldOfView = 10;
projection.ClipStyle = Pdf3DProjectionClipStyle.ExplicitNearFar;
projection.NearClipDistance = 10;

Pdf3DActivation activation = new Pdf3DActivation();
activation.ActivationMode = Pdf3DActivationMode.ExplicitActivation;
activation.ShowToolbar = false;
annot.Activation = activation;

Pdf3DBackground background = new Pdf3DBackground();
background.Color = color;

Pdf3DRendermode rendermode = new Pdf3DRendermode();
rendermode.Style = Pdf3DRenderStyle.Solid;

Pdf3DLighting lighting = new Pdf3DLighting();
```

```

lighting.Style = Pdf3DLightingStyle.Headlamp;

// Create the default view.
Pdf3DProjection projection1 = new
Pdf3DProjection(Pdf3DProjectionType.Perspective);
projection1.FieldOfView = 50;
projection1.ClipStyle = Pdf3DProjectionClipStyle.ExplicitNearFar;
projection1.NearClipDistance = 10;

Pdf3DView defaultView = CreateView("Default View", new float[] { -0.382684f,
0.92388f, -0.000000766026f, 0.18024f, 0.0746579f, 0.980785f, 0.906127f,
0.37533f, -0.19509f, -122.669f, -112.432f, 45.6829f }, 131.695f, background,
projection, rendermode, lighting);

annot.Views.Add(defaultView);

// Add the pdf Annotation.
page.Annotations.Add(annot);

```

[VB .NET]

```

' Pdf 3D Annotation
Dim annot As Syncfusion.Pdf.Interactive.Pdf3DAnnotation = New
Syncfusion.Pdf.Interactive.Pdf3DAnnotation(New RectangleF(10, 70, 150, 150),
"..\..\Data\AQuick Example.u3d")

' Create font, font style and brush.
Dim f As Font = New Font("Calibri", 11, FontStyle.Regular)
Dim font As Syncfusion.Pdf.Graphics.PdfFont = New
Syncfusion.Pdf.Graphics.PdfTrueTypeFont(f, False)
Dim brush As Syncfusion.Pdf.Graphics.PdfBrush = New
Syncfusion.Pdf.Graphics.PdfSolidBrush(color.DarkBlue)
Dim bgbrush As Syncfusion.Pdf.Graphics.PdfBrush = New
Syncfusion.Pdf.Graphics.PdfSolidBrush(color.WhiteSmoke)
Dim color As Syncfusion.Pdf.Graphics.PdfColor = New
Syncfusion.Pdf.Graphics.PdfColor(color.Thistle)

page.Graphics.DrawRectangle(bgbrush, New RectangleF(10, 270, 150, 150))

' Pdf 3D Appearance
annot.Appearance = New PdfAppearance(annot)
annot.Appearance.Normal.Graphics.DrawString("Click to activate", font,
brush, New PointF(40, 40))
annot.Appearance.Normal.Draw(page, New PointF(annot.Location.X,
annot.Location.Y))

Dim projection As Syncfusion.Pdf.Interactive.Pdf3DProjection = New

```

```

Syncfusion.Pdf.Interactive.Pdf3DProjection()
projection.ProjectionType = Pdf3DProjectionType.Perspective

projection.FieldOfView = 10
projection.ClipStyle = Pdf3DProjectionClipStyle.ExplicitNearFar
projection.NearClipDistance = 10

Dim activation As Syncfusion.Pdf.Interactive.Pdf3DActivation = New
Syncfusion.Pdf.Interactive.Pdf3DActivation()
activation.ActivationMode = Pdf3DActivationMode.ExplicitActivation
activation.ShowToolbar = False
annot.Activation = activation

Dim background As Syncfusion.Pdf.Interactive.Pdf3DBackground = New
Syncfusion.Pdf.Interactive.Pdf3DBackground()
background.Color = color

Dim rendermode As Syncfusion.Pdf.Interactive.Pdf3DRendermode = New
Syncfusion.Pdf.Interactive.Pdf3DRendermode()
rendermode.Style = Pdf3DRenderStyle.Solid

Dim lighting As Syncfusion.Pdf.Interactive.Pdf3DLighting = New
Syncfusion.Pdf.Interactive.Pdf3DLighting()
lighting.Style = Pdf3DLightingStyle.Headlamp

' Create the default view.
Dim projection1 As Syncfusion.Pdf.Interactive.Pdf3DProjection = New
Syncfusion.Pdf.Interactive.Pdf3DProjection(Pdf3DProjectionType.Perspective)
projection1.FieldOfView = 50
projection1.ClipStyle = Pdf3DProjectionClipStyle.ExplicitNearFar
projection1.NearClipDistance = 10

Dim defaultView As Syncfusion.Pdf.Interactive.Pdf3DView =
CreateView("Default View", New Single() {-0.382684F, 0.92388F, -
0.000000766026F, 0.18024F, 0.0746579F, 0.980785F, 0.906127F, 0.37533F, -
0.19509F, -122.669F, -112.432F, 45.6829F}, 131.695F, background, projection,
rendermode, lighting)

annot.Views.Add(defaultView)

' Add the pdf Annotation.
page.Annotations.Add(annot)

```

5.1.1.7 How To Embed Fonts?

You can embed fonts by using True type fonts. The following code example illustrates this.

[C#]

```
Font fnt = new Font("c:\\\\Arial.ttf", 12f);

PdfFont font = new PdfTrueTypeFont(fnet, true);
page.Graphics.DrawString("Embedded font", font, brush, 0, 100);
```

[VB.NET]

```
Dim fnt As PdfFont = New Font("c:\\\\Arial.ttf", 12f)
Dim font As PdfFont = New PdfTrueTypeFont(fnet, True)
page.Graphics.DrawString("Embedded font", font, brush, 0, 100)
```

5.1.1.8 How To Find the End Position Of a Table?

There is often a need to determine the end position of a published table since the next table needs to be positioned after the first one. This position is determined by using the **PdfLightTableLayoutResult** object of the table returned by the **PdfLightTable.Draw** method. This finds the exact position of the table, even if it spans for multiple pages. **PdfLightTableLayoutResult** has several methods to work with the bounds and page where the table ends.

[C#]

```
// Draw a large table that can span for multiple pages and get the result.
PdfLightTableLayoutResult result = table.Draw(page, PointF.Empty);

// Get the location where the table ends
PointF location = new PointF(bounds.Left, bounds.Bottom);

// Draw another table just below the previous table
table.Draw(result.Page, location);
```

[VB.NET]

```
' Draw a large table that can span for multiple pages and get the result.
Dim result As PdfLightTableLayoutResult = table.Draw(page, PointF.Empty)

' Get the location where the table ends
Dim location As PointF = New PointF(bounds.Left, bounds.Bottom)

' Draw another table just below the previous table
table.Draw(result.Page, location)
```

5.1.1.9 How To Render Unicode Text [CJK Fonts] In a PDF Document?

Essential PDF provides support for Unicode languages. You can render unicode text with the help of the **PdfCjkStandardFont** class. Languages like Japanese, Korean, Simplified Chinese, Traditional Chinese, and so on, are classified under CJK Fonts.

[C#]

```
// Apply Cjk font for the chinese text.
PdfCjkStandardFont font = new
PdfCjkStandardFont(PdfCjkFontFamily.HeiseiKakuGothicW5, 12F,
PdfFontStyle.Regular);
page.Graphics.DrawString(unicode_text, font, PdfBrushes.Black, new
RectangleF(0F, 0F, (float)(page.Size.Width), (float)(page.Size.Height)));
```

[VB.NET]

