ComponentOne

PDF for .NET

GrapeCity US

GrapeCity
201 South Highland Avenue, Suite 301
Pittsburgh, PA 15206
Tel: 1.800.858.2739 | 412.681.4343
Fax: 412.681.4384
Website: https://www.grapecity.com/en/
E-mail: us.sales@grapecity.com

Trademarks

The ComponentOne product name is a trademark and ComponentOne is a registered trademark of GrapeCity, Inc. All other trademarks used herein are the properties of their respective owners.

Warranty

ComponentOne warrants that the media on which the software is delivered is free from defects in material and workmanship, assuming normal use, for a period of 90 days from the date of purchase. If a defect occurs during this time, you may return the defective media to ComponentOne, along with a dated proof of purchase, and ComponentOne will replace it at no charge. After 90 days, you can obtain a replacement for the defective media by sending it and a check for $2 5 (to cover postage and handling) to ComponentOne.

Except for the express warranty of the original media on which the software is delivered is set forth here, ComponentOne makes no other warranties, express or implied. Every attempt has been made to ensure that the information contained in this manual is correct as of the time it was written. ComponentOne is not responsible for any errors or omissions. ComponentOne's liability is limited to the amount you paid for the product. ComponentOne is not liable for any special, consequential, or other damages for any reason.

Copying and Distribution

While you are welcome to make backup copies of the software for your own use and protection, you are not permitted to make copies for the use of anyone else. We put a lot of time and effort into creating this product, and we appreciate your support in seeing that it is used by licensed users only.

Table of Contents

PDF for .NET Overview 2
Help with WinForms Edition 2
Key Features 3-4
Using ComponentOne PDF for .NET 5
Creating Documents 5-7
Adding Text 7
Drawing Text 7-8
Measuring Text 8-10
Making Text Flow from Page to Page 10
Drawing RTF Text 10-11
Adding Images 11-13
Adding Graphics 13-15
Creating Pages and Overlays 15-17
Adding Bookmarks and Annotations 17
Adding Bookmarks to a PDF Document 17-19
Adding Links to a PDF Document 19-20
Attaching Files to a PDF Document 20-22
Applying Security and Permissions 22-23
Using Metafiles 23-24
Rendering RTF Text 24-27
PDF for .NET Samples 28-29
PDF for .NET Task-Based Help 30
Setting Document Properties 30-33
Setting Document Restrictions 33-36
Setting Paper Size 36
Setting Landscape Page Orientation 36-37
Setting Image Quality 37-38
Printing PDF Files 38
Writing PDF for .NET Content in Web Applications 38-39
Using PDF for .NET with Other ComponentOne Tools 39-40
Adding ComponentOne FlexGrid for WinForms Data into PDF for .NET 40
Creating a PDF from a Report Rendered by VSPrinter 40-41
Removing PDF Bookmarks from a C1Report Document 41-42

Copyright © 2018 GrapeCity, Inc. All rights reserved.

PDF for .NET Overview

PDF for .NET is a library that allows you to create PDF documents from your applications. The PDF (Portable Document Format) format was created by Adobe in 1993 and has since become the industry standard for the representation of printed material in electronic systems. PDF is popular because it is high-quality, portable, device-independent, and because there are many tools available for creating, manipulating, and viewing PDF documents. These tools include the free multi-platform document viewer Adobe Acrobat. The portability and high quality of the PDF documents make PDF an excellent choice for Web publishing. For more information on the PDF format and PDF utilities from Adobe and other sources, visit the Adobe Web site (www.adobe.com).	Getting Started To get started, review the following topics: • Key Features • Creating Documents • Samples

Help with WinForms Edition

Getting Started

For information on installing **ComponentOne Studio WinForms Edition**, licensing, technical support, namespaces and creating a project with the control, please visit Getting Started with WinForms Edition.

Copyright © 2018 GrapeCity, Inc. All rights reserved.

Key Features

PDF for .NET supports most of the advanced features included in the PDF specification, including security, compression, outlining, hyperlinking, and attachments.

But the main feature in **PDF for .NET** is ease of use. The commands provided for adding content to documents are similar to the ones available in the .NET **Graphics** class. If you know how to display text and graphics in .NET, you already know how to use **PDF for .NET**.

The following are some of the features of **PDF for .NET** that you may find useful:

- **Fast rendering and compression of images in Metafiles**
  Metafiles are not converted into bitmaps; they are parsed and converted into vector graphics commands and thus retain the best possible resolution. If you want to add charts or technical drawings to your PDF document, metafiles are better than bitmap images.
- **AcroForms support**
  Use the new AddField method to add Acrobat form fields to your PDF documents. PDF for .NET supports the following field types: textbox, checkbox, radio button, push button, combo box, and list box.
- **HTML Support**
  Use the new DrawStringHtml method to render HTML into your PDF documents. You can flow HTML content into multiple pages or columns, use existing style sheets, and mix HTML with other types of content (images, RTF, plain text, form fields, and so on).
- **Familiar syntax using the DrawImage method**
  Adding images to PDF documents is easy; all the work is done by the DrawImage method. DrawImage draws a given image at a specified location and has parameters that provide control over the image alignment and scaling. You can render any regular .NET Image object, including metafiles.
- **Manage document restrictions**
  Allow users to copy and edit content, restrict users from printing the document, set annotation edit permission to the user, and more.
- **Add attachments to PDF files**
  Attachments can contain any kind of file, including spreadsheets with detailed information that would clutter the main document, multimedia files with movies and sound, sample code, and more. Adding an attachment to your PDF file is easy. Simply specify which file you want to attach, what area of the page should contain the attachment, and optionally, the appearance of the attachment.
- **Owner and user password protection**
  If your PDF documents contain sensitive information, you can encrypt them so that only authorized users can access it. There is a separate password for the owner of the document and for all other users. The user's access can be selectively restricted to allow only certain operations, such as viewing, printing, or editing the document.
- **Add graphical elements**
  Add lines, rectangles, ellipses, pies, arcs, rounded rectangles, polygons, Bezier curves, and more.
- **Create an Outline structure**
  Most long PDF documents contain an outline structure that is displayed on a pane on the left of the reader. The outline makes it easy to browse through a document's structure and find specific topics. With PDF for .NET, you can build this outline structure by adding outline entries (bookmarks).
- **Add hyperlinks and local links**
  PDF provides methods for adding hyperlinks and hyperlink targets to your PDF documents. You can also add local links, that when clicked take the user to another location within the same PDF document. This type of link is useful when you want to implement some type of cross-referencing within the document, such as a table of contents or an index.
- **Control document information and viewer preferences**
  PDF allows you to add meta data to the PDF documents you create. Specify author, creation date, keywords, and so on. You can also provide default viewer preferences to be applied when the document is opened in the Adobe Reader. Specify the initial page layout, window position, as well as reader toolbar and menu visibility.
- **Support for PDF/A**

Copyright © 2018 GrapeCity, Inc. All rights reserved.

PDF/A is commonly used by users creating invoices, brochures, manuals or research reports to store their reports to PDF/A formats. It enables export of JPEG2000 Images, provisions for digital signatures, and support for embedded fonts.

Copyright © 2018 GrapeCity, Inc. All rights reserved.

Using ComponentOne PDF for .NET

The following topics provide details on creating documents using **PDF for .NET**, as well as how to add text, images, graphics, pages and overlays, bookmarks and annotations, and security and permissions to **PDF for .NET** documents.

Creating Documents

To create PDF documents using **PDF for .NET**, you simply create a C1PdfDocument object, add content to the document, and save the document. To follow tradition, here's how to create a "hello world" document using **PDF for .NET**.

1. Add the following **Imports** or **using** directive to your code.

Visual Basic

```
Imports System.Drawing
```

C#

```
using System.Drawing;
```

2. Create a C1PdfDocument object.

Visual Basic

```
' Create the C1PdfDocument object.
Dim pdf As New C1.C1Pdf.C1PdfDocument()
```

C#

```
// Create the C1PdfDocument object.
C1.C1Pdf.C1PdfDocument pdf = new C1.C1Pdf.C1PdfDocument();
```

3. Add your content to the **Form_Load** event. This usually involves calling the DrawString method.

Visual Basic

```
' Add content to the page.
Dim rect As RectangleF =  pdf.PageRectangle
rect.Inflate(-72, -72)
Dim font As Font =  New Font("Arial",12)
pdf.DrawString("Hello World!", font, Brushes.Black, rect)
```

C#

```
// Add content to the page.
RectangleF rect = pdf.PageRectangle;
rect.Inflate(-72, -72);
Font font = new Font("Arial", 12);
```

Copyright © 2018 GrapeCity, Inc. All rights reserved.

```
pdf.DrawString("Hello World!", font, Brushes.Black, rect);
```

4. Save the document to a file or to a stream using the **Save** method.

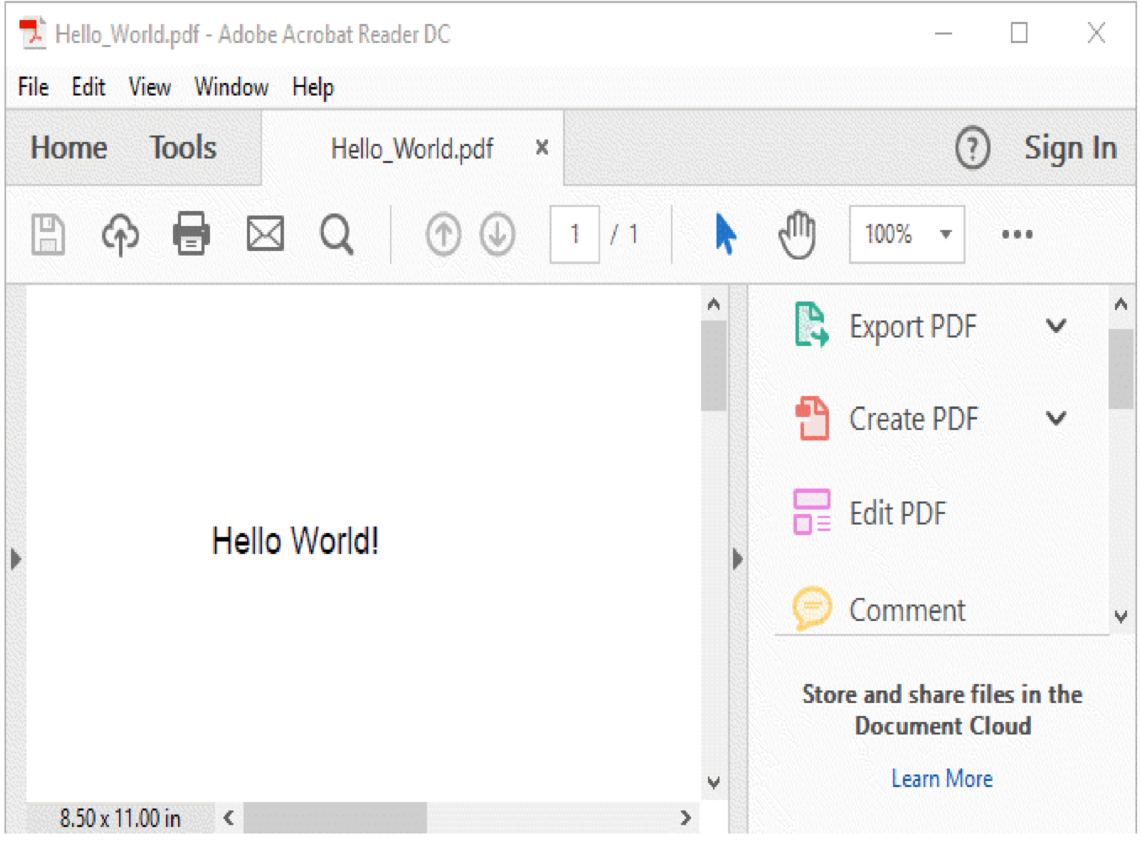
Visual Basic

```
' Save the document to a file.
pdf.Save(@"C:\PDF\Hello_World.pdf")
```

C#

```
// Save the document to a file.
pdf.Save(@"C:\PDF\Hello_World.pdf");
```

After the application is run, the hello world.pdf document will look like this:



Step 3 is the most interesting one. The code starts by retrieving the page rectangle, expressed in points. It then adds a one-inch margin around the page (72 points). Finally, the code creates a **Font** object and calls the DrawString method to write "Hello World!" on the page. This is exactly what you would do if you were writing to a **Graphics** object in .NET and is what makes **PDF for .NET** so easy to use.

