Table of Contents

Overview

GrapeCity Documents is a cross-platform solution for document management which aims to provide a universal document, editor and viewer solution for all popular document formats.

GrapeCity Documents for PDF library, referred to as **GcPdf**, is a part of GrapeCity Documents that handles majority of the PDF related needs as it conforms to a large part of Adobe PDF specification 1.7. The extensive library supported on .NET Standard 2.0, can be used to read, create, modify and save PDF files without using any external tool like Adobe Acrobat. It offers a rich feature set that allows developers to create PDF files with advanced font support and features, images, graphics, barcode, annotations, outlines, stamps, watermark and more. It also allows the developers to make changes at the document level; for example, working with document properties, page size, orientation, security and signatures, file compression, generating linearized PDF document are few to mention. Moreover, all these features are fully supported on Windows, Linux, and MAC systems.

In addition, GcPdf provides full text support in .NET Standard 2.0, despite the fact that major classes related to text and image are missing in .NET Core. This makes it a cross-platform solution for many developers looking for a PDF library to generate PDF files for multi-device applications.

Key Features

GcPdf provides many different features that enable the developers to build intuitive and professional-looking applications. The main features for GcPdf Library are as follows:

- **Generate, load, modify and save PDFs**
  Using GcPdf, you can create PDF documents with simple or complex business requirements in .NET Standard applications. Moreover, you can also load, modify PDFs from any source and save them again.
- **Save PDF document as an Image**
  GcPdf enables you to save PDF as Image without hampering the image quality. Further, you can execute this feature with minimal lines of code.
- **Supported PDF versions**
  GcPdf supports PDF 1.3, 1.4, 1.5, 1.6, and 1.7 and PDF/A version. Moreover, you can also set the PDF/A conformance levels.
- **Advanced text handling**
  GcPdf supports standard PDF, True Type, Open Type, and WOFF fonts along with features such as automatic font embedding and subsetting. It provides full text supporting libraries built for .NET Standard 2.0 target, which are system-independent and work on all supported platforms such as .NET Core, .NET Framework etc. Moreover, it provides numerous text handling features like text formatting, paragraph formatting, multiline text, text alignment, text wrap, text extract, line spacing, bidirectional text etc. and support for multiple languages.
- **Add PDF security**
  GcPdf library allows you to apply robust security while generating PDF documents. GcPdf easily protects your documents using some basic security properties like EncryptHandler, OwnerPassword, UserPassword, AllowCopyContent, AllowEditContent, AllowPrint and more. It is also possible to secure the PDF documents by signing them digitally.
- **Incremental Update**
  GcPdf supports incremental update, which among other things allows to add multiple digital signatures to a PDF document, while keeping them all valid.
- **Add form fields**
  GcPdf allows you to add, modify, and delete different form fields, such as text, check box, radio buttons, signature etc., to create interactive form. With the help of form fields, you can easily create fillable forms in your PDF document.
- **Import and export form data**
  GcPdf provides the capability to import or export PDF forms data from or to XML, FDF and XFDF files.
- **Generate linearized PDF**
  GcPdf allows generation of linearized PDF files to help you load your files quickly.
- **Rich set of features**
  GcPdf library provides a rich set of features that allow you to generate complex PDF documents with content including text, graphics, images, annotations, outlines and more.
- **GrapeCity Documents PDF Viewer**
  GrapeCity Documents PDF Viewer is a fast javascript based client-side viewer that allows you to view PDF documents. It supports many of the standard PDF features.

For additional information about the supported features in GcPdf, see Features topic.

Getting Started

System requirements

GcPdf system requirements, depending upon the framework you are using to create an application, are:

- Our packages include two targets, .NET Standard 2.0 and .NET Framework 4.6.1. In order to use them, your application needs to target either of the following:
  - .NET Core 2.0 or later
  - .NET Framework 4.6.1 or later
- Visual Studio 2015+/Visual Studio for MAC/Visual Studio Code for Linux

For OS versions supported in .NET Core 2.0+, see .NET Core 2.0+ - Supported OS versions.

Setting up an application

GcPdf references are available through NuGet, a Visual Studio extension that adds the required libraries and references to your project automatically. To work with GcPdf, you need to have following references in your application:

Reference	Purpose
GrapeCity.Documents.Pdf	To use GcPdf in an application, you need to reference (install) just the GrapeCity.Documents.Pdf package. It will pull in the required infrastructure packages.
GrapeCity.Documents.BarCode	To render barcodes, install the GrapeCity.Documents.Barcode (aka GcBarcode) package. It provides extension methods allowing to render barcodes when using GcPdf.
GrapeCity.Documents.Imaging	GrapeCity.Documents.Imaging provides image handling. You do not need to reference it directly.
GrapeCity.Documents.Common	GrapeCity.Documents.Common is an infrastructure package used by GcPdf and GcBarcode. You do not need to reference it directly.
GrapeCity.Documents.Common.Windows	On a Windows system, you can optionally install GrapeCity.Documents.Common.Windows. It provides support for font linking specified in the Windows registry, and access to native Windows imaging APIs, improving performance and adding some features (e.g. TIFF support).
GrapeCity.Documents.DX.Windows	GrapeCity.Documents.DX.Windows is an infrastructure package used by GrapeCity.Documents.Common.Windows. You do not need to reference it directly.

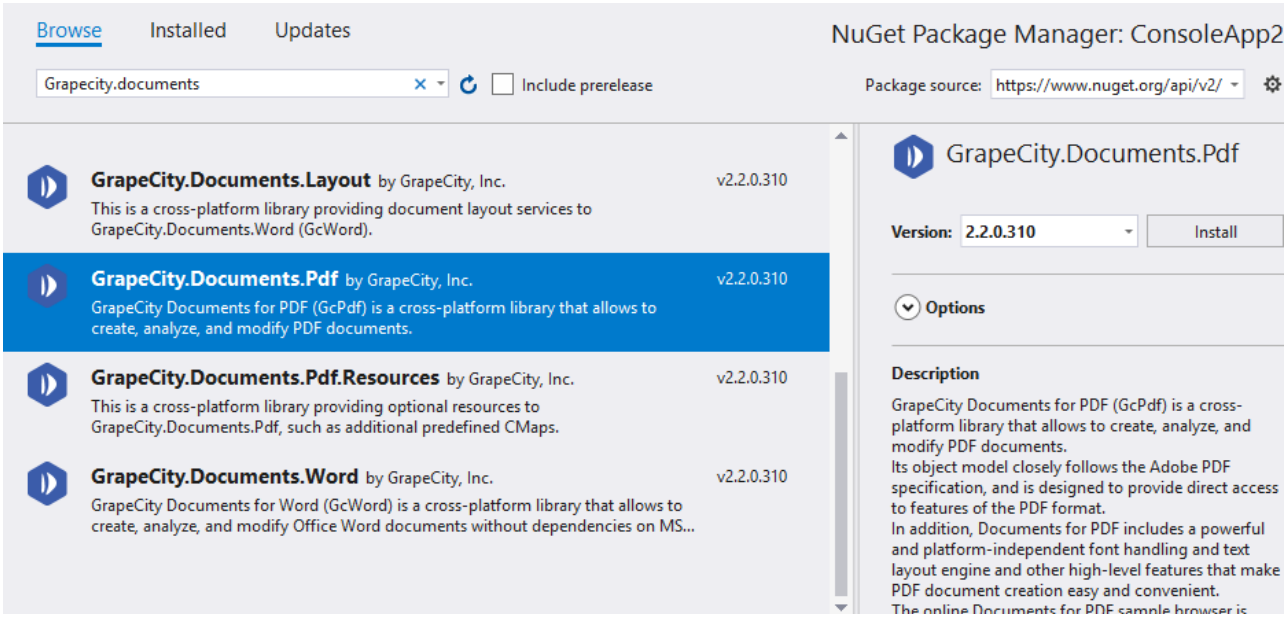
Add reference to GcPdf in your application from NuGet.org

In order to use GcPdf in a .NET Core, ASP.NET Core, .NET Framework application (any target that supports .NET Standard 2.0), install the NuGet packages in your application using the following steps:

Visual Studio for Windows

1. Open Visual Studio for Windows.
2. Create any application (any target that supports .NET Standard 2.0).
3. Right-click the project in Solution Explorer and choose **Manage NuGet Packages**.
4. In the **Package source** on top right, select **nuget.org**.
5. Click **Browse** tab on top left and search for "Grapecity.Documents".
6. On the left panel, select **GrapeCity.Documents.Pdf**

7. On the right panel, click **Install**.



8. In the **Preview Changes** dialog, click **OK** and choose **I Accept** in the next screen.
9. (Optional)If you want to add barcodes in your PDF file, you need to install the package **GrapeCity.Documents.Barcode** using the steps 5 to 8 above.

This adds all the required references of the package to your application. After this step, follow the steps in the Quick Start section.

Visual Studio for Mac

1. Open Visual Studio for MAC.
2. Create any application (any target that supports .NET Standard 2.0).
3. In tree view on the left, right-click **Dependencies** and choose **Add Packages**.
4. In the Search panel, type "GrapeCity.Documents".
5. From the list of packages displayed in the left panel, select **GrapeCity.Documents.Pdf** (and **GrapeCity.Documents.Barcode** if you want to render barcodes in your Pdfs) and click **Add Packages**.
6. Click **Accept**.

This automatically adds references of the package and its dependencies to your application. After this step, follow the steps in the Quick Start section.

Visual Studio Code for Linux

1. Open Visual Studio Code.
2. Install **Nuget Package Manager** from **Extensions**.
3. Create a folder "MyApp" in your **Home** folder.
4. In the Terminal in Visual Studio Code, type "`cd MyApp`"
5. Type command "`dotnet new console`"
   Observe: This creates a .NETCore application with MyApp.csproj file and Program.cs.
6. Press **Ctrl+P**. A command line opens at the top.
7. Type command: ">"
   Observe: "**Nuget Package Manager: Add Package**" option appears.
8. Click the above option.
9. Type "**Grapecity**" and press Enter.
   Observe: GrapeCity packages get displayed in the dropdown.

10. Choose **GrapeCity.Documents.Pdf**.
11. (Optional) Repeat above steps to add **GrapeCity.Documents.Barcode** if you want to add barcodes to your PDF. *Observe*: The packages would be added to your .csproj file.
12. Type following command in the Terminal window: "`dotnet restore`"

This adds references of the package to your application. After this step, follow the steps in the Quick Start section.

Quick Start

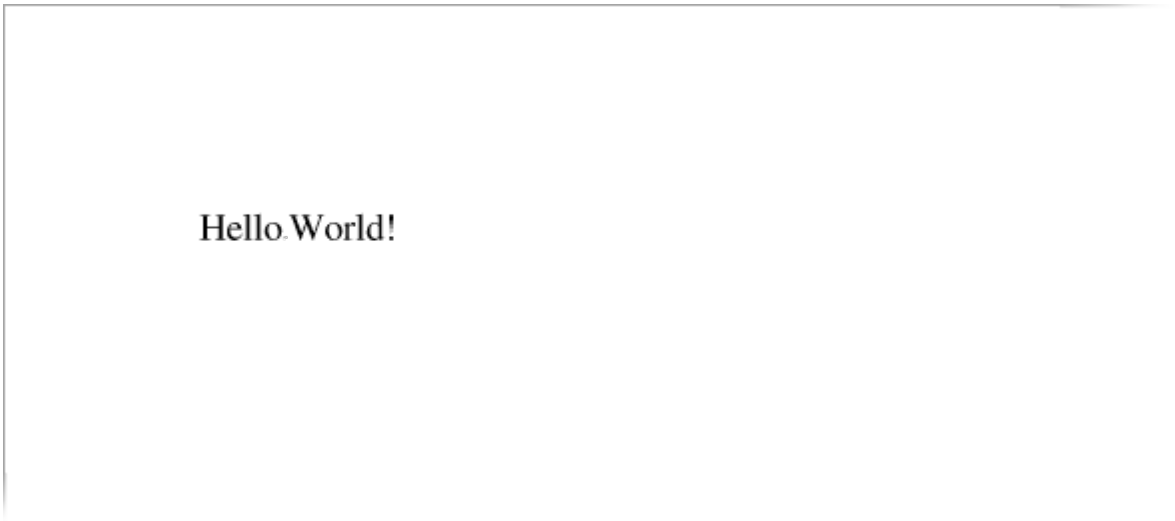
The following quick start sections help you in getting started with the GcPdf library:

- **Create and save a PDF document**
- **Load and modify a PDF document**

Create and save a PDF document

This quick start helps you in getting started with the GcPdf library. It covers how to create a simple PDF document having a single page and draw string on it in a specified font using a .NET Core or .NET Standard application. Follow the steps below to get started:

1. **Create a new PDF document**
2. **Draw a string on the PDF document**
3. **Save the PDF document**



Step 1: Create a new PDF document

1. Create a new application (.NET Core Console App\Windows Forms App) and add the references.
2. Include the following namespaces
   - using GrapeCity.Documents.Pdf;
   - using GrapeCity.Documents.Text;
3. Create a new PDF document using an instance of GcPdfDocument and define a text format for drawing a string, through code.

C#

```
// Create a new PDF document:
GcPdfDocument doc = new GcPdfDocument();
// Add a page, and get its Graphics object to draw on:
GcPdfGraphics g = doc.NewPage().Graphics;
```

```
// Create a text format for the "Hello World!" string:
TextFormat tf = new TextFormat();
// Use standard Times font
tf.Font = StandardFonts.Times;
// Pick a font size:
tf.FontSize = 14;
```

Back to Top

Step 2: Draw a string on the PDF document

Add the following code that uses DrawString method of GcGraphics class to draw string.

C#

```
// Draw the string at (1",1") from top/left of page
//(72 dpi is the default PDF graphics' resolution):
g.DrawString("Hello World!", tf, new PointF(72, 72));
```

Back to Top

Step 3: Save the document

Save the document using **Save** method of the GcPdfDocument class.

C#

```
//Save PDF document
doc.Save("filename.pdf");
```

Back to Top

Load and modify a PDF document

This quick start covers how to load an existing PDF document, modify and save it using a .NET Core or .NET Standard application. Follow the steps below to get started:

1. **Load an existing document in GcPdf**
2. **Modify the PDF document**
3. **Save the PDF document**

Step 1: Load an existing document in GcPdf

1. Create a new application (.NET Core Console App\Windows Forms App) and add the references.
2. Include the following namespace
   - using GrapeCity.Documents.Pdf;
3. Load an existing document using Load method of the GcPdfDocument class.

C#

```
GcPdfDocument doc = new GcPdfDocument();

//Create an object of filestream
var fs = new FileStream(Path.Combine("DocAttachment.pdf"), FileMode.Open,
             FileAccess.Read);
//Load the document
doc.Load(fs);
```

Back to Top

Step 2: Modify the document

1. Add a new page to the document using NewPage method of the GcPdfDocument class.

```
C#
//Add a new page in the document
GcPdfGraphics g = doc.NewPage().Graphics;
```

2. Add the following code that uses DrawString method of GcGraphics class to draw string.

```
C#
//Add text on the new page
g.DrawString("This is a newly added page in the modified document.", new
TextFormat()
{
      Font = StandardFonts.Times,
      FontSize = 12
}, new PointF(72, 72));
```

Back to Top

Step 3: Save the document

Save the document using **Save** method of the GcPdfDocument class.

```
C#
//Save the document
doc.Save("ModifiedDocument.pdf");
```

Back to Top

License Information

Types of Licenses

GrapeCity Documents for PDF supports the following types of license:

- **Unlicensed**
- **Evaluation License**
- **Licensed**

Unlicensed

After downloading the product, the product works in unlicensed mode. The following limitations are imposed when the product is used without license:

- Only 5 pages of the PDF file can be loaded for analyzing.
- On saving the PDF file, a watermark is displayed on all the pages in that file.
  'Unlicensed copy of GrapeCity PDF. Loading is limited to 5 pages. Contact us.sales@grapecity.com to get your 30-day evaluation key.'

> Note that if you run a sample that uses a signed PDF without a valid license key of GcPdf, the original signature in

> the generated PDF is invalidated. This happens because a license header is added to the PDF in such cases which changes the original signed document.

Evaluation License

GcPdf evaluation license is available to users for 30 days to evaluate the product. If you want to evaluate the product, you can ask for evaluation license key by sending an email to us.sales@grapecity.com.

The evaluation version has an expiration date that is determined when an evaluation key is generated. After applying the evaluation license key, you can use the complete product until the license expiry date.

After the expiry date, the product works in unlicensed mode with the above mentioned limitations.

In such case, following watermark is displayed in the PDF file:

'Created with expired evaluation copy of GrapeCity PDF. Loading is limited to 5 pages. Contact us.sales@grapecity.com to purchase license.'

> Note that if you run a sample that uses a signed PDF without a valid license key of GcPdf, the original signature in the generated PDF is invalidated. This happens because a license header is added to the PDF in such cases which changes the original signed document.

Licensed

GcPdf production license is issued at the time of purchase of the product. If you have a production license, you can access all the features of GcPdf without any limitations.

Apply License

To apply evaluation/production license in GcPdf, the long string key needs to be copied to the code in one of the following two ways.

- Pass it as an argument to the GcPdfDocument's ctor:
  `var doc = new GcPdfDocument("key")`
  This licenses the instance being created.

- Call a static method on GcPdfDocument:
  `GcPdfDocument.SetLicenseKey("key");`
  This licenses all the instances while the program is running.

Upgrade to Latest Version

To upgrade GcPdf license from v1 to v2, you will obtain a new license key from GrapeCity Sales team. This is a **free** upgrade for customers who purchased GcPdf v1 license. Once you have received a new license key through email, follow these steps:

1. Open an existing application created using GcPdf v1 license.
2. Right-click the project in **Solution Explorer** and choose **Manage NuGet Packages**.
3. In the **Package source** on top right, select **nuget.org**.
4. Click **Updates** tab on the top. A list of all the installed NuGet packages is displayed.
5. On the left panel, select the **Select all packages** checkbox and click **Update**.
6. In the **Preview Changes** dialog, click **OK** and choose **I Accept** in the next screen.
7. Switch to the code view and replace the old key with new v2 key received through email.
   - To upgrade the license of a particular instance:
     `var doc = new GcPdfDocument("new key")`
   - To upgrade the license of all the instances:

```
GcPdfDocument.SetLicenseKey("new key");
```

Technical Support

If you have a technical question about this product, consult the following source:

- Product forum: https://www.grapecity.com/forums
- Email: us.sales@grapecity.com

Redistribution

In order to distribute the application, make sure you meet the installation criteria specified in the System Requirements page in this documentation. Further, you also need to have a valid Distribution License to successfully distribute the application.

> GcPdf makes it easy to deploy your application to your local servers or cloud offerings such as Azure.

For more information about Distribution License, contact our Sales department using one of these methods:

World Wide Web site	https://www.grapecity.com/
E-mail	us.sales@grapecity.com
Phone	1.800.858.2739 or 412.681.4343
Fax	(412) 681-4384

End-User License Agreement

The GrapeCity licensing information, including the GrapeCity end-user license agreement, frequently asked licensing questions, and the GrapeCity licensing model, is available online. For detailed information on licensing, see GrapeCity Licensing. For GrapeCity end-user license agreement, see End-User License Agreement For GrapeCity Software.

Product Architecture

Packaging

GcPdf is a collection of .NET Standard 2.0 class libraries written in C#, providing an API that allows to create PDF files from scratch and to load, analyze and modify existing documents.

GcPdf works on all platforms supported by .NET Standard, including .NET Core, ASP.NET Core, .NET Framework and so on.

GcPdf and supporting packages are available on nuget.org:

- GrapeCity.Documents.Pdf
- GrapeCity.Documents.BarCode
- GrapeCity.Documents.Common
- GrapeCity.Documents.Common.Windows
- GrapeCity.Documents.DX.Windows

To use GcPdf in an application, you need to reference just the **Grapecity.Documents.Pdf** package. It pulls in the required infrastructure packages.

To render barcodes, install the **GrapeCity.Documents.Barcode** package (**GcBarcode** for short). It provides extension methods allowing to draw barcodes when using GcPdf.

On a Windows system, you can optionally install **Grapecity.Documents.Common.Windows.** It provides support for font linking specified in the Windows registry. This library can be referenced on a non-Windows system, but does nothing.

GrapeCity.Documents.Common is an infrastructure package used by GcPdf and GcBarcode.
GrapeCity.Documents.DX.Windows provides access to the native imaging APIs to GcPdf if it runs on a Windows system.

GcPdf API Overview

Classes and other types in the GcPdf and related libraries expose a PDF object model that closely follows the Adobe PDF specification version 1.7 published by Adobe. GcPdf is designed to provide, whenever feasible, direct access to all features of the PDF format, including the low-level features. In addition, GcPdf provides a powerful and platform-independent text layout engine and some other high-level features that make document creation using GcPdf easy and convenient.

Namespaces

Namespaces	Description
GrapeCity.Documents.Common	Infrastructure and utility types (including fonts support).
GrapeCity.Documents.Drawing	Framework for drawing on the abstract GcGraphics surface.
GrapeCity.Documents.Pdf	Types used to create, process and modify PDF documents includes GcPdfGraphics). Nested namespaces contain types supporting specific PDF spec areas: • GrapeCity.Documents.Pdf.AcroForms • GrapeCity.Documents.Pdf.Actions • GrapeCity.Documents.Pdf.Annotations • GrapeCity.Documents.Pdf.Graphics • GrapeCity.Documents.Pdf.Log • GrapeCity.Documents.Pdf.Parser

	• GrapeCity.Documents.Pdf.Security
GrapeCity.Documents.Text	Text processing sub-system.

GcPdfDocument

A PDF document in GcPdf is represented by an instance of the GrapeCity.Documents.Pdf.GcPdfDocument class. To create a new PDF, create an instance of GcPdfDocument, add content to it and then call one of the GcPdfDocument.Save() overloads to write the document to a file. Save() method can be called multiple times on an instance of GcPdfDocument, so that many (possibly different) PDF documents can be created.

GcPdfDocument also provides a Load() method, allowing to analyze and/or modify an existing PDF. When Load() method is called on an instance of GcPdfDocument, the instance is cleared first. It is important to note that the Load() method accepts a Stream that is opened by the caller on the PDF which is loaded, and the stream must be readable and must be kept open for the duration of working with the loaded document. This is because Load() method does not actually load the whole document into memory, rather it loads the required parts on demand, which keeps the memory footprint to a minimum and improves performance. Note that Load() is a "read-only" method. GcPdfDocument does not try to write back to the loaded stream - In order to save any changes made to the document, Save() method must be called, specifying the output file or stream as a newly created document.

A number of properties and collections on the GcPdfDocument provide access to the content and properties of the document. The most important collection is Pages (see **The Pages Collection**), others include Outlines, AcroForm, Security and so on.

The Pages Collection

The **Pages** collection represents the collection of a document's pages. When a new GcPdfDocument is created, this collection is initially empty. The usual collection modifying methods are available and can be used to fetch, add, insert, remove or move pages around. When an existing PDF is loaded into a GcPdfDocument, the Pages collection is filled with the pages loaded from that document. It can then be modified in the same way as in a document created from scratch.

Modifying Existing Documents

Using the GcPdfDocument.Load() method, existing documents can be inspected and modified. The possible modifications include:

- Changing the writable properties of the loaded document and its elements.
- Adding arbitrary new content. Anything that can be added to a new document, can also be added to a loaded one: pages, page content, annotations, fields and so on.
- Modifying collections on the document and document pages. Elements of the following collections can be moved around, removed or added:
  - At the document level:
    - Pages
    - NamedDestinations
    - Outlines
    - AcroForm.Fields
  - At the page level:
    - ContentStreams
    - Annotations

No other modifications are supported at this time. For example, it is currently not possible to replace existing text or graphics, except by removing existing and adding new content streams.

It should be noted again that when an existing document is loaded into a **GcPdfDocument** instance, the connection with the original document is read-only, i.e. content is fetched as needed from the underlying stream, but no attempt is made to write back the changes. The GcPdfDocument.Save() method should be called if preserving the changes is

required.

Sequential (StartDoc/EndDoc) Mode

In addition to the Save() method mentioned above, GcPdfDocument provides a sequential mode for creating a PDF. To use this mode, start by calling the StartDoc() method on the document, specifying a writable Stream as the method's only parameter. After that content can be added to the document as usual, but with following limitations. When done, call the EndDoc() method which completes writing the document.

The limitations of the sequential method are as follows:

- The only allowed modification of the Pages collection is adding a page to the end of it. Removing, inserting or moving pages is not allowed.
- You can only draw on the last page of the Pages collection. Once another page has been added after it, modifying any of the preceding pages is not allowed.
- Certain features (e.g. linearization) are not available in this mode.

The advantage of the sequential mode is that the pages of the document are written to the underlying stream as soon as they are completed, so especially if creating a very large PDF the memory footprint can be much smaller.

Text

Text measuring and layout is supported by a specialized set of classes in the **GrapeCity.Documents.Text** namespace. These classes provide a rich object model that includes, and allows access to text elements from high-level (paragraphs) all the way down to the lowest levels, such as individual font and glyph features. Text processing is completely platform-independent and does not rely on any operating system-provided APIs.

The most important class in the GrapeCity.Documents.Text namespace is TextLayout, it represents one or more paragraphs of text, and supports the following features:

- Layout of paragraphs in an arbitrary rectangular area using a specified text flow direction
- Line wrapping according to the Unicode standard recommendations
- OpenType, TrueType and WOFF fonts, including extensions for handling national languages
- Individual formatting of text fragments using different fonts, font styles and colors (see TextFormat class)
- Typography features such as tabs, text alignment, char and line spacing, etc.
- Text flow around rectangular areas
- Inline and anchored objects
- Kashida text justification in Arabic scripts
- Splitting of large bodies of text into several layouts (columns or pages), including support for column balancing and control over widow/orphan lines

All features are fully supported for vertical (Chinese or Japanese) and RTL/bidirectional text.

After a text has been added to, and processed by, an instance of the TextLayout class, a representation of the text is generated using the glyphs from the specified fonts, and coordinates of any fragment of the original text in the generated layout can be fetched, if necessary.

A **TextLayout** instance can also be directly rendered onto GcGraphics (see **Graphics**) using the DrawTextLayout method. Simple MeasureString/DrawString methods on GcGraphics are also provided for convenience.

Graphics

GcPdf provides a graphics surface to draw on, represented by a GcPdfGraphics class, which is an implementation of the abstract GcGraphics base class. GcPdfGraphics provides a flexible and rich object model for measuring, stroking, and filling the usual graphic primitives such as lines, rectangles, polygons, ellipses and so on. Drawing (Stroking) can be done with solid or dashed lines, shapes can be filled with solid, or gradient brushes. For an example of shape rendering methods, see **GcPdfGraphics.DrawEllipse()** or **GcPdfGraphics.FillEllipse()** method. Complex shapes can be created and rendered using graphic paths. For example, see **GcPdfGraphics.DrawPath()** method.

Graphics transformations using 3x2 matrices are fully supported (including text). For more information, see **GcPdfGraphics.Transform()** method.

Units of Measurement

The default units of measurement used by GcPdfGraphics and TextLayout are printer points (1/72 of an inch). If desired, these can be changed to an arbitrary resolution using the **Resolution** property available on both **GcPdfGraphics** and **TextLayout** classes.

Coordinates

Coordinates of all graphic objects are measured from the top left corner of the graphics surface (which in GcPdfGraphics is usually a page). **GcPdfGraphics.Transform** can be used to change that.

Page Graphics

To draw on a page in a PDF document, an instance of **GcPdfGraphics** for that page must be used. Each page in the **GcPdfDocument.Pages** collection has the **Graphics** property that fetches the graphics for that page. You can simply get that property and draw on the returned graphics instance. Initially each page has just one graphics associated with it. But if the page contains multiple context streams, each context stream will have its own graphics, and the Page.Graphics property will return the graphics of the last (top-most) content stream. (All content streams of the page can be accessed via its ContentStreams collection.)

Features

This section comprises the features available in the GcPdf.

Attachment
: Work with the document and file attachments in GcPdf.

Annotations
: Add, get, modify, and delete annotations from a page.

Document
: Work with document properties and merge documents.

Font
: Work with fonts, font collections, and font embedding.

Forms
: Create AcroForms and add, modify, and delete form fields.

Form XObjects
: Work with Form XObjects.

Graphics
: Add shapes, fill them, and use gradient and transformation on a page.

Images
: Add images, adjust their scalability and extract images.

Incremental Update
: Update a document incrementally and sign an already signed PDF.

Linearization
: Generate linearized PDF.

Links
: Add hyperlinks.

Outline
: Add, get, modify, and delete document outlines from a page.

Pages
: Insert a page in a PDF, get a particular page, set it's orientation and size, and work with content streams.

Security
: Encrypt PDF, set permissions, and add or remove digital signature.

Soft Mask
: Create soft mask in a PDF document.

Stamps
: Add, modify, and delete stamps.

Tagged PDF
: Create tagged PDF.

Text
: Work with text along with paragraph handling.

Text Search
: Perform text search in a PDF document.

Watermark
: Add watermark.

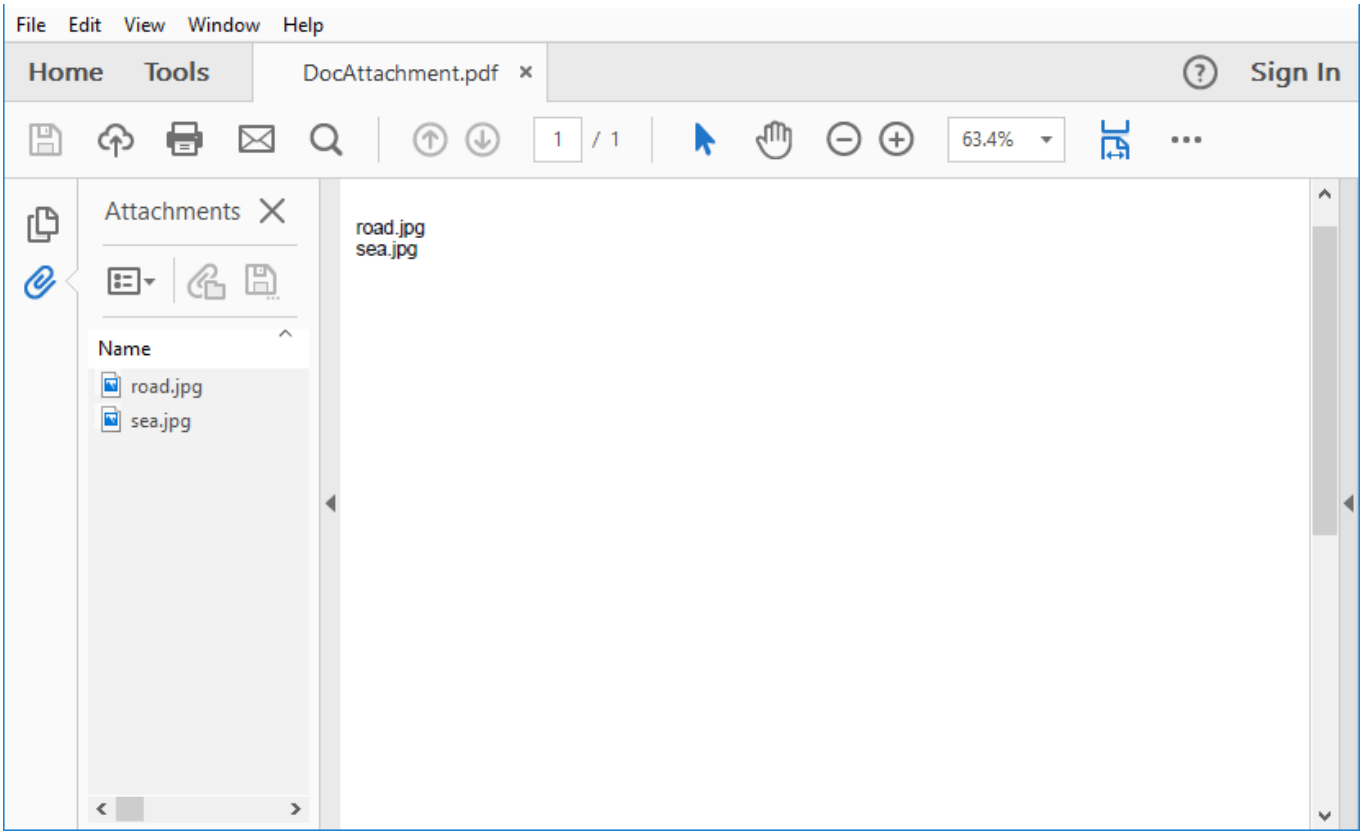
Attachment

Attachments contain reference to documents or files which are embedded in a PDF document. The content of these external files can be referred by a PDF using file specification which is represented by FileSpecification class in GcPdf. The file specification refers to an embedded file within the referring PDF file which allows the file contents to be stored or transmitted along with the PDF document. When a Pdf file containing file specification that refers to a external file is transmitted, it needs to be ensured that the references remain valid. This can be handled by the

embedded file streams which are represented by the EmbeddedFileStream class in GcPdf. The embedded file stream allows the content of the referenced files to be embedded directly within the PDF file. For more information on file specification and embedded file streams, see PDF specification 1.7 (Section 7.11.1 and 7.11.4).

Document Attachment

An attachment which is attached to a PDF document at the document level is a document attachment. GcPdf allows you to embed the files in a PDF document and refer to them through file specifications. These files are attached to the PDF document using the Add method.



To attach files to a PDF document at document level:

1. Create a variable of type string to store the path of the files to be attached.
2. Create an object of FileSpecification class to refer to the embedded file.
3. Add the attachments to the document using the Add method.

C#

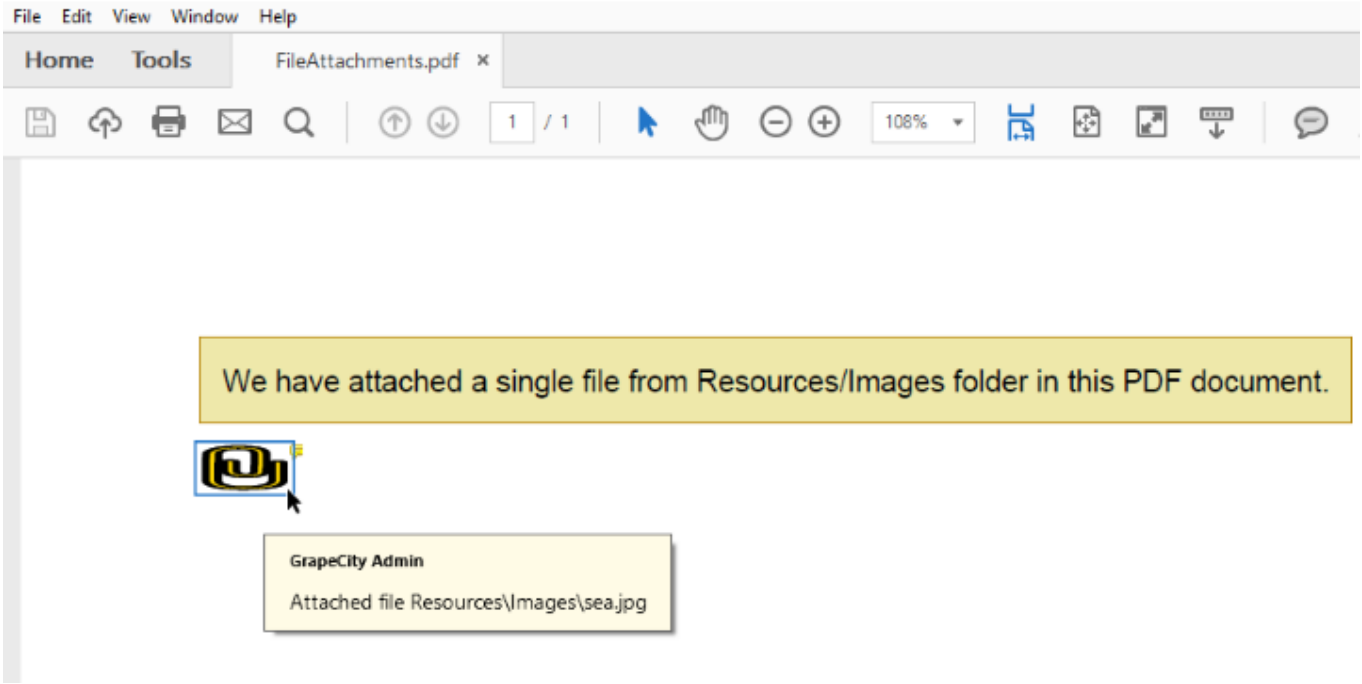
```
GcPdfDocument doc = new GcPdfDocument();
    Page page = doc.NewPage();

    string[] files = new string[]
    {
    "road.jpg",
    "sea.jpg"
    };

StringBuilder sb = new StringBuilder();
    foreach (var fn in files)
        sb.AppendLine(fn);
```

```
//Add string related to the attachment names
page.Graphics.DrawString(sb.ToString() , new TextFormat(), new PointF(10, 50));

//Add attachments
    foreach (string fn in files)
    {
        string file = Path.Combine("Resources", fn);
        FileSpecification fspec = FileSpecification.FromEmbeddedFile(
                                  EmbeddedFileStream.FromFile(doc, file));
        doc.EmbeddedFiles.Add(file, fspec);
    }
    //Save the document
    doc.Save("DocAttachment.pdf");
```

Back to Top

File Attachment

File attachment in a PDF document is attached on a page and is displayed as a link that jumps to the attached file on clicking the drawn graphics. GcPdf allows you to attach files to a PDF using the FileAttachmentAnnotation class. This class also allows you to set the icon to display the attachment using Icon property which accepts value from the FileAttachmentAnnotationIcon enum.