```
' Apply Cjk font for the chinese text.
Dim font As PdfCjkStandardFont = New
PdfCjkStandardFont(PdfCjkFontFamily.HeiseiKakuGothicW5, 12.0F,
PdfFontStyle.Regular)
page.Graphics.DrawString(unicode_text, font, PdfBrushes.Black, New
RectangleF(0.0F, 0.0F, CSng(page.Size.Width), CSng(page.Size.Height)))
```

You can also apply cjk fonts from disk and embed them. The following code example illustrates this.

[C#]

```
//Add CJK font to the font collection
string font = "gulim.ttf";
PrivateFontCollection fcol = new PrivateFontCollection();
fcol.AddFontFile(font);
Font f = new Font(fcol.Families[0], 14);
PdfTrueTypeFont font = new PdfTrueTypeFont(f, true);
string koreanText = "korean.txt";

//Read the text from text file
StreamReader reader = new StreamReader(koreanText, Encoding.Unicode);
string text = reader.ReadToEnd();
reader.Close();

page.Graphics.DrawString(text, font, brush, PointF.Empty);
```

[VB .NET]

```
//Add CJK font to the font collection
string font = "gulim.ttf";
PrivateFontCollection fcol = new PrivateFontCollection();
fcol.AddFontFile(font);
Font f = new Font(fcol.Families[0], 14);
PdfFont font = new PdfTrueTypeFont(f, true);
string koreanText = "korean.txt";

//Read the text from text file
StreamReader reader = new StreamReader(koreanText, Encoding.Unicode);
string text = reader.ReadToEnd();
reader.Close();

page.Graphics.DrawString(text, font, brush, PointF.Empty);
```

5.1.1.10 How To Set Graphic Units?

Essential PDF sets the size of an element in terms of points [1/72 inch]. It has a utility class, **PdfUnitConvertor**, which enables to convert different measure units and use them for resizing pages.

[C#]

```
// Setting Page size after converting units to points
PdfUnitConvertor con = new PdfUnitConvertor();
doc.PageSettings.Width = con.ConvertUnits(100f, PdfGraphicsUnit.Millimeter,
PdfGraphicsUnit.Point);
```

[VB .NET]

```
' Setting Page size after converting units to points
Dim con As New PdfUnitConvertor()
doc.PageSettings.Width = con.ConvertUnits(100F, PdfGraphicsUnit.Millimeter,
PdfGraphicsUnit.Point)
```



Note: The width or height property of the document is always represented by using Points [1/72 inch]. However, the helper method enables you to set page sizes in the desired unit.

5.1.1.11 How To Dispose The Pdf document Object?

You can dispose the Pdf object by using the **Close** method. Note that without closing this object, it is not possible to use the same document again for any other manipulation.

Close method releases the commonly used memory. Its overload with its parameter set to true [Close(true)], releases the entire document stream, enabling it to be reused.

[C#]

```
// Release the common resources.  
pdfDoc.Close();  
  
// (or)  
  
// Releases document stream. This releases the entire document.  
PdfDoc.Close(true);
```

[VB.NET]

```
' Release the common resources.  
pdfDoc.Close()  
  
' (or)  
  
' Releases document stream. This releases the entire document.  
PdfDoc.Close(True)
```

5.1.1.12 How to open the generated PDF document into the browser instead of displaying the Open/Save dialog in browser?

You can open the PDF document directly in the browser with the help of the Response object.

Follow the below steps to generate the PDF document inline:

1. Save the PDF document as Stream object.
2. Write the stream in the Response object.

[C#]

```
//Save the PDFDocument as a Stream.  
  
MemoryStream stream = new MemoryStream();
```

```

doc.Save(stream);

//Stream the output to the browser.

Response.ContentType = "application/pdf";

Response.AddHeader("content-disposition", "inline;
filename=MyPDF.PDF");

Response.AddHeader("content-length",
stream.Length.ToString());

Response.BinaryWrite(stream.ToArray());

Response.End();

```

[VB]

```

Save the PDFDocument as a Stream.

Dim stream As New MemoryStream()

doc.Save(stream)

'Stream the output to the browser.

Response.ContentType = "application/pdf"

Response.AddHeader("content-disposition", "inline;
filename=MyPDF.PDF")

Response.AddHeader("content-length",
stream.Length.ToString())

Response.BinaryWrite(stream.ToArray())

Response.End()

```

5.1.1.13 How to set the default view of Navigation Pane in Viewer?

When a PDF document is opened in Adobe, by default, any one of the tabs in the Navigation pane can be expanded. It can be set using the PdfPageMode enumeration in Essential PDF. The following code snippet when executed, make Attachments as a default view.

[C#]

```
PdfDocument document = new PdfDocument();  
document.ViewerPreferences.PageMode = PdfPageMode.UseAttachments;
```

[VB.NET]

```
Dim document As New PdfDocument()  
  
document.ViewerPreferences.PageMode = PdfPageMode.UseAttachments
```

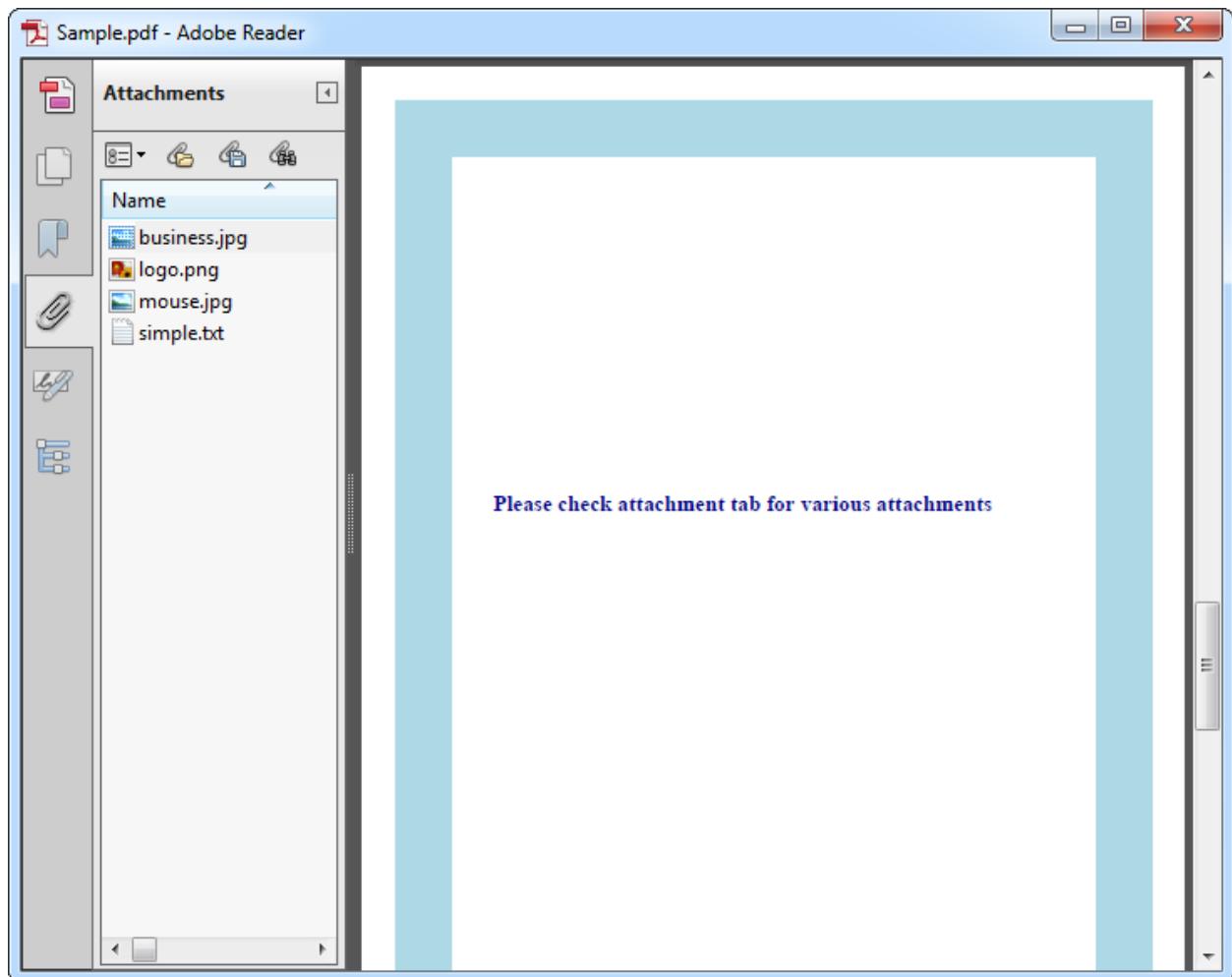


Figure 66: PDF with default view

5.1.2 Advanced

This section shows some advanced tasks in a PDF Generator.

5.1.2.1 How To Add the Tables One After Another?

You can draw the table relative to the position of the previous table by using the **PdfLayoutResult** class. This class stores the boundary values of the table that is drawn. By using the boundary values, you can set the starting position of the table relative to the height of the previous table. The following code example illustrates this.

[C#]