One important thing to remember is that **PDF for .NET** uses a point-based coordinate system with the origin at the top-left corner of the page. This is similar to the default coordinate system used by .NET, but is different from the default PDF coordinate system (where the origin is on the bottom-left corner of the page). In this example, the top left point of the "H" in "Hello World" is located at [1,1].

Because the coordinate system is based on points, rather than pixels, **PDF for .NET** uses **RectangleF**, **SizeF**, and **PointF** structures, which have members of type **float**, rather than **Rectangle**, **Size**, and **Point**, which have members of

Copyright © 2018 GrapeCity, Inc. All rights reserved.

type **int**.

Adding Text

The following topics provide information on drawing, measuring, and managing the flow of text, as well as drawing RTF text.

Drawing Text

Adding text to **PDF for .NET** documents is easy – all the work is done by the DrawString method.

DrawString draws a given string at a specified location using a given font and brush. For example:

To write code in Visual Basic

Visual Basic

```
pdf.DrawString("Hello World!", font, Brushes.Black, rect)
```

To write code in C#

C#

```
pdf.DrawString("Hello World!", font, Brushes.Black, rect);
```

By default, DrawString will align the text to the left and to the top of the given rectangle, will wrap the string within the rectangle, and will not clip the output to the rectangle. You can change all these options by specifying a *StringFormat* parameter in the call to DrawString. The **StringFormat** has members that allow you to specify the horizontal alignment (**Alignment**), vertical alignment (**LineAlignmnent**), and flags that control wrapping and clipping (**FormatFlags**).

For example, the code below creates a **StringFormat** object and uses it to align the text to the center of the rectangle, both vertically and horizontally:

To write code in Visual Basic

Visual Basic

```
Dim font As New Font("Arial", 12)
Dim rect As New RectangleF(72, 72, 100, 50)
Dim text As String = "Some long string to be rendered into a small rectangle. "
text = text & text & text & text & text & text
 ' Center align string.
Dim sf As New StringFormat()
sf.Alignment = StringAlignment.Center
sf.LineAlignment = StringAlignment.Center
pdf.DrawString(text, font, Brushes.Black, rect, sf)
pdf.DrawRectangle(Pens.Gray, rect)
```

To write code in C#

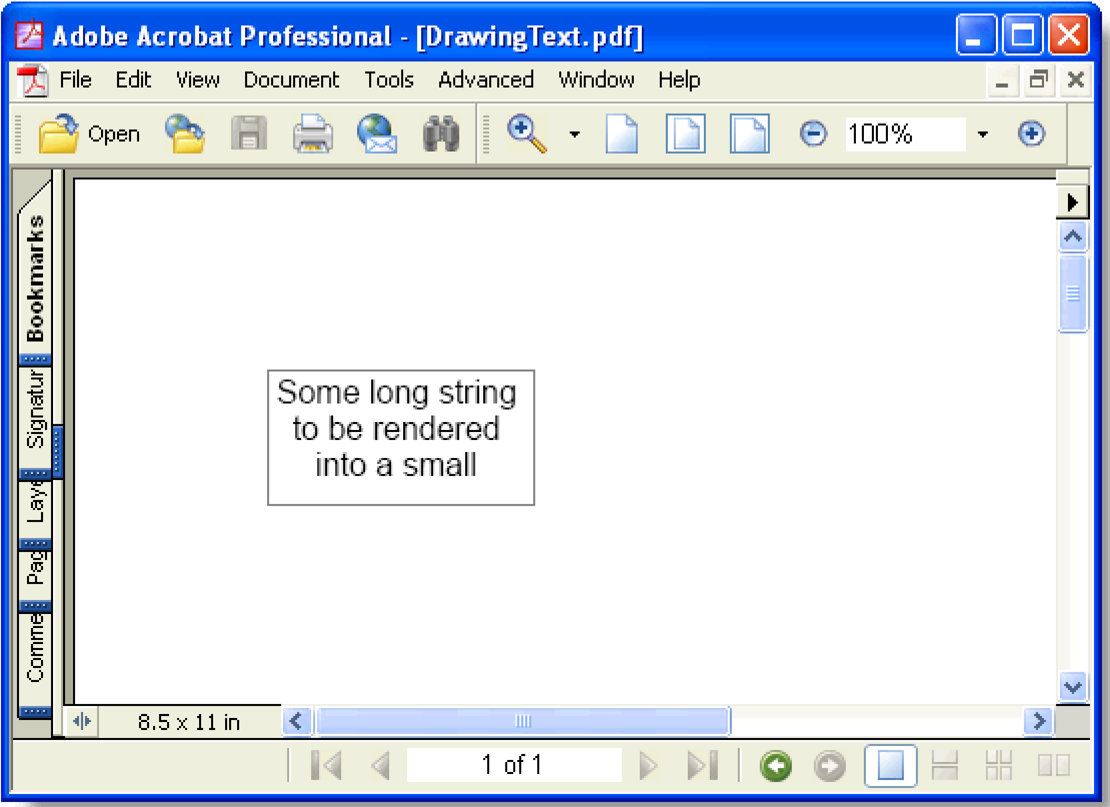
C#

```
Font font = new Font("Arial", 12);
RectangleF rect = new RectangleF(72, 72, 100, 50);
```

Copyright © 2018 GrapeCity, Inc. All rights reserved.

```
string text = "Some long string to be rendered into a small rectangle. ";
text = text + text + text + text + text + text;
 // Center align string.
StringFormat sf = new StringFormat();
sf.Alignment = StringAlignment.Center;
sf.LineAlignment = StringAlignment.Center;
pdf.DrawString(text, font, Brushes.Black, rect, sf);
pdf.DrawRectangle(Pens.Gray, rect);
```

Here is the resulting PDF document:



Measuring Text

In many cases, you will need to check whether the string will fit on the page before you render it. You can use the MeasureString method for that. MeasureString returns a **SizeF** structure that contains the width and height of the string (in points) when rendered with a given font.

For example, the code below checks to see if a paragraph will fit on the current page and creates a page break if it has to. This will keep paragraphs together on a page:

To write code in Visual Basic

Visual Basic

```
Private Function RenderParagraph(text As String, font As Font, rect As RectangleF,
rectPage As RectangleF) As RectangleF
      ' Calculate the necessary height.
    Dim sz As SizeF = _c1pdf.MeasureString(text, font, rect.Width)
```

Copyright © 2018 GrapeCity, Inc. All rights reserved.

```
    rect.Height = sz.Height
      ' If it won't fit this page, do a page break.
    If rect.Bottom > rectPage.Bottom Then
        _c1pdf.NewPage()
        rect.Y = rectPage.Top
    End If
      ' Draw the string.
    _c1pdf.DrawString(text, font, Brushes.Black, rect)
      ' Update rectangle for next time.
    Rect.Offset(0, rect.Height)
    Return rect
End Function
  ' Use the RenderParagraph method.
Dim font As New Font("Arial", 10)
Dim rectPage As RectangleF = _c1pdf.PageRectangle
rectPage.Inflate(-72, -72)
Dim rect As RectangleF = rectPage
Dim s As String
For Each s In  myStringList
    rect = RenderParagraph(s, font, rect, rectPage)
Next s
```

To write code in C#

C#

```
private RectangleF RenderParagraph(string text, Font font, RectangleF rect,
RectangleF rectPage)
{
// Calculate the necessary height.
    SizeF sz = _c1pdf.MeasureString(text, font, rect.Width);
    rect.Height = sz.Height;
     // If it won't fit this page, do a page break.
    If (rect.Bottom > rectPage.Bottom)
    {
        _c1pdf.NewPage();
        rect.Y = rectPage.Top;
    }
     // Draw the string.
    _c1pdf.DrawString(text, font, Brushes.Black, rect);
     // Update rectangle for next time.
    Rect.Offset(0, rect.Height);
    return rect;
}
 // Use the RenderParagraph method.
Font font = new Font("Arial", 10);
RectangleF rectPage = _c1pdf.PageRectangle;
rectPage.Inflate(-72, -72);
RectangleF rect = rectPage;
foreach (string s in myStringList)
{
    rect = RenderParagraph(s, font, rect, rectPage);
```

Copyright © 2018 GrapeCity, Inc. All rights reserved.

```
}
```

Making Text Flow from Page to Page

The DrawString method returns an integer. This is the index of the first character that was not printed because it did not fit the output rectangle. You can use this value make text flow from page to page or from one frame to another within a page. For example:

To write code in Visual Basic

Visual Basic

```
' Render a string spanning multiple pages.
While True
     ' Render as much as will fit into the rectangle.
    Dim nextChar As Integer
    nextChar = _c1pdf.DrawString(text, font, Brushes.Black, rectPage)
     ' Break when done.
    If nextChar >= text.Length Then
        Exit While
    End If
     ' Get rid of the part that was rendered.
    Text = text.Substring(nextChar)
     ' Move on to the next page.
    _c1pdf.NewPage()
End While
```

To write code in C#

C#

```
// Render a string spanning multiple pages.
While (true)
{
    // Render as much as will fit into the rectangle.
    Int nextChar = _c1pdf.DrawString(text, font, Brushes.Black, rectPage);
     // Break when done.
    If (nextChar >= text.Length)
    {
        break;
    }
     // Get rid of the part that was rendered.
    Text = text.Substring(nextChar);
     // Move on to the next page.
    _c1pdf.NewPage();
}
```

By combining the MeasureString and DrawString methods, you can develop rendering routines that provide extensive control over how paragraphs are rendered, including keeping paragraphs together on a page, keeping with the next paragraph, and controlling widows and orphans (single lines that render on the current or next page).

Copyright © 2018 GrapeCity, Inc. All rights reserved.

Drawing RTF Text

DrawString provides all the functionality you need for rendering paragraphs using a single font and color.

If you want to render paragraphs with rich format, mixing fonts and colors, you should use the DrawStringRtf method instead. DrawStringRtf is similar to DrawString, except the string is interpreted as RTF.

For example, the code below renders a line of text with some bold and italic characters in it:

To write code in Visual Basic

Visual Basic

```
Dim font As New Font("Arial", 12)
Dim rect As RectangleF = pdf.PageRectangle
rect.Inflate(- 72, - 72)
Dim s As String
s = "To {\b boldly} go where {\i no one} has gone before!"
pdf.DrawStringRtf(s, font, Brushes.Black, rect)
```

To write code in C#

C#

```
Font font = new Font("Arial", 12);
RectangleF rect = pdf.PageRectangle;
rect.Inflate(-72, -72);
pdf.DrawStringRtf(@"To {\b boldly} go where {\i no one} has gone before!", font,
Brushes.Black, rect);
```

Notice that the string does not contain a full RTF document. It has no \rtf1 header, font or color tables. The DrawStringRtf method recognizes that and builds the necessary RTF header automatically, based on the *Font* and *Brush* parameters provided.

If the string contained a complete RTF header, that would be used in lieu of the *font* and *brush* parameters specified in the call DrawStringRtf. This would be the case if the string were copied from a **RichTextBox** control or loaded from an RTF file.

Adding Images

Adding images to **PDF for .NET** documents is also easy, all the work is done by the DrawImage method.

DrawImage draws a given image at a specified location and has parameters that provide control over the image alignment and scaling. In the following example, this image is:

- Stretched to fill the given rectangle
- Center-aligned within the rectangle, scaled to keep the aspect ratio
- Aligned to the top-left corner of the rectangle, with the original size

This code is used to draw the same image three times:

To write code in Visual Basic

Visual Basic

```
Dim rect As RectangleF = pdf.PageRectangle
rect.Inflate(- 72, - 72)
```

Copyright © 2018 GrapeCity, Inc. All rights reserved.