To add an attachment to a PDF document on a page:

1. Create an object of GcPdfDocument and FileAttachmentAnnotation class.
2. Set the required properties of FileAttachmentAnnotation object.
3. Call the Add method to add the file attachment.

C#

```
public void CreatePDF(Stream stream)
{
      var doc = new GcPdfDocument();
```

```
        var page = doc.NewPage();
        var g = page.Graphics;

        var rc = Common.Util.AddNote("We have attached a single file from" +
            "Resources/Images folder in this PDF document.", page);
        var ip = new PointF(rc.X, rc.Bottom + 9);
        var attSize = new SizeF(36, 18);

        string file = Path.Combine("Resources", "Images", "sea.jpg");
        FileAttachmentAnnotation faa = new FileAttachmentAnnotation()
        {
              Color = Color.Gold,
              UserName = "GrapeCity Admin",
              Rect = new RectangleF(ip.X, ip.Y, attSize.Width, attSize.Height),
              Contents = $"Attached file {file}",
              Icon = FileAttachmentAnnotationIcon.Paperclip,
              File = FileSpecification.FromEmbeddedFile(EmbeddedFileStream.FromFile(doc,
file)),
        };
        page.Annotations.Add(faa);

        // Done:
        doc.Save(stream);
}
```

Back to Top

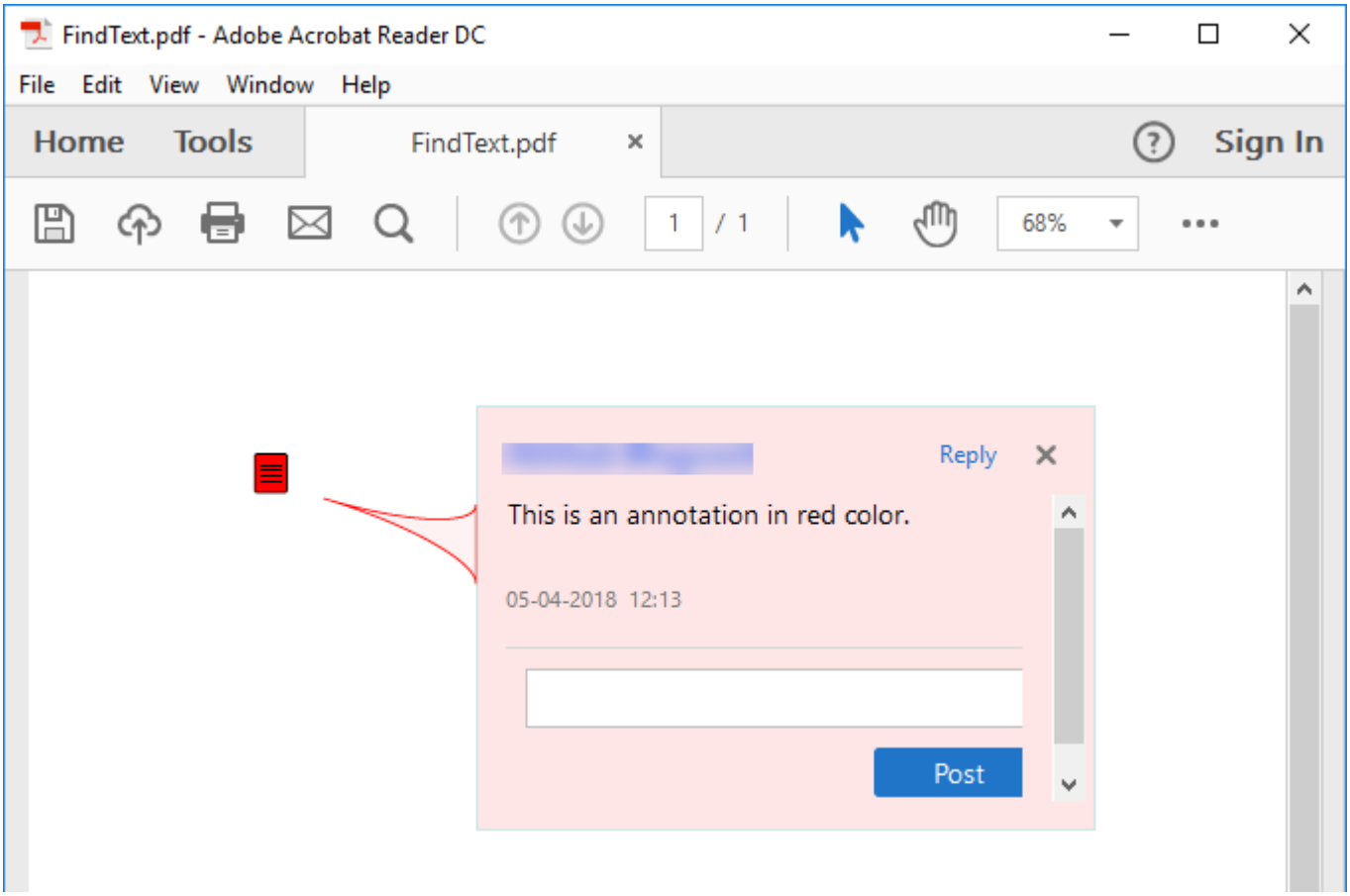
For more information about how to work with file attachments using GcPdf, see GcPdf sample browser.

Annotations

An annotation is used to mark or highlight texts, images and other visual elements on a page. Annotations can be text, image, shape, sound or even file attachments. The purpose of using annotation is to simply associate information or a note with an item on a page. A number of annotations can be displayed either in open or closed state. In the closed state, they appear on the page as a note, icon, or a box, depending on the annotation type. In the opened state, these annotations display the associated object such as a pop-up window containing associated text. For more information on annotations and its types, see PDF specification 1.7 (Section 12.5).

GcPdf offers a variety of standard annotation types. It is listed in the the topic Annotation Types.

All the listed annotations have a dedicated class and properties in the GcPdf library which makes it easier to implement different annotations. GcPdf also allows you to specify various characteristics of annotation such as visibility, printing, etc. using **Flags** property that accepts the values from AnnotationFlags enum.



Add Annotations

GcPdf allows you to add annotations to a page in the PDF document. These annotations reside in the Page object on which they are placed.

To add an annotation on a page:

1. Create an instance of class corresponding to annotation type you want to add to a page, for example, TextAnnotation class.
2. Call the Add method to add the annotation on the page.

C#

```
var textAnnot = new TextAnnotation()
{
    Contents = "This is an annotation in red color.",
    Name = "Text Annotation",
    Rect = new RectangleF(72, 72, 72 * 2, 72),
    Color = Color.Red,
};
//Add the text annotation
page.Annotations.Add(textAnnot);
```

Back to Top

Get Annotations

To get the annotations from a page:

1. Create an instance of the AnnotationCollection class.
2. Use the AnnotationCollection object to access the annotation by specifying its index.

C#

```
//Get Annotation
AnnotationCollection acol = doc.Pages[0].Annotations;
// Display the property values
Console.WriteLine("Annotation Type: {0}", acol[0].Name);
```

Back to Top

Modify Annotations

To modify the annotation, you can set the properties of the type of annotation you used on a page. For instance, setting Contents and Color property of the TextAnnotation class modifies the existing content and color of the annotation.

C#

```
//Modify annotation
textAnnot.Color = Color.BlueViolet;
textAnnot.Contents = "This is a Text annotation.";
```

Back to Top

Delete Annotations

To delete all the annotations from a page, use the Clear method. Apart from this, RemoveAt method can be used to remove a particular annotation by specifying its index value.

C#

```
// Delete all annotations
page.Annotations.Clear();

// Delete a particular annotation
page.Annotations.RemoveAt(0);
```

Back to Top

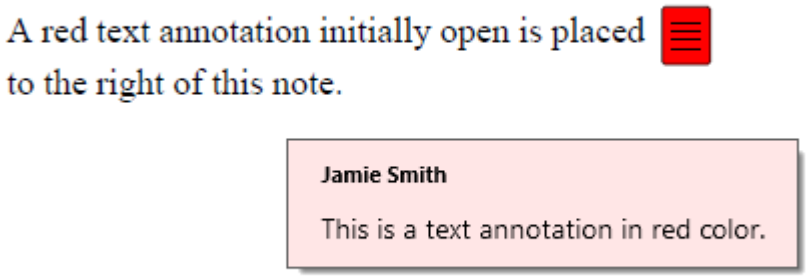
For more information about how to implement annotations using GcPdf, see GcPdf sample browser.

Annotation Types

GcPdf supports various types of annotation standardized by Adobe. The following section describes different types of annotations and their implementation.

Text Annotation

Text annotation represents a sticky note attached to a point in a PDF file. Upon closing, the annotation appears as an icon, and upon opening, it displays a pop-up window with the text of the note, in a size and font as selected by the viewer application. GcPdf provides TextAnnotation class to enable the users to apply text annotations in the PDF document.



The following code illustrates how to add a text annotation to a PDF document.

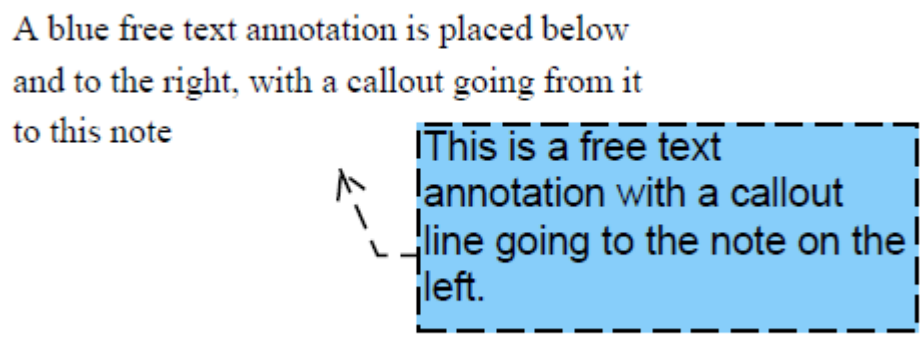
C#

```
public void CreateTextAnnotation()
{
    GcPdfDocument doc = new GcPdfDocument();
    Page page = doc.NewPage();
    RectangleF rc = new RectangleF(50, 50, 200, 50);
    page.Graphics.DrawString("A red text annotation initially open is placed to the
right of this note.", new TextFormat() { Font = StandardFonts.Times, FontSize=11
},rc);

    //Create an instance of TextAnnotation class and set its relevant properties
    var textAnnot = new TextAnnotation()
    {
        UserName = "Jamie Smith",
        Contents = "This is a text annotation in red color.",
        Rect = new RectangleF(rc.Right, rc.Top, 72 * 2, 72),
        Color = Color.Red,
        Open=true
    };

    page.Annotations.Add(textAnnot); //Add the text annotation
    doc.Save("TextAnnotation.pdf");
}
```

Back to Top

Free Text Annotation

A free text annotation displays text directly on the page. Unlike text annotations, the free text annotations do not have an open or closed state. The text remains visible instead of being displayed in a pop-up window. GcPdf provides FreeTextAnnotation class to enable the users to apply free text annotations to the PDF file.



The following code illustrates how to add a free text annotation to a PDF document.

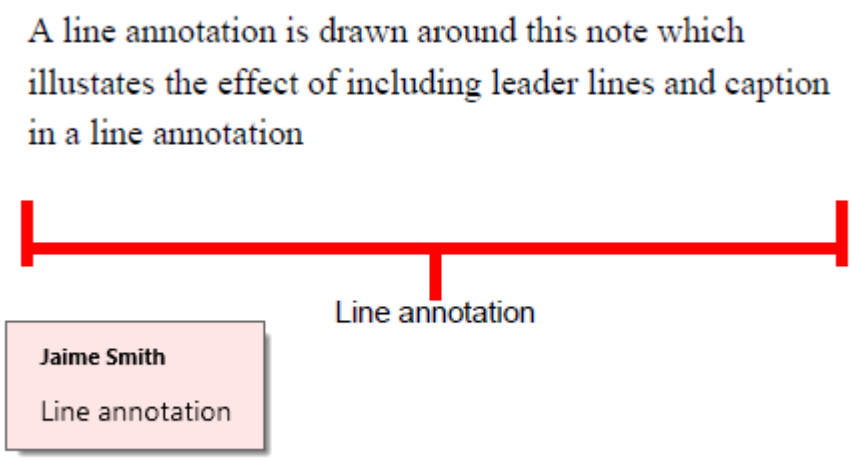
C#

```
public void CreateFreeTextAnnotation()
{
    GcPdfDocument doc = new GcPdfDocument();
    Page page = doc.NewPage();
    RectangleF rc = new RectangleF(50, 50, 200, 50);
    page.Graphics.DrawString("A blue free text annotation is placed below and to the
right, with a callout going from it to this note", new TextFormat() { Font =
StandardFonts.Times, FontSize = 11 }, rc);

    //Create an instance of FreeTextAnnotation class and set its relevant properties
    var freeAnnot = new FreeTextAnnotation()
    {
        Rect = new RectangleF(rc.Right + 18, rc.Bottom + 9, 72 * 2, 72),
        CalloutLine = new PointF[]
       {
            new PointF(rc.Left + rc.Width / 2, rc.Bottom),
            new PointF(rc.Left + rc.Width / 2, rc.Bottom + 9 + 36),
            new PointF(rc.Right + 18, rc.Bottom + 9 + 36),
       },
        LineWidth = 1,
        LineEndStyle = LineEndingStyle.OpenArrow,
        LineDashPattern = new float[] { 8, 4 },
        Contents = "This is a free text annotation with a callout line going to the
note on the left.",
        Color = Color.LightSkyBlue,
    };

    page.Annotations.Add(freeAnnot); //Add the free text annotation
    doc.Save("FreeTextAnnotation.pdf");
}
```

Back to Top

Line Annotation

A line annotation displays a single straight line on the page. Upon opening, the annotation displays a pop-up window containing the associated note. GcPdf provides LineAnnotation class to enable the users to apply line annotations to the PDF file.



The following code illustrates how to add a line annotation to a PDF document.

C#

```
public void CreateLineAnnotation()
{
    GcPdfDocument doc = new GcPdfDocument();
    Page page = doc.NewPage();
    RectangleF rc = new RectangleF(50, 50, 250, 50);
    page.Graphics.DrawString("A line annotation is drawn around this note which
illustates the effect of including leader lines and caption in a line annotation",
new TextFormat() { Font = StandardFonts.Times, FontSize = 11 }, rc);

    //Create an instance of LineAnnotation class and set its relevant properties
    var lineAnnot = new LineAnnotation()
    {
        UserName = "Jaime Smith",
        Start = new PointF(rc.X, rc.Bottom),
        End = new PointF(rc.Right, rc.Bottom),
        LineWidth = 3,
        Color = Color.Red,
        LeaderLinesLength = -15,
        LeaderLinesExtension = 5,
        LeaderLineOffset = 10,
        Contents = "Line annotation",
        VerticalTextOffset = -20,
        TextPosition = LineAnnotationTextPosition.Inline,
    };
```

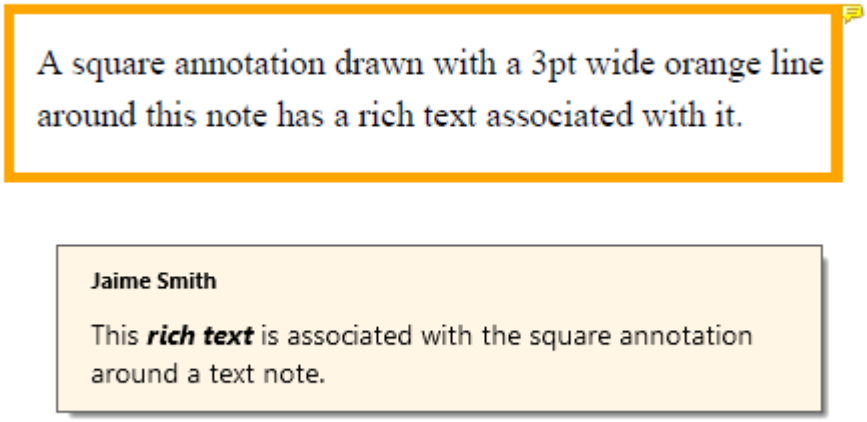
```
    page.Annotations.Add(lineAnnot); //Add the square annotation
    doc.Save("LineAnnotation.pdf");
}
```

Back to Top

Square Annotation

A square annotation displays a rectangle/square on the page. When opened, the annotation displays a pop-up window with the text of the associated note. GcPdf provides SquareAnnotation class to enable the users to apply square annotations to the PDF file.

Note that square annotation need not always imply that the annotation is square in shape. The height and width of the annotation may vary. The image given below depicts a rectangle-shaped square annotation.



The following code illustrates how to add a square annotation to a PDF document.

C#

```
public void CreateSquareAnnotation()
{
    GcPdfDocument doc = new GcPdfDocument();
    Page page = doc.NewPage();
    RectangleF rc = new RectangleF(50, 50, 250, 50);
    page.Graphics.DrawString("A square annotation drawn with a 3pt wide orange line
around this note has a rich text associated with it.", new TextFormat() { Font =
StandardFonts.Times, FontSize = 11 }, rc);
    rc.Inflate(10, 10);

    //Create an instance of SquareAnnotation class and set its relevant properties
    var squareAnnot = new SquareAnnotation()
    {
        UserName = "Jaime Smith",
        Rect = rc,
        LineWidth = 3,
        Color = Color.Orange,
```

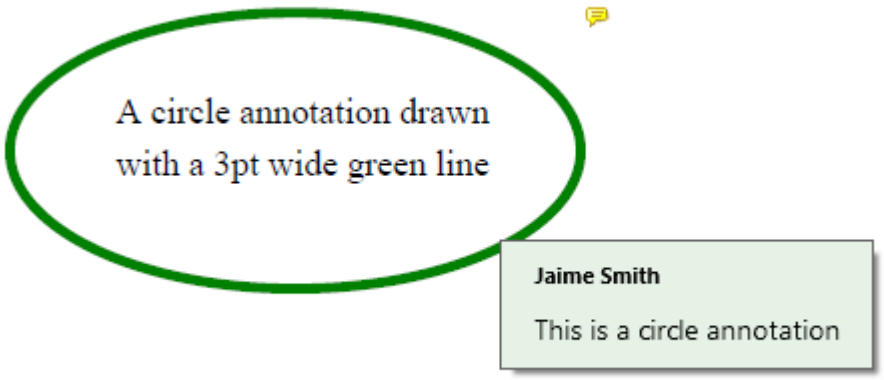
```
        RichText =
            "<body><p>This <b><i>rich text</i></b> is associated with the square
annotation around a text note.</p></body>"
    };
    page.Annotations.Add(squareAnnot); //Add the square annotation
    doc.Save("SquareAnnotation.pdf");
}
```

Back to Top

Circle Annotation

A circle annotation displays an ellipse/circle on a page. When open, the annotation displays a pop-up window with the text of the associated note. GcPdf provides CircleAnnotation class to enable the users to apply circle annotations to the PDF file.

Note that circle annotation need not always imply that the annotation is circular in shape. The height and width of the annotation may vary. The image given below depicts an ellipse-shaped circle annotation.



The following code illustrates how to add a circle annotation to a PDF document.

C#

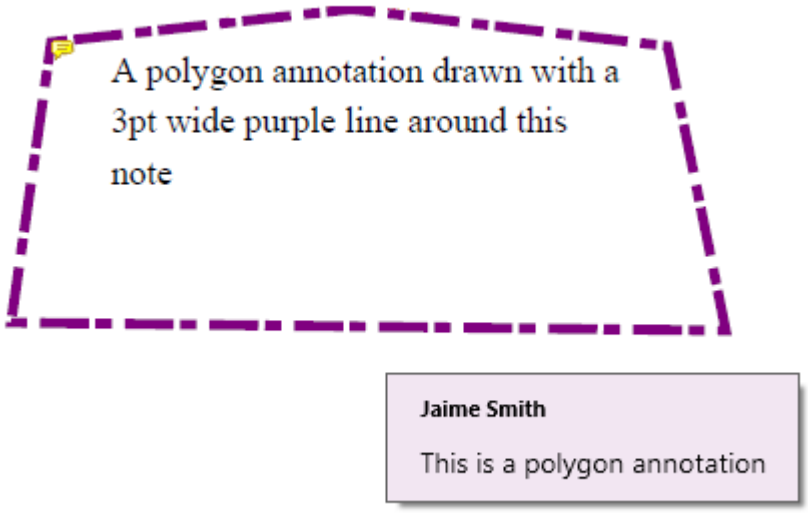
```
public void CreateCircleAnnotation()
{
    GcPdfDocument doc = new GcPdfDocument();
    Page page = doc.NewPage();
    RectangleF rc = new RectangleF(50, 50, 120, 50);
    page.Graphics.DrawString("A circle annotation drawn with a 3pt wide green line",
new TextFormat() { Font = StandardFonts.Times, FontSize = 11 }, rc);
    rc.Inflate(15, 24);

    //Create an instance of CircleAnnotation class and set its relevant properties
    var circleAnnot = new CircleAnnotation()
    {
        UserName = "Jaime Smith",
        Rect = rc,
```

```
        LineWidth = 3,
        Color = Color.Green,
        Contents = "This is a circle annotation",
    };

    page.Annotations.Add(circleAnnot); //Add the circle annotation
    doc.Save("CircleAnnotation.pdf");
}
```

Back to Top

Polygon Annotation

A polygon annotation displays a polygon on a page. On opening the annotation, it displays a pop-up window containing the text of the associated note. GcPdf provides PolygonAnnotation class to enable the users to apply polygon annotations to the PDF file.



The following code illustrates how to add a polygon annotation to a PDF document.

C#

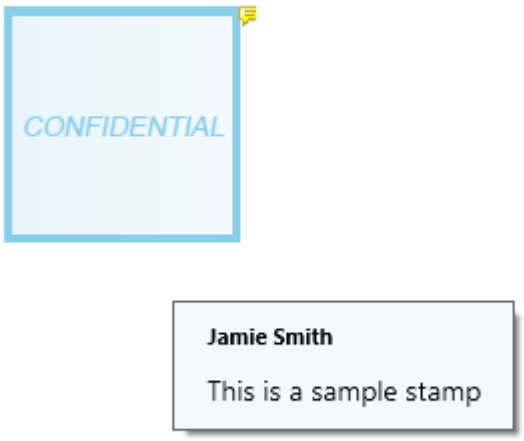
```
public void CreatePolygonAnnotation()
{
    GcPdfDocument doc = new GcPdfDocument();
    Page page = doc.NewPage();
    RectangleF rc = new RectangleF(140, 30, 160, 70);
    page.Graphics.DrawString("A polygon annotation drawn with a 3pt wide purple line around this note", new TextFormat() { Font = StandardFonts.Times, FontSize = 11 }, rc);

    //Create an instance of PolygonAnnotation class and set its relevant properties
    var polygonAnnot = new PolygonAnnotation()
    {
```

```
        Points =new List<PointF>()
        {
            new PointF(rc.X-5, rc.Y),
            new PointF(rc.X+75, rc.Y-10),
            new PointF(rc.X+rc.Width-5, rc.Y),
            new PointF(rc.X+rc.Width-5, rc.Y+rc.Height),
            new PointF(rc.X-5, rc.Y+rc.Height),
        },
        UserName = "Jaime Smith",
        LineWidth = 3,
        LineDashPattern = new float[]{ 5, 2, 15, 4},
        Color = Color.Purple,
        Contents = "This is a polygon annotation",
    };

    page.Annotations.Add(polygonAnnot); //Add the polygon annotation
    doc.Save("PolygonAnnotation.pdf");
}
```

Back to Top

Stamp Annotation

A stamp annotation displays graphics, images or texts to look as if they were stamped on a page. Upon opening, the stamp annotations display a pop-up window with the text of the associated note. GcPdf provides StampAnnotation class to enable the users to apply stamp annotations to the PDF file.



The following code illustrates how to add a stamp annotation to a PDF document.

C#

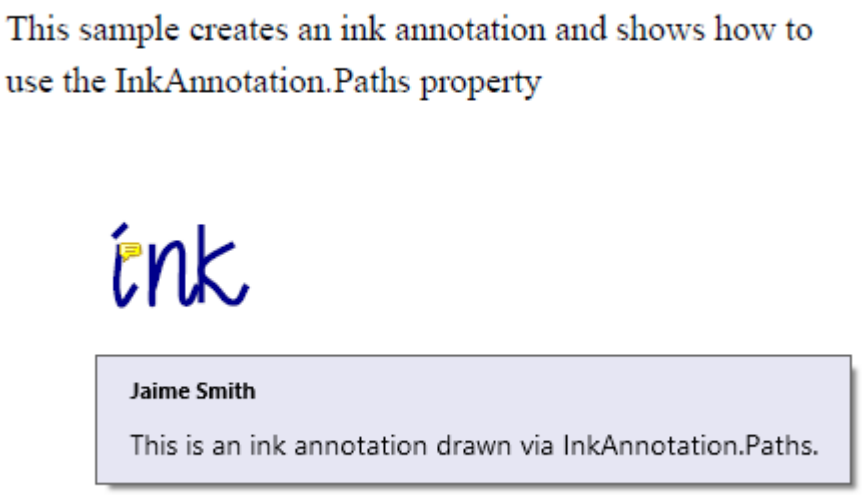
```
public void CreateStampAnnotation()
{
    GcPdfDocument doc = new GcPdfDocument();
```

```
    var page = doc.NewPage();

    //Create an instance of StampAnnotation class and set its relevant properties
    var stamp = new StampAnnotation()
    {
        Contents = "This is a sample stamp",
        UserName = "Jamie Smith",
        Color = Color.SkyBlue,
        Icon = StampAnnotationIcon.Confidential.ToString(),
        CreationDate = DateTime.Today,
        Rect = new RectangleF(100.5F, 110.5F, 72, 72),
    };

    page.Annotations.Add(stamp); //Add the stamp annotation
    doc.Save("StampAnnotation.pdf");
}
```

Back to Top

Ink Annotation

An ink annotation represents a freehand scribble composed of one or more disjoint paths. When an ink annotation is opened, it displays a pop-up window containing the text of the related note. GcPdf provides InkAnnotation class to enable the users to apply ink annotations to the PDF file.



The following code illustrates how to add an ink annotation to a PDF document.

C#

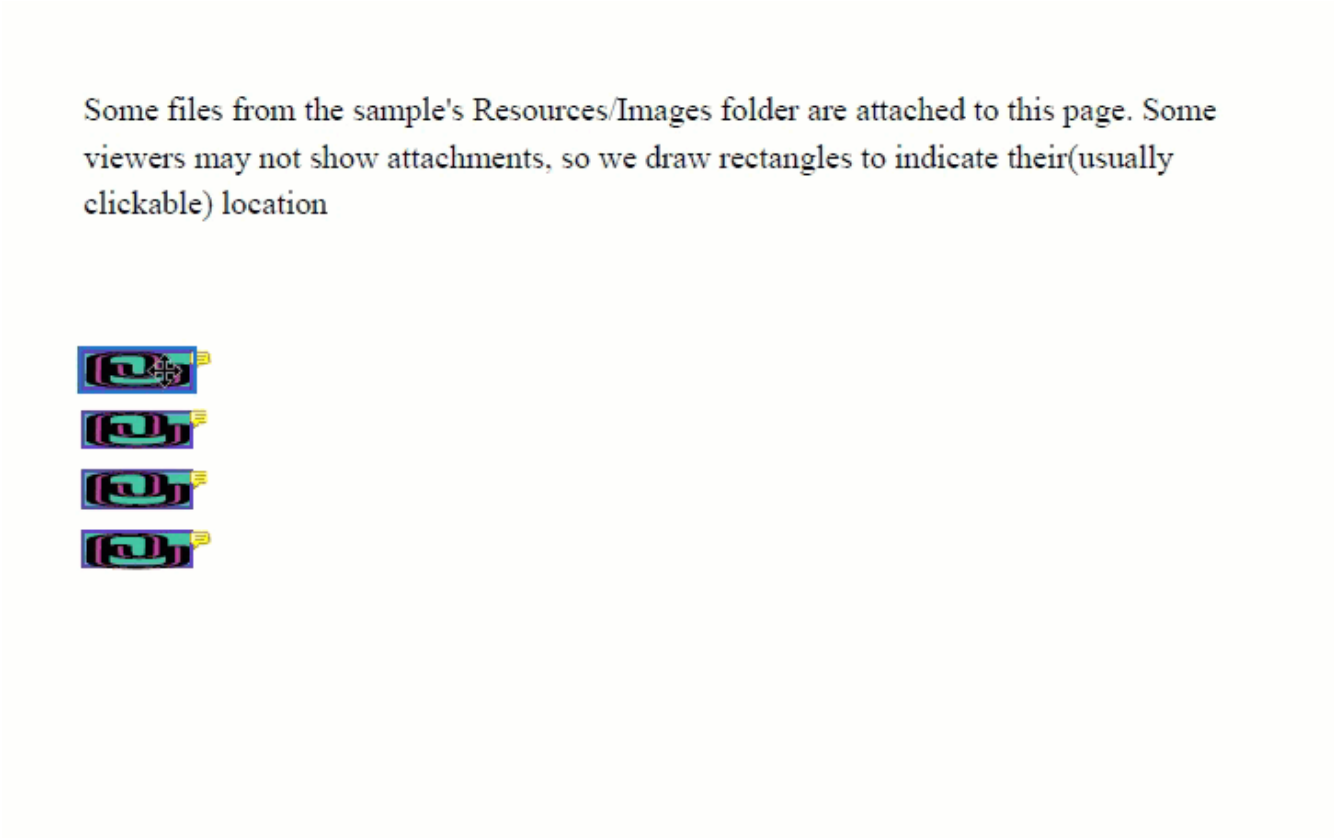
```
public void CreateInkAnnotation()
{
    var doc = new GcPdfDocument();
    var page = doc.NewPage();
    RectangleF rc = new RectangleF(50, 50, 250, 50);
```

```
    page.Graphics.DrawString("This sample creates an ink annotation and shows how to
use the InkAnnotation.Paths property", new TextFormat() { Font = StandardFonts.Times,
FontSize = 11 }, rc);

    //Create an instance of InkAnnotation class and set its relevant properties
    var inkAnnot = new InkAnnotation()
    {
        UserName = "Jaime Smith",
        Rect = new RectangleF(rc.Left, rc.Bottom + 20, 72 * 5, 72 * 2),
        LineWidth = 2,
        Color = Color.DarkBlue,
        Contents = "This is an ink annotation drawn via InkAnnotation.Paths."
    };
    float x = 80, y = rc.Bottom + 24, h = 18, dx = 2, dy = 4, dx2 = 4, w = 10, xoff =
15;

    // Scribble 'ink annotation' text:

    // i
    inkAnnot.Paths.Add(new[] { new PointF(x + w / 2, y), new PointF(x + w / 2, y +
h), new PointF(x + w, y + h * .7f) });
    inkAnnot.Paths.Add(new[] { new PointF(x + w / 2 - dx, y - h / 3 + dy), new
PointF(x + w / 2 + dx, y - h / 3) });
    // n
    x += xoff;
    inkAnnot.Paths.Add(new[] { new PointF(x, y), new PointF(x, y + h), new PointF(x,
y + h - dy), new PointF(x + w*0.7f, y),
        new PointF(x + w - dx/2, y + h*.6f), new PointF(x + w, y + h), new PointF(x +
w + dx2, y + h*.7f) });
    // k
    x += xoff;
    inkAnnot.Paths.Add(new[] { new PointF(x, y - h / 3), new PointF(x, y + h) });
    inkAnnot.Paths.Add(new[] { new PointF(x + w, y), new PointF(x + dx, y + h/2 -
dy), new PointF(x, y + h/2),
    new PointF(x + dx2, y + h/2 + dy), new PointF(x + w, y + h), new PointF(x + w +
dx2, y + h*.7f) });

    page.Annotations.Add(inkAnnot);
    doc.Save("InkAnnotation.pdf");
```

Back to Top

File Attachment Annotation

A file attachment annotation represents a reference to a file which typically is embedded in the PDF file. The file attachment annotation appears as a paper clip icon on the PDF file. Users can double-click the icon to open the embedded file. This gives users a chance to view or store the file in the system. GcPdf provides FileAttachmentAnnotation class to enable the users to apply file attachment annotations to the PDF file.



The following code illustrates how to add the file attachment annotation to a PDF document.

C#

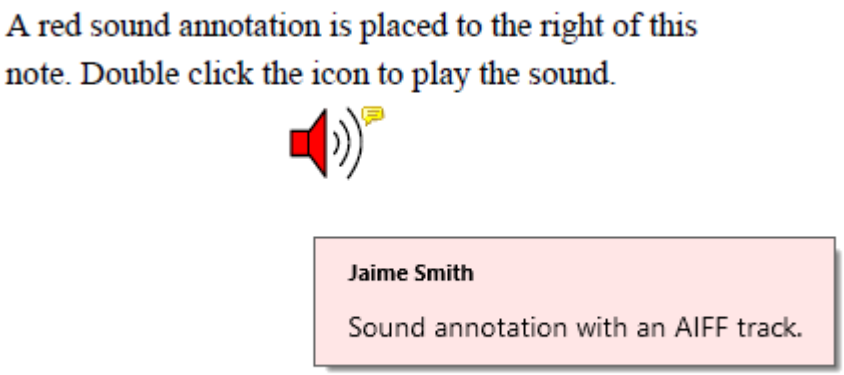
```
public void CreateFileAttachmentAnnotation()
{
    var doc = new GcPdfDocument();
    var page = doc.NewPage();
    var g = page.Graphics;
    RectangleF rc = new RectangleF(50, 50, 400, 80);
    g.DrawString("Some files from the sample's Resources/Images folder are attached
to this page. Some viewers may not show attachments, so we draw rectangles to
indicate their(usually clickable) location", new TextFormat() { Font =
StandardFonts.Times, FontSize = 11}, rc);

    var ip = new PointF(rc.X, rc.Bottom + 9);
    var attSize = new SizeF(36, 12);
    var gap = 8;
    string[] files = new string[]
    {
        "tudor.jpg",
        "sea.jpg",
        "puffins.jpg",
        "lavender.jpg",
    };
    foreach (string fn in files)
    {
```

```
        string file = Path.Combine("Resources", "Images", fn);
        //Create an instance of FileAttachmentAnnotation class and set its relevant
properties
        FileAttachmentAnnotation faa = new FileAttachmentAnnotation()
        {
            Color = Color.FromArgb(unchecked((int)0xFFc540a5)),
            UserName = "Jaime Smith",
            Rect = new RectangleF(ip.X, ip.Y, attSize.Width, attSize.Height),
            Contents = "Attached file: " + file,
            Icon = FileAttachmentAnnotationIcon.Paperclip,
            File =
FileSpecification.FromEmbeddedFile(EmbeddedFileStream.FromFile(doc, file)),
        };
        page.Annotations.Add(faa); //Add the file attachment annotation
        g.FillRectangle(faa.Rect, Color.FromArgb(unchecked((int)0xFF40c5a3)));
        g.DrawRectangle(faa.Rect, Color.FromArgb(unchecked((int)0xFF6040c5)));
        ip.Y += attSize.Height + gap;
    }

    doc.Save("FileAttachmentAnnotation.pdf");
}
```

Back to Top

Sound Annotation

Sound annotation is analogous to a text annotation except that instead of a text note, it contains sound (.au, .aiff, or .wav format) imported from a file or recorded from the computer's microphone. GcPdf provides SoundAnnotation class to enable the users to apply sound annotations to the PDF file.