```
// Drawing first table.
PdfLightTable table = new PdfLightTable();
table.DataSource = dt;
table.Style.ShowHeader = true;
PdfLayoutResult result = table.Draw(page, new PointF(0, 20));

// Calculating Second table position.
RectangleF bounds = result.Bounds;
PointF location = new PointF(bounds.Left, bounds.Bottom + 30);

// Drawing the second table.
table.Draw(page, location);
```

[VB .NET]

```
' Drawing first table.
Dim table As PdfLightTable = New PdfLightTable()
table.DataSource = dt
table.Style.ShowHeader = True
Dim result As PdfLayoutResult = table.Draw(page, New PointF(0, 20))

' Calculating Second table position.
Dim bounds As RectangleF = result.Bounds
Dim location As PointF = New PointF(bounds.Left, bounds.Bottom + 30)

' Drawing the second table.
table.Draw(page, location)
```

5.1.2.2 How To Draw an Image In a Table Cell?

You can draw an image in a particular table cell by using the **BeginCellLayout** event handler and its arguments. This is done by using the **Graphics** object in the event handler.

The following code example illustrates how to draw a borderless table.

[C#]

```
// Adding the Event handler
table.BeginCellLayout += new
BeginCellLayoutEventHandler(table_BeginCellLayout);

// Drawing an image in a cell
void table_BeginCellLayout(object sender, BeginCellLayoutEventArgs args)
{
    if (args.RowIndex == 0 && args.CellIndex == 0)
    {
        PdfImage img = new PdfBitmap(Image.FromFile(@"..\..\Data\Image.png"));
        args.Graphics.DrawImage(img, args.Bounds);

        // To dispose the image
        (img as PdfBitmap).Dispose();
    }
}
```

[VB .NET]

```
' Adding the Event handler
Private table.BeginCellLayout += New BeginCellLayoutEventHandler(AddressOf
table_BeginCellLayout)

' Drawing an image in a cell

Private Sub table_BeginCellLayout(ByVal sender As Object, ByVal args As
BeginCellLayoutEventArgs)
    If args.RowIndex = 0 AndAlso args.CellIndex = 0 Then
        Dim img As PdfImage = New
        PdfBitmap(Image.FromFile("../..\..\Data\Image.png"))
        args.Graphics.DrawImage(img, args.Bounds)

        ' To dispose the image
        TryCast(img, PdfBitmap)).Dispose()
    End If
End Sub
```

5.1.2.3 How To Implement Column Span In PdfLightTable?

You can span any number of columns with the help of the **BeginRowLayout** event handler and **ColumnSpan** property. ColumnSpanMap property of this event enables column spanning, which merges cells within a row. To specify the span, you should create an array of integers with size equal to the number of cells in a row, and assign it to the **ColumnSpanMap** property.

The description of the map values are as follows

- 0 or 1: denotes an ordinary cell; 0 allows you to omit explicit specification of 1
- Negatives: denotes that the cell is covered by a merged cell; you should not specify those values
- 2 and more: denotes how many cells should be merged

The following code example illustrates this.

[C#]

```
void table_BeginRowLayout(object sender, BeginRowLayoutEventArgs args)
{
    if (args.RowIndex == 1)
    {
        PdfLightTable table = (PdfLightTable)sender;
        int count = table.Columns.Count;

        int[] spanMap = new int[count];

        // Set just spanned cells. Other values are not important
        // Except negatives that are not allowed.
        spanMap[0] = 2;
        spanMap[2] = 3;

        args.ColumnSpanMap = spanMap;

        //Sets row height.
        args.MinimalHeight = 30f;
    }
}
```

[VB .NET]

```
Private Sub table_BeginRowLayout(ByVal sender As Object, ByVal args As
BeginRowLayoutEventArgs)
    If args.RowIndex = 1 Then
        Dim table As PdfLightTable = CType(sender, PdfLightTable)
        Dim count As Integer = table.Columns.Count
```

```

Dim spanMap As Integer() = New Integer(count - 1) {}

    ' Set just spanned cells. Other values are not important
    ' Except negatives that are not allowed.
    spanMap(0) = 2
    spanMap(2) = 3

    args.ColumnSpanMap = spanMap

    'Sets row height.
    args.MinimalHeight = 30f
End If
End Sub

```

5.1.2.4 How To Improve Performance?

Performance is the one of the initial requirements of Essential PDF. This section describes performance tips that help using the product in the most appropriate way.

Text and Font

- It is good to use PdfStandardFont during text output than Unicode characters. Standard fonts take less space in the file than other fonts.
- If PdfTrueType font is used, and it is not expected to use Unicode characters, switching off the Unicode support will reduce the size of the output PDF file. Support of Unicode by PdfTrueTypeFont is specified in the constructor.
- Do not enable RightToLeft property of PdfStringFormat, if right-to-left language is not used. Setting this property to true enables a special RTL engine that decreases the text layouting speed.

Compression and File Structure

Changing these can change the resulting file size, however it can increase the time spent on saving.

Compression is responsible for compressing internal data streams in PDF. You might be surprised by the number of streams in an ordinary PDF document. High quality compression involves longer file saves, but requires less space.

If **CrossReferenceType** property is specified by CrossReferenceStream, the cross-reference is represented by cross-reference stream. It may reduce the file size, especially if compression is turned on, but increases the file generation time.

5.1.2.5 How To Insert a Table In The Header?

PDFLightTable model of Essential PDF provides an option to insert tables in headers by using page templates.

The following steps illustrate how to insert tables in the header:

1. Create a table with some datasource by using the PdfLightTable class.
2. Create a Page template where the table has to be placed.
3. Draw the table in the template.

You can also insert images and other graphical objects in the table cell and render it in the header.