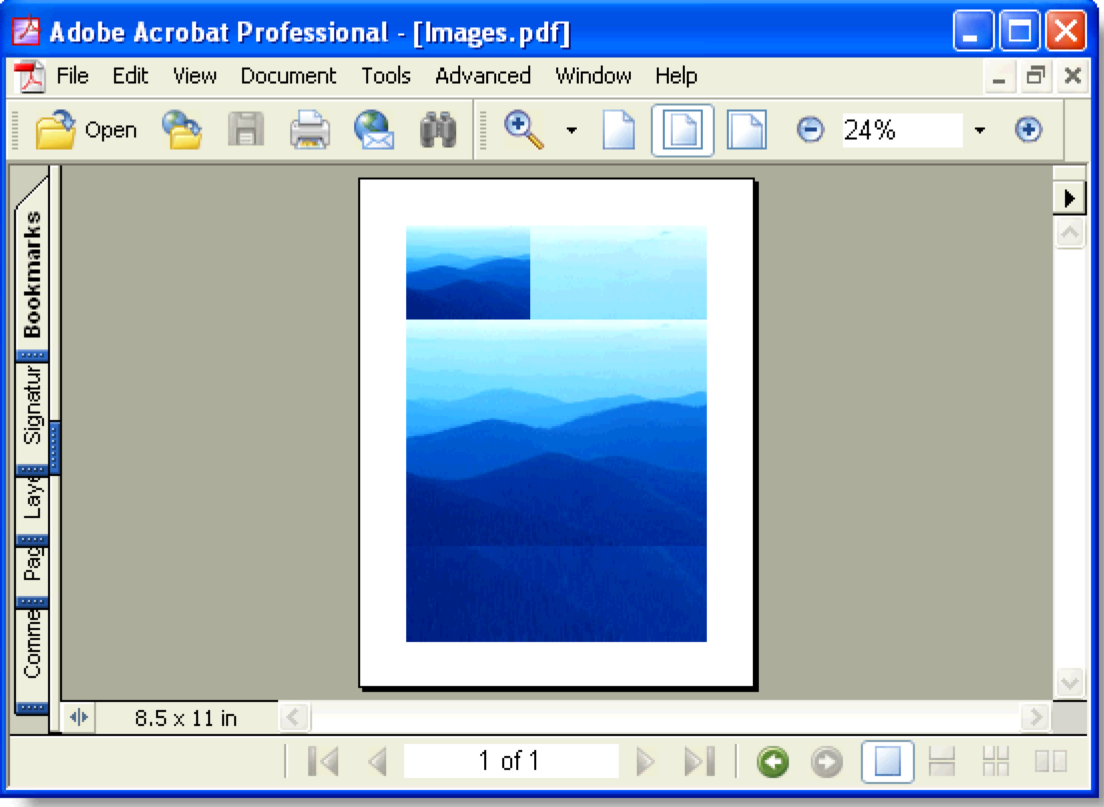
```
 ' Stretch image to fill the rectangle.
pdf.DrawImage(pictureBox1.Image, rect)
 ' Center image within the rectangle, scale keeping aspect ratio.
pdf.DrawImage(pictureBox1.Image, rect, ContentAlignment.MiddleCenter,
C1.C1Pdf.ImageSizeModeEnum.Scale)
 ' Render the image to the top left corner of the rectangle.
pdf.DrawImage(pictureBox1.Image, rect, ContentAlignment.TopLeft,
C1.C1Pdf.ImageSizeModeEnum.Clip)
```

To write code in C#

C#

```
RectangleF rect = pdf.PageRectangle;
rect.Inflate(-72, -72);
 // Stretch image to fill the rectangle.
pdf.DrawImage(pictureBox1.Image, rect);
 // Center image within the rectangle, scale keeping aspect ratio.
pdf.DrawImage(pictureBox1.Image, rect, ContentAlignment.MiddleCenter,
C1.C1Pdf.ImageSizeModeEnum.Scale);
 // Render the image to the top left corner of the rectangle.
pdf.DrawImage(pictureBox1.Image, rect, ContentAlignment.TopLeft,
1.C1Pdf.ImageSizeModeEnum.Clip);
```

The PDF document will look similar to this:



Notice that you can render any regular .NET Image object, including Metafiles. Metafiles are not converted into bitmaps; they are played into the document and thus retain the best possible resolution. If you want to add charts or

Copyright © 2018 GrapeCity, Inc. All rights reserved.

technical drawings to your PDF document, Metafiles are better than bitmap images. **PDF for .NET** also supports EMF+ metafiles.

Bitmap images are managed automatically by **PDF for .NET**. If you render the same image several times (in a page header for example), only one copy of the image is saved into the PDF file. You can control the quality of the images using the **C1PdfDocumentBase.ImageQuality** property, which allows you to trade-off image quality versus file size.

Adding Graphics

The C1PdfDocument class exposes several methods that allow you to add graphical elements to your documents, including lines, rectangles, ellipses, pies, arcs, rounded rectangles, polygons, Bezier curves, and so on.

The methods are a subset of those found in the .NET **Graphics** class, and use the same **Brush** and **Pen** classes to control the color and style of the lines and filled areas.

It is important to remember that **PDF for .NET** uses a coordinate system based on points, with the origin located at the top left of the page. (The default coordinate system for the .NET **Graphics** class is pixel-based.)

The example below illustrates how similar the graphics methods are between **PDF for .NET** and the .NET **Graphics** class. The sample declares a C1PdfDocument class called 'g' and calls methods to draw pies, splines, and other graphical elements.

The point of the sample is that if you replaced the C1PdfDocument class with a regular .NET **Graphics** object, you would be able to compile the code and get the same results:

To write code in Visual Basic

Visual Basic

```
' Create PDF document.
Dim g As New C1.C1Pdf.C1PdfDocument()
 ' Set up to draw.
Dim rect As New RectangleF(0, 0, 300, 200)
Dim text As String = "Hello world of .NET Graphics and PDF." + ControlChars.Cr +
ControlChars.Lf + "Nice to meet you."
Dim font As New Font("Times New Roman", 12, FontStyle.Italic Or FontStyle.Underline)
Dim bezierPoints() As PointF = {New PointF(10F, 100F), New PointF(20F, 10F), New
PointF(35F, 50F), New PointF(50F, 100F), New PointF(60F, 150F), New PointF(65F,
100F), New PointF(50F, 50F)}
 ' Draw some pie slices.
Dim penWidth As Integer = 0
Dim penRGB As Integer = 0
g.FillPie(Brushes.Red, rect, 0, 20F)
g.FillPie(Brushes.Green, rect, 20F, 30F)
g.FillPie(Brushes.Blue, rect, 60F, 12F)
g.FillPie(Brushes.Gold, rect, - 80F, - 20F)
 ' Draw some arcs.
Dim startAngle As Single
For startAngle = 0 To 360 - 40 Step 40
    Dim penColor As Color = Color.FromArgb(penRGB, penRGB, penRGB)
    penWidth = penWidth + 1
    Dim pen As New Pen(penColor, penWidth)
    penRGB = penRGB + 20
    g.DrawArc(pen, rect, startAngle, 40F)
Next
 ' Draw a rectangle and some bezier splines.
```

Copyright © 2018 GrapeCity, Inc. All rights reserved.

```
g.DrawRectangle(Pens.Red, rect)
g.DrawBeziers(Pens.Blue, bezierPoints)
g.DrawString(text, font, Brushes.Black, rect)
```

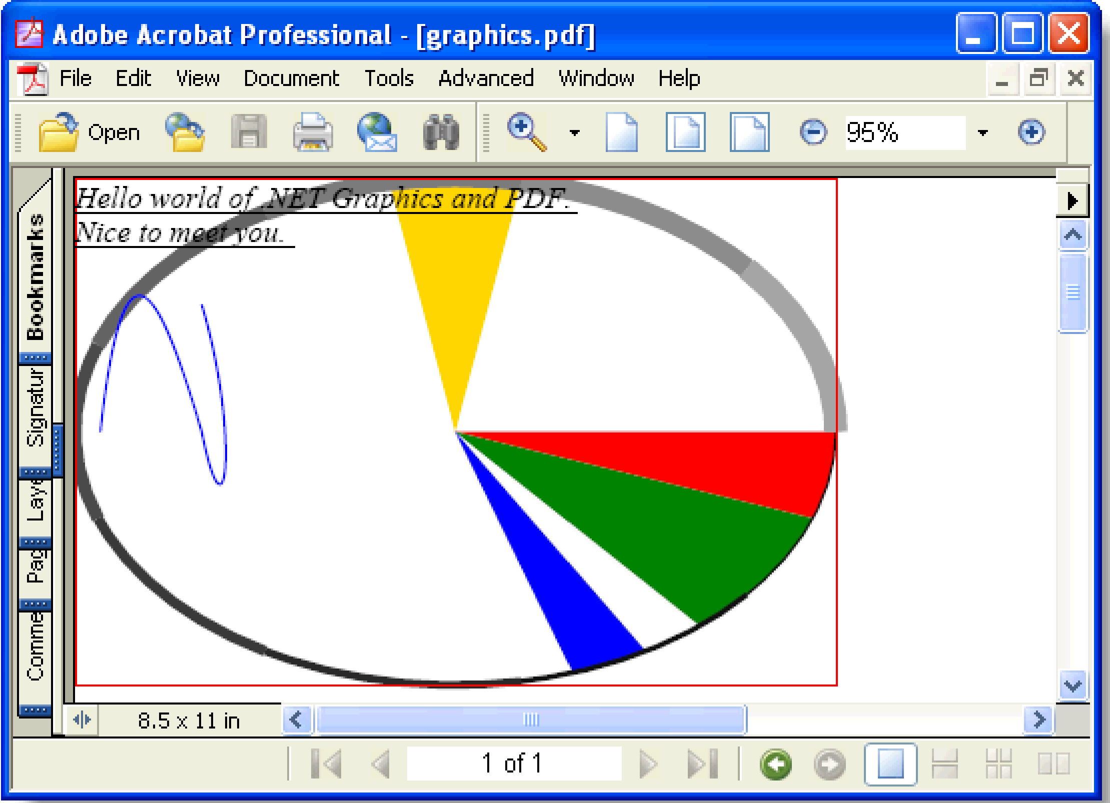
To write code in C#

C#

```
// Create PDF document.
C1.C1Pdf.C1PdfDocument g = new C1.C1Pdf.C1PdfDocument();
 // Set up to draw.
Rectangle rect = new RectangleF(0,0,300,200);
string text = "Hello world of .NET Graphics and PDF.\r\n" + "Nice to meet you.";
Font font = new Font("Times New Roman", 12, FontStyle.Italic | FontStyle.Underline);
PointF[] bezierPoints = new PointF[]
{
    new PointF(10f, 100f), new PointF(20f, 10f), new PointF(35f, 50f), new
PointF(50f, 100f), new PointF(60f, 150f), new PointF(65f, 100f), new PointF(50f, 50f)
};
 // Draw some pie slices.
int penWidth = 0;
int penRGB = 0;
g.FillPie(Brushes.Red, rect, 0, 20f);
g.FillPie(Brushes.Green, rect, 20f, 30f);
g.FillPie(Brushes.Blue, rect, 60f, 12f);
g.FillPie(Brushes.Gold, rect, -80f, -20f);
 // Draw some arcs.
for (float startAngle = 0; startAngle < 360; startAngle += 40)
{
    Color penColor = Color.FromArgb(penRGB, penRGB, penRGB);
    Pen pen = new Pen(penColor, penWidth++);
    penRGB = penRGB + 20;
    g.DrawArc(pen, rect, startAngle, 40f);
}
// Draw a rectangle and some bezier splines.
g.DrawRectangle(Pens.Red, rect);
g.DrawBeziers(Pens.Blue, bezierPoints);
g.DrawString(text, font, Brushes.Black, rect);
```

Here is the resulting PDF document:

Copyright © 2018 GrapeCity, Inc. All rights reserved.



Creating Pages and Overlays

You may have noticed that in the previous examples, we started adding content to the document right after creating the C1PdfDocument object. This is possible because when you create the C1PdfDocument, it automatically adds an empty page to the document, ready to receive any type of content.

When you are done filling up the first page, you can add a new one using the **C1PdfDocumentBase.NewPage** method.