The following code illustrates how to add a sound annotation to a PDF document.

C#

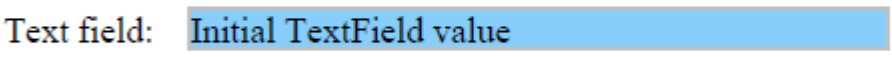
```
public void CreateSoundAnnotation()
{
    GcPdfDocument doc = new GcPdfDocument();
    Page page = doc.NewPage();
```

```
    RectangleF rc = new RectangleF(50, 50, 250, 50);
    page.Graphics.DrawString("A red sound annotation is placed to the right of this
note. Double click the icon to play the sound.", new TextFormat() { Font =
StandardFonts.Times, FontSize = 11 }, rc);

    //Create an instance of SoundAnnotation class and set its relevant properties
    var aiffAnnot = new SoundAnnotation()
    {
        UserName = "Jaime Smith",
        Contents = "Sound annotation with an AIFF track.",
        Rect = new RectangleF(rc.Right, rc.Top, 24, 24),
        Icon = SoundAnnotationIcon.Speaker,
        Color = Color.Red,
        Sound = SoundObject.FromFile(Path.Combine("Resources", "Sounds",
"ding.aiff"), AudioFormat.Aiff)
    };
    page.Annotations.Add(aiffAnnot);
    doc.Save("SoundAnnotation.pdf");
}
```

Back to Top

Widget Annotation

Widget annotations are used in interactive forms to represent the appearance of fields. It is also used to manage user interaction. GcPdf provides WidgetAnnotation class to to enable the users to apply widget annotations to the PDF file.



The following code illustrates how to add a widget annotation to a PDF document.

C#

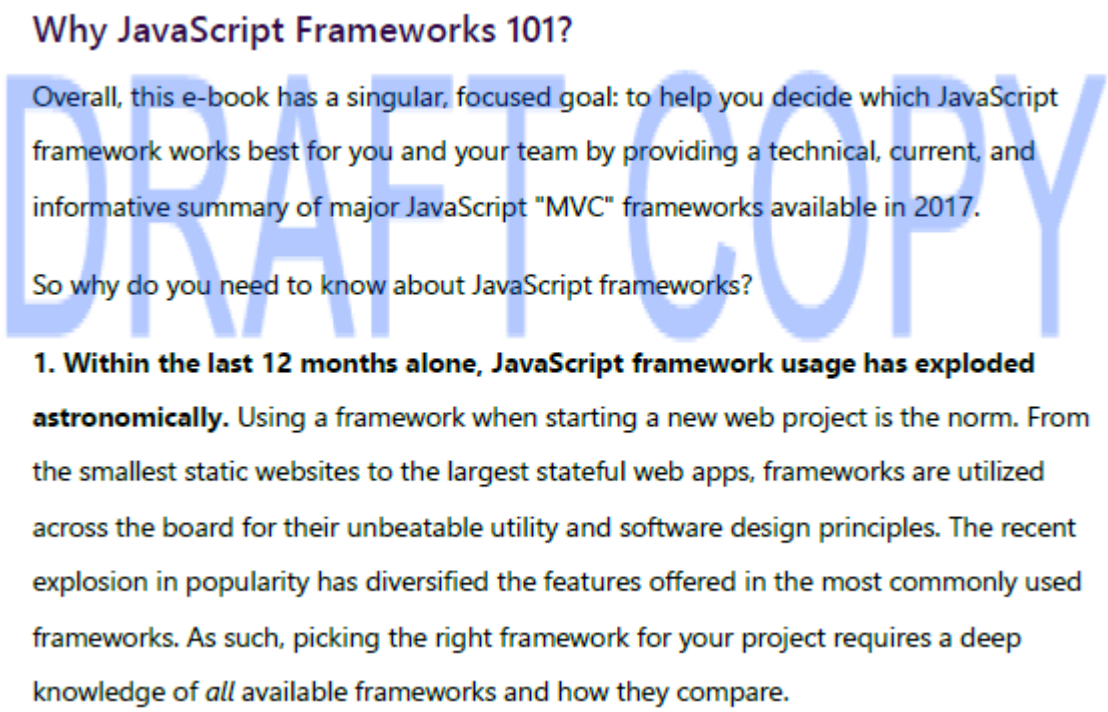
```
public void CreateWidgetAnnotation()
{
    var doc = new GcPdfDocument();
    var page = doc.NewPage();
    var g = page.Graphics;
    TextFormat tf = new TextFormat();
    tf.Font = StandardFonts.Times;
    tf.FontSize = 11;
    PointF ip = new PointF(72, 72);
    float fldOffset = 72 * 2;
    float fldHeight = tf.FontSize * 1.2f;
    float dY = 32;

    // Text field:
    g.DrawString("Text field:", tf, ip);
    var fldText = new TextField();
    fldText.Value = "Initial TextField value";
```

```
    //Get the WidgetAnnotation to specify view properties of the text field.
    WidgetAnnotation widgetAnnotation =fldText.Widget;
    widgetAnnotation.Page = page;
    widgetAnnotation.Rect = new RectangleF(ip.X + fldOffset, ip.Y, 72 * 3,
fldHeight);
    widgetAnnotation.TextFormat.Font = tf.Font;
    widgetAnnotation.TextFormat.FontSize = tf.FontSize;
    widgetAnnotation.Border.Color = Color.Silver;
    widgetAnnotation.BackColor = Color.LightSkyBlue;
    doc.AcroForm.Fields.Add(fldText);
    ip.Y += dY;

    doc.Save("WidgetAnnotation.pdf");
}
```

Back to Top

Watermark Annotation

A watermark annotation defines graphics that is expected to be printed at a fixed size and position on a page, regardless of the dimensions of the printed page. GcPdf provides WatermarkAnnotation class to enable the users to apply watermark annotations to the PDF file.



The following code illustrates how to add a watermark annotation to a PDF document.

C#

```
public void CreateWatermarkAnnotation()
{
    GcPdfDocument doc = new GcPdfDocument();
```

```
    var fs = new FileStream(Path.Combine("CompleteJavaScriptBook.pdf"),
FileMode.Open, FileAccess.Read);
    doc.Load(fs);  //Load the document

    //Loop over pages, add a watermark to each page:
    foreach (var page in doc.Pages)
    {
        //Create an instance of WatermarkAnnotation class and set its relevant
properties
        var watermark = new WatermarkAnnotation()
        {
            Rect = new RectangleF(50, 300, 500, 150),
            Image = Image.FromFile("draft-copy.png"),
            Page = page // Add watermark to page
        };
        doc.Save("WatermarkAnnotation.pdf");
    }

}
```

Back to Top

Redact Annotation

Redact annotation helps in hiding the content that is not supposed to be shared. In place of the removed content, a marking or highlight appears to show that the region has been redacted. With GcPdf class library, you can find all instances of texts and mark the content for redaction. This allows the people in charge for redaction to open the PDF in Acrobat Reader and apply redactions on the marked content. GcPdf provides RedactAnnotation class to enable the users to apply redact annotations to the PDF file.

EMPLOYEE NAME: Jaime Smith				TITLE: Senior Developer	
EMPLOYEE NUMBER: 12345				STATUS: Full Time	
DEPARTMENT: Research & Development				SUPERVISOR: Jane Donahue	
DATE	START TIME	END TIME	REGULAR HOURS	OVERTIME HOURS	TOTAL HOURS
Sun 1/6/19	7:55 AM	8:06 PM	00:00	12:11	12:11
Mon 1/7/19	6:28 AM	2:43 PM	08:00	00:15	08:15
Tue 1/8/19	7:13 AM	8:18 PM	08:00	05:05	13:05
Wed 1/9/19	8:48 AM	7:55 PM	08:00	03:07	11:07
Thu 1/10/19	11:53 AM	9:05 PM	08:00	01:12	09:12
Fri 1/11/19	8:11 AM	8:56 PM	08:00	04:45	12:45
Sat 1/12/19	11:16 AM	11:59 PM	00:00	12:43	12:43
WEEKLY TOTALS			40.000	39.300	79.300

The following code illustrates how to add a redact annotation to a PDF document.

C#

```
public void CreatePDF()
{
    GcPdfDocument doc = new GcPdfDocument();
    var fs = new FileStream(Path.Combine("TimeSheet.pdf"), FileMode.Open,
FileAccess.Read);
    doc.Load(fs);  //Load the document

    //Create Redact annotation
    RedactAnnotation redactAnnotation = new RedactAnnotation();
    var l=doc.FindText(new FindTextParams("Jaime Smith", true, false),null); //search
the text(e.g employee name) which needs to be redacted
    redactAnnotation.Area.Add(l[0].Bounds); // add the text's fragment area to the
annotation
    redactAnnotation.OverlayFillColor = System.Drawing.Color.Gray;
    redactAnnotation.OverlayText = "REDACTED";
    redactAnnotation.OverlayTextRepeat = true;
    redactAnnotation.Justification = VariableTextJustification.Centered;
    redactAnnotation.MarkBorderColor = Color.Black;
    redactAnnotation.OverlayTextAppearanceString="/Helv 0 Tf"; //In this string, the
Tf operator sets the font and size of the text in the annotation
```

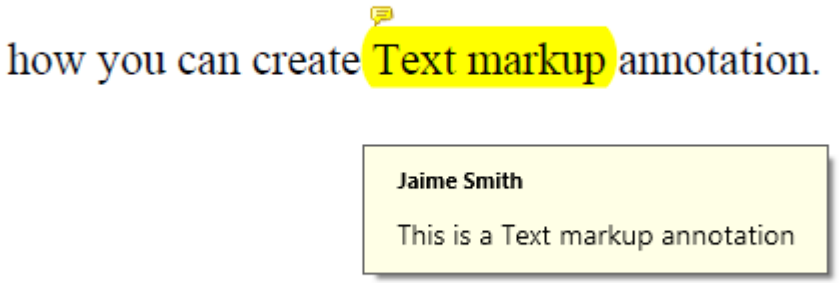
```
    doc.Pages[0].Annotations.Add(redactAnnotation); //Add the redact annotation to
the page
    doc.Save("TimeSheet_Redacted.pdf");
}
```

Note: The redact annotation supports marking of content meant to be redacted, but it currently does not support the actual removal of content.

Back to Top

Text Markup Annotation

Text markup annotations is the simplest type of markup annotation for marking up page text. Text markup annotation appears as underlines, highlights, strikeouts or jagged underlines in the document's text. GcPdf provides TextMarkupAnnotation class to enable the users to apply text markup annotations to the PDF file.



The following code illustrates how to add a text markup annotation to a PDF document.

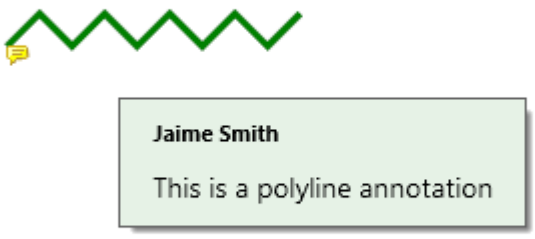
C#

```
public void CreateTextMarkupAnnotation()
{
    GcPdfDocument doc = new GcPdfDocument();
    var page = doc.NewPage();
    var tl = page.Graphics.CreateTextLayout();
    tl.DefaultFormat.Font = StandardFonts.Times;
    tl.DefaultFormat.FontSize = 14;
    tl.Append("This sample demonstrates how you can create Text markup annotation.");
    page.Graphics.DrawTextLayout(tl, new PointF(72, 72));

    //Create an instance of TextMarkupAnnotation class and set its relevant
properties
    var textMarkupAnnot = new TextMarkupAnnotation();
    textMarkupAnnot.MarkupType = TextMarkupType.Highlight;
    textMarkupAnnot.UserName = "Jaime Smith";
    textMarkupAnnot.Contents = "This is a Text markup annotation";
```

```
    var found = doc.FindText(new FindTextParams("Text markup", true, false), null);
//search the text(e.g employee name) which needs to be redacted
    foreach (var f in found)
        textMarkupAnnot.Area.Add(f.Bounds);
    textMarkupAnnot.Color = Color.Yellow;

    page.Annotations.Add(textMarkupAnnot); //Add the text markup annotation to the
page
    doc.Save("TextMarkupAnnotation.pdf");
}
```

Polyline Annotation

Polyline annotations display closed or open shapes of multiple edges on the page. When clicked, it displays a pop-up window containing the text of the associated note. GcPdf provides PolyLineAnnotation class to enable the users to apply polyline annotations to the PDF file.



The following code illustrates how to add a polyline annotation to a PDF document.

C#

```
public void CreatePolyLineAnnotation()
{
    GcPdfDocument doc = new GcPdfDocument();
    Page page = doc.NewPage();
    RectangleF rc = new RectangleF(50, 50, 400, 40);
    page.Graphics.DrawString("This sample demonstrates how you can create a polyline
annotation.", new TextFormat() { Font = StandardFonts.Times, FontSize = 14 }, rc);

    //define the points of the polyline
    var points = new List<PointF>();
    float x = rc.Left,y=rc.Bottom;
    for (int i=0 ;i<10 ;i++,x+=10)
    {
        points.Add(new PointF(x,y));
        y = i % 2 == 0 ? y - 10 : y+10;
```

```
    }

    //Create an instance of PolyLineAnnotation class and set its relevant properties
    var polyLineAnnot = new PolyLineAnnotation()
    {
        Points = points,
        UserName = "Jaime Smith",
        LineWidth = 2,
        Color = Color.Green,
        Contents = "This is a polyline annotation",
    };

    page.Annotations.Add(polyLineAnnot); //Add the polyline annotation to the page
    doc.Save("PolyLineAnnotation.pdf");
}
```

Document

Apart from content, a PDF file holds some additional information in the form of document properties. These properties define various attributes of document as a whole.

GcPdf provides following document properties through GcPdfDocument class:

Compression

GcPdf allows you to compress or reduce the original file size of the document using CompressionLevel property. The compression level can be set to Fastest, Nocompression or Optimal. The default value is System.IO.Compression.CompressionLevel.Fastest.

Document Info

GcPdf contains DocumentInfo property which includes basic information about the document such as title, author, subject etc., that helps in its identification. This data is generated automatically, if not set explicitly.

Font Embedding

GcPdf allows you to set the mode of font embedding using FontEmbedMode property. By default, font subsets are embedded in a document. However, you can change this property to embed whole fonts or not to embed fonts.

Metadata

GcPdf provides Metadata property which allows you to get the metadata associated with the document. Metadata such as keywords, descriptions are used by the search engines to narrow down the searches. This property provides a number of predefined accessors, such as contributors, creators, copyright, description, etc.

Actions

GcPdf contains OpenAction method which provides a value specifying an action that should be performed when a document is opened.

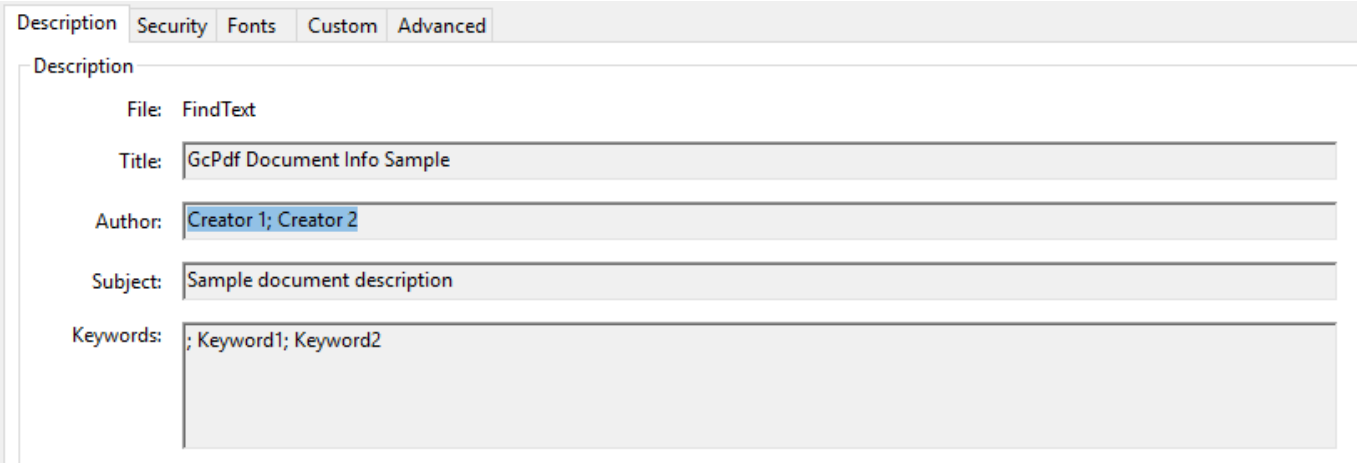
Pdf Version

GcPdf allows you to set the PDF version of the selected document using PdfVersion property. Although, the version of the document is determined automatically but it can be set explicitly.

Viewer Preferences

GcPdf provides ViewerPreferences property to specify how a document should be displayed on opening in a viewer. This property allows you to set the predominant reading order for text, set the number of copies to be

printed when the print dialog is opened for this file, and more preferences.



Get Document Properties

To get the document properties from a particular PDF document:

1. Create an object of GcPdfDocument class.
2. Load any existing PDF file using the Load method.
3. Use the GcPdfDocument object to get the document properties of the PDF file.

C#

```
static void Main(string[] args)
{
    // Load an existing PDF using FileStream
    FileStream fileStream = File.OpenRead(args[0].ToString());
    GcPdfDocument doc = new GcPdfDocument();
    doc.Load(fileStream, null);

    // Get and Display the property values
    Console.WriteLine("Author of the document is {0}", doc.DocumentInfo.Author);
    Console.WriteLine("Document subject is {0}", doc.DocumentInfo.Subject);
    Console.WriteLine("Documentation title {0}", doc.DocumentInfo.Title);
}
```

Back to Top

Set Document Properties

To set the document properties while generating a PDF document:

1. Create an object of GcPdfDocument class.
2. Set the document properties using the created object.

C#

```
public void PDFDoc(Stream stream)
{
    const float In = 150;
    // Create a new PDF document:
    var doc = new GcPdfDocument();
```

```
    var page = doc.NewPage();
    var g = page.Graphics;

    var tf = new TextFormat() { Font = StandardFonts.Times, FontSize = 12 };
    // Set a PDF Version
    doc.PdfVersion = "1.7";

    doc.DocumentInfo.Title = "GcPdf Document Info Sample";
    doc.DocumentInfo.Author = "John Doe";
    doc.DocumentInfo.Subject = "GcPdfDocument.DocumentInfo";
    doc.DocumentInfo.Producer = "GcPdfWeb Producer";
    doc.DocumentInfo.Creator = "GcPdfWeb Creator";

    // Set CreationDate
    doc.DocumentInfo.CreationDate = DateTime.Today;

    // Document metadata is available via the GcPdfDocument.Metadata property.
    // It provides a number of predefined accessors, such as:
    doc.Metadata.Contributors.Add("contributor 1");
    doc.Metadata.Contributors.Add("contributor 2");
    doc.Metadata.Copyright = "GrapeCity Inc.";
    doc.Metadata.Creators.Add("Creator 1");
    doc.Metadata.Creators.Add("Creator 2");
    doc.Metadata.Description = "Sample document description";
    doc.Metadata.Keywords.Add("Keyword1");
    doc.Metadata.Keywords.Add("Keyword2");
    doc.Metadata.Source = "Sourced by GcPdfWeb";
    // Finally, add some text to the document and save the document
    g.DrawString("1. Test string. This is a sample text",tf, new PointF(In, In));
    doc.Save(stream);
}
```

Back to Top

Merge Documents

To merge two PDF documents into a single document, useMergeWithDocument method of the GcPdfDocument class which appends one PDF document into another.

C#

```
//Create a basic pdf
GcPdfDocument doc1 = new GcPdfDocument();
GcPdfGraphics g = doc1.NewPage().Graphics;
g.DrawString("Hello World!", new TextFormat() { Font = StandardFonts.Times,
              FontSize = 12 }, new PointF(72, 72));

//Create second pdf
GcPdfDocument doc2 = new GcPdfDocument();
GcPdfGraphics g1 = doc2.NewPage().Graphics;
g1.DrawString("This PDF will be merged with another PDF.", new TextFormat()
{
    Font = StandardFonts.Times, FontSize = 12
```

```
},
new PointF(72, 72));

//Merge the two documents
doc1.MergeWithDocument(doc2, new MergeDocumentOptions());

doc1.Save("MergedDocument.pdf");
```

Back to Top

Save PDF as Image

PDF pages often contain important information, which can be used for Powerpoint presentations, webpages or word processing documents. In such cases, you might want to make small changes in the PDF pages. With GcPdf library, you can save PDF documents as high quality image files, without turning to online PDF-to-Image converter tools.

There are two ways to save a PDF document as an image.

Using SaveAs methods

GcPdf library provides methods to save the entire PDF document or a specific range as an image. The user can provide the file names and call the SaveAsBmp, SaveAsPng, SaveAsGif, SaveAsJpeg, and SaveAsTiff methods of the GcPdfDocument class.

To save a PDF as an image, follow the steps given below:

1. Create an instance of GcPdfDocument class.
2. Load the PDF document using the Load method.
3. Call the OutputRange method to define the pages of the document that need to be saved.
4. Call the SaveAsImageOptions method to save the image in different formats (JPEG, BMP, PNG and GIF).

C#

```
GcPdfDocument doc = new GcPdfDocument();
var fs = new FileStream(Path.Combine("Wetlands.pdf"), FileMode.Open,
FileAccess.Read);
doc.Load(fs);  //Load the document

//Create an output range object which defines which pages of the document should be
saved
//If no output range is defined then all the pages of the document will be saved
OutputRange pageRange = new OutputRange(1, 2);

//Specify the options that should be used while saving the document's pages to image
SaveAsImageOptions op = new SaveAsImageOptions();
SaveAsImageOptions saveOptions = new SaveAsImageOptions()
{
    BackColor = Color.LightCyan,
    DrawAnnotations = false,
    DrawFormFields = false,
    Resolution = 100
};
doc.SaveAsJpeg("WetlandsImage{0}.jpeg", pageRange, saveOptions); //Saves the document
pages as images in JPEG format
doc.SaveAsBmp("WetlandsImage{0}.bmp", pageRange, saveOptions); //Saves the document
```

```
pages as images in BMP format
doc.SaveAsGif("WetlandsImage{0}.gif", pageRange, saveOptions); //Saves the document
pages as images in GIF format
doc.SaveAsPng("WetlandsImage{0}.png", pageRange, saveOptions); //Saves the document
pages as images in PNG format
```

Back to Top

GcPdf also enables a user to save PDF pages as images by simply calling methods of the Page class like SaveAsBmp, SaveAsPng, SaveAsGif, SaveAsTiff and SaveAsJpeg methods.

To save a PDF page directly as an image, follow the steps given below:

1. Create an instance of the GcPdfDocument class.
2. Load the PDF document.
3. Set BackColor and Resolution properties of the PDF page in SaveAsImageOptions class.
4. Save the required page of the PDF document by invoking the appropriate method of Page class.

C#

```
GcPdfDocument doc = new GcPdfDocument();
var fs = new FileStream(Path.Combine("Wetlands.pdf"), FileMode.Open,
FileAccess.Read);
doc.Load(fs);  //Load the document

//Specify the options that should be used while saving the page to image
SaveAsImageOptions saveOptions = new SaveAsImageOptions()
{
    BackColor = Color.LightCyan,
    DrawAnnotations = false,
    DrawFormFields = false,
    Resolution = 100
};

//Saves the document's first page as an image to a file in JPEG format
doc.Pages[0].SaveAsJpeg("WetlandsImage.jpeg",saveOptions);

//Saves the document's first page as an image to a stream in JPEG format
MemoryStream stream = new MemoryStream();
doc.Pages[0].SaveAsJpeg(stream,saveOptions);
```

Render PDF pages on GcGraphics

GcPdf allows a user to render specific PDF pages, annotations or a mix of PDF pages to images. The user can draw PDF pages on the image graphics using Draw method of the Page class. If the PDF pages contain annotations, you can draw the PDF page annotations on the graphics object with DrawAnnotations method of the Page class. After drawing on the graphics object, the bitmap image can be saved in any image format by calling the SaveAsPng, SaveAsJpeg or SaveAsTiff methods of the GcBitmap class.

Note: To implement this method, you will need the license for GcImaging library.

To save a PDF as an image, follow the steps given below:

1. Create an instance of GcPdfDocument class.
2. Load any existing PDF file using the Load method.

3. Create an instance of GcBitmap class to render the PDF into an image.
4. Call the Draw method to render the PDF file pages content with or without annotations and the DrawAnnotations method to render only the page annotations on the image graphics.
5. Save the rendered PDF page into the required image format by calling SaveAsPng, SaveAsJpeg or SaveAsTiff methods of the GcBitmap class.

The code snippet below illustrates how to save a PDF document as an image.

C#

```
GcPdfDocument doc = new GcPdfDocument();
doc.Load(new FileStream("SampleDoc.pdf", FileMode.Open, FileAccess.Read));

var page = doc.Pages[0];
var sz = page.Bounds;
GcBitmap bmp1 = new GcBitmap((int)(sz.Width + 0.5f), (int)(sz.Height + 0.5f), true);
using (GcGraphics g = bmp1.CreateGraphics(Color.White))
{
    //render whole page content (including the annotations)
    page.Draw(g, new RectangleF(0, 0, sz.Width, sz.Height));
}

GcBitmap bmp2 = new GcBitmap((int)(sz.Width + 0.5f), (int)(sz.Height + 0.5f), true);
using (GcGraphics g = bmp2.CreateGraphics(Color.White))
{
    //render page content without annotations
    page.Draw(g, new RectangleF(0, 0, sz.Width, sz.Height), false);
}

GcBitmap bmp3 = new GcBitmap((int)(sz.Width + 0.5f), (int)(sz.Height + 0.5f), true);
using (GcGraphics g = bmp3.CreateGraphics(Color.White))
{
    //render only the page's annotations
    page.DrawAnnotations(g, new RectangleF(0, 0, sz.Width, sz.Height));
}

/*Once the PDF page has been rendered on GcGraphics,
 *then the rendered PDF page can be saved as an image in various image formats
 *such as, JPEG, PNG, BMP, TIFF, and GIF.
*/
bmp1.SaveAsPng("WholePageContents.png");
bmp2.SaveAsJpeg("PageContentsWithoutAnnotations.jpeg");
bmp3.SaveAsTiff("PageAnnotations.tiff");
```

Back to Top

For more information about how to implement document properties and merge documents using GcPdf, see GcPdf sample browser.

Font

To work with a PDF document, you need a library that supports different kinds of fonts. GcPdf provides support for following font types:

- OpenType
- TrueType
- WOFF

To make sure that any of these listed fonts used in a layout can be viewed as it is after downloading or saving the file, GcPdf library provides the following techniques:

- Automatic Font Embedding: Ensures the fonts used in a PDF document are displayed as it is intended even if the fonts are not installed on a machine.
- Font Fallback: Used when a specified (in the user's source code) font does not have glyphs for the characters to be rendered in the text.

While creating a PDF file, you may want to use fonts other than the standard fonts. To do so, usually you need to add a font from C:\Windows\Fonts directory. Registering the whole directory every time you want to use different fonts can become unmanageable and time consuming. GcPdf library solves this issue with FontCollection class, that adds global font management services to your application. The **FontCollection** class provides font related services to different program modules. Fonts can be registered with a FontCollection via RegisterFont or RegisterDirectory methods. Both these methods register disk files, and do not load the actual fonts into memory. This saves space and improves performance as fonts are loaded only when needed.

Using Standard PDF Fonts

GcPdf supports the 14 standard fonts that are mentioned in the PDF specification 1.7 (section 9.6.2). To use these standard PDF fonts, specify one of the standard fonts using the StandardFonts enum members.

C#

```
public void CreatePDF(Stream stream)
{
var doc = new GcPdfDocument();
var page = doc.NewPage();
var g = page.Graphics;

var textFormat = new TextFormat()
   {
      Font = StandardFonts.HelveticaBold,
      FontSize = 14
   };
// Render text using DrawString method
g.DrawString("1. Test string.", textFormat, new PointF(72, 72));
// Save Document
doc.Save(stream);
}
```

Back to Top

Using Font from File

To use an external font file for applying fonts:

1. Create a new font from a font file, using the Font.FromFile method.
2. Use the font (for example, Gabriola) to render a text with the DrawString method.

C#

```
public void CreatePDF(Stream stream)
{
    GrapeCity.Documents.Text.Font gabriola =
GrapeCity.Documents.Text.Font.FromFile("Gabriola.ttf");

    // Now that we have our font, use it to render some text
    TextFormat tf = new TextFormat()
    {
        Font = gabriola,
        FontSize = 16
    };
    GcPdfDocument doc = new GcPdfDocument();
    GcPdfGraphics g = doc.NewPage().Graphics;
    g.DrawString("Sample text drawn with font {gabriola.FontFamilyName}.",
     tf, new PointF(72, 72));

    // Save Document
    doc.Save(stream);
}
```

Back to Top

Font Embedding

To embed font in a PDF file, you can use the FontEmbedMode property provided by the GcPdfDocument class. By default, font subsets containing only glyphs used in the document, are embedded. However, this can be changed and set to embed full font using FontEmbedMode enum, which leads to huge file size.

C#

```
// Use GcPdfDocument object to set the FontEmbedMode
doc.FontEmbedMode = FontEmbedMode.EmbedFullFont;
```

Back to Top

Font Collection

To use FontCollection:

1. Create an instance of the FontCollection class.
2. Register font from a specified file using RegisterFont method. Also, you can register one or more directories containing fonts using the RegisterDirectory method.
3. Assign the font collection instance to GcGraphics.FontCollection property.
4. Use DrawString method to render text and specify the font name stored in font collection.

C#

```
public void CreatePDF(Stream stream)
{
    // Create a FontCollection instance:
    FontCollection fc = new FontCollection();

    // Get the font file using RegisterFont method:
```

```
    fc.RegisterFont("georgia.ttf");

    // Generate a sample document using the font collection to provide fonts:
    var doc = new GcPdfDocument();
    var page = doc.Pages.Add();
    var g = page.Graphics;

    // Allow the TextLayout created internally by GcGraphics
    // to find the specified fonts in the font collection:
    g.FontCollection = fc;

    // Use GcGraphics.DrawString to show that the font collection is also used
    // by the graphics once the FontCollection has been set on it:
    g.DrawString("Text rendered using Georgia bold, drawn by GcGraphics.DrawString()
method.",
        new TextFormat() { FontName = "georgia", FontSize = 10 },
        new PointF(72, 72 * 4));
    // Done:
    doc.Save(stream);
}
```

Back to Top

Fallback Fonts

To use fallback fonts:

1. Create an instance of the GcPdfDocument class.
2. Get the list of fallback font families using GetFallbackFontFamilies method.
3. Add the list of fallback font families to global SystemFonts using AppendFallbackFontFamilies method.
4. Load your fallback file that includes Japanese glyphs using AppendFallbackFonts method.
5. Use DrawString method to render Japanese text.

C#

```
public void CreatePDF(Stream stream)
{
    // Set up GcPdfDocument:
    GcPdfDocument doc = new GcPdfDocument();
    GcPdfGraphics g = doc.NewPage().Graphics;

    // Set up some helper vars for rendering lines of text:
    const float margin = 36;
    PointF ip = new PointF(margin, margin);

    // Initialize a text format with one of the standard fonts. Standard fonts are
minimal
    // and contain very few glyphs for non-Latin characters.
    TextFormat tf = new TextFormat() { Font = StandardFonts.Courier, FontSize = 14 };

    // Get the list of fallback font families:
    string[] fallbacks = FontCollection.SystemFonts.GetFallbackFontFamilies();
```

```
    // Add the original list of fallback font families to global SystemFonts:
    FontCollection.SystemFonts.AppendFallbackFontFamilies(fallbacks);

    // On some systems, default system fallbacks might not provide Japanese glyphs,
    // so we add our own fallback:
    Font arialuni = Font.FromFile(Path.Combine("Resources", "Fonts",
"ARIALUNI.TTF"));
    FontCollection.SystemFonts.AppendFallbackFonts(arialuni);

    // As the fallback fonts are available, the Japanese text will render
    // correctly as an appropriate fallback will have been found:
    g.DrawString("Sample text with fallbacks available: あなたは日本語を話せますか？", tf,
ip);
    ip.Y += 36;

    // Done:
    doc.Save(stream);
}
```

Back to Top

Enumerate Fonts

To list all fonts in a PDF document along with some of the key font properties, use the following code example. The example code loads the PDF document into a temporary document to get the listing of all fonts and creates a Font object from each of those PDF fonts, and reports whether the operation succeeded.

C#

```
// Open an arbitrary PDF, load it into a temp document and get all fonts:
using (var fs = new FileStream(Path.Combine("Resources", "PDFs", "Test.pdf"),
    FileMode.Open, FileAccess.Read))
{
    var doc1 = new GcPdfDocument();
    doc1.Load(fs);
    var fonts = doc1.GetFonts();
    tl.AppendLine($"Total of {fonts.Count} fonts found in {sourcePDF}:");
    tl.AppendLine();
    int i = 0;
    foreach (var font in fonts)
    {
        var nativeFont = font.CreateNativeFont();
        tl.Append($"{i}:\tBaseFont: {font.BaseFont}; IsEmbedded:
{font.IsEmbedded}.");
        tl.AppendParagraphBreak();
        if (nativeFont != null)
            tl.AppendLine($"\tCreateNativeFont succeeded, family:
{nativeFont.FontFamilyName};" +
                $" bold: {nativeFont.FontBold}; italic: {nativeFont.FontItalic}.");
        else
            tl.AppendLine("\tCreateNativeFont failed");
        tl.AppendLine();
        ++i;
```

```
    }
    tl.PerformLayout(true);
```

Back to Top

Advanced Features

GcPdf library supports variety of fonts that can work with multilingual characters to write a PDF in different languages. It also provides support for font features along with special characters, such as End-User Defined Characters (EUDC) support, surrogates, ligatures, and Unicode characters.

For more information about implementation of font features using GcPdf, see GcPdf sample browser.