```
[C#]

//Create page template
PdfPageTemplateElement top = new PdfPageTemplateElement(rect);

//Create a table
PdfLightTable table = new PdfLightTable();
table.DataSource = dataTable;
table.Style.CellPadding = 16;

//Draw the table in template
table.Draw(top.Graphics);

//Place the template at the top.
doc.Template.Top = top;
```

[VB .NET]

```
'Create page template
Dim top As PdfPageTemplateElement = New PdfPageTemplateElement(rect)

'Create a table
Dim table As PdfLightTable = New PdfLightTable()
table.DataSource = dataTable
table.Style.CellPadding = 16

'Draw the table in template
table.Draw(top.Graphics)

'Place the template at the top.
doc.Template.Top = top
```

5.1.2.6 How To Measure the String Whose End Position Is Not Known?

PdfFont provides the **MeasureString** method which determines the rectangle that a string should occupy on a PDF page. This information is used for relative positioning of several paragraphs of text.

[C#]

```
// Create a font.
PdfStandardFont font = new PdfStandardFont(PdfFontFamily.Symbol, 12,
PdfFontStyle.Bold);

// Measure the size of the text based on string format and font.
SizeF textSize = pdfFont.MeasureString(text, rect.Size, format);

// Draw the rectangle for the size of the text.
page.Graphics.DrawRectangle(PdfPens.Red, new RectangleF(rect.Location,
textSize));
```

[VB .NET]

```
' Create a font.
Dim font As PdfStandardFont = New PdfStandardFont(PdfFontFamily.Symbol, 12,
PdfFontStyle.Bold)

' Measure the size of the text based on string format and font.
Dim textSize As SizeF = pdfFont.MeasureString(text, rect.Size, format)
```

```
' Draw the rectangle for the size of the text.
page.Graphics.DrawRectangle(PdfPens.Red, New RectangleF(rect.Location,
textSize))
```

5.1.2.7 How to set margins for the PDF pages?

Margins can be set on all sides or on a particular side of the page using the **PageSettings.Margins** property. Following are the options of this property:

- **All**-Sets margins size on all the sides
- **Bottom**-Sets the bottom margin size
- **Left**-Sets the left margin size
- **Top**-Sets the top margin size
- **Right**-Sets the right margin size

[C#]

```
//Adds a section to the document
PdfSection section = doc.Sections.Add();

//Adds a page to the section
PdfPage page= section.Pages.Add();

//Sets Margin to the page
section.PageSettings.Margins.All = 0;
```

[VB.NET]

```
'Adds a section to the document
Dim section As PdfSection = doc.Sections.Add()

'Adds a page to the section
PdfPage = section.Pages.Add()

'Sets Margin to the page
section.PageSettings.Margins.All = 0
```

5.1.2.8 How to edit header in PdfGrid?

When data source is set to PdfGrid, captions from source will automatically add as header. This header can be edited to display any custom text.

Note: *Editing text will affect only the PdfGrid and will not reflect in data source.*

The following is the code snippet:

[C#]

```
// Edit cell value in header.
pdfGrid.Headers[0].Cells[0].Value = "Employee ID";
```

[VB.NET]

```
' Edit cell value in header.
pdfGrid.Headers(0).Cells(0).Value = "Employee ID"
```

5.1.2.9 How to set width for Table?

By default, both PdfLightTable and PdfGrid classes automatically calculate width if one is not specified during Draw. However, it is possible to specify width using one of the overloads in Draw method. The following is the code snippet.

[C#]

```
// Draw PdfLightTable specified width.
pdfLightTable.Draw(pdfGraphics, PointF.Empty, width);
pdfLightTable.Draw(pdfGraphics, xPos, yPos, width);
pdfLightTable.Draw(pdfPage, xPos, yPos, width);
pdfLightTable.Draw(pdfPage, xPos, yPos, width,
pdfLightTableLayoutFormat);

// Draw PdfGrid with specified width.
pdfGrid.Draw(pdfGraphics, PointF.Empty, width);
pdfGrid.Draw(pdfGraphics, xPos, yPos, width);
pdfGrid.Draw(pdfPage, xPos, yPos, width);
pdfGrid.Draw(pdfPage, xPos, yPos, width, pdfGridLayoutFormat);
```

[VB.NET]

```
' Draw PdfLightTable specified width.
pdfLightTable.Draw(pdfGraphics, PointF.Empty, width)
```

```

pdfLightTable.Draw(pdfGraphics, xPos, yPos, width)
pdfLightTable.Draw(pdfPage, xPos, yPos, width)
pdfLightTable.Draw(pdfPage, xPos, yPos, width,
pdfLightTableLayoutFormat)

' Draw PdfGrid with specified width.
pdfGrid.Draw(pdfGraphics, PointF.Empty, width)
pdfGrid.Draw(pdfGraphics, xPos, yPos, width)
pdfGrid.Draw(pdfPage, xPos, yPos, width)
pdfGrid.Draw(pdfPage, xPos, yPos, width, pdfGridLayoutFormat)

```

5.1.2.10 How to specify bounds for Table during pagination?

When PdfLightTable or PdfGrid is paginated, by default it will occupy the entire client area of the PdfPage. But, it is possible to specify bounds for the additional pages using PaginateBounds property of PdfLightTableLayoutFormat or PdfGridLayoutFormat class.

[C#]

```
format.PaginateBounds = new RectangleF(50, 50, 500, 300);
```

[VB.NET]

```
format.PaginateBounds = New RectangleF(50, 50, 500, 300)
```

5.2 PDF Editing

This section shows some specific tasks that are supported in a PDF document editing.

5.2.1 How To Access Pages In an Existing Document?

Pages in the existing document are different from pages in the newly created document. PdfPageBase class is used to manipulate the existing page in a loaded document. The following code example illustrates how to access the existing page.

[C#]

```

PdfLoadedDocument lDoc = new PdfLoadedDocument(txtUrl.Text);
PdfPageBase lPage = lDoc.Pages[0];

```

```

PdfGraphics g = lPage.Graphics;
g.DrawString("Writing text in loaded page", font, PdfPens.Red,
PdfBrushes.Red, new PointF(-150, 450));
lDoc.Save("Sample.pdf");

```

[VB.NET]

```

Dim lDoc As PdfLoadedDocument = New PdfLoadedDocument(txtUrl.Text)
Dim lPage As PdfPageBase = lDoc.Pages(0)
Dim g As PdfGraphics = lPage.Graphics
g.DrawString("Writing text in loaded
page", font, PdfPens.Red, PdfBrushes.Red, New PointF(-150, 450))
lDoc.Save("Sample.pdf")

```

5.2.2 How To Add Watermarks Or Stamps In an Existing Document?

You can add watermarks or stamps in an existing document by adding transparent images or text on the pages. The following code example illustrates this.

[C#]

```

PdfLoadedDocument lDoc = new PdfLoadedDocument(txtUrl.Text);
PdfFont font = new PdfStandardFont(PdfFontFamily.Helvetica, 36f);

foreach (PdfPageBase lPage in lDoc.Pages)
{
    PdfGraphics g = lPage.Graphics;
    PdfGraphicsState state = g.Save();
    g.SetTransparency(0.25f);
    g.RotateTransform(-40);
    g.DrawString("Stamping text", font, PdfPens.Red, PdfBrushes.Red, new
    PointF(-150, 450));
}
lDoc.Save("Sample.pdf");

```

[VB.NET]

```

Dim lDoc As PdfLoadedDocument = New PdfLoadedDocument(txtUrl.Text)
Dim font As PdfFont = New PdfStandardFont(PdfFontFamily.Helvetica, 36.0F)

Dim lPage As PdfPageBase
For Each lPage In lDoc.Pages

```

```

Dim g As PdfGraphics = lPage.Graphics
Dim state As PdfGraphicsState = g.Save()
    g.SetTransparency(0.25f)
    g.RotateTransform(-40)
    g.DrawString("Stamping text", font, PdfPens.Red, PdfBrushes.Red, New
PointF(-150, 450))
Next
lDoc.Save("Sample.pdf")

```

5.2.3 How To Convert Units In PDF / What Are the Units Of the Elements In PDF?

Essential PDF measure unit of the elements are "points". A point is equal to 1/72 of an "inch". Points are represented in terms of float values. **PdfUnitConvertor** enables to convert different measure units.

The following code example illustrates how to convert pixels to points.

[C#]

```

//Converts inches to points
float height = con.ConvertUnits(800, PdfGraphicsUnit.Inch,
PdfGraphicsUnit.Point);
float width = con.ConvertUnits(500, PdfGraphicsUnit.Inch,
PdfGraphicsUnit.Point);
SizeF pageSize = new SizeF(width, height);
doc.PageSettings.Size = pageSize;

```

[VB .NET]