By default, all pages in the document have the same size and orientation. These parameters can be specified in the C1PdfDocument constructor. You can also change the page size and orientation at any time by setting the PaperKind, PageSize, and Landscape properties. For example, the code below creates a document with all paper sizes defined by the **PaperKind** enumeration:

To write code in Visual Basic

Visual Basic

```
Dim font As New Font("Arial", 9)
Dim sf As New StringFormat()
sf.Alignment = StringAlignment.Center
sf.LineAlignment = StringAlignment.Center
 ' Create one page with each paper size.
Dim firstPage As Boolean = True
Dim pk As PaperKind
For Each pk In System.Enum.GetValues(GetType(PaperKind))
     ' Skip custom size.
    If pk = PaperKind.Custom Then
        GoTo ContinueForEach1
```

Copyright © 2018 GrapeCity, Inc. All rights reserved.

```
    End If
     ' Add new page for every page after the first one.
    If Not firstPage Then
        _c1pdf.NewPage()
    End If
    firstPage = False
     ' Set paper kind.
    _c1pdf.PaperKind = pk
     ' Draw some content on the page.
    _c1pdf.DrawString("PaperKind: " + pk.ToString(), font, Brushes.Black,
_c1pdf.PageRectangle, sf)
     ContinueForEach1:
```

To write code in C#

C#

```
Font font = new Font("Arial", 9);
StringFormat sf = new StringFormat();
sf.Alignment = StringAlignment.Center;
sf.LineAlignment = StringAlignment.Center;
 // Create one page with each paper size.
bool firstPage = true;
foreach (PaperKind pk in Enum.GetValues(typeof(PaperKind)))
{
     // Skip custom size.
    if (pk == PaperKind.Custom)
    {
        continue;
    }
    // Add new page for every page after the first one.
    if (!firstPage)
    {
        _c1pdf.NewPage();
    }
    firstPage = false;
     // Set paper kind.
    _c1pdf.PaperKind = pk;
     // Draw some content on the page.
    _c1pdf.DrawString("PaperKind: " + pk.ToString(), font, Brushes.Black,
_c1pdf.PageRectangle, sf);
}
```

You are not restricted to writing on the last page that was added to the document. You can use the CurrentPage property to select which page you want to write to, and then use the regular drawing commands as usual. This is useful for adding content to pages after you are done rendering a document. For example, the code below adds footers to each page containing the current page number and the total of pages in the document (page n of m):

To write code in Visual Basic

Visual Basic

```
Private Sub AddFooters()
    Dim font As New Font("Tahoma", 7, FontStyle.Bold)
```

Copyright © 2018 GrapeCity, Inc. All rights reserved.

```
    Dim sf As New StringFormat()
    sf.Alignment = StringAlignment.Center
    Dim page As Integer
    For page = 0 To _c1pdf.Pages.Count - 1
         ' Select page.
        _c1pdf.CurrentPage = page
         ' Build rectangle for rendering the footer.
        Dim rect As RectangleF = _c1pdf.PageRectangle
        rect.Y = rect.Bottom
         ' Write the footer.
        Dim text As String
        text = String.Format("Page {0} of {1}", page + 1, _c1pdf.Pages.Count)
        _c1pdf.DrawString(text, font, Brushes.Gray, rect, sf)
    Next page
End Sub
```

To write code in C#

C#

```
private void AddFooters()
{
    Font font = new Font("Tahoma", 7, FontStyle.Bold);
    StringFormat sf = new StringFormat();
    sf.Alignment = StringAlignment.Center;
    for (int page = 0; page < _c1pdf.Pages.Count; page++)
    {
        // Select page.
        _c1pdf.CurrentPage = page;
         // Build rectangle for rendering the footer.
        RectangleF rect = _c1pdf.PageRectangle;
        rect.Y = rect.Bottom - 36;
         // Write the footer.
        string text = string.Format("Page {0} of {1}", page+1, _c1pdf.Pages.Count);
        _c1pdf.DrawString(text, font, Brushes.Gray, rect, sf);
    }
}
```

Note that the code uses the **C1PdfDocumentBase.Pages** property to get the page count. **Pages** is a collection based on the **ArrayList** class, and has methods that allow you to count and enumerate pages, as well as add and remove pages at specific positions. You can use the **Pages** collection to remove pages from certain locations in the document and re-insert them elsewhere.

Adding Bookmarks and Annotations

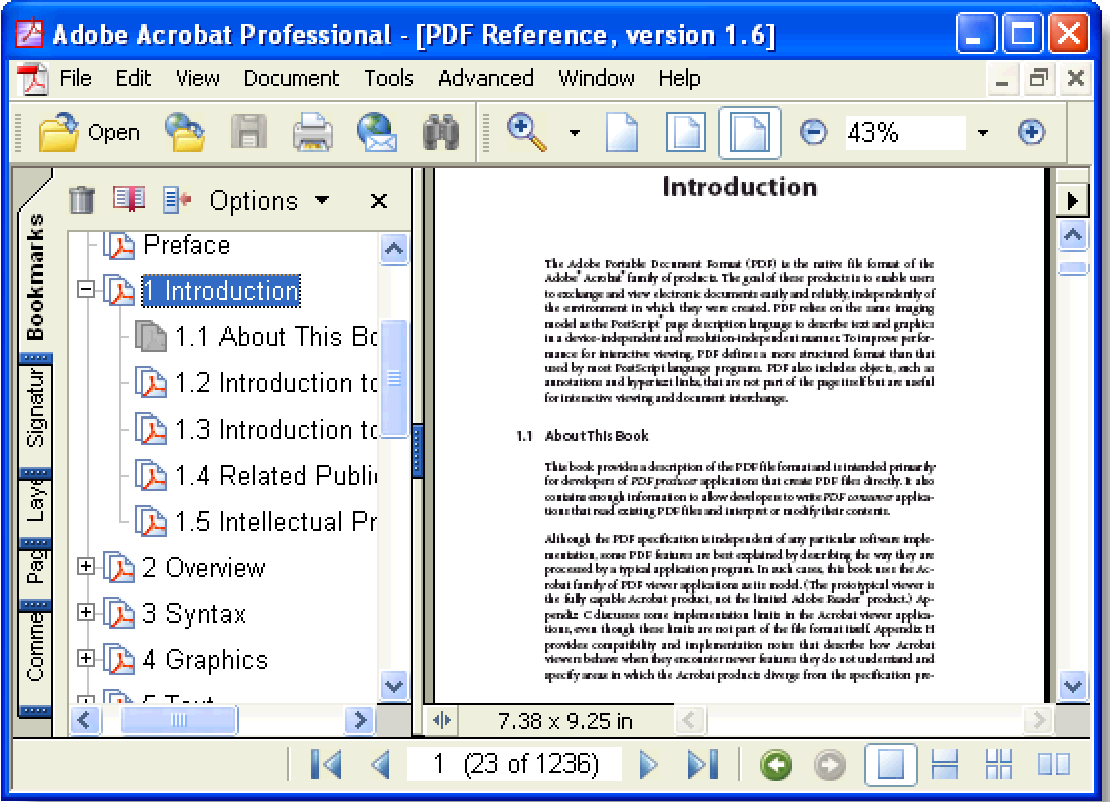
The following topics describe how to add bookmarks and annotations to a PDF document.

Adding Bookmarks to a PDF Document

When you open a PDF document using Adobe's Acrobat Reader application, you will notice that most long documents contain an outline structure that is displayed on a pane on the left of the reader. The outline makes it easy

Copyright © 2018 GrapeCity, Inc. All rights reserved.

to browse through a document's structure and find specific topics. The picture below shows a PDF document with an outline:



The outline entries are called Bookmarks, and you can add them to your **PDF for .NET** documents using the AddBookmark method. The **AddBookmark(String,Int32,Double)** method takes three parameters: the title of the outline entry, the outline level, and the 'y' position of the entry on the current page (measured in points from the top of the page).

For example, the routine below adds a paragraph to a document and optionally marks it as a level-zero outline entry:

To write code in Visual Basic

Visual Basic

```
Private Function RenderParagraph(text As String, font As Font, rect As RectangleF,
rectPage As RectangleF, outline As Boolean) As RectangleF
     ' If it doesn't fit on this page, add a page break.
    rect.Height = _c1pdf.MeasureString(text, font, rect.Width).Height
    If rect.Bottom > rectPage.Bottom Then
        _c1pdf.NewPage()
        rect.Y = rectPage.Top
    End If
     ' Draw the string.
    _c1pdf.DrawString(text, font, Brushes.Black, rect)
     ' Add headings to outline.
    If outline Then
        _c1pdf.DrawLine(Pens.Black, rect.X, rect.Y, rect.Right, rect.Y)
        _c1pdf.AddBookmark(text, 0, rect.Y)
    End If
```

Copyright © 2018 GrapeCity, Inc. All rights reserved.

```
    ' Update rectangle for next time.
    rect.Offset(0, rect.Height)
    Return rect
End Function
```

To write code in C#

C#

```
private RectangleF RenderParagraph(string text, Font font, RectangleF rect,
RectangleF rectPage, bool outline)
{
    // If it doesn't fit on this page, add a page break.
    rect.Height = _c1pdf.MeasureString(text, font, rect.Width).Height;
    if (rect.Bottom > rectPage.Bottom)
    {
        _c1pdf.NewPage();
        rect.Y = rectPage.Top;
    }
     // Draw the string.
    _c1pdf.DrawString(text, font, Brushes.Black, rect);
     // Add headings to outline.
    if (outline)
    {
        _c1pdf.DrawLine(Pens.Black, rect.X, rect.Y, rect.Right, rect.Y);
        _c1pdf.AddBookmark(text, 0, rect.Y);
    }
     // Update rectangle for next time.
    rect.Offset(0, rect.Height);
    return rect;
}
```

Note: You can also use more overloads for AddBookmark() method - to pass the boolean to specify whether the children of bookmark are initially visible or to pass the target name of the document.

Adding Links to a PDF Document

The PDF specification allows you to add several types of annotations to your documents. Annotations are often added by hand, as highlights and notes. But they can also be added programmatically. C1PdfDocument provides methods for adding hyperlinks, hyperlink targets, and file attachments to your PDF documents.

To add a hyperlink to your document, use the AddLink method. AddLink method takes two parameters: a string that specifies a *url* and a *RectangleF* that specifies the area on the current page that should behave as a link.

Note that the AddLink method does not add any visible content to the page, so you will usually need another command along with AddLink to specify some text or an image that the user can see. For example, the code below adds a string that says "Visit ComponentOne" and a link that takes the user to the ComponentOne home page:

Visual Basic

```
Dim rect As New RectangleF(50, 50, 100, 15)
Dim font As New Font("Arial", 10, FontStyle.Underline)
```

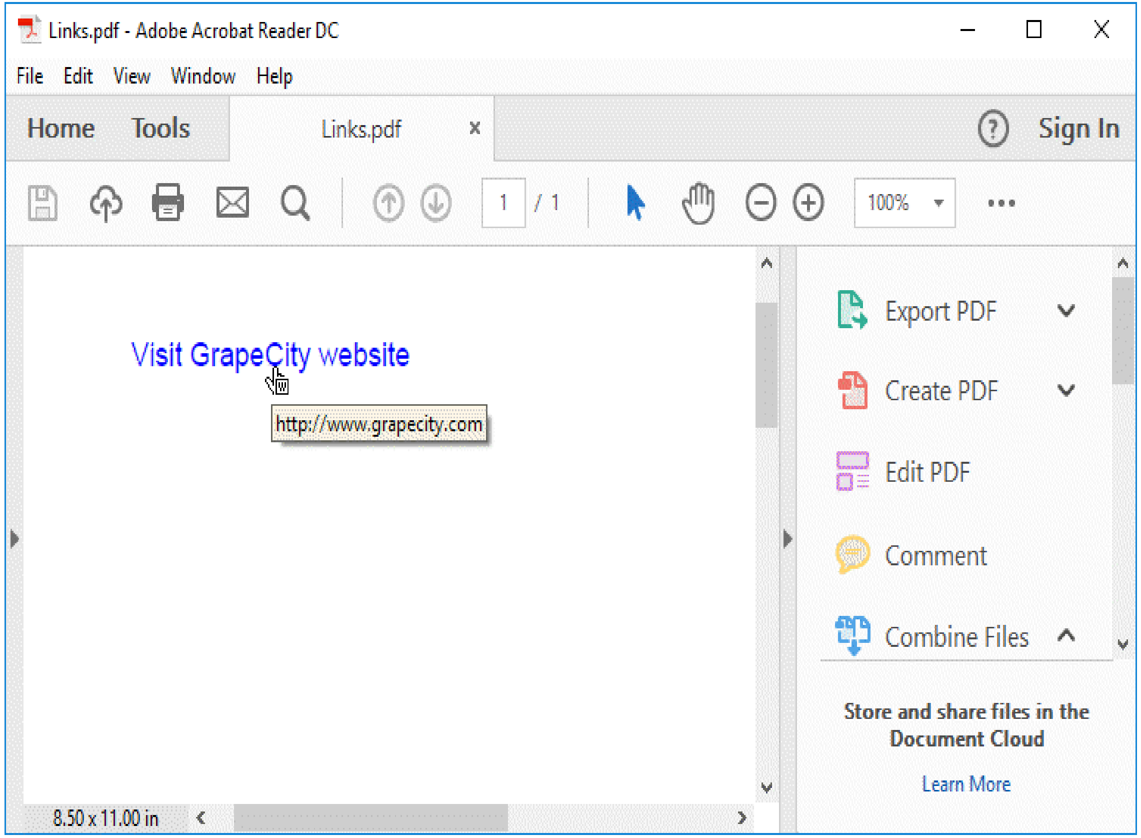
Copyright © 2018 GrapeCity, Inc. All rights reserved.

```
_c1pdf.AddLink("https://www.grapecity.com/", rect)
_c1pdf.DrawString("Visit ComponentOne", font, Brushes.Blue, rect)
```

C#

```
RectangleF rect = new RectangleF(50, 50, 100, 15);
Font font = new Font("Arial", 10, FontStyle.Underline);
_c1pdf.AddLink("https://www.grapecity.com/", rect);
_c1pdf.DrawString("Visit ComponentOne", font, Brushes.Blue, rect);
```

Here is the resulting PDF document:



You can also add local links, which when clicked take the user to another location within the same PDF document. This type of link is useful when you want to implement some type of cross-referencing within the document, such as a table of contents or an index.