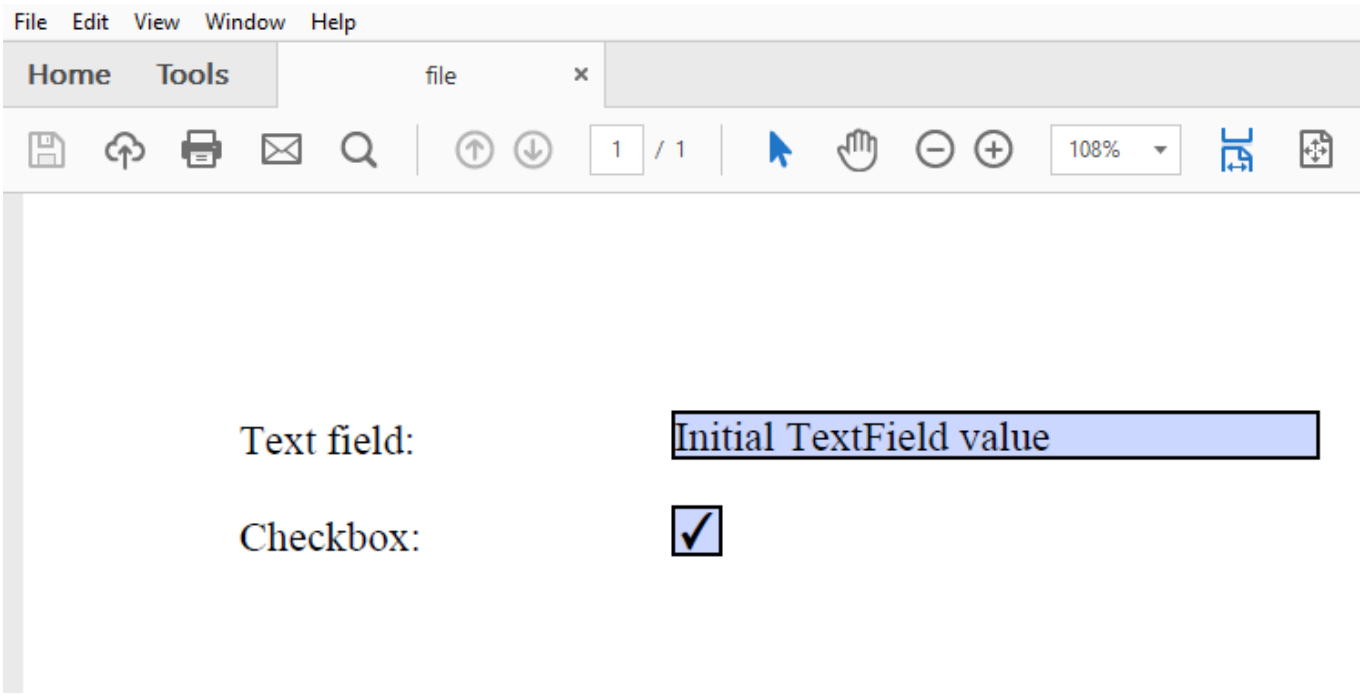
Back to Top

Forms

The PDF-based fillable form, also known as AcroForm, is an interactive form with a collection of fields, such as TextBox, Button, CheckBox, etc. These fields can be filled with data programmatically or manually in order to take information as input from the user. For more information on AcroForm, see PDF specification 1.7 (Section 12.7).

GcPdf allows you to create AcroForms using AcroForm class and define common properties of AcroForm using Fields property of **AcroForm** class. The library lets you add, get, modify, and delete different form fields. You can use the following fields in AcroForms.

- TextField
- CheckBoxField
- CombTextField
- ComboBoxField
- ListBoxField
- PushButtonField
- RadioButtonField
- SignatureField



Add AcroForm Fields

To add AcroForm fields in a PDF document using GcPdf:

1. Create an instance of class corresponding to field you want to add to the form, for example, TextField class.
2. Set the basic properties of the field.
   Observe that the field is also being filled in the code through Value property.
3. Add the field to the form using the Add method.

C#

```
public void CreatePDF(Stream stream)
{
    var doc = new GcPdfDocument();
    var page = doc.NewPage();
    var g = page.Graphics;
    TextFormat tf = new TextFormat();
    tf.Font = StandardFonts.Times;
    tf.FontSize = 14;
    PointF ip = new PointF(72, 72);
    float fldOffset = 72 * 2;
    float fldHeight = tf.FontSize * 1.2f;
    float dY = 32;

    // Add TextField
    g.DrawString("Text field:", tf, ip);
    var fldText = new TextField();
    fldText.Value = "Initial TextField value";
    fldText.Widget.Page = page;
    fldText.Widget.Rect = new RectangleF(ip.X + fldOffset, ip.Y, 72 * 3, fldHeight);
    fldText.Widget.TextFormat.Font = tf.Font;
    fldText.Widget.TextFormat.FontSize = tf.FontSize;
    doc.AcroForm.Fields.Add(fldText);
    ip.Y += dY;

    // Add Checkbox:
    g.DrawString("Checkbox:", tf, ip);
    var fldCheckbox = new CheckBoxField();
    fldCheckbox.Value = true;
    fldCheckbox.Widget.Page = page;
    fldCheckbox.Widget.Rect = new RectangleF(ip.X + fldOffset, ip.Y, fldHeight,
fldHeight);
    doc.AcroForm.Fields.Add(fldCheckbox);
    ip.Y += dY;

    // Save Document
    doc.Save(stream);
}
```

Back to Top

Modify AcroForm Fields

To modify form fields in PDF document, get the particular form field by specifying its index and set a new value for the field using Value property. This property fills the field with the specified string value.

C#

```
doc.AcroForm.Fields[0].Value = "Sample Text";
```

Back to Top

Delete AcroForm Fields

To delete a particular form field in PDF document, use RemoveAt method to remove any field by specifying its index value. Apart from this, Clear method can be used to remove all the AcroForm fields from the document.

C#

```
// Delete a particular field
doc.AcroForm.Fields.RemoveAt(0);

// Delete all the fields
doc.AcroForm.Fields.Clear();
```

Back to Top

For more information about implementation of AcroForms using GcPdf, see GcPdf sample browser.

Import and Export Forms Data

PDFs contain forms, which after being filled can be transferred over the web as forms data. The common formats used to transfer forms data over the web are FDF, XFDF and XML. These files are convenient to transfer since they are much smaller in size than the original PDF form file. The GcPdf library supports the import and export of PDF forms data in FDF, XFDF and XML file formats.

- **FDF:** An FDF file stands for Forms Data Format file and stores the values of form fields in a key/value pair fashion.
- **XFDF:** An XFDF file is an encoded XML version of FDF and stores the value of form fields in a hierarchical manner using XML tags.
- **XML:** An XML file is a plain text format, and majority of the applications prefer this format for storing and sharing data.

Note that the forms data can also be imported or exported from or to streams (in-memory objects) files.

Import or Export Forms Data from FDF

The forms data can be imported from FDF by calling the ImportFormDataFromFDF method of GcPdfDocument class, while the forms data can be exported to FDF by calling the ExportFormDataToFDF method of GcPdfDocument class.

The following code snippet illustrates how to import from FDF and export to FDF.

C#

```
public void ImportDataFromFDF()
{
    var doc = new GcPdfDocument();
    //Load the document
    doc.Load(new FileStream(Path.Combine("Pdf_BlankForm.pdf"), FileMode.Open,
```

```
FileAccess.Read));
    //Open the FDF file
    FileStream stream = new FileStream(Path.Combine("FDF_Data.fdf"), FileMode.Open,
FileAccess.Read);
    doc.ImportFormDataFromFDF(stream); //Import the form data
    doc.Save("PdfForm_FDF.pdf"); //Save the document
}
public void ExportDataToFDF()
{
    var doc = new GcPdfDocument();
    //Load the document
    doc.Load(new FileStream(Path.Combine("Pdf_FilledForm.pdf"), FileMode.Open,
FileAccess.Read));
    //Export the form data to a stream
    MemoryStream stream = new MemoryStream();
    doc.ExportFormDataToFDF(stream);

    //Alternatively, we can even export to a file of appropriate format
    //Export the form data to a FDF file
    //doc.ExportFormDataToFDF("FDF_Data.fdf");

}
```

Back to Top

Import or Export Forms Data from XFDF

The forms data can be imported from XFDF by calling the ImportFormDataFromXFDF method of GcPdfDocument class, while the forms data can be exported to XFDF by calling the ExportFormDataToXFDF method of GcPdfDocument class.

The following code snippet illustrates how to import from XFDF and export to XFDF.

C#

```
public void ImportDataFromXFDF()
    {
        var doc = new GcPdfDocument();
        //Load the document
        doc.Load(new FileStream(Path.Combine("Pdf_BlankForm.pdf"), FileMode.Open,
FileAccess.Read));
        //Open the XFDF file
        FileStream stream = new FileStream(Path.Combine("XFDF_Data.xfdf"),
FileMode.Open, FileAccess.Read);
        //Import the form data
        doc.ImportFormDataFromXFDF(stream);
        //Save the document
        doc.Save("PdfForm_XFDF.pdf");
    }
    public void ExportDataToXFDF()
    {
        var doc = new GcPdfDocument();
        //Load the document
        doc.Load(new FileStream(Path.Combine("Pdf_FilledForm.pdf"), FileMode.Open,
```

```
FileAccess.Read));

        MemoryStream stream = new MemoryStream();
        //Export the form data to a stream
        doc.ExportFormDataToXFDF(stream);

        //Alternatively, we can even export to a file of appropriate format
        //Export the form data to a XFDF file
        //doc.ExportFormDataToXFDF("XFDF_Data.xfdf");
}
```

Back to Top

Import or Export Forms Data from XML

The forms data can be imported from XML by calling the ImportFormDataFromXML method of GcPdfDocument class, while the forms data can be exported to XML by calling the ExportFormDataToXML method of GcPdfDocument class.

The following code snippet illustrates how to import from XML and export to XML.

C#

```
public void ImportDataFromXML()
{
    var doc = new GcPdfDocument();
    //Load the document
    doc.Load(new FileStream(Path.Combine("Pdf_BlankForm.pdf"), FileMode.Open,
FileAccess.Read));
    //Open the XML file
    FileStream stream = new FileStream(Path.Combine("XML_Data.xml"), FileMode.Open,
FileAccess.Read);
    //Import the form data
    doc.ImportFormDataFromXML(stream);
    //Save the document
    doc.Save("PdfForm_XML.pdf");
}
public void ExportDataToXML()
{
    var doc = new GcPdfDocument();
    //Load the document
    doc.Load(new FileStream(Path.Combine("Pdf_FilledForm.pdf"), FileMode.Open,
FileAccess.Read));

    MemoryStream stream = new MemoryStream();
    //Export the form data to a stream
    doc.ExportFormDataToXML(stream);

    //Alternatively, we can even export to a file of appropriate format
    //Export the form data to an XML file
    //doc.ExportFormDataToXML("XML_Data.xml");
}
```

Back to Top

Form XObjects

Form XObject is a technique for representing objects, such as text, graphics, page and images within a PDF document. The purpose of Form XObject is specifically for recurrent objects that gets stored once in a PDF document but can be referenced multiple times, either on several pages or at several locations on the same page and produces the same result each time. Hence, one of the common use cases of Form XObjects could be to import the content of existing PDF document to another. For more information on Form XObject, see PDF specification 1.7

To import content from one PDF document to another using FormXObject:

1. Initialize two instances of GcPdfDocument class, one to load the existing PDF file from which the content is to be imported and the other one is the target PDF to which the imported content is to be rendered.
2. Create a list of the FormXObject using the pages from the loaded PDF document.
3. Loop through the FormXObject list to add pages to the target PDF document and render the corresponding FormXObject to each page using the **DrawForm** method of the **GcPdfGraphics** class.

C#

```
static void Main(string[] args)
{
    //Create a temporary pdf document to load an existing document
    GcPdfDocument tempDoc = new GcPdfDocument();
    FileStream fs = new FileStream(("SlidePages.pdf"), FileMode.Open,
FileAccess.Read);
    tempDoc.Load(fs);

    // Create a new pdf document
    GcPdfDocument mainDoc = new GcPdfDocument();
    Page p;
    GcPdfGraphics g;

    TextFormat tf = new TextFormat()
    {
        Font = StandardFonts.HelveticaBold,
        FontSize = 16,
        ForeColor = Color.FromArgb(128, Color.Red),
    };

    // Create a list of FormXObject for the pages of the loaded PDF:
    var fxos = new List<FormXObject>();
    tempDoc.Pages.ToList().ForEach(p_ => fxos.Add(new FormXObject(mainDoc, p_)));
    for (int i = 0; i < fxos.Count; ++i)
    {
        p = mainDoc.NewPage();
        g = p.Graphics;

        var rcfx = new RectangleF(10, 50, 500, 600);
        // Draw on the main document through FormX object
        g.DrawForm(fxos[i], rcfx, null, ImageAlign.ScaleImage);
        g.DrawRectangle(rcfx, Color.Red);
        g.DrawString($"Page {i + 1}", tf, rcfx, TextAlignment.Center,
ParagraphAlignment.Center, false);
    }
```

```
    mainDoc.Save("FormXResult.pdf");

    Console.WriteLine("Press any key to exit.");
    Console.ReadKey();
}
```

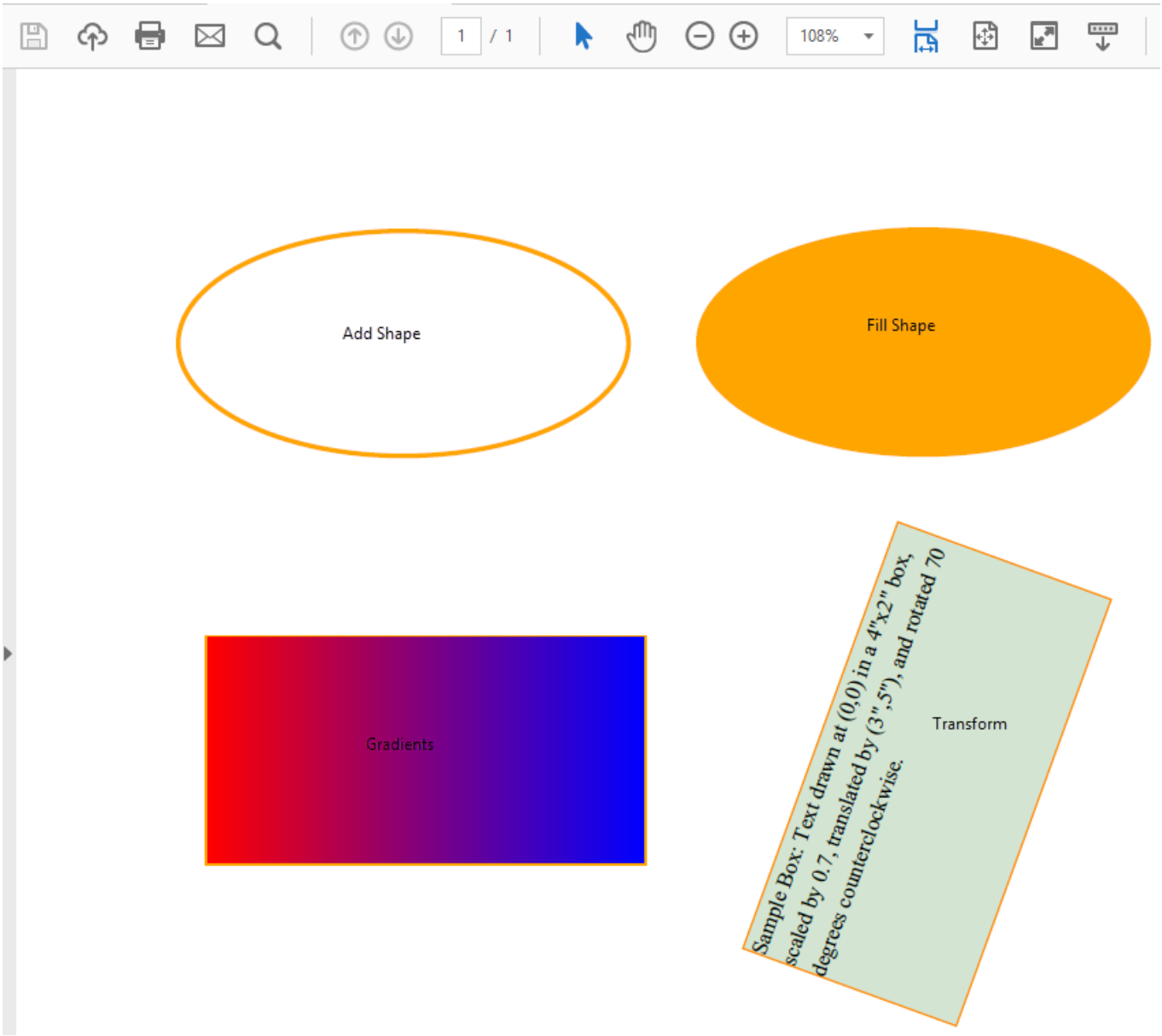
For more information about implementation of Form XObject feature using GcPdf, see GcPdf sample browser.

Graphics

Graphics are visual elements that can be displayed in the form of different shapes, lines, curves or images in a document. These are generally used to supplement text for better illustration of a theory or concept.

GcPdf allows you to draw graphics in a document using methods such as DrawRectangle, DrawEllipse, etc., available in GcGraphics class. These methods use an object of GcPdfGraphics class to draw graphics on a page. Following is a list of graphic elements supported by GcPdf:

- Line
- Rectangle
- Ellipse
- Polygon
- Path



Add Shape

To add a shape in a PDF document:

1. Create an object of GcPdfDocument class.
2. Add a blank page to the document using GcPdfDocument object.
3. Draw an ellipse using DrawEllipse method provided by GcGraphics class.

The following code snippet shows how to add a shape in a PDF document.

C#

```
public void CreatePDF(Stream stream)
{
    // Create a new PDF document:
    var doc = new GcPdfDocument();
    var page = doc.NewPage();
    var g = page.Graphics;
```

```
    // Pen used to draw shape
    var pen = new Pen(Color.Orange, 2);
    // Draw a shape
    g.DrawEllipse(new RectangleF(0.0F, 0.0F, 200.0F, 100.0F), pen);

    // Save document
    doc.Save(stream);
}
```

Back to Top

Fill Shape

To fill a shape:

1. Create an object of GcPdfDocument class.
2. Add a blank page to document using the GcPdfDocument object.
3. Draw an ellipse using the DrawEllipse method provided by the GcGraphics class.
4. Fill the shape using the FillEllipse method.

C#

```
public void CreatePDF(Stream stream)
{
    // Create a new PDF document:
    var doc = new GcPdfDocument();
    var page = doc.NewPage();
    var g = page.Graphics;
    // Pen used to draw shape
    var pen = new Pen(Color.Orange, 2);
    // Draw a Shape
    g.DrawEllipse(new RectangleF(0.0F, 0.0F, 200.0F, 100.0F), pen);
    // Fill a Shape
    g.FillEllipse(new RectangleF(0.0F, 0.0F, 200.0F, 100.0F), Color.Orange);
    // Save document
    doc.Save(stream);
}
```

Back to Top

Add Gradients

To use gradients in a PDF document:

1. Draw a shape by creating an instance of class corresponding to shape you want to add to a page, for example, DrawRectangle class.
2. Create a linear gradient brush by initializing the LinearGradientBrush class and specify the start color and end color for the gradient.
3. Fill the shape by passing the object of LinearGradientBrush in the corresponding fill method, for example, FillRectangle.

C#

```
public void CreatePDF(Stream stream)
```

```
{
    // Create a new PDF document:
    var doc = new GcPdfDocument();
    var page = doc.NewPage();
    var g = page.Graphics;
    // Pen used for Drawing
    var pen = new Pen(Color.Orange, 2);
    // Draw a Shape
    g.DrawRectangle(new RectangleF(0.0F, 0.0F, 200.0F, 100.0F), pen);
    // Create a linear gradient brush
    LinearGradientBrush linearGradBrush = new LinearGradientBrush(Color.Red,
Color.Blue);
    // Fill a Shape
    g.FillRectangle(new RectangleF(0.0F, 0.0F, 200.0F, 100.0F), linearGradBrush);
    // Save document
    doc.Save(stream);
}
```

Similarly, the RadialGradientBrush class provides radial gradient brush and the HatchBrush class provides hatch patterns to fill the shapes.

Back to Top

Add Transformations

To perform transformations using GcPdf, set the Transform property provided by the GcGraphics class. In this example, we have transformed the rectangle box, which is scaled by 0.7, translated by (3',5'), and rotated 70 degrees counterclockwise. The values for transformation are calculated with the help of different methods provided by the Matrix3x2 class.

C#

```
public void CreatePDF(Stream stream)
{
    // Create a PDF document
    var doc = new GcPdfDocument();
    var page = doc.NewPage();
    var g = page.Graphics;

    // Translation Values
    var translate0 = Matrix3x2.CreateTranslation(72 * 3, 72 * 5);
    var scale0 = Matrix3x2.CreateScale(0.7F);
    var rotate0 = Matrix3x2.CreateRotation((float)(-70 * Math.PI) / 180F);

    //Draw Rectangle and Apply Transformations
    var box = new RectangleF(0, 0, 72 * 4, 72 * 2);
    g.Transform = rotate0 * translate0 * scale0;
    g.FillRectangle(box, Color.FromArgb(100, Color.DarkSeaGreen));
    g.DrawRectangle(box, Color.DarkOrange, 1);
    g.DrawString("Sample Box: Text drawn at (0,0) in a 4\"x2\" box"+
    ", scaled by 0.7, translated by (3\",5\"), " +
     "and rotated 70 degrees counterclockwise.",
    new TextFormat() { Font = StandardFonts.Times, FontSize = 14, }, box);
```

```
    // Save document
    doc.Save(stream);
}
```

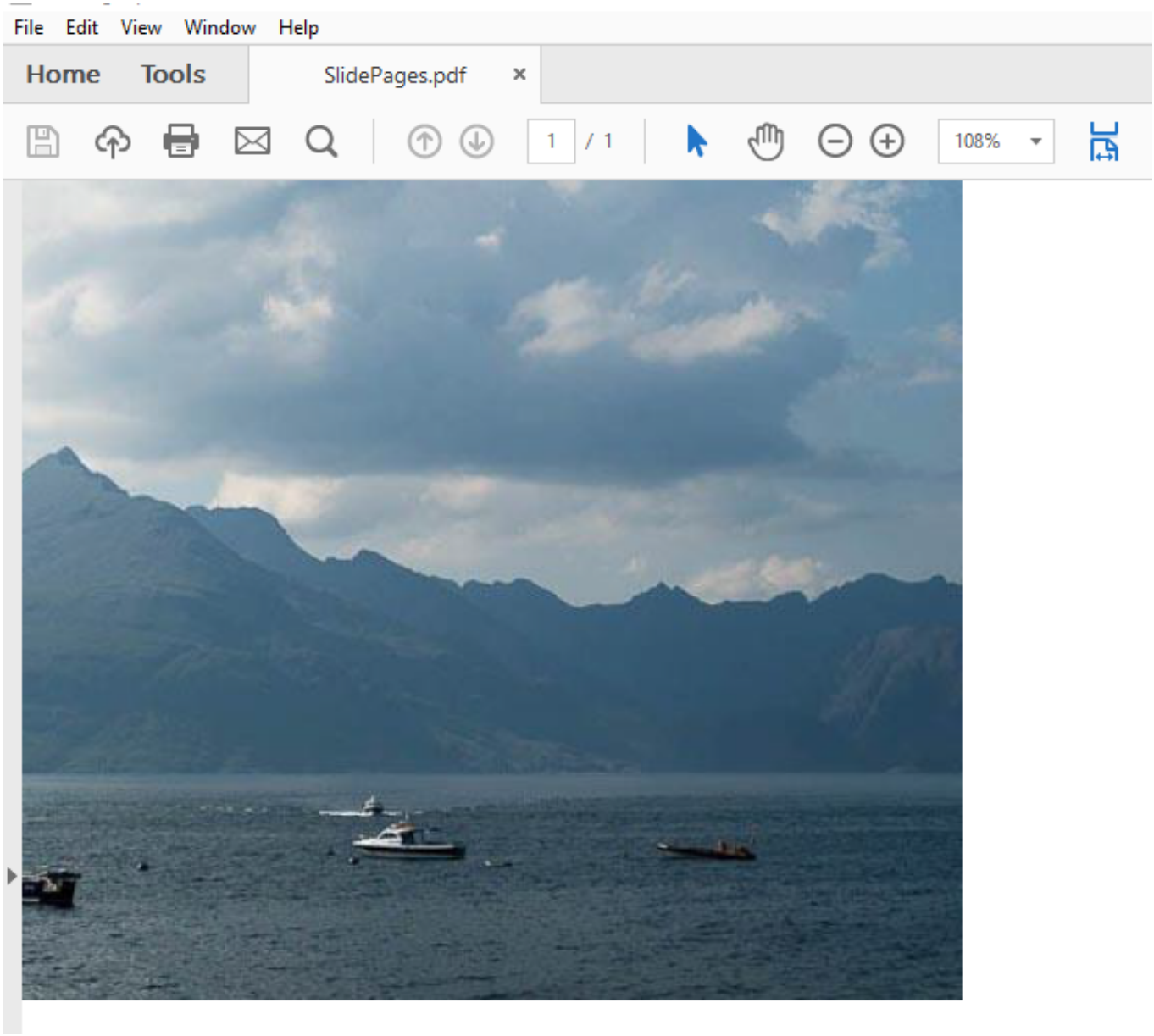
Back to Top

For more information about implementation of graphics using GcPdf, see GcPdf sample browser.

Images

Images are generally used to illustrate important information in your document and highlight points raised in the text. GcPdf allows you to draw an image on a page using DrawImage method of GcGraphics class. In case the same Image object (identified by reference, not by content) is rendered multiple times in a GcPdfDocument, GcPdf considers it to be the same image, and adds only one image data to the PDF, referencing it from all places where it is used in the document. The library supports various image formats, such as BMP, GIF (single frame only), JPEG, and PNG. Additionally, on Windows, TIFF, JpegXR, and ICO formats are also supported.

In addition, GcPdf library provides ImageAlign class to let you align images in different ways using properties such as AlignHorz, AlignVert, BestFit, TileHorz, etc. The library also allows you to control the image quality such as compressing color values, setting JPEG image quality, etc. through ImageOptions property available in GcPdfDocument class.



Add Image From File

To add an image in a PDF document, load the image in your application using Image.FromFile method. This method will store the image in an object of Image class. Once, the image is added, you can use the DrawImage method provided by the GcGraphics class to render the image.

C#

```
public void CreatePDF(Stream stream)
{
    GcPdfDocument doc = new GcPdfDocument();
    var page = doc.NewPage();
    var g = page.Graphics;
    // Add image to the application

    var image = GrapeCity.Documents.Drawing.Image.FromFile
     (Path.Combine("Resources", "Images", "clouds.jpg"));
```

```
    // Use DrawImage to render the image
    g.DrawImage(image, new RectangleF(30.6F, 30.7F, 40.8F, 100.9F),
        null, ImageAlign.CenterImage);
    // Save the PDF file
    doc.Save(stream);
}
```

Back to Top

Add Image From Stream

To add an image in a PDF document using stream, you need to store image in a stream using Image.FromStream method. Once the image is stored, it can be added to the application. Then, you can use the DrawImage method provided by the GcGraphics class to render the image.

C#

```
public void CreatePDF(Stream stream)
{
GcPdfDocument doc = new GcPdfDocument();
var page = doc.NewPage();
var g = page.Graphics;
string fileName = @"C:\Users\Admin\Desktop\clouds.png";
FileStream fs = new FileStream(fileName, System.IO.FileMode.Open);

// Add image to the application
var image = GrapeCity.Documents.Drawing.Image.FromStream(fs);
// Use DrawImage to render the image
g.DrawImage(image, new RectangleF(30.6F, 30.7F, 40.8F, 100.9F), null,
ImageAlign.CenterImage);
// Save the PDF file
doc.Save(stream);
}
```

Additionally, GcPdf allows you to save an image to memory stream as JPEG using ToJpegStream method, PNG using ToPngStream method, and grayscale using ConvertToGrayscale method.

Back to Top

Set Image Opacity

To render an image with a specified transparency, you can add an image to a PDF document using the DrawImage method that takes opacity as one of the parameters.

C#

```
//Create a basic pdf
GcPdfDocument doc = new GcPdfDocument();
GcPdfGraphics g = doc.NewPage().Graphics;
g.DrawString("A sample document showing an image with controlled opacity.",
    new TextFormat() { Font = StandardFonts.Times, FontSize = 12 }, new PointF(72,
72));
```

```
//Add an image by controlling its opacity
var image = RawImage.FromFile(Path.Combine("Resources", "sea.jpg"),
                              RawImageFormat.Jpeg, 800, 532);
ImageAlign ia = new ImageAlign(ImageAlignHorz.Center, ImageAlignVert.Center,
                               true, true, true, false, false);
g.DrawImage(image, new RectangleF(100, 100, 180, 100), null,
ImageAlign.ScaleImage,0.3F);


//Save the final pdf
doc.Save("AddImage_Opacity.pdf");

Console.WriteLine("Press any key to exit");
Console.ReadKey();
```

Back to Top

Extract Image

To extract an image from a PDF document, use GetImages method:

1. Load a PDF document containing image using Load method of the GcPdfDocument class.
2. Extract the image(s) from the PDF document using GetImages method of the GcPdfDocument class.
3. Draw the extracted image(s) on another PDF document using the Graphics.DrawImage method.
4. Save the document using Save method of the GcPdfDocument class.

C#

```
using (FileStream fs = new FileStream(Path.Combine("Resources", "Wetlands.pdf"),
       FileMode.Open, FileAccess.Read))
{
    GcPdfDocument docSrc = new GcPdfDocument();

    // Load an existing PDF with some images
    docSrc.Load(fs);

    //Extract information about images from the loaded PDF
     var imageInfos = docSrc.GetImages();

    GcPdfDocument doc = new GcPdfDocument();
    var textPt = new PointF(72, 72);

    foreach (var imageInfo in imageInfos)
    {
        // The same image may appear on multiple locations,
        // imageInfo includes page indices and locations on pages;

        var g = doc.NewPage().Graphics;
        g.DrawImage(imageInfo.Image, new RectangleF(10,0,400,400), null,
ImageAlign.ScaleImage);
    }
    doc.Save("ExtractImage.pdf");
}
Console.WriteLine("Press any key to exit");
```

```
Console.ReadKey();
```

Similarly, you can also extract image(s) from a page by using GetImage method of the GcPdfDocument class.

Back to Top

For more information about implementation of images using GcPdf, see GcPdf sample browser.

Incremental Update

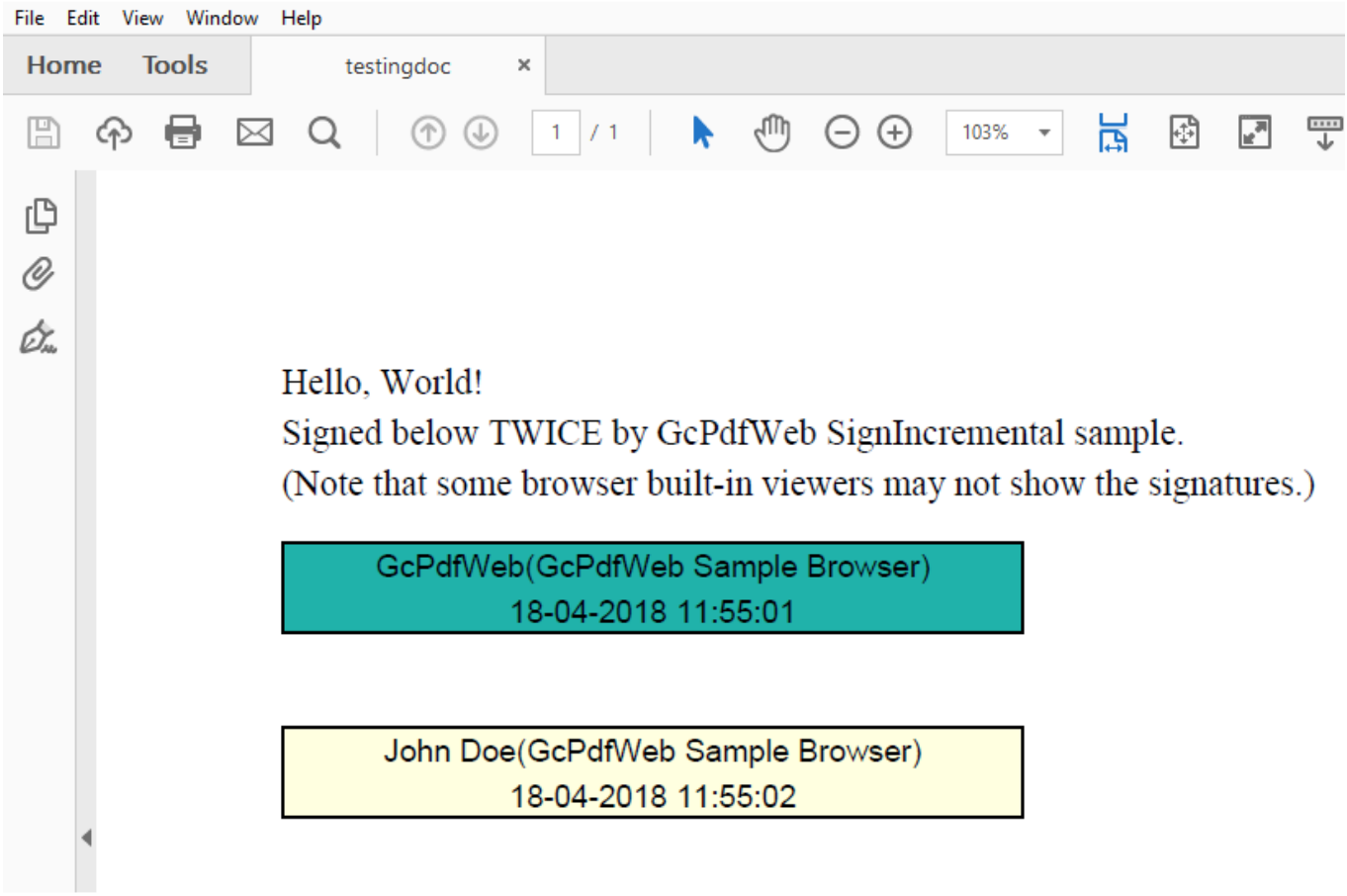
Incremental update is a method to modify a PDF document without affecting its original content. It simply appends the changes at the end of the file that not only saves the time required to re-write the entire document but also minimizes the risk of data loss. This feature is especially important while updating the digitally signed PDF documents as it allows to add new signature to a document without invalidating the original signature.

GcPdf allows you to incrementally update and save a modified document using Save method. By default, this method saves the document without an incremental update. However, you can save the document incrementally by setting the incrementalUpdate parameter of the Save method to true.

Additionally, GcPdf provides Sign method to sign and save a document which by default updates the document incrementally. This method lets you sign a document multiple times without invalidating the original signature and without changing its original content. GcPdf allows three levels of subsequent changes on a signed document:

- No changes
- Modify fields
- Modify fields and add annotations

Note that once a document has been signed, adding a new field invalidates the existing signature. Hence, a document must already have enough signature fields to accommodate all the subsequent signatures. Also, if you run a sample that uses a signed PDF without a valid license key of GcPdf, then the original signature in the generated PDF is invalidated. This happens because a license header is added to the PDF in such cases which changes the original signed document.



Update PDF Incrementally

To incrementally update a PDF file:

1. Create an object of GcPdfDocument class.
2. Load any existing PDF file using the Load method.
3. Modify the document. For example, add some text or graphical element to the document.
4. Save the document using Save method and set incremental update parameter to true.

C#

```
static void Main(string[] args)
{
    // Load an existing PDF using FileStream
    FileStream fileStream = File.OpenRead(args[0].ToString());
    GcPdfDocument doc = new GcPdfDocument();
    doc.Load(fileStream, null);

    const float In = 72;
    var tf = new TextFormat()
    {
        Font = StandardFonts.CourierItalic,
        FontSize = 12
    };

    doc.Pages[0].Graphics.DrawString
```

```
        ("This is a sample text for incremental update", tf, new PointF(In, In));

    doc.Save("IncUpdate", true);
}
```

Back to Top

Add Multiple Signatures

To add multiple digital signatures in a PDF document:

1. Use the SignatureProperties class to set up the first certificate for digital signature.
2. Initialize the SignatureField class to hold the first signature.
3. Add the signature field to the PDF document using the Add method.
4. Connect the signature field to signature properties.
5. Add the signature field to hold the second signature.
6. Sign the document using the Sign method.
7. Load the signed document using the Load method.
8. Setup the second certificate for signing.
9. Sign the document to include second signature.