```

'Converts inches to points
Dim height As Single = con.ConvertUnits(800, PdfGraphicsUnit.Inch,
PdfGraphicsUnit.Point)
Dim width As Single = con.ConvertUnits(500, PdfGraphicsUnit.Inch,
PdfGraphicsUnit.Point)
Dim pageSize As SizeF = New SizeF(width, height)
doc.PageSettings.Size = pageSize

```

5.2.4 How To Store And Retrieve a PDF Document From the Database?

Essential PDF provides support for reading and writing PDF documents from / to the System.IO.Stream. You can store a PDF document stream as a binary object in the database. The following code example illustrates this.

[C#]

```
// Store the PDF document in Database.
// Initialize a stream.
MemoryStream stream = new MemoryStream();

// Save the document to stream.
doc.Save(stream);

// Retrieve and display the stream in PDFformat.
OleDbDataReader Reader = command.ExecuteReader();
byte[] PdfFile = (byte[])Reader[1];
Stream strm = new MemoryStream(PdfFile);
using (FileStream fstream = new FileStream("sample.pdf",
 FileMode.OpenOrCreate, FileAccess.ReadWrite))
{
    fstream.Write(PdfFile, 0, PdfFile.Length);
}
```

[VB .NET]

```
' Store the PDF document in Database.
' Initialize a stream.
Dim stream As MemoryStream = New MemoryStream()

' Save the document to stream.
doc.Save(stream)

' Retrieve and display the stream in PDFformat.
Dim Reader As OleDbDataReader = Command.ExecuteReader()
Dim PdfFile As Byte() = CType(Reader(1), Byte())
Dim strm As Stream = New MemoryStream(PdfFile)
Using fstream As FileStream = New FileStream("sample.pdf",
 FileMode.OpenOrCreate, FileAccess.ReadWrite)
    fstream.Write(PdfFile, 0, PdfFile.Length)
End Using
```

5.2.5 How To Enable and Disable PDF Layers In a PDF document?

You can enable and disable PDF layers for customization purposes. For more information on enabling and disabling PDF layers, refer to the section [Creating and Embedding PDF Layers in the PDF document](#).

5.2.6 How To Read Conformance Level From PDF?

You can read conformance level applied to a PDF file using **PdfLoadedDocument.Conformance** property. This property will return the levels supported by **PdfConformanceLevel** enum.



Note: It will return none when the PDF file is not applied with any levels.

Following is the code snippet to read conformance level from PDF.

[C#]

```
PdfLoadedDocument loadedDocument = new PdfLoadedDocument(fileName);
PdfConformanceLevel appliedConformance = loadedDocument.Conformance;
```

[VB .NET]

```
Dim loadedDocument As New PdfLoadedDocument(fileName)
Dim appliedConformance As PdfConformanceLevel =
loadedDocument.Conformance
```

5.3 PDF Form

This section shows some specific tasks that are supported in a PDF document creation.

5.3.1 How To Create a Form Which Transfers Data To the Server?

You can create a form and transfer the data to the server by using the **PdfSubmitAction** class.

The following code example illustrates how to create a document with a PDF form in it. When the Submit button is clicked, data is sent to the processing script which displays the received data. You can write your own data processing script and manipulate data as needed.

[C#]

```
PdfDocument document = new PdfDocument();

PdfFont font = new PdfStandardFont(PdfFontFamily.Courier, 12f);
PdfBrush brush = PdfBrushes.Black;

PdfGraphics g = page.Graphics;
g.DrawString("First Name", font, brush, 10, 25);

//Create a text box
PdfTextBoxField firstNameTextBox = new PdfTextBoxField(page,
"firstNameTextBox");

//Set properties
firstNameTextBox.Bounds = new RectangleF(100, 20, 200, 20);
firstNameTextBox.Font = font;
firstNameTextBox.Password = true;
firstNameTextBox.Multiline = true;

//Add the text box in document
document.Form.Fields.Add(firstNameTextBox);
g.DrawString("Last Name", font, brush, 10, 55);

PdfTextBoxField lastNameTextBox = new PdfTextBoxField(page,
"lastNameTextBox");
lastNameTextBox.Bounds = new RectangleF(100, 53, 200, 20);
lastNameTextBox.Font = font;
document.Form.Fields.Add(lastNameTextBox);

g.DrawString("Company", font, brush, 10, 85);

PdfTextBoxField companyTextBox = new PdfTextBoxField(page, "companyTextBox");
companyTextBox.Bounds = new RectangleF(100, 80, 200, 20);
companyTextBox.Font = font;
document.Form.Fields.Add(companyTextBox);
g.DrawString("Position", font, brush, 10, 115);

//Create a combo box
PdfComboBoxField positionComboBox = new PdfComboBoxField(page,
"positionComboBox");

//Set properties
positionComboBox.Bounds = new RectangleF(100, 115, 200, 20);
positionComboBox.Font = font;
positionComboBox.Editable = true;
```

```

//Add it to document
document.Form.Fields.Add(positionComboBox);

//Create the field item to be added in the combobox
PdfListFieldItem item1 = new PdfListFieldItem("Developer", "Developer");
PdfListFieldItem item2 = new PdfListFieldItem("Accountant", "Accountant");
PdfListFieldItem item3 = new PdfListFieldItem("CEO", "CEO");
PdfListFieldItem item4 = new PdfListFieldItem("Sales Manager", "Sales Manager");
PdfListFieldItem item5 = new PdfListFieldItem("Director", "Director");
PdfListFieldItem item6 = new PdfListFieldItem("HR Manager", "HR Manager");

//Add the items in combo box.
positionComboBox.Items.Add(item1);
positionComboBox.Items.Add(item2);
positionComboBox.Items.Add(item3);
positionComboBox.Items.Add(item4);
positionComboBox.Items.Add(item5);
positionComboBox.Items.Add(item6);

g.DrawString("Number of Employees", font, brush, new RectangleF(10, 145, 80, 60));

//Create a Radio button
PdfRadioButtonListField employeesRadioList = new PdfRadioButtonListField(page, "employeesRadioList");

//Add to document
document.Form.Fields.Add(employeesRadioList);

//Create radio button items
PdfRadioButtonListItem radioItem1 = new PdfRadioButtonListItem("1-9");
radioItem1.Bounds = new RectangleF(100, 140, 20, 20);
g.DrawString("1-9", font, brush, new RectangleF(150, 145, 180, 20));

PdfRadioButtonListItem radioItem2 = new PdfRadioButtonListItem("10-49");
radioItem2.Bounds = new RectangleF(100, 170, 20, 20);
g.DrawString("10-49", font, brush, new RectangleF(150, 175, 180, 20));

PdfRadioButtonListItem radioItem3 = new PdfRadioButtonListItem("50-99");
radioItem3.Bounds = new RectangleF(100, 200, 20, 20);
g.DrawString("50-99", font, brush, new RectangleF(150, 205, 180, 20));

PdfRadioButtonListItem radioItem4 = new PdfRadioButtonListItem("100-499");
radioItem4.Bounds = new RectangleF(100, 230, 20, 20);
g.DrawString("100-499", font, brush, new RectangleF(150, 235, 180, 20));

```

```
PdfRadioButtonListItem radioItem5 = new PdfRadioButtonListItem("500-more");
radioItem5.Bounds = new RectangleF(100, 260, 20, 20);
g.DrawString("500-more", font, brush, new RectangleF(150, 265, 180, 20));

//add the items to radio button group
employeesRadioList.Items.Add(radioItem1);
employeesRadioList.Items.Add(radioItem2);
employeesRadioList.Items.Add(radioItem3);
employeesRadioList.Items.Add(radioItem4);
employeesRadioList.Items.Add(radioItem5);

g.DrawString("Platform", font, brush, new RectangleF(10, 290, 80, 60));

//Create a check box
PdfCheckBoxField checkBox = new PdfCheckBoxField(page, ".NET");