Local links are identical to regular hyperlinks, except for two things:

- The *url* parameter must start with a "#".
- You must specify the target location for the link using the AddTarget method. The AddTarget method takes the same parameters as AddLink, a string that specifies the name of the target and a rectangle that marks the area on the page that will be displayed when the user selects the link.

Attaching Files to a PDF Document

Adding file attachments to PDF files is often a useful feature. Attachments can contain any kind of file, including spreadsheets with detailed information that would clutter the main document, multimedia files with movies and sound, sample code, and so on.

Copyright © 2018 GrapeCity, Inc. All rights reserved.

Adding file attachments to your **PDF for .NET** documents is easy. All you have to do is call the AddAttachment method and specify which file you want to attach, what area of the page should contain the attachment, and optionally, the appearance of the attachment.

For example, the following code attaches all files in the application directory to the PDF document:

To write code in Visual Basic

Visual Basic

```
Dim rect As New RectangleF(100, 100, 60, 10)
Dim font As New Font("Arial", 9)
 ' Attach some files.
Dim path As String = "c:\temp\files"
Dim file As String
For Each file In Directory.GetFiles(path)
    Dim width As Single = rect.Width
    rect.Width = rect.Height
    _c1pdf.FillRectangle(Brushes.Gray, rect)
    _c1pdf.AddAttachment(file, rect)
    rect.Width = width
    rect.X += rect.Height
    _c1pdf.DrawString(Path.GetFileName(file), font, Brushes.Black, rect)
    rect.X -= rect.Height
    rect.Y += 2 * rect.Height
Next file
```

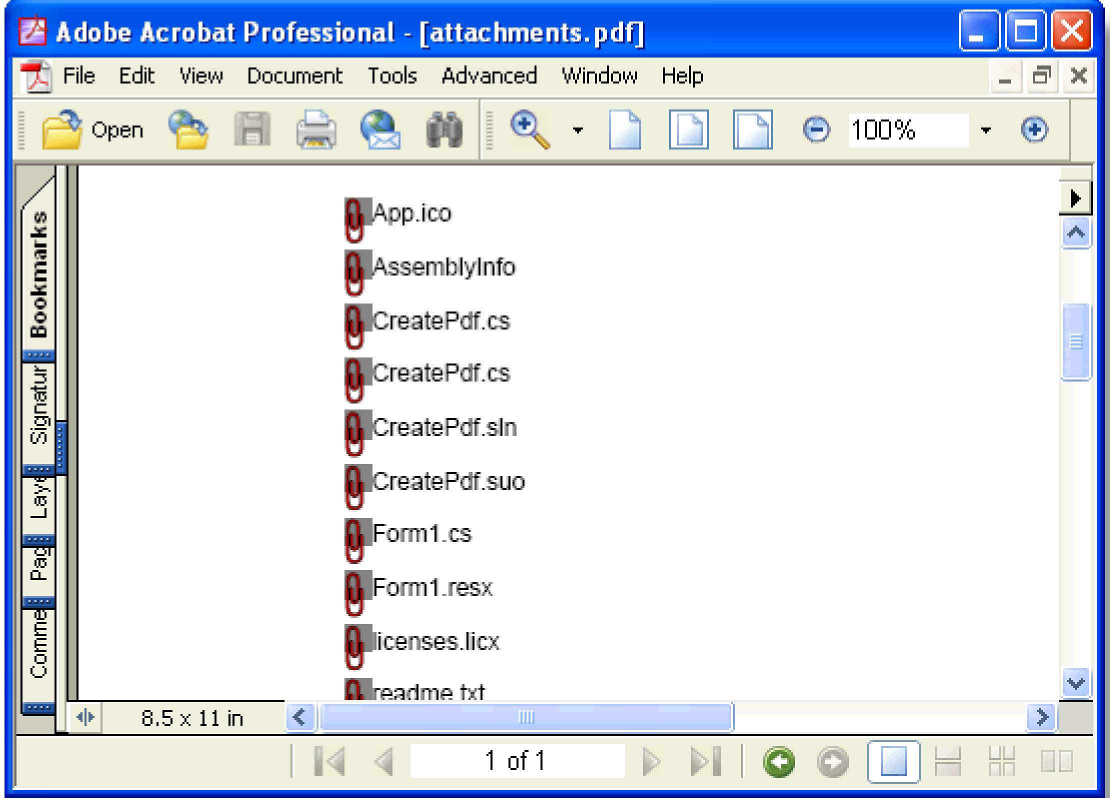
To write code in C#

C#

```
RectangleF rect = new RectangleF(100, 100, 60, 10);
Font font = new Font("Arial", 9);
 // Attach some files.
string path = @"c:\temp\files";
string[] files = Directory.GetFiles(path);
foreach (string file in files)
{
    float width = rect.Width;
    rect.Width = rect.Height;
    _c1pdf.FillRectangle(Brushes.Gray, rect);
    _c1pdf.AddAttachment(file, rect);
    rect.Width = width;
    rect.X += rect.Height;
    _c1pdf.DrawString(Path.GetFileName(file), font, Brushes.Black, rect);
    rect.X -= rect.Height;
    rect.Y += 2 * rect.Height;
}
```

Here's what the PDF document looks like in Adobe's Acrobat Reader:

Copyright © 2018 GrapeCity, Inc. All rights reserved.



The attachments are displayed as icons (you can select from four predefined icons in the AttachmentIconEnum enumeration and you can also select the icon color). When the user moves the mouse over the attachment, the file name is displayed and the cursor changes to indicate there is an attachment available. The user can then right-click the attachment name to open the attachment or save it to a file.

Applying Security and Permissions

By default, anyone can open, copy, print, and edit PDF files. If your PDF documents contain sensitive information, however, you can encrypt them so that only authorized users can access it.

There is a separate password for the owner of the document and for all other users. The user's access can be selectively restricted to allow only certain operations, such as viewing, printing, or editing the document.

PDF for .NET provides a **C1PdfDocumentBase.Security** property that returns a PdfSecurity object. This object has properties that allow you to specify the **owner password** (required to change passwords and permissions for the document) and the **user password** (required to open the document). Additionally, the PdfSecurity object allows you to specify what permissions a regular user should have. For example you may allow users to see the document but not to print or edit it.

To use the **PDF for .NET** security features, simply set the passwords before you save the document. For example:

To write code in Visual Basic

Visual Basic

```
 ' Create the document as usual.
CreateDoc()
  ' Set passwords.
_c1pdf.Security.OwnerPassword = "2mds%dffgd"
_c1pdf.Security.UserPassword = "anyone"
_c1pdf.Security.AllowEditAnnotations = False
```

Copyright © 2018 GrapeCity, Inc. All rights reserved.

```
_c1pdf.Security.AllowEditContent = False
_c1pdf.Security.AllowPrint = False
  ' Save the encrypted document.
_c1pdf.Save("")
```

To write code in C#

C#

```
// Create the document as usual.
CreateDoc();
 // Set passwords.
_c1pdf.Security.OwnerPassword = "2mds%dffgd";
_c1pdf.Security.UserPassword = "anyone";
_c1pdf.Security.AllowEditAnnotations = false;
_c1pdf.Security.AllowEditContent = false;
_c1pdf.Security.AllowPrint = false;
 // Save the encrypted document.
_c1pdf.Save(@"c:\reports\secure.pdf");
```

Note that you can specify permissions and set only the owner password, leaving the user password empty. In this case, anyone will be allowed to open the document, but only the owner will be allowed to change the permissions.

Note also that the encryption scheme used by **PDF for .NET** is public and is not 100% secure. There are ways to crack encrypted PDF documents. The security provided is adequate to protect your documents from most casual attacks, but if your data is truly sensitive you should not rely on PDF encryption alone.

Using Metafiles

PDF for .NET makes it very easy to create documents, mainly because the object model mimics the well-known .NET **Graphics** model. However, not all methods available in the **Graphics** class are available in **PDF for .NET**. Plus, you may have existing code that draws to a **Graphics** object and that you do not want to rewrite even if most methods are very similar.

In these cases, you can reuse your existing .NET code by sending the drawing commands to a **Metafile**, then rendering the **Metafile** into **PDF for .NET** using the DrawImage command. This method allows you to expose any graphics you create as images or as PDF documents.

For example, suppose you have an application that generates documents using the **PrintDocument** pattern of drawing each page into a **Graphics** object. You could then use the same methods to create a collection of metafiles, one per page, and then convert the list into a PDF document using the following code:

To write code in Visual Basic

Visual Basic

```
' Get the document as a list of Metafiles, one per page.
Dim pages As ArrayList = GetMetafiles()
 ' Loop through the pages and create a PDF document.
_c1pdf.Clear()
Dim i As Integer
for i = 0 i <= pages.Count
     ' Get ith page.
    Dim page As Metafile = CType(Metafile.FromFile(pages[i]), Metafile)
    If Not (page Is Nothing) Then
```

Copyright © 2018 GrapeCity, Inc. All rights reserved.

```
        ' Calculate the page size.
        Dim sz As SizeF = page.PhysicalDimension
        sz.Width = Math.Round(sz.Width * 72.0F / 2540.0F, 2)
        sz.Height = Math.Round(sz.Height * 72.0F / 2540.0F, 2)
        ' Add a page and set the size.
        If i > 0 Then
            _c1pdf.NewPage()
        End If
        _c1pdf.PageSize = sz
        ' Draw the page into the PDF document.
        _c1pdf.DrawImage(page, _c1pdf.PageRectangle)
    End If
Next
' Save to file.
_c1pdf.Save("c:\temp\mydoc.pdf")
```

To write code in C#

C#

```
// Get the document as a list of Metafiles, one per page.
ArrayList pages = GetMetafiles();
// Loop through the pages and create a PDF document.
_c1pdf.Clear();
for (int i = 0; i < pages.Count; i++)
{
    // Get ith page.
    Metafile page = (Metafile)Metafile.FromFile(pages[i]);
    if (page == null)
    {
        continue;
    }
    // Calculate the page size.
    SizeF sz = page.PhysicalDimension;
    sz.Width  = (float)Math.Round(sz.Width  * 72f / 2540f, 2);
    sz.Height = (float)Math.Round(sz.Height * 72f / 2540f, 2);
    // Add a page and set the size
    if (i > 0) _c1pdf.NewPage();
    _c1pdf.PageSize = sz;
    // Draw the page into the PDF document
    _c1pdf.DrawImage(page, _c1pdf.PageRectangle);
}
// Save to file
_c1pdf.Save(@"c:\temp\mydoc.pdf");
```

The code gets each metafile on the list, calculates its size in points (each page could have a different size), then draws the metafile into the page. The metafiles could be generated by a reporting engine, drawing or charting program, or any application that can create metafile images.