C#

```
public class SignIncremental
{
    public void CreatePDF(Stream stream)
    {
        GcPdfDocument doc = new GcPdfDocument();

        // Load a signed document (we use code similar to the SignDoc sample):
        doc.Load(CreateAndSignPdf());

        // Init a second certificate:
        var pfxPath = Path.Combine("Resources", "Misc", "JohnDoe.pfx");
        X509Certificate2 cert = new X509Certificate2(File.ReadAllBytes(pfxPath),
"secret",
        X509KeyStorageFlags.MachineKeySet | X509KeyStorageFlags.PersistKeySet
        | X509KeyStorageFlags.Exportable);
        SignatureProperties sp2 = new SignatureProperties()
        {
            Certificate = cert,
            Location = "GcPdfWeb Sample Browser",
            SignerName = "John Doe",
        };

        // Find the 2nd (not yet filled) signature field:
        var sfld2 = doc.AcroForm.Fields["SecondSignature"] as SignatureField;
        // Connect the signature field and signature props:
        sp2.SignatureField = sfld2 ?? throw new Exception
            ("Unexpected: could not find 'SecondSignature' field");

        // Sign and save the document:
        // NOTES:
        // - Signing and saving is an atomic operation, the two cannot be separated.
```

```
        // - The stream passed to the Sign() method must be readable.
        doc.Sign(sp2, stream);

        // Rewind the stream to read the document just created
        // into another GcPdfDocument and verify all signatures:
        stream.Seek(0, SeekOrigin.Begin);
        GcPdfDocument doc2 = new GcPdfDocument();
        doc2.Load(stream);
        foreach (var fld in doc2.AcroForm.Fields)
            if (fld is SignatureField sfld)
                if (!sfld.Value.VerifySignature())
                    throw new Exception($"Failed to verify signature for field
{sfld.Name}");

        // Done (the generated and signed docment has already been saved to
'stream').
    }

    // This method is almost exactly the same as the DigitalSignature sample,
    // but adds a second signature field (does not sign it though):
    private Stream CreateAndSignPdf()
    {
        GcPdfDocument doc = new GcPdfDocument();
        Page page = doc.NewPage();
        TextFormat tf = new TextFormat() { Font = StandardFonts.Times, FontSize = 14
};
        page.Graphics.DrawString(
            "Hello, World!\r\nSigned below TWICE by GcPdfWeb SignIncremental sample"
+
            ".\r\n(Note that some browser built-in viewers may not show the
signatures.)",
            tf, new PointF(72, 72));

        // Init a test certificate:
        var pfxPath = Path.Combine("Resources", "Misc", "GcPdfTest.pfx");
        X509Certificate2 cert = new X509Certificate2(File.ReadAllBytes(pfxPath),
"qq",
        X509KeyStorageFlags.MachineKeySet | X509KeyStorageFlags.PersistKeySet
        | X509KeyStorageFlags.Exportable);
        SignatureProperties sp = new SignatureProperties();
        sp.Certificate = cert;
        sp.Location = "GcPdfWeb Sample Browser";
        sp.SignerName = "GcPdfWeb";

        // Init a signature field to hold the signature:
        SignatureField sf = new SignatureField();
        sf.Widget.Rect = new RectangleF(72, 72 * 2, 72 * 4, 36);
        sf.Widget.Page = page;
        sf.Widget.BackColor = Color.LightSeaGreen;
        sf.Widget.TextFormat.Font = StandardFonts.Helvetica;
        sf.Widget.ButtonAppearance.Caption = $"Signer: {sp.SignerName}"+
```

```
            "\r\nLocation: {sp.Location}";
        // Add the signature field to the document:
        doc.AcroForm.Fields.Add(sf);

        // Connect the signature field and signature props:
        sp.SignatureField = sf;

        // Add a second signature field:
        SignatureField sf2 = new SignatureField() { Name = "SecondSignature" };
        sf2.Widget.Rect = new RectangleF(72, 72 * 3, 72 * 4, 36);
        sf2.Widget.Page = page;
        sf2.Widget.BackColor = Color.LightYellow;
        // Add the signature field to the document:
        doc.AcroForm.Fields.Add(sf2);

        var ms = new MemoryStream();
        doc.Sign(sp, ms);
        return ms;
    }
```

Back to Top

For more information about how to make incremental updates in a PDF document using GcPdf, see GcPdf sample browser.

Linearization

A linearized PDF, when opened in a browser, allows the first page of the document to be loaded and displayed before the entire file is loaded on the browser. GcPdf library allows you to generate linearized PDF documents using Linearized property of GcPdfDocument class which takes boolean values. For more information on linearization, see PDF specification 1.7 (Annexure F).

Generate Linearized PDF

To generate a linearized PDF, set the Linearized property to true.

LinearizedPDF.cs

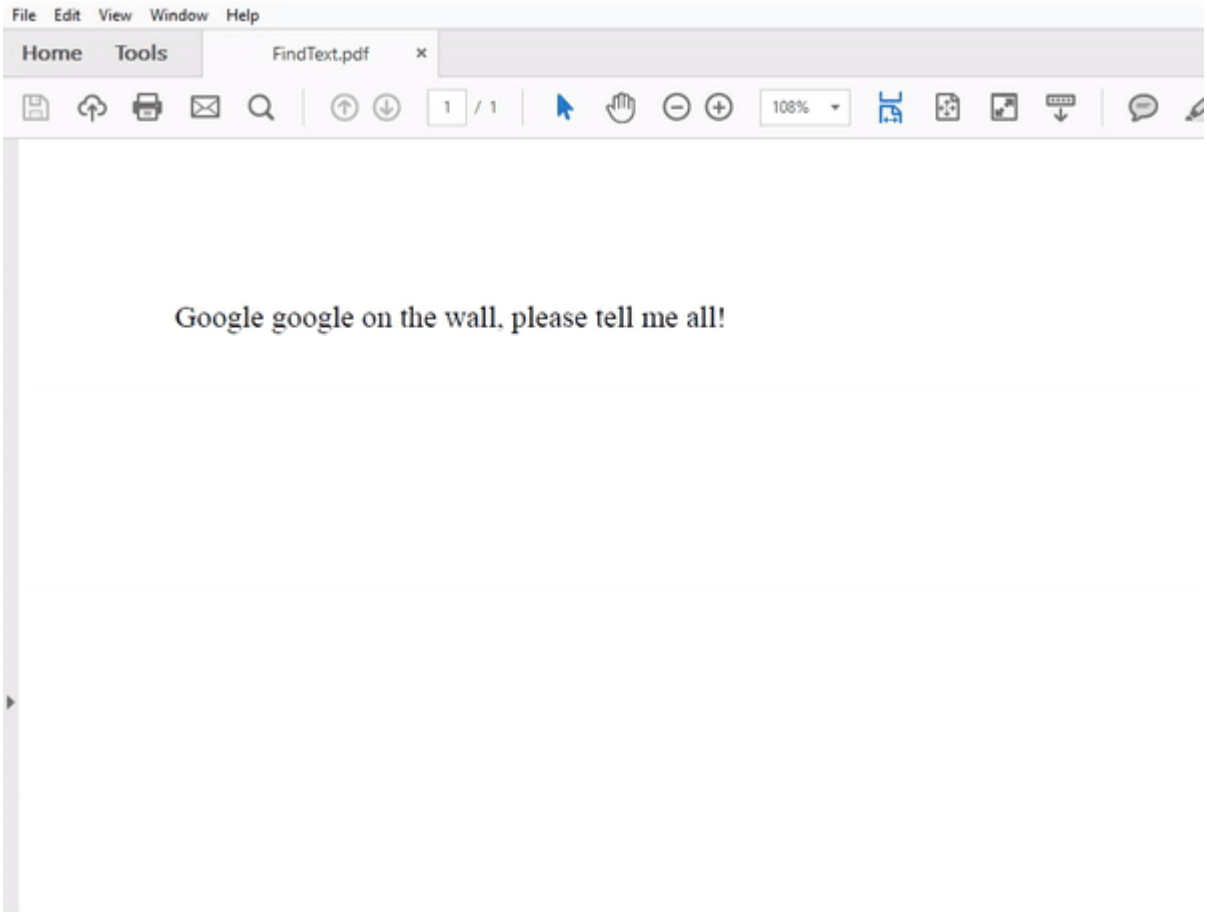
```
// To create a linearized PDF, the only thing we need to do is raise a flag:
doc.Linearized = true;
```

For more information about creating a linearized PDF using GcPdf, see GcPdf sample browser.

Links

Among all the static content of a PDF, links are required to jump from one location to other location within the document or outside the document. Creating hyperlinks is one such way which not only helps in navigating through the content but also makes it interactive. For more information on link annotations, see PDF specification 1.7 (Section 12.5.6.5).

GcPdf allows you to add hypertext links to a PDF document through LinkAnnotation class.



Add Hyperlink

To add a hyperlink in a PDF document, use the LinkAnnotation class. The LinkAnnotation class provides essential properties for creating a hyperlink.

To add a hyperlink:

1. Create an object of GcPdfDocument class.
2. Draw text to represent the hyperlink.
3. Pass the instance of LinkAnnotation class as a parameter to the Add method.

C#

```
public void CreatePDF(Stream stream)
{
    GcPdfDocument doc = new GcPdfDocument();
    var page = doc.NewPage();
    var g = page.Graphics;

    // Draw some text that will represent the link
    var tf = new TextFormat()
    {
        Font = StandardFonts.Times,
        FontSize = 14
    };
    var tl = new TextLayout();
    tl.MarginLeft = tl.MarginTop = tl.MarginRight = tl.MarginBottom = 72;
```

```
    tl.Append("Google google on the wall, please tell me all!", tf);
    tl.PerformLayout(true);
    g.DrawTextLayout(tl, PointF.Empty);

    // Add a link associated with the text area
    page.Annotations.Add
    (new LinkAnnotation(tl.ContentRectangle, new
ActionURI("http://www.google.com")));

    // Done
    doc.Save(stream);
}
```

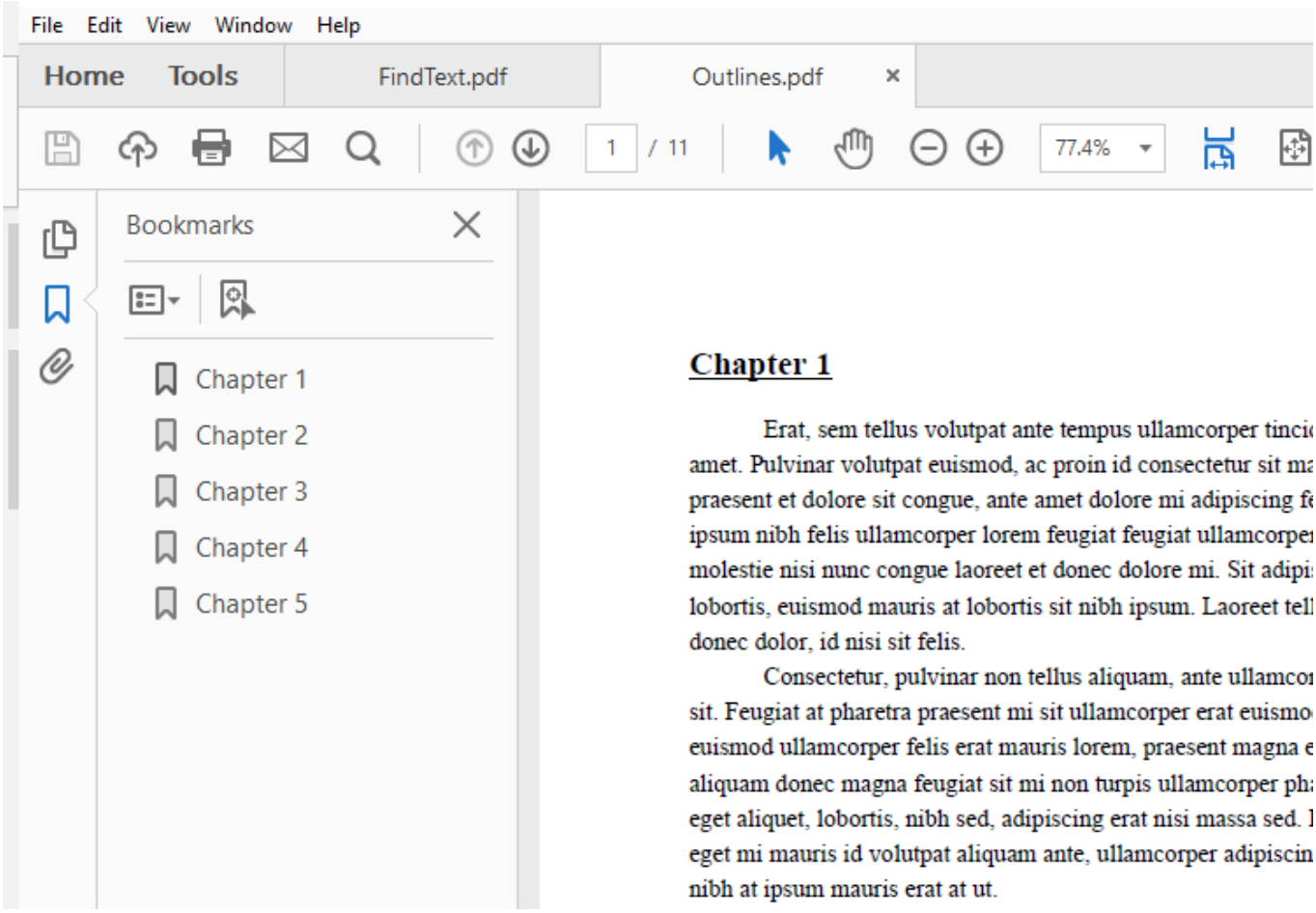
Back to Top

For more information about implementation of links using GcPdf, see GcPdf sample browser.

Outline

Outline is a hierarchical list of items used to organize and display the document structure to the user so that the user can interactively navigate to a particular topic or a location in a document. For more information on outline, see PDF specification 1.7 (Section 12.3.3).

GcPdf allows you to define an outline node in a PDF document using OutlineNode class. You can also add child nodes to the outline nodes and choose whether to display the expanded list with visible child nodes or a compact list using Expanded property of the **OutlineNode** class. This class also provides methods and properties to manipulate document's outline.



Add Outline Node

To add an outline node in the PDF document, pass the instance of OutlineNode class as a parameter to the Add method.

C#

```
// Add  outline node using Add method
doc.Outlines.Add(new OutlineNode("Chapter 5", new DestinationFitH(8, null)));
```

Back to Top

Get Outline Node

To get a specific outline node from the PDF document:

1. Create an object of OutlineNodeCollection class.
2. Use the OutlineNodeCollection object to access a particular outline node using the node index.

C#

```
// Get the OutlineNodeCollection
OutlineNodeCollection nodecol = doc.Outlines;
Console.WriteLine("Outline Title: {0}", nodecol[0].Title);
```

Back to Top

Modify Outline Node

To modify a specific outline node in a PDF file, get the outline from OutlineNodeCollection by specifying its index and set the new value to its properties such as Title property.

C#

```
// Modify an outline node
nodecol[6].Title = "Testing Chapter";
```

Back to Top

Delete Outline Node

To delete all the outline nodes from a PDF document, use Clear method. Apart from this, RemoveAt method can be used to delete a particular outline by specifying its index value.

C#

```
// Delete all the outline nodes
doc.Outlines.Clear();

// Delete a particular outline node
doc.Outlines.RemoveAt(1);
```

Back to Top

For more information about implementation of outlines using GcPdf, see GcPdf sample browser.

Pages

Each page of a document is represented by a page object that includes references to the content and other attributes of a page. GcPdf provides Page class held in GrapeCity.Documents.Pdf assembly to allow you to work with pages. The **Page** class represents a page in GcPdfDocument. To get started, you need to add a page to your PDF document using NewPage method. When a new page is created, it is added to the page collection which is a collection of document's pages. The collection allows standard collection operations, such as adding, inserting, deleting, and moving elements(pages). These pages can be modified using the following page properties for individual pages while creating a PDF document.

- **Page orientation**: Allows you to set the orientation of the current page using Landscape property. The default orientation of the page is set to portrait.
- **Boundaries**: Allows you to set five boundaries of a page, namely Art box, Bleed box, Crop box, Media box, and Trim box, which are defined below.
    - **Art box**: Defines the area covered by the meaningful content including potential white space.
    - **Bleed box**: Defines the region up to which the page's content shall be cropped when the page is to be printed.
    - **Crop box**: Defines the region up to which the page's content shall be cropped when the page is printed or viewed on a system. This is set as default page boundary.
    - **Media box**: Defines the boundaries of the medium on which the page will be printed.
    - **Trim box**: Defines the intended dimensions of the printable page after trimming.

  For more information on page boundaries, see PDF specification 1.7 (Section 14.11.2).
- **Page size**: Allows you to set the size of current page through Size property.
- **Rotation**: Allows you to set the degrees by which a page can be rotated clockwise through Rotate property.
- **Content stream**: Allows you to get the collection of content stream representing content of the page using ContentStreams property. A content stream is a PDF stream object which contains data comprising sequence

of instructions, in the form of PDF objects, describing the graphical elements to be drawn on a page.

Insert a Page

To insert an empty page in a PDF document:

1. Create an object of GcPdfDocument class.
2. Access the NewPage method using the GcPdfDocument object.

Page.cs

```
GcPdfDocument doc = new GcPdfDocument();
// Adds a new blank page
var page = doc.NewPage();
```

Back to Top

Get a Particular Page

To get a particular page from a document:

1. Create an instance of PageCollection class that includes all the pages added in a PDF document.
2. Use the PageCollection object to access any particular page using its index value.

Page.cs

```
// Load an existing PDF using FileStream
FileStream fileStream = File.OpenRead(args[0].ToString());
GcPdfDocument doc = new GcPdfDocument();
doc.Load(fileStream, null);

// Use the PageCollection object to get page properties
PageCollection pageCollection = doc.Pages;
// Get the owner of the page
Console.WriteLine("Page Owner: {0}", pageCollection[0].Owner);
```

Back to Top

Get Page Properties

To get page properties:

1. Create an instance of PageCollection class that includes all the pages added in a PDF document.
2. Use the PageCollection object to access any particular page using its index value.
3. Access the properties associated to a particular page through its page index, for example, Size property.

C#

```
// Load an existing PDF using FileStream
FileStream fileStream = File.OpenRead(args[0].ToString());
GcPdfDocument doc = new GcPdfDocument();
var page = doc.NewPage();
doc.Load(fileStream, null);

// Use the PageCollection object to get a particular page
PageCollection pageCollection = doc.Pages;
```

```
// Get the size of first page
Console.WriteLine("Paper Size: {0}", pageCollection[0].Size);
```

Back to Top

Set Page Properties

To set page properties:

1. Create an object of GcPdfDocument class.
2. Access the NewPage method using the GcPdfDocument object.
3. Use the page object to set a page property, for example, Rotate property.

C#

```
GcPdfDocument doc = new GcPdfDocument();
// Adds a new blank page
var page = doc.NewPage();

// Set the page property
page.Rotate = 90;
```

Back to Top

Set PageSize and Orientation

To set a new page size and orientation in a document:

1. Add a new page in the PDF document using NewPage method.
2. Set the PaperKind and Landscape property using the page object.

PageSize.cs

```
var doc = new GcPdfDocument();
// The default page size is Letter (8 1/2" x 11") with portrait orientation
var page = doc.NewPage();

// Change the page size and orientation
page.PaperKind = PaperKind.A4;
page.Landscape = true;
```

Back to Top

Add Page Labels

GcPdf allows to define page labels with meaningful descriptions rather than just page numbers for identifying a page in a PDF document. Page labels allow to subdivide the document into sequences of logically related page ranges. In addition, it allows you to add multiple page labeling ranges in a single PDF document, that do not intersect each other. This can be very helpful when the PDF document contains different sections such as preface, acknowledgment, main body, index etc.

In GcPdf, the PageLabelingRange class represents a page labeling range which helps in defining the page numbering style for the range and a meaningful prefix that denotes the range. To add page labels in a PDF document, use the PageLabelingRanges property provided by the GcPdfDocument class as shown in the code below.

C#

```
public void CreatePDF()
{
    //Initialize GcPdfDocument
    var doc = new GcPdfDocument();

    //Define text layout
    var tl = new TextLayout(72);
    tl.MaxWidth = doc.PageSize.Width;
    tl.MaxHeight = doc.PageSize.Height;
    TextSplitOptions to = new TextSplitOptions(tl)
    {
        MinLinesInFirstParagraph = 2,
        MinLinesInLastParagraph = 2
    };
    doc.Pages.Add();
    // Generate random text for the document
    doc.Pages.Last.Graphics.DrawTextLayout(tl, PointF.Empty);
    tl.Clear();
    tl.Append(Common.Util.LoremIpsum(17));
    tl.PerformLayout(true);
    // Print the random text
    while (true)
    {
        var splitResult = tl.Split(to, out TextLayout rest);
        doc.Pages.Last.Graphics.DrawTextLayout(tl, PointF.Empty);
        if (splitResult != SplitResult.Split)
            break;
        tl = rest;
        var p = doc.Pages.Add();
    }
    //Define PageLabelingRange for content pages
    //PageLabelingRange uses DecimalArabic NumberingStyle and "Content Page, p. " as
pre
    //of the page label
    doc.PageLabelingRanges.Add(2, new PageLabelingRange($"Content Page, p. ",
    NumberingStyle.DecimalArabic, 1));

    // Done:
    doc.Save("NewPageLabel.pdf");
}
```

Working with ContentStreams

ContentStream object consists a sequence of instructions describing the graphical elements to be rendered on a page. ContentStream is a useful feature, when you are working with multiple graphical elements in a single PDF document. All the content stream added in a PDF document is stored in PageContentStreamCollection. You can access this class to add or remove items to the content stream.

To use content stream on a page:

1. Create an object of PageContentStream class.
2. Add graphic elements to the content stream using the DrawString method.

3. Save the document.

C#

```
public void CreatePDF(Stream stream)
{
    GcPdfDocument doc = new GcPdfDocument();
    var page = doc.NewPage();
    var g = page.Graphics;
    const float In = 72;
    var tf = new TextFormat()
    {
        Font = StandardFonts.Times,
        FontSize = 12
    };

    // Creating PageContentStream object
    PageContentStream contentStream = new PageContentStream(doc);
    // Adding Graphics to the ContentStream
    contentStream.Doc.Pages[0].Graphics.DrawString(
     "1. Test string. This is a sample string",
     tf, new PointF(In, In));
    // Saving the document
    doc.Save(stream);
}
```

Back to Top

For more information about implementation of pages using GcPdf, see GcPdf sample browser.

Security

PDF security can be maintained by controlling access to PDF documents by encrypting PDF, setting permission levels and adding or removing digital signature that will prevent unauthorized users from stealing information in your PDF document. For more information on PDF security, see PDF specification 1.7 (Section 7.6.3).

The GcPdf library supports some of the standard security options in the PDF file format. The following section describes the different types of security features.

Encrypt PDF

PDF documents with sensitive or confidential information require encryption to restrict access to intruders. GcPdf provides Security class to encrypt a document and decline access to unauthorized users.

To encrypt a PDF file:

1. Create an object of StandardSecurityHandlerRev4 class.
2. Set the required properties of the StandardSecurityHandlerRev4 object, such as passwords, encryption algorithm, etc.
3. Pass the object to the EncryptHandler property of the Security class to encrypt the PDF document.

C#

```
public void CreatePDF(Stream stream)
{
```

```
    GcPdfDocument doc = new GcPdfDocument();
    var page = doc.NewPage();
    var g = page.Graphics;
    const float In = 150;
    //Add Encryption
    var std = new StandardSecurityHandlerRev4();
    std.OwnerPassword = "abc";
    std.UserPassword = "qwe";
    // Set EncyptionAlgorithm
    std.EncryptionAlgorithm = EncryptionAlgorithm.RC4;
    std.EncryptionKeyLength = 128;
    // Set the EncryptHandler property.
    doc.Security.EncryptHandler = std;
    // Render text using DrawString method
    g.DrawString("Welcome to GrapeCity, Inc", new TextFormat()
    { Font = StandardFonts.TimesBold, FontSize = 12 }, new PointF(In, In));
    // Save document
    doc.Save(stream);
}
```

Back to Top

Set Permissions

Setting permissions restricts users from copying, printing and editing the contents in a PDF document. The Security class of the GcPdf library allows a user to set up permissions in a PDF document.

To set permissions in a PDF document:

1. Create an object of StandardSecurityHandlerRev3 class.
2. Use the required properties of the StandardSecurityHandlerRev3 object to set the permissions such as editing, printing, etc.
3. Pass the object to the EncryptHandler property of the Security class.

C#

```
public void CreatePDF(Stream stream)
{
    GcPdfDocument doc = new GcPdfDocument();
    var page = doc.NewPage();
    var g = page.Graphics;
    int In = 72;

    // Create a security handler variable
    var std = new StandardSecurityHandlerRev3();
    std.EditingPermissions = EditingPermissions.Enabled;
    std.OwnerPassword = "abc";
    std.UserPassword = "qwe";

    // Set permissions
    std.EditingPermissions = EditingPermissions.Enabled;
    std.CopyContentPermissions = CopyContentPermissions.Enabled;
    std.PrintingPermissions = PrintingPermissions.Disabled;
    doc.Security.EncryptHandler = std;
```

```
    // Render text using DrawString method
    g.DrawString("Welcome to GrapeCity, Inc.", new TextFormat()
    { Font = StandardFonts.TimesBold, FontSize = 12 }, new PointF(In, In));

    // Save document
    doc.Save(stream);
}
```

Back to Top

Digital Signature

GcPDF enables a user to digitally sign a PDF document to secure the authenticity of the content. The library supports digital signature in the PDF document using the SignatureField class. Further, GcPdf allows a user to reuse a signed PDF template by removing the signatures and keeping the Signature Field, or simply removing the Signature Field.

Add Digital Signature

To add digital signature in a PDF document:

1. Use the SignatureProperties class to set up the certificate for digital signature.
2. Initialize the SignatureField class to hold the signature.
3. Add the signature field to the PDF document using the Add method.
4. Connect the signature field to signature properties.
5. Sign the document using the Sign method. It also saves the document.

C#

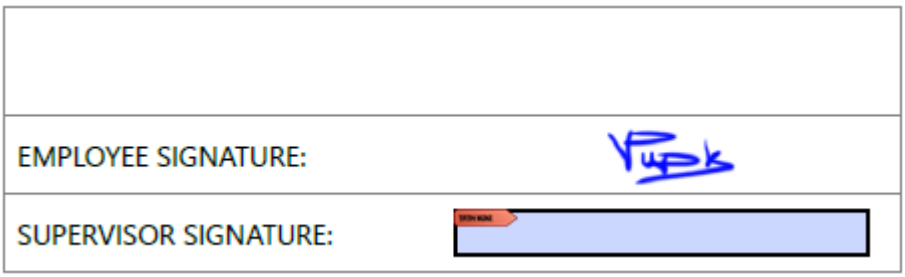
```
public void CreatePDF(Stream stream)
{
    GcPdfDocument doc = new GcPdfDocument();
    Page page = doc.NewPage();
    TextFormat tf = new TextFormat() { Font = StandardFonts.Times, FontSize = 14 };
    page.Graphics.DrawString(
        "Hello, World!\r\nSigned below by GcPdfWeb SignDoc sample." +
        "\r\n(Note that some browser built-in viewers may not show the signature.)",
        tf, new PointF(72, 72));

    // Initialize a test certificate:
    var pfxPath = Path.Combine("Resources", "Misc", "GcPdfTest.pfx");
    X509Certificate2 cert = new X509Certificate2(File.ReadAllBytes(pfxPath), "qq",
    X509KeyStorageFlags.MachineKeySet | X509KeyStorageFlags.PersistKeySet
    | X509KeyStorageFlags.Exportable);
    SignatureProperties sp = new SignatureProperties();
    sp.Certificate = cert;
    sp.Location = "GcPdfWeb Sample Browser";
    sp.SignerName = "GcPdfWeb";

    // Initialize a signature field to hold the signature:
    SignatureField sf = new SignatureField();
    sf.Widget.Rect = new RectangleF(72, 72 * 2, 72 * 4, 36);
    sf.Widget.Page = page;
    sf.Widget.BackColor = Color.LightSeaGreen;
```

```
    sf.Widget.TextFormat.Font = StandardFonts.Helvetica;
    sf.Widget.ButtonAppearance.Caption = $"Signer: " +
        $"{sp.SignerName}\r\nLocation: {sp.Location}";
    // Add the signature field to the document:
    doc.AcroForm.Fields.Add(sf);

    // Connect the signature field and signature properties:
    sp.SignatureField = sf;

    // Sign and save the document:
    // NOTES:
    // - Signing and saving is an atomic operation, the two cannot be separated.
    // - The stream passed to the Sign() method must be readable.
    doc.Sign(sp, stream);

    // Rewind the stream to read the document just created
    // into another GcPdfDocument and verify the signature:
    stream.Seek(0, SeekOrigin.Begin);
    GcPdfDocument doc2 = new GcPdfDocument();
    doc2.Load(stream);
    SignatureField sf2 = (SignatureField)doc2.AcroForm.Fields[0];
    if (!sf2.Value.VerifySignature())
        throw new Exception("Failed to verify the signature");

    // Done (the generated and signed document has already been saved to 'stream').
}
```

Back to Top

Remove Digital Signature

With GcPdf, it is easy to remove a digital signature from a PDF file. The library allows users to remove a signature from signature field, so that the contents of the PDF file can be used again.



To remove the signature and keep the signature field in the PDF document, follow these steps:

1. Initialize an instance of GcPdfDocument class and load the PDF file.
2. To remove all the signatures in the document, call a recursive method which loops through all the signature fields in the PDF file and set the Value property of the SignatureField class to null.
3. Save the document.

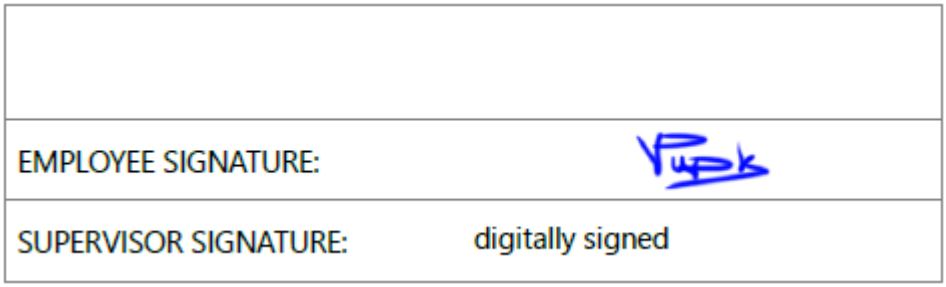
C#

```
var doc = new GcPdfDocument();
using (var fs = new FileStream( "TimeSheet.pdf", FileMode.Open, FileAccess.Read))
```

```
{
    doc.Load(fs);

    // Fields can be children of other fields, so we use
    // a recursive method to iterate through the whole tree:
    removeSignatures(doc.AcroForm.Fields);

    doc.Save("TimeSheet_NoSign.pdf"); //Save the document

    void removeSignatures(FieldCollection fields)
    {
        foreach (var f in fields)
        {
            if (f is SignatureField sf)
                sf.Value = null; //removes the signatures from the document
            removeSignatures(f.Children);
        }
    }
}
```

Back to Top

The GcPdf class library also lets you remove the desired signature field from a list of Form fields. This removes the signature along with the signature field.



To remove the signature field in a PDF document, follow these steps:

1. Initialize an instance of GcPdfDocument class and load the PDF file.
2. Call a recursive method to loop through all signature fields in the PDF file.
3. Call the RemoveAt method to remove a signature field.
4. Save the document.

C#

```
var doc = new GcPdfDocument();
using (var fs = new FileStream("TimeSheet.pdf", FileMode.Open, FileAccess.Read))
{
    doc.Load(fs);

    // Fields can be children of other fields, so we use
    // a recursive method to iterate through the whole tree:
    removeSignatureFields(doc.AcroForm.Fields);

    doc.Save("TimeSheet_NoSignField.pdf"); //Save the document

    void removeSignatureFields(FieldCollection fields)
```

```
    {
        for (int i = fields.Count - 1; i >= 0; --i)
        {
            removeSignatureFields(fields[i].Children);
            if (fields[i] is SignatureField)
                fields.RemoveAt(i); //removes the signature fields from the document
        }
    }
}
```

Back to Top

For more information on applying security using GcPdf, see GcPdf sample browser.

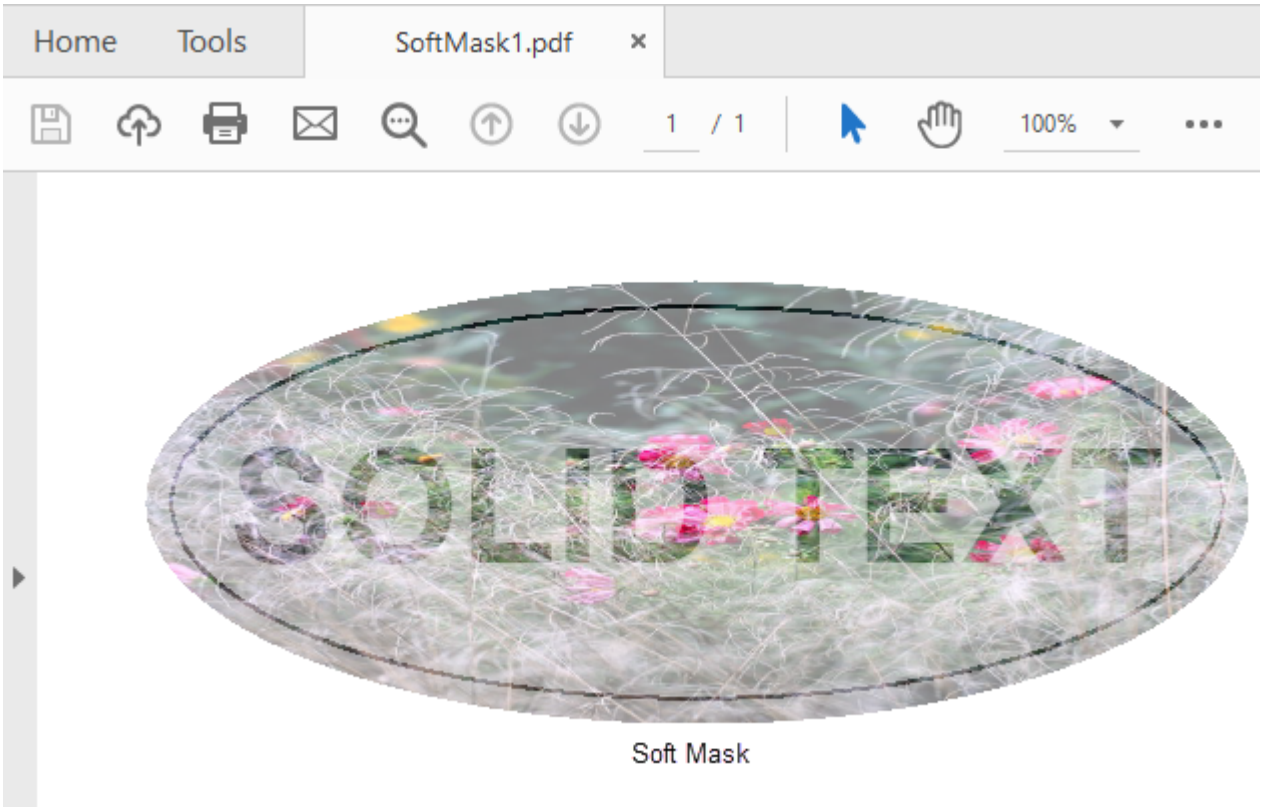
Soft Mask

Soft mask is represented by a transparency group XObject to be used as source of position dependent mask values and the backdrop color space for the group compositing information. It also contains some other entries that control the conversion from the group results to mask values. Soft masks are used to modify the shape of an object or group and produce effects such as a gradual transition between an object and its background. For more information on soft mask, see PDF specification 1.7

In GcPdf, soft mask can be created using the Create method of the SoftMask class which accepts the target document and the bounds where mask is to be applied as parameters. Then, you need to retrieve the graphics of the soft mask by using the Graphics property of FormXObject of the SoftMask class. You can design the mask by drawing on these graphics and once the mask is created, apply the mask to PDF document graphics by assigning it to SoftMask property of the PdfDocumentGraphics class.

Note:

- Only the alpha channel from the mask is used. Solid areas do not mask at all however, transparent areas mask completely. Semi-transparent areas mask in inverse proportion to the alpha value.
- Some PDF viewers do not handle changing the soft masks correctly unless the mask is reset prior to assigning a new one. This can be done by setting the SoftMask property of Graphics object to 'none'.



Soft Mask

To create soft mask using GcPdf:

1. Initialize GcPdfDocument class to create the target PDF document.
2. Invoke the Create method of the SoftMask class to create the SoftMask class object.
3. Get the transparency group FormXobject to be used as the source of alpha for this mask, using the FormXObject property of the SoftMask class.
4. To generate the content for the FormXObject, access its graphics using the Graphics property of the FormXObject class which returns an instance of the GcPdfGraphics class.
5. Use the different drawing methods of the returned GcPdfGraphics object to design the soft mask.
6. Apply the soft mask to the target PDF document by assigning the created soft mask to the SoftMask property of the target PDF document graphics object.