//Set properties
checkBox.Bounds = new RectangleF(100, 290, 20, 20);
document.Form.Fields.Add(checkBox);

checkBox.HighlightMode = PdfHighlightMode.Push;
checkBox.BorderStyle = PdfBorderStyle.Beveled;

//Set the value for the check box
checkBox.Checked =true;
g.DrawString(".NET", font, brush, new RectangleF(150, 290, 180, 20));

checkBox = new PdfCheckBoxField(page, "Java");
checkBox.Bounds = new RectangleF(100, 320, 20, 20);
document.Form.Fields.Add(checkBox);
checkBox.HighlightMode = PdfHighlightMode.Push;
checkBox.BorderStyle = PdfBorderStyle.Beveled;
checkBox.Checked = false;
g.DrawString("Java", font, brush, new RectangleF(150, 320, 180, 20));
g.DrawString("Language known:", font, brush, new RectangleF(10, 350, 80, 60));

//Create list box
PdfListBoxField listBox = new PdfListBoxField(page, "list1");

//Add the field to list box.
document.Form.Fields.Add(listBox);

//Set the properties.
listBox.Bounds = new RectangleF(100, 350, 100, 50);
```

```

listBox.HighlightMode = PdfHighlightMode.Outline;

//Add the items to the list box
listBox.Items.Add(new PdfListFieldItem("English", "English"));
listBox.Items.Add(new PdfListFieldItem("French", "French"));
listBox.Items.Add(new PdfListFieldItem("German", "German"));

//Select the item
listBox.SelectedIndex = 2;

//Set the multiselect option
listBox.MultiSelect = true;

PdfSubmitAction submitAction = new
PdfSubmitAction("http://stevex.net/dump.php");
submitAction.DataFormat = SubmitDataFormat.Html;

PdfResetAction resetAction = new PdfResetAction();

//Create submit button to transfer the values in the form
PdfButtonField submitButton = new PdfButtonField(page, "submitButton");
submitButton.Bounds = new RectangleF(100, 420, 90, 20);
submitButton.Font = font;
submitButton.Text = "Submit";
submitButton.Actions.MouseUp = submitAction;
document.Form.Fields.Add(submitButton);

//Create reset button to reset all the values
PdfButtonField resetButton = new PdfButtonField(page, "resetButton");
resetButton.Bounds = new RectangleF(210, 420, 90, 20);
resetButton.Font = font;
resetButton.Text = "Reset";
resetButton.Actions.MouseUp = resetAction;
document.Form.Fields.Add(resetButton);

```

[VB]

```

Dim document As PdfDocument = New PdfDocument()

Dim font As PdfFont = New PdfStandardFont(PdfFontFamily.Courier, 12f)
Dim brush As PdfBrush = PdfBrushes.Black

Dim g As PdfGraphics = page.Graphics
g.DrawString("First Name", font, brush, 10, 25)

'Create a text box

```

```

Dim firstNameTextBox As PdfTextBoxField = New PdfTextBoxField(page,
"firstNameTextBox")

' Set properties
firstNameTextBox.Bounds = New RectangleF(100, 20, 200, 20)
firstNameTextBox.Font = font
firstNameTextBox.Password = True
firstNameTextBox.Multiline = True

' Add the text box in document
document.Form.Fields.Add(firstNameTextBox)
g.DrawString("Last Name", font, brush, 10, 55)

Dim lastNameTextBox As PdfTextBoxField = New PdfTextBoxField(page,
"lastNameTextBox")
lastNameTextBox.Bounds = New RectangleF(100, 53, 200, 20)
lastNameTextBox.Font = font
document.Form.Fields.Add(lastNameTextBox)

g.DrawString("Company", font, brush, 10, 85)

Dim companyTextBox As PdfTextBoxField = New PdfTextBoxField(page,
"companyTextBox")
companyTextBox.Bounds = New RectangleF(100, 80, 200, 20)
companyTextBox.Font = font
document.Form.Fields.Add(companyTextBox)
g.DrawString("Position", font, brush, 10, 115)

' Create a combo box
Dim positionComboBox As PdfComboBoxField = New PdfComboBoxField(page,
"positionComboBox")

' Set properties
positionComboBox.Bounds = New RectangleF(100, 115, 200, 20)
positionComboBox.Font = font
positionComboBox.Editable = True

' Add it to document
document.Form.Fields.Add(positionComboBox)

' Create the field item to be added in the combo box
Dim item1 As PdfListFieldItem = New PdfListFieldItem("Developer",
"Developer")
Dim item2 As PdfListFieldItem = New PdfListFieldItem("Accountant",
"Accountant")
Dim item3 As PdfListFieldItem = New PdfListFieldItem("CEO", "CEO")
Dim item4 As PdfListFieldItem = New PdfListFieldItem("Sales Manager", "Sales

```

```

Manager")
Dim item5 As PdfListFieldItem = New PdfListFieldItem("Director", "Director")
Dim item6 As PdfListFieldItem = New PdfListFieldItem("HR Manager", "HR
Manager")

'Add the items in combo box.
positionComboBox.Items.Add(item1)
positionComboBox.Items.Add(item2)
positionComboBox.Items.Add(item3)
positionComboBox.Items.Add(item4)
positionComboBox.Items.Add(item5)
positionComboBox.Items.Add(item6)

g.DrawString("Number of Employees", font, brush, New RectangleF(10, 145, 80,
60))

'Create a Radio button
Dim employeesRadioList As PdfRadioButtonListField = New
PdfRadioButtonListField(page, "employeesRadioList")

'Add to document
document.Form.Fields.Add(employeesRadioList)

'Create radio button items
Dim radioItem1 As PdfRadioButtonListItem = New PdfRadioButtonListItem("1-9")
radioItem1.Bounds = New RectangleF(100, 140, 20, 20)
g.DrawString("1-9", font, brush, New RectangleF(150, 145, 180, 20))

Dim radioItem2 As PdfRadioButtonListItem = New PdfRadioButtonListItem("10-
49")
radioItem2.Bounds = New RectangleF(100, 170, 20, 20)
g.DrawString("10-49", font, brush, New RectangleF(150, 175, 180, 20))

Dim radioItem3 As PdfRadioButtonListItem = New PdfRadioButtonListItem("50-
99")
radioItem3.Bounds = New RectangleF(100, 200, 20, 20)
g.DrawString("50-99", font, brush, New RectangleF(150, 205, 180, 20))

Dim radioItem4 As PdfRadioButtonListItem = New PdfRadioButtonListItem("100-
499")
radioItem4.Bounds = New RectangleF(100, 230, 20, 20)
g.DrawString("100-499", font, brush, New RectangleF(150, 235, 180, 20))

Dim radioItem5 As PdfRadioButtonListItem = New PdfRadioButtonListItem("500-
more")
radioItem5.Bounds = New RectangleF(100, 260, 20, 20)
g.DrawString("500-more", font, brush, New RectangleF(150, 265, 180, 20))

```

```

'add the items to radio button group
employeesRadioList.Items.Add(radioItem1)
employeesRadioList.Items.Add(radioItem2)
employeesRadioList.Items.Add(radioItem3)
employeesRadioList.Items.Add(radioItem4)
employeesRadioList.Items.Add(radioItem5)

g.DrawString("Platform", font, brush, New RectangleF(10, 290, 80, 60))

'Create a check box
Dim checkBox As PdfCheckBoxField = New PdfCheckBoxField(page, ".NET")

'Set properties
checkBox.Bounds = New RectangleF(100, 290, 20, 20)
document.Form.Fields.Add(checkBox)

checkBox.HighlightMode = PdfHighlightMode.Push
checkBox.BorderStyle = PdfBorderStyle.Beveled