PDF for .NET also supports EMF+ metafiles. According to Microsoft, EMF+ is an extension to EMF that allows GDI+ records to be stored. Previously, all metafiles were transformed to EFM Only. EMF+ metafiles are no longer transformed; therefore, they do not lose their context due to transformation.

Copyright © 2018 GrapeCity, Inc. All rights reserved.

Rendering RTF Text

Another powerful feature of **PDF for .NET** is the ability to render rich text format (RTF) text using the DrawStringRtf method. This is an extremely useful feature mainly because there are so many tools for creating RTF, and it's easy to create templates in RTF and have your application customize, then render the RTF strings. This makes it easy to create richly formatted documents practically without writing any code.

For example, the code below uses an RTF template created with WordPad. It scans a directory and builds an RTF string based on the template, then renders the RTF using the DrawStringRtf method.

To write code in Visual Basic

Visual Basic

```
' Get RTF templates.
Dim rtfHdr As String = "" & _
   "{\rtf1\ansi\ansicpg1252\deff0\deflang1033" & _
   "{\fonttbl{\f0\fswiss\fcharset0 " & _
   "Arial;}{\f1\froman\fprq2\fcharset0 Book Antiqua;}}" & _
   "{\colortbl ;\red0\green0\blue0;}" & _
   "\viewkind4\uc1\pard\f0\fs20\par" & _
   "\pard\tx1440\tx2880\tx4320\tx5760\cf1\b\f1\fs24 " & _
   "Directory Report created on <>\par" & _
   "\ul\par Name\tab Extension\tab Size\tab Date\tab " & _
   "Attributes\par"
Dim rtfEntry As String = "" & _
   "\cf0\ulnone\b0\f0\fs16 <>\tab " & _
   "<>\tab <>\tab <>\tab <>\par"
 ' Build RTF string.
Dim sb As New StringBuilder()
sb.Append(rtfHdr.Replace("<>", DateTime.Today.ToShortDateString()))
Dim file As String
For Each file in Directory.GetFiles(@"c:\winnt", "*.bmp")
   Dim s As String = rtfEntry
   Dim fi As FileInfo = New FileInfo(file)
   s = s.Replace("<>", Path.GetFileNameWithoutExtension(file))
   s = s.Replace("<>", fi.Extension)
   s = s.Replace("<>", String.Format("{0:#,##0}", fi.Length))
   s = s.Replace("<>", fi.LastWriteTime.ToShortDateString())
   s = s.Replace("<>", fi.Attributes.ToString())
   sb.Append(s)
Next
sb.Append("}")
 ' Render it.
Dim pdf As New C1PdfDocument()
Dim rect As RectangleF = _c1pdf.PageRectangle
rect.Inflate(-72, -72)
_c1pdf.DrawStringRtf(sb.ToString(), Font, Brushes.Black, rect)
 ' Save.
_c1pdf.Save("c:\temp\dir.pdf")
```

To write code in C#

Copyright © 2018 GrapeCity, Inc. All rights reserved.

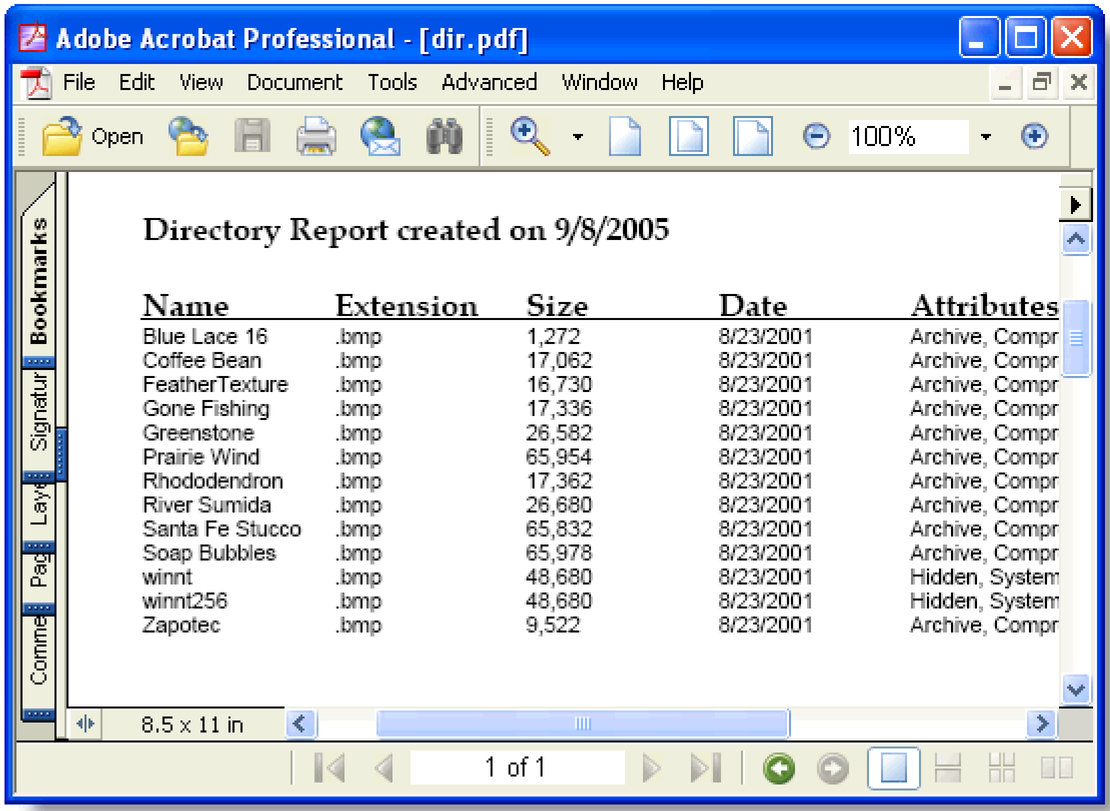
C#

```
// Get RTF templates.
string rtfHdr = @"{\rtf1\ansi\ansicpg1252\deff0\deflang1033" +
       @"{\fonttbl{\f0\fswiss\fcharset0 " +
       @"Arial;}{\f1\froman\fprq2\fcharset0 Book Antiqua;}}" +
       @"{\colortbl ;\red0\green0\blue0;}" +
       @"\viewkind4\uc1\pard\f0\fs20\par" +
       @"\pard\tx1440\tx2880\tx4320\tx5760\cf1\b\f1\fs24 " +
       @"Directory Report created on <>\par" +
       @"\ul\par Name\tab Extension\tab Size\tab Date\tab " +
       @"Attributes\par";
string rtfEntry = @"\cf0\ulnone\b0\f0\fs16 <>\tab " +
       @"<>\tab <>\tab <>\tab <>\par";
 // Build RTF string.
StringBuilder sb = new StringBuilder();
sb.Append(rtfHdr.Replace("<>", DateTime.Today.ToShortDateString()));
foreach (string file in Directory.GetFiles(@"c:\winnt", "*.bmp"))
{
       string s = rtfEntry;
       FileInfo fi = new FileInfo(file);
       s = s.Replace("<>", Path.GetFileNameWithoutExtension(file));
       s = s.Replace("<>",  fi.Extension);
       s = s.Replace("<>", string.Format("{0:#,##0}", fi.Length));
       s = s.Replace("<>", fi.LastWriteTime.ToShortDateString());
       s = s.Replace("<>", fi.Attributes.ToString());
       sb.Append(s);
}
sb.Append("}");
 // Render it.
C1.C1Pdf.C1PdfDocument pdf = new C1.C1Pdf.C1PdfDocument();
RectangleF rect = pdf.PageRectangle;
rect.Inflate(-72, -72);
_c1pdf.DrawStringRtf(sb.ToString(), Font, Brushes.Black, rect);
 // Save.
_c1pdf.Save(@"c:\temp\dir.pdf");
```

The code is very simple. The only complicated part is the RTF definition, but that was copied and pasted from an RTF file created with WordPad. The RTF template string contains tags (for example, "<<NAME>>") that are then replaced with the actual contents.

When the RTF string is built, it is rendered into the PDF document using the DrawStringRtf command. The result looks like this:

Copyright © 2018 GrapeCity, Inc. All rights reserved.



Directory Report created on 9/8/2005

Name	Extension	Size	Date	Attributes
Blue Lace 16	.bmp	1,272	8/23/2001	Archive, Compr
Coffee Bean	.bmp	17,062	8/23/2001	Archive, Compr
FeatherTexture	.bmp	16,730	8/23/2001	Archive, Compr
Gone Fishing	.bmp	17,336	8/23/2001	Archive, Compr
Greenstone	.bmp	26,582	8/23/2001	Archive, Compr
Prairie Wind	.bmp	65,954	8/23/2001	Archive, Compr
Rhododendron	.bmp	17,362	8/23/2001	Archive, Compr
River Sumida	.bmp	26,680	8/23/2001	Archive, Compr
Santa Fe Stucco	.bmp	65,832	8/23/2001	Archive, Compr
Soap Bubbles	.bmp	65,978	8/23/2001	Archive, Compr
winnt	.bmp	48,680	8/23/2001	Hidden, System
winnt256	.bmp	48,680	8/23/2001	Hidden, System
Zapotec	.bmp	9,522	8/23/2001	Archive, Compr

Copyright © 2018 GrapeCity, Inc. All rights reserved.

PDF for .NET Samples

Please be advised that this ComponentOne software tool is accompanied by various sample projects and/or demos which may make use of other development tools included with the ComponentOne Studio.

Please refer to the pre-installed product samples through the following path:

Documents\ComponentOne Samples\WinForms

Click one of the following links to view a list of **PDF for .NET** samples:

Visual Basic Samples

Sample	Description
CoolPrintPreview	Compares standard Print Preview Dialog and enhanced PDF Print Preview Dialog.
PdfForm	Creates a PDF version of a paper form. This sample loads an empty fax cover form (created in Visio and saved as an enhanced metafile), and then uses the DrawString method to fill in the fields based on information you enter on the main form.
PrintDocument	Renders a PrintDocument into a PDF file. This sample uses the C1PdfDocument control.

C# Samples

Sample	Description
CharTable	Creates a PDF document containing a character table. The sample shows a simple way to create tables. It creates a rectangle and then moves it across to create each cell in a row. At the end of the row, it moves the rectangle back to the left and one row down. The sample also shows all ASCII characters rendered in PDF.
Charts	Creates a PDF document containing charts created with the **C1Chart** control. The sample creates metafiles for several 2 and 3D charts, and then renders the metafiles into a PDF document using the DrawImage method. The charts are rendered as a sequence of graphics commands (they are not translated into bitmaps). As a result, the charts are rendered in a high-quality, resolution-independent way. This sample uses the C1PdfDocument, **C1Chart** and **C1Chart3D** controls.
CoolPrintPreview	Compares standard Print Preview Dialog and enhanced PDF Print Preview Dialog.
CreatePdf	Creates several PDF documents that show different aspects of the C1Pdf library. This sample uses the C1PdfDocument control.
FlowHtml	This sample shows how to render HTML into PDF documents and uses the C1PdfDocument control.
FlowRtf	This sample shows how to render Rtf text that flows onto several pages. This sample uses the C1PdfDocument control.
ManualSamples	Contains a variety of examples, such as how to render a long string into a short string, how to add links, and how to draw lines. Each example is described in the PDF created. This sample uses the C1PdfDocument control.
MixedOrientation	This sample renders two **C1Reports** (one portrait, one landscape) and then merges them into a single C1PdfDocument using the **C1Report.PageImages** property and the DrawImage method.
PdfAcroform	This sample shows how to create a PDF AcroForm with a variety of fields and uses the C1PdfDocument control.
PdfAnnotations	This sample creates a PDF document with a form containing several notes, including rectangle,

Copyright © 2018 GrapeCity, Inc. All rights reserved.

	circle, line and text box. This sample uses the C1PdfDocument control.
PdfChart	The sample creates a pie chart, converts it into a metafile, and then renders the metafile into a PDF document using the DrawImage method. The chart is rendered as a sequence of graphics commands (it is not translated into a bitmap). As a result, the chart is rendered in a high-quality, resolution-independent way. This sample uses **C1Chart**.
PdfSignature	This sample enables you to render a PDF document with digital signature in the form of text or an image.
PdfTables	This sample exports an HTML document to the PDF format and uses the C1PDF component.
Security	Creates a PDF document with password protection and permissions. This sample uses the C1PdfDocument control.