C#

```
public int CreatePDF(Stream stream)
{
    var doc = new GcPdfDocument();
    var page = doc.NewPage();
    var g = page.Graphics;

    var rMask = new RectangleF(0, 0, 72 * 5, 72 * 2);
    var rDoc = new RectangleF(36, 36, rMask.Width, rMask.Height);

    var softMask = SoftMask.Create(doc, rDoc);
    var smGraphics = softMask.FormXObject.Graphics;
    smGraphics.FillEllipse(rMask, Color.FromArgb(128, Color.Black));
    smGraphics.DrawString("SOLID TEXT",
    new TextFormat() { Font = StandardFonts.HelveticaBold, FontSize = 52, ForeColor =
Color.Black },
    new RectangleF(rMask.X, rMask.Y, rMask.Width, rMask.Height),
```

```
    TextAlignment.Center, ParagraphAlignment.Center, false);
    var rt = rMask;
    rt.Inflate(-8, -8);
    // Color on the mask does not matter, only alpha channel is important:
    smGraphics.DrawEllipse(rt, Color.Red);
    g.SoftMask = softMask;
    g.DrawImage(Image.FromFile(Path.Combine("Resources", "Images", "reds.jpg")),
        rDoc, null, ImageAlign.StretchImage);
    // Done:
    doc.Save(stream);
    return doc.Pages.Count;
}
```

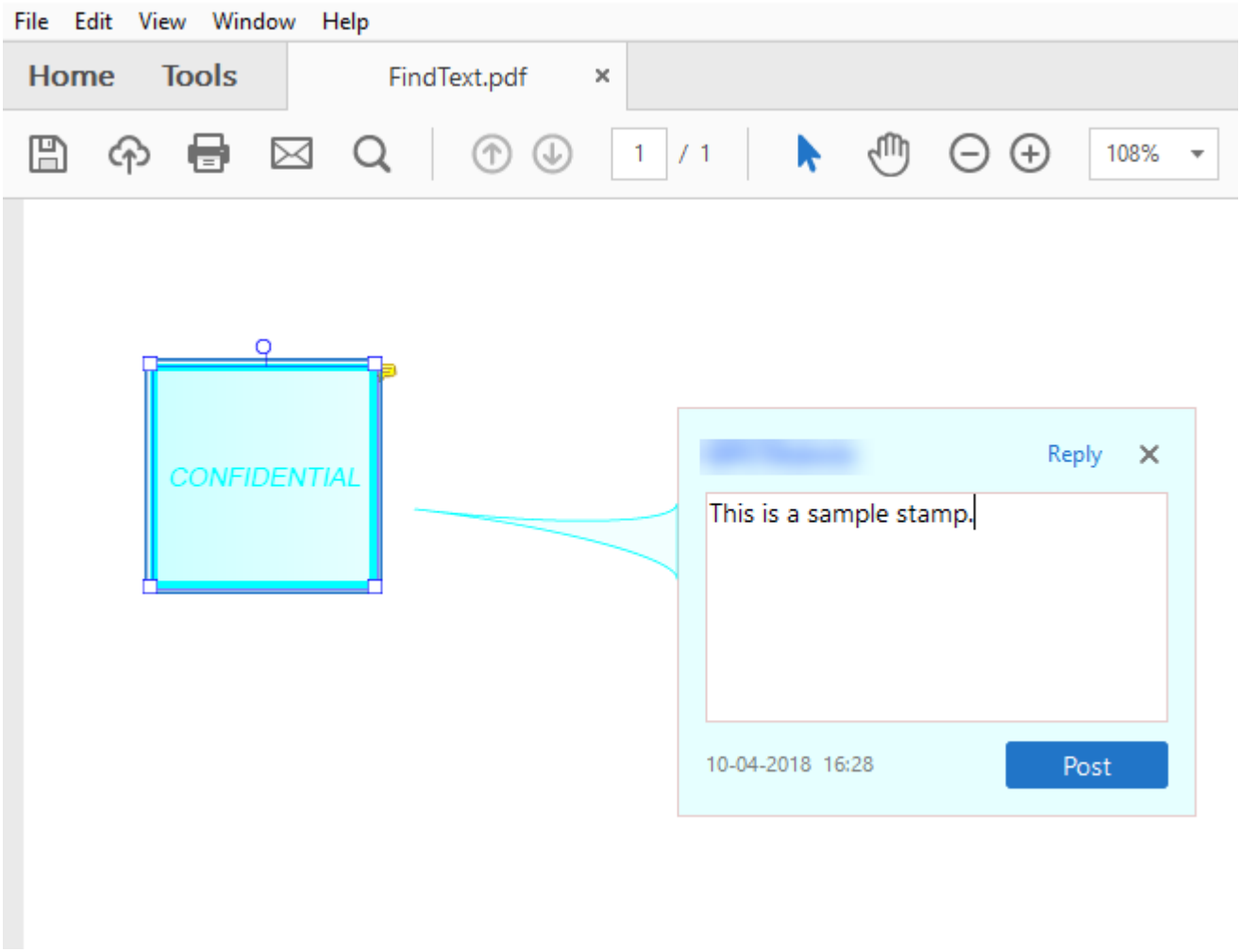
Back to Top

For more information about implementation of soft mask feature using GcPdf, see GcPdf sample browser.

Stamps

Stamps are generally used to personalize, apply branding, and indicate status of the documents, which can be easily moved and modified. A stamp annotation is intended to look as if it is stamped on the page with a rubber stamp just like conventional paper documents. On clicking, it displays a pop-up window containing the text of the associated note. For more information on stamp annotation, refer PDF specification 1.7 (Section 12.5.6.12).

GcPdf allows you to add stamps to a PDF document, using StampAnnotation class. These stamps are text stamp and image stamp. Text stamp allows you to add page numbers, text, and dynamic text stamps like Date, Time, Author to the document. On the other hand, image stamps allow you to add images as stamps and create stamps from PDF, JPG, JPG2000, BMP, and PNG files. The library also lets you to specify possible icons, such as "Confidential", "Departmental", etc., to display the stamps using Icon property of **StampAnnotation** class, which takes the value from StampAnnotationIcon enum.



Add Stamp

To add a stamp in a PDF document, use the StampAnnotation class. The StampAnnotation class provides the essential properties for creating an image or text stamp that looks similar to a rubber stamp.

To add a stamp:

1. Create an object of GcPdfDocument and StampAnnotation class.
2. Set the required properties of StampAnnotation object.
3. Call the Add method to add the stamp on the page.

C#

```
public void CreatePDF(Stream stream)
{
    GcPdfDocument doc = new GcPdfDocument();
    var page = doc.NewPage();

    //Add Stamp
    var stamp = new StampAnnotation()
    {
        Contents = "This is a sample stamp",
        Color = Color.Aqua,
        Icon = StampAnnotationIcon.Confidential.ToString(),
```

```
        CreationDate = DateTime.Today,
        Rect = new RectangleF(100.5F, 110.5F, 72, 72),
    };

    // Add stamp to page
    page.Annotations.Add(stamp);
    //Save Document
    doc.Save(stream);
}
```

Back to Top

Modify Stamp

To modify stamp annotation, you can set the properties of stamp annotation you used on a page. For instance, setting Contents and Color property of the StampAnnotation class modifies the existing content and color of the annotation.

C#

```
stamp.Contents = "Draft Copy";
stamp.Color = Color.Red;
```

Back to Top

Delete Stamp

To delete a particular stamp annotation in PDF document, use RemoveAt method to remove the stamp by specifying its index value. Apart from this, Clear method can be used to remove all the stamp annotations from the document.

C#

```
// Delete a particular stamp annotation
page.Annotations.RemoveAt(0);

// Delete all stamp annotations
page.Annotations.Clear();
```

Back to Top

For more information about implementation of stamps using GcPdf, see GcPdf sample browser.

Tagged PDF

Tagged PDF is a PDF containing accessibility markup at the back end which provides it a logical structure that manages the reading order and presentation of the document content. It also allows you to extract the page content, such as text, images, etc., and reuse it. Tagged PDF makes it easy to read a PDF by screen reader software for users who rely on assistive technology. GcPdf allows you to create tagged PDF or structured PDF document by adding different structural elements to the document and rendering the content as marked by using the BeginMarkedContent and EndMarkedContent methods.

Create Tagged PDF files

To create a tagged PDF:

1. Create a Part element using the StructElement class.

2. Add the structure element to the document's logical structure, represented by StructTreeRoot class, using the Add method.
3. Create paragraph elements using the **StructElement** class and add it to the Part element.
4. Mark the beginning and end of the tagged content using BeginMarkedContent and EndMarkedContent methods of the GcPdfGraphics class respectively.
5. Add content item to the paragraph element.
6. Mark the document as tagged using MarkInfo.Marked property.
7. Save the tagged PDF using Save method of the GcPdfDocument class.

C#

```
public void CreateTaggedPdf()
{
    var doc = new GcPdfDocument();
    int pageCount = 5;

    // create Part element, it will contain P (paragraph) elements
    StructElement sePart = new StructElement("Part");
    doc.StructTreeRoot.Children.Add(sePart);

    // Add some pages, on each page add some paragraphs and tag them:
    for (int pageIndex = 0; pageIndex < pageCount; ++pageIndex)
    {
        // Add page:
        var page = doc.Pages.Add();
        var g = page.Graphics;
        const float margin = 36;
        const float dy = 18;

        // Add some paragraphs:
        int paraCount = 4;
        float y = margin;
        for (int i = 0; i < paraCount; ++i)
        {
            // Create paragraph element:
            StructElement seParagraph = new StructElement("P") { DefaultPage =
page };

            // Add it to Part element:
            sePart.Children.Add(seParagraph);

            // Create text layout:
            var tl = g.CreateTextLayout();
            tl.DefaultFormat.Font = StandardFonts.Helvetica;
            tl.DefaultFormat.FontSize = 12;
            tl.Append(i+1 + " .Test the pdf for tags");
            tl.MaxWidth = page.Size.Width;
            tl.MarginLeft = tl.MarginRight = margin;
            tl.PerformLayout(true);

            // draw TextLayout within tagged content
            g.BeginMarkedContent(new TagMcid("P", i));
            g.DrawTextLayout(tl, new PointF(0, y));
```

```
            g.EndMarkedContent();

            y += tl.ContentHeight + dy;

            // add content item to paragraph StructElement
            seParagraph.ContentItems.Add(new McidContentItemLink(i));
        }
    }

    // mark as tagged
    doc.MarkInfo.Marked = true;

    //Save the document
    doc.Save("TaggedPdf.pdf");
}
```

Back to Top

For more information about how to work with tagged PDF using GcPdf, see GcPdf sample browser.

Text

GcPdf provides the following two main approaches to render text through GcGraphics class which is a member of GrapeCity.Documents.Drawing namespace:

- **Using TextLayout/DrawTextLayout method**: The DrawTextLayout method is the main method for rendering text in GcPdf. It uses an instance of TextLayout class to draw the text layout at a specified location.
- **Using MeasureString/DrawString pair**: The DrawString method can be used in pair with MeasureString method to render a short string on a page at an arbitrary location when the string can fit in the available space.

In addition, GcPdf offers GrapeCity.Documents.Text namespace which supports the following features to work with text:

Text alignment

GcPdf provides TextAlignment property to control how text is aligned horizontally along the reading direction axis. The property takes the values from the TextAlignment enum.

Text layout

GcPdf provides TextLayout class that represents one or more paragraphs of text with same formatting. It can also be directly used for text shaping and layout.

Text formatting

GcPdf offers TextFormat class to format text and set font color and decorations. Font is the mandatory property that must be set on a text format. In addition, GcPdf allows you to mix different text formats in the same paragraph using **TextLayout** and GcPdfGraphics.DrawTextLayout method to draw the text layout at a specified location.

Text rotation

GcPdf allows text rotation in a PDF document using Transform property of GcPdfGraphics class to rotate a text string.

Vertical text

GcPdf allows rendering vertical text in **LeftToRight** and **RightToLeft** modes. The library provides FlowDirection property of **TextLayout** class which allows setting the direction of the text. Moreover, it allows you to

render vertical text for many common East Asian languages, such as Chinese, Japanese and Korean.

Text stroking and filling

GcPdf allows rendering text with stroked glyph outlines and filling glyphs with solid or gradient color in a PDF document using Hollow and FillBrush property of the TextFormat class.

Text trimming and wrapping

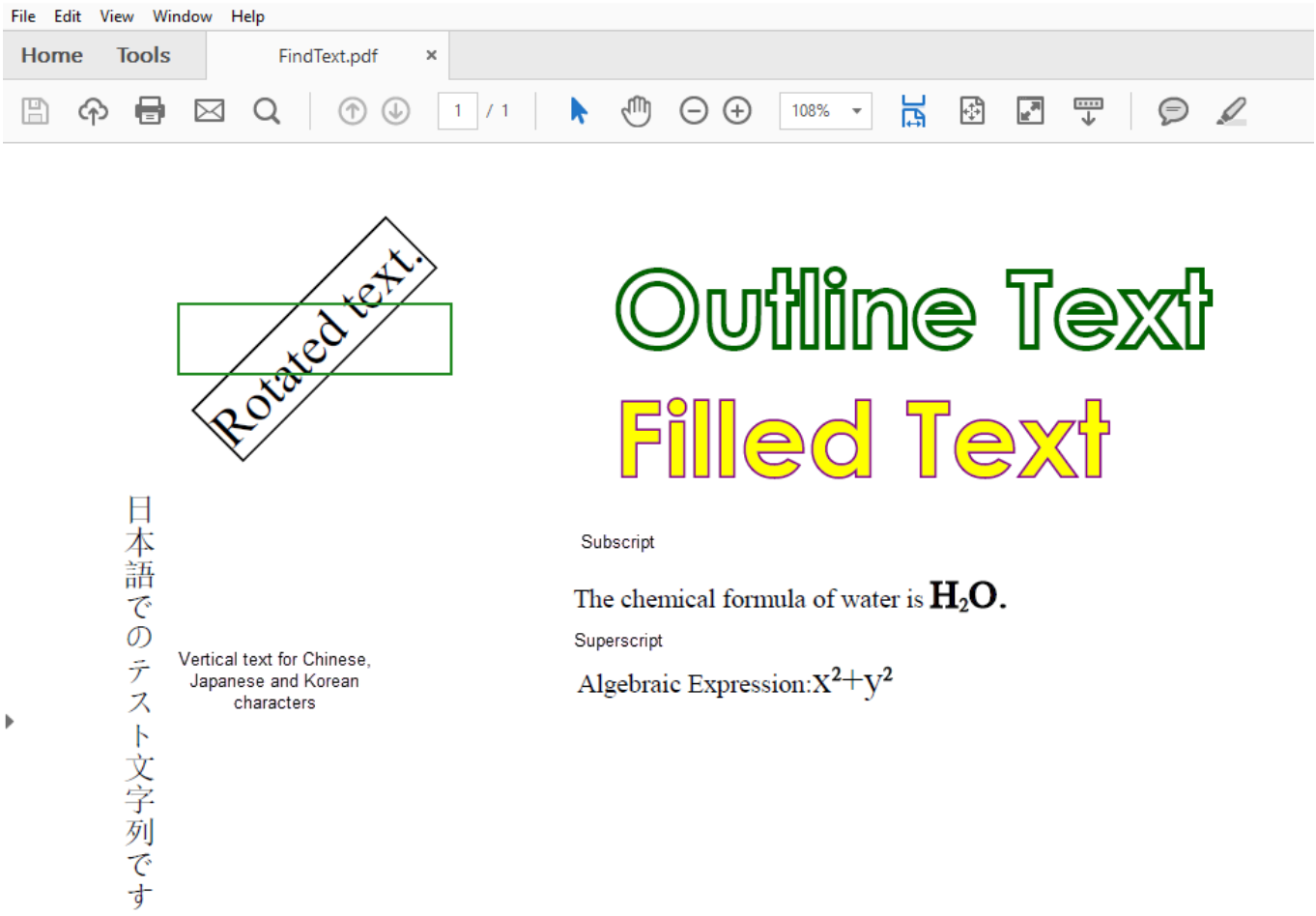
GcPdf offers TrimmingGranularity and WrapMode properties of the TextLayout class that allows trimming the text at word or character, and wrap text in a PDF document. This allows you to display ellipsis or any other character at the end of a text line that does not fit in the allocated space.

Subscript and superscript

GcPdf allows to display superscript and subscript texts through TextFormat class. This class provides Subscript property to set text as subscript (for example, "2" in H_2O) and Superscript property to set text as superscript (for example, "3" in x^3).

Paragraph alignment and formatting

GcPdf provides all the properties to align and format paragraph in **TextLayout** class. This class provides ParagraphAlignment property to set the alignment of paragraphs along the flow direction axis. The **ParagraphAlignment** property takes the values from ParagraphAlignment enum. In addition, basic paragraph formatting options such as line indentation and spacing can also be applied to the paragraphs using FirstLineIndent and LineSpacingScaleFactor properties of the **TextLayout** class.



Render Text

To render text in a PDF document using GcPdf, you can either use DrawString method provided by the GcGraphics class or the TextLayout class. In case you are using the TextLayout class, you need to create a layout using Append method, and then prepare it for rendering by calling the PerformLayout method. Finally, render the text in the document by calling the DrawTextLayout method provided by the GcGraphics class.

C#

```
public void CreatePDF(Stream stream)
{
    var doc = new GcPdfDocument();
    var page = doc.NewPage();
    var g = page.Graphics;
    const float In = 150;
    PointF ip = new PointF(72, 72);
    var tl = g.CreateTextLayout();
    tl.DefaultFormat.Font = StandardFonts.Times;
    tl.DefaultFormat.FontSize = 12;
    // TextFormat class is used throughout all GcPdf text rendering to specify
    // font and other character formatting:
    var tf = new TextFormat(tl.DefaultFormat)
    {
        Font = StandardFonts.Times,
        FontSize = 12
    };

    // Render text using Append method
    tl.Append("Turpis non ante pulvinar et massa nibh laoreet amet volutpat laoreet "
+
     "molestie aliquet massa ullamcorper ac nisi ante massa lobortis Massa at
laoreet" +
     "mauris aliquamfelis feugiat et non euismod magna eget molestie euismod elit
dolor " +
     "eget erat euismod laoreetPharetra sit mauris nibh molestie ac nunc proin felis"
+
     " erat lorem volutpat elit mi nunc magnamauris molestie tincidunt" +
     " sedMassa congue nibh volutpat eget non", tf);
    tl.PerformLayout(true);
    g.DrawTextLayout(tl, ip);

    // Render text using DrawString method
    g.DrawString("1. Test string.", tf, new PointF(In, In));

    // Save document
    doc.Save(stream);
}
```

Back to Top

Align Text

To set the text alignment in a PDF document, use TextAlignment property provided by the TextLayout class. This property accepts value from TextAlignment enum.

C#

```
tl.TextAlignment = TextAlignment.Trailing;
```

Back to Top

Format Text

To format text in a PDF document, use the TextFormat class. This class is used for any type of text rendering, to specify the font and other character formatting. You can use different properties, such as FontSize, FontStyle, etc. provided by the TextFormat class to apply required text format to the rendered text.

C#

```
TextFormat tf = new TextFormat()
{
  Font = StandardFonts.Courier,
  FontSize = 14,
  FontStyle = FontStyle.Bold,
  ForeColor = GrapeCity.Documents.Drawing.Color.Cyan,
  Language = Language.English
};
```

Back to Top

Rotate Text

To rotate text at various angles in a PDF document:

1. Rotate the text using Transform property provided by the GcGraphics class, which accepts the value calculated by the CreateRotation method of Matrix3x2 class.
2. Draw the rotated text and bounding rectangle using DrawTextLayout method.

C#

```
public void CreatePDF(Stream stream)
{
    // Rotation angle, degrees clockwise
    float angle = -45;
    var doc = new GcPdfDocument();
    var g = doc.NewPage().Graphics;
    // Create a text layout, pick a font and font size:
    TextLayout tl = g.CreateTextLayout();
    tl.DefaultFormat.Font = StandardFonts.Times;
    tl.DefaultFormat.FontSize = 24;
    // Add a text, and perform layout:
    tl.Append("Rotated text.");
    tl.PerformLayout(true);
    // Text insertion point at (1",1"):
    var ip = new PointF(72, 72);
    // Now that we have text size, create text rectangle
    var rect = new RectangleF(ip.X, ip.Y, tl.ContentWidth, tl.ContentHeight);
    // Rotate the text around its bounding rect's center:
    // we now have the text size, and can rotate it about its center:
    g.Transform = Matrix3x2.CreateRotation((float)(angle * Math.PI) / 180f,
```

```
    new Vector2(ip.X + tl.ContentWidth / 2, ip.Y + tl.ContentHeight / 2));
    // Draw rotated text and bounding rectangle:
    g.DrawTextLayout(tl, ip);
    g.DrawRectangle(rect, Color.Black, 1);
    // Remove rotation and draw the bounding rectangle
    g.Transform = Matrix3x2.Identity;
    g.DrawRectangle(rect, Color.ForestGreen, 1);
    // Save Document
    doc.Save(stream);
}
```

Back to Top

Vertical Text

GcPdf supports vertical text through FlowDirection property of the GcGraphics class which accepts value from the FlowDirection enumeration. To set the vertical text alignment, this property needs to be set to VerticalLeftToRight or VerticalRightToLeft.

Additionally, the TextFormat class of GcPdf provides you an option to rotate the sideways text in counter clockwise direction using the RotateSidewaysCounterclockwise property.
Further, SidewaysInVerticalText and UprightInVerticalText property of the TextFormat class also provides options to display the text sideways or upright respectively. These properties are especially useful for rendering Latin text within the East-Asian language text.

C#

```
// Set vertical text layout using TextLayout properties
tl.RotateSidewaysCounterclockwise = true;
tl.FlowDirection = FlowDirection.VerticalLeftToRight;

// Setup the vertical text layout for Chinese, Japanese and Korean characters
TextFormat tfvertical = new TextFormat()
{
 UprightInVerticalText = false,
 GlyphWidths = GlyphWidths.Default,
 TextRunAsCluster = false,
};
tl.Append("日本語でのテスト文字列です", tfvertical);
```

Back to Top

Outline Text and Fill Text

To render an outline text, draw the outline using the StrokePen property, and then set the Hollow property to **true**. And, in case of fill text, use the FillBrush property provided by the TextFormat class.

C#

```
// Outline Text
TextFormat tf0 = new TextFormat() {
        StrokePen = Color.DarkGreen,
        Hollow = true,
        FontSize = 48,
};
```

```
tl.AppendLine("Outline Text", tf0);

// Filled Text
TextFormat tf1 = new TextFormat() {
        StrokePen = Color.DarkMagenta,
        FillBrush = new SolidBrush(Color.Yellow),
        FontSize = 48,
};
tl.AppendLine("Filled Text", tf1);
```

Back to Top

Text Trimming and Wrapping

There are two ways of handling the text that does not fit into the available space; one is to wrap the text and other is to trim a character or a word and append it with a character such as ellipsis. To wrap the text in a PDFdocument, use the WrapMode property provided by the TextLayout class. This class also provides the TrimmingGranularity and EllipsisCharCode properties to set the trimming options and to display a particular character at the end of the text respectively.

C#

```
// Character trimming
tl.TrimmingGranularity = TrimmingGranularity.Character;

tl.EllipsisCharCode = 0x007E;

// Set wrap mode to character wrap
tl.WrapMode = WrapMode.CharWrap;
```

Back to Top

Subscript and Superscript

To render subscript and superscript text in a PDF document, use the Subscript and Superscript properties provided by the TextFormat class.

C#

```
//Apply Subscript
var tf = new TextFormat() {FontSize = 18};
var tfsub = new TextFormat() {Subscript = true, FontBold = true};
var tfbold = new TextFormat() {FontBold = true, FontSize = 18};
tl.Append("The chemical formula of water is ");
tl.Append("H", tfbold);
tl.Append("2", tfsub);
tl.Append("O.", tfbold);


//Apply Superscript
var tf = new TextFormat() {FontSize = 18};
var tfsup = new TextFormat() {Superscript = true, FontBold = true};
tl.Append("Example of a math equation : ");
tl.Append("x", tf);
tl.Append("2", tfsup);
```

```
tl.Append("+", tf);
tl.Append("y", tf);
tl.Append("2", tfsup);
```

Back to Top

Handle Paragraph

To handle paragraph formatting, use the properties provided by TextLayout class to set the paragraph alignment and formatting.

C#

```
public void CreatePDF(Stream stream)
{
    var doc = new GcPdfDocument();
    var page = doc.NewPage();
    var g = page.Graphics;
    // By default, GcPdf uses 72dpi:
    PointF ip = new PointF(72, 72);
    var tl = g.CreateTextLayout();

    tl.MaxWidth = doc.PageSize.Width;
    tl.MaxHeight = doc.PageSize.Height;
    tl.MarginLeft = tl.MarginTop = tl.MarginRight = tl.MarginBottom = 72;
    var tf = new TextFormat(tl.DefaultFormat)
    {
        Font = StandardFonts.Times,
        FontSize = 12,
    };

    // Render text using Append method
    tl.Append("Turpis non ante pulvinar et massa nibh laoreet amet volutpat laoreet "
+
     "molestie aliquet massa ullamcorper ac nisi ante massa lobortis Massa at
laoreet" +
     "mauris aliquamfelis feugiat et non euismod magna eget molestie euismod elit
dolor" +
     "eget erat euismod laoreetPharetra sit mauris nibh molestie ac nunc proin felis"
+
     "erat lorem volutpat elit mi nunc magnamauris molestie tincidunt" +
     "sedMassa congue nibh volutpat eget non", tf);

    // Set first line offset
    tl.FirstLineIndent = 72 / 2;
    // Set line spacing
    tl.LineSpacingScaleFactor = 1.5f;
    tl.PerformLayout(true);
    g.DrawTextLayout(tl, ip);
    // Save document
    doc.Save(stream);
}
```

Back to Top

Extract Text

To extract text from a PDF:

1. Load a PDF document using Load method of the GcPdfDocument class.
2. Extract text from the last page of the PDF using GetText method of the Page class.
3. Add the extracted text to another PDF document using the Graphics.DrawString method.
4. Save the document using Save method of the GcPdfDocument class.

C#

```
GcPdfDocument doc = new GcPdfDocument();

FileStream fs = new FileStream("GcPdf.pdf",FileMode.Open,FileAccess.Read);
doc.Load(fs);

//Extract text present on the last page
String text=doc.Pages.Last.GetText();

//Add extracted text to a new pdf
GcPdfDocument doc1 = new GcPdfDocument();
PointF textPt = new PointF(72, 72);
doc1.NewPage().Graphics.DrawString(text, new TextFormat()
        { FontName = "ARIAL", FontItalic = true }, textPt);

doc1.Save("NewDocument.pdf");

Console.WriteLine("Press any key to exit");
Console.ReadKey();
```

Similarly, you can also extract all the text from a document by using GetText method of the GcPdfDocument class.

Extract Text using ITextMap

GcPdf provides ITextMap interface that represents the text map of a page in a GcPdf document. It helps you to find the geometric positions of the text lines on a page and extract the text from a specific position.

The text map for a specific page in the document can be retrieved using the GetTextMap method of the Page class, which returns an object of type **ITextMap**. ITextMap provides four overloads of the GetFragment method, which helps to retrieve the text range and the text within the range. The text range is represented by the TextMapFragment class and each line of text in this range is represented by the TextLineFragment class.

The example code below uses the GetFragment(out TextMapFragment range, out string text) overload to retrieve the geometric positions of all the text lines on a page and the GetFragment(MapPos startPos, MapPos endPos, out TextMapFragment range, out string text) overload to retrieve the text from a specific position in the page.

C#

```
// Open an arbitrary PDF, load it into a temp document and use the map to find some
texts:
using (var fs = new FileStream("Test.pdf", FileMode.Open, FileAccess.Read))
{
    var doc1 = new GcPdfDocument();
    doc1.Load(fs);
```

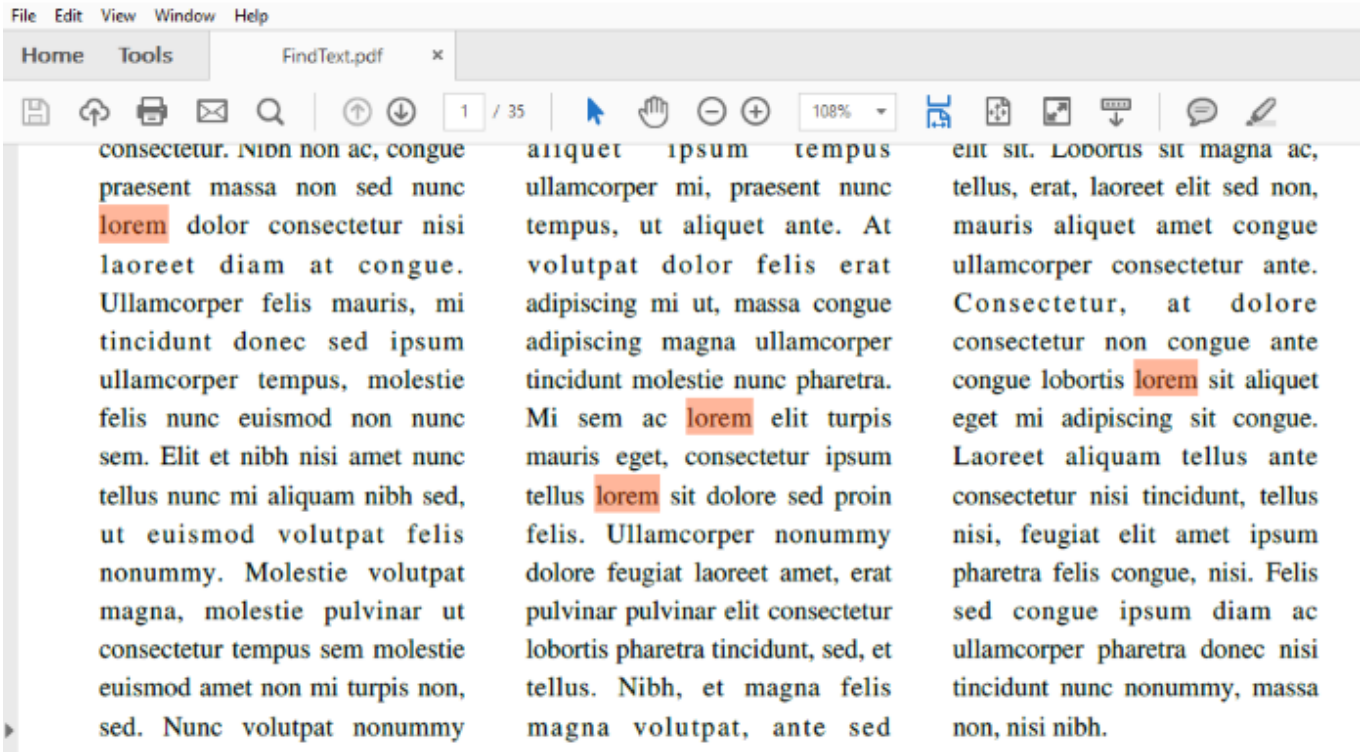
```
    var tmap = doc1.Pages[0].GetTextMap();

    // We retrieve the text at a specific (known to us) geometric location on the
page:
    float tx0 = 2.1f, ty0 = 3.37f, tx1 = 3.1f, ty1 = 3.5f;
    HitTestInfo htiFrom = tmap.HitTest(tx0 * 72, ty0 * 72);
    HitTestInfo htiTo = tmap.HitTest(ty0 * 72, ty1 * 72);
    tmap.GetFragment(htiFrom.Pos, htiTo.Pos, out TextMapFragment range1, out string
text1);
    tl.AppendLine($"Looked for text inside rectangle x={tx0:F2}\", y={ty0:F2}\", " +
        $"width={tx1 - tx0:F2}\", height={ty1 - ty0:F2}\", found:");
    tl.AppendLine(text1);
    tl.AppendLine();

    // Get all text fragments and their locations on the page:
    tl.AppendLine("List of all texts found on the page");
    tmap.GetFragment(out TextMapFragment range, out string text);
    foreach (TextLineFragment tlf in range)
    {
        var coords = tmap.GetCoords(tlf);
        tl.Append($"Text at ({coords.B.X / 72:F2}\",{coords.B.Y / 72:F2}\"):\t");
        tl.AppendLine(tmap.GetText(tlf));
    }
    // Print the results:
    tl.PerformLayout(true);
}
```

Back to Top

For more information about implementation of text properties using GcPdf, see GcPdf sample browser.

Text Search

GcPdf allows text search in a PDF document using FindText method of GcPdfDocument class. This method accepts object of FindTextParams and OutputRange class as parameters to find all the occurrences of the searched string in the loaded document.



Search Text

To search text in PDF document:

1. Create an object of GcPdfDocument class.
2. Load any existing PDF file using the Load method.
3. Use the FindText method to perform text search.

C#

```
public void CreatePDF(Stream stream)
{
    var doc = new GcPdfDocument();

    // The original file stream must be kept open while working with the loaded PDF
    using (var fs = new
FileStream(Path.Combine("Resources","PDFs","BalancedColumns.pdf"),
      FileMode.Open, FileAccess.Read))
    {
        doc.Load(fs);
        // Find all 'lorem', using case-insensitive word search:
        var findsLorem = doc.FindText
       (new FindTextParams("lorem", true, false), OutputRange.All);

        // Highlight all 'lorem' using semi-transparent orange red:
        foreach (var find in findsLorem)
            doc.Pages[find.PageIndex].Graphics.FillPolygon
           (find.Bounds, Color.FromArgb(100, Color.OrangeRed));
```

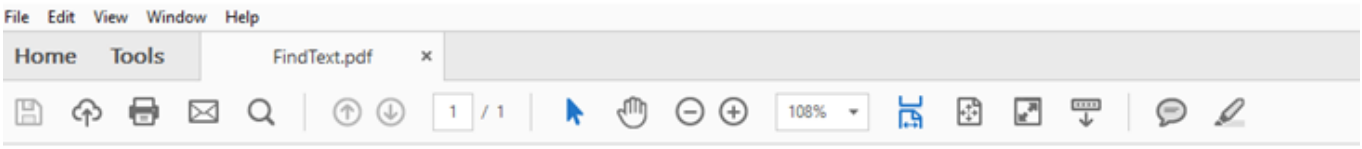
```
        // Done:
        doc.Save(stream);
    }
```

For more information about implementation of text search using GcPdf, see GcPdf sample browser.

Watermark

Watermarks are one of the ways to add security to your documents that contain highly confidential information or can be used to verify if a PDF document is a copy or original, or from the authorized company. These are similar to stamps but cannot be moved or changed like stamps. For more information on watermarks, see PDF specification 1.7 (Section 12.5.6.22).

GcPdf library provides easy mechanism to add watermarks to your PDF documents with WatermarkAnnotation class and provides additional properties to enhance them.



Add Watermark

To add a watermark in a PDF document, use the WatermarkAnnotation class. The WaterMarkAnnotation class provides the essential properties for creating an image based watermark.

To add a watermark:

1. Create an object of GcPdfDocument and WaterMarkAnnotation class.
2. Set the required properties of WaterMarkAnnotation object.

3. Call the Add method to add the watermark on the page.

Note: If both Annotations.WatermarkAnnotation.Text and Annotations.WatermarkAnnotation.Image are specified then WatermarkAnnotation.Image is used as watermark content.