'Set the value for the check box
checkBox.Checked =True
g.DrawString(".NET", font, brush, New RectangleF(150, 290, 180, 20))

checkBox = New PdfCheckBoxField(page, "Java")
checkBox.Bounds = New RectangleF(100, 320, 20, 20)
document.Form.Fields.Add(checkBox)
checkBox.HighlightMode = PdfHighlightMode.Push
checkBox.BorderStyle = PdfBorderStyle.Beveled
checkBox.Checked = False
g.DrawString("Java", font, brush, New RectangleF(150, 320, 180, 20))
g.DrawString("Language known:", font, brush, New RectangleF(10, 350, 80, 60))

'Create list box
Dim listBox As PdfListBoxField = New PdfListBoxField(page, "list1")

'Add the field to listbox.
document.Form.Fields.Add(listBox)

'Set the properties.
listBox.Bounds = New RectangleF(100, 350, 100, 50)
listBox.HighlightMode = PdfHighlightMode.Outline

'Add the items to the list box
listBox.Items.Add(New PdfListFieldItem("English", "English"))
listBox.Items.Add(New PdfListFieldItem("French", "French"))

```

```

listBox.Items.Add(New PdfListFieldItem("German", "German"))

'Select the item
listBox.SelectedIndex = 2

'Set the multiselect option
listBox.MultiSelect = True

Dim submitAction As PdfSubmitAction = New
PdfSubmitAction("http://stevex.net/dump.php")
submitAction.DataFormat = SubmitDataFormat.Html

Dim resetAction As PdfResetAction = New PdfResetAction()

'Create submit button to transfer the values in the form
Dim submitButton As PdfButtonField = New PdfButtonField(page, "submitButton")
submitButton.Bounds = New RectangleF(100, 420, 90, 20)
submitButton.Font = font
submitButton.Text = "Submit"
submitButton.Actions.MouseUp = submitAction
document.Form.Fields.Add(submitButton)

'Create reset button to reset all the values
Dim resetButton As PdfButtonField = New PdfButtonField(page, "resetButton")
resetButton.Bounds = New RectangleF(210, 420, 90, 20)
resetButton.Font = font
resetButton.Text = "Reset"
resetButton.Actions.MouseUp = resetAction
document.Form.Fields.Add(resetButton)

```

5.4 PDF Convertor

This section shows some specific tasks that are supported in a PDF document creation.

5.4.1 How To Convert Secure Webpages?

The **HtmlConverter** class provides support to access secured webpages. To access secured webpages, you have to pass the username and password information as arguments to the **ConvertToImage** method. The following code example illustrates this.

[C#]

```
// Convert web page into image.
HtmlConverter html = new HtmlConverter()
System.Drawing.Image img = html.ConvertToImage("www.syncfusion.com",
ImageType.Metafile, (int)width, -1,
AspectRatio.KeepWidth,"UserName","Password");

// Draw the image into the PDF document as metafile
PdfMetafile metafile = (PdfMetafile)(PdfImage.FromImage(img));
metafile.Quality = 100;
PdfMetafileLayoutFormat format = new PdfMetafileLayoutFormat();
format.Break = PdfLayoutBreakType.FitPage;
format.Layout = PdfLayoutType.Paginate;
doc.PageSettings.Height = img.Height;
format.SplitTextLines = false;
metafile.Draw(page, new RectangleF(0F, 0F, (float)(pageSize.Width), -1F),
format);

// Draw the image into the PDF document as Bitmap
PdfImage image = new PdfBitmap(img);
PdfLayoutFormat format = new PdfLayoutFormat();
format.Break = PdfLayoutBreakType.FitPage;
format.Layout = PdfLayoutType.Paginate;
image.Draw(page, new RectangleF(0F, 0F, (float)(img.Width),
(float)(img.Height)), format);
```

[VB.NET]

```
' Convert web page into image.
Dim html As HtmlConverter = New HtmlConverter()
Dim img As System.Drawing.Image = html.ConvertToImage("www.syncfusion.com",
ImageType.Metafile, CInt(Fix(Width)), -1, AspectRatio.KeepWidth, "UserName",
>Password")

' Draw the image into the PDF document as metafile
Dim metafile As Syncfusion.Pdf.Graphics.PdfMetafile =
 CType(PdfImage.FromImage(img), Syncfusion.Pdf.Graphics.PdfMetafile)
metafile.Quality = 100
Dim format As Syncfusion.Pdf.Graphics.PdfMetafileLayoutFormat = New
Syncfusion.Pdf.Graphics.PdfMetafileLayoutFormat()
format.Break = PdfLayoutBreakType.FitPage
format.Layout = PdfLayoutType.Paginate
doc.PageSettings.Height = img.Height
format.SplitTextLines = False
metafile.Draw(page, New RectangleF(0F, 0F, CSng(pageSize.Width), -1F),
```

```

format)

' Draw the image into the PDF document as Bitmap
Dim image As Syncfusion.Pdf.Graphics.PdfImage = New
Syncfusion.Pdf.Graphics.PdfBitmap(img)
Dim format As Syncfusion.Pdf.Graphics.PdfLayoutFormat = New
PdfLayoutFormat()
format.Break = PdfLayoutBreakType.FitPage
format.Layout = PdfLayoutType.Paginate
image.Draw(page, New RectangleF(0F, 0F, CSng(img.Width), CSng(img.Height)),
format)

```

5.4.2 How To Extract Images From an Existing PDF Document?

You can extract images from an existing PDF document page by page, using the **ExtractImages** method of the **PdfLoadedPage** class.

The following code example illustrates how to extract images from a document.

[C#]

```

// Load an existing PDF
PdfLoadedDocument ldoc = new PdfLoadedDocument("Sample.pdf");

// Loading Page collections
PdfLoadedPageCollection loadedPages = ldoc.Pages;

// Extract Image from PDF document pages
foreach (PdfLoadedPage lpage in loadedPages)
{
    Image[] img = lpage.ExtractImages();
    foreach (Image img1 in img)
    {
        img1.Save("Image" + Guid.NewGuid().ToString() + ".png",
ImageFormat.Png);
    }
}

```

[VB .NET]

```

' Load an existing PDF
Dim ldoc As PdfLoadedDocument = New PdfLoadedDocument("Sample.pdf")

' Loading Page collections
Dim loadedPages As PdfLoadedPageCollection = ldoc.Pages

```

```
' Extract Image from PDF document pages
For Each lpage As PdfLoadedPage In loadedPages
    Dim img As Image() = lpage.ExtractImages()
    For Each img1 As Image In img
        img1.Save("Image" & Guid.NewGuid().ToString() & ".png",
ImageFormat.Png)
    Next img1
Next lpage
```

5.4.3 How To Extract Text From an Existing PDF Document?

You can extract text from an existing PDF document page by page by using the **ExtractText** method of the **PdfLoadedPage** class.

The following code example illustrates how to extract text from a document.

[C#]

```
// Load an existing PDF
PdfLoadedDocument ldoc = new PdfLoadedDocument(txtUrl.Text);

// Loading Page collections
PdfLoadedPageCollection loadedPages = ldoc.Pages;
string pdftext = "";

// Extract text from PDF document pages
foreach (PdfLoadedPage lpage in loadedPages)
{
    pdftext += lpage.ExtractText();
}
```

[VB.NET]

```
' Load an existing PDF
Dim ldoc As PdfLoadedDocument = New PdfLoadedDocument(txtUrl.Text)

' Loading Page collections
Dim loadedPages As PdfLoadedPageCollection = ldoc.Pages
Dim pdftext As String = ""