Copyright © 2018 GrapeCity, Inc. All rights reserved.

PDF for .NET Task-Based Help

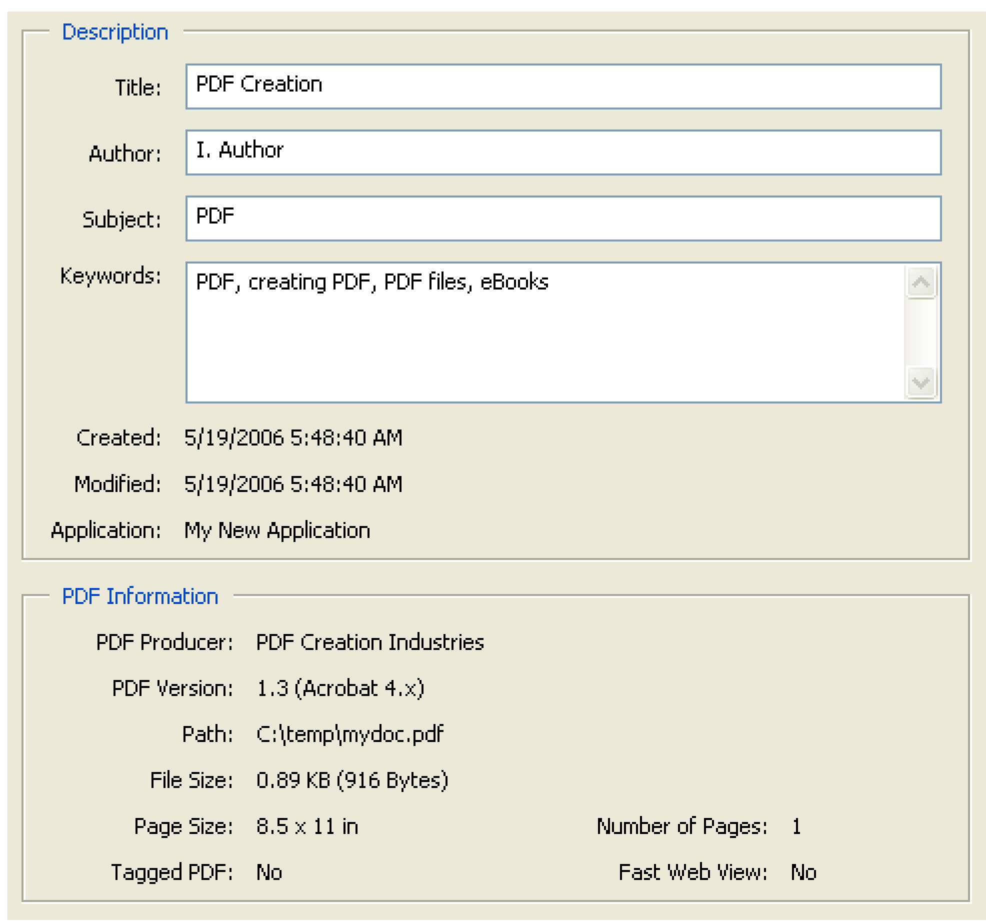
The task-based help section assumes that you are familiar with programming in the Visual Studio environment, and know how to use the **PDF for .NET** control in general. If you are a novice to the PDF for .NET, please see Using ComponentOne PDF for .NET first.

Each topic provides a solution for specific tasks using the **PDF for .NET** product. By following the steps outlined in the help, you will be able to utilize the features of **PDF for .NET**.

Each task-based help topic also assumes that you have created a new .NET project.

Setting Document Properties

Setting the document properties for a PDF document allows you to customize the **Title**, **Author**, **Subject**, **Keywords**, and **Application** under **Description**, as well as **PDF Producer** under **PDF Information**. To set the document properties, enter information in the Title, Author, Subject, Keywords, Creator, and Producer properties either in the designer or in code.

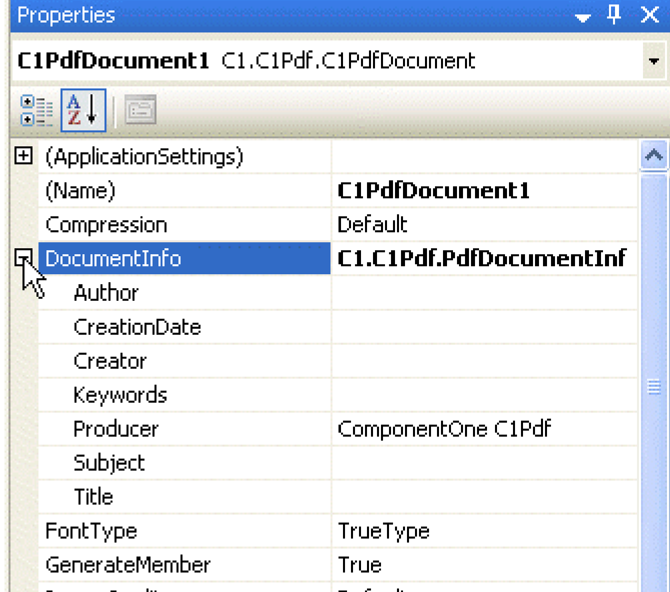


Note: To set the Application under Description, use the Creator property. The document properties appear in the Document Properties dialog box in the PDF document.

In the Designer

In the Properties window, locate the **C1PdfDocumentBase.DocumentInfo** property and expand the property node.

Copyright © 2018 GrapeCity, Inc. All rights reserved.



Type in the information for a property and press ENTER when finished to set the property.

In Code

Add the following code to the **Form_Load** event:

1. Create the C1PdfDocument object:

   To write code in Visual Basic

   Visual Basic

   ```
   ' Create the C1PdfDocument object.
   Dim pdf As New C1.C1Pdf.C1PdfDocument()
   ```

   To write code in C#

   C#

   ```
   // Create the C1PdfDocument object.
   C1.C1Pdf.C1PdfDocument pdf = new C1.C1Pdf.C1PdfDocument();
   ```

2. Set the document properties:

   To set the Title property, add the following code:

   To write code in Visual Basic

   Visual Basic

   ```
   pdf.DocumentInfo.Title = "PDF Creation"
   ```

   To write code in C#

   C#

   ```
   pdf.DocumentInfo.Title = "PDF Creation";
   ```

Copyright © 2018 GrapeCity, Inc. All rights reserved.

To set the Author property, add the following code:

To write code in Visual Basic

```
Visual Basic
pdf.DocumentInfo.Author = "I. Author"
```

To write code in C#

```
C#
pdf.DocumentInfo.Author = "I. Author";
```

To set the Subject property, add the following code:

To write code in Visual Basic

```
Visual Basic
pdf.DocumentInfo.Subject = "PDF"
```

To write code in C#

```
C#
pdf.DocumentInfo.Subject = "PDF";
```

To set the Keywords property, add the following code:

To write code in Visual Basic

```
Visual Basic
pdf.DocumentInfo.Keywords = "PDF, creating PDF, PDF files, eBooks"
```

To write code in C#

```
C#
pdf.DocumentInfo.Keywords = "PDF, creating PDF, PDF files, eBooks";
```

To set the Creator property, add the following code:

To write code in Visual Basic

```
Visual Basic
pdf.DocumentInfo.Creator = "My New Application"
```

To write code in C#

```
C#
pdf.DocumentInfo.Creator = "My New Application";
```

To set the Producer property, add the following code. The default is "ComponentOne C1Pdf".

To write code in Visual Basic

Copyright © 2018 GrapeCity, Inc. All rights reserved.

Visual Basic

```
pdf.DocumentInfo.Producer = "PDF Creation Industries"
```

To write code in C#

C#

```
pdf.DocumentInfo.Producer = "PDF Creation Industries";
```

3. Save the document to a file:

To write code in Visual Basic

Visual Basic

```
' Save the document to a file.
pdf.Save("c:\mydoc.pdf")
```

To write code in C#

C#

```
// Save the document to a file.
pdf.Save(@"c:\mydoc.pdf");
```

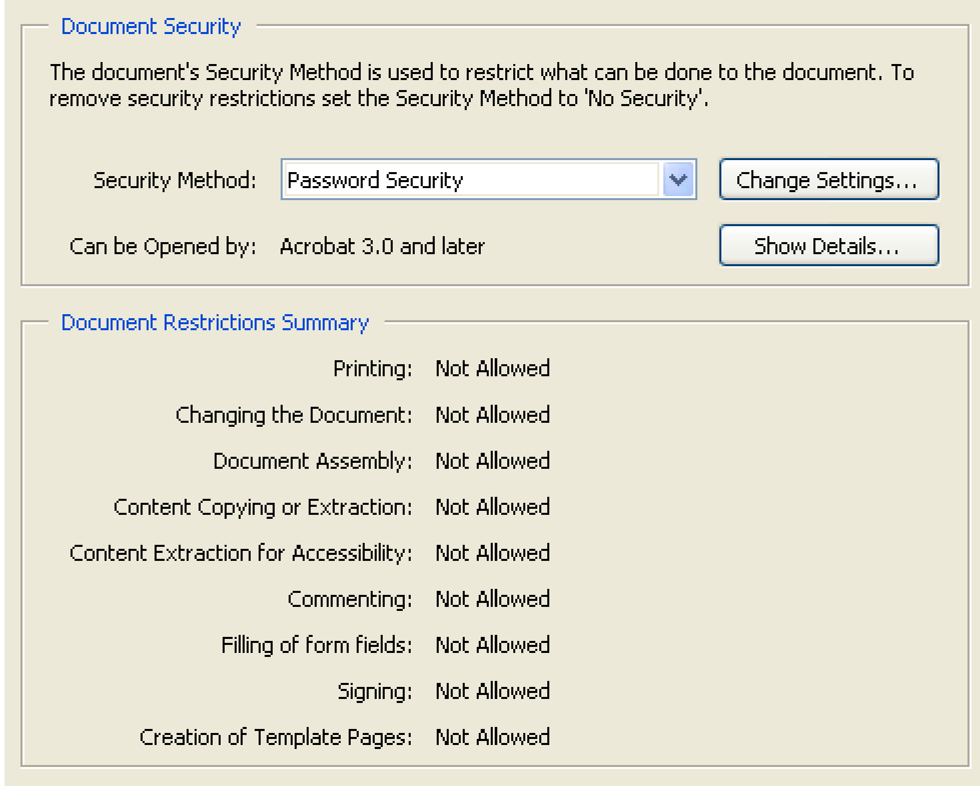
Setting Document Restrictions

Setting document restrictions allows you to customize if a user can copy content, edit annotations, edit content, or print the PDF file. To enable document restrictions, set the following properties to **False** either in the designer or in code:

- AllowCopyContent
- AllowEditAnnotations
- AllowEditContent
- AllowPrint

For more information on setting passwords and permissions, see Applying Security and Permissions.

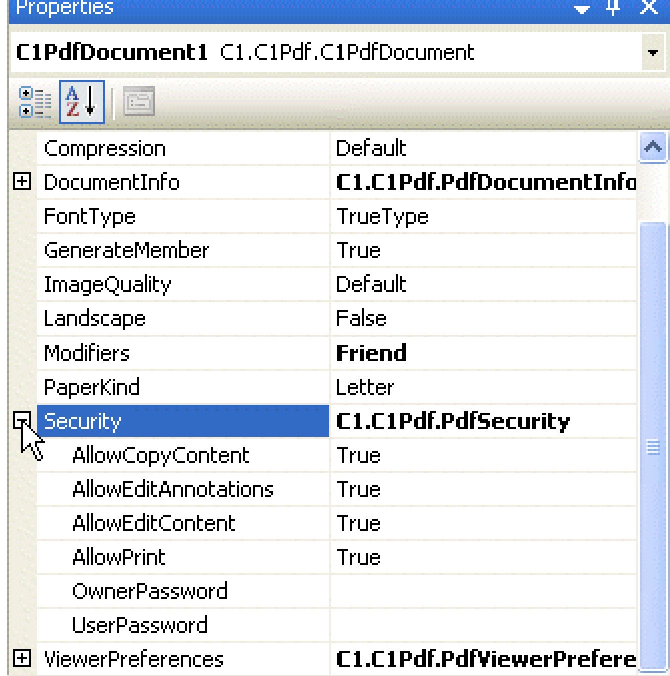
Copyright © 2018 GrapeCity, Inc. All rights reserved.