C#

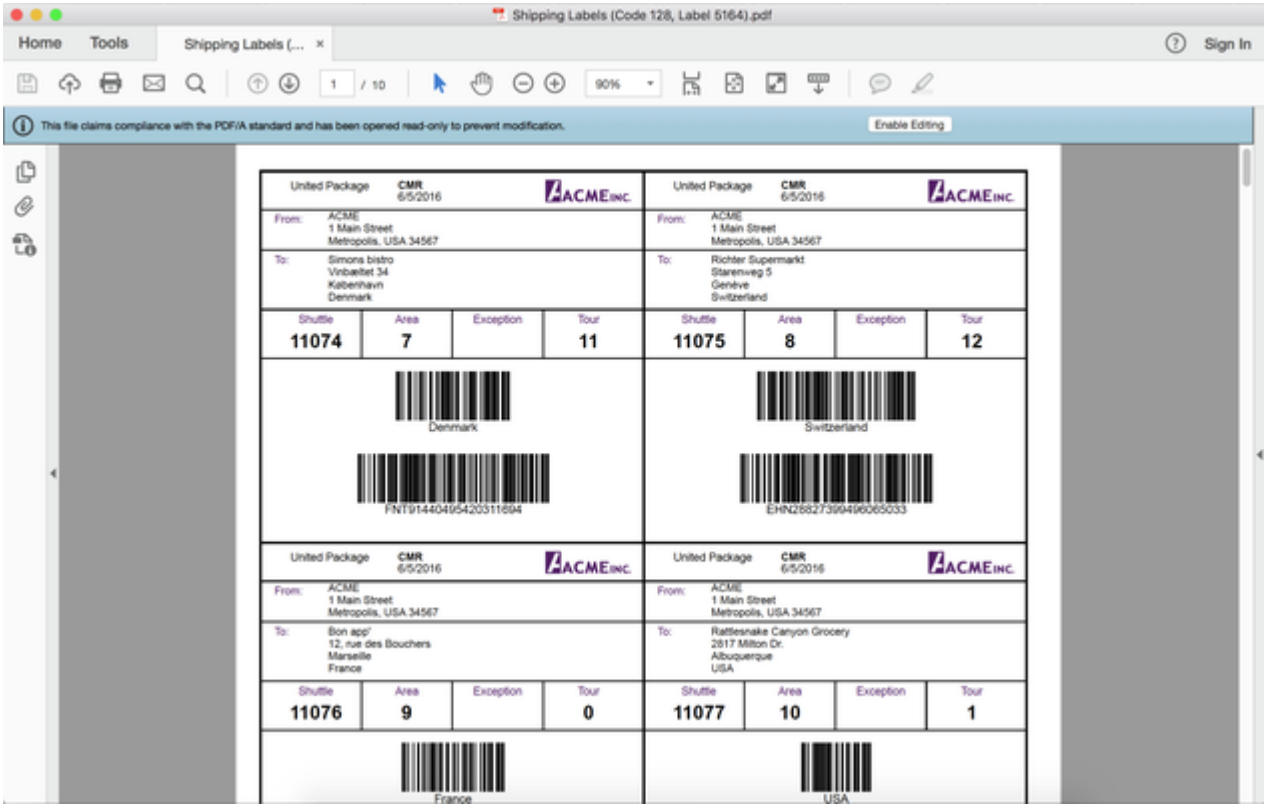
```
public void CreatePDF(Stream stream)
{
    GcPdfDocument doc = new GcPdfDocument();
    var page = doc.NewPage();
    var g = page.Graphics;

    TextFormat tf = new TextFormat()
    {
        Font = StandardFonts.HelveticaBold,
        FontSize = 72
    };

    var watermark = new WatermarkAnnotation()
    {
        Name = "WaterMark Sample",
        Image = Image.FromFile(@ "puffins.jpg"),
        Rect = new RectangleF(100.5F, 110.5F, 500, 250),
        TextFormat = tf,
        Text = "DraftCopy",
    };

    // Add watermark to page
    page.Annotations.Add(watermark);
    doc.Save(stream);
}
```

Back to Top

Barcodes in PDF

Apart from text, images, tables, annotations etc, you might need to add barcodes to your PDF documents. Barcodes can be helpful when you create PDF for inventory management, ticketing system, advertising, invoice, shipping labels etc.

GcPdf supports barcodes through GcBarcode class under **GrapeCity.Documents.Barcode** namespace that belongs to **GrapeCity.Documents.Barcode.dll**, an independent assembly that exclusively contains barcode related methods and properties. GcBarcode references are available through NuGet package named GrapeCity.Documents.BarCode. Also, note that GrapeCity.Documents.Barcode.dll is not a part of GcPdf and hence, there are no dependencies between GcPdf and GcBarcode. The assembly just adds extension methods to GcGraphics that allow to draw barcodes on any GcGraphics implementation including GcPdfGraphics.



Supported barcode symbologies

GcPdf offers 38 different barcode symbologies (1D and 2D) or code types which are described in the table below. The **GcBarCode** assembly provides CodeType property which accepts the values from CodeType enum to set the type of barcode to any of these listed barcode types.

Barcode type	Description
Ansi39	ANSI 3 of 9 (Code 39) uses upper case, numbers, - , * $ / + %. This is the default barcode style.
Ansi39x	ANSI Extended 3 of 9 (Extended Code 39) uses the complete ASCII character set.
Bc412	The BC412 barcode was invented by IBM to meet the needs of the semiconductor wafer identification application.
Codabar	Codabar uses A B C D + - : . / and numbers.

Code11	Code11, also known as USD-8, is a high-density barcode symbology developed by Intermec in 1977. It is primarily used to label telecommunication equipments. This symbology is discrete and is able to encode numeric digits through 0-9, dash (-), and start/stop characters.
Code_128_A	Code 128 A uses control characters, numbers, punctuation, and upper case.
Code_128_B	Code 128 B uses punctuation, numbers, upper case, and lower case.
Code_128_C	Code 128 C uses only numbers.
Code_128auto	Code 128 Auto uses the complete ASCII character set. Automatically selects between Code 128 A, B, and C to give the smallest barcode.
Code_2_of_5	Code 2 of 5 uses only numbers.
Code93	Code 93 uses uppercase, % $ * / , + -, and numbers.
Code25intlv	Interleaved 2 of 5 uses only numbers.
Code39	Code 39 uses numbers, % * $ /. , - +, and upper case.
Code39x	Extended Code 39 uses the complete ASCII character set.
Code49	Code 49 is a two-dimensional high-density stacked barcode containing two to eight rows of eight characters each. Each row has a start code and a stop code. Encodes the complete ASCII character set.
Code93x	Extended Code 93 uses the complete ASCII character set.
DataMatrix	Data Matrix is a high density, two-dimensional barcode with square modules arranged in a square or rectangular matrix pattern.
EAN_13	EAN-13 uses only numbers (12 numbers and a check digit). It takes only 12 numbers as a string to calculate a check digit (CheckSum) and add it to the thirteenth position. If there are thirteen numbers, it validates the checksum and throws an error if it is incorrect.
EAN_8	EAN-8 uses only numbers (7 numbers and a check digit).
EAN128FNC1	EAN-128 is an alphanumeric one-dimensional representation of Application Identifier (AI) data for marking containers in the shipping industry. This type of bar code contains the following sections: • Leading quiet zone (blank area) • Code 128 start character • FNC (function) 1 character which allows scanners to identify this as an EAN-128 barcode • Data (AI plus data field) • Symbol check character (Start code value plus product of each character position plus value of each character divided by 103. The checksum is the remainder value.) • Stop character • Trailing quiet zone (blank area) The AI in the Data section sets the type of the data to follow (i.e. ID, dates, quantity, measurements, etc.). There is a specific data structure for each type of data. This AI is what distinguishes the EAN-128 code from Code 128. Multiple AIs (along with their data) can be combined into a single bar code.

	EAN128FNC1 is a UCC/EAN-128 (EAN128) type barcode that allows you to insert FNC1 character at any place and adjust the bar size, etc., which is not available in UCC/EAN-128. To insert FNC1 character, set “\n” (for C#), or “vbLf” (for VB) to Text property at runtime.
HIBCCode128	HIBCCode128 is a Health Industry Bar Code 128 implementation.
HIBCCode39	HIBCCode39 is a Health Industry Bar Code 39 implementation.
Iata25	Represents an IATA 2 of 5 barcode.
IntelligentMail	Intelligent Mail, formerly known as the 4-State Customer Barcode, is a 65-bar code used for domestic mail in the U.S.
IntelligentMailPackage	Intelligent Mail Package Barcode.
ISBN	The International Standard Book Number (ISBN) is special commercial book identifier which encodes 9 numeric digits apart from the start number "978", "979".
ISMN	The International Standard Music Number or ISMN (ISO 10957) is a thirteen-character alphanumeric identifier for printed music developed by ISO.
ISSN	The International Standard Serial Number (ISSN) is an eight-digit number used for printed or electronic periodical publications like magazines, etc. This ISSN system was drafted as an International Standard in 1971 and published as ISO 3297 in 1975.
ITF14	ITF14 barcode is the GS1 implementation of an Interleaved 2 of 5 bar code to encode a Global Trade Item Number. It is continuous, self-checking, bidirectionally decodable and it will always encode 14 digits. ITF14 is used on packaging levels of a product in general.
JapanesePostal	This is the barcode used by the Japanese Postal system. Encodes alpha and numeric characters consisting of 20 digits including a 7-digit postal code number, optionally followed by block and house number information. The data to be encoded can include hyphens.
Matrix_2_of_5	Matrix 2 of 5 is a higher density barcode consisting of three black bars and two white bars.
MicroPDF417	MicroPDF417 is two-dimensional, multi-row symbology, derived from PDF417. Micro-PDF417 is designed for applications that need to encode data in a two-dimensional symbol (up to 150 bytes, 250 alphanumeric characters, or 366 numeric digits) with the minimal symbol size. MicroPDF417 allows you to insert an FNC1 character as a field separator for variable length Application Identifiers (AIs). To insert FNC1 character, set “\n” (for C#), or “vbLf” (for VB) to Text property at runtime.
MicroQRCode	MicroQRCode is a variant of QR Code 2005. Compared with other regular QR Codes, it has only one position detection pattern which reduces the barcode size so that it can be used to applications where the space for barcode image is severely restricted.
MSI	MSI Code uses only numbers.
Pdf417	Pdf417 is a popular high-density two-dimensional symbology that encodes up to 1108 bytes of information. This barcode consists of a stacked set of smaller barcodes. Encodes the full ASCII character set. It has ten error correction levels and

	three data compaction modes: Text, Byte, and Numeric. This symbology can encode up to 2725 data characters.
Pharmacode	Pharmacode, also known as Pharmaceutical Binary Code, is a barcode standard, 1D barcode that is used in the pharmaceutical manufacturing industry as a packing control system.
Plessey	MSI barcode, also known as Modified Plessey, is a numeric symbology developed by the MSI Data Corporation, which is used primarily for marking retail shelves for inventory control. Though continuous and self-checking, MSI Plessey provides several module checksum situations.
PostNet	PostNet uses only numbers with a check digit.
PZN	PZN or Pharma-Zentral-Nummer is a barcode standard used in the German pharmaceutical industry for identification of medicines and health-care products.
QRCode	QRCode is a two-dimensional symbology that is capable of handling numeric, alphanumeric and byte data as well as Japanese kanji and kana characters. This symbology can encode up to 7,366 characters.
RM4SCC	Royal Mail RM4SCC uses only letters and numbers (with a check digit). This is the barcode used by the Royal Mail in the United Kingdom.
RSS14	RSS14 is a 14-digit Reduced Space Symbology that encodes Composite Component (CC) extended EAN and UPC information in less space. This version is EAN.UCC item identification for use with omnidirectional point-of-sale scanners.
RSS14Stacked	RSS14Stacked symbology encodes CC extended EAN and UPC information in less space. This version is same as RSS14Truncated, but stacked in two rows for a smaller width. RSS14Stacked allows you to set Composite Options, where you can select the type of the barcode in the **Type** drop-down list and the value of the composite barcode in the **Value** field.
RSS14StackedOmnidirectional	RSS14StackedOmnidirectional symbology encodes CC extended EAN and UPC information in less space. This version is same as RSS14, but stacked in two rows for a smaller width.
RSS14Truncated	RSS14Truncated symbology encodes CC extended EAN and UPC information in less space. This version is a 14-digit EAN.UCC item identification and Indicator digits of zero or one for use on small items not for point-of-sale scanners.
RSSExpanded	RSSExpanded symbology encodes CC extended EAN and UPC information in less space. This version is a 14-digit EAN.UCC item identification and adds AI element strings such as, weight and best-before dates, for use with omnidirectional point-of-sale scanners. RSSExpanded allows you to insert an FNC1 character as a field separator for variable length Application Identifiers (AIs). To insert FNC1 character, set “\n” (for C#), or “vbLf” (for VB) to Text property at runtime.
RSSExpandedStacked	RSSExpandedStacked symbology encodes CC extended EAN and UPC information in less space. This version is same as RSSExpanded, but stacked in two rows for a smaller width. RSSExpandedStacked allows you to insert an FNC1 character as a field separator for

	variable length Application Identifiers (AIs). To insert FNC1 character, set “\n” (for C#), or “vbLf” (for VB) to Text property at runtime.
RSSLimited	RSS Limited symbology encodes CC extended EAN and UPC information in less space. This version is a 14-digit EAN.UCC item identification with indicator digits of 0 to 1 in small symbol that is not scanned by point-of-sale scanners. RSSLimited allows you to set Composite Options, where you can select the type of the barcode in the **Type** drop-down list and the value of the composite barcode in the **Value** field.
SSCC18	Serial Shipping Container Code-18 (SSCC-18) Barcode is a type of barcode that can print in the lower 2-inch (or local equivalent) extended area of the Thermal 4" x 8" or 4" x 8¼" (or local equivalent) label.
Telepen	Telepen is a name of a barcode symbology designed in the UK, in 1972, to directly represent the full ASCII character set without using shift characters for code switching, and use only two different widths for bars and spaces.
UCCEAN128	UCC/EAN –128 uses the complete ASCII character Set. This is a special version of Code 128 used in HIBC applications.
UPC_A	UPC-A uses only numbers (11 numbers and a check digit).
UPC_E0	UPC-E0 uses only numbers. Used for zero-compression UPC symbols. For the Caption property, you may enter either a six-digit UPC-E code or a complete 11-digit (includes code type, which must be zero) UPC-A code. If an 11-digit code is entered, the Barcode control will convert it to a six-digit UPC-E code, if possible. If it is not possible to convert from the 11-digit code to the six-digit code, nothing is displayed.
UPC_E1	UPC-E1 uses only numbers. Used typically for shelf labeling in the retail environment. The length of the input string for U.P.C. E1 is six numeric characters.

Back to Top

Barcode properties

The **GcBarcode** class provides the following common properties for all the barcode types.

Properties	Description
CodeType	Allows you to set the barcode encoding
HorizontalAlignment	Allows you to set the horizontal alignment of a barcode
Options	Gets the BarcodeOptions object to define the additional barcode options
ScaleFactor	Allows you to set the scale factor applied to a barcode image
Text	Allows you to provide the value to be encoded into barcode
TextFormat	Allows you to set the text format to draw the barcode label
VerticalAlignment	Allows you to set the vertical alignment of a barcode

Back to Top

Add Barcodes

To add barcode using GcPdf:

1. Create an object of GcBarcode class.
2. Set the required properties of the GcBarcode object.
3. Draw the barcode using DrawBarcode method provided by the GcPdfGraphics class.

Barcode.cs

```
public void CreatePDF(Stream stream)
   {
      GcPdfDocument doc = new GcPdfDocument();
      var page = doc.NewPage();
      var g = page.Graphics;
      GcBarcode barcode = new GcBarcode()
      {
        CodeType = CodeType.QRCode,
        Text = "QR Code",
      };
      barcode.TextFormat.Font = StandardFonts.Helvetica;
      barcode.Options.TextAlign = TextAlignment.Center;
      barcode.Options.QRCode.ConnectionNumber = 123456;
      g.DrawBarcode(barcode, new RectangleF(72/2, 72/2, 72, 72));
      doc.Save(stream);
    }
```

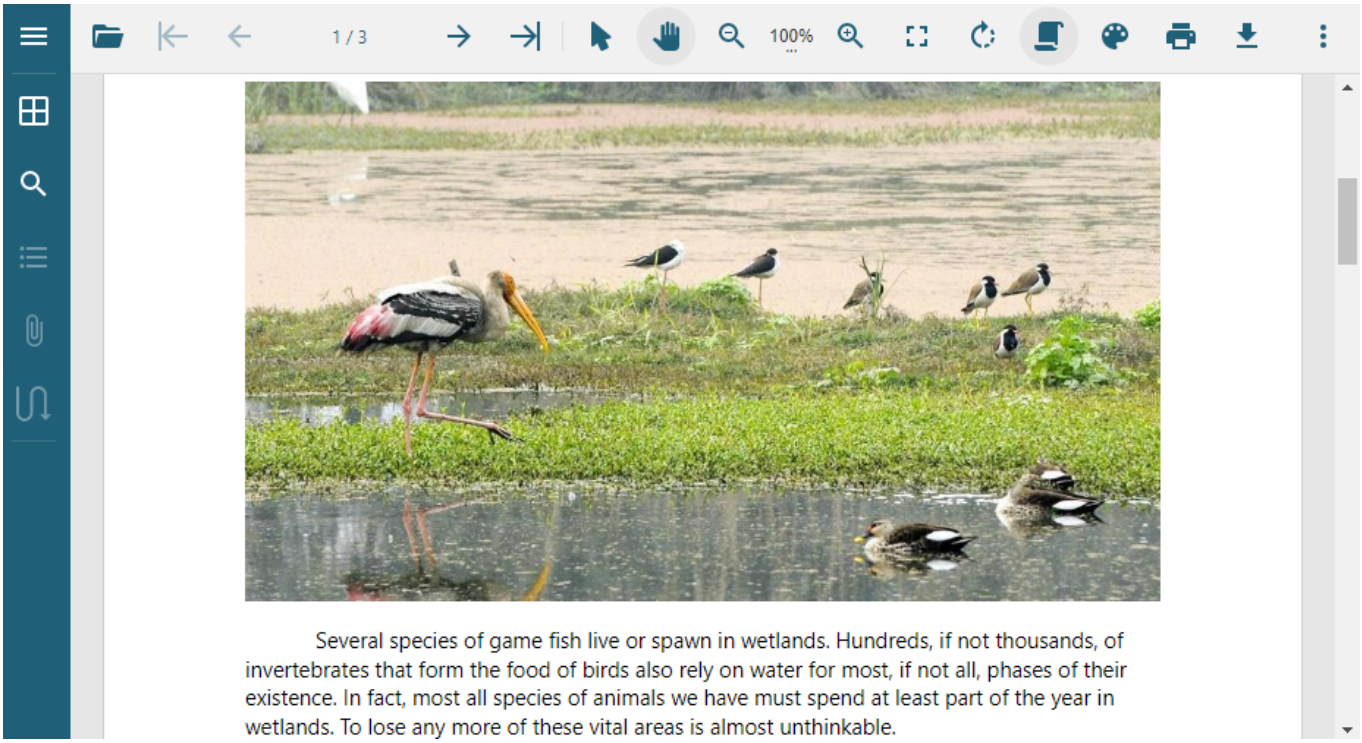
Back to Top

For more information about implementation of barcodes in GcPdf, see GcPdf sample browser.

GrapeCity Documents PDF Viewer

GrapeCity Documents PDF Viewer (GcDocs PDF Viewer) is a fast javascript based client-side viewer that can work on any major browser. It is a cross platform solution for viewing PDF files on Windows, MAC, Linux, iOS and Android devices in respective browsers. The GcDocs PDF Viewer can be conveniently embedded in major web frameworks such as Pure Javascript, Angular, Vue, ASP.NET Core, ASP.NET MVC, HTML5, React and Preact.

By using GrapeCity Documents for PDF with the GcPDF API, you can achieve full-fledged PDF needs of your application and can also load several real time PDFs based on Adobe PDF specification 1.7. The power of a server-side API and client-side viewer lets you implement full workflow of an application to collect user inputs and store them as PDF documents.



GcDocs PDF Viewer supports many standard PDF features:

- **Fill, submit and reset forms**
  The GcDocs PDF Viewer supports filling, submitting and resetting filled forms. To save them as a PDF on server, you can also use GrapeCity Documents for PDF API on the server.
- **Print filled forms**
  The GcDocs PDF Viewer allows you to directly print the filled-in forms from the Print option.
- **Print rotated document**
  The GcDocs PDF Viewer enables a user to rotate the document pages and directly print the rotated document.
- **Display page label titles**
  The GcDocs PDF Viewer supports the display of page label titles, so that you can distinguish the content topic in the document.
- **Annotations**
  The GcDocs PDF Viewer supports many annotations in the PDF document, without loss of any properties.
- **JavaScript actions**
  The GcDocs PDF Viewer supports JavaScript actions related to form fields, buttons and document.
- **Outline panel**
  The GcDocs PDF Viewer provides outline panel to list outlines and navigate to different positions in the document.
- **Text selection using caret**

The GcDocs PDF Viewer supports selecting horizontal text, vertical and RTL text with the help of default text selection caret.

- **Password-protected documents**
  The GcDocs PDF Viewer supports documents that are password protected and lets you open PDF file through password input dialog.
- **Page-level and document-level attachments.**
  The GcDocs PDF Viewer supports both page-level and document-level attachments. The user can double click the attachment files to open the attachments.
- **Article threads**
  The GcDocs PDF Viewer supports navigating through article threads in a PDF file via a separate panel in the sidebar.

Licensing and Redistribution

License Information

GrapeCity Documents for GcDocs PDF Viewer supports the following types of license:

- **Evaluation License**
- **Licensed**

Evaluation License

You can obtain a free 30-day evaluation key by contacting us.sales@grapecity.com. The evaluation version is fully functional and displays the below watermark:

'Powered by GrapeCity Documents PDF Viewer. Your temporary deployment key expires in [x] days.'

The evaluation key will allow you to develop and test your application on both your development machine and staging server for 30 days.

Licensed

Once you purchase the license, you will receive a license key that removes all watermarks.

How to apply your license key

To apply evaluation/production license in GcDocs PDF Viewer, set GcPdfViewer Deployment key to GcPdfViewer.License property before creating and initialising GcPdfViewer:

```
<script>
     // Add your license
     GcPdfViewer.LicenseKey = 'your_license_key';
     // Add your code
     window.onload = function(){
      const viewer = new GcPdfViewer("#viewer1", { file: 'helloworld.pdf' });
      viewer.addDefaultPanels();
     }
    </script>
```

This must precede the code that references the js files:

```
<!DOCTYPE html>
```

```
<html lang="en">
<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-
fit=no">
    <meta name="theme-color" content="#000000">
    <title>GC Viewer Demo | PDF Plugin</title>
    <script>
        function loadPdfViewer(selector) {
            GcPdfViewer.LicenseKey = 'your_license_key';
            var viewer = new GcPdfViewer(selector, { renderInteractiveForms: true /*,
documentListUrl: "/documentslist.json" */ });
            //viewer.addDocumentListPanel();
            viewer.addDefaultPanels();
        }
    </script>
</head>
<body onload="loadPdfViewer('#root')">
<div id="root"></div>
<script type="text/javascript" src="gcpdfviewer.js"></script><script
type="text/javascript" src="gcpdfviewer.vendor.js"></script></body>
</html>
```

Redistribution

Please review this information concerning redistribution of GcDocs PDF Viewer.

Script Files

- gcpdfviewer.js and gcpdfviewer.worker.js

CSS Files

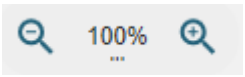
- dark-yellow.css, light-blue.css, and viewer.css
  These files can be omitted, if you use default theme. If you want to use a different theme, make a themes subfolder in the folder where the viewer files are already placed. Place the theme css files in the subfolder. This will allow it to work automatically. In order to specify an alternative location in the code, see the version.txt in the downloaded zip file.

GcDocs PDF Viewer Features

GcDocs PDF Viewer supports standard viewer features which can be accessed by either using the toolbar options in the toolbar displayed at the top or the feature specific panels available in the side bar.

The key features of toolbar and side panel for GrapeCity Documents PDF Viewer are listed below.

Features	Toolbar Icons	Description

Open PDF file		Enables you to open a PDF file in the Viewer.
Easy page navigation with Pan tool		Allows you to view the page by dragging it up or down.
Zoom in and zoom out PDF document		Enables you to zoom in and zoom out the PDF pages, and set the zoom percentage.
Switch to Full screen		Enables you to toggle to full-screen mode and access the mini-toolbar from the bottom of the Viewer window.
Print PDF document		Allows you to print PDF files in the Viewer.
Single page and Continuous view mode		Enables you to view one page at a time, with no portion of other pages visible for Single page mode, and view all pages in a continuous vertical column for Continuous mode.
Rotate PDF document		Allows you to rotate pages in a PDF file.
Built-in Viewer themes		Enables you to choose from three different themes in the Viewer.
Download PDF document		Enables you to download the PDF file you want to view in the Viewer.
Navigate to first and last pages	 	Enables you to navigate instantly to the first and last pages with page navigation icons in the Toolbar.
View the current page number		Allows you to display and set the number of the current page being previewed in the Viewer.
Navigate between preceding/succeeding pages	 	Enables you to navigate through all the pages with page navigation icons in the Toolbar.
Select text		Allows you to select text or rows of text in the PDF Viewer.

Features	Panel Icons	Description
Thumbnail navigation		Allows you to see the preview of all available pages in the PDF document.
Advanced search option		Allows you to search for text with match-case and whole-word search options.
Page-level and document-level attachments		Allows you to view the attachments in the left pane and open the attachments by double-clicking the attachment files.
Article thread navigation		Enables you to navigate with article threads in a PDF document through a separate panel.
Bookmark navigation		Enables you to list the outlines/bookmarks and navigate to different positions in the document.
View sidebar options		Allows you to view sidebar options with complete names of the options.

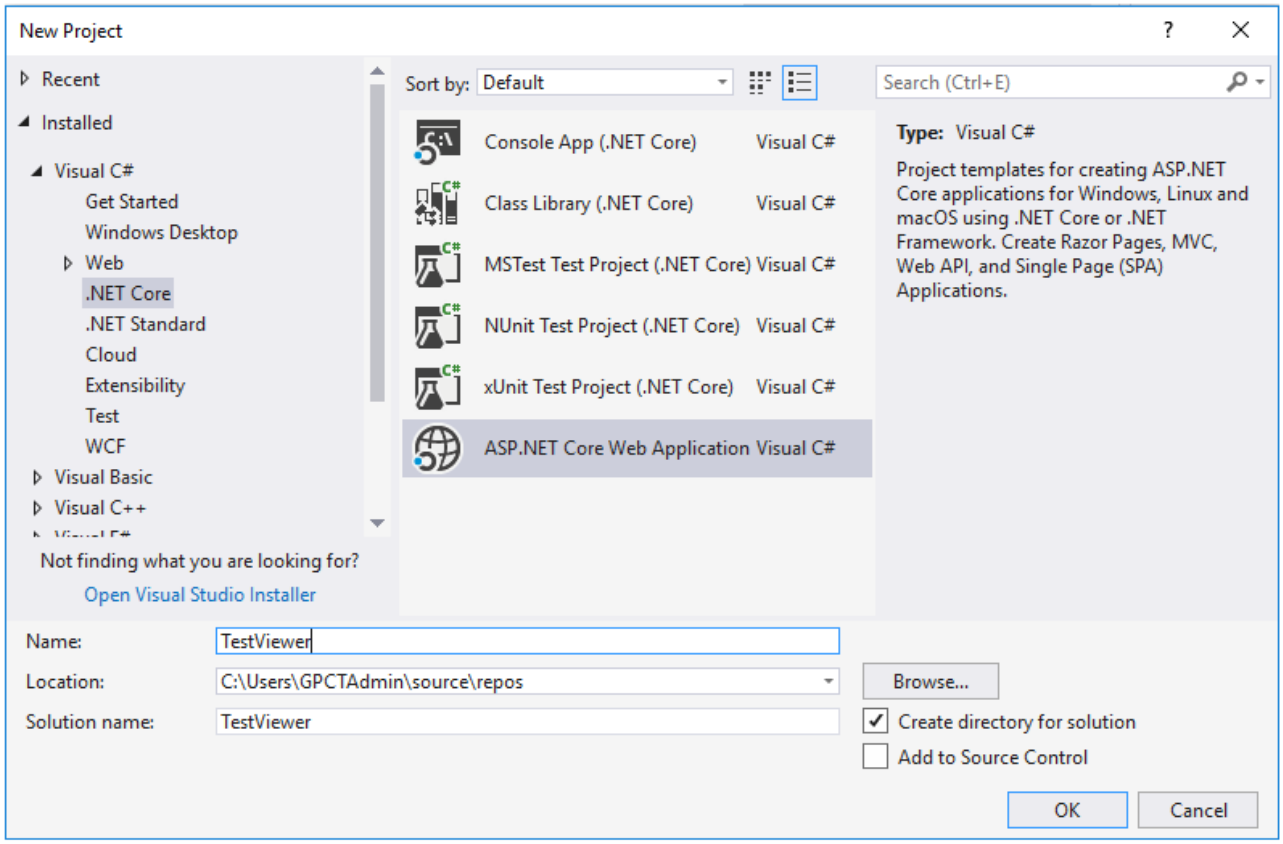
To see demo of GcDocs PDF Viewer, visit the GcPdf Sample browser. Here, you can see all the PDF features that are supported and running in the viewer.

Configure GcDocs PDF Viewer

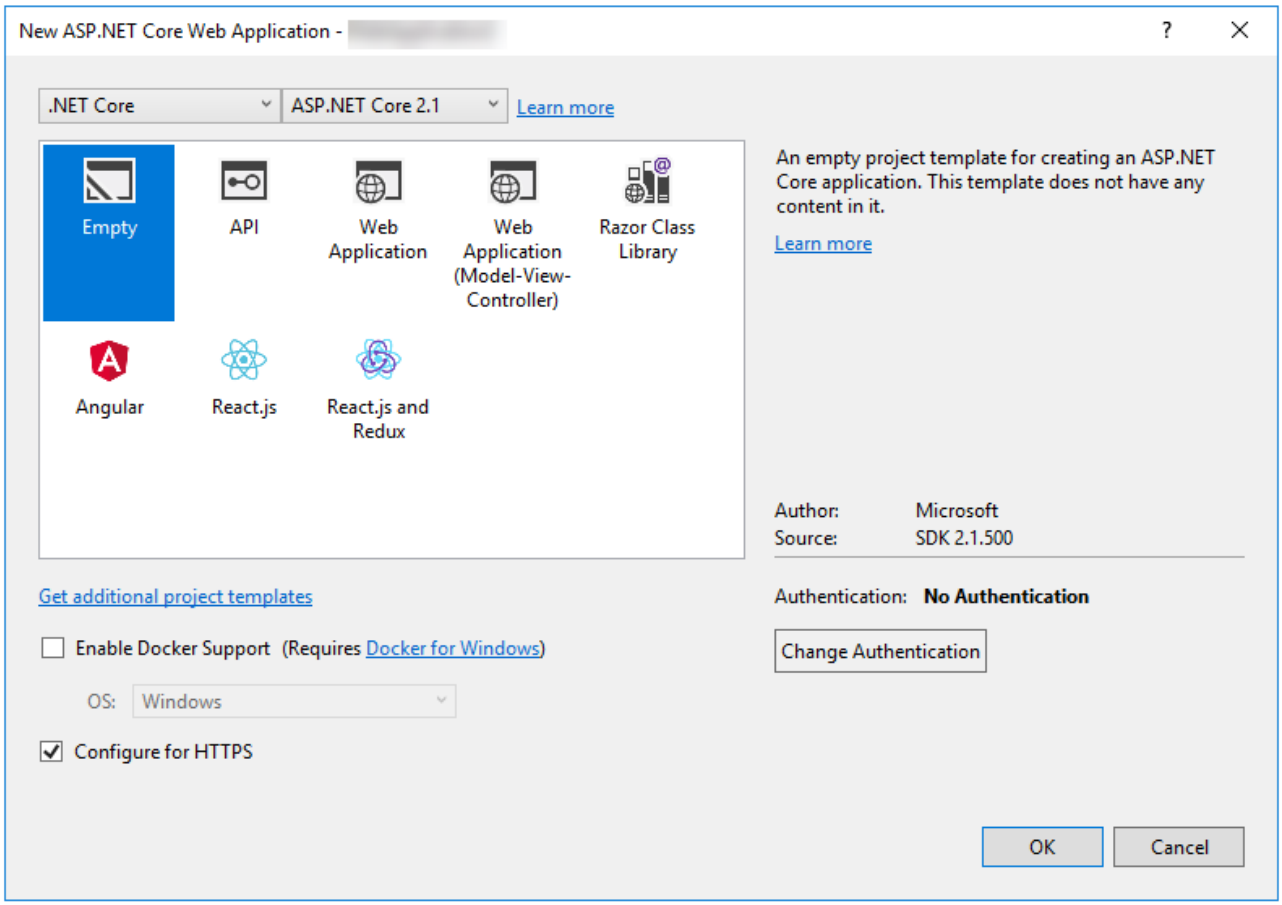
GrapeCity Documents PDF Viewer is a JavaScript based PDF Viewer that can be used in any web application and framework to view PDF documents on Windows, MAC, Linux, iOS and Android devices.

The steps listed below describe how to create an application that uses GcDocs PDF Viewer to view PDF Files.

1. Open Microsoft Visual Studio and create a new ASP .NET Core Web Application.



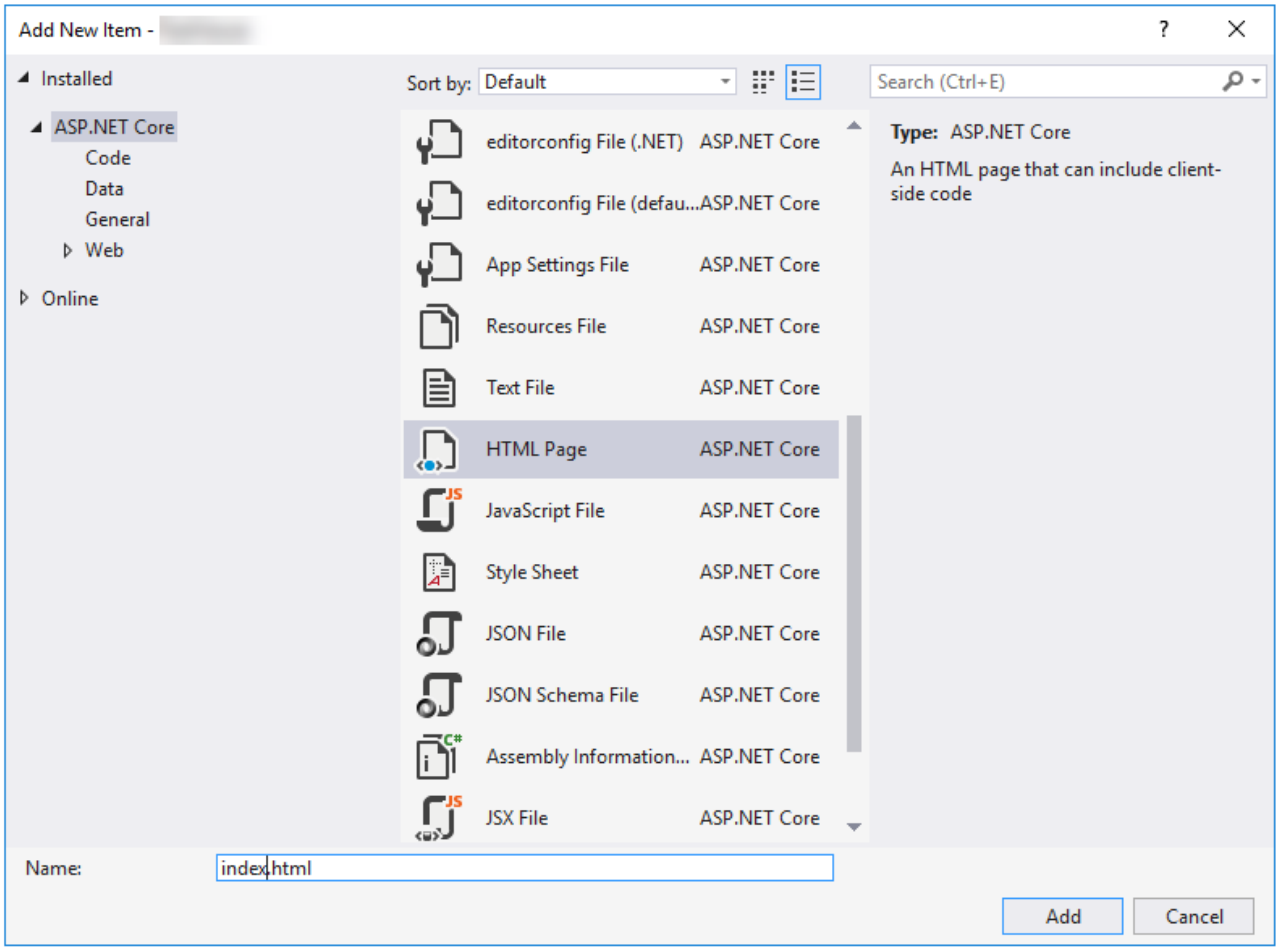
2. Create an empty ASP.NET Core application.