' Extract text from PDF document pages
For Each lpage As PdfLoadedPage In loadedPages
    pdftext &= lpage.ExtractText()
```

[Next](#) | [1page](#)

5.4.4 Register the Syncfusion Gecko Wrapper manually?

To register the Gecko wrapper manually:

1. Copy the **Syncfusion.GeckoWrapper.dll** to the **Bin** folder of **XulRunner-SDK 2.0**.
2. Register the **Syncfusion.GeckoWrapper.dll** using the following command in the command prompt:
regsvr32 Syncfusion.GeckoWrapper.dll

5.4.5 How can we make IE9 to render metafile?

Webpages can be converted to searchable metafile format using IE9 if the following registry value is set to (DWORD) 00000001.

HKEY_LOCAL_MACHINE (or HKEY_CURRENT_USER)\SOFTWARE\Microsoft\Internet Explorer\MAIN\FeatureControl\FEATURE_IVIEWOBJECTDRAW_DMLT9_WITH_GDI



Note: This registry setting will not be effective for webpages displayed in IE9 Standards mode.

5.5 PDF Grid

5.5.1 How to position the background image as a tile?

We can position the background image as a tile within the PDF Grid, by setting the property “ImagePosition” to PdfGridImagePosition.Tile

```
grid.Rows(1).Cells(3).ImagePosition =  
PdfGridImagePosition.Tile
```

For more information on positioning background images, refer to the section [Background Image Positioning](#) in [PdfGrid Cell](#).

5.5.2 How to make background images centered, stretched and fitted within the PDF Grid?

Background image positioning such as stretch, center and fit within the PDF Grid can be done by setting the property “ImagePosition” to PdfGridImagePosition.Stretch, PdfGridImagePosition.Centre and PdfGridImagePosition.Fit respectively.

```
grid.Rows(1).Cells(0).ImagePosition =  
    PdfGridImagePosition.Stretch  
  
grid.Rows(1).Cells(1).ImagePosition =  
    PdfGridImagePosition.Centre  
  
grid.Rows(1).Cells(2).ImagePosition =  
    PdfGridImagePosition.Fit
```

For more information on positioning background images, refer to the section **Background Image Positioning** in [PdfGrid Cell](#).

Index

3

3D Annotation 172

A

Actions 168

Adding a New Page 261

Adding a QR Barcode to an application 119

Adding an Attachment 263

Adding Custom File Fields and Attributes into a PDF Portfolio 189

Advanced 354

Annotation 170

ASP.NET 38

ASP.NET MVC 50

Asynchronous Support 337

Attachments 183

Automatic Fields 85

B

Barcode 107

BeginCellLayout 145

BeginPageLayout 143

BeginRowLayout 143

Booklet 279

Bookmark 256

C

Cell 141, 161

Class Diagram 28

ColorSpace 95

Column 140, 160

Common 342

Compression 209

Concepts and Features 61

Conversion of HTML to PDF using Gecko Rendering Engine 323

Creating a PDF Portfolio 186

Creating and Embedding PDF Layers in the PDF Document 69

Creating Platform Application 30

Custom Metadata 206

D

Data 130, 149

Deploying Essential PDF 34

Deployment Requirements 25

DLLs 26, 283

Doc To PDF 328

Document Information 278

Document Object Model 61

Document Settings 204

Documentation 11

Draw Rich text 216

Draw Right-To-Left Text 218

Draw Simple text 211

Drawing 79

Drawing Images 223

Drawing Shapes 220

Drawing Table 230

Drawing Text 80, 211

Dynamic Fields 265

E

Encryption 193

EndCellLayout 146

EndPageLayout 143

EndRowLayout 144

Error correction level 118

Event Handlers 133

External DataSource 130

F

Fonts 77
Form Fields 297
Form Filling 308
Free Text Annotation 180
Frequently Asked Questions 342

G

Getting Started 28
Graphic Elements 94
Graphics 90

H

Header 139, 156
Headers and Footers 250
How can we make IE9 to render metafile? 379
How To Access Pages In an Existing Document? 363
How To Add Sections And Pages? 342
How To Add the Tables One After Another? 355
How To Add Watermarks Or Stamps In an Existing Document? 364
How To Compress a PDF Document? 343
How To Convert Secure Webpages? 375
How To Convert Units In PDF / What Are the Units Of the Elements In PDF? 365
How To Create a Borderless Table? 343
How To Create a Form Which Transfers Data To the Server? 367
How To Create Page Labels? 344
How To Dispose The Pdf document Object? 352
How To Draw a SoftMask Image Into a PDF Document? 345
How To Draw an Image In a Table Cell? 355
How to edit header in PdfGrid? 362
How To Embed 3D Files In a PDF Document? 345
How To Embed Fonts? 348
How To Enable and Disable PDF Layers In a PDF document? 366

How To Extract Images From an Existing PDF Document? 377
How To Extract Text From an Existing PDF Document? 378
How To Find the End Position Of a Table? 349
How To Implement Column Span In PdfLightTable? 356
How To Improve Performance? 358
How To Insert a Table In The Header? 359
How to make background images centered, stretched and fitted within the PDF Grid? 380
How To Measure the String Whose End Position Is Not Known? 360
How to open the generated PDF document into the browser instead of displaying the Open/Save dialog in browser? 352
How to position the background image as a tile? 379
How To Read Conformance Level From PDF? 367
How To Render Unicode Text [CJK Fonts] In a PDF Document? 350
How To Set Graphic Units? 351
How to set margins for the PDF pages? 361
How to set the default view of Navigation Pane in Viewer? 353
How to set width for Table? 362
How to specify bounds for Table during pagination? 363
How To Store And Retrieve a PDF Document From the Database? 365
Html Styled Text 83
HTML To PDF 316
HTML to PDF Conversion using Gecko Rendering Engine 322
Hyperlinks for other external files 179
I
IEnumarable 132
ImageExtraction 327
Images 103

- Import Pages 271
- Import Pages As Templates 289
- Importing 335
- Input mode 119
- Installation 14
- Installation and Deployment 14
- Installation Steps 323
- Interactive Features 167
- Introduction 61
- Introduction to Essential PDF 8
- L**
 - Layout 135, 151
 - Links 89
 - Lists 165
 - Load Document 253
- M**
 - Merge PDF 275, 287
 - Methods 118
- O**
 - OCR Support 282
 - Overview 8, 293
- P**
 - Page Settings 246
 - Page Templates 123
 - Pagination 120
 - PDF Convertor 316, 375
 - PDF Editing 253, 363
 - PDF Features 57
 - PDF Form 292, 367
 - PDF Generator 61, 342
 - PDF Grid 379
 - PDF Layers 69
 - PDF Version Compatibility 56
 - PDF/A-1b 200
 - PDF/X-1a 203
- PdfGrid 146
- PdfGrid Creation 148
- PdfGrid Formatting 153
- PdfLightTable 126
- PdfLightTable Creation 129
- PdfLightTable Customization 143
- PdfLightTable Formatting 137
- Pen and Brush 75
- Perform OCR for a Specific Region of PDF Document 285
- Performing OCR for a Complete PDF Document 284
- Portfolio 185
- Prerequisites 283, 323
- Prerequisites and Compatibility 10
- Properties 117, 189
- Properties Tables 69
- Properties, Methods and Events 127, 146
- Q**
 - QR Barcode 117
- R**
 - Register the Syncfusion Gecko Wrapper manually? 379
 - Replacing Images 273
 - Row 140, 158
- S**
 - Samples Location 14
 - Security 193
 - Security Settings 234
 - Settings 204
 - Signature 289
 - Signing 195
 - Silverlight 46
 - Split PDF 275
 - Standards for PDF/A-1 Compliance 200
 - Stream Support 233

Supported Elements 338

T

Table Direct 131

Tables 125

Tables for Properties and Methods 282

Tagged PDF 324

Text 80

Text Extraction 326

Text Pagination 213

Transform PDF 277

Tutorial 210, 287, 335

U

Use Case Scenario 69, 323

Use Case Scenarios 117

V

Version 118

Viewing Layers Samples 71

W

Web Application deployment 26

Web Deployment 284

Windows 34

Windows Deployment 283

WPF 42

X

XPS to PDF 332