Note: The document restrictions appear in the **Document Properties** dialog box in the PDF document.>

In the Designer

1. In the Properties window, locate the **C1PdfDocumentBase.Security** property and expand the property node.



2. Set the OwnerPassword property by typing in a password and pressing ENTER when finished.
3. Set the AllowCopyContent, AllowEditAnnotations, AllowEditContent, and AllowPrint properties to **False** to not allow copying, editing, and printing.

Copyright © 2018 GrapeCity, Inc. All rights reserved.

In Code

Add the following code to the **Form_Load** event.

1. Create the C1PdfDocument object:

   To write code in Visual Basic

   Visual Basic
   ```
   ' Create the C1PdfDocument object.
   Dim pdf As New C1.C1Pdf.C1PdfDocument()
   ```

   To write code in C#

   C#
   ```
   //Create the C1PdfDocument object.
   C1.C1Pdf.C1PdfDocument pdf = new C1.C1Pdf.C1PdfDocument();
   ```

2. Add the following code to set the OwnerPassword property:

   To write code in Visual Basic

   Visual Basic
   ```
   pdf.Security.OwnerPassword = "Password"
   ```

   To write code in C#

   C#
   ```
   pdf.Security.OwnerPassword = "Password";
   ```

3. Add the following code to set the AllowCopyContent, AllowEditAnnotations, AllowEditContent, and AllowPrint properties to **False** to not allow copying, editing, and printing:

   To write code in Visual Basic

   Visual Basic
   ```
   pdf.Security.AllowCopyContent = False
   pdf.Security.AllowEditAnnotations = False
   pdf.Security.AllowEditContent = False
   pdf.Security.AllowPrint = False
   ```

   To write code in C#

   C#
   ```
   pdf.Security.AllowCopyContent = False;
   pdf.Security.AllowEditAnnotations = False;
   pdf.Security.AllowEditContent = False;
   pdf.Security.AllowPrint = False;
   ```

4. Save the document to a file:

   To write code in Visual Basic

Copyright © 2018 GrapeCity, Inc. All rights reserved.

Visual Basic

```
' Save the document to a file.
pdf.Save("c:\mydoc.pdf")
```

To write code in C#

C#

```
// Save the document to a file.
pdf.Save(@"c:\mydoc.pdf");
```

Setting Paper Size

To set the paper size, use the PaperKind property. This property can be set either in the designer or in code. For more information on paper size, see Creating Pages and Overlays.

In the Designer

Locate the PaperKind property in the Properties window and choose the new paper size.

In Code

Add the following code to the **Form_Load** event to set the PaperKind property to Legal size paper:

To write code in Visual Basic

Visual Basic

```
' Create the C1PdfDocument object.
Dim pdf As New C1.C1Pdf.C1PdfDocument()
 ' Use Legal size paper.
pdf.PaperKind = Printing.PaperKind.Legal
 ' Save the document to a file.
pdf.Save("c:\mydoc.pdf")
```

To write code in C#

C#

```
// Create the C1PdfDocument object.
C1.C1Pdf.C1PdfDocument pdf = new C1.C1Pdf.C1PdfDocument();
 // Use Legal size paper.
pdf.PaperKind = Printing.PaperKind.Legal;
 // Save the document to a file.
pdf.Save(@"c:\mydoc.pdf");
```

Setting Landscape Page Orientation

To set the page orientation, use the Landscape property. This property can be set either in the designer or in code. For more information on page settings, see Creating Pages and Overlays.

Copyright © 2018 GrapeCity, Inc. All rights reserved.

In the Designer

Locate the Landscape property in the Properties window and set it to **True**.

In Code

Add the following code to the **Form_Load** event to set the Landscape property:

To write code in Visual Basic

Visual Basic

```
' Create the C1PdfDocument object.
Dim pdf As New C1.C1Pdf.C1PdfDocument()
 ' Use landscape page orientation.
pdf.Landscape = True
 ' Save the document to a file.
pdf.Save("c:\mydoc.pdf")
```

To write code in C#

C#

```
// Create the C1PdfDocument object.
C1.C1Pdf.C1PdfDocument pdf = new C1.C1Pdf.C1PdfDocument();
// Use landscape page orientation.
pdf.Landscape = true;
 // Save the document to a file.
pdf.Save(@"c:\mydoc.pdf");
```

Setting Image Quality

To set the image quality in the PDF document, use the **C1PdfDocumentBase.ImageQuality** property. This property can be set either in the designer or in code. For more information on images and image quality, see Adding Images.

In the Designer

Locate the **C1PdfDocumentBase.ImageQuality** property in the Properties window and set it to one of the following ImageQualityEnum settings: Low, Medium, Default, or High.

In Code

Add the following code to the **Form_Load** event to set the **C1PdfDocumentBase.ImageQuality** property to high quality images:

To write code in Visual Basic

Visual Basic

```
' Create the C1PdfDocument object.
Dim pdf As New C1.C1Pdf.C1PdfDocument()
 ' Use high quality images.
pdf.ImageQuality = C1.C1Pdf.ImageQualityEnum.High
 ' Save the document to a file.
```

Copyright © 2018 GrapeCity, Inc. All rights reserved.

```
pdf.Save("c:\mydoc.pdf")
```

To write code in C#

C#
```
// Create the C1PdfDocument object.
C1.C1Pdf.C1PdfDocument pdf = new C1.C1Pdf.C1PdfDocument();
 // Use high quality images.
pdf.ImageQuality = C1.C1Pdf.ImageQualityEnum.High;
 // Save the document to a file.
pdf.Save(@"c:\mydoc.pdf");
```

Printing PDF Files

The following code allows you to add print functionality to **PDF for .NET**. In this example, the code was added to the **Button1_Click** event. Note that you must use the **Imports System.Diagnostics** (Visual Basic) or **using System.Diagnostics** (C#) directive at the top of your form in order for the following code to work.

To write code in Visual Basic

Visual Basic
```
Process.Start("C:\Program Files\Adobe\Acrobat 5.0\Acrobat\Acrobat.exe", "/p /h " &
"c:\hello world.pdf")
```

To write code in C#

C#
```
Process.Start(@"C:\Program Files\Adobe\Acrobat 5.0\Acrobat\Acrobat.exe", "/p /h " +
@"c:\hello world.pdf");
```

Writing PDF for .NET Content in Web Applications

This topic provides an example of how to write **PDF for .NET** content directly into a Web page's output buffer with no temporary files.

1. In your Web application project, select **View | Component Designer**.
2. Add the C1PdfDocument control to your form.
3. Select **View | Code**, and add the following directives to your project:

   To write code in Visual Basic

   Visual Basic
   ```
   Imports C1.C1Pdf
   Imports System.IO
   ```

   To write code in C#

   C#
   ```
   using C1.C1Pdf;
   ```

Copyright © 2018 GrapeCity, Inc. All rights reserved.

```
using System.IO;
```

4. Then add the following code:

To write code in Visual Basic

Visual Basic

```
Public Sub RenderPdf(ByVal doc As C1PdfDocument)
  ' Render PDF document into memory-based PDF stream.
    Dim ms As New MemoryStream()
    doc.Save(ms)
     ' Get response object
    Dim length As Integer
    Dim rsp As HttpResponse = Me.Page.Response
        ' Clear it.
    rsp.Clear()
      Write PDF stream into response buffer
    rsp.ContentType = "Application/pdf"
    length = CInt(ms.Length)
    rsp.OutputStream.Write(ms.GetBuffer(), 0, length)
        ' Finish.
    rsp.Flush()
        ' This is required!
    rsp.SuppressContent = True
End Sub
```

To write code in C#

C#

```
protected void RenderPDF(C1PdfDocument doc)
{
    // Render PDF document into memory-based PDF stream.
    MemoryStream ms = new MemoryStream();
    doc.Save(ms);
     // Get response object.
    int length;
    HttpResponse rsp = this.Page.Response;
     // Clear it
    rsp.Clear();
     // Write PDF stream into response buffer
    rsp.ContentType = "Application/pdf";
    length = (int)ms.Length;
    rsp.OutputStream.Write(ms.GetBuffer(), 0, length);
     // Finish.

        rsp.Flush();
     // This is required.
    rsp.SuppressContent = true;
}
```

Copyright © 2018 GrapeCity, Inc. All rights reserved.

Using PDF for .NET with Other ComponentOne Tools

The following topics discuss using the **PDF for .NET** control with other development tools included with the ComponentOne Studio.

Adding ComponentOne FlexGrid for WinForms Data into PDF for .NET

To add **FlexGrid for WinForms** data into **PDF for .NET**, use **C1FlexGrid**'s **CreateImage** method and render the images using the DrawImage method. The images created by the grid are metafiles.

Note: For a complete sample using this method, download the PdfExport sample located on http://our.componentone.com/samples/.

Creating a PDF from a Report Rendered by VSPrinter

To convert documents created with the **VSPrinter** ActiveX control to PDF, you can retrieve the page images using the **VSPrinter.Picture** property, then render the images into **PDF for .NET** using the DrawImage method.

The code below creates a VSPrinter document and then converts it into PDF:

To write code in Visual Basic

Visual Basic

```
' Initialize PDF document.
_pdf.Clear()
 ' Create a VSPrinter document (or load it from an existing document).
.StartDoc()
For i = 1 To 1000
    .Paragraph = "This is paragraph " & i.ToString()
Next
.EndDoc()
 ' Scan the document page by page (1-based).
For i = 1 To .PageCount
     ' Set the current preview page.
    .PreviewPage = i
    ' Retrieve the image.
    Dim img As Image = .Picture
     ' Save image into PDF document.
    If i > 1 Then
        _pdf.NewPage()
    End If
    _pdf.DrawImage(img, Me._pdf.PageRectangle)
Next
 ' Return preview to first page.
.PreviewPage = 0
```

To write code in C#

C#

Copyright © 2018 GrapeCity, Inc. All rights reserved.

```
// Initialize PDF document.
_pdf.Clear();
 // Create a VSPrinter document (or load it from an existing document).
.StartDoc();
for (i = 1; i <= 1000; i++)
{
    .Paragraph = "This is a paragraph " + i.ToString() ;
}
.EndDoc();
 // Scan the document page by page (1-based).
for (i = 1; i <= .PageCount; i++)
{
     // Set the current preview page.
    .PreviewPage = i;
     // Retrieve the image.
    Image img = .Picture;
     // Save the image into PDF document.
    if (i > 1)
    {
        _pdf.NewPage();
    }
    _pdf.DrawImage(img, this._pdf.PageRectangle);
}
 // Return preview to first page.
.PreviewPage = 0;
```

Removing PDF Bookmarks from a C1Report Document

You can remove a PDF bookmark from a **C1Report** document by completing one of the following tasks:

- Set the OutlineRootLevel property to -1.

  OR

- Create a PDF filter and disable the outline.

To write code in Visual Basic

Visual Basic

```
' Set the OutlineRootLevel to -1.
Me.C1Report1.OutlineRootLevel = - 1
Me.C1Report1.RenderToFile("report.pdf", C1.C1Report.FileFormatEnum.PDF)
' OR: Create a PDF filter and disable the outline.
Dim f As New C1.C1Report.PdfFilter("report.pdf")
f.Outline = False
Me.C1Report1.RenderToFilter(f)
```

To write code in C#

C#

Copyright © 2018 GrapeCity, Inc. All rights reserved.

```
// Set the OutlineRootLevel to -1.
this.c1Report1.OutlineRootLevel = -1;
this.c1Report1.RenderToFile("report.pdf", C1.C1Report.FileFormatEnum.PDF);
// OR: Create a PDF filter and disable the outline.
C1. C1Report.PdfFilter f = new C1. C1Report.PdfFilter("report.pdf");
f.Outline = false;
this.c1Report1.RenderToFilter(f);
```

Copyright © 2018 GrapeCity, Inc. All rights reserved.