3. Download and unzip the GcDocs PDF Viewer distribution file. Place **gcpdfviewer.js** and **gcpdfviewer.worker.js** files in 'wwwroot' project folder. Alternatively, install the GcDocs PDF Viewer from npm by executing the following

command from the NodeJS command prompt:
`npm install @grapecity/gcpdfviewer.`

4. Add new HTML page to 'wwwroot' folder, name it 'index.html'.



5. Paste the following code in the HTML file.

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, maximum-scale=1, user-scalable=no, shrink-to-fit=no">
    <meta name="theme-color" content="#000000">
    <title>GC Viewer Demo | PDF Plugin</title>
    <link rel="stylesheet" href="https://cdn.materialdesignicons.com/2.8.94/css/materialdesignicons.min.css">
    <script>
        function loadPdfViewer(selector) {
            var viewer = new GcPdfViewer(selector, { /* Specify options here */ });
            viewer.addDefaultPanels();
            viewer.open("Wetlands.pdf");
        }
    </script>
</head>
<body onload="loadPdfViewer('#root')">
    <div id="root"></div>
    <script type="text/javascript" src="gcpdfviewer.js"></script>
```

```
</body>
</html>
```

Note: The above code loads a PDF file in the Viewer by using the Open method. So, please make sure that you have placed the PDF file in the project folder and accordingly altered the file name here.

6. You can even configure various viewer options when initializing the control, as depicted in the code below.

html

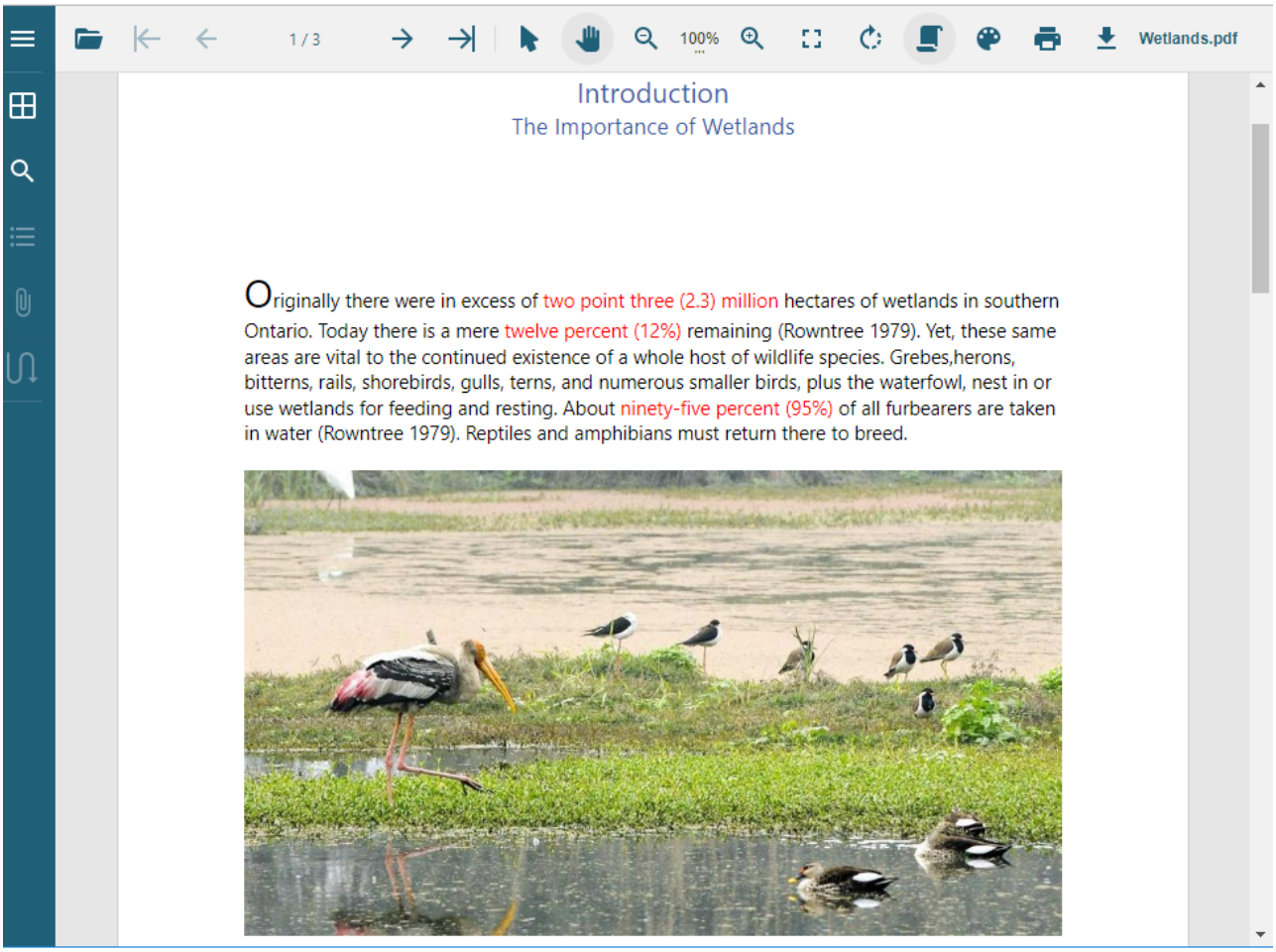
```
<script>
    function loadPdfViewer(selector) {
        var options = {
            file: "Wetlands.pdf",
            friendlyFileName: "my-sample.pdf",
            renderInteractiveForms: true,
            password: "optional-pdf-password",
            externalLinkTarget: "blank",
            theme: "themes/dark-yellow",
            themes: ["themes/viewer", "themes/light-blue", "themes/dark-yellow"]
        };
        var viewer = new GcPdfViewer(selector, options);
        viewer.addDefaultPanels();
    }
</script>
```

7. Modify the Startup.cs file to load static files and default files with the code snippet given below. This will load the index.html page.

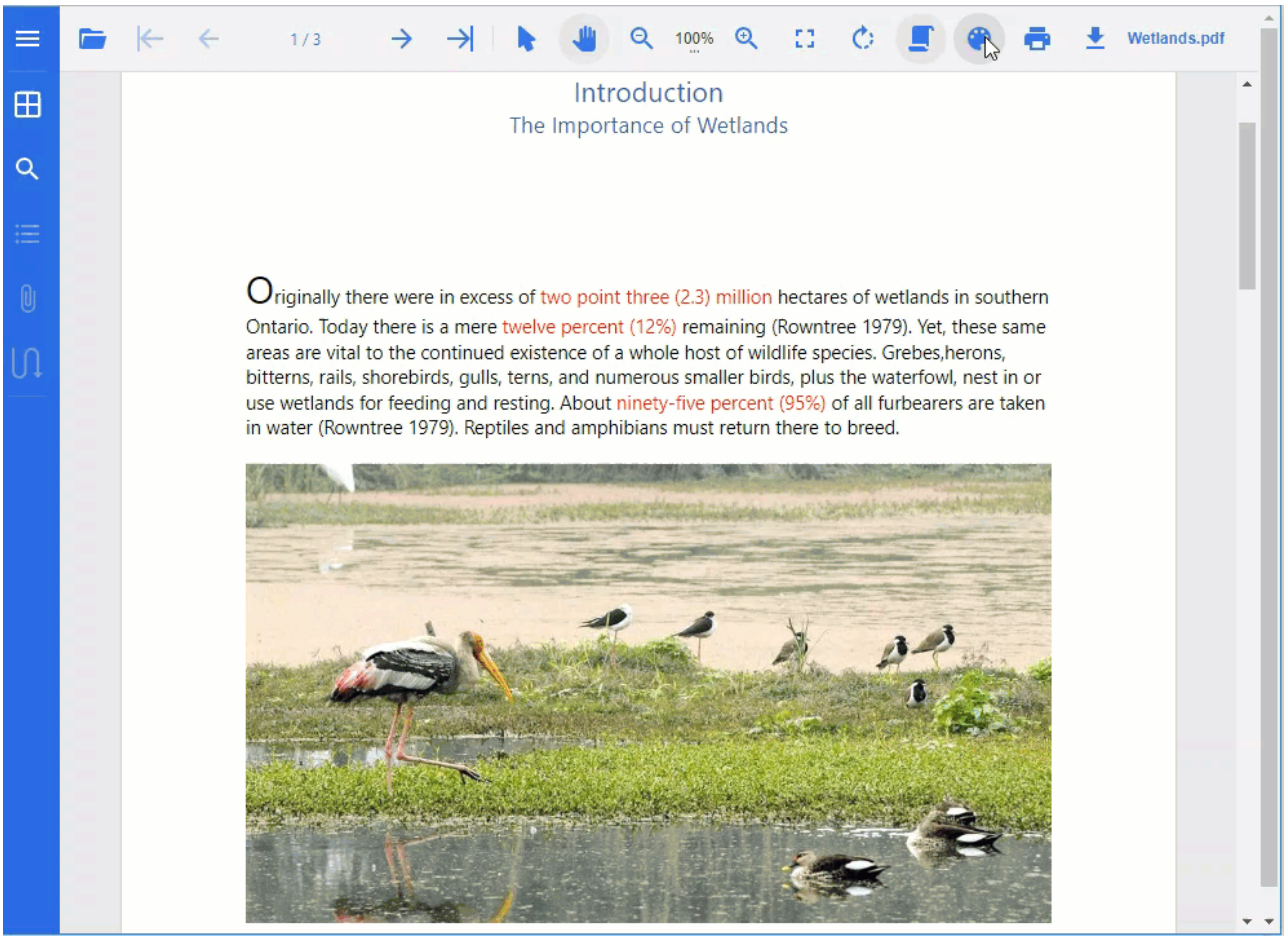
C#

```
public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }
    app.UseDefaultFiles();
    app.UseStaticFiles();
}
```

8. Deploy and run the application, or view the index.html in the browser.



Once you have run the application and viewed the GcDocs PDF Viewer, use the Themes icon in the Toolbar to change the theme.



GrapeCity Documents PDF Viewer API

Options

The following options can be set during initialization or at run time while working with GcDocs PDF Viewer.

Options	Descriptions	Types	Example
baseUrl	Optional. Used for form submission and theme URLs.	string	var viewer = new GcPdfViewer ("#root", {baseUrl: "http://mysite/subdomain", themes: ["mytheme.css"], theme: "mytheme.css" }); viewer.addDefaultPanels();

button	Enables you to specify the visibility for the toolbar buttons in the viewer. Invokes the applyOptions method to propagate these settings to the viewer.
	Available Values: ['open', '$navigation', '$split', 'text-selection', 'pan', '$zoom', '$fullscreen', 'rotate', 'view-mode', 'theme-change', 'print', 'save', 'doc-title'], 'all', 'none'
		object	var viewer =
			new GcPdfViewer(selector);
			viewer.options.buttons = ['print', 'rotate'];
			viewer.applyOptions();

caret	Shows caret for selection. This option is available only when useCanvasForSelection option is set to true.	object	viewer.options.caret = {/** * milliseconds, caret blink period * */ caretBlinkTime: 300, /** * milliseconds, stop caret blink time, 0 - don't stop * */ caretStopBlinkTime: 5000, /** * Caret color * */ color: "#000000", /** * Caret width * */

			width: 1, /** * Allow to move caret using keys even when pan tool activated. * */ allowForPan: false };
disablePageLabels	Allows you to enable or disable page labels in the viewer. Default value is false.	boolean	viewer.options.disablePageLabels=true;
externalLinkTarget	Specifies the target attribute of a link annotation in the PDF file. The option specifies where to open the linked document. Values: 'blank', 'self', 'parent', 'top', 'none', default is 'none'.	string	viewer.options.externalLinkTarget = 'blank';
useCanvasForSelection	If this option is set to false, then static forms are rendered. If the option is set to true, then interactive form elements are rendered.	object	viewer.options.useCanvasForSelection = {selectionColor: "rgba(145, 44, 39, 0.2)", highlightColor: "rgba(83, 145, 39, 1)", inactiveHighlightColor: "rgba(180, 0, 170, 0.35)"};
renderInteractiveForms	To render static forms, set this option to false and to render interactive form elements, set this option to true.	boolean	viewer.options.renderInteractiveForms = true;
keepFileData	Set this option to true if you want to use fileData property even when the document cannot be parsed as a valid PDF document.	boolean	viewer.options.keepFileData = true;
renderer	Specify the renderer type; canvas or svg. Default value is set to canvas.	string	viewer.options.renderer = 'canvas';
friendlyFileName	Sets an alternative file name to be displayed in the title field of the Viewer while saving the PDF document.	string	viewer.options.friendlyFileName = "my-sample.pdf";
file	Specifies the PDF file name, URL, or binary data to be loaded in the Viewer.	string	viewer.options.file = 'GcPdf.pdf';
password	Sets the predefined password for a protected PDF document. This option is useful when you do not wish to enter password in the Viewer.	string	viewer.options.password = 'abc123';
workerSrc	Specifies a custom JavaScript worker script. The worker script works in the background and is used to load PDF files in Viewer.	string	viewer.options.workerSrc = "gcpdflib.worker.js";
themes	Specifies the available themes for Viewer.	string	viewer.options.themes = ["themes/viewer", "themes/light-blue", "themes/dark-yellow"];
zoomByMouseWheel	Sets the options available for zooming the PDF document using mouse wheel.	object	viewer.options.zoomByMouseWheel = { always: false, ctrlKey: false, metaKey: true };

shortcut	Sets keyboard shortcuts.	object	viewer.options.shortcuts =
			{
			107: {// + (numpad)
			ctrl: true,
			tool: "zoomin"
			},
			"S": {
			tool: "selection"
			},

theme	Modifies the default theme for Viewer. List of supported themes: viewer, light-blue and dark-yellow.	string	viewer.options.theme = "dark-yellow.css";

restoreViewStateOnLoad	Tracks view changes and restores the previous state on the next page load.
	Available values: false true, { trackViewMode: boolean, trackMouseMode: boolean, trackScale: boolean, trackPageRotation: boolean, trackFullScreen: boolean, trackTheme: boolean }
		object	// enable all
			viewer.options .restoreViewStateOnLoad = true;
			// disable view changes tracking
			viewer.options .restoreViewStateOnLoad = false
			// enable only scale tracking
			viewer.options.restoreViewStateOnLoad = {trackScale: true}
			var viewer = new GcPdfViewer("#root", { restoreViewStateOnLoad: { trackScale: true } });
			viewer.addDefaultPanels();

documentListUrl	Used by DocumentListPanel for URL to document list service.
	Default value is "/documents". For more information, see addDocumentListPanel method. The service should return json string with available documents array, e.g.:
	["pdf1.pdf", "pdf2.pdf"]
		string	// Using GetDocumentsList service method to get array of the available PDF documents
			var viewer = new GcPdfViewer("#root", { documentListUrl: "/GetDocumentsList" });
			viewer.addDocumentListPanel();
			You can even replace «GetDocumentsList» API by json file
			(see example from addDocumentListPanel method description)

Methods

Method	Description	Parameters	Return Value	Example

addDefaultPanels()	Adds default set of sidebar panels in the following order: 'Thumbnails', 'Search', 'Outline', 'Attachments', 'Articles'	none	void	viewer.addDefaultPanels();
addDocumentListPanel()	Adds document list panel to the Viewer with available document array specified in documentslist.json file (URL specified by documentListUrl option), located in the root directory of your application. Example content for the documentslist.json file: [“file1.pdf”, “file2.pdf”]. Another way is to specify service at the end point for documentListUrl option. The service should return json string with available documents array, e.g.: ["pdf1.pdf", "pdf2.pdf"]	none	void	var viewer = new GcPdfViewer(“#root”, { documentListUrl: "/documentslist.json" }); viewer.addDocumentListPanel();
addThumbnailsPanel()	Adds Thumbnail panel to the Viewer.	none	void	viewer.addThumbnailsPanel();
addOutlinePanel()	Adds Outline panel to the Viewer.	none	void	viewer.addOutlinePanel();
addArticlesPanel()	Adds Articles panel to the Viewer.	none	void	viewer.addArticlesPanel();
addAttachmentsPanel()	Adds Attachment panel to the Viewer.	none	void	viewer.addAttachmentsPanel();

open()	Opens PDF document.	URI to PDF document(string) or binary data(Uint8Array)	file	// open PDF from URI:
				var viewer = new GcPdfViewer("#root");
				viewer.open("RegistrationFormControl.pdf");
				// open binary data:
				var data = new Uint8Array([37, 80, 68, 70, 45, 49, 46, 55, 10, 49, 32, 48, 32, 111, 98, 106, 32, 32, 37, 32, 101, 110, 116, 114, 121, 32, 112, 111, 105, 110, 116, 10, 60, 60, 10, 32, 32, 47, 84, 121, 112, 101, 32, 47, 67, 97, 116, 97, 108, 111, 103, 10, 32, 32, 47, 80, 97, 103, 101, 115, 32, 50, 32, 48, 32, 82, 10, 62, 62, 10, 101, 110, 100, 111, 98, 106, 10, 10, 50, 32, 48, 32, 111, 98, 106, 10, 60, 60, 10, 32, 32, 47, 84, 121, 112, 101, 32, 47, 80, 97, 103, 101, 115, 10, 32, 32, 47, 77, 101, 100, 105, 97, 66, 111, 120, 32, 91, 32, 48, 32, 48, 32, 50, 48, 48, 32, 50, 48, 48, 32, 93, 10, 32, 32, 47, 67, 111, 117, 110, 116, 32, 49, 10, 32, 32, 47, 75, 105, 100, 115, 32, 91, 32, 51, 32, 48, 32, 82, 32, 93, 10, 62, 62, 10, 101, 110, 100, 111, 98, 106, 10, 10, 51, 32, 48, 32, 111, 98, 106, 10, 60, 60, 10, 32, 32, 47, 84, 121, 112, 101, 32, 47, 80, 97, 103, 101, 10, 32, 32, 47, 80, 97, 114, 101, 110, 116, 32, 50, 32, 48, 32, 82, 10, 32, 32, 47, 82, 101, 115, 111, 117, 114, 99, 101, 115, 32, 60, 60, 10, 32, 32, 32, 32, 47, 70, 111, 110, 116, 32, 60, 60, 10, 32, 32, 32, 32, 32, 32, 47, 70, 49, 32, 52, 32, 48, 32, 82, 32, 10, 32, 32, 32, 32, 62, 62, 10, 32, 32, 62, 62, 10, 32, 32, 47, 67, 111, 110, 116, 101, 110, 116, 115, 32, 53, 32, 48, 32, 82, 10, 62, 62, 10, 101, 110, 100, 111, 98, 106, 10, 10, 52, 32, 48, 32, 111, 98, 106, 10, 60, 60, 10, 32, 32, 47, 84, 121, 112, 101, 32, 47, 70, 111, 110, 116, 10, 32, 32, 47, 83, 117, 98, 116, 121, 112, 101, 32, 47, 84, 121, 112, 101, 49, 10, 32, 32, 47, 66, 97, 115, 101, 70, 111, 110, 116, 32, 47, 84, 105, 109, 101, 115, 45, 82, 111, 109, 97, 110, 10, 62, 62, 10, 101, 110, 100, 111, 98, 106, 10, 10, 53, 32, 48, 32, 111, 98, 106, 32, 32, 37, 32, 112, 97, 103, 101, 32, 99, 111, 110, 116, 101, 110, 116, 10, 60, 60, 10, 32, 32, 47, 76, 101, 110, 103, 116, 104, 32, 52, 52, 10, 62, 62, 10, 115, 116, 114, 101, 97, 109, 10, 66, 84, 10, 55, 48, 32, 53, 48, 32, 84, 68, 10, 47, 70, 49, 32, 49, 50, 32, 84, 102, 10, 40, 72, 101, 108, 108, 111, 44, 32, 119, 111, 114, 108, 100, 33, 41, 32, 84, 106, 10, 69, 84, 10, 101, 110, 100, 115, 116, 114, 101, 97, 109, 10, 101, 110, 100, 111, 98, 106, 10, 10, 120, 114, 101, 102, 10, 48, 32, 54, 10, 48, 48, 48, 48, 48, 48, 48, 48, 48, 32, 54, 53, 53, 51, 53, 32, 102, 32, 10, 48, 48, 48, 48, 48, 48, 48, 48, 49, 48, 32, 48, 48, 48, 48, 48, 32, 110, 32, 10, 48, 48, 48, 48, 48, 48, 48, 48, 55, 57, 32, 48, 48, 48, 48, 48, 32, 110, 32, 10, 48, 48, 48, 48, 48, 48, 48, 48, 49, 55, 51, 32, 48, 48, 48, 48, 48, 32, 110, 32, 10, 48, 48, 48, 48, 48, 48, 48, 48, 51, 48, 49, 32, 48, 48, 48, 48, 48, 32,

<table>
<tr><td></td><td></td><td></td><td></td><td>110, 32, 10, 48, 48, 48, 48, 48, 48, 48, 51, 56, 48, 32, 48, 48, 48, 48, 48, 32, 110, 32, 10, 116, 114, 97, 105, 108, 101, 114, 10, 60, 60, 10, 32, 32, 47, 83, 105, 122, 101, 32, 54, 10, 32, 32, 47, 82, 111, 111, 116, 32, 49, 32, 48, 32, 82, 10, 62, 62, 10, 115, 116, 97, 114, 116, 120, 114, 101, 102, 10, 52, 57, 50, 10, 37, 37, 69, 79, 70,]);
viewer.open(data);
// open binary data saved to base64-encoded string:
var viewer = new GcPdfViewer("#root");
viewer.addDefaultPanels();
var pdfBase64Data = "JVBERi0xLjcKCjEgMCBvYmogICUgZW50cnkgcG9pbnQKPDwKICAvVHlwZSAvQ2F0YWxvZwogIC9QYWdlcyAyIDAgUgo+PgplbmRvYmoKCjIgMCBvYmoKPDwKICAvVHlwZSAvUGFnZXMKICAvTWVkaWFCb3ggWyAwIDAgMjAwIDIwMCBdCiAgL0NvdW50IDEKICAvS2lkcyBbIDMgMCBSIF0KPj4KZW5kb2JqCgozIDAgb2JqCjw8CiAgL1R5cGUgL1BhZ2UKICAvUGFyZW50IDIgMCBSCiAgL1Jlc291cmNlcyA8PAogICAgL0ZvbnQgPDwKICAgICAgL0YxIDQgMCBSIAogICAgPj4KICA+PgogIC9Db250ZW50cyA1IDAgUgo+PgplbmRvYmoKCjQgMCBvYmoKPDwKICAvVHlwZSAvRm9udAogIC9TdWJ0eXBlIC9UeXBlMQogIC9CYXNlRm9udCAvVGltZXMtUm9tYW4KPj4KZW5kb2JqCgo1IDAgb2JqICAlIHBhZ2UgY29udGVudAo8PAogIC9MZW5ndGggNDQKPj4Kc3RyZWFtCkJUCjcwIDUwIFRECi9GMSAxMiBUZgooSGVsbG8sIHdvcmxkISkgVGoKRVQKZW5kc3RyZWFtCmVuZG9iagoKeHJlZgowIDYKMDAwMDAwMDAwMCA2NTUzNSBmIAowMDAwMDAwMDEwIDAwMDAwIG4gCjAwMDAwMDAwNzkgMDAwMDAgbiAKMDAwMDAwMDE3MyAwMDAwMCBuIAowMDAwMDAwMzAxIDAwMDAwIG4gCjAwMDAwMDAzODAgMDAwMDAgbiAKdHJhaWxlcgo8PAogIC9TaXplIDYKICAvUm9vdCAxIDAgUgo+PgpzdGFydHhyZWYKNDkyCiUlRU9G";

viewer.open(base64ToUint8Array(pdfBase64Data));
function base64ToUint8Array(base64)
{
var raw = atob(base64);
var uint8Array = new Uint8Array(raw.length);
for (var i = 0; i < raw.length; i++) {
uint8Array[i] = raw.charCodeAt(i);
}
return uint8Array;
}</td></tr>
<tr><td>openLocalFile()</td><td>Opens open local file dialog.</td><td>none</td><td>void</td><td>viewer.openLocalFile();</td></tr>
<tr><td>print()</td><td>Prints the currently displayed PDF document.</td><td>none</td><td>void</td><td>viewer.print();</td></tr>
<tr><td>save()</td><td>Downloads the PDF document loaded in the Viewer to the local disk.</td><td>file name (optional)</td><td>void</td><td>viewer.save('Test.pdf');</td></tr>
<tr><td>goBack()</td><td>Navigate to the previous page in the document.</td><td>none</td><td>void</td><td>viewer.goBack();</td></tr>
<tr><td>goForward()</td><td>Navigate to the next page in the document.</td><td>none</td><td>void</td><td>viewer.goForward();</td></tr>
<tr><td>setTheme()</td><td>Set a theme of your choice.
List of supported themes: viewer, light-blue, dark-yellow.</td><td>theme name – Name of the theme as specified in themes option.</td><td>void</td><td>viewer.setTheme(“themes/dark-yellow”);</td></tr>
<tr><td>enablePdfToolButtons()</td><td>Allows you to enable PDF specific toolbar buttons according to your requirements. Note that you don’t need to call this method if you are using buttons option.</td><td>buttons: string[] | "all" | 'none' = 'all', Available values are: ['open', '$navigation', '$split', 'text-selection', 'pan', '$zoom', '$fullscreen', 'rotate', 'view-mode', 'theme-change', 'print', 'save', 'doc-title'] 'all' 'none'</td><td>void</td><td>viewer.enablePdfToolButtons(['single-page']);</td></tr>
<tr><td>applyOptions()</td><td>Enables you to populate changed options.</td><td>none</td><td>void</td><td>var viewer = new GcPdfViewer(selector);
viewer.options.buttons = ['print', 'rotate'];
viewer.applyOptions();</td></tr>
</table>

Properties

Property	Description	Syntax	Return Type	Example

get fileData()	Gets the file data. This property is available when keepFileData option is set to true.	viewer.fileData;	Uint8Array	var filedata = viewer.fileData;

get options()	Gets the options available for PDF Viewer.	viewer.options	ViewerOptions	viewer.options.renderInteractiveForms = true;
				viewer.applyOptions();
				viewer.open("Test.pdf");

set options()	Sets the options available for PDF Viewer.	viewer.options	void	viewer.options.renderInteractiveForms = true;
				viewer.applyOptions();
				viewer.open("Test.pdf");

get rotation()	Gets the rotation value for PDF document in the Viewer.	viewer.rotation;	number	var rotation = viewer.rotation;
set rotation()	Sets the rotation for PDF document in the Viewer.	viewer.rotation;	void	viewer.rotation = 270;

Events

Event	Description	Example

get onError(): EventFan<ErrorEventArgs>	Indicates errors while loading PDF documents.	function loadPdfViewer()
		{
		var viewer = new GcPdfViewer('#root');
		viewer.addDefaultPanels();
		viewer.onError.register(handleError);
		viewer.open('Test.pdf');
		function handleError(eventArgs)
		{
		var message = eventArgs.message;
		if (message.indexOf("Invalid PDF structure") !== -1)
		
		alert('Unable to load pdf document, pdf document is broken.');
		}
		else {
		alert('Unexpected error: ' + message);
		}
		}

Samples

All the GcPdf samples are available through the online sample browser. You can browse the source code of samples, run them on the server and view the generated PDFs online, or download individual samples to build and run on your own system (Windows, Mac or Linux). For more information, see Quick Start, introductory page for the samples.

If you choose to download the samples, you can run them using following simple steps:

1. Click the **Download** action on the top right of the sample page.
2. Unzip the downloaded .zip file of sample.
3. Run the sample.

List of samples

The complete list of available GcPdf sample projects is mentioned below:

Basic Feature Samples

Features	Sample		Description
	Hello, World!		Demonstrates how to create a simple PDF document with some text.
Document	Document Properties		Demonstrates how to set the document properties such as PDF version, document info and metadata.
	Linearized PDF		Demonstrates how to create a linearized PDF file.
	Page Size/Orientation		Demonstrates how to specify page size and orientation.
	PDF/A		Demonstrates how to create a PDF/A-3u conformant document.
	Large Document	StartDoc/EndDoc	Demonstrates how to use StartDoc/EndDoc to create a large PDF using less memory.
		Large Document	Demonstrates how to create a large document.
		Large TextLayout	Demonstrates how to create a large PDF using a single TextLayout.
Fonts	Font Collection		Demonstrates how to manage and use fonts.
	Font from File		Demonstrates how to load a font from a font file.
	Standard PDF Fonts		Demonstrates how to use the available 14 standard PDF fonts.
	Fallback Fonts		Demonstrates how to use fallback fonts.
	Bold/Italic Emulation		Demonstrates how to control bold and/or italic emulation with normal fonts.
	EUDC Fonts		Demonstrates how to render private use Unicode characters (PUA) with custom EUDC fonts (.tte).
	Font Features		Demonstrates how to use different font features in action.
	Surrogates etc.		Demonstrates how to render surrogate pairs and some other interesting characters.
	Surrogates Portable		Demonstrates how to render surrogate pairs in platform-independent way.
	Unicode Ranges		Demonstrates how to list Unicode ranges available in each system

			font.
Text	Text Rendering		Demonstrates how to implement basics of rendering text.
	Character Formatting		Demonstrates how to implement character formatting.
	Paginated Text		Demonstrates how to render a long text spanning multiple pages in a simple way.
	Text Trimming		Demonstrates how to display ellipsis at end of string that does not fit in the allocated space.
	Multi-formatted Text		Demonstrates how to use different text formats in a single paragraph.
	Word and Char Wrap		Demonstrates how to use word wrapping vs character wrapping when rendering text.
	Sub and Super Script		Demonstrates how to render text as subscript and superscript.
	Multiple Languages		Demonstrates how to render text in multiple languages.
	Paragraph Alignment		Demonstrates how to align paragraphs along the flow direction.
	Paragraph Formatting		Demonstrates how to use the most basic paragraph formatting options, first line indent and line spacing.
	Text Alignment		Demonstrates how to align and justify text in paragraphs.
	Numbered List		Demonstrates how to render a numbered list.
	Nested Lists		Demonstrates how to render various styles of numbered lists.
	Keep With Next		Demonstrates how to prevent page breaks between paragraphs.
	Advanced	Balanced Columns	Demonstrates how to create magazine style, multi-column page layout with balanced columns and document outline.
		Multi-column Text	Demonstrates how to create magazine style, multi-column page layout.
		Tate Chu Yoko	Demonstrates how to render horizontal runs in a vertical text (tate chu yoko).
		Outlined Text	Demonstrates how to draw outlined text, filling text using brushes.
		Ligatures	Demonstrates how to automatically generate typographic ligatures.
		Drop Caps	Demonstrates how create a drop cap.
		Text Around Images	Demonstrates how to flow a large block of text around rectangular areas (Images in this sample).
		Inline Images	Demonstrates how to insert arbitrary objects (Images in this sample) into a block of text.
		Arabic Columns	Demonstrates how to render Arabic text using a columnar layout.
		Japanese Columns	Demonstrates how to draw vertical right-to-left Japanese text using a layout with horizontal columns.
		Rotated Text	Demonstrates how to rotate text by an arbitrary angle.
		Rotated Text (alt)	Demonstrates how to rotate text by an arbitrary angle using matrix multiplication.

		Tabs Alignment	Demonstrates how to render columns of floating point numbers aligned in different ways.
		Vertical Text	Demonstrates how to render vertical text in LTR and RTL modes.
Images	Slide Pages		Demonstrates how to create pages of 'slides' from all images found in a directory.
	Raw Images		Demonstrates how to add JPEG images to documents, using image alignment options.
	Get Images		Demonstrates how to retrieve images from a PDF.
	Image Transparency		Demonstrates how to add render images with a specified transparency.
Graphics	Gradients		Demonstrates how to create gradient fills using LinearGradientBrush and RadialGradientBrush.
	Round Rectangles		Demonstrates how to draw round rectangles.
	Shapes		Demonstrates how various geometric shapes can be drawn and filled.
	Transformations		Demonstrates how to use graphics transformations.
	Soft Mask		Demonstrates how to use soft mask to for clipping and transparency.
Navigation	Link To URL		Demonstrates how to create a link to an external URL and associate it with a text on a page.
	Outlines		Demonstrates how to add outline entries to a document.
	Destinations		Demonstrates how to create destinations and associate them with outline nodes or links in the document body.
	Image Links		Demonstrates how to create a document with thumbnails linked to full-sized images.
	Page Labels		Demonstrates how to add page labeling ranges to a document.
	Image Article Threads		Demonstrates how to create article threads over related images in a PDF.
Annotations	Annotations		Demonstrates how to add different annotations to a PDF document.
	File Attachments		Demonstrates how to attach files to a PDF document.
	Stamp Annotation		Demonstrates how to create a stamp annotation with a custom image.
	Sound Annotations		Demonstrates how to add sound annotations.
	Ink Annotations		Demonstrates how to render an ink annotation.
	Find and Redact		Demonstrates how to find and redact texts in a PDF.
Forms	AcroForm Fields		Demonstrates how to create the various AcroForm fields such as textbox, checkbox, push buttons and so on.
	Submit Form		Demonstrates how to submit form data to a server.
	Import XML		Demonstrates how to import submitted form data from XML.
Tags	Tagged Paragraphs		Demonstrates how to create tagged (structured) PDF.

	Tagged TextLayout	Demonstrates how to render a long text spanning multiple pages.
Merge PDFs	Merge PDFs	Demonstrates how to merge two PDFs into a single document.
	Page FormXObject	Demonstrates how to create a FormXObject from a page and draw it in another document.
Editing	Load PDF	Demonstrates how to load and modify an existing PDF file.
	Fill PDF Form	Demonstrates how to load a PDF form, read and update its fields.
	List Fonts	Demonstrates how to lists fonts found in a loaded PDF.
	Extract Text	Demonstrates how to extract text from a PDF file.
	Find Text	Demonstrates how to find and highlight all occurrences of a string.
	Find Transformed	Demonstrates how to find and highlight all occurrences of a string in a graphically transformed text.
Miscellaneous	Signing PDF	Demonstrates how to sign a created PDF with a .pfx file.
	Incremental Update	Demonstrates how to sign a signed PDF preserving the validity of the original signature.
	Visual Signature	Demonstrates how to sign a PDF using a visual signature with a .pfx file.
	Document Attachments	Demonstrates how to attach files to a PDF document at document level.
	Easy Page Headers	Demonstrates how to generate left/centered/right aligned page headers and footers.
	JavaScript Action	Demonstrates how to associate a link with a JavaScript action.
	Security Handlers	Demonstrates how to use security encrypt/decrypt handlers.
	Remove Signature Fields	Demonstrates how to find and remove existing signature fields.
	Remove Signatures	Demonstrates how to find and remove existing signatures.
	Save as Image	Demonstrates how to save PDF pages as images.

Barcode Samples

Supported Barcodes	Demonstrates how to render different barcode symbologies supported by the GcBarcode library.
Shipping Labels	Demonstrates how to generate shipping labels using barcodes.

Fancy Doc Samples

Sample	Description
Goods Return Form	Demonstrates how to create an AcroForm with multiple input fields and a complex layout.
Time Sheet Form	Demonstrates how to create a time sheet form PDF.
Data Sheet	Demonstrates how to create a two-page data sheet with a complex layout.
Wetlands	Demonstrates how to create a text document with some images and text highlights.

Use Case Samples

Sample	Description
Time Sheet	Implements a scenario involving generating, filling and signing a time sheet.
Time Sheet Incremental	Implements a scenario involving generating, filling and signing a time sheet using two signatures.
Word Index	Implements a scenario where alphabetical index is generated and appended to a PDF